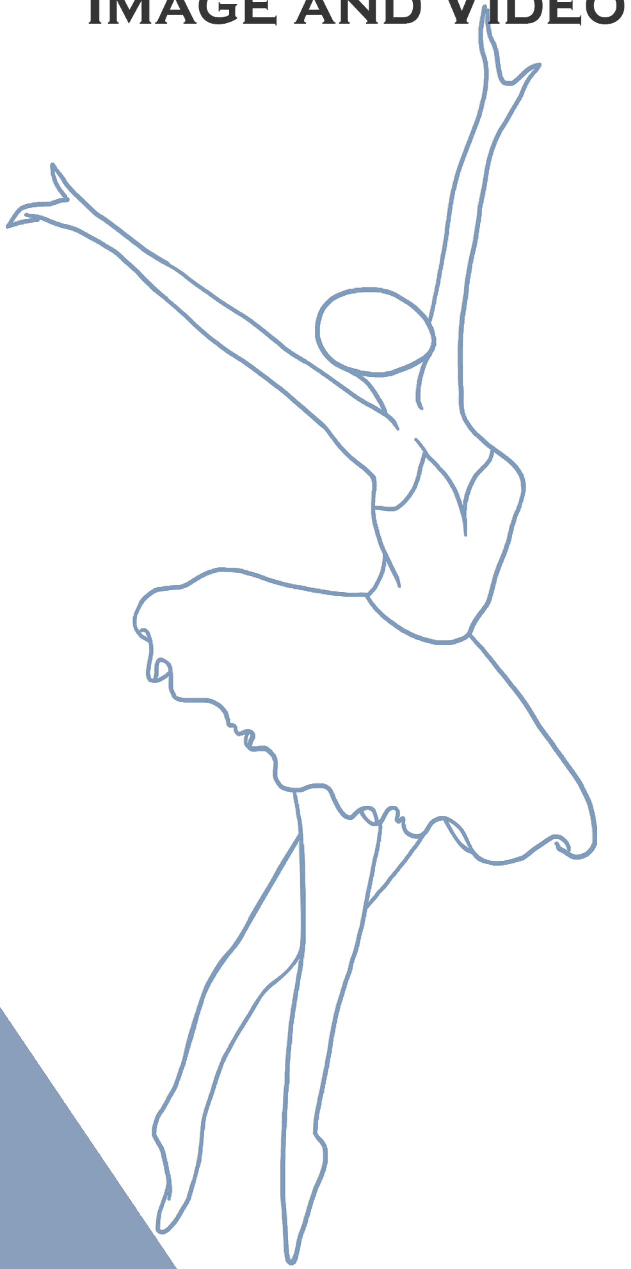
Learning Continuity for Image and Video Recognition Jiaojiao Zhao

Learning Continuity for Image and Video Recognition



JIAOJIAO ZHAO

Learning Continuity for Image and Video Recognition

This book was typeset by the author using $\LaTeX\,2_\varepsilon$.

Learning Continuity for Image and Video Recognition

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. ir. K.I.J. Maex
ten overstaan van een door het college voor promoties
ingestelde commissie,
in het openbaar te verdedigen in de Aula
op dinsdag 22 xx 2022 te 11:00 uur

door

Jiaojiao ZHAO

geboren te Anhui, China

Promotiecommissie

Advanced School for Computing and Imaging

Universiteit van Amsterdam

This work was carried out in the ASCI graduate school, dissertation series number 436.

山河志

西山临雪，旷野茫茫。
远修其道，路阻且长。
草木翛翛，山河泱泱。
数载寒窗，一纸难尝。

日出其灿，星出其皇。
春秋遗志，三千恢章。
浊斯濯足，清斯濯裳。
唯君明月，吾与杜康。

姣

Written at Amsterdam
by Jiaojiao Zhao
2022. 1. 3

Photo credit JAYHUANGPHOTOGRAPHY

Contents

Chapter 1

Introduction

1.1 Continuity in Vision

Biological vision is the marvelous ability of an organism to be informed about its surroundings with a high degree of spatial and temporal resolution [148]. Continuity is the innate nature of biological vision because the real-world experience of an organism is continuous. This property of visual perception determines that human understanding of visual signals is based on continuity in space and time.

Continuity is ubiquitous in the visual world around us. Imagine an image is broken up into hundreds of small pieces and mixed together, it becomes a Jigsaw puzzle. The key cue for solving the puzzle is the spatial continuity in images. For example, in Figure 1.1, one never puts piece B, that has a green color and a grass texture, on area X on the dog. Otherwise, it may violate human common sense, as a dog's fur is naturally never covered with a green grass-like texture. Piece A, which holds the fur-texture and a color that is consistent with the dog's body, fits area X much better. Hence, the continuity in space ensures the semantic coherence in an image, and consequently human understanding of the puzzle image.

Compared to still images, videos, that record visual changes over time, are even better portrayals of the real world. There is substantial evidence that the human

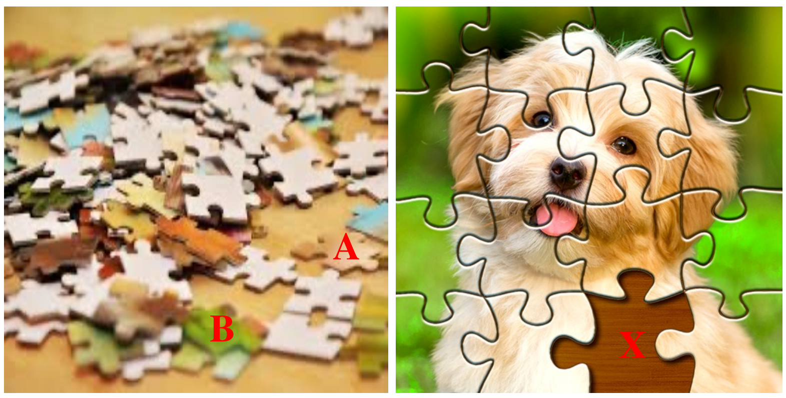
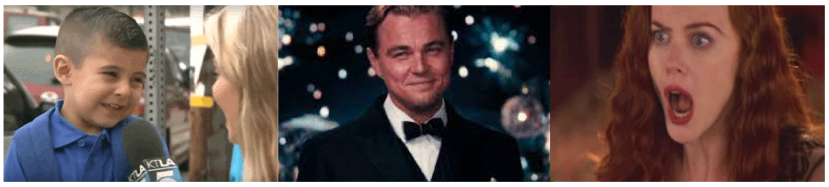
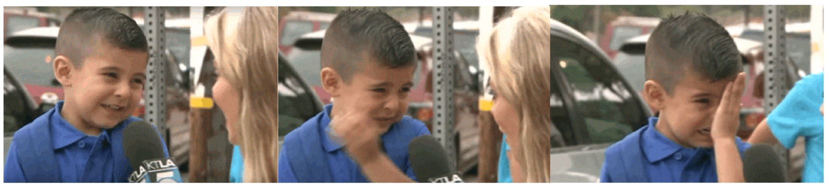


FIGURE 1.1: **Spatial continuity in images**. Placing pieces on correct areas according to pixel coherence in Jigsaw Puzzle.



(a) Independent images



(b) Video clip

FIGURE 1.2: **Spatio-temporal continuity in videos**. (a) Independent images without spatio-temporal continuity. One can only know the state of characters at the current moment but not as part of a coherent story. (b) Video clip with spatio-temporal continuity telling a continuous story.

vision system leverages the continuity in perceivable spatio-temporal dimensions to understand videos [80, 147, 168]. Starting between 1900 and 1910 when film editors utilized editing techniques to create clear and coherent stories, continuity is the most dominant theory throughout film history. In [80], Susan Hayward states that continuity editing produces seamlessness that is used to refer to the Hollywood film style in the name of realism. The spectator is presented with a narrative in such a way that it appears to have no breaks, no disconcerting unexplained transitions in time and space. As illustrated in Figure 1.2, (a) one knows the boy is smiling at this moment, then a man suddenly occurs and a woman is scared in the next shot. It is difficult to understand what happens between them because they are not related in either space or time. By contrast, in Figure 1.2 (b), one clearly sees the boy is initially smiling but later he is crying. The video clip, that is continuous in space and time, tells what is happening. It is noted that temporal continuity makes a video different from a set of independent images.

Akin to its importance for human vision, continuity is an indispensable attribute in computer vision. Continuity has played a vital role since early computer vision was developed, such as in low-level image and video processing. A good example is image denoising [135, 30, 15], which leverages the continuity property of images to suppress noise. Image segmentation [79, 27, 127], as a very basic and classical vision task, relies on coherence of pixel brightness, colors and texture. Besides 2D tasks, 3D reconstruction [146, 62] techniques like Structure from Motion [204] and Simultaneous Localization and Mapping [49] heavily depend on successive camera frames. Typically for videos, temporal continuity is commonly used in video compression formats, for instance, H.262 [52] and MPEG-4 [180]. Additionally, continuity is also the theoretical basis of machine learning models including the

Markov Chain [64], the Long Short-Term Memory [85] and the Recurrent Neural Network [144], which are also widely used in computer vision.

Computers usually can not directly understand raw real-world data such as images, videos, and sensor data. Therefore, features or representations are normally required to learn for visual recognition. As a natural property of images and videos, continuity has shown to be important for learning good representations for visual recognition. For example, spatio-temporal continuity is utilized to achieve efficient and effective representations for video object segmentation [12, 126, 90], tracking [77, 227] and detection [212, 9] in autonomous driving, robot sensing, and augmented-reality. In recent years, the rapid rise of deep learning [73] has reduced the gap between academic research and industrial applications of computer vision considerably. Especially, deep convolutional neural networks (ConvNets) [117] that learn deep representations became omnipresent in almost all vision tasks. Despite ConvNets capturing local continuity in images and videos to some extent, continuity is not fully explored for visual representation. In this thesis, we focus on investigating the benefits of continuity for learning better representations in visual recognition. We structure the thesis into two parts, investigating spatial continuity on images and spatio-temporal continuity on videos.

1.2 Research Questions

Images and videos are the two main modalities in visual recognition. An image is not just a random collection of pixels. It relies on the spatial partitions of pixels corresponding to coherent image properties such as brightness, color and texture [127]. To humans, meaningful regions and objects on images are essentially based on the spatial continuity. And for videos, besides the spatial continuity, temporal coherence is another critical factor. Considering the continuous nature of images and videos, in this thesis, we ask as research question:

What is the benefit of continuity for image and video recognition?

Our research covers several fundamental research challenges in computer vision, including image colorization [97, 236, 94], image classification [40, 117, 45], image semantic segmentation [3, 70, 139], video action detection [18, 229, 198], action recognition [209, 104, 181] and video object segmentation [13, 155, 222]. We observe the continuity in image and video representations is still not fully explored for these computer vision tasks. In this thesis, we dive into our research question by answering a specific sub-question for each of the five chapters.

1.2.1 Part I: Learning Continuity for Image Recognition

Pixels with coherent brightness, color and texture constitute continuous regions or objects that can be understood by humans [127]. An example is shown in Figure 1.3

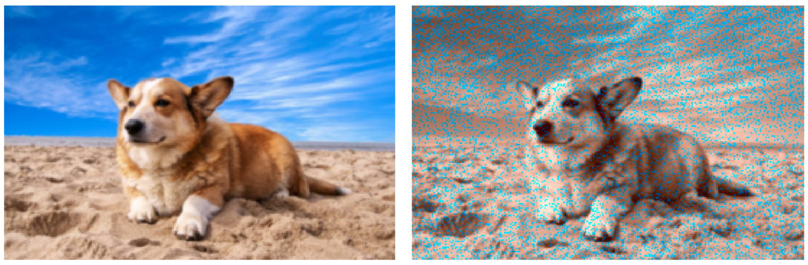

(a) Coherent color (b) Randomly shuffle color

FIGURE 1.3: **Coherent color** (a) provides a more realistic and understandable image, while randomly shuffling color (b) violates the principle of human vision.

(a). If we violate the coherence of colors like in Figure 1.3 (b), the recognition accuracy may significantly drop in computer vision systems [84]. Even a human being is unable to distinguish whether the dog is sitting on 'land' or on 'sand'. In Chapter 2, as a starting point for our investigation into continuity, we research it on the pixel-level image task of image colorization, and pose the following research question:

What is the benefit of spatial continuity for image colorization?

Image colorization requires to reasonably color each pixel in a given gray-scale image. Previous approaches either directly regress color maps [94, 123] or classify each pixel into a color bin [236] using deep ConvNets. Without considering spatial continuity, these methods may suffer from context confusion and edge color bleeding. Inspired by the way humans colorize an image, we propose a solution to produce a more plausible pixel-level color by drawing support from semantic coherence. Semantic coherence is build on the spatial continuity in images. It is explicitly shown in image segmentation [79, 27, 56] and semantic segmentation [149, 60, 234]. The intuition behind our proposal is that we usually assign colors to an object according to what the object is. For instance, sky is normally blue and a dog is naturally not green. The semantics help to discriminate continuous spatial regions of different objects or scenes so that reasonable colors can be assigned to them. To achieve this, we adopt an autoregressive model guided by semantic learning to generate more precise and continuous color maps. An autogressive model predicts color for each pixel based on all the previous seen pixels, which guarantees the spatial continuity from one side. Secondly, simultaneously learning semantic segmentation endows pixel colors with semantic coherence. In the end, our method effectively produces more realistic and finer results showing image colorization profits from spatial continuity. This brings us to the next research question:

How to endow a 2D-ConvNet with spatial continuity?

The ConvNet stacks many layers for large receptive fields. In order to reduce the computation and improve the robustness, pooling is applied to downsize feature maps. However, downsampling is usually a lossy process and weakens the spatial continuity of feature maps. It can be a more serious disadvantage in image-to-image translation tasks, in which up-pooling is required for generating prediction maps with the same size as the input image. The lost information in down-pooling can never be recovered because most existing pooing methods are not invertible. Therefore, the up-pooled feature maps may have many 'holes', which is particularly prevalent in MaxUpPool [167, 7]. The discontinuous feature maps lack details for producing finer results, for example, in semantic segmentation and image colorization. In Chapter 3, we propose an invertible pooling method for better maintaining spatial continuity. Our down-pooling decomposes a feature map into various downsized sub-bands, each of which contains information with different frequencies. Our up-pooling generates more continuous upsampled feature map using these detail sub-bands. Experiments support that our proposal performs well on both image classification and semantic segmentation, and is also more robust to input corruptions and perturbations.

1.2.2 Part II: Learning Continuity for Video Recognition

In the previous two chapters, We have focused on spatial continuity in images. In Part II, we move on to explore continuity for video understanding.

The continuity along the time dimension is the most salient character of videos in vision. Therefore, modeling the temporal continuity is a fundamental question for video understanding. In the computer vision literature, optical flow [86] is traditionally used to represent the continuous change along the time dimension for video tasks. Optical flow is defined as the velocities of movement of brightness patterns in a video. It explicitly models temporal correlations between frames and is deemed useful for continuity (motion) compensation [2]. As an explicit temporal representation, optical flow is widely applied to many video tasks, such as video action detection [18, 229, 198] and video action recognition [209, 104, 181]. Learning only from a single still frame without considering temporal continuity causes partial understanding of a video [181]. Generally, besides training a network from RGB inputs, an additional network is often independently trained on optical flow inputs. To incorporate temporal information, results from the two streams are fused for the final decision. Two-stream methods [181, 211, 154] are proven effective, but not efficient. We are encouraged to the research question:

What is the benefit of temporal continuity for video understanding?

In Chapter 4, we start by exploring a typical video understanding task, spatio-temporal action detection, which requires a system to output when, where and what kind of action occurs during a video. We propose a method to efficiently and effectively utilize optical flow for better action detection in space and time. Instead of training a separate network based on optical flow, we propose a modulation method [39, 156] which applies low-level optical flow features to affect RGB features. More specifically, we learn a set of affine transformation parameters from optical flow features to transform RGB features. In this sense, actions and backgrounds on RGB frames are more distinctive because motion on foreground and background are usually different. This favors the localization of actions. Moreover, performing transformations in an embedding space injects temporal information into the RGB features, which benefits the recognition of action categories. Additionally, by omitting the need for training a whole network on optical flow, our proposal is more efficient with less computation and model parameters. We conclude that video action detection profits from learning temporal continuity.

Apart form optical flow, 3D-ConvNets [104, 21, 55] enable modeling spatio-temporal information in videos. Naturally, we further arrive at the following question:

How to endow a 3D-ConvNet with spatio-temporal continuity?

As we discussed in Chapter 4, the two-stream model for action analysis in video is effective but not efficient due to its requirement to learn two separate networks. 3D-ConvNets are proposed to process videos by integrating spatio-temporal information [104]. However, 3D convolutions aggregate information from the same spatial locations in successive feature maps. Hence, it ignores the motion between successive frames caused by changes in content and/or camera usage. In Chapter 5, we present the aligned 3D convolution which learns spatial offsets from bilinear products of successive temporal feature maps, to compensate for the spatio-temporal discontinuity. The 3D convolution operates on the locations aligned by the learned offsets rather than the original dislocated positions. Indeed, we find the learned offsets resemble feature-level motion representations of the input frames. Without relying on pre-computed optical flow, the aligned 3D convolutions are effective for action recognition and video object segmentation.

Due to the locality of convolutional kernels, the 3D convolutions model the spatio-temporal continuity locally. Referring to [213], capturing non-local dependencies brings significant profits to image and video recognition. Thanks to the self-attention mechanism, the transformer [206] is able to build global relations and has acquired huge success on many vision tasks [20, 1, 45, 138, 248]. This leads us to the following question:

How to endow a transformer with spatio-temporal continuity?

In Chapter 6, we aim for spatio-temporal action detection by designing a transformer-based method, which is capable to handle long-range spatio-temporal information. The task requires to detect action tubelets, each of which contains a sequence of bounding boxes depicting positions of an actor per-frame in a video clip and the categories of the occurring actions. In order to detect action tubelets, most existing methods [106, 240, 185, 131] independently regress each bounding box from the corresponding frame feature, which weakens the temporal coherence between these boxes. We model the tubelet-level action detection as a sequence-to-sequence(s) learning task. Intuitively, we should treat an action tubelet with multiple boxes along time as a whole. Thus, instead of separately localizing each box, our proposal is able to jointly regress these boxes from a single tubelet-level representation, while considering temporal correlations along all the frames in an input video clip. In this sense, our model produces more precise and longer action tubelets.

To summarize, this thesis aims at studying continuity for image and video recognition. In depth, we start with the benefit of learning continuity for images or videos in each part, and then respectively dig into technological innovations of exploiting continuity in various network architectures. In breadth, the thesis explores spatial continuity for images and spatio-temporal continuity for videos. Specifically, it covers image colorization, image classification, semantic segmentation, video action detection, video action recognition, and video object segmentation. We hope our journey is able to stimulate more research on image and video continuity.

1.3 Publications, Co-authorship and Roles

For each chapter of this thesis we here declare the authors' contributions:

Chapter 2

Jiaojiao Zhao, Jungong Han, Ling Shao, Cees G.M. Snoek (2020). *"Pixelated Semantic Colorization"*. In: International Journal of Computer Vision. [243]. A preliminary version appeared in the British Machine Vision Conference 2018. [242].

