TWO-STREAM FUSION EDGE DETECTION NETWORK

by

Hasan W. Almawi
Honours BSc, Western University, Canada, 2015

A thesis
presented to Ryerson Univesity
in partial fulfillment of the
requirements for the degree of
Master of Science
in the Program of
Computer Science

Toronto, Ontario, Canada, 2018

AUTHOR'S DECLARATION FOR
ELECTRONIC SUBMISSION OF A THESIS

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public.

TWO-STREAM FUSION EDGE DETECTION NETWORK

Master of Science in Computer Science, 2018

Hasan W. Almawi

Ryerson University

Abstract

This thesis introduces a method to combine static and dynamic features in a convolutional neural network (CNN) to produce a motion and object boundary prediction map. This approach provides the CNN with dynamic and static cues and information, thus improving its predictions. The spatial stream of the CNN learns to compute an object boundary prediction map from a single RGB frame, while the temporal stream learns to compute a motion boundary prediction map from the corresponding optical flow map. The streams are then combined through an encoder-decoder architecture, where the decoder learns to fuse the features from both streams to obtain a task specific output. The proposed method yields state-of-the-art results on a motion boundaries benchmark, and systematic improvements in object boundaries benchmarks over methods that solely rely on static features extracted from a single RGB frame.

Acknowledgements

Firstly, I would like to sincerely thank my advisor, Kosta Derpanis, for the opportunities and direction throughout my graduate studies. Secondly, I thank my family for their endless support and enabling me to pursue my goals. Lastly, I thank Salmiyeh Khan for the motivation and encouragement through and through.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Historically, the edge detection problem has been at the center of computer vision with multiple approaches, from basic operators [8] to complex machine learning models [52, 28, 34], attempting to find a solution to this low-level task.

Edge detection is a classic computer vision problem that aims to label the discontinuities between objects in the scene as boundary pixels. The significance of this problem is that edges are important for feature detection in image processing for tasks such as segmentation [16, 49], and optical flow estimation [40, 31]. There are several types of edges that researchers have attempted to extract from various scenes, namely: object boundaries, motion boundaries, and material boundaries. This thesis focuses on the tasks of detecting object and motion boundaries in consecutive images.

The object boundaries detection task garners the most notable attention out of all edge detection tasks due to the significance of edge-like features with tasks targeting objects in a scene. An object boundary is defined as the subset of contours affiliated with objects in the scene. The object boundaries

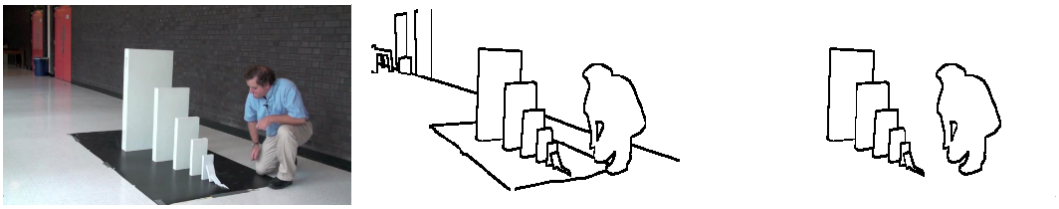

Figure 1.1: An example of object and motion boundary in a scene. The first image (left) is the raw input frame. The second image (center) includes the object boundaries in the scene, where the edges include the static objects in the scene such as the door and table in the upper left corner of the image. The final image (right) depicts the motion boundaries in the scene, where the boundaries are a subset of the object boundaries image (center) highlighting the dynamic objects in the scene. Source: VSB100 Dataset

in the scene are those of the subjects in the scene absent of any boundaries pertaining to materials such as clothing and other textures. This can be seen in Figure 1.1, where the groundtruth object boundaries are drawn around all the objects in the scene, static or otherwise.

On the other hand, motion boundaries are a subset of object boundaries specific to dynamic or moving entities in a scene. While object boundaries provide important cues for static tasks, motion boundaries provide cues around boundaries for tasks such as motion segmentation and optical flow detection. In an example image, the motion boundaries in the scene are those of the moving subjects in the scene, absent of any boundaries pertaining to static objects. This can be seen in Figure 1.1, where the groundtruth motion boundaries are a subset of the object boundaries and are specific to the moving objects in the scene, specifically the dominoes and person.

With the introduction of convolutional neural networks, current object boundary detection methods [28, 34] have achieved and surpassed human per-

formance on certain real world datasets such as the BSDS500 Dataset[1], being able to produce an accurate object boundary map in milliseconds. These convolutional neural networks, or ConvNets, learn to extract edges at the earlier levels and abstractions at higher levels, mimicking human perception. This is shown in the VGG-16 network trained on the ImageNet dataset [11], and other convolutional neural networks trained on real image datasets. Illustrated in Figure 1.2, the convolutional layer side outputs throughout the network show edge maps that progressively become coarser and broader as the network grows deeper.

While these approaches produce impressive results on standard benchmarks, these models are limited to analyzing static imagery and forgo the additional rich information in temporal imagery. This thesis introduces a dynamic ConvNet stream in conjunction with the static ConvNet stream to fuse learned features from both streams to produce more accurate object and motion boundary prediction maps.

1.1 Motivation

While the state-of-the-art boundary detection approaches have been quite impressive [28, 34], the approaches proposed still suffer from issues arising from the absence of integrating dynamic cues. Presently, learning static cues relies on the assumption that world is static and the objects in the scene are not moving. These assumptions are problematic when it comes to issues such as occlusion, where the object may be present in a frame and then occluded in the next. This may cause the network to assume that the non-occluded parts of the object are part of the background or part of a separate entity. Another problem is the scenario where the object textures blend in with the

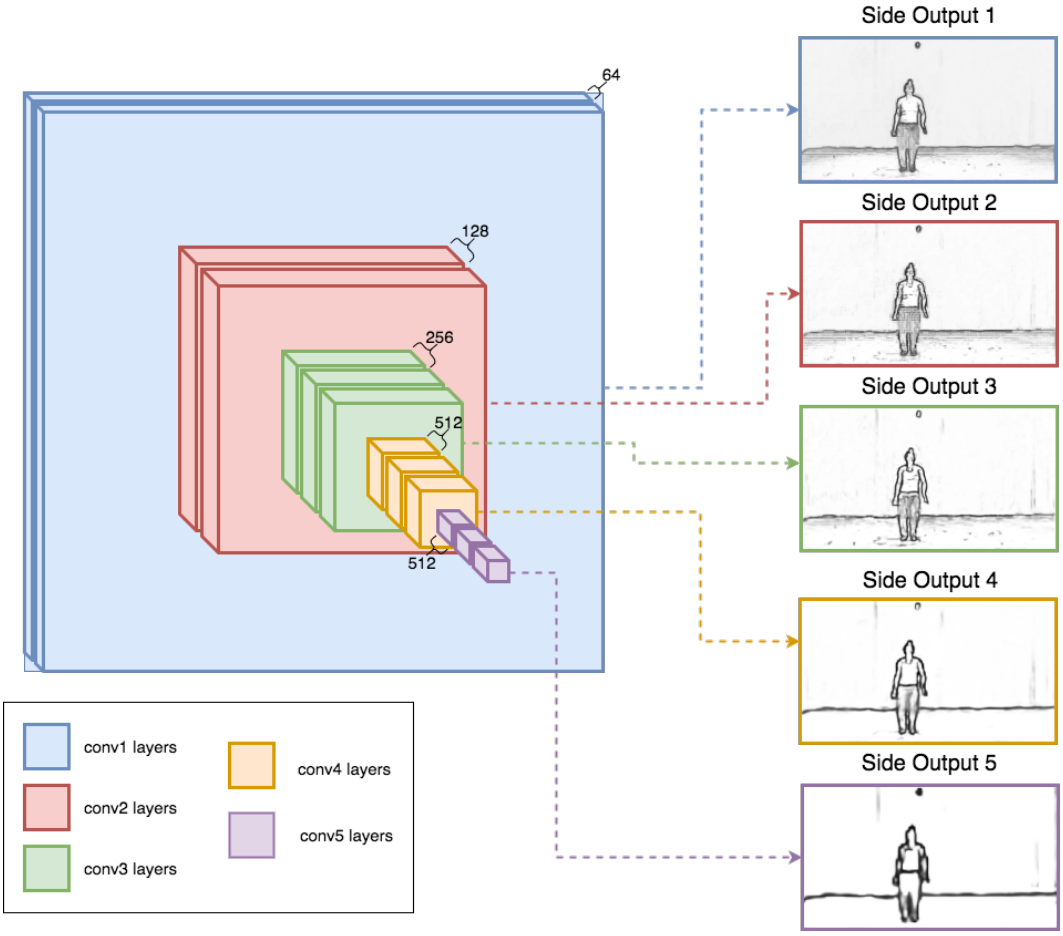

Figure 1.2: An illustration of the VGG16 network. The different colours in the image correspond to the different convolution layers in the VGG16 network. The side outputs are produced by convolving the features of the final convolution operation in each convolution layer. As the number of layers in the network grows, the representations grow more abstract.

background, where the lack of dynamic cues will likely fool the network into believing that the object is part of the background. With this in mind, this thesis proposes a method to alleviate the performance issues that arise from relying solely on static, single images to extract boundaries.

This thesis proposes a method to integrate dynamic cues along with static cues to produce a more refined boundary detection output. The method combines the learned static features from the raw input frame and dynamic features from the raw frame's corresponding optical flow field prediction. This is achieved through a fusion mechanism that refines the boundary detection output. Specifically, the method is comprised of a two-stream, encoder-decoder edge detection network that learns static and dynamic features to detect task-dependent boundaries, and fuses the features to produce a more refined, task-dependent boundary output. The encoder network is comprised of a static and dynamic stream, where the network learns to produce boundaries around all objects (object boundaries) from raw input frame, and boundaries around moving objects (motion boundaries) from optical flow respectively. In this case, the proposed method is learning to fuse static and dynamic cues to improve the results of a boundary detection network that would have access to an exclusive set of cues only.

1.2 Contributions

This thesis makes the following three contributions:

1. **Optical flow to motion boundaries.** An end-to-end trainable convolutional network that learns to produce motion boundaries from optical flow. This is enabled by a "curriculum learning" [4, 23] training scheme

designed to gradually train the network on increasingly difficult examples.

2. **Feature fusion methodology.** An ablative study on three fusion methods that combine spatial and dynamic features in latent space and image space to produce a more refined boundary detection output.

3. **Static and dynamic encoder-decoder architecture.** The state-of-the-art, end-to-end motion boundary detection convolutional network architecture, and a convolutional neural network architecture that produces systematic improvements over object boundary detection methods.

While the evaluations presented in this thesis are exclusively boundary detection related, the method and architecture proposed may provide improvements on tasks that rely on static and dynamic cues, such as motion and object segmentation.

1.3 Outline of thesis

The thesis is organized as follows:

- Chapter 2 provides the background knowledge on convolutional networks and their components.

- Chapter 3 summarizes different approaches closely related to this thesis: object boundaries, motion boundaries, fusion methodologies, and curriculum learning.

- Chapter 4 introduces the components of the approach to produce static and dynamic cues from the input. The chapter then presents the fusion

method to combine the cues produced from both streams. The chapter concludes with the loss function and implementation details that enable the network to produce the edge detection maps.

- Chapter 5 presents empirical evaluations. Quantitative and qualitative results for both the motion boundary and object boundary detection tasks are presented, including evaluation of the different dynamic stream inputs, fusion methodologies, and training schemes.

- Chapter 6 provides a summary of the contributions and results, and explores different possibilities for future work.

Chapter 2

Technical Background

This chapter is made up of three sections: convolution operator, architectures, and learning. While the depth of knowledge entailed in convolutional network theory cannot be conveyed in this thesis alone, the following sections provide a brief overview of the core components of convolutional networks that are most significant to this body of work. The first section defines the "convolution operator" in ConvNets (Sec. 2.1). The next section (Sec. 2.2), "architectures", provides an overview of how layers interact with one another to produce a desired output from the input. The chapter concludes with the "learning" subsection (Sec. 2.3) that clarifies how learning occurs in a convolutional network.