• J. Zhao	All aspects
• J. Han	Technical support
• L. Shao	Guidance and technical advice
• C.G.M. Snoek	Insight and supervision

Chapter 3

Jiaojiao Zhao, Cees G. M. Snoek (2021). *"LiftPool: Bidirectional ConvNet Pooling"*. In: International Conference on Learning Representations. [241].

• J. Zhao	All aspects
• C.G.M. Snoek	Insight and supervision

Chapter 4

Jiaojiao Zhao, Cees G.M. Snoek (2019). *"Dance with Flow: Two-in-one Stream Action Detection"*. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. [240].

• J. Zhao	All aspects
• C.G.M. Snoek	Insight and supervision

Chapter 5

Jiaojiao Zhao, Cees G.M. Snoek. *"Go with the Flow: Aligned 3D Convolutions for Action Recognition"*. Unpublished.

• J. Zhao	All aspects
• C.G.M. Snoek	Insight and supervision

Chapter 6

Jiaojiao Zhao, Yanyi Zhang, Xinyu Li, Hao Chen, Bing Shuai, Mingze Xu, Kaustav Kundu, Davide Modolo, Yuanjun Xiong, Ivan Marsic, Cees G.M. Snoek, Joseph Tighe (2022). *"TubeR: Tubelet-Transformer for Video Action Detection"*. Accepted in the IEEE/CVF Conference on Computer Vision and Pattern Recognition. [244].

• J. Zhao	All aspects
• Y. Zhang	Designing and technical advice.
• X. Li	Guidance and technical advice
• H. Chen	Technical advice
• B. Shuai	Technical advice
• M. Xu	Technical advice
• C. Liu	Technical advice
• K. Kundu	Technical advice
• Y. Xiong	Technical advice
• D. Modolo	Technical support

• I. Marsic	Technical support
• C.G.M. Snoek	Insight and supervision
• J. Tighe	Guidance and technical advice

Part I

Learning Continuity for Image Recognition

Chapter 2

Pixelated Semantic Colorization

2.1 Introduction

Color has been at the center stage of computer vision for decades, *e.g.* ([194, 31, 157, 109, 175, 141, 207]). Many vision challenges, including object detection and visual tracking, benefit from color ([109, 110, 37, 207]). Consequently, color constancy ([66]) and color correction ([174]) methods may further enhance visual recognition . Likewise, color is commonly added to gray-scale images to increase their visual appeal and perceptually enhance their visual content, *e.g.* ([217, 94, 236, 169, 42]). This chapter is about image colorization.

Human beings excel in assigning colors to gray-scale images since they can easily recognize the objects and have gained knowledge about their colors. No one doubts the sea is typically blue and a dog is never naturally green. Although many objects have diverse colors, which makes their prediction quite subjective, humans can get around this by simply applying a bit of creativity. However, it remains a significant challenge for machines to acquire both the world knowledge and "imagination" that humans possess.

Previous works in image colorization require reference images ([76, 137, 22]) or color scribbles ([128]) to guide the colorization. Recently, several automatic approaches ([94, 123, 236, 169, 75]) have been proposed based on deep convolutional neural networks. Despite the improved colorization, there are still common pitfalls that make the colorized images appear less realistic. We show some examples in Figure 2.1. The cases in (a) without semantics suffer from incorrect semantic understanding. For instance, the cow is assigned a blue color. The cases in (b) without semantics suffer from color pollution. Our objective is to effectively address both problems to generate better colorized images with high quality.

Both traditional ([29, 97]) and recent colorization solutions ([123, 94, 82, 236, 237]) have highlighted the importance of semantics. However, they only explore image-level class semantics for colorization. As stated by [34], image-level classification favors translation invariance. Obviously, colorization requires representations that are, to a certain extent, translation-variant. From this perspective, semantic segmentation ([139, 23, 149]), which also requires translation-variant representations, provides more reasonable semantic guidance for colorization. It predicts a class label for each pixel. Similarly, according to ([236, 123]), colorization assigns each pixel a color

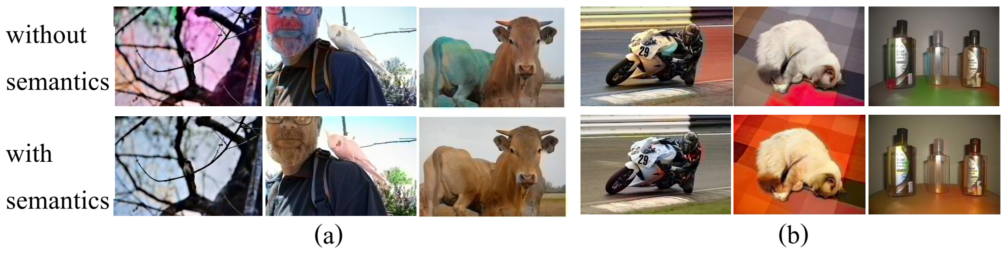


FIGURE 2.1: **Colorization without and with semantics** generated using the network from this chapter. We rescale all output images to their original proportions. (a) The method without semantics assigns unreasonable colors to objects, such as the colorful sky and the blue cow. The method with semantics generates realistic colors for the sky (first column), the man (second column) and the cow (third column). (b) The method without semantics fails to capture long-range pixel interactions ([169]). With semantics, the model performs better.

distribution. Both challenges can be viewed as an image-to-image prediction problem and formulated as a pixel-wise prediction task. We show several colorized examples after using pixelated semantic-guidance in Figure 2.1 (a) and (b). Besides providing sharp boundaries which helps to prevent color bleeding, the color distributions of specific object types enforce additional constraints, which helps to alleviate the ambiguity in color recovery. Together, the fine-grained semantic information helps to precisely colorize specific objects.

In this chapter, we study the relationship between colorization and semantic segmentation. Our proposed network is able to be harmoniously trained for semantic segmentation and colorization. By using such multi-task learning, we explore how pixelated semantics affects colorization. Differing from the preliminary conference version of this work ([242]), we view colorization here as a sequential pixel-wise color distribution generation task, rather than a pixel-wise classification task. We design two ways to exploit pixelated semantics for colorization, one by guiding a color embedding function and the other by guiding a color generator. Using these strategies, our methods produce diverse vibrant images on two datasets, Pascal VOC2012 ([50]) and COCO-stuff ([17]). We further study how colorization can help semantic segmentation and demonstrate that the two tasks benefit each other. We also propose a new quantitative evaluation method using semantic segmentation accuracy.

The rest of the chapter is organized as follows. In Section 2, we introduce related work. Following, in Section 3, we describe the details of our colorization network using pixelated semantic guidance. Experiments and results are presented in Section 4. We conclude our work in section 5.

2.2 Related Work

2.2.1 Colorization by Reference

Colorization using references was first proposed by [217], who transferred the colors by matching the statistic within the pixel's neighborhood. Rather than relying on independent pixels, [97] transferred colors from a segmented example image based on their observation that pixels with the same luminance value and similar neighborhood statics may appear in different regions of the reference image, which may have different semantics and colors. [196] and [29] also performed local color transfer by segmentation. [16] and [76] proposed to transfer colors at pixel level and super-pixel level. Generally, finding a good reference with similar semantics is key for this type of methods. Previously, [137] and [29] relied on image retrieval methods to choose good references. Recently, deep learning has supplied more automatic methods in ([28, 83]). In our approach, we use a deep network to learn the semantics from data, rather than relying on a reference with similar semantics.

2.2.2 Colorization by Scribble

Another interactive way to colorize a gray-scale image is by placing scribbles. This was first proposed by [128]. The authors assumed that pixels nearby in space-time, which have similar gray levels, should have similar colors as well. Hence, they solved an optimization problem to propagate sparse scribble colors. To reduce color bleeding over object boundaries, [92] adopted an adaptive edge detection to extract reliable edge information. [160] colorized manga images by propagating scribble colors within the pattern-continuous regions. [226] developed a fast method to propagate scribble colors based on color blending. [142], further extended [128] by grouping not only neighboring pixels with similar intensity but also remote pixels with similar texture. Several more current works ([237, 177]) used deep neural networks with scribbles trained on a large dataset and achieved impressive colorization results. In all these methods, which use hints like strokes or points, provide an important means for segmenting an image into different color regions. We prefer to learn the segmentation rather than manually labelling it.

2.2.3 Colorization by Deep Learning

The earliest work applying a deep neural network was proposed by [28]. They first grouped images from a reference database into different clusters and then learned deep neural networks for each cluster. Later, [94] pre-trained a network on ImageNet for a classification task, which provided global semantic supervision. The authors leveraged a large-scale scene classification database to train a model, exploiting the class-labels of the dataset to learn the global priors. Both of these works treated colorization as a regression problem. In order to generate more saturated results, [123] and [236] modeled colorization as a classification problem. [236] applied cross-channel encoding

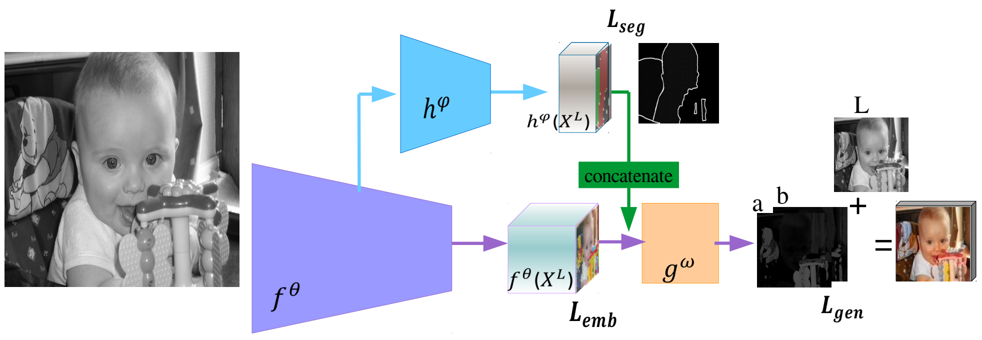


FIGURE 2.2: **Pixelated semantic colorization**. The three colored flows (arrows) represent three variations of our proposal. The purple flow illustrates the basic pixelated colorization backbone (Section 3.1). The purple flow combined with the blue flow obtains a better color embedding with more semantics (Section 3.2.1). The purple flow, blue flow and green flow together define our final model, a pixelated colorization model conditioned on gray-scale image and semantic labels (Section 3.2.2). Here, f^{θ} is a color embedding function, h^{φ} is a semantic segmentation head and g^{ω} is the autoregressive generation model. There are three loss functions L_{seg}, L_{emb} and L_{gen} (Section 3.3).

as self-supervised feature learning with semantic interpretability. [123] claimed that interpreting the semantic composition of the scene and localizing objects were key to colorizing arbitrary images. Nevertheless, these works only explored image-level classification semantics. Our method takes the semantics one step further and utilizes finer pixelated semantics from segmentation.

Further, generative models have more recently been applied to produce diverse colorization results. Currently, several works ([19, 98, 57]) have applied a generative adversarial network (GAN) ([161]). They were able to produce sharp results but were not as good as the approach proposed by [236]. Variational autoencoders (VAE) ([112]) have also been used to learn a color embedding ([42]). This method produced results with large-scale spatial co-ordination but tonelessness. [169] and [75] applied PixelCNN ([151, 173]) to generate better results. We use PixelCNN as the backbone in this chapter.

2.3 Methodology

In this section, we will detail how pixelated semantics improves colorization. We will first introduce our basic colorization backbone. Then, we will present two ways to exploit object semantics for colorization. Our network structure is summarized in Figure 5.1.

2.3.1 Pixelated Colorization

To arrive at image colorization with pixelated semantics, we start from an autoregressive model. It colorizes each pixel conditioned on the input gray image and previously colored pixels. Specifically, a conditional PixelCNN ([151]) is utilized to generate per-pixel color distributions, from which we sample diverse colorization results.

We rely on the CIE **Lab** color space to perform the colorization, since it was designed to be perceptually uniform with respect to human color vision and only two channels **a** and **b** need to be learned. An image with a height H and a width W is defined as $X \in R^{H\times W\times 3}$. X contains n $(= H \times W)$ pixels. In raster scan order: row by row and pixel by pixel within every row, the value of the i-th pixel is denoted as X_i. The input gray-scale image, represented by light channel **L**, is defined as $X^L \in R^{H\times W\times 1}$. The objective of colorization is to predict the **a** and **b** channels $\hat{Y} \in R^{H\times W\times 2}$. Different from the **RGB** color space, **Lab** has the range $[0;100] \times [-127;128] \times [-127;128]$.

To reduce computation and memory requirements, we prefer to produce color images with low resolution. This is reasonable since the human visual system resolves color less precisely than intensity ([87]). As stated in ([169]), image compression schemes, such as JPEG, or previously proposed techniques for automatic colorization also apply chromatic subsampling. The output images can be easily converted back to their original proportions. We can rescale the generated color channels and concatenate them with the original gray channel to produce the final colorized images with their original sizes.

By adopting PixelCNN for image colorization, a joint distribution with condition is modelled as [151]:

$$p(\hat{Y}|X^L) = \prod_{i=1}^{n} p(\hat{Y}_i|\hat{Y}_1, ..., \hat{Y}_{i-1}, X^L). \tag{2.1}$$

All the elementary per-pixel conditional distributions are modelled using a shared convolutional neural network. As all variables in the factors are observed, training can be executed in parallel.

Furthermore, X^L can be replaced by a good embedding learned from a neural network. Taking g^ω as the generator function and f^θ as the embedding function, each distribution in Equation (1) can be rewritten as:

$$p(\hat{Y}_i|\hat{Y}_1, ..., \hat{Y}_{i-1}, X^L) = g_i^\omega(\hat{Y}_1, ..., \hat{Y}_{i-1}, f^\theta(X^L)). \tag{2.2}$$

As the purple flow in Figure 5.1 shows, there are two components included in our model. A deep convolutional neural network (f_θ) produces a good embedding of the input gray-scale image. Then an autoregressive model uses the embedding to generate a color distribution for each pixel. The final colorized results are sampled from the distributions using a pixel-level sequential procedure. We first sample $\hat{Y}_1$ from $p(\hat{Y}_1|X^L)$, then sample $\hat{Y}_i$ from $p(\hat{Y}_i|\hat{Y}_1, ..., \hat{Y}_{i-1}, X^L)$ for all i in $\{2, ...n\}$.

Color embedding $f^\theta(X^L)$			
Module	Resolution	Channels	Dilation
convolution $3 \times 3/1$	128	64	-
Residual block $\times 2$	128	64	-
convolution $3 \times 3/2$	64	128	-
Residual block $\times 2$	64	128	-
convolution $3 \times 3/2$	32	256	-
Residual block $\times 2$	32	256	-
convolution $3 \times 3/1$	32	512	-
Residual block $\times 3$	32	512	2
convolution $3 \times 3/1$	32	512	-
Residual block $\times 3$	32	512	4
convolution $3 \times 3/1$	32	160	-

TABLE 2.1: **Color embedding branch structure**. Feature spatial resolution, number of channels and dilation rate are listed for each module. The gray rows indicate the bottom layers are shared with the semantic segmentation branch (detailed in Table B.2).

2.3.2 Pixelated Semantic Colorization

Intuitively, semantics is the key to colorizing objects and scenes. We will discuss how to embed pixelated semantics in our colorization model for generating diverse colored images.

Pixelated Semantic Embedding

Considering the conditional pixelCNN model introduced above, a good embedding of the gray-scale image $f^\theta(X^L)$ greatly helps to generate the precise color distribution of each pixel. We first incorporate semantic segmentation to improve the color embedding. We use X^S to denote the corresponding segmentation map. Then, we learn an embedding of the gray-scale image conditioned on X^S. We replace $f^\theta(X^L)$ with $f^\theta(X^L|X^S)$. Thus, the new model learns the distribution in Equation (2) as:

$$\begin{aligned} & p(\hat{Y}_i|\hat{Y}_1, ..., \hat{Y}_{i-1}, X^L, X^S) \\ &= g_i^\omega(\hat{Y}_1, ..., \hat{Y}_{i-1}, f^\theta(X^L|X^S)). \end{aligned} \tag{2.3}$$

Here, the semantics only directly affects the color embedding generated from the gray-scale image, but not the autoregressive model.

Incorporating semantic segmentation can be straightforward, i.e., using segmentation masks to guide the colorization learning procedure. Such a way enables the training phase to directly obtain guidance from the segmentation masks, which clearly

Semantic segmentation $h^{\varphi}(X^L)$			
Module	Resolution	Channels	Dilation
convolution $3 \times 3/1$	128	64	-
Residual block $\times 2$	128	64	-
convolution $3 \times 3/2$	64	128	-
Residual block $\times 2$	64	128	-
convolution $3 \times 3/2$	32	256	-
Residual block $\times 2$	32	256	-
convolution $3 \times 3/1$	32	512	-
Residual block $\times 3$	32	512	2
convolution $3 \times 3/1$	32	512	-
Residual block $\times 3$	32	512	2
convolution $3 \times 3/1$	32	#class	6
convolution $3 \times 3/1$	32	#class	12
convolution $3 \times 3/1$	32	#class	18
add	32	#class	-

TABLE 2.2: **Semantic segmentation branch structure**. Feature spatial resolution, number of channels and dilation rate are listed for each module. #class means the number of semantic categories. The gray rows indicate the bottom layers are shared with the color embedding branch (detailed in Table 4.1).

and correctly contain semantic information. However, it is not suitable for the test phase as segmentation masks are needed. Naturally, we can rely on an off-the-shelf segmentation model to gain segmentation masks for all the test images, but it is not elegant. Instead, we believe it is best to simultaneously learn the semantic segmentation and the colorization, making the two tasks benefit each other, as we originally proposed in ([242]).

Modern semantic segmentation can easily share low-level features with the color embedding function. We simply need to plant an additional segmentation branch h^{φ} following a few bottom layers, like the blue flow shown in Figure 5.1. Specifically, we adopt the semantic segmentation strategies from [23]. At the top layer, we apply atrous spatial pyramid pooling, which expoits multiple scale features by employing multiple parallel filters with different dilation rates. The final prediction ($h^{\varphi}(X^L)$) is the fusion of the features from the different scales, which helps to improve segmentation. The two tasks have different top layers for learning the high-level features. In this way, semantics is injected into the color embedding function. By doing so, a better color embedding with more semantic awareness is learned as input to the generator. This is illustrated in Figure 5.1, by combining the purple flow and the blue flow.

Pixelated Semantic Generator

A good color embedding with semantics aids the generator to produce more correct color distributions. Furthermore, the generator is likely to be further improved with semantic labels. Here, we propose to learn a distribution conditioned on previously colorized pixels, a color embedding of gray-scale images with semantics ($f^{\theta}(X^L|X^S)$), and pixel-level semantic labels. We rewrite Equation (3) as:

$$\begin{aligned} &p(\hat{Y}_i|\hat{Y}_1,...,\hat{Y}_{i-1},X^L,X^S) \\ &= g_i^{\omega}(\hat{Y}_1,...,\hat{Y}_{i-1},f^{\theta}(X^L|X^S),h^{\varphi}(X^L)). \end{aligned} \tag{2.4}$$

Intuitively, this method is capable of using semantics to produce more correct colors of objects and more continuous colors within one object. It is designed to address the two issues mentioned in Figure 2.1. The whole idea is illustrated in Figure 5.1 by combining the purple flow with the blue and green flows.