2.1 Convolution operator

A convolutional network is comprised of several convolution operations that process patches of the input [29]. Mathematically, a convolution is an operation on two functions to provide a third function, in which the operation

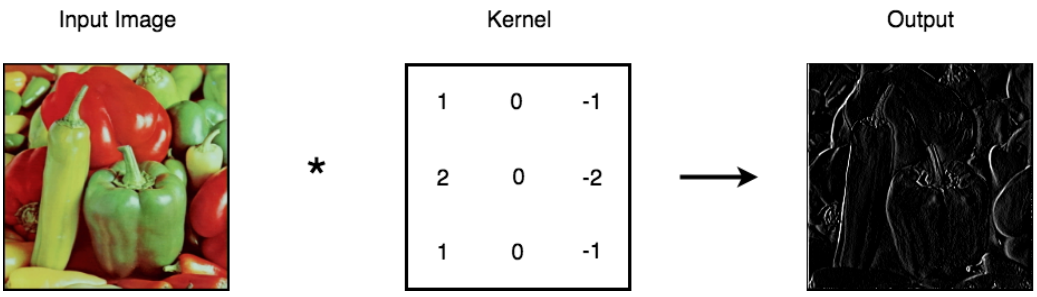

Figure 2.1: An example of a convolution operation where the input image is convolved (*) with a filter in order to produce an output.

amplifies certain parts of the function and suppresses other parts of the function. This is demonstrated in Figure 2.1, where a handcrafted Sobel filter is applied to an input image.

In the context of computer vision, a convolution is a correlation operation between features and input patch. The convolution operation is made up of two tunable parameters: the filter (also known as the features or weights) and bias. The filter is small spatially but deep in volume. For example, if the input image was an RGB image, the depth of the filter to transform the input to an output volume would be $H \times W \times 3$ for the three color channels, and the output volume depth channel could be $H \times W \times D$ of any arbitrary size D, depending on the desired size of the output volume.

To perform the convolution operation, each of the filters in the convolution operation slide across the input map and produces a two-dimensional activation map with responses at each spatial location:

$$F * G = \sum_{u=-K}^{K} \sum_{v=-K}^{K} F[x+u, y+v]G[u, v] \tag{2.1}$$

Where the window size is $(2K+1) \times (2K+1)$, the spatial location is (x, y), F is the input, and G is the filter.

2.2 Architectures

Convolutional networks are comprised of convolution, sub-sampling, and non-linearity layers. The purpose of the convolution layer is to apply convolution filters to inputs and produce activation maps. The sub-sampling layer reduces the input size to increase the receptive field of the convolution window. Finally, the non-linearity layer introduces non-linearity to the network solution since the solution to a complex function such as learning to detect edges in an image is likely not a linear combination. Different architectures employ different combinations of the layers mentioned above depending on the task at hand.

Convolutional network architectures are designed depending on the task they are attempting to address. Their depth is tailored depending on the task difficulty and the input size. For example, a simpler task, such as digit classification, may require a shallow ConvNet architecture, while an object classification network may require a much deeper architecture. Depending on the network output, the architecture may incorporate a fully connected layer to produce a classification score output, or a fully convolutional output to produce a prediction map.

Several architectures in the field of machine learning and computer vision are modifications of other notable architectures, such as VGG16 [43] or AlexNet [29]. These architectures are often adopted due to the availability of the weights associated with the network that has been trained on different datasets, such as ImageNet. Training these networks anew would be notoriously difficulty as they are very sensitive to the weight initialization as well as the training methodology to achieve the results on datasets, such as ImageNet. Adopting available weights and architectures also reduces the required size of

the training set on the target task.

2.3 Learning

For a network to "learn" to transform a given input to a desired output, it must be provided with a signal to identify how precise the network output is. In supervised training, the network is provided with numerous examples that include the input and groundtruth to compare the network output with the intended output.

To measure the difference between the groundtruth and the output of the network, the network employs a *loss or energy function*. Depending on the loss function output, the network will tune the weights of the ConvNet significantly or minimally to improve the accuracy of the network. For example, a network may utilize an L2 or square difference loss function:

$$L(x, y) = \sum_{i=0}^{N} (y_i - x_i)^2, \tag{2.2}$$

where L is the loss associated with the output of the network x and the groundtruth y.

Depending on the gradient of the loss, the network will then update the weights either drastically if the loss is large or minimally if loss is small. This weight update function is done through the process of *backpropagation*. Backpropagation [41] applies the chain rule and computes the partial derivative with respect to all the parameters in the network. This allows the network to shift the parameters of the network towards the negative gradient and thus minimize the loss. It is important to note that the loss will likely not converge to the global minimum but rather a local minimum, and this is due to the greedy nature of the gradient descent algorithm and the non-convex nature of

the error surface.

While the network may not converge to the global minimum, there are methods to ensure that the network is not gridlocked in a local minimum. Namely, manually adjusting hyperparameters that are set when the network is training, specifically the learning rate and momentum.

The learning rate and momentum of a neural network dictate the speed at which the network "learns". The *learning rate* is a scalar value that is multiplied with the gradient computed from the backpropagation step. A high learning rate value will not find an acceptable solution, while a low learning rate value will prolong the learning process of the neural network. *Momentum* dictates the velocity of the "learning" process, meaning that if the gradients computed in consecutive iterations are in the same direction, the network weights will shift more favourably in that direction, thus allowing the network to reach a local minimum faster.

Chapter 3

Related Work

This chapter provides an overview of the closely related topics to this thesis: Motion boundaries (Sec. 3.1), object boundaries (Sec. 3.2), network fusion (Sec. 3.3), and curriculum learning (Sec. 3.4).

3.1 Motion boundaries

Motion boundaries have played a vital role in several computer vision applications varying from optical flow estimation [40] to improving motion segmentation detection prediction maps [24]. Sundberg et al. [47] improved upon their previously introduced static edge detector [35] through the incorporation of motion cues. Their approach computes motion boundaries from optical flow patches, using the norm of the optical flow gradient, in the vicinity of the edge detection output of a static image, thus removing edges related to background clutter in the scene.

The task of predicting motion boundaries has relied on hand-crafted features since its inception. The work of Weinzaepfel et al. [51] relies on producing

a feature vector for each patch, consisting of the color and color gradient, optical flow and flow gradients, image warping errors, and the backwards flow along with its gradients. The structured random forest then predicts whether or not each patch matches a boundary template from the set of hand crafted boundary templates. While this approach has yielded the state-of-the-art motion boundary estimation method, it relies on too many inputs that are costly to compute as well as hand-crafted features to produce an output.

An analogous task to motion boundary estimation is the task of motion segmentation in videos where a subset of objects, particularly those in motion, are segmented. Fragkiadaki et al. [16] utilized optical flow and static frames in a video sequence to propose a set of regions in the video to produce accurate video segmentations. This set of regions acted as a filter to the static background that is not found in the set of regions in the motion stream. Tsai et al. [49] set out to improve the boundaries around optical flow estimations through a method that jointly optimized both optical flow estimation and video segmentation. This was achieved through employing the optical flow prediction output of the network as an input cue for the video segmentation task, and vice-versa employing the video segmentation output to compute the optical flow independently in the segmented regions, thus improving results for both tasks.

Most closely related is the work of Jain et al. [24] which uses a fusion network that learns to fuse motion segmentation prediction maps from the motion and static streams to produce a final prediction map for a given frame. While this approach is similar to ours, both streams of their network have identical tasks in predicting motion segmentation, while each stream in our network has a task corresponding to its input. Another difference is the fusion methodology, where their late fusion method fuses prediction maps in image

space, while our approach learns to fuse features from both streams in latent space and produce a final, task-dependent output.

3.2 Object boundaries

One of the first tasks that the computer vision community set out to address since its inception was edge detection. Earlier approaches utilized color intensity and other hand-crafted features such as thresholding the gradient map of an image in the case of the Sobel operator [44], and later on applying Gaussian smoothing and bi-thresholding as an extension in the Canny approach [8]. While these approaches yielded some promising results, their performance is poor compared to modern approaches that utilize data and deep learning.

Later works utilized machine learning along with hand-crafted features to produce object boundary prediction maps. Their limitations are due to their hand-crafted features that may not be optimal for the object boundary detection task. Martin et al. [36] used brightness, color and texture to produce a probability map of boundaries and their orientations. Lim et al. [32] incorporated local learned features of hand drawn edges to use as patch-based templates to detect local edges in images. Dollar et al. [12] utilized random decision forests to predict local edge maps in images.

Bertasius et al. [5] inverted the edge detection pipeline by utilizing the output from the Canny edge detector [8] as input to the network. The network takes as input the Canny [8] edge map output and extracts patches, at four different scales, around each candidate point as input to the network to output a pixel-wise classification of edge/non-edge, as well as the confidence in the edge prediction. Hwang et al. [22] produced a simpler approach to the edge detection problem by simply extracting features from [21] as input to

a binary-SVM to produce a per-pixel edge detection output. This approach inspired Xie et al. Holistically-Nested Edge Detection (HED) approach, [52] consisting of a deeply supervised convolutional network that produces an object boundary map from an input image. The HED approach modifies the VGG16 architecture [43] to produce side outputs at each convolution layer which are later on fused to produce a final object boundary prediction output.

Several of the recent deep learning edge detection approaches are variants of the HED network, for example Kokkinos [28] uses a variant of the network that incorporates a multi-resolution architecture inspired by [5]. Kokkinos [28] also introduces Multiple Instance Learning (MIL) to improve upon annotation inconsistencies among the groundtruth training set examples by labeling pixels in a local vicinity around the edge pixel as edges, and the Graduated Deep Supervision Network (G-DSN) training approach which temporally decreases the weight of the intermediate loss layers. Liu et al. [34] utilized the representations from each convolution output in each layer to produce each side output, improving results with a simple modification to the HED network architecture.

Other approaches to the edge detection problem include an unsupervised approach to edge detection from Li et al. [31]. Accurately labeled edge data is very expensive to generate on a large scale, with this mind Li et al. [31] produced an unsupervised pipeline to extract edges from consecutive frame dense matches [50]. The network learns to compute motion edges from optical flow as a form of supervision, and in turn improves the optical flow estimation and edge map output. The work Liu et al. [33] approaches the edge detection problem from a different angle: the input data. Liu et al. [33] combines edge maps predicted from the Canny edge detector output [8] to the labeled edge groundtruth data as a form of relaxed supervision. As training progresses, the network gradually increases the loss associated with labeling the edges from

[8] as groundtruth edges.

This static stream of this thesis is based on the HED network architecture and incorporates the G-DSN training scheme to produce object boundary maps. However, the approaches in relation to this thesis do not utilize dynamic input in producing an object boundary but rather solely rely on static cues to produce edge maps. This in turn can be seen in the results of the approaches above, where edges relating to textures and materials are found in the output.

3.3 Network fusion

There are various representations of raw imagery that highlight different information to a ConvNet. Some of these examples include optical flow for motion between two frames in a video sequence, or a RGB image for spatial information about a scene. There have been several notable works on the fusion of these different sensory data types to produce accurate task-dependent outputs. These works have looked into the type of data to combine, how to combine the information in latent space or otherwise, and when to combine the features learned in the network.

Ji et al. [25] introduced the 3D convolution operator in their work, enabling a ConvNet to learn features spatially and temporally. This operation allowed the network to fuse information, both static and dynamic cues in a volumetric input of a video sequence. Karpathy et al. [27] built upon the work from [25] by applying 3D convolutions to the action recognition problem. The work from [27] produced a study on the multiple approaches for spatiotemporal fusion techniques, including early fusion, and late fusion. Sun et al. [46] separated the 3D convolution operator into a 2D spatial kernel and a 1D temporal kernel in order to alleviate the need of large quantities of training data.

Concatenating input frames and applying a 3D convolution is not the only method of fusion. Simonyan et al. [42] considered a two-stream convolutional network for action recognition in videos, in which they trained a two-stream network for action recognition. Their network included a spatial stream and a temporal stream which fused features in the deeper layers of the network architecture. Similarily, Cheron et al. [10] fused human pose information along with spatial information to produce more accurate video action recognition results. Cheron et al. [10] included a spatial stream that learned appearance information, and a dynamic stream that learned motion information from flow. The features produced by both streams are then aggregated and fully connected layer predicted action. In contrast to Cheron etl a. [10], the work from Gkioxari et al. [18] included a dynamic stream that learned motion features from kinematic cues extracted from optical flow.

A follow up from Simonyan et al. [42] was later published by Feichtenhofer et al. [15] which presented an ablative study on fusion methods and trade-offs in layer selection to fuse the temporal and spatial streams. Results from Feichtenhofer et al. [15] found that fusing features from both streams at a later convolution layer allows for a reduction in parameter size without sacrificing network accuracy performance. Jain et al. [24] applied the fusion methodology from [15] on the semantic segmentation problem. The method from [24] learns to combine spatial and dynamic information to produce pixel level segmentation masks for objects through a series of atrous convolution and pooling layers.

This thesis draws inspiration from the works above in the fusion of both static and dynamic streams to produce a task-dependent boundary map. Specifically, learning motion cues from optical flow similar to the approach from Cheron et al. [10], and incorporating a two-stream convolutional network ar-

chitecture to fuse both static and dynamic streams in latent space from Simonyan et al. [42]. This work however differs from Jain et al. [24] in which each stream has a different task, and the fusion methodology fuses the streams in both latent space and image space, as opposed to image space only. This thesis differs from the work from Feichtenhofer et al. [15] in terms of tasks and the multiple fusion points as opposed to a single point of fusion.

3.4 Curriculum learning

Various computer vision and machine learning approaches and ideas are inspired by biology and psychology, e.g., curriculum learning. The idea of curriculum learning originates from cognitive development psychology and was first introduced to machine learning by Elman et al. [14]. Elman et al. studied the effects of complexity of training data on training an artificial intelligence model by attempting to train a recurrent neural network (RNN) on complex sentences and on examples gradually increasing in difficulty. The results of the experiments suggested that gradually increasing complexity of training examples throughout the training phase preconditions the model to be able to solve more complex problems.

Bengio et al. [4] presented an ablative study on the effects of gradually increasing the amount of concepts as well as the complexity of concepts in the context of machine learning. The experimental results showed that implementing curriculum learning in the training phase significantly improved the speed of convergence as well as the accuracy of the machine learning model. Krueger et al. [30] examined the role of decomposing tasks into various subtasks to ease the complexity of learning the task in the context of training neural networks. Their findings showed that further decomposing tasks into

subtasks improved the performance of the model and decreased the training time.

While the focus of curriculum learning has been on the ordering of training data, the hyperparameter selections in training the network also play a vital role in its success. Bengio [3] considered different hyperparameters involved in training a neural network and the best practices when fine-tuning on different input data. Some of these best practices include adjusting the learning rate when fine-tuning a network on different data, selecting the optimal batch size input, and the number of epochs to train the network on each dataset. These practices were applied by Simonyan et al. [43] to train the VGG16 and VGG19 networks, where shallow networks were used to initialize deeper networks thus allowing the deeper layers to learn higher level features while fixing the lower level features.

Recently, Gülçhere et al. [19] built on the work from [4] and posed the question of introducing prior information to aid the network in the training phase. In [19], a two-tiered multilayer perceptron (MLP) is trained on a dataset consisting of images of a shape from three distinct viewpoints and the network must output whether or not the images are of the same shape. Standard neural networks training techniques have failed at the task, however introducing an intermediate supervisory signal allowed the MLP to produce accurate results.

This thesis incorporates curriculum learning and considers the various practices presented by Bengio [3] when training the dynamic stream network on optical flow input. In contrast to the works listed above, this thesis also provides an ablative study on the effects of different data input types, such as synthetic and realistic data, to improve upon the accuracy of the networks.

Chapter 4

Technical Approach

This chapter presents the technical components of the thesis. The first section (Sec. 4.1) focuses on the encoder networks and their implementation details, while the second section (Sec. 4.2) focuses on the decoder network, along with the different fusion implementations, as well as the implementation details. The final section (Sec. 4.3) includes a short summary of the technical approach of this thesis.

4.1 Encoder network

This section introduces the encoder streams: the spatial encoder network (Sec. 4.1.1), and the dynamic encoder network (Sec. 4.1.2). Each subsection will introduce the general architecture as well as the implementation details of the encoder network.

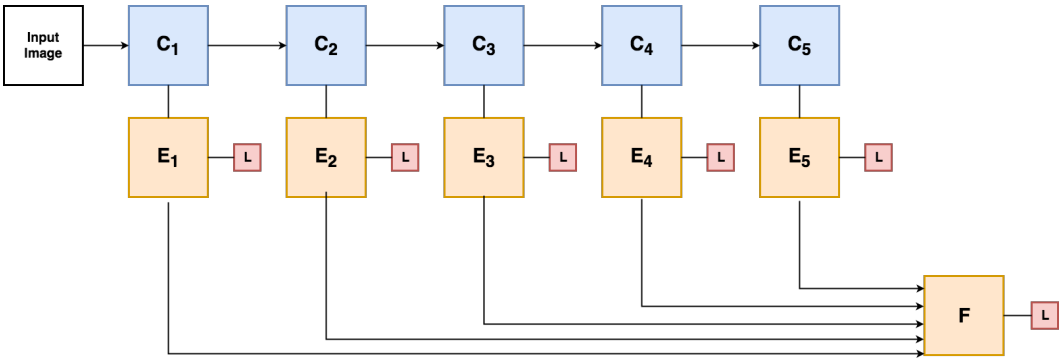

Figure 4.1: The Holistically-Nested Edge Detection network. The network is a modification of the VGG16 architecture, where each convolution layer (C) corresponds to the VGG16 convolution layers. The side outputs (E) take in as input the features volume produced from the final convolution operation in each layer, and transforms the feature volume into a prediction map through a bilinear interpolation and convolution step to calculate the loss (L). Each of the side outputs of the five layers are then concatenated and fused to produce a final prediction output (F).

4.1.1 Spatial stream

Given an input RGB colour channel image, the goal of the spatial stream network is to produce an object boundary edge map as output. The spatial stream convolutional network architecture was chosen to be the HED [52] network architecture, since it is the underlying architecture of state-of-the-art approaches in the object boundary detection task.

The HED network is comprised of the VGG16 [43] architecture with the exception of the fully connected layers and final pooling layer, as seen in Figure 4.1. The reason behind trimming the network is twofold: A stride of size 32 is too small to produce a meaningful prediction map after interpolation, and the network architecture is fully convolutional and thus does not require a fully connected layer.

The network is modified with the addition of side output layers at the last convolutional layer in each stage; specifically, conv1_2, conv2_2, con3_3, conv4_3, and conv5_3. These side output layers are comprised of a 1×1 convolution, followed by a bilinear interpolation, and a sigmoid cross entropy loss function, and produce prediction maps at each receptive field level. The final fusion stage of the network architecture consolidates the prediction maps produced from the side output layers into a final prediction through a concatenation operation and a 1×1 convolution layer.

The loss function employed by the network to learn to produce object boundaries is a variation of the sigmoid cross entropy loss function. This variation of the sigmoid cross entropy loss function is modified so that the ratio of boundary to non-boundary pixels is reflected in the loss calculation:

$$L = -\beta \sum_{j \epsilon Y^+} \log \sigma(y_j = 1|X_j) - (1 - \beta) \sum_{j \epsilon Y^-} \log \sigma(y_j = 0|X_j), \tag{4.1}$$

where $\beta = \{Y_-\}/\{Y\}$ and $1 - \beta = \{Y_+\}/\{Y\}$. Y_- and Y_+ denote the non-boundary and boundary pixels in the groundtruth image respectively, X_j denotes the prediction map, and $\sigma(y_j = 1|X_j)$ is the sigmoid function on the activation value at pixel location j.

The only modification to the spatial stream network is the addition of Graduated Deep Supervised Network [28] (or Graduated DSN) loss weight term during the training phase. Graduated DSN decreases the significance of the side loss functions as the training progresses over the predefined number of epochs and shifts the focus of the network on minimizing the loss associated with the fusion output temporally. Graduated DSN training is defined as:

$$L = (1 - t/T)L_{side} + L_{fusion}, \tag{4.2}$$

where t is the current epoch and T is the total number of epochs.

4.1.2 Dynamic stream

The dynamic stream is identical to the spatial stream in terms of architecture, loss function, and the use of Graduated DSN training; however, due to the nature of the input being optical flow, the dynamic network cannot be initialized with the VGG16 network weights trained on the ImageNet [11] dataset.

To train the network to predict motion boundaries from optical flow, the curriculum learning [4] training scheme must be utilized. In particular, the dynamic stream network training begins with the "easy" task of mapping groundtruth optical flow to motion boundaries, where the motion boundaries

groundtruth is extracted from the groundtruth optical flow by thresholding the norm of the optical flow. Gradually, as training progresses and saturates, the groundtruth optical flow is replaced with optical flow estimations that are less accurate, specifically FlowNet2 [23], then EpicFlow [40], and finally the Classic+NL optical flow estimation method [45] in the motion boundary estimation experiments. This form of curriculum learning is illustrated in Figure 4.2.

When training the convolutional network to detect object boundaries, the curriculum learning took the form of gradually training on increasingly difficult datasets as illustrated in Figure 4.3. Specifically, training on groundtruth optical flow for rigid motions in synthetic data, FlowNet2 optical flow prediction for rigid motions in synthetic data, FlowNet2 optical flow prediction for non-rigid motion in synthetic data, and finally FlowNet2 optical flow prediction for non-rigid motion in real data.

A method to circumvent having to train a network on optical flow from scratch, is to map the optical flow map to a RGB image using the method from [2]. In the method from [2], the optical flow map is normalized based on the maximum flow in both u and v directions, and the colour is extracted from the corresponding coordinate in a colour wheel. This enables the use of the VGG16 weights trained on the ImageNet [11] dataset since the number of channels in the input is now also three.