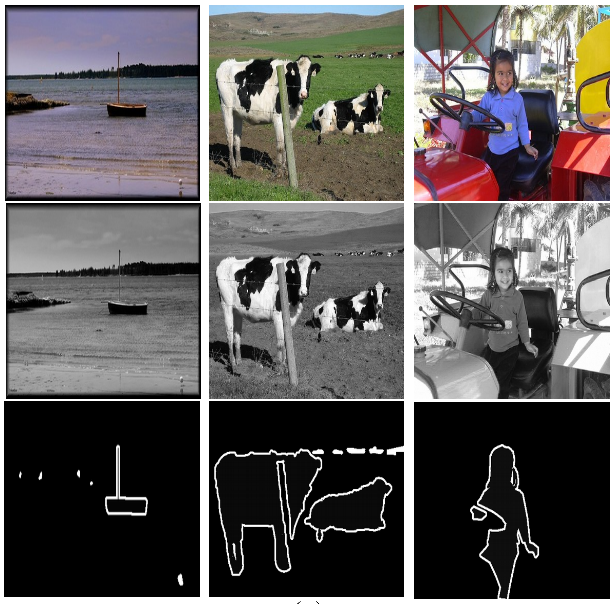
We consider two different ways to use pixelated semantic information to guide the generator. The first way is to simply concatenate the color embedding $f^{\theta}(X^L)$ and the segmentation prediction $h^{\varphi}(X^L)$ along the channels and then input the fusion to the generator. The second way is to apply a feature transformation introduced by [156] and [214]. Specifically, we use convolutional layers to learn a pair of transformation parameters from the segmentation predictions. Then, a transformation is applied to the color embedding using these learned parameters. We find the first way works better. Results will be shown in the Experiment section.

2.3.3 Networks

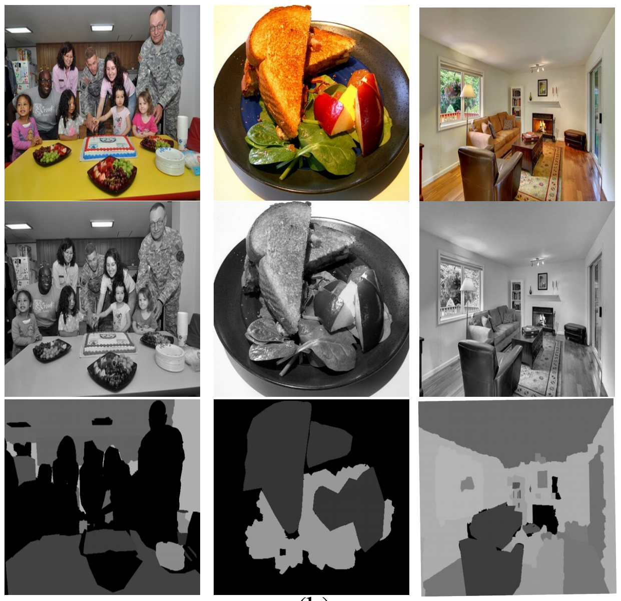
In this section, we provide the details of the network structure and the optimization procedure.

Network structure. Following the scheme in Figure 5.1, three components are included: the color embedding function f^{θ}, the semantic segmentation head h^{φ} and the autoregressive model g^{ω}. Correspondingly, three loss functions are jointly learned, which will be introduced later. The three flows represent the three different methods introduced above. The purple flow illustrates the basic pixelated colorization. The purple flow combined with the blue flow results in the pixelated semantic embedding. The purple flow combined with the blue and green flows, results in the pixelated semantic generator.

Inspired by the success of the residual block ([82, 23]) and following [169], we apply gated residual blocks ([151, 173]), each of which has two convolutions with 3×3 kernels, a skip connection and a gating mechanism. We apply atrous (dilated) convolutions to several layers to increase the network's field-of-view without reducing its spatial resolution. Table 4.1 and B.2 list the details of the color embedding branch and the semantic segmentation branch, respectively. The gray rows are shared by the two branches.



(a)



(b)

FIGURE 2.3: **Color images, gray-scale images and segmentation maps** from (a) Pascal VOC and (b) COCO-stuff. COCO-stuff has more semantic categories than Pascal VOC.

Loss functions. During the training phase, we train the colorization and segmentation simultaneously. We try to minimize the negative log-likelihood of the probabilities:

$$\arg\min_{\theta,\varphi,\omega} \sum -\log p(\hat{Y}|f^{\theta}(X^L), h^{\varphi}(X^L)). \tag{2.5}$$

Specifically, we have three loss functions L_{emb}, L_{seg} and L_{gen} to train the color embedding, the semantic segmentation and the generator, respectively. The final loss function L_{sum} is the weighted sum of these loss functions:

$$L_{sum} = \lambda_1 * L_{emb} + \lambda_2 * L_{seg} + \lambda_3 * L_{gen}. \tag{2.6}$$

Following [173], we use discretized mixture logistic distributions to approximate the distribution in Equation (3) and Equation (4). A mixture of 10 logistic distributions is applied. Thus, both L^{emb} and L^{gen} are discretized mixture logistic losses.

As for semantic segmentation, generally it should be performed in the RGB image domain as colors are important for semantic understanding. However, the input of our network is a gray-scale image which is more difficult to segment. Fortunately, the network incorporating colorization learning supplies color information which in turn strengthens the semantic segmentation for gray-scale images. The mutual benefit among the three learning parts is the core of our network. It is also important to realize that semantic segmentation, as a supplementary means for colorization, is not required to be very precise. We use the cross entropy loss with the standard softmax function for semantic segmentation ([23]).

2.4 Experiments

2.4.1 Experimental Settings

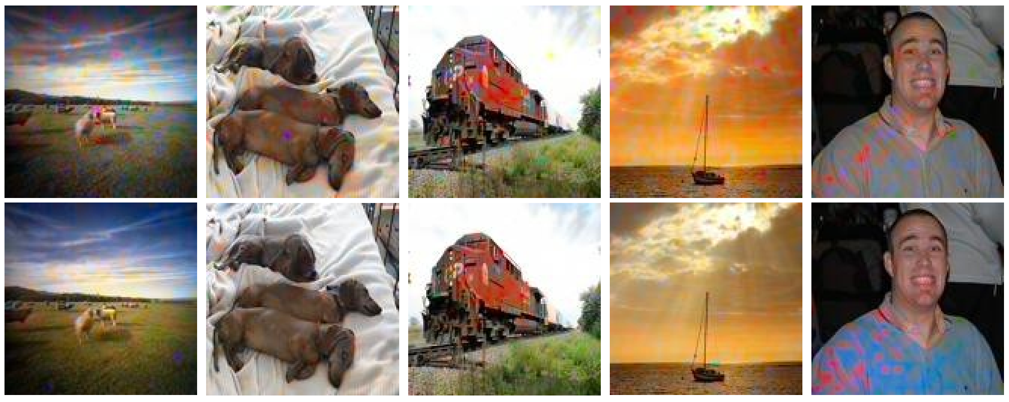
Datesets. We report our experiments on Pascal VOC2012 ([50]) and COCO-stuff ([17]). The former is a common semantic segmentation dataset with 20 object classes and one background class. Our experiments are performed on the 10582 images for training and the 1449 images in the validation set for testing. COCO-stuff is a subset of the COCO dataset ([134]) generated for scene parsing, containing 182 object classes and one background class on 9000 training images and 1000 test images. We train separate networks for Pascal VOC2012 and COCO-stuff. In order to reduce the computation and memory requirements, we rescale each input gray-scale image to 128×128 and produce the color maps with 32×32, as shown in Table 1. The resolution of the color maps is $1/4$ of the input image. Figure 2.3 shows some examples with natural scenes, objects and artificial objects from the datasets.

Implementation. Commonly available pixel-level annotations intended for semantic segmentation are sufficient for our colorization method. We do not need new pixel-level annotations for colorization. We train our network with joint color embedding loss, semantic segmentation loss and generating loss with the weights

$\lambda_1 : \lambda_2 : \lambda_3 = 1 : 100 : 1$, so that the three losses are similar in magnitude. Our multi-task learning for simultaneously optimizing colorization and semantic segmentation effectively avoids overfitting. The Adam optimizer ([111]) is adopted. We set an initial learning rate equal to 0.001, momentum to 0.95 and second momentum to 0.9995. We apply Polyak parameter averaging ([159]).

2.4.2 Effect of segmentation on the embedding function f^θ

We first study how semantic segmentation helps to improve the color embedding function f^θ. Following the method introduced in Section 3.2.1, we jointly train the


(a)

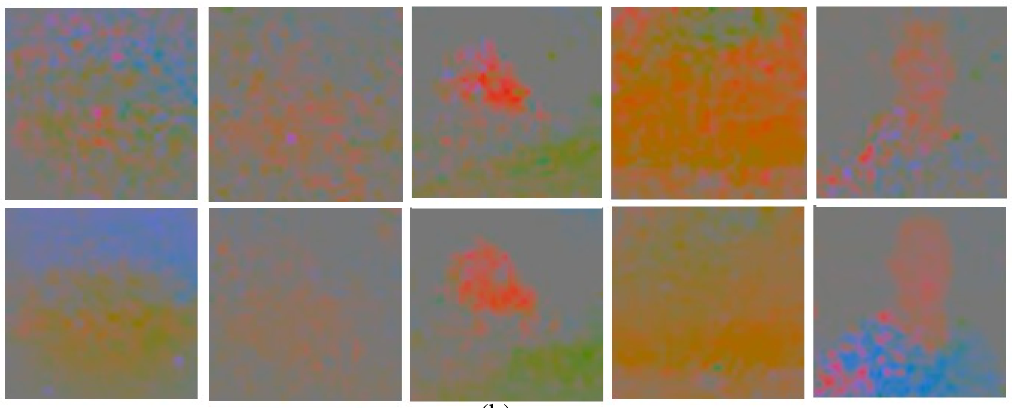
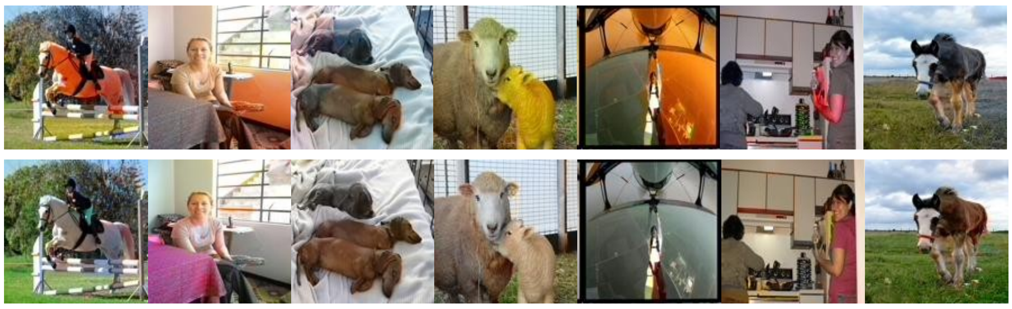
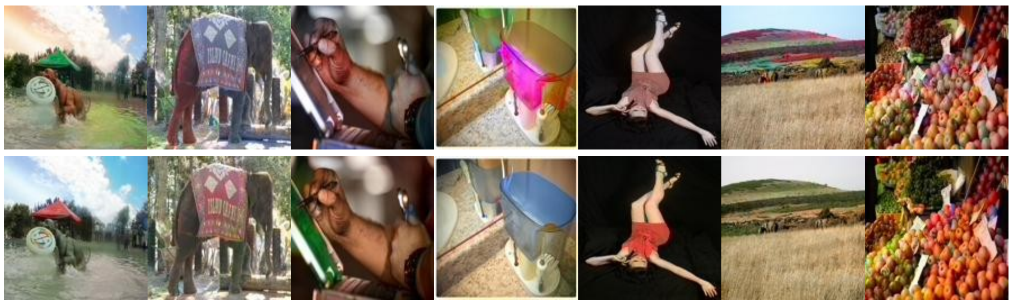

(b)

FIGURE 2.4: **Colorizations from the embedding functions f^θ using the purple flow and the purple-blue flow.** (a) Colorization without semantic-guidance (first row) and with semantic-guidance (second row). With semantics, better colorizations are produced. (b) Visualization of the predicted **a** and **b** color channels of the colorizations. The top row shows the results without semantic-guidance and the bottom row with semantic-guidance. With semantics, the predicted colors have less noise and look more consistent.



(a)



(b)

FIGURE 2.5: **Colorization from the generators** g^{ω}, when relying on the purple flow and the purple-blue flow. Examples from (a) Pascal VOC and (b) COCO-stuff are shown. For both datasets, the top row shows results from the model without semantic-guidance and the bottom row shows the ones with semantic-guidance. The results with semantic-guidance have more reasonable colors and better object consistency.

purple and blue flows shown in Figure 5.1. In this case, the semantic segmentation branch only influences the color embedding function. To illustrate the effect of pixelated semantics, we compare the color embeddings generated from the embedding function f^{θ} in Figure 2.4. Obviously, as can be seen, semantic-guidance enables better color embeddings. For example, the sky in the first picture looks more consistent, and the sheep are assigned reasonable colors. However, the results without semantic-guidance appear less consistent. For instance, there is color pollution on the dogs and the sheet in the second picture.

Further, in order to more clearly show the predicted color channels of the color embeddings, we remove the light channel **L** and only visualize the chrominances **a** and **b** in Figure 2.4 (b). Interestingly, without semantic-guidance, the predicted colors are more noisy, as shown in the top row. However, with semantic-guidance, the colors are more consistent and echo the objects well. From these results, one clearly sees that colorization profits from semantic information. These comparisons support our idea and illustrate that pixelated semantics is able to enhance semantic understanding, leading to more consistent colorization.

In theory, we should obtain better colorization when a better color embedding is

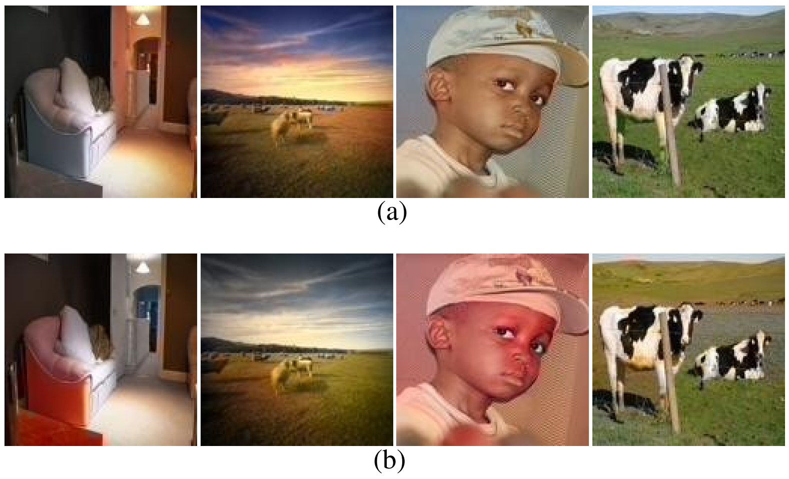


FIGURE 2.6: **Incorporating pixelated semantics.** Result comparisons between using (a) concatenation and (b) feature transformation to incorporate pixelated semantics. Concatenation generates more natural color images.

input into the generator. In Figure 2.5, we show some final colorizations produced

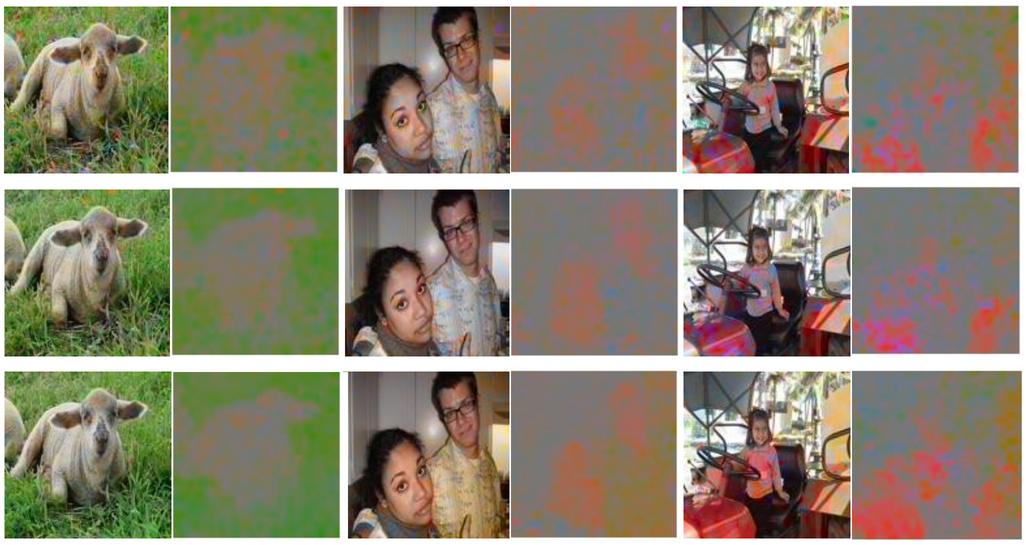


FIGURE 2.7: **Colorizations generated by the embedding functions f^{θ}, using three variants of our network**. The top row shows the results of the purple flow. The second row shows the results of the purple-blue flow. The bottom row shows the results of the purple-blue-green flow. Each colorization is followed by the corresponding predicted chrominances. The purple-blue-green flow produces the best colorization.

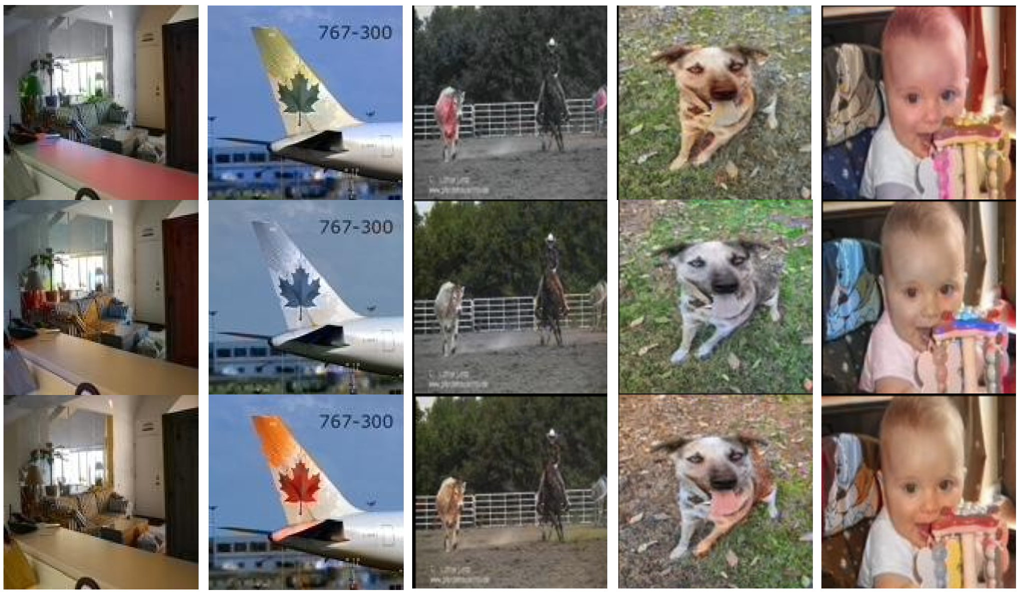
by the generator g^{ω}. Our method using pixelated semantics works well on the two datasets. The results look more realistic. For instance, the fifth example in the Pascal VOC dataset is a very challenging case. The proposed method generates consistent and reasonable color for the earth even with an occluded object. For the last example in Pascal VOC, it is surprising that the horse bit is assigned a red color although it is very tiny. The proposed method processes details well. We also show various examples from COCO-stuff, including animals, humans, fruits, and natural scenes. The model trained with semantics performs better. Humans are given normal skin color in the third and fifth examples. The fruits have uniform colors and look fresh.