4.2 Decoder network

Different forms of input provide different signals that may aide in producing a better result than a network relying on a single form of input. An image provides information about textures, colours, and other important signals,

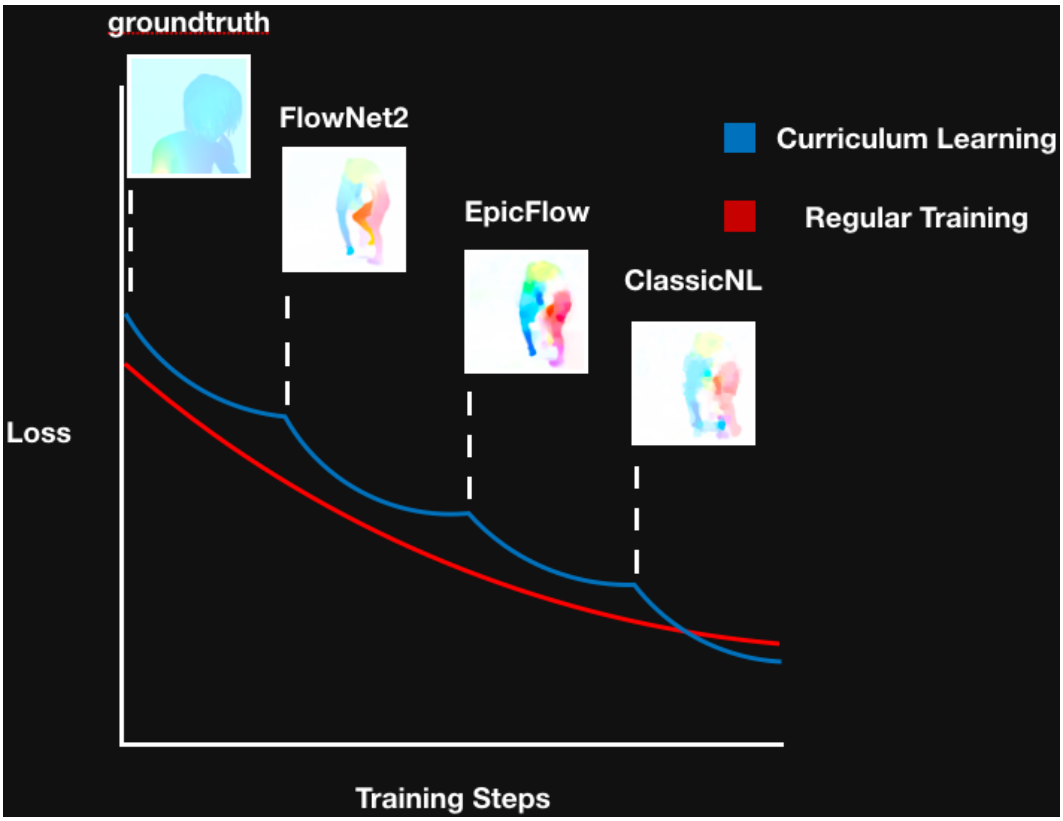

Figure 4.2: An illustration of the curriculum learning training scheme for the dynamic stream. The network first learns to map groundtruth optical flow to motion boundaries, then FlowNet2 optical flow estimation, followed by EpicFlow optical flow estimation, and finally ClassicNL optical flow estimation.

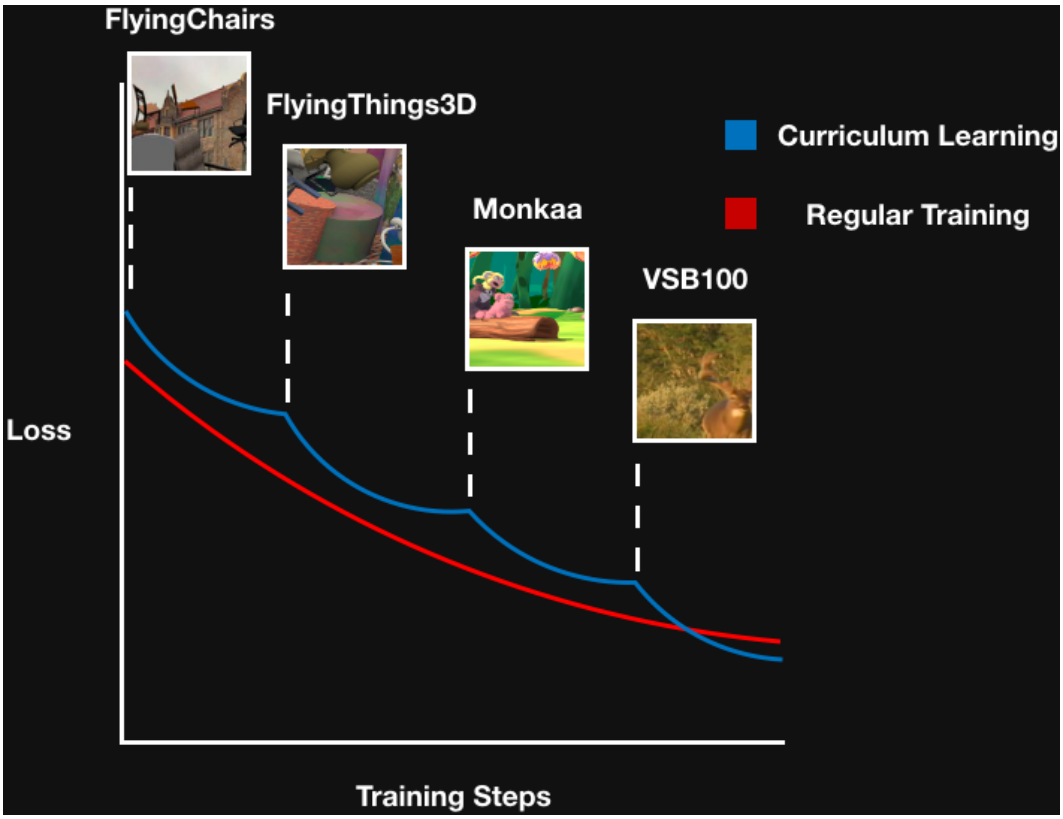

Figure 4.3: An illustration of the curriculum learning training scheme for the spatial stream. The network first learns 2D rigid motions from the synthetic FlyingChairs dataset, then 3D rigid motions from the synthetic FlyingThings3D dataset, following by 3D non-rigid motions from the synthetic Monkaa dataset, and finally 3D non-rigid motions from the real-world VSB100 dataset.

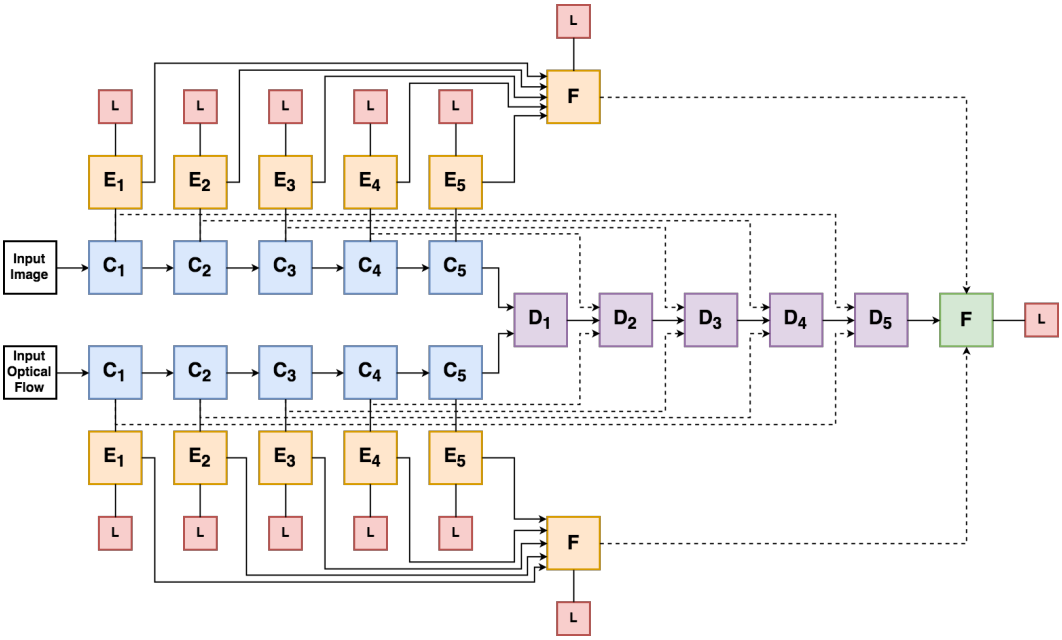

Figure 4.4: The two-stream network consists of two HED networks, where C is a convolution block, E is a side output block, and F is a fusion block (each producing dynamic and static features, respectively). The features from the convolution blocks (C) of each stream are fused together in a series of convolutions and concatenation (D). The final step in the network (green F) is to produce a fusion output. The fusion output can be one of three variations: vanilla fusion, selection fusion, or residual fusion.

while optical flow provides motion-related information. In comparison to the encoder networks mentioned earlier in the chapter, the decoder network learns to fuse the features from each convolutional layer in the encoder networks and fuses them in latent space to produce a more refined object or motion boundary output map. This is shown in Figure 4.4, where the encoder networks provide input to the decoder network and the features of both streams are fused.

Feature fusion in the decoder network is done by concatenating features

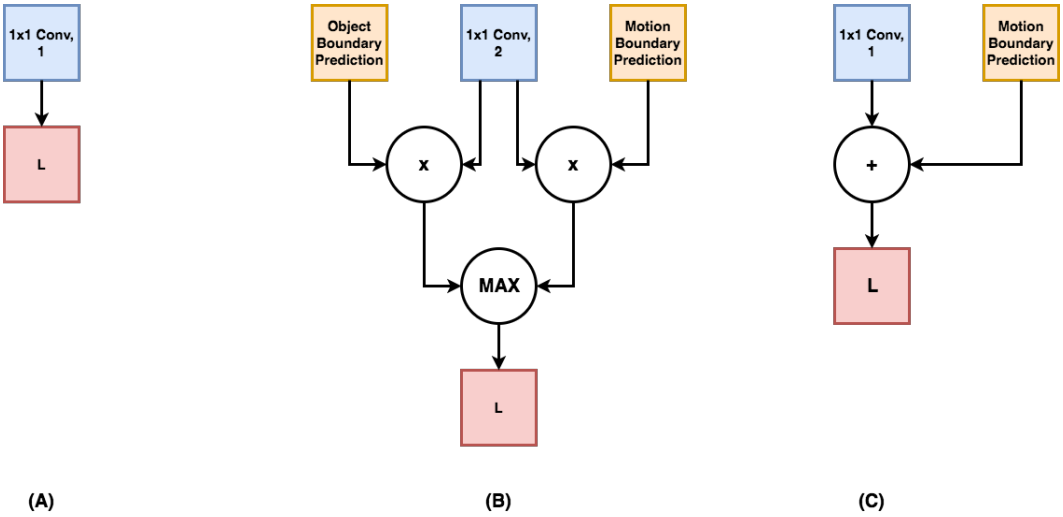

Figure 4.5: The fusion variation. (A) is a representation of the vanilla fusion technique which convolves the input to produce a final output. (B) is the selection fusion which selectively masks out the outputs of the encoder network to produce a final output. (C) is the residual fusion method which combines a learned residual map with the output of the encoder network to produce an output.

produced at the same convolution layer level from both streams, convolving the concatenated feature volume by a 1×1 convolution, and applying a non-linearity to the output of the convolution operation. This method of feature fusion is inspired by previous work [15]. There are three different fusion methods that were experimented with: vanilla fusion, selective fusion, and residual fusion. These fusion methods are illustrated in Figure 4.5 and replace the Fusion placeholder in Figure 4.4.

4.2.1 Vanilla fusion

The vanilla fusion method is the simplest form of fusion, where the decoder network concatenates the feature volumes from previous encoder layers and convolves the concatenated output to produce the final prediction edge map. This method of fusion is similar to a U-Net [48] ConvNet architecture, where features are repeatedly upsampled and concatenated with feature volumes of similar size from previous encoder layers to produce the input volume to the next convolution layer. The final convolution layer outputs a $H \times W$ motion or object boundary map depending on the task.

The vanilla fusion method takes the output of the final interpolation applied to the feature volume as input, and applies a 1×1 convolution to produce a $H \times W$ activation map output. In the case of training, a sigmoid cross entropy loss is applied on the prediction map output to calculate the loss; while in the case of testing, a sigmoid function is applied on the prediction map to produce the edge map output.

4.2.2 Selection fusion

The second fusion method is selective fusion. Selective fusion is inspired by domain knowledge, specifically that motion boundaries are a subset of object boundaries (i.e. motion boundaries are the object boundaries of the moving objects in the scene). The selective fusion method enables the decoder network to learn to produce two masks, a static stream mask and dynamic stream mask, to selectively mask out edges from each of the predictions, object boundary and motion boundary, produced from the encoder networks. These masked out predictions are then combined to produce a refined object or motion boundary map depending on the task.

In the case of selection fusion, the input is also the final interpolation applied to the feature volume. This fusion method, however, applies a 1×1 convolution to produce a $H \times W \times 2$ output volume. This output volume is sliced along the depth, producing two separate, $H \times W$ masks, where each mask corresponds to the a stream in the encoder network. Each mask is multiplied in element-wise fashion to the corresponding boundary map prediction generated earlier in each encoder network. This operation selectively masks out the edges that do not correspond to the desired output boundary prediction map. The products of the mask and boundary prediction element-wise multiplications are then overlayed and an element-wise maximum operation is performed over the overlayed products, producing the activation map of the decoder network. This activation map is then inputed to a sigmoid cross entropy function in the case of training, and a sigmoid function in the case of testing, to produce the final prediction map.

4.2.3 Residual fusion

The final fusion method is the residual fusion method. Inspired by the residual network (ResNet) [20], the decoder network learns to predict a residual map to amplify or suppress certain edge predictions in the encoder predicted edge map. Similar to the vanilla fusion method, the decoder network fuses features from both encoder streams and produces a residual map that is then combined with the object or motion boundary map produced from the encoder stream in a pixel-wise manner.

Similar to the other fusion operations, the residual fusion technique takes as input the final, interpolated feature volume of the decoder network. A 1×1 convolution is applied to the input to produce a $H \times W$ residual map,

filling in the gaps of the encoder network output with predictions produced from the combination of dynamic and static features. This residual map is then combined in an element-wise addition operation with the motion or object boundary prediction map produced in the encoder network to output an activation map. This activation map is used as an input to the sigmoid cross entropy loss function in the training phase, and to the sigmoid function in the testing phase.

4.2.4 Implementation details

This section delves into the implementation details of the thesis. All three fusion methods are placed at the final layer of the decoder network. The architecture of the decoder network is comprised of five concatenation, convolution, and ReLU layers. The input to the first layer is the feature volume output of conv5_3 layer in both the spatial and dynamic encoder networks. These features are then concatenated to produce a $H \times W \times 2D$ volume, where H is the height, W is the width, and D is the dimension of the volume respectively. The concatenation output is then used as input to a convolution layer, with a window size of 3×3, to produce an output of size $H \times W \times D$. Finally, a non-linear activation function is applied to the output of the convolution, and the output volume is upsampled using the bilinear interpolation method. This process is repeated for the subsequent layers with the output of the interpolation as an additional feature volume to be concatenated with the convolution features from the encoder network.

4.3 Summary

This chapter provided the proposed technical approach for object and motion boundary prediction. The encoder network is comprised of a spatial and dynamic stream. The spatial encoder stream learns to produce an object boundary map from a static image input, while the dynamic stream learns to produce motion boundaries from an optical flow map input. The learned features in the encoder network are then combined at each convolution level in the decoder network. The decoder network then predicts the final output by improving upon previous predictions through suppressing or amplifying certain predicted edge pixels.

Chapter 5

Experiments

This chapter presents the empirical evaluation of the proposed approach. The first section (Sec. 5.1) introduces the datasets and metrics used in the empirical evaluation. The second section (Sec. 5.2) presents a quantitative ablative study. The final section (Sec. 5.3) provides a discussion of the results.

5.1 Overview

This section (Sec. 5.1.1) introduces the datasets and the metrics (Sec. 5.1.2) used in the quantitative evaluation of the thesis.

5.1.1 Datasets

This subsection (Sec. 5.1.1) is divided into two categories: motion boundaries and object boundaries. The motion boundaries category provides an overview of the datasets employed in the motion boundaries experiments, while the object boundaries category provides an overview of the datasets employed in the object boundaries experiments.

Motion boundaries

Two motion boundaries datasets were used in the motion boundaries experiments: Sintel [7] and Youtube Motion Boundaries (YMB) [51]. The reason for the selection of the datasets is for comparability purposes with the state-of-the-art approach in producing motion boundaries. Examples of the datasets are found in Figure 5.1.

Sintel is a dataset composed of synthetic frame sequences generated using computer graphics. The dataset includes the clean RGB image, final RGB image (containing motion blur and noise), and ground truth optical flow for each frame in the sequences. While the optical flow test set is not available for use, the training set along with augmentations [13] is sufficient to train the dynamic encoder network. The training set contains 1064 720×1080 RGB frames and groundtruth optical flow vector maps, capturing 23 different scenes. While the groundtruth motion boundaries are not included in the dataset, similar to [51] they are created by thresholding the norm of the optical flow.

Youtube Motion Boundaries (YMB) dataset is composed of 60 real-world videos frames sequences randomly sampled from the YouTube Objects dataset [39, 6], and the Sports1M dataset [27]. The frame sequences contain moderate to large amounts of noise and include a wide range of subjects in the sequences. The underlying assumptions of optical flow are not met, e.g. brightness constancy, making it particularly challenging for optical flow estimation. For each video frame sequence, the middle frame is manually annotated with motion boundaries by three independent annotators.

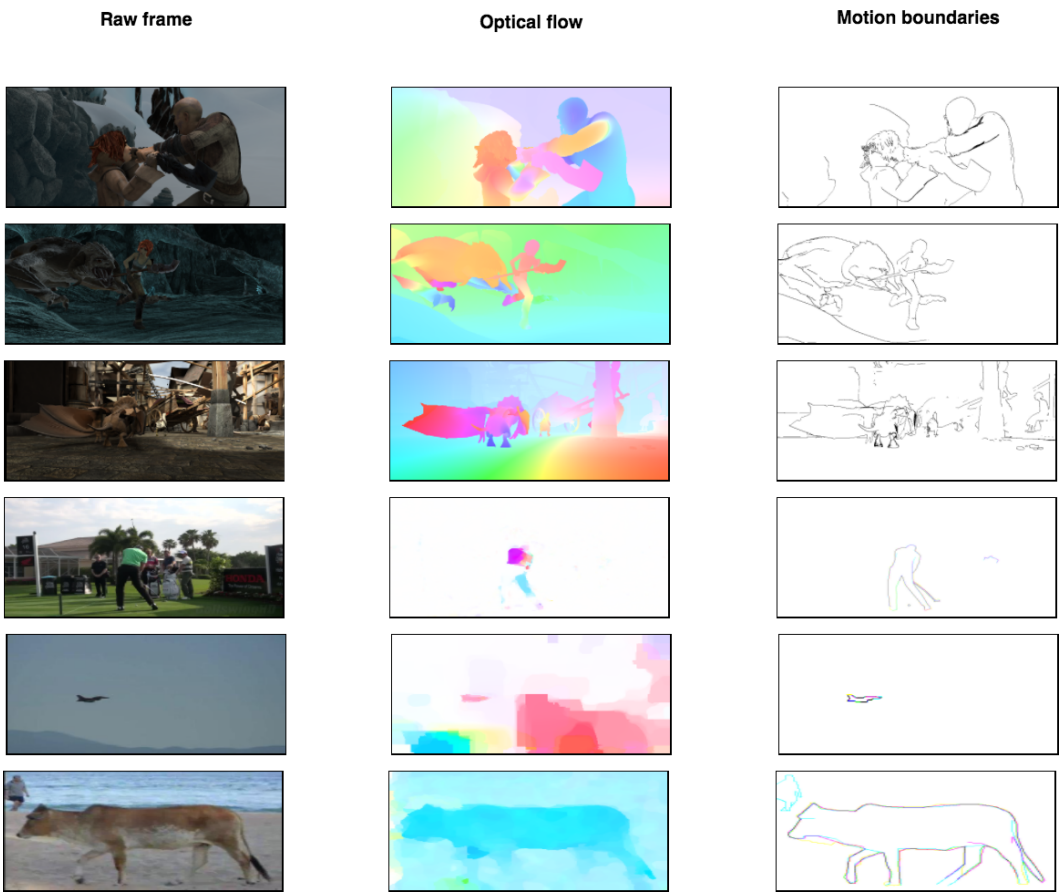

Figure 5.1: The top three rows are from the Sintel dataset. The motion boundaries are extracted from the groundtruth optical flow by thresholding the norm of the optical flow. The bottom three rows are from the YMB dataset. The motion boundaries are provided by annotators, while the optical flow is estimated using the Classic+NL optical flow method.

Object boundaries

Four object boundaries datasets used in the object boundaries experiments: FlyingChairs [13], FlyingThings3D [38], Monkaa [38], and VSB100 [17]. The selection of the stated datasets is due to their gradual increase in difficulty. FlyingChairs provides synthetic, rigid 2D training examples, FlyingThings3D provides synthetic, rigid 3D training and testing examples, Monkaa provides synthetic, non-rigid 3D training and testing examples, and VSB100 provides realistic, non-rigid 3D training examples. Samples of each dataset are found in Figure 5.2.

FlyingChairs is a synthetic dataset made up of 22872 image pairs of chairs in front of random background images. The dataset was generated by applying an affine transformations to images collected from Flickr and a publicly available set of 3D chair renderings. The backgrounds are comprised of 964 images from Flickr from various categories including: 'city', 'landscape', and 'mountain'. These images are then cut into four quadrants and used as background, while the foreground is made up of chair renderings. Motion between the image pairs is generated through randomly sampling a 2D affine transformation matrix for the background images and the chair renderings. The dataset includes the image pairs and the optical flow field between the two image pairs.

FlyingThings3D is a dataset made up of 21818 synthetic frame sequences, ground truth optical flow, ground truth motion boundaries, and ground truth object boundaries. The background of each frame sequence is a large textured plane, made up of 200 generated static objects with shapes that were randomly chosen from cuboids and deformed cylinders that are randomly scaled, rotated, and textured. The foreground objects are comprised of objects from the 35927

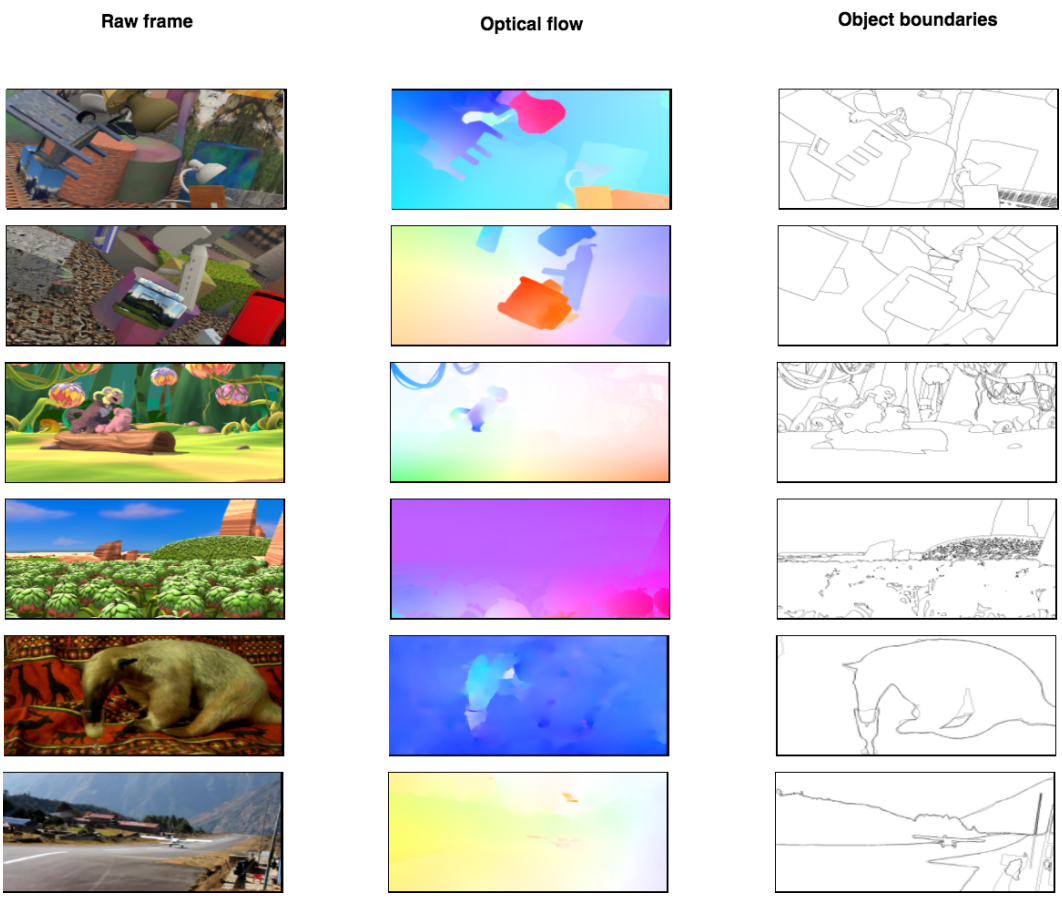

Figure 5.2: The top two rows are from FlyingThings3D dataset which is used to train the dynamic encoder network on optical flow to produce motion boundaries. The middle two rows are from the Monkaa dataset, which provides the network with non-rigid synthetic training examples. The bottom two rows are from the VSB100 training dataset. The ground truth optical flow and annotations are provided by the dataset creators.