2.4.3 Effect of segmentation on the generator g^{ω}

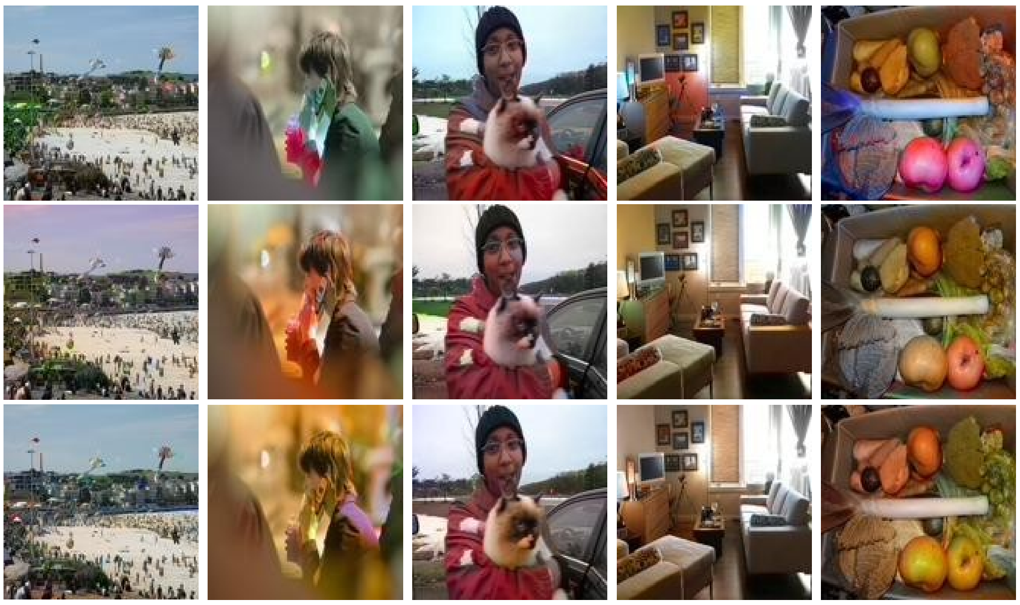
In the next experiment, we add semantics to the generator as described in Section 3.2.2 (combining the purple flow with the blue and green flows). This means the generator produces a current pixel color distribution conditioned not only on the previous colorized pixels and the color embeddings from the gray image, but also on the semantic labels. We apply two different ways to incorporate semantics as introduced in section 3.2.2. Using concatenation generates more natural colorful images than using feature transformation. Qualitative results are shown in Figure 2.6. We prefer concatenation for the following experiments.

As we train the three loss functions L^{emb}, L^{seg} and L^{gen} simultaneously, we want to know whether the color embeddings produced by the embedding function are further improved. In Figure 2.7, we compare the color embeddings generated by the embedding functions of the purple flow (shown in the top row), the purple-blue flow (shown in the second row) and the purple-blue-green flow (shown in the bottom row). Visualizations of color embeddings followed by the corresponding predicted chrominances are given. As can be seen, the addition of the green flow further improves the embedding function. From the predicted **a** and **b** visualizations, we observe better cohesion of colors for the objects. Clearly, the colorization benefits from the multi-task learning by jointly training the three different losses.

Indeed, using semantic labels as condition to train the generator results in better color embeddings. Moreover, the final generated colorized results will also be better. In Figure 2.8, we compare the results from the three methods: pixelated colorization without semantic guidance (the purple flow), pixelated semantic color embedding (the purple-blue flow), and pixelated semantic color embedding and generator (the purple-blue-green flow). The purple flow does not always understand the object semantic well, sometimes assigning unreasonable colors to objects, such as the cow in the third example of Pascal VOC, the hands in the second example and the apples in the last example of COCO-stuff. In addition, it also suffers from inconsistency and noise on objects. Using pixelated semantics to guide the color embedding function reduces the color noise and somewhat improves the results. Adding semantic labels to guide the generator improves the results further. As shown in Figure 2.8, the purple-blue-green flow produces the most realistic and plausible results. Note that it is particularly apt at



(a)



(b)

FIGURE 2.8: **Colorizations produced by the generators g^{ω}, using three variants of our network** on (a) Pascal VOC and (b) COCO-stuff: the purple flow (first row), the purple-blue flow (second row) and the purple-blue-green flow (third row). Using pixel-level semantics to guide the generator in addition to the color embedding function achieves the most realistic results.

processing the details and tiny objects. For instance, the tongue of the dog is red and the lip and skin of the baby have very natural colors.

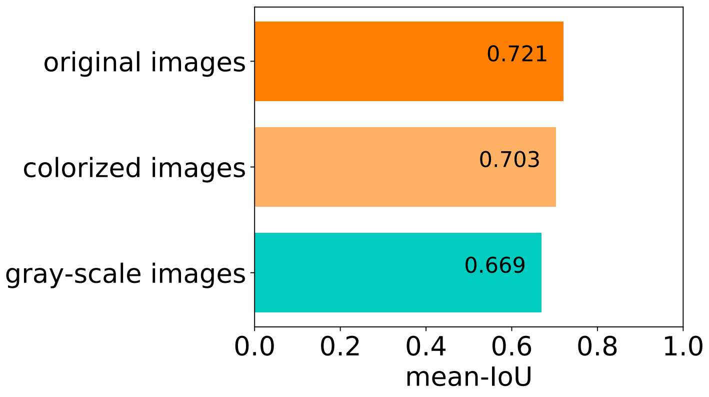


FIGURE 2.9: **Segmentation results** in terms of mean-IoU on gray-scale images, proposed colorized images and original color images, on the Pascal VOC2012 validation dataset. Color aids semantic segmentation.

To conclude, these experiments demonstrate our strategies using pixelated semantics for colorization are effective.

2.4.4 Effect of colorization on the segmentation

From the previous discussion, it is concluded that semantic segmentation aids in training the color embedding function and the generator. The color embedding function and the generator also help each other. As stated in Section 3, the three learnings could benefit each other. Thus, we study whether colorization is able to improve semantic segmentation.

Color is important for semantic segmentation. As we observed in ([242]), color is quite critical for semantic segmentation since it captures some semantics. A simple experiment is performed to stress this point. We apply the Deeplab-ResNet101 model ([23]) without conditional random field as post-processing, trained on the Pascal VOC2012 training set for semantic segmentation. We test three versions of the validation images, including gray-scale images, original color images and our colorized images. The mean intersection over union (mean-IoU) is adopted to evaluate the segmentation results. As seen in Figure 2.9, with the original color information, the accuracy of 72.1% is much better than the 66.9% accuracy of the gray images. The accuracy obtained using our proposed colorized images is only 1.8% lower than using the original RGB images. This again demonstrates that our colorized images are realistic. More importantly, the proposed colorized images outperform the gray-scale images by 3.4%, which further supports the importance of color for semantic understanding.

Colorization helps semantic segmentation. In order to illustrate how colorization influences semantic segmentation, we train three semantic segmentation models on gray-scale images using our network structure: (1) We jointly train semantic

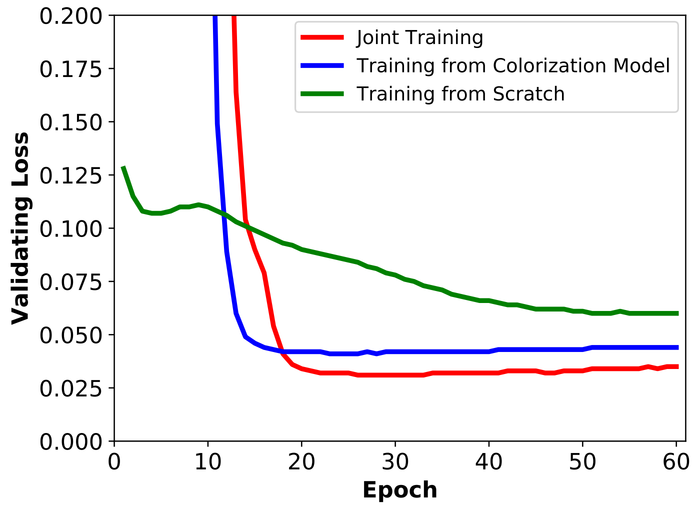


FIGURE 2.10: **Semantic segmentation validating loss comparisons**. Three models are trained for 50 epochs. Training from a pre-trained colorization model is better than training from scratch. Jointly training obtains the lowest validating loss, which demonstrates colorization helps to improve semantic segmentation.

segmentation and colorization; (2) we only train semantic segmentation from a pre-trained colorization model; (3) we only train semantic segmentation from scratch. We train all models on the training set of Pascal VOC 2012 and test them on the validation set. As validating loss reflects the semantic segmentation accuracy on the validation set, we compare the validating loss of the three models.

As seen in Figure 2.10, the model trained on a pre-trained colorization model converges first. The loss is stable from the 18-th epoch and the stable loss value is about 0.043. The model trained from scratch has the lowest starting loss but converges very slowly. Starting from the 55-th epoch, the loss plateaus at 0.060. As expected, the pre-trained colorization model makes semantic segmentation achieve better accuracy. We believe the colorization model has already learned some semantic information from the colors, as also observed by [236]. Further, our multi-task model jointly trained with semantic segmentation and colorization obtains the lowest validating loss of 0.030, around the 25-th epoch. This supports our statement that the two tasks with the three loss functions are able to be learned harmoniously and benefit each other.

2.4.5 Sample Diversity

As our model is capable of producing diverse colorization results for one gray-scale input, it is of interest to know whether or not pixelated semantics reduces the sample diversity. Following [75], we compare two outputs from the same gray-scale image with multiscale structural similarity (SSIM) ([215]). We draw the distribution of SSIM scores for all the compared pairs on the Pascal VOC validation dataset. As shown in

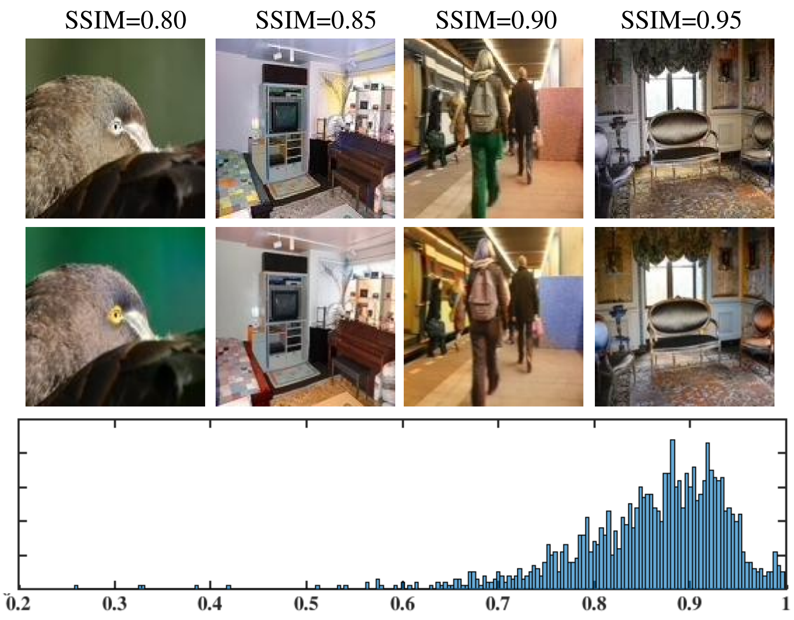


FIGURE 2.11: **Samples diversity**. Histogram of SSIM scores on the Pascal VOC validation dataset shows the diversity of the multiple colorized results. Some examples with their specific SSIM scores are also shown. Our model is able to produce appealing and diverse colorizations.

	mean-IoU (%)	**PSNR(dB)**	**RMSE**
Iizuka *et al.* [94]	67.6	24.20	10.66
Larsson *et al.* [123]	68.8	**24.56**	**10.26**
Zhang *et al.* [236]	68.1	22.81	12.37
Ours	**70.3**	23.15	11.43
Ground-truth (color)	*72.1*	NA	NA

TABLE 2.3: **Quantitative evaluation**. Comparisons of semantic segmentation accuracies, and PSNRs and RMSEs between colorized results and the ground-truth, on the Pascal VOC validation set. Our method performs better according to the mean-IoU value.

Figure 2.11, most of the output pairs have an SSIM score between 0.8 $\sim$ 0.95. The examples shown in the figure demonstrate the pairs have the same content but different colors for details, such as the eyes of the bird and the pants of the lady. Usually, the large backgrounds or objects with different colors in a pair of outputs cause lower SSIM scores. For instance, the backgrounds and birds in the first example. We believe pixelated semantics does not destroy the sample diversity. We will show more diverse colorization results in the next section.

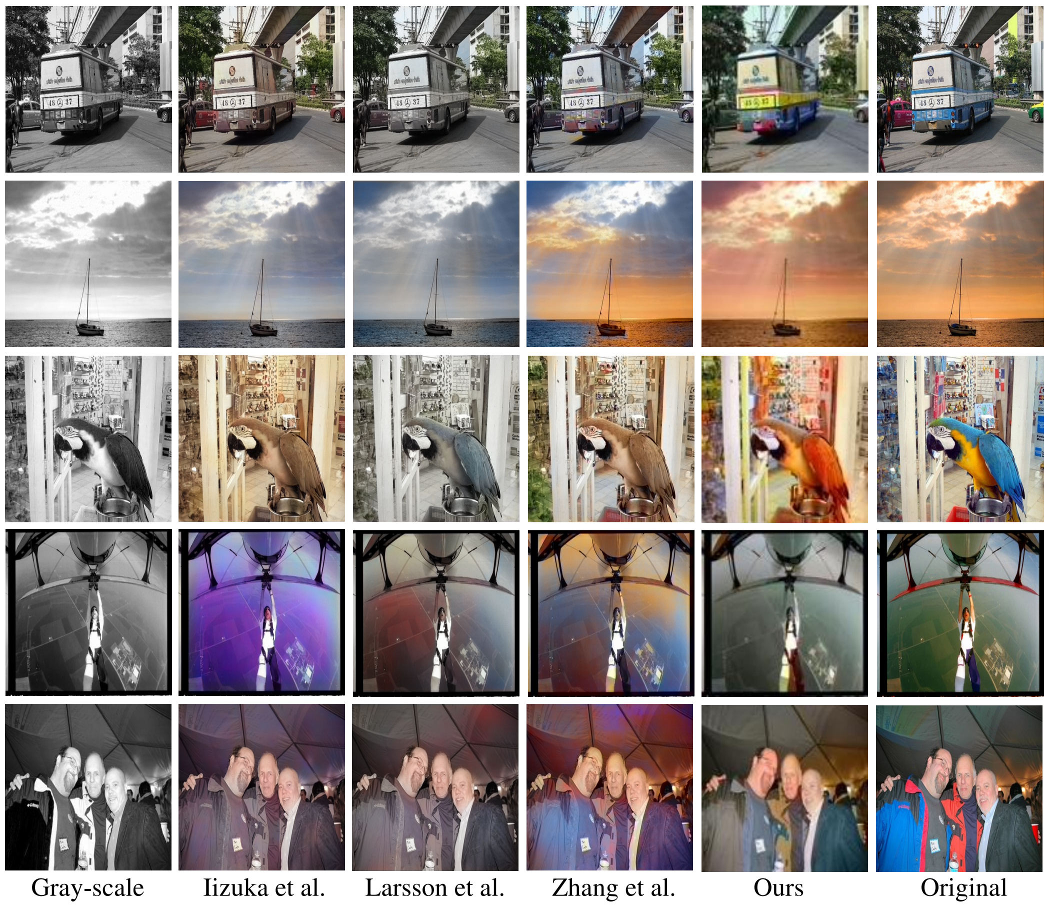


FIGURE 2.12: **Comparisons with single colorization state-of-the-art**. Our results look more saturated and realistic.

2.4.6 Comparisons with State-of-the-art

Generally, we want to produce visually compelling colorization results, which can fool a human observer, rather than recover the ground-truth. As discussed previously, colorization is a subjective challenge. Thus, both qualitative and quantitative evaluations are difficult. As for quantitative evaluation, some papers ([236, 94]) apply *Top-5* and/or *Top-1* classification accuracies after colorization to assess the performance of the methods. Other papers ([83, 123]) use the peak signal-to-noise ratio (PSNR), although it is not a suitable criteria for colorization, especially not for a method like ours, which produces multiple results. Also, color fidelity is suitable to be used for evaluating methods generating a single colorization result, which share a common goal to provide a color image closer to the original one. In ([41, 123]), the authors apply root mean squared error (RMSE) of the 2-channel images compared to the ground-truth to evaluate color fidelity. For qualitative evaluation, human observation is mostly used ([236, 94, 83, 169, 19]).

	Naturalness (%)
Single colorization	
Iizuka *et al.* [94]	88.61
Larsson *et al.* [123]	86.99
Zhang *et al.* [236]	88.66
Diverse colorization	
Deshpande *et al.* [42]	75.30
Cao *et al.* [19]	80.00
Royer *et al.* [169]	89.89
Ours	**94.65**
Ground-truth	*99.58*

TABLE 2.4: **Qualitative evaluation**. Comparisons of the naturalness. Our colorizations are more natural than others.

In this chapter, we propose a new evaluation method. We use semantic segmentation accuracy to assess the performance of each method, since we know semantics is key to colorization. This is more strict than classification accuracies. Specifically, we calculate the mean-IoU for semantic segmentation results from the colorized images. We use this procedure to compare our method with single colorization methods. For qualitative evaluation, we use the method from our previous work ([242]). We ask 20 human observers, including research students and people without any image processing knowledge, to do a test on a combined dataset including the Pascal VOC2012 validation and the COCO-stuff subset. Given a colorized image or the real ground-truth image, the observers should decide whether it looks natural or not.

Single Colorization State-of-the-art

We compare the proposed method with the single colorization state-of-the-art ([236, 94, 123]). In addition to the proposed semantic segmentation accuracy evaluation, we also report PSNR. We report RMSE of the 2 color channels **a** and **b** compared to the ground truth. We use the Deeplab-ResNet101 model again for semantic segmentation. In this case, we only sample one result for each input, using our method.

Result comparisons are shown in Table 2.3. Our method has a lower PSNR than [94] and [123]. The comparisons of RMSE are similar with that of PSNR. Both depend on the ground-truth and over-penalize semantically plausible results with a colorization that differs from the ground-truth ([83]). Both [94] and [123] obtain lower RMSEs because their objective is to minimize the distance between their outputs and ground-truth. However, our method outperforms all the others in semantic segmentation accuracy. This demonstrates that our colorizations are more realistic and contain more perceptual semantics.