3D models from Stanford's ShapeNet6 [9] database. To achieve the 3D aspect of the dataset, the camera and foreground objects were translated and rotated along linear 3D trajectories.

Monkaa is a dataset made up of open source Blender assets of the animated short film Monkaa10. Unlike FlyingThings3D, Monkaa contains non-rigid motion, meaning objects in the scene are not bound to be in the same shape throughout the sequence and can move parts of their bodies. Unlike the Sintel dataset, Monkaa includes 8591 readily available labeled motion and object boundaries along with the raw frame sequences and ground truth optical flow.

VSB100 is a real-world dataset consisting of 102 image sequences containing different scenes and subjects. The dataset is an extension of the BSDS500 dataset, including motion and object boundary annotations for every 20th frame in the sequence done by different, independent annotators. The train and test set are available in full resolution (1280×720) and half resolution (640×360), and are divided into 200 training images and 300 test images.

5.1.2 Data augmentation

This section describes the online data augmentations performed on the input types: RGB frames and optical flow. These augmentations allow the network to have additional training examples that are randomly generated, improving generalization and avoiding overfitting.

RGB frames

To create additional RGB images for training, a geometric and photometric augmentation layer [13] is created to perform online augmentations when the

network is training. The geometric augmentation layer operation is randomly chosen from the following list: rotation between [-17°, 17°], scaling between [0.9, 1.5], translation between [-20%, 20%], and left-right flipping. Any geometric change performed on the input image is also performed on the groundtruth boundary image as well. The photometric augmentation layer include: additive Gaussian noise with a sigma randomly sampled from [0, 0.04], additive brightness using Gaussian with sigma of 0.2, contrast randomly sampled from [-0.8, 0.4], gamma randomly sampled from [0.7, 1.5], and multiplicative colour channel changes randomly sampled from [0.5, 2.0]. These online augmentations are performed on the CPU.

Optical flow

Creating additional optical flow input data is also crucial to improving the performance of the work presented in this thesis. Online geometric augmentations, similar to those mentioned above, are performed on the optical flow input as well. For experiments where RGB optical flow is required as an input, the optical flow is mapped to an RGB image using the method from [2] after augmentation.

5.1.3 Metrics

The performance measure for both motion boundary and object boundary experiments is the mean average precision (mAP). The average precision is calculated for each boundary annotation in the test set and averaged to produce the mAP.

The output of the network is a prediction map of size $H \times W$, where each pixel is a probability of the pixel being labeled a boundary. With this in mind,

the prediction map must be thresholded by a sequence of values to produce edge maps. The thresholds are increments of 0.01 from 0.00 to 1.00.

For every thresholded prediction map, the precision and recall are calculated against the groundtruth boundary map. The precision is the fraction of correctly labeled boundary pixels; The recall is the fraction of retrieved boundary pixels:

$$Precision = \frac{TP}{(TP + FP)}, \tag{5.1}$$

where TP is the number of pixels labeled as boundary in both the groundtruth and the thresholded output, and FP is the number of pixels labeled as non-boundary in the groundtruth and boundary in the thresholded output.

$$Recall = \frac{TP}{(TP + FN)}, \tag{5.2}$$

where TP is the number of pixels labeled as boundary in both the groundtruth and the thresholded output, and FN is the number of pixels labeled as boundary in the groundtruth and non-boundary in the thresholded output.

The average precision is then calculated as the sum, over all the thersholds, of the product of the precision and the change in recall:

$$AP = \sum_{i=0.01}^{1.00} P_i * (R_i - R_{i-0.01}), \tag{5.3}$$

where P_i is the precision calculated at threshold i, R_i is the recall calculated at threshold i, and $R_{i-0.01}$ is the recall calculated at the previous threshold. The average precision is calculated for each set of annotations, and the mean of those average precision are calculated to produce the mean average precision:

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i, \tag{5.4}$$

where N is the number of different annotators, and AP_i is the average precision of annotations i.

5.1.4 Training details

This subsection (Sec. 5.1.4) details the training procedure for each of the networks: spatial encoder network, dynamic encoder network, and the decoder network. The subsection is split into two parts: motion boundaries experiments, and the object boundaries experiments.

Motion boundaries

Due to the sensitivity of the weight initialization and the training scheme of the original VGG16 architecture on the ImageNet dataset, training the spatial encoder network in the motion boundaries experiments from scratch would be extremely time consuming and difficult; Therefore the set of weights in the network were initialized from the weights learned in the HED project. The spatial encoder network was then trained on the Sintel dataset with a learning rate of $1e-7$ for 30,000 iterations, and dividing the learning rate by 10 whenever the validation loss appeared to converge. The other hyperparameters set were: a momentum of 0.9, and a mini-batch size of 10. The input data was geometrically and photometrically augmented once.

In contrast, the dynamic encoder network required to be trained from scratch due to the absence of ConvNet weights trained to produce motion boundaries from an optical flow input. To do so, curriculum learning in the form of optical flow training data had to be heavily relied on throughout the training process. The network weights were first initialized with Xavier initialization, and trained on groundtruth optical flow from the Sintel dataset. The

network was trained for 50,000 iterations with Graduated-DSN, and a learning rate initially set to $1e-8$ that was divided by 10 whenever the validation loss appeared to converge. Next, the network was fine-tuned on FlowNet2 [23], then EpicFlow [40], and then Classic+NL [45] optical flow prediction of the Sintel dataset due to the gradually increasing difficulty with noisier optical flow predictions. The network trained for 50,0000 iterations with an initial learning rate of $1e-8$ that was divided by 10 whenever the validation loss appeared to converge for each optical flow prediction set. The momentum was 0.9, and the mini-batch size was 10. The input optical flow was geometrically augmented once during the training phase.

As for the motion boundaries experiments that utilized the RGB mapping of optical flow, the dynamic encoder network weights are initialized from the weights learned in the HED project. Since the input is a three channel image (RGB), it can be initialized and trained in a similar manner as the spatial encoder network. The only difference between the dynamic encoder trained on the RGBFlow data and the spatial encoder is the augmentation of the input data. The optical flow input is geometrically augmented once and then mapped to a RGB image map.

After both encoder networks are trained, the decoder network training can begin. The decoder network weights were initialized with Xavier initialization. Since the network task is to produce motion boundaries, the spatial encoder weights are fixed, while the dynamic encoder weights were fine-tuned throughout the training phase of the decoder network on the Classic+NL optical flow prediction and RGB input frame of the Sintel training set. The learning rate was set at $1e-8$, with a momentum of 0.9, and a mini-batch size of 10. Training was performed over 30,000 iterations, and the learning rate was divided by 10 when the validation loss appeared to converge.

Object boundaries

The object boundaries spatial encoder was trained similar to the motion boundaries spatial encoder, with the only difference being the dataset used in training. The spatial encoder was initialized with the weights from the HED project and fine-tuned on the VSB100 dataset, a real-world dataset.

As for the dynamic encoder, the network must be trained from scratch. The network is initialized with Xavier initialization and trained with the groundtruth optical flow of FlyingChairs dataset for 30,000 iterations, a learning rate of $1e-7$, a momentum of 0.9, and a mini-batch size of 10. The network is then fine-tuned on FlowNet2 [23] optical flow predictions of the FlyingChairs training dataset, followed by FlowNet2 optical flow predictions of the FlyingThings3D training dataset, FlowNet2 optical flow predictions of the Monkaa training dataset, and finally FlowNet2 optical flow predictions of the VSB100 training dataset. The learning rate in the fine-tuning process on the FlowNet2 datasets were all initialized as $1e-8$, and the learning rate was divided by 10 as the validation loss appeared to converge.

As for the RGBFlow experiments, the method used to train the network was similar to the dynamic encoder method with the exception being the network initialization being the weights from HED [52].

After the encoder networks training was completed, the decoder network weights were initialized with the weights from the encoder networks and trained on the RGB input frame and FlowNet2 optical flow predictions of the VSB100 dataset. The decoder network was trained in a similar manner as the motion boundaries experiment decoder network, with the differences being fixing the dynamic encoder weights and the iteration size reduced to 10,000 due to the size of the training dataset.

5.2 Evaluation

This section discusses the results produced from the motion and object boundaries experiments.

Given the binary ground truth edge map and the sigmoid boundary prediction map, we first apply a non-maximum suppression step on the boundary prediction map. We use the BSDS edge detection code benchmark [37] to calculate the precision and recall for each threshold on the boundary prediction map. A higher threshold retrieves fewer predicted pixels, implying that the recall will be low and the precision will be high. A lower threshold retrieves most pixels in the prediction map, meaning that the recall will be high and the precision will be low. To recover the mean average precision, the average precision for each annotator is calculated and the PR curves for all the annotators are averaged.

The section first details the results of using optical flow vs RGB Flow (Sec. 5.2.1), the effects of using different optical flow estimation methods (Sec. 5.2.2), the the different fusion methods (Sec. 5.2.3), and the effects of fusing dynamic and spatial data to improve object boundaries prediction (Sec. 5.2.4).

5.2.1 Optical flow vs RGB flow

The first evaluation is on the trade-off between using optical flow and RGBFlow as input to the dynamic encoder. The trade-off between using optical flow over the RGBFlow is the length of the training phase. The use of optical flow is dependent on the highly sensitive initialization, and increases the difficulty in training the network as described earlier in the thesis.

As seen in Table 5.1 and Table 5.3, the mAP performance of the networks

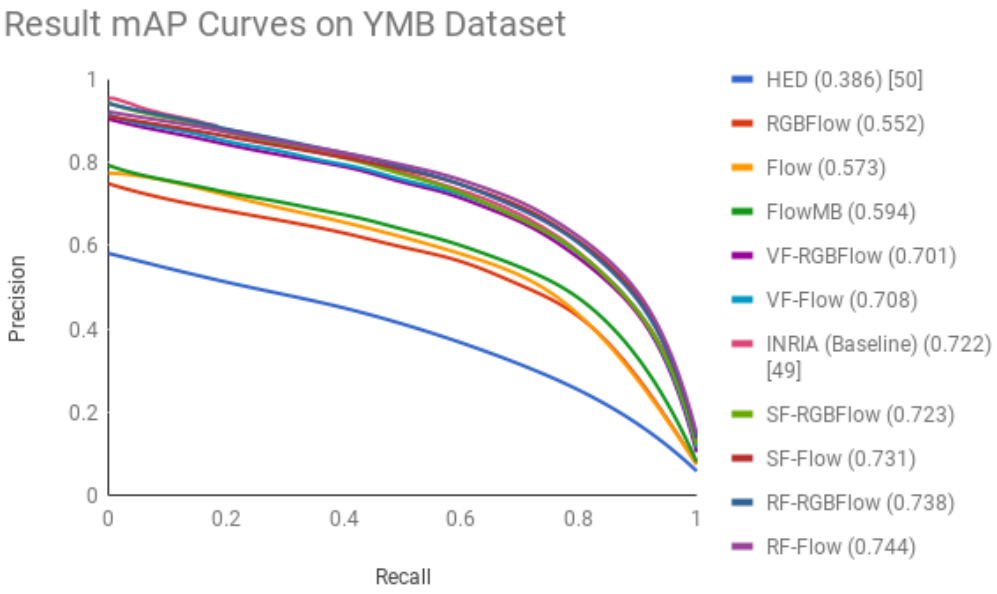

Figure 5.3: The averaged precision-recall curves of the motion boundaries approaches on the Youtube Motion Boundaries dataset. The number in parenthesis indicates the mAP result associated with the approach. HED, represented by [52], indicates the result of the spatial stream with a single RGB frame input. Flow indicates the input was an optical flow prediction, and RGBFlow indicates the input was a colour-mapped optical flow prediction. The FlowMB result was the different thresholds applied on the normalized output using the method from [51]. VF, SF, and RF represent vanilla fusion, selection fusion, residual fusion respectively. The baseline is the result reported by [51], using a RGB image, optical flow, backward optical flow, and the error result from warping the image onto the optical flow.

Method	ClassicNL (mAP)
FlowMB	0.594
INRIA (Baseline) [51]	0.722
HED (RGB) [52]	0.386
Ours (Flow)	0.573
Ours (RGBFlow)	0.552
Ours (VF RGBFlow + RGB)	0.701
Ours (VF Flow + RGB)	0.708
Ours (SF RGBFlow + RGB)	0.723
Ours (SF Flow + RGB)	0.731
Ours (RF RGBFlow + RGB)	0.738
Ours (RF Flow + RGB)	0.744

Table 5.1: Mean average precision results for YouTube Motion Boundaries (YMB) dataset. The test set is comprised of 60 examples, where each test example is made up of three groundtruth annotations done by three different, independent annotators. RGBFlow indicates that the dynamic stream input was a colorized optical flow image. VF, SF, and RF represent vanilla fusion, selection fusion, and residual fusion respectively from Figure 4.2. All inputs to the networks are RGB frames on the spatial stream and Classic+NL flow on the dynamic stream. The output is the merged result from [52]

Optical Flow Method	mAP
Classic+NL [45]	0.744
EpicFlow [40]	0.778
FlowNet2 [23]	0.791

Table 5.2: Mean average precision for YouTube Motion Boundaries (YMB) dataset using different optical flow input. The optical flow methods compared are: FlowNet2 [23], EpicFlow [40], and Classic+NL [45]. The model used is the decoder network with residual fusion trained on the Sintel Clean dataset [7].

that use optical flow as input rather than RGBFlow sees an improvement of ≈ 0.02 mAP. The mAP quantitative results are also summarized in the form of a PR curve in Figure 5.3 and Figure 5.4. This is consistent in all experiments performed: motion boundaries from the dynamic decoder, and motion boundaries from the decoder networks with all fusion methods.

5.2.2 Optical flow estimation methods

The second evaluation to consider is the effect of the optical flow input on the output accuracy; Specifically, the use of better performing optical flow estimation methods (lower endpoint error (EPE)). The optical flow estimation methods that were considered are FlowNet2 [23] (3.959 EPE), EpicFlow [40] (4.115 EPE), and Classic+NL [45] (7.961 EPE). Due to the time and resource cost to prepare the datasets and train the models, this experiment was performed on the YouTube Motion Boundaries [51] dataset after training on the Sintel estimated optical flow dataset [7].

As seen in Table 5.2, the mAP results on the YMB dataset improved

Method	FlyingThings3D	Monkaa	VSB100
HED (Baseline) [52]	0.796	0.748	0.708
HED (FC+VSB100)	-	-	0.643
Ours (RF RGBFlow + RGB)	0.774	0.748	0.692
Ours (RF Flow + RGB)	0.827	0.783	0.731

Table 5.3: Mean average precision results for Object Boundaries. The datasets tested on are: FlyingChairs (FC), FlyingThings3D, Monkaa, and VSB100. Similar to Table 5.1, the fusion method used was Residual Fusion (RF). The training method used was fine-tuning the dynamic stream on FlyingChairs, FlyingThings3D, then Monkaa, and then VSB100 for all networks apart from the second row. HED (FC+VSB100) was fine-tuned only on VSB100.

as the optical flow prediction improved in accuracy. While the difference in EPE between FlowNet2 and EpicFlow for the Sintel dataset is insignificant, the FlowNet2 optical flow predictions on the YMB dataset are more accurate around the boundaries of the moving objects due to the FlowNet2 model architecture specializing in small displacements [23]. Despite Classic+NL optical predictions having an EPE difference of ≈ 4, the mAP difference when compared to the other methods was only ≈ 0.03. This is due to the fact that the YMB dataset contains test samples that do not obey the underlying optical flow assumptions, such as brightness constancy.

5.2.3 Fusion methods

The next evaluation to consider is the different fusion techniques used in the decoder network. This evaluation was performed exclusively on the motion boundaries experiment, as seen in Table 5.1. Due to the time and resource

cost of having to train the decoder network on each fusion technique, the object boundary experiments were only performed using the residual fusion technique.

Referring back to Table 5.1, it is shown that the fusion methods rank ascendantly in terms of mAP performance as: vanilla fusion, selection fusion, and residual fusion. Residual fusion and selection fusion both achieve better scores than the baseline, with 0.744 and 0.722 mAP respectively. It is important to note that the baseline input includes the backward optical flow and the warping error associated with the input optical flow in addition to the inputs used in this thesis.

5.2.4 Dynamic and static fusion

The concept of using dynamic and static input to detect motion boundaries is not new. The work from [51] did so with the use of the random forest decision trees model. The ultimate purpose of this evaluation is analyzing the effects of introducing a dynamic input in addition to a static input to produce a static output with the detection of object boundaries.

Summarized in Table 5.3, it is clearly shown that introducing a dynamic signal in addition to a static signal improves the results of the experiment. Introducing optical flow in addition to the raw frame input improved results by ≈ 0.03 mAP. This demonstrates that different signals provide information that cannot be gathered from static signals and fusing them translates into improvements in static output tasks.

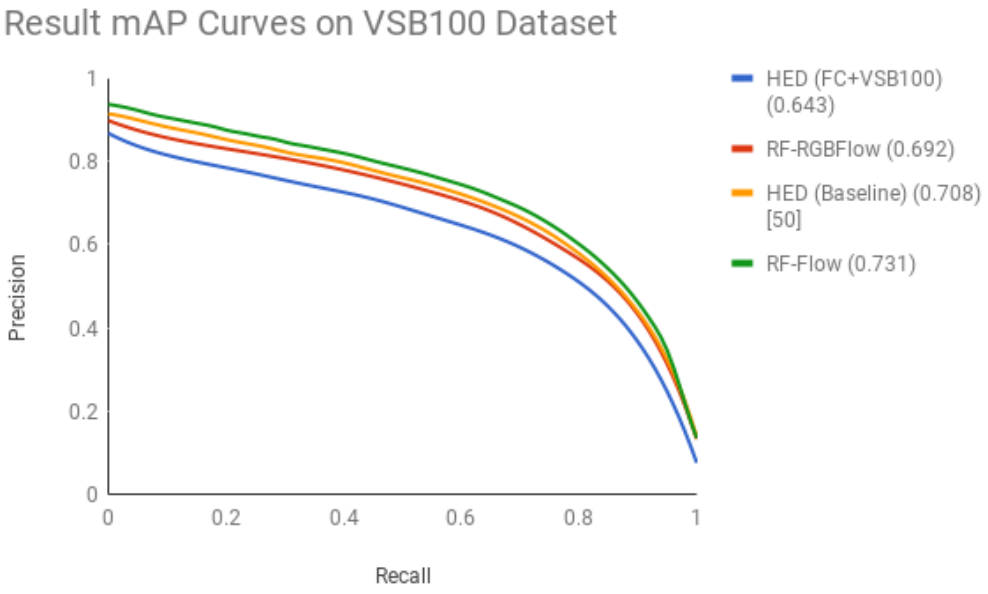

Figure 5.4: The averaged precision-recall curves of the object boundaries approaches on the VSB100 dataset. The baseline HED is the spatial stream fine-tuned on all the datasets: FlyingChairs, FlyingThings3D, Monkaa, and VSB100. HED (FC+VSB100) represents the mAP of the HED network fine-tuned on the FlyingChairs dataset and VSB100 dataset only. RF indicates that residual fusion was used in the model. Flow indicates that the dynamic stream input was optical flow, while RGBFlow indicates the dynamic stream input was a colour-mapped optical flow prediction.

5.3 Discussion

The purpose of this section is to delve deeper into the evaluations that were provided above. The section first discusses the implications of optical flow performing better than RGB flow, then moves on to discuss the results associated with the fusion methods, and finally describes the improvements associated with fusing dynamic and static inputs to produce object boundaries.

As seen earlier in Section 5.2.1, the network produced better results when the input was optical flow rather than RGB flow input. The reason behind this performance boost is the information lost when optical flow is mapped to a RGB image. Specifically, the boundaries of moving objects and discontinuities in optical flow become colours. Depending on the optical flow vector direction and intensity, the colours produced blend together despite that the vectors may belong to different moving objects in the scene.

In Section 5.2.2, the motion boundary estimation results improved as the optical flow prediction input improved. This outcome is due to the increase in accuracy of the optical flow prediction which directly translates to an increase in the detection of objects in motion in the scene. The degree of improvement in mAP is highly contingent on the data obeying the underlying optical flow assumptions, which can be seen in Figure 5.5.

In Section 5.2.3, the evaluations using the different fusion methods showed that residual fusion and selection fusion far outperformed vanilla fusion. This result can be attributed to the difficulty in training a deep network to fuse learned features in earlier layers together to produce a more refined result.

In comparison, the selection fusion method produced better results than the vanilla fusion method and outperformed the state-of-the-art method. Selection

fusion uses the domain knowledge that motion boundaries are a subset of object boundaries, and produces masks that selectively amplify and suppress pixels in the previously predicted motion and object boundaries. Selection fusion fails when the optical flow assumption is not satisfied in the input optical flow, predicting edges despite the optical flow input being incorrect. The incorrectly predicted edges are then selected due to the max operation, which forces the network to assume that the edge prediction in the dynamic encoder to be the correct edge pixels.

On the other hand, residual fusion seems to alleviate issues that arise from inputs where the optical flow assumptions are violated. With the residual fusion method, the network learns a residual map to be combined with the previously predicted edge map, which suppresses or amplifies certain edge predictions. The absence of the max operation enables a "softer" fusion, allowing the network to increase the probabilities of edge pixels in certain areas rather than enforcing a maximum operation over the entire edge prediction map.

Finally, the dynamic and static fusion section (Sec. 5.2.4) demonstrated the impact of fusing dynamic features and static features in the decoder network to produce improved object boundaries. A dynamic input alongside a static input aides the network in suppressing edges that may be related to textures, and increasing the edge probability of pixels related to moving objects in the scene. This is evident in the examples provided in Figure 5.6.