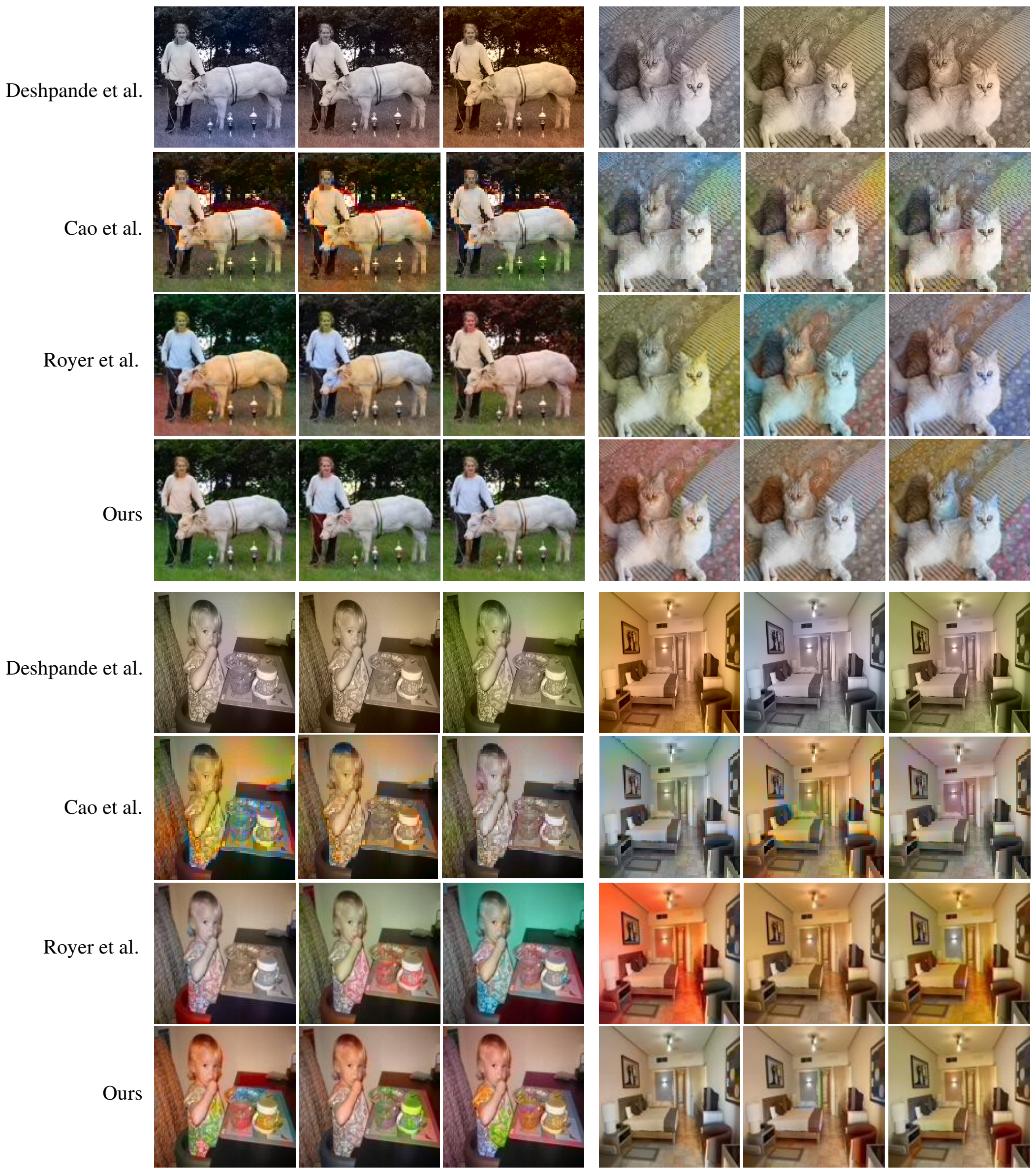


FIGURE 2.13: **Comparisons with diverse colorization state-of-the-art**. The diverse results generated by our method look fairly good.

For qualitative comparison, we report the naturalness of each method according to 20 human observations in Table 2.4. Three of the single colorization methods perform comparatively. Our results are more natural. Selected examples are shown in Figure 2.12. The method by [94] produces good results, but sometimes assigns unsuitable colors to objects, like the earth in the fourth example. The results from [123] look somewhat grayish. [236]'s method can generate saturated results but suffers from color pollution. Compared to these, our colorizations are spatially coherent and visually appealing. For instance, the color of the bird in the third example and the skin

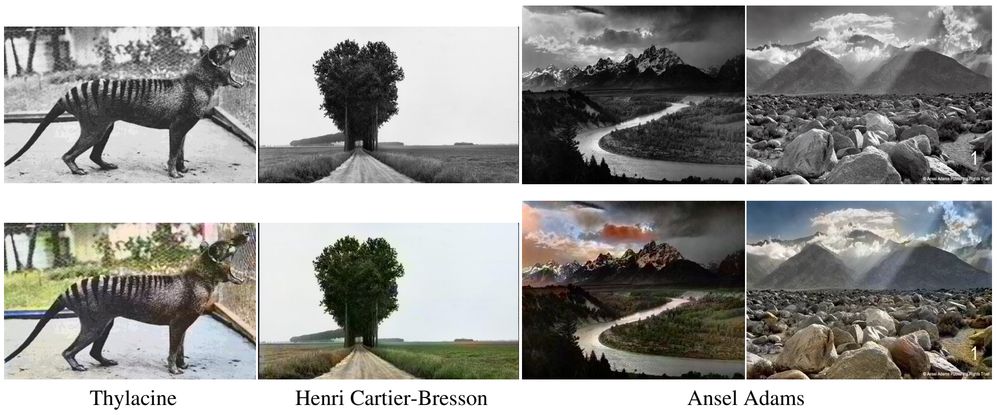


FIGURE 2.14: **Results on legacy black and white photos**. The model also works well on old black and white photos.

of the human in the last example, both look very natural.

Diverse Colorization State-of-the-art

We also compare our method with the diverse colorization state-of-the art ([169, 19, 42]). All of these are based on a generative model. We only qualitatively compare these by human observation. We use each model to produce three colorized samples. We report the results in Table 2.4. [169] apply PixelCNN to get natural images. Our results are even more natural. Several examples are shown in Figure 2.13. [42], using a VAE, generate sepia toned results. [19], applying a GAN, output plausible results but with mixed colors. [169] also produces saturated results but with color pollution. Our generated colored images have fine-grained and vibrant colors and look realistic.

Results on Legacy Black and White Photographs

We also colorize some legacy black and white photographs from renowned photographers Henri Cartier-Bresson and Ansel Adams, along with Thylacine which went extinct in 1936. Results are shown in Figure 14. The model also works well on old black and white photos.

2.4.7 Result Analysis

To further analyze the effect of object semantics on colorization, we quantify the colorization performance for each object class in the Pascal VOC validation set. We report RMSEs per object in Figure 15. Naturally, RMSE is highest on background as no specific object semantics is utilized for this class when we train our model. A person is the most difficult object to colorize as the colors of clothing are so diversiform. Low RMSEs for other objects, like bicycle and sheep, illustrate that incorporating semantics helps to precisely assign reasonable colors to objects.

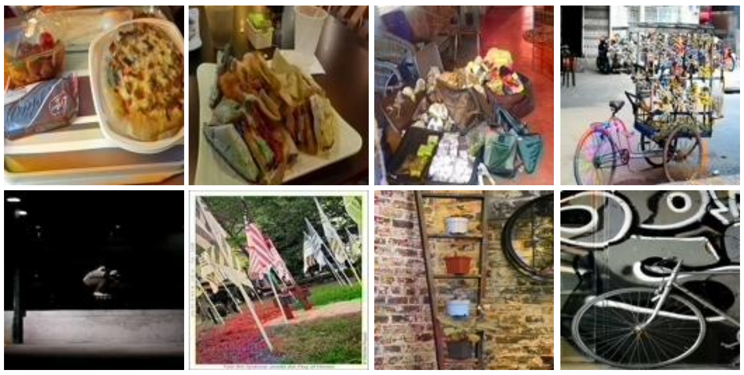


FIGURE 2.15: **Failure cases**. Food, tiny objects and artificial objects are still very challenging.

2.4.8 Failure Cases

Our method is able to output realistic colorized images but it is not perfect. There are still some failure cases encountered by the proposed approach as well as other automatic systems. We provide a few failure cases in Figure 2.15. Usually, it is highly challenging to colorize different kinds of food. They are artificial and variable. It is also difficult to learn the semantics of images containing several tiny and occluded objects. Moreover, our method cannot handle the objects with unclear semantics. Although we exploit semantics for improving colorization, we do not have very many categories. We believe a finer semantic segmentation with more class labels will further enhance the results.

2.5 Conclusion

We propose pixelated semantic colorization to address a limitation of automatic colorization: object color inconsistency due to limited semantic understanding. We study how to effectively use pixelated semantics to achieve good colorization. Specifically, we design a pixelated semantic color embedding and a pixelated semantic generator. Both of these strengthen semantic understanding so that content confusion can be reduced. We train our network to jointly optimize colorization and semantic segmentation. The final colorized results on two datasets demonstrate the proposed strategies generate plausible, realistic and diverse colored images. Although we have achieved good results, our system is not perfect yet and has some challenges remaining. For instance, it cannot well process images with artificial objects, like food, or tiny objects. More learning examples and finer semantic segmentation may further improve the colorization results in the future.

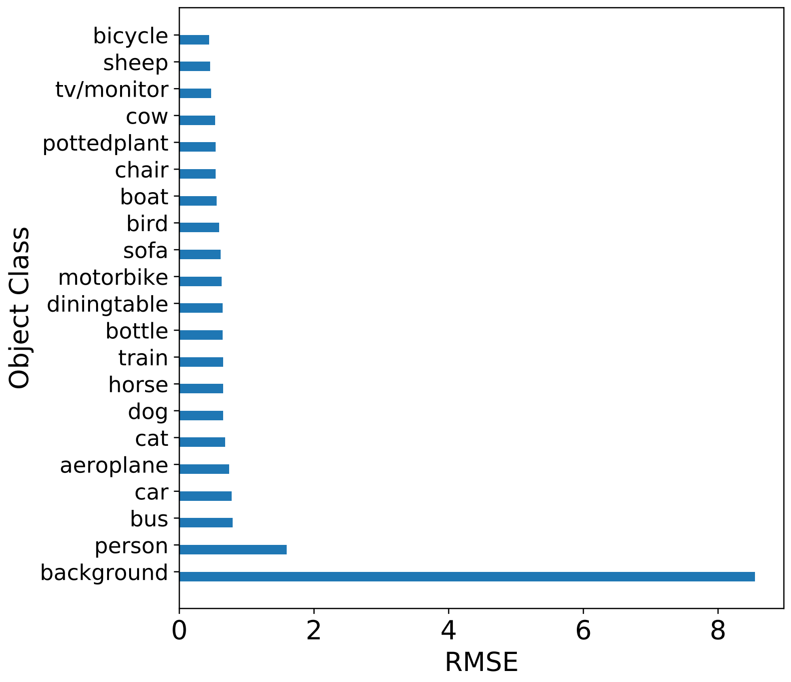


FIGURE 2.16: **Colorization results of per object class on the Pascal VOC validation set using RMSE.** Without considering background, the model performs worse for persons as the clothing has too much diversity in color. The RMSEs are lower for all other objects. Our model helps to assign reasonable colors to objects.

Chapter 3

LiftPool: Bidirectional ConvNet Pooling

3.1 Introduction

Spatial pooling has been a critical ConvNet operation since its inception [59, 124, 117, 82, 24]. It is crucial that a pooling layer maintains the most important activations for the network's discriminability [170, 11]. Several simple operations, such as average pooling or max pooling, have been explored for aggregating features in a local area. [188] employ a convolutional layer with an increased stride to replace a pooling layer, which is equivalent to downsampling. While effective and efficient, simply using the average or maximum activation may ignore local structures. In addition, as these functions are not invertible, upsampling the downscaled feature maps can not recover the lost information. Different from existing pooling operations, we propose in this chapter a bidirectional pooling called LiftPool, including LiftDownPool which preserves details when downsizing the feature maps, and LiftUpPool for generating finer upsampled feature maps.

LiftPool is inspired by the classical Lifting Scheme [195] from signal processing, which is commonly used for information compression [158], reconstruction [44], and denoising [220]. The perfect invertibility of the Lifting Scheme stimulates some works on invertible networks [43, 100, 5, 99] . The Lifting Scheme decomposes an input signal into various sub-bands with downscaled size and this process is perfectly invertible. Applying the idea of Lifting Scheme, LiftDownPool factorizes an input feature map into several downsized spatial sub-bands with different correlation structures. As shown in Figure 3.1, for an image feature map, the LL sub-band is an approximation removing several details. The LH, HL and HH represent details along horizontal, vertical and diagonal directions. LiftDownPool respects preserving any sub-band(s) as the pooled result. Moreover, due to the invertibility of the pooling function, LiftUpPool is introduced for upsampling feature maps. Upsampling a feature map is more challenging as seen for the MaxUpPool [7], which generates an output with many 'holes' (shown in Figure 3.1). LiftUpPool utilizes the recorded details to recover a refined output by performing LiftDownPool backwards.

We analyze the proposed LiftPool from several viewpoints. LiftDownPool allows a flexible choice for any sub-band(s) as the pooled result. It outperforms baselines on

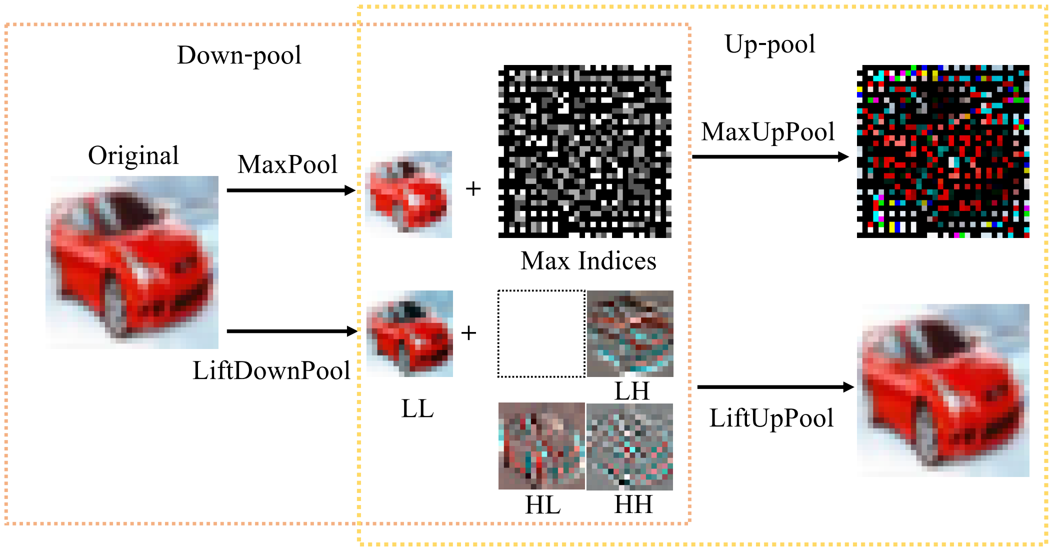


FIGURE 3.1: **Illustration of the proposed LiftDownPool and LiftUpPool** *vs.* MaxPool and MaxUpPool on an image from CIFAR-100. Where MaxPool takes the maximum activations from the input, LiftDownPool decomposes the input into four sub-bands: LL, LH, HL and HH. LL contains low-pass coefficients. It better reduces aliasing compared to MaxPool. LH, HL and HH represent details along horizontal, vertical and diagonal directions. For simplicity, we just upsample the down-pooled results for illustrating the up-pooling. MaxUpPool generates a sparse map with lost details. LiftUpPool produces a refined output from the recorded details by performing LiftDownPool backwards.

image classification with various ConvNet backbones. LiftDownPool also presents better stability to corruptions and perturbations of inputs. By performing LiftDownPool backwards, LiftUpPool generates a refined upsampling feature map for semantic segmentation.

3.2 Methods

The down-pooling operator is formulated as minimizing the information loss caused by downsizing feature maps, as in image downscaling by [170, 114]. The Lifting Scheme [195] naturally matches the problem. The Lifting Scheme was originally designed to exploit the correlated structures present in signals to build a downsized approximation and several detail sub-bands in the spatial domain [38]. The inverse transform is realizable and always provides a perfect reconstruction of the input. LiftPool is derived from the Lifting Scheme for bidirectional pooling layers.

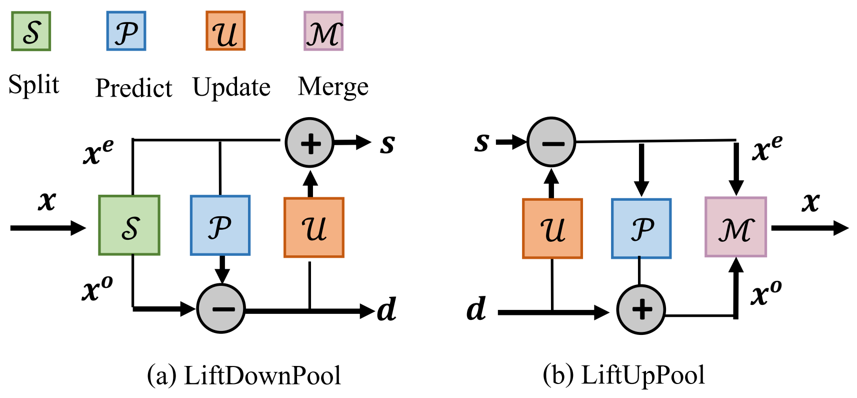


FIGURE 3.2: **LiftDownPool and LiftUpPool implementations.** (a) LiftDownPool-1D. $\boldsymbol{x}$ is split into $\boldsymbol{x}^e$ and $\boldsymbol{x}^o$. The Predictor and Updater generate details $\boldsymbol{d}$ and an approximation $\boldsymbol{s}$. (b) LiftUpPool-1D. By running LiftDownPool backwards, $\boldsymbol{x}^e$ and $\boldsymbol{x}^o$ are generated from $\boldsymbol{s}$ and $\boldsymbol{d}$, and then merged into $\boldsymbol{x}$.

3.2.1 LiftDownPool

Taking a one-dimension (1D) signal as an example, LiftDownPool decomposes a given signal $\boldsymbol{x}=[x_1, x_2, x_3, ..., x_n], x_n \in \mathbb{R}$ into a *downscaled* approximation signal $\boldsymbol{s}$ and a difference signal $\boldsymbol{d}$ by,

$$\boldsymbol{s}, \boldsymbol{d} = F(\boldsymbol{x}). \tag{3.1}$$

where $F(\cdot)=f_{\text{update}} \circ f_{\text{predict}} \circ f_{\text{split}}(\cdot)$, consisting of three functions: split (downsample), predict and update. Here $\circ$ indicates the function composition operator. The LiftDownPool-1D is illustrated in Figure 3.2(a). Specifically,

Split $f_{\text{split}} : \boldsymbol{x} \mapsto (\boldsymbol{x}^e, \boldsymbol{x}^o)$. The given signal $\boldsymbol{x}$ is split into two disjoint sets $\boldsymbol{x}^e=[x_2, x_4, ..., x_{2k}]$ with even indices and $\boldsymbol{x}^o=[x_1, x_3, ..., x_{2k+1}]$ with odd indices. The two sets are typically closely correlated.

Predict $f_{\text{predict}} : (\boldsymbol{x}^e, \boldsymbol{x}^o) \mapsto \boldsymbol{d}$. Given one set *e.g.* $\boldsymbol{x}^e$, another set $\boldsymbol{x}^o$ is able to be predicted by a predictor $\mathcal{P}(\cdot)$. The predictor is not required to be precise, so the difference with the high-pass coefficients $\boldsymbol{d}$ is defined as:

$$\boldsymbol{d} = \boldsymbol{x}^o - \mathcal{P}(\boldsymbol{x}^e). \tag{3.2}$$

Update $f_{\text{update}} : (\boldsymbol{x}^e, \boldsymbol{d}) \mapsto \boldsymbol{s}$. Taking $\boldsymbol{x}^e$ as an approximation of $\boldsymbol{x}$ causes a serious aliasing because $\boldsymbol{x}^e$ is simply downsampled from $\boldsymbol{x}$. Particularly, the running average of $\boldsymbol{x}^e$ is not the same as that of $\boldsymbol{x}$. To correct it, a smoothed version $\boldsymbol{s}$ is generated by adding $\mathcal{U}(\boldsymbol{d})$ to $\boldsymbol{x}^e$:

$$\boldsymbol{s} = \boldsymbol{x}^e + \mathcal{U}(\boldsymbol{d}). \tag{3.3}$$

The update procedure is equivalent to applying a low-pass filter to $\boldsymbol{x}$. Thus, $\boldsymbol{s}$ with low-pass coefficients is taken as an approximation of the original signal.

The classific Lifting Scheme method applies pre-defined low-pass filters and high-pass filters to decompose an image into four sub-bands. However, pre-designing filters in $\mathcal{P}(\cdot)$ and $\mathcal{U}(\cdot)$ is difficult [246]. Earlier, [246] proposed to optimize these filters by a back-propagation network. All functions in LiftDownPool are differentiable. $\mathcal{P}(\cdot)$ and $\mathcal{U}(\cdot)$ are able to be simply implemented by convolution operators followed by non-linear activation functions [72]. Specifically, we design $\mathcal{P}(\cdot)$ and $\mathcal{U}(\cdot)$ as:

$$\mathcal{P}(\cdot) = \text{Tanh()} \circ \text{Conv(k=1,s=1,g=}G_2) \circ \text{ReLU()} \circ \text{Conv(k=}K\text{,s=1,g=}G_1), \tag{3.4}$$

$$\mathcal{U}(\cdot) = \text{Tanh()} \circ \text{Conv(k=1,s=1,g=}G_2) \circ \text{ReLU()} \circ \text{Conv(k=}K\text{,s=1,g=}G_1). \tag{3.5}$$

Here K is the kernel size and G_1 and G_2 are the number of groups. We prefer to learn the filters in $\mathcal{P}(\cdot)$ and $\mathcal{U}(\cdot)$ with deep neural networks in an end-to-end fashion. To that end, two constraints need to be added to the final loss function. Recall, $\boldsymbol{s}$ is the downsized approximation of $\boldsymbol{x}$. As $\boldsymbol{s}$ is updated from $\boldsymbol{x}^e$ according to Eq 3.3, $\boldsymbol{s}$ is essentially close to $\boldsymbol{x}^e$. Thus, $\boldsymbol{s}$ is naturally required to be close to $\boldsymbol{x}^o$ as well. Therefore, one constraint term c_u is for minimising the L2-norm distance between $\boldsymbol{s}$ and $\boldsymbol{x}^o$. With Eq 3.3,

$$\begin{aligned} c_u &= \|\boldsymbol{s} - \boldsymbol{x}^o\|_2 \\ &= \|\mathcal{U}(\boldsymbol{d}) + \boldsymbol{x}^e - \boldsymbol{x}^o\|_2. \end{aligned} \tag{3.6}$$

The other constraint term c_p is for minimising the detail $\boldsymbol{d}$, with Eq 3.2,

$$c_p = \|\boldsymbol{x}^o - \mathcal{P}(\boldsymbol{x}^e)\|_2. \tag{3.7}$$

The total loss is:

$$\mathcal{L} = \mathcal{L}_{\text{task}} + \lambda_u c_u + \lambda_p c_p, \tag{3.8}$$

where $\mathcal{L}_{\text{task}}$ is the loss for a specific task, like a classification or semantic segmentation loss. We set λ_u=0.01 and λ_p=0.1. Our experiments show the two terms bring good regularization to the model.

LiftDownPool-2D is easily decomposed into several 1D LiftDownPool operators. Following the standard Lifting Scheme, we first perform a LiftDownPool-1D along the horizontal direction to obtain an approximation part $\boldsymbol{s}$ (low frequency in the horizontal direction) and a difference part $\boldsymbol{d}$ (high frequency in the horizontal direction). Then, for each of the two parts, we apply the same LiftDownPool-1D along the vertical direction. By doing so, $\boldsymbol{s}$ is further decomposed into LL (low frequency in vertical and horizontal directions) and LH (low frequency in the vertical direction and high frequency in the horizontal direction). $\boldsymbol{d}$ is further decomposed into HL (high frequency in the vertical direction and low frequency in the horizontal direction) and HH (high frequency in vertical and horizontal directions). We can flexibly choose sub-band(s) for down-pooling and keep the other sub-band(s) for reversing the operation.

Naturally, LiftDownPool-1D can be generalized further for any n-dimensional signal. In Figure 3.3, we show several feature maps from the first LiftDownPool layer based on VGG13. LL has smoothed features with less details. LH, HL and HH capture the details along horizontal, vertical and diagonal directions.

Discussion MaxPool is usually formulated as first performing Max and then downsampling: $\text{MaxPool}_{k,s}$=$\text{downsample}_s \circ \text{Max}_k$ [235]. By contrast, LiftDownPool is: $\text{LiftDownPool}_{k,s}$=$\text{update}_k \circ \text{predict}_k \circ \text{downsample}_s$. First downsampling and then performing two lifting steps (prediction and updating) helps anti-aliasing. A simple analysis is provided in the Appendix. As shown in Figure 3.1, LiftDownPool keeps more structured information and better reduces aliasing then MaxPool.

3.2.2 LiftUpPool

LiftPool inherits the invertibility of the Lifting Scheme. Taking the 1D signal as an example, LiftUpPool generates an upsampled signal $\boldsymbol{x}$ from $\boldsymbol{s}, \boldsymbol{d}$ by:

$$\boldsymbol{x} = \mathcal{G}(\boldsymbol{s}, \boldsymbol{d}). \tag{3.9}$$

where $\mathcal{G}(\cdot)$=$f_{\text{merge}} \circ f_{\text{predict}} \circ f_{\text{update}}(\cdot)$, including the functions: update, predict and merge. Specifically, $\boldsymbol{s}, \boldsymbol{d} \mapsto \boldsymbol{x}^e, \boldsymbol{d} \mapsto \boldsymbol{x}^e, \boldsymbol{x}^o \mapsto \boldsymbol{x}$ are realized by:

$$\boldsymbol{x}^e = \boldsymbol{s} - \mathcal{U}(\boldsymbol{d}), \tag{3.10}$$

$$\boldsymbol{x}^o = \boldsymbol{d} + \mathcal{P}(\boldsymbol{x}^e), \tag{3.11}$$

$$\boldsymbol{x} = f_{\text{merge}}(\boldsymbol{x}^e, \boldsymbol{x}^o). \tag{3.12}$$

We simply get the even part $\boldsymbol{x}^e$ and odd part $\boldsymbol{x}^o$ from $\boldsymbol{s}$ and $\boldsymbol{d}$, and then merge $\boldsymbol{x}^e$ and $\boldsymbol{x}^o$ into $\boldsymbol{x}$. In this way, we generate upsampled feature maps with rich information.

Discussion Up-pooling has been used in image-to-image translation tasks such as semantic segmentation [23], super-resolution [179], and image colorization [243]. It is generally used in encoder-decoder networks such as SegNet [7] and UNet [167]. However, most existing pooling functions are not invertible. Taking MaxPool as the baseline, it is required to record the maximum indices during max pooling. For simplicity, we use the down-pooled results as inputs to the up-pooling in Figure 3.1. When performing MaxUpPool, the values of the input feature maps are directly filled on the corresponding maximum indices of the output and other indices will be given zeros. By doing so, the output looks sparse and loses much of the structured information, which is harmful for generating good-resolution outputs. LiftUpPool performing an inverse transformation of LiftDownPool, is able to produce finer outputs by using the multiphase sub-bands.

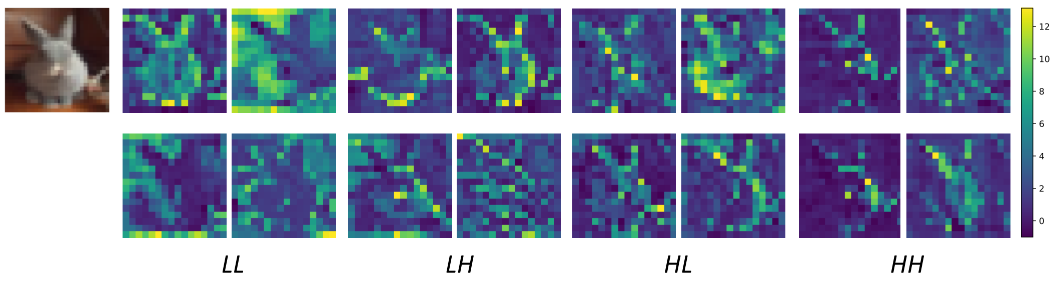


FIGURE 3.3: **LiftDownPool visualization.** Selected feature maps of an image in CIFAR-100, from the first LiftDownPool layer in VGG13. LL represents smoothed feature maps with less details. LH, HL and HH represent detailed features along horizontal, vertical and diagonal directions. Each sub-bands contains different correlation structures.

3.3 Related Work

Taking the average over a feature map region was the pooling mechanism of choice in the Neocognitron [59, 58] and LeNet [124]. Average pooling is equivalent to blurred-downsampling. Max pooling later proved even more effective [178] and became popular in deep ConvNets. Yet, averaging activations or picking the maximum activations causes loss of details. [232] and [233] introduced a stochastic process to pooling and downsampling, respectively, for a better regularization. [125] mixed AveragePool and MaxPool by a gated mask to adapt to complex and variable input patterns. [170] introduced detail-preserving pooling (DPP) for maintaining structured details. By contrast, [235] proposed a BlurPool by applying a low-pass filter, which removes details. Interestingly, both methods improved image classification, indicating that (empirically) determining the best pooling strategy is beneficial [170]. [218] introduced the wavelet transform into pooling for reducing jagged edges and other artifacts. [164] suggested pooling in the frequency domain, which enabled flexibility in the choice of the pooling output dimensionality. Pooling based on a probabilistic model was proposed in [114] and [115]. [114] first used a Gaussian distribution to model the local activations and then aggregates the activations into the two statistics of mean and standard deviation. [115] estimated parameters from global statistics in the input feature map, to flexibly represent various pooling types. Our proposed LiftDownPool decomposes the input feature map into a downsized approximation and several details. It is flexible to choose any sub-band(s) as pooled result.

While existing pooling functions are not invertible, our proposed LiftPool is able to perform both down-pooling and up-pooling. Previously, MaxUpPool [7] was introduced for semantic segmentation. As the max pooling function is not invertible, the lost details can not be recovered during up-pooling. Hence, the output suffers from aliasing. Although adding a BlurPool to MaxUpPool may help to reduce the aliasing [235], several details are still lost. LiftUpPool, performing the LiftDownPool functions backwards, is capable of producing a refined high-resolution output with the help of the details sub-bands.

Earlier, [246] introduce back-propagation for the Lifting Scheme to perform nonlinear wavelet decomposition. They propose an update-first Lifting Scheme and use back-propagation to replace the Updater and Predictor in the Lifting Scheme. In this way, they realize a back-propagation neural network in lifting steps for signal processing. There is no pooling layer used. We develop down-pooling and up-pooling layers by leveraging the idea of the Lifting Scheme for image processing. We utilize convolution layers and ReLU layers to implement the Updater and Predictor, which are optimized end-to-end with the deep neural network. Our pooling layers are easily plugged into various backbones. Recently, [165] introduce the Lifting Scheme for multiresolution analysis in a network. Specifically, they develop an adaptive wavelet network by stacking several convolution layers and Lifting Scheme layers. They focus on an interpretable network by integrating multiresolution analysis, rather than pooling. This chapter aims at developing a pooling layer by utilizing the lifting steps. We develop a down-pooling that constructs various sub-bands with different information, and an up-pooling which generates refined upsampled feature maps.

3.4 Experiments

3.4.1 ConvNet Testbeds

Image Classification We first verify the proposed LiftDownPool for image classification on ***CIFAR-100*** [116] with 32×32 low-resolution images. CIFAR-100 has 100 classes with 600 images each. There are 500 training images and 100 testing images per class. A VGG13 [182] network is trained on this dataset. For experiments conducted on CIFAR-100, we repeat each experiment three times with different initial random seeds during training and report the averaged error rate with the standard deviation. We also report results on ***ImageNet*** [40] with 1.2M training and 5000 validation images for 1000 classes. We plug the LiftDownPool into several popular ConvNet backbones to verify its generalizability for image classification. We replace the local pooling layers by LiftDownPool in all the networks. Error rate is utilized as the evaluation metric. All training settings are provided in the Appendix.

Semantic Segmentation We also test the LiftDownPool and LiftUpPool for semantic segmentation on ***PASCAL-VOC12*** [50], which contains 20 foreground object classes and one background class. An augmented version with 10582 training images and 1449 validation images is used. We consider SegNet [7] with VGG13 and DeeplabV3Plus [24] with ResNet50 as ConvNets for semantic segmentation. The performance is measured in terms of pixel mean-intersection-over-union (*mIoU*) across the 21 classes. Code is available at https://github.com/jiaozizhao/LiftPool/.

	Top-1
LL	25.64 $\pm$ 0.04
LH	25.71 $\pm$ 0.04
HL	24.88 $\pm$ 0.05
HH	25.18 $\pm$ 0.08
LL+LH+HL+HH (w/o c_u and c_p)	26.43 $\pm$ 0.07
LL+LH+HL+HH (w/ c_u and c_p)	**24.35** $\pm$ 0.11

TABLE 3.1: **Flexibility.** Top-1 image classification error rate with varying sub-bands on CIFAR-100. Mixing low-pass and high-pass obtains the best result. Adding c_u and c_p helps improve the result.

Kernel	*Top-1*
2	25.53 $\pm$ 0.13
3	25.06 $\pm$ 0.22
4	24.89 $\pm$ 0.07
5	**24.35** $\pm$ 0.11
7	24.40 $\pm$ 0.08

TABLE 3.2: **Effectiveness.** Top-1 image classification error rate with varying kernel size on CIFAR-100. Kernel size 5 achieves better result.

	Top-1
Skip	27.09 $\pm$ 0.11
MaxPool	25.71 $\pm$ 0.13
AveragePool	25.87 $\pm$ 0.03
LiftDownPool	**24.35** $\pm$ 0.11

TABLE 3.3: **Effectiveness.** Top-1 image classification error rate with various pooling methods on CIFAR-100. LiftDownPool outperforms baselines.

3.4.2 Ablation Study

Flexibility We first test VGG13 on CIFAR-100. Different from previous pooling methods, LiftDownPool generates four sub-bands, each of which contains a different type of information. LiftDownPool allows to flexibly choose which sub-band(s) to keep as the final pooled results. In Table 3.1, we show the *Top-1* error rate for the classification based on different sub-bands. Interestingly, it is observed that vertical details contribute more for image classification. Low-pass coefficients and high-pass coefficients along horizontal direction get similar error rate. Whether the two spatial dimensions should be treated equally we leave for our future work. To realize a less lossy pooling, we combine all the sub-bands by summing them up with almost no additional compute cost. Such a pooling significantly improves the results. In addition, the constrains c_u and c_p help to decrease the error rate. Moreover, seen from Table 3.1 and Table 3.3, we further conclude LiftDownPool outperforms other baselines even based on any single sub-band. We believe the learned LiftDownPool provides an effective regularization to the model.

	ResNet18		ResNet50		MobileNet-V2	
	Top-1	*Top-5*	*Top-1*	*Top-5*	*Top-1*	*Top-5*
Skip [188]	30.22	10.23	24.31	7.34	28.66	9.70
MaxPool	28.60	9.77	24.26	7.22	28.65	9.82
AveragePool	28.03	9.55	24.40	7.35	28.32	9.72
S^3Pool [233]	33.91	13.09	27.98	9.34	40.56	17.91
WaveletPool [218]	30.33	10.82	24.43	7.36	29.27	10.26
BlurPool* [235]	29.88	10.58	24.60	7.73	30.58	11.26
DPP* [170]	29.12	10.21	24.62	7.49	29.85	10.53
SpectralPool [164]	28.69	9.87	24.81	7.57	33.38	12.56
GatedPool [125]	27.78	9.44	23.79	7.06	28.94	9.90
MixedPool [125]	27.76	9.50	24.08	7.32	29.00	9.97
GFGP* [115]	26.88	8.66	22.76	6.34	28.42	9.59
GaussPool* [114]	26.58	8.86	22.95	6.30	27.13	8.92
LiftDownPool	**25.80**	**8.14**	**22.36**	**6.11**	**26.09**	**8.22**

TABLE 3.4: **Generalizability** of LiftDownPool on ImageNet. LiftDownPool outperforms alternative pooling methods, no matter what ConvNet backbone is used. $\star$ means the numbers are based on running the code provided by authors. Others are based on our re-implementation.

Effectiveness Table 3.2 ablates the performance when varying kernel sizes for the filters in $\mathcal{P}(\cdot)$ and $\mathcal{U}(\cdot)$. A larger kernel size, covering more local information, performs slightly better. When kernel size equals 7, it brings more computations but no performance gain. Unless specified otherwise, we use for all experiments from now on a kernel size of 5 and we sum up all the sub-bands. We also compare our LiftDownPool with the commonly-used MaxPool, AveragePool, as well as the convolutional layer with stride 2 [188], which is called Skip by [114]. Seen from Table 3.3, LiftDownPool outperforms other pooling methods on CIFAR-100.

Generalizability We apply LiftDownPool to several backbones including ResNet18, ResNet50 [82] and MobileNet-V2 [176] on ImageNet. In Table 4.3, LiftDownPool has 2% lower *Top-1* error rate than MaxPool and AveragePool. While combining MaxPool and AveragePool in a Gated [125] or Mixed [125] fashion, still has a 1% gap with LiftDownPool. Gauss [114] and GFGP [115] are comparable to LiftDownPool with ResNet50, but not with lighter networks. Compared to Spectral pooling [164] and Wavelet pooling [218], which are applied in the frequency or space-frequency domain, LiftDownPool offers an advantage by learning correlated structures and details in the spatial domain. Compared to DPP [170], which preserves details, and BlurPool [235], smoothing feature maps by a low-pass filter, our LiftDownPool retains all sub-bands which proves to be more powerful for image classification. Stochastic approaches like

	Normalized		Unnormalized	
	ImageNet-C	ImageNet-P	ImageNet-C	ImageNet-P
	mCE	*mPR*	*mCE*	*mPR*
Skip	72.71	61.75	57.05	7.56
MaxPool	73.09	62.64	57.40	7.57
AveragePool	72.09	56.23	56.56	6.90
BlurPool [235]	72.14	56.54	56.58	6.90
DPP [170]	72.12	62.30	56.67	7.62
GatedPool [125]	72.58	58.05	57.00	7.23
GaussPool [114]	69.27	54.83	54.40	6.76
LiftDownPool	**68.45**	**52.91**	**53.80**	**6.55**

TABLE 3.5: **Out-of-distribution robustness** of LiftDownPool on ImageNet-C and ImageNet-P. LiftDownPool is more robust to corruptions and perturbations compared to baselines.