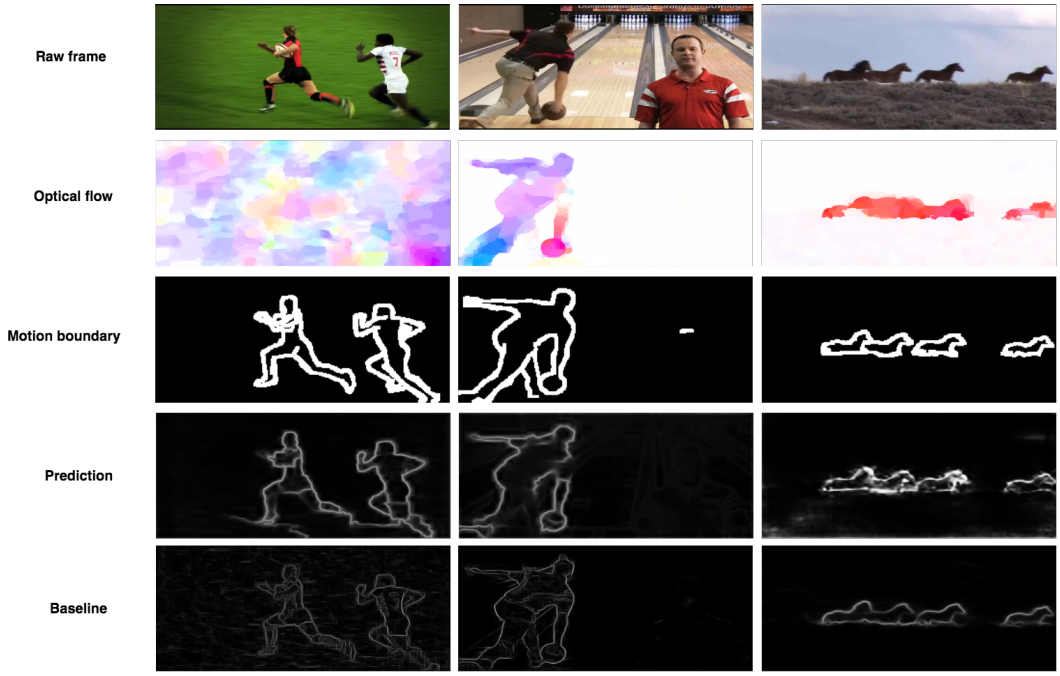

Figure 5.5: Examples of the test set output of the motion boundaries experiments. The first example demonstrates the importance of the static output when the optical flow fails. The second examples illustrates the importance of optical flow when a static input alone would not suffice to produce the correct motion boundaries. The third example illustrates the combination of the static and dynamic stream in order to produce a refined motion boundary.

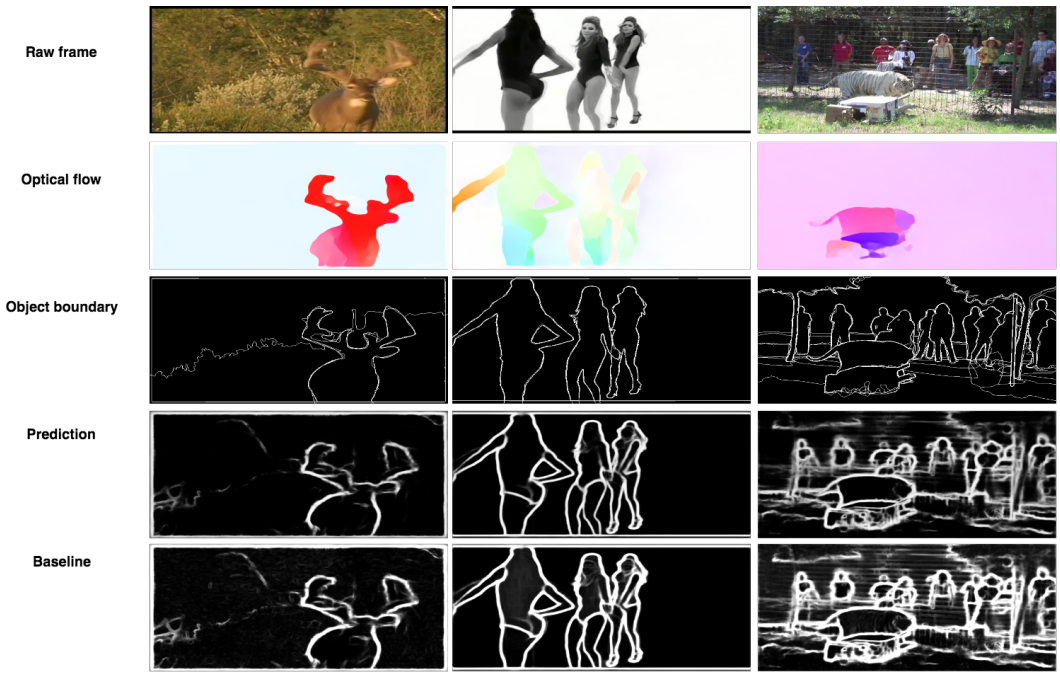

Figure 5.6: Examples of the object boundary VSB100 test set. In all the examples provided, the output is produced through a residual fusion decoder network. The addition of the dynamic stream suppresses the edge probabilities found within the objects.

Chapter 6

Conclusion

This chapter summarizes the work presented in this thesis (Sec. 6.1) and provides a brief overview of directions for future work (Sec. 6.2).

6.1 Thesis summary

This thesis presented the fusion of static and dynamic features to produce motion and object boundaries through three major contributions. Firstly, training a dynamic encoder network which learns, using curriculum learning, to produce a motion boundary prediction map output from an optical flow input. Secondly, an ablative study for different feature fusion techniques that includes a vanilla fusion, selection fusion, and residual fusion method. Thirdly, an encoder and decoder network architecture which learns to fuse dynamic and static features to produce a more refined object or motion boundary prediction map, depending on the target task.

This work addresses the issue of feature fusion to produce object and motion boundaries. Compared to the state-of-the-art approaches, the motion

boundary approach presented in this thesis produced better results and required less inputs. In terms of the object boundary approach presented, the results are systematically higher than those presented in [52] due to the incorporation of the dynamic and static fusion.

The results revealed that fusing dynamic and static features through residual fusion provides improved results compared to those based on a single signal. Another important finding is that the optical flow encoded as a 2D feature map rather than RGB produces better results despite being more difficult to train.

The contributions from this thesis are implemented in the Caffe [26] deep learning framework.

6.2 Future work

There are multiple directions to expand on the current work for different tasks as well as potential improvements on the technical approach.

In terms of technical approach, the state-of-the-art ConvNet architecture [28] employs multiple, multi-scale copies of the HED network and fuses the outputs of the networks to produce a more precise object boundaries prediction map. This approach still relies on single, static images, and would likely benefit from using dynamic inputs along with static inputs to produce more refined outputs and improve on the benchmark.

Another improvement of the technical approach is incorporating side outputs in the decoder network similar to the encoder network side outputs. These side outputs may refine the decoder network outputs further and allow the network to train faster and potentially produce more accurate results.

The work in this thesis experimented with the object and motion boundaries detection task. While this approach provided improvements in these

tasks, the approach may be used to potentially improve tasks that deal with static and dynamic inputs. An example of such tasks is the motion and object segmentation tasks, where motion segmentation is a subset of object segmentation and motion segmentation is a function of dynamic input while object segmentation is a function of static input.

References

[1] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2011.

[2] Simon Baker, Daniel Scharstein, JP Lewis, Stefan Roth, Michael J Black, and Richard Szeliski. A database and evaluation methodology for optical flow. *International Journal of Computer Vision*, 2011.

[3] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*. 2012.

[4] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *International Conference on Machine Learning*, 2009.

[5] Gedas Bertasius, Jianbo Shi, and Lorenzo Torresani. Deepedge: A multi-scale bifurcated deep network for top-down contour detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[6] Thomas Brox and Jitendra Malik. Object segmentation by long term analysis of point trajectories. In *IEEE European Conference on Computer Vision*, 2010.

[7] Daniel J Butler, Jonas Wulff, Garrett B Stanley, and Michael J Black. A naturalistic open source movie for optical flow evaluation. In *IEEE European Conference on Computer Vision*, 2012.

[8] J. Canny. A computational approach to edge detection. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1986.

[9] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[10] Guilhem Chéron, Ivan Laptev, and Cordelia Schmid. P-cnn: Pose-based cnn features for action recognition. In *IEEE International Conference on Computer Vision*, 2015.

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[12] Piotr Dollar and Larry Zitnick. Structured forests for fast edge detection. In *IEEE International Conference on Computer Vision*, 2013.

[13] Alexey Dosovitskiy, Philipp Fischery, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. In *IEEE International Conference on Computer Vision*, 2015.

[14] J. L. Elman. Learning and development in neural networks: The importance of starting small. In *Cognition*, 1993.

[15] C. Feichtenhofer, A. Pinz, and A. Zisserman. Convolutional two-stream network fusion for video action recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[16] Katerina Fragkiadaki, Pablo Arbelaez, Panna Felsen, and Jitendra Malik. Learning to segment moving objects in videos. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[17] F. Galasso, N.S. Nagaraja, T.J. Cardenas, T. Brox, and B.Schiele. A unified video segmentation benchmark: Annotation, metrics and analysis. In *IEEE International Conference on Computer Vision*, 2013.

[18] Georgia Gkioxari and Jitendra Malik. Finding action tubes. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[19] Çağlar Gülçehre and Yoshua Bengio. Knowledge matters: Importance of prior information for optimization. *Journal of Machine Learning Research*, 2016.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[21] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.

[22] Jyh-Jing Hwang and Tyng-Luh Liu. Pixel-wise deep learning for contour detection. *arXiv preprint arXiv:1504.01989*, 2015.

[23] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[24] Suyog Jain, Bo Xiong, and Kristen Grauman. Fusionseg: Learning to combine motion and appearance for fully automatic segmention of generic objects in videos. 2017.

[25] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013.

[26] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[27] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[28] Iasonas Kokkinos. Pushing the boundaries of boundary detection using deep learning. *International Conference on Learning Representations*, 2016.

[29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*. 2012.

[30] Kai A Krueger and Peter Dayan. Flexible shaping: How learning in small steps helps. *Cognition*, 2009.

[31] Yin Li, Manohar Paluri, James M Rehg, and Piotr Dollár. Unsupervised learning of edges. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[32] J. J. Lim, C. L. Zitnick, and P. Dollar. Sketch tokens: A learned mid-level representation for contour and object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2013.

[33] Yu Liu and Michael S. Lew. Learning relaxed deep supervision for better edge detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[34] Yun Liu, Ming-Ming Cheng, Xiaowei Hu, Kai Wang, and Xiang Bai. Richer convolutional features for edge detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[35] Michael Maire, Pablo Arbeláez, Charless Fowlkes, and Jitendra Malik. Using contours to detect and localize junctions in natural images. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[36] D. R. Martin, C. C. Fowlkes, and J. Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2004.

[37] David Martin, Charless Fowlkes, Doron Tal, and Jitendra Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *IEEE International Conference on Computer Vision*, 2001.

[38] Nikolaus Mayer, Eddy Ilg, Philip Hausser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[39] Alessandro Prest, Christian Leistner, Javier Civera, Cordelia Schmid, and Vittorio Ferrari. Learning object class detectors from weakly annotated video. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[40] Jerome Revaud, Philippe Weinzaepfel, Zaid Harchaoui, and Cordelia Schmid. Epicflow: Edge-preserving interpolation of correspondences for optical flow. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[41] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations by error propagation. 1986.

[42] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in Neural Information Processing Systems*, 2014.

[43] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[44] I. Sobel. Camera models and machine perception. Technical report, DTIC Document, 1970.

[45] D. Sun, S. Roth, and M. J. Black. Secrets of optical flow estimation and their principles. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2010.

[46] Lin Sun, Kui Jia, Dit-Yan Yeung, and Bertram E. Shi. Human action recognition using factorized spatio-temporal convolutional networks. In *IEEE International Conference on Computer Vision*, 2015.

[47] Patrik Sundberg, Thomas Brox, Michael Maire, Pablo Arbeláez, and Jitendra Malik. Occlusion boundary detection and figure/ground assignment from optical flow. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

[48] Pavel Tokmakov, Cordelia Schmid, and Karteek Alahari. Learning to segment moving objects. *arXiv preprint arXiv:1712.01127*, 2017.

[49] Yi-Hsuan Tsai, Ming-Hsuan Yang, and Michael J Black. Video segmentation via object flow. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[50] Philippe Weinzaepfel, Jerome Revaud, Zaid Harchaoui, and Cordelia Schmid. Deepflow: Large displacement optical flow with deep matching. In *IEEE International Conference on Computer Vision*, 2013.

[51] Philippe Weinzaepfel, Jerome Revaud, Zaid Harchaoui, and Cordelia Schmid. Learning to Detect Motion Boundaries. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[52] Saining Xie and Zhuowen Tu. Holistically-nested edge detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.