S^3Pool obtain poor results on the large-scale dataset because randomness in pooling hampers network training, as earlier observed by [115]. To conclude, LiftDownPool performs better no matter what backbone is used.

Parameter Efficiency. For all pooling layers in one network, we use the same kernel size in LiftPool. For the trainable parameters, recall $\mathcal{P}$ or $\mathcal{U}$ has a 1D convolution, so each has $C/G_1 \times C \times K + G_2$ parameters. C is the number of the input channels and G_2 equals the number of internal channels. A 2D LiftDownPool shares these parameters three times without extra parameters. We compare our LiftDownPool (with 25.58 M) to two other parameterized pooling methods using ResNet50 on ImageNet: GFGP (31.08 M) and GaussPool (33.85 M). We achieve a lower error rate compared to GFGP and GaussPool with less parameters. Our performance boost is due to the LiftPool scheme, not the added capacity.

3.4.3 Stability Analysis

Out-of-distribution Robustness A good down-pooling method is expected to be stable to perturbations and noise. Following [235], we test the robustness of LiftDownPool to corruptions on ***ImageNet-C*** and stability to perturbations on ***ImageNet-P*** using ResNet50. Both datasets come from [84]. We report the mean Corruption Error (*mCE*) and mean Flip Rate (*mFR*) for the two tasks, with both unnormalized raw values and normalized values by AlexNet's *mCE* and *mFR*, following [84].

From Table 5.8, LiftDownPool effectively reduces raw *mCE* compared to the baselines. We show *CE* for each corruption type for further analysis in Figure A.4 in the Appendix. LiftDownPool enables robustness to both "high-frequency" corruptions,

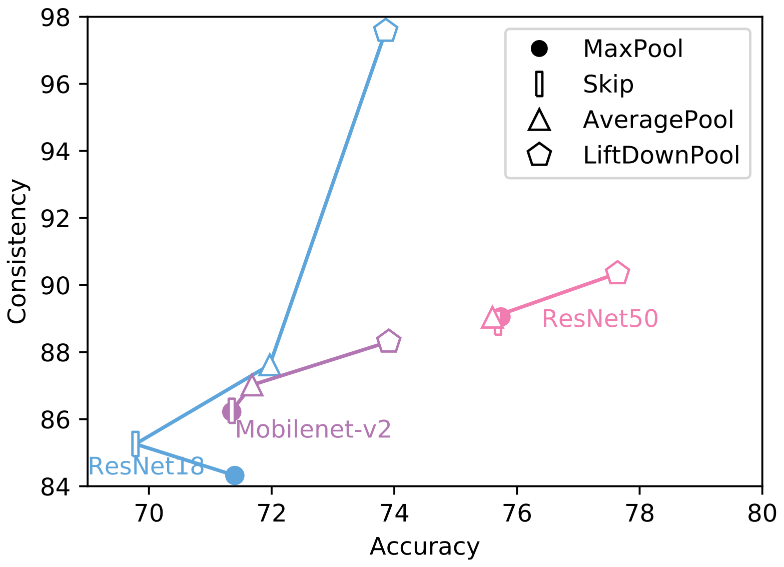


FIGURE 3.4: **Shift Robustness** comparisons between various pooling methods including MaxPool, Skip, AveragePool and the proposed LiftDownPool. LiftDownPool improves classification consistency and meanwhile boosts the accuracy, independent of the backbone used.

such as noise or spatter, and "low-frequency" corruptions, like blur and jpeg compression. We believe LiftDownPool benefits from the mechanism that all sub-bands are used. A similar conclusion is obtained for robustness to perturbations on ImageNet-P from Table 5.8 and Figure A.4 in the Appendix. ImageNet-P contains short video clips of a single image with small perturbations added. Such perturbations are generated by several types of noise, blur, geometric changes, and simulated weather conditions. The metric *FR* measures how often the Top-1 classification changes in consecutive frames. It is designed for testing a model's stability under small deformations. Again, LiftDownPool achieves lower *FR* for most perturbations.

Shift Robustness We then test the shift-invariance of our model. Following [235], we use classification consistency to measure the shift-invariance. It represents how often the network outputs the same classification, given the same image with two different shifts. We test the models with varying backbones trained on ImageNet. In Figure 3.4, LiftDownPool boosts classification accuracy as well as consistency no matter which backbone is used. Besides, we have other interesting findings. The deeper ResNet50 network has more stable shift-invariance. Various pooling methods including MaxPool, Skip, AveragePool, do not make significant difference on consistency. However, a lighter ResNet18 network is influenced much by the pooling method. LiftDownPool brings more than 10% improvement on consistency using ResNet18. We leave for future work how the depth of the network affects the shift-invariance of the network itself.

	mIoU
MaxUpPool	62.7
MaxUpPool + BlurPool	64.0
LiftUpPool	**68.9**

TABLE 3.6: **LiftUpPool for Semantic Segmentation** on PASCAL-VOC12 based on SegNet with varying up-pooling methods.

	mIoU
Skip	76.1
MaxPool	76.2
AveragePool	76.4
Gauss [114]	77.4
LiftDownPool	**78.7**

TABLE 3.7: **Semantic Segmentation with DeepLabV3Plus** on PASCAL-VOC12 with various pooling methods. LiftDownPool performs best.

3.4.4 Results for Semantic Segmentation

LiftUpPool for Semantic Segmentation LiftDownPool functions are invertible as described in Eq 3.10 and Eq 3.11. It naturally benefits a corresponding up-pooling operation, which is popularly used in Encoder-Decoder networks for image-to-image translation tasks. Usually, the Encoder downsizes feature maps layer by layer to generate a high-level embedding for understanding the image. Then the Decoder needs to translate the embedding with a tiny spatial size to a required map with the same spatial size as the original input image. Interpreting details is pivotal for producing high-resolution outputs. We replace all down-pooling and up-pooling layers with LiftDownPool and LiftUpPool in SegNet for semantic segmentation on PASCAL-VOC12. For LiftDownPool we only keep the LL sub-band. For LiftUpPool, the detail-preserving sub-bands LH, HL and HH are used for generating upsampled feature maps. MaxUpPool is taken as the baseline. We also test MaxUpPool followed by a BlurPool [235], which is expected to help anti-aliasing. Table 3.6 reveals LiftUpPool improves over the baselines with a considerable margin. As illustrated in Figure 3.1, MaxUpPool is unable to compensate for the lost details. Although BlurPool helps smoothing local areas, it can only provide a small improvement. As LiftUpPool is capable of refining the feature map by fusing it with details, it is beneficial for per-pixel prediction tasks like semantic segmentation. We show several examples for semantic segmentation in Figure 3.5. LiftUpPool is more precise on details and edges. We also show the feature maps per predicted class in the Appendix.

Semantic Segmentation with DeepLabV3Plus As discussed, LiftDownPool helps to lift ConvNets on accuracy and stability for image classification. ImageNet-trained ConvNets often serve as the backbones for downstream tuning. It is expected to transfer the nature of LiftDownPool to other tasks. We still consider semantic segmentation

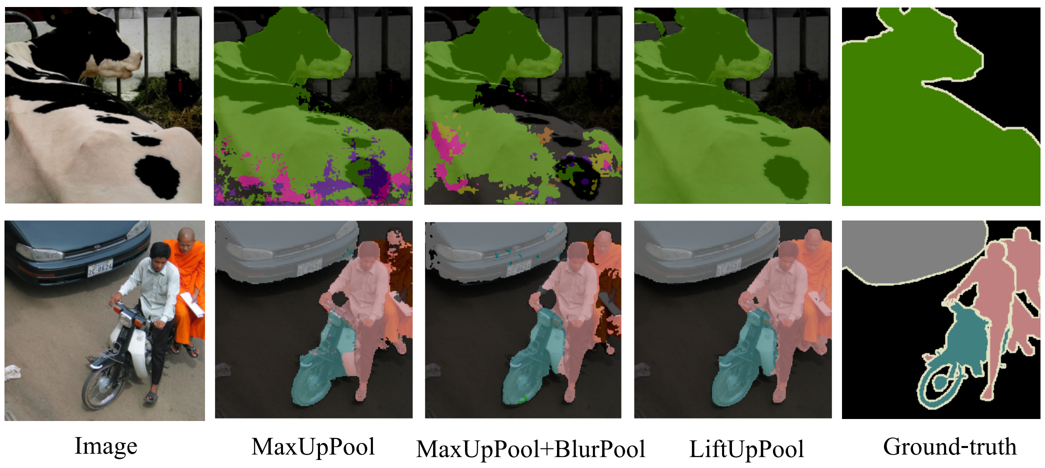


FIGURE 3.5: **LiftUpPool for Semantic Segmentation.** Visualization of semantic segmentation maps on PASCAL-VOC12 based on SegNet with varying up-pooling methods. LiftUpPool presents more completed, precise segmentation maps with smooth edges.

as our example. We leverage the state-of-the-art DeeplabV3Plus-ResNet50 [24]. The input image has size 512$\times$512. The output feature map of the encoder is 32$\times$32. The decoder upsamples the feature map to 128$\times$128 and concatenates them with the low-level feature map for the final pixel-level classification. As before, all local pooling layers are replaced by LiftDownPool. We use the pre-trained weights for image classification to initialize the corresponding model. As shown in Table 3.7, LiftDownPool outperforms all the baselines with considerable gaps.

3.5 Conclusion

Applying classical signal processing theory to modern deep neural networks, we propose a novel pooling method: LiftPool. LiftPool is able to perform both down-pooling and up-pooling. LiftDownPool improves both accuracy and robustness for image classification. LiftUpPool, generating refined upsampling feature maps, outperforms MaxUpPool by a considerable margin on semantic segmentation. Future work may focus on applying LiftPool to fine-grained image classification, super-resolution challenges or other tasks with high demands for detail preservation.

Part II

Learning Continuity for Video Recognition

Chapter 4

Dance with Flow: Two-in-One Stream Action Detection

4.1 Introduction

This chapter strives for the spatio-temporal detection of human actions in video, which is a crucial ability for self-driving cars, autonomous care robots, and advanced video search engines. The leading approach for this challenging problem relies on fast detectors at the frame level [154, 184], which are then linked [71, 184, 8] or tracked [216] over time. Kalogeiton *et al.* [106] and Singh *et al.* [183] further showed it is advantageous to stack the features from subsequent frames before predicting action class scores and determining the enclosing tube. Most of the state-of-the-art action detectors exploit a two-stream architecture [181], one for RGB and one for optical-flow, which are individually trained before fusion. However, the double computation and parameter demand of two-stream methods does not lead to double accuracy compared to a single stream. We propose to embed RGB and optical-flow into a single stream for action detection.

We are inspired by progress on feature normalization, especially conditional normalization [96, 48, 65], which has been successfully employed to visual question answering [39], visual reasoning [156], image style transfer [91] and super-resolution [214]. Peretz *et al.* [156] propose a feature-wise linear modulation layer which enables a recurrent neural network over an input question to influence convolutional neural network computation over an image. It demonstrates that features are capable to be modulated via a simple feature-wise affine transformation based on conditioning information. However, as their modulation layer is agnostic to spatial location, it is unsuited for action detection. In [214], Wang *et al.* developed a spatial feature transform layer, which is conditioned on categorical semantic probability maps, to modulate a super-resolution network. Encouraged by these works, we propose a motion condition layer and a motion modulation layer to adjust an RGB-stream for spatio-temporal action detection.

We make the following contributions in this chapter. We propose to embed RGB and optical-flow into a single stream for spatio-temporal action detection. It reduces the computational costs of conventional two-stream detection networks by half while maintaining its high accuracy. We introduce the two-in-one stream with motion

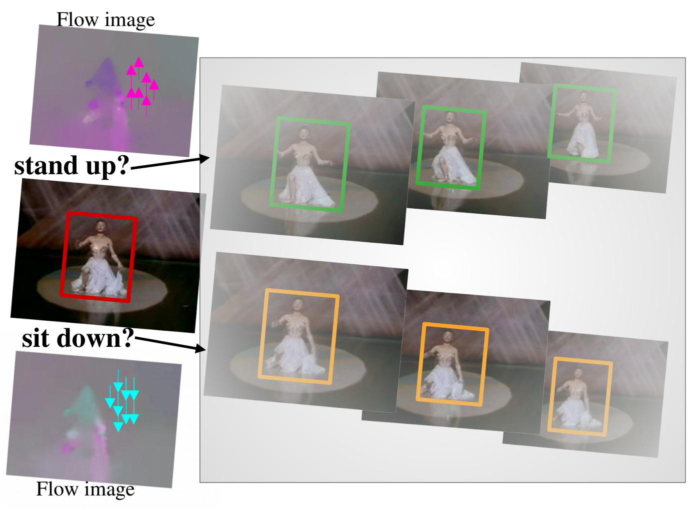


FIGURE 4.1: **Two-in-one stream.** We propose to embed RGB and optical-flow into a single stream for spatio-temporal action detection. Besides efficiency gains, it helps recognizing whether the dancer in the current frame is standing up or sitting down without considering the future. By utilizing information from flow images, the dancer is given a moving direction, up or down, better indicating the action.

condition layer and motion modulation layer, which learns video representations of appearance-stream features conditioned on optical-flow. As shown in Figure 4.1, the motion condition will guide the model to pay more attention on what moves, rather than the static background. The method is easily embedded in existing appearance- or two-stream action detection networks, and trained end-to-end, leading to new state-of-the-art on *UCF101-24*, *UCFSports* and *J-HMDB*.

4.2 Related Work

The spatio-temporal detection of human actions in video has a long tradition in computer vision, *e.g.* [36, 113, 18]. Early success came from detection based on exhaustive cuboid search, efficient feature representations, and SVM-based learning, [229, 199, 198]. This was later extended with more flexible sequences of bounding boxes [121, 200, 228], or spatio-temporal proposals [205, 102], together with engineered appearance and motion features, most notably the dense trajectories [150]. The past few years, architectures integrating detection and deep representation learning have been leading [191, 67, 225, 88, 129, 221, 47], typically combining appearance and flow streams [184, 106, 74, 81]. We follow this tradition.

The two-stream network was first introduced by Simonyan and Zisserman in [181]. Their convolutional architecture included a separate RGB-stream and a flow-stream, which were combined by late fusion, for SVM-based action classification. In [54], Feichtenhofer *et al.* investigated a number of ways to fuse the RGB and flow streams in order to best take advantage of their fused representation for action classification. While we concentrate on action detection in the chapter, we are interested in RGB and flow as well, but rather than combining the two streams in a late fusion, we prefer a single stream.

Gkioxari and Malik [71] introduced a two-stream architecture with R-CNN detectors in action detection. They fused features from the last layer of an RGB- and a flow-stream, and then trained action specific SVM classifiers. A Viterbi algorithm [189] was adopted to link the detection boxes per frame into a tube. Weinzaepfel *et al.* [216] also used a two-stream R-CNN detector but replaced the linking by a tracking-by-detection method. Both methods are not end-to-end trainable and restricted to trimmed videos.

End-to-end two-stream detectors based on faster-RCNN were proposed in [154, 172]. In [154], Peng and Schmid performed region of interest pooling and score fusion to incorporate an RGB-stream and a flow-stream. In [88], Hou *et al.* extended 2D region of interest pooling to 3D tube-of-interest pooling with 3D convolutions, which directly generate tubelets for action detection. Singh *et al.* adopted a two-stream single-shot-multibox detector (SSD)[136] for realizing real time detection in [184]. Singh *et al.* [183] also introduced a transition matrix to generate a set of action proposals on pairs of frames. Kalogeiton *et al.* [106] proposed to exploit temporal continuity by taking as much as six frames as input for their single-shot multibox detector, leading to state-of-the-art results. In this chapter, we take the single-shot multibox detector network as our backbone, using single [184] or multiple [106] frames as input, but rather than separating the streams for RGB and flow we introduce a single two-in-one stream.

Li *et al.* [132] proposed an action detector using an LSTM architecture with motion-based attention. Our two-in-one stream not only takes motion as attention, which helps to locate actions, but also uses motion to modulate RGB features which helps to better classify actions. Moreover, our method is easily embedded in existing appearance- or two-stream action detection and classification networks.

4.3 Two-in-One Network

We define the RGB-stream network $\mathcal{D}_\theta^{rgb}$ trained on single frame for spatio-temporal action detection as:

$$(L^{rgb}, S^{rgb}) = \mathcal{D}_\theta^{rgb}(I^{rgb}) \tag{4.1}$$

where $I^{rgb} \in \mathbb{R}^{H\times W\times 3}$ is a single RGB frame of height H and width W which is the input for the network $\mathcal{D}_\theta^{rgb}$. $L^{rgb} \in \mathbb{R}^{Q\times 4}$ and $S^{rgb} \in \mathbb{R}^{Q\times(P+1)}$ are Q box locations and corresponding box classification scores for P action classes and a background class. θ represents the parameters of the learned network. Similarly, we define a flow-stream network on single frame for spatio-temporal action detection as:

$$(L^{of}, S^{of}) = \mathcal{D}_\theta^{of}(I^{of}) \tag{4.2}$$

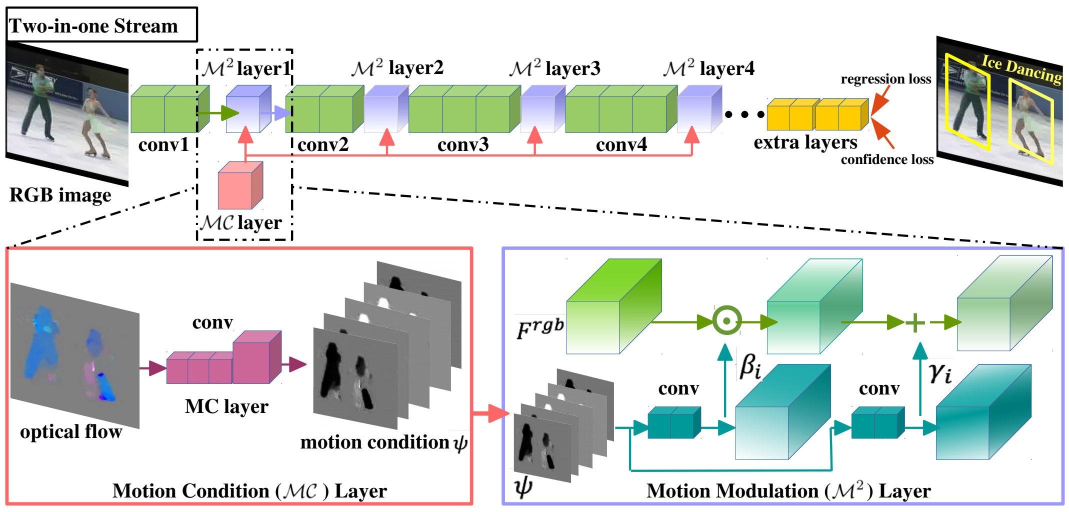


FIGURE 4.2: **Two-in-one network architecture**. The motion condition layer (pink cube) maps flow images to prior condition information. The condition inputs to the motion modulation layer (purple cube) to generate transformation parameters which are used to modulate RGB features (F^{rgb}). The network has half the computation and parameters of a two-stream equivalent, while obtaining better action detection accuracy.

$I^{of} \in \mathbb{R}^{H\times W\times 2}$ is a single optical flow image with x and y components of the velocity respectively in two channels. The two-stream method includes training the two networks $\mathcal{D}_\theta^{rgb}$ and $\mathcal{D}_\theta^{of}$ independently, and fuses the results (L^{rgb}, S^{rgb}) and (L^{of}, S^{of}).

Motion condition layer. In our method, I^{of} is regarded as a motion map with the same resolution as the corresponding RGB image I^{rgb}. We take I^{of} as prior information Ψ when applying an RGB-stream network $\mathcal{D}_\theta^{rgb}$ to estimate where and what actions may occur. Then we formulate our two-in-one network as a condition network:

$$\begin{aligned}(L^{\twoheadrightarrow}, S^{\twoheadrightarrow}) &= \mathcal{D}_\theta^{\twoheadrightarrow}(I^{rgb}|\Psi)\\ &= \mathcal{D}_\theta^{\twoheadrightarrow}(I^{rgb}|\mathcal{MC}(I^{of}))\end{aligned} \tag{4.3}$$

$$\Psi = \mathcal{MC}(I^{of}) = \mathcal{MC}((I^{of_x}, I^{of_y})) \tag{4.4}$$

$\twoheadrightarrow$ means two-in-one stream, $\mathcal{MC}(\cdot)$ is a mapping function to generate simple features from the flow images. So the two-in-one stream $\mathcal{D}_\theta^{\twoheadrightarrow}$ learns a model conditioned on motion information by a motion condition layer.

Motion modulation layer. We introduce a motion modulation ($\mathcal{M}^2$) layer to modify the features learned from RGB images. An $\mathcal{M}^2$ layer is able to influence the appearance network by incorporating motion and weighting the action area. We first learn a pair of affine transformation parameters (β, γ) from the prior flow condition Ψ by a function $\mathcal{F} : \Psi \longmapsto (\beta, \gamma)$. Concretely, the two-in-one network is further expressed as:

$$\begin{aligned}(\beta, \gamma) &= \mathcal{F}(\Psi), \\ (L^{\twoheadrightarrow}, S^{\twoheadrightarrow}) &= \mathcal{D}_{\theta}^{\twoheadrightarrow}(I^{rgb}|\beta, \gamma)\end{aligned} \tag{4.5}$$

In order to modulate the appearance network, we apply a transform function $\mathcal{M}^2(\cdot)$ with the learned transformation parameters (β, γ) to the RGB features F^{rgb}.

$$\mathcal{M}^2(F^{rgb}) = \beta \odot F^{rgb} + \gamma \tag{4.6}$$

$\odot$ is an element-wise multiplication operation. The RGB feature maps F^{rgb} has the same dimensions with parameters β and γ. The flow information represented by (β, γ) influences the appearance network by both feature-wise and spatial-wise manipulations. The complete network with the motion condition layer and the motion modulation layer is shown in Figure 4.2.

Network architecture. Due to sparsity of flow images, we adopt simple convolutional layers to extract low-level motion condition information. 1×1 convolutional layers attempt to keep the spatial pixel-wise motion vectors. The motion condition then inputs to a motion modulation ($\mathcal{M}^2$) layer in which it is separately mapped to a pair of transformation parameters β and γ. Two groups of 1×1 convolutional layers are independently adopted for generating each of the parameters β and γ. The low-level RGB features from the appearance network are adjusted by β and γ. The motion modulation layer is capable to be added to any bottom layer of the appearance network, including conv1, conv2, conv3 and conv4. All of them share the motion condition layer. The whole network is end-to-end trainable.

Feature visualization. In order to intuitively understand the method, we show the generated feature maps from the appearance network before and after modulation by motion condition in Figure 4.3. We randomly select some feature maps from the motion condition layer in the first row. The features are low-level and sparse, which are taken as prior conditions. From the second row to the last row, we show the corresponding scale (β) and shift (γ) maps generated from conditions, RGB features without modulation and features modulated by β and γ. It is interesting to see the difference between the features without and with modulation in Figure 4.3. For example the modified features of the actor areas in feature maps 0 and 43, after modulation, especially for the female ice skater, which is blended into the background on the regular RGB stream. On the 28-th feature map, a feature response is even hard to see on both actors before modulation. Feature maps 10 and 127 show the change in x-direction features and y-direction features. The flow condition pushes the model to focus on moving actors.

Training loss. In order to demonstrate the generalization and flexibility of the proposed method, we embed the motion condition layer and the motion modulation layer in a single-frame appearance stream and a multi-frame appearance stream. The basic loss function is derived from the one for object detection [136, 162]. Defining

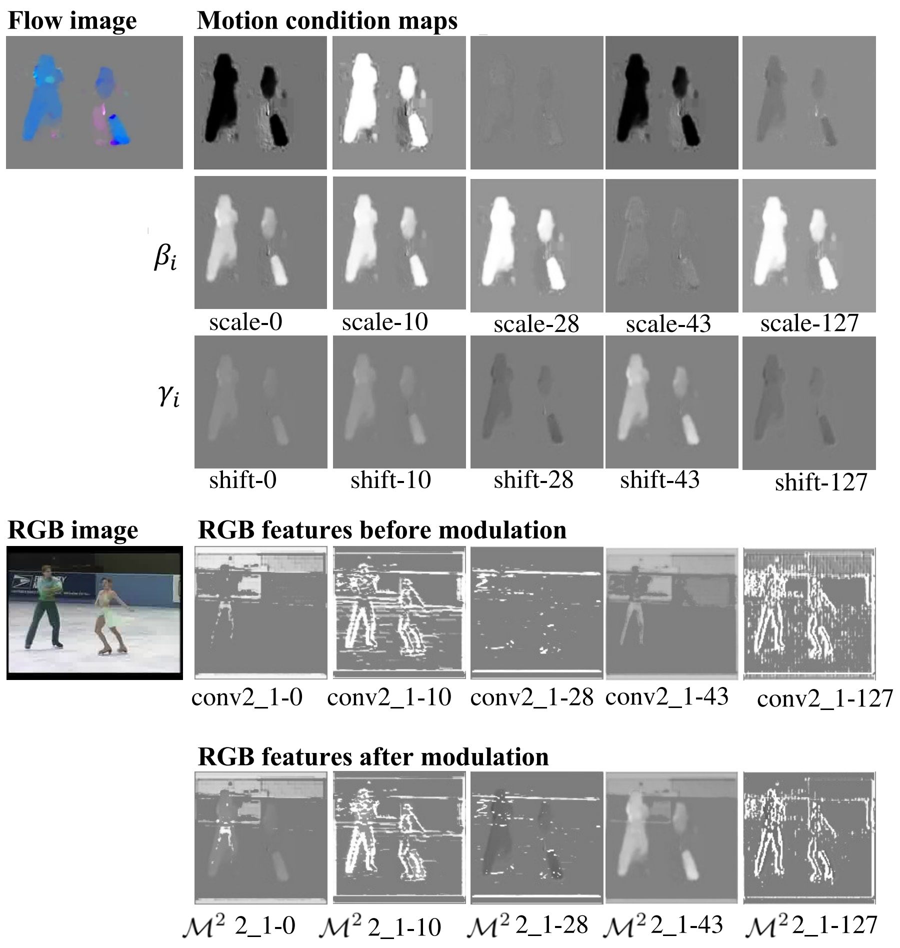


FIGURE 4.3: **Feature maps.** Visualization of the motion condition maps, scale maps, shift maps, RGB features without modulation and features with modulation. The modulated features focus more on moving actors.

${x^p}_{ij} = \{1, 0\}$ as an indicator for matching the i-th default box to the j-th ground truth box of action category p. The overall loss function contains the localization (loc) loss and the confidence ($conf$) loss:

$$\mathcal{L}(x, c, l, g) = \frac{1}{N}(\mathcal{L}_{loc}(x, l, g) + \mathcal{L}_{conf}(x, c)) \tag{4.7}$$

with N representing the number of matched default boxes. c represents multiple classes confidences. l and g are the predicted box and the ground truth box.

The confidence loss applies the softmax loss as below:

$$\begin{aligned}\mathcal{L}_{conf}(x,c) &= -\sum_{i\in Pos}^{N} x_{ij}^{p} log(\hat{c}_i^p) - \sum_{i\in Neg} log(\hat{c}_i^0)\\ \hat{c}_i^p &= \frac{exp(c_i^p)}{\sum_p exp(c_i^p)}\end{aligned} \tag{4.8}$$

The localization loss applies a smooth L1 loss [69] between the predicted box and the ground truth box. The network regresses to offsets for the center (cx, cy) of the default box(d) and for its width (w) and height (h).

$$\begin{aligned}\mathcal{L}_{loc}(x,l,g) &= \sum_{i\in Pos}^{N} \sum_{m\in\{cx,cy,w,h\}} x_{ij}^{k} smooth_{\text{L1}}(l_i^m - \hat{g}_j^m)\\ \hat{g}_j^{cx} &= (g_j^{cx} - d_i^{cx})/d_i^w \qquad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h\\ \hat{g}_j^{w} &= log(\frac{g_j^w}{d_i^w}) \qquad \hat{g}_j^{h} = log(\frac{g_j^h}{d_i^h})\end{aligned} \tag{4.9}$$

For the multi-frame appearance stream, we follow Kalogeiton *et al.* [106] to train the network.

Two-in-one two-stream. Our method emphasizes to utilize RGB and optical flow information in one stream. Furthermore, it is possible to follow the standard practice of two-stream action detection. We train a two-in-one detector conditioned on flow images, and a separate flow detector which only takes as input the flow images. For a single-frame two-in-one two-stream, we use average fusion method to merge the results from each stream following [184]. And for multi-frame two-stream, the late fusion [54] is a better choice [106].

Linking. Once the frame-level detections or tubelet detections are achieved, we link them to build action tubes. We adopt the linking method described in [184] for frame-level detections and the method in [106] for tubelet detections.

Code is available at https://github.com/jiaozizhao/Two-in-One-ActionDetection.

4.4 Experiments

4.4.1 Datasets, Metrics & Implementation

Datasets. We perform experiments on three action detection datasets. *UCF101-24* [187] is a subset of *UCF101*. It contains 24 sport classes in 3207 untrimmed videos. Each video contains a single action category. Multiple action instances with the same class, but different spatial and temporal boundaries may occur. We use the revised annotations for *UCF101-24* from [184]. *UCF-Sports* [166] contains 10 sport classes in 150 trimmed videos. We follow [121] to divide the training and test splits.

	Action Detection			**Action Classification**		
Method	*mAP*	*Efficiency*		*Top1 Accuracy*	*Efficiency*	
	%	sec/frame	# param. (M)	%	sec/frame	# param. (M)
flow-stream	11.60	0.04	26.82	81.65	1.10	58.35
RGB-stream	18.49	0.04	26.82	84.99	1.10	58.35
two-stream	19.79	0.09	53.64	91.14	2.10	116.70
two-in-one stream	20.15	0.04	26.93	86.94	1.15	58.48
two-in-one two stream	**22.02**	0.09	53.75	**92.00**	2.13	116.83

TABLE 4.1: **Two-in-one *vs.* baselines** for action detection on *UCF101-24* and action classification on *UCF101*. Two-in-one with motion modulation works well for both action detection and action classification.

J-HMDB [103] contains 21 action categories in 928 trimmed videos. We report the average results on three splits.

Metrics. Following [186, 216, 172], we utilize video mean Average Precision (mAP) to evaluate action detection accuracy. We calculate an average of per-frame Intersection-over-Union (IoU) across time between tubes. A detection is correct if it's IoU with the ground truth tube is greater than a threshold and its action label is correctly assigned. We compute the average precision for each class and report the average over all classes.

Implementation. We adopt a real-time single shot multibox detector (SSD) network [136] as the backbone. We insert the developed motion layers into two state-of-the-art appearance SSD networks, one based on single frame [184] and the other based on multiple frames [106]. We use VGG-16 pre-trained weights on ImageNet as model initialization. The input size is 300x300 for both of them. We follow [106] to use 6 continuous frames as input to the multi-frame SSD. The initial learning rate is set to 0.001 for the single-frame network and 0.0001 for the multi-frame network on all the three datasets and changed by applying step decay strategy. We trained a flow-stream, an RGB-stream and our two-in-one stream for 13.2, 13.2 and 15.5 hours, respectively.

Alternatively, we considered to use appearance information to modulate flow stream. However, it does not work well. It appears difficult to modulate features from flow images which are sparse, using RGB images which are more dense.

4.4.2 Ablation Study

All the ablation studies are performed on *UCF101-24*. We only report mAP at the most challenging high IoU thresholds 0.5:0.95 (with step 0.05). Initially, in order to maintain the spatial pixel-wise motion vectors, we apply 1x1 convolution kernels to all layers in the motion condition layer and the motion modulation layer. We use layer parameter stride to control the size of β and γ. Then the motion modulation layer is applied to conv1 of SSD. Flow images are generated using the method in [14], which we refer to as BroxFlow.

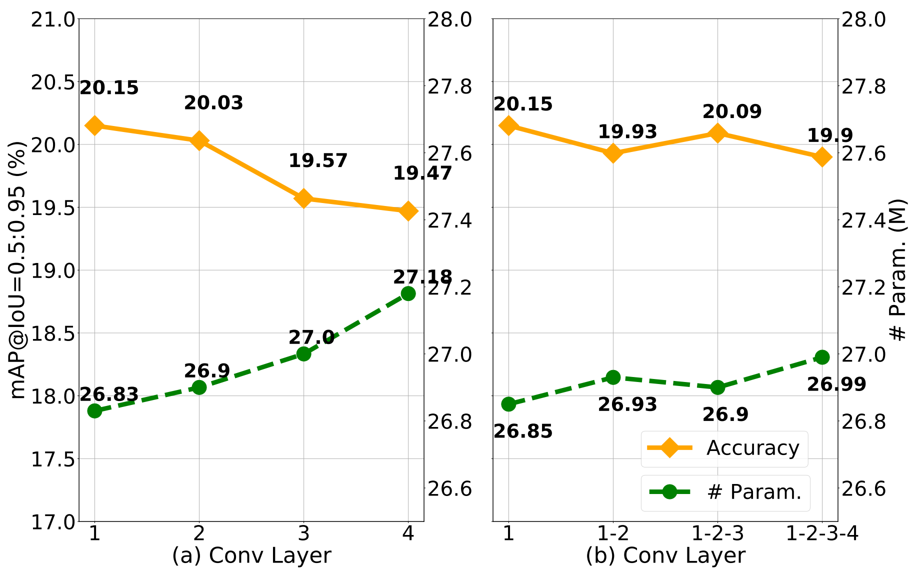


FIGURE 4.4: **Where to add the modulation layer?** Accuracy on *UCF101-24* and # param. with varying: (a) single modulated layer, and (b) multiple modulated layers. A single modulation layer at conv1 gets the best result.

Two-in-one *vs.* baselines. We compare the two-in-one stream to its corresponding RGB-stream, flow-stream and two-stream in Table 4.1. Runtime and # param. are also reported for comparing the efficiency. Our single two-in-one stream exceeds a single RGB-stream by 1.5%. Notably, two-in-one even outperforms the corresponding two-stream with only half the computation cost and # param..

We also consider action classification, on *UCF101*. We follow [211], with ResNet152 as backbone. The *Top 1* accuracy and efficiency shown in Table 4.1 illustrate our strategy also works for action classification and generalizes beyond SSD with VGG16. For training, our two-in-one stream converges at the 100-th epoch, but the RGB- and flow-stream converge at 200-th and 300-th epoch, respectively. Our motion modulation strategy works better for the detection task, which needs localization representations that are translation-variant, compared to the classification task which favors translation invariance.

Where to add the modulation layer? The motion condition layer is leveraged to generate low-level motion features as flow images are more sparse. We add the motion modulation layer to the bottom convolutional layers with low-level RGB features. We conduct two experiments on which layer to add the modulation. We compare the accuracy and # param. after applying a modulation layer to conv1, conv2, conv3 and conv4 in Figure 4.4 (a). Accuracy decreases and # param. increases slightly for deeper layers. Next we add the modulation layers to multiple convolutional layers simultaneously in Figure 4.4 (b). Applying multiple modulation layers does not change the results much. Thus, we prefer to use a single modulation layer. Note that accuracy drops for deeper layers as we use 1x1 convolution kernels to process flow images,

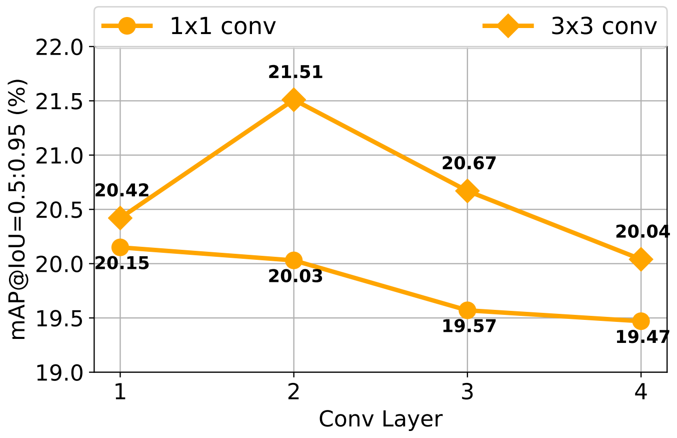


FIGURE 4.5: **How to design the condition layer?** Comparing accuracy on *UCF101-24* when applying 1x1 conv or 3x3 conv to the last layer of the motion condition layer. The 3x3 conv performs better.

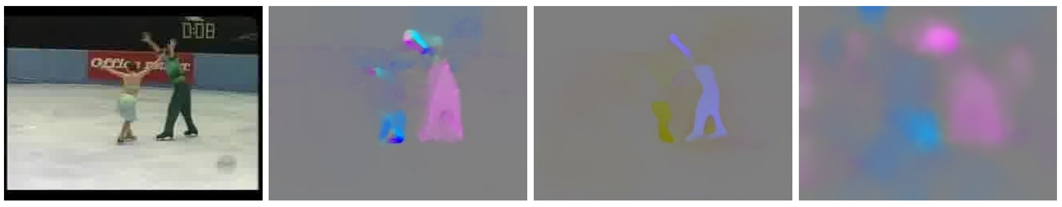


FIGURE 4.6: **What flow?** Examples of flow images generated by different flow methods.

leading to smaller receptive field for deeper layers.

How to design the condition layer? To further improve the method, we consider whether the 1x1 convolution kernel for the motion condition layer is the best choice. Besides keeping the spatial pixel-wise motion, it may need to consider some context of motion to better fit the RGB features. We adopt the 3x3 convolution kernels to the last layer of the condition network. Figure 4.5 demonstrates that considering motion context boosts the accuracy for all layers. As a bigger receptive field is used, the conv2 model achieves the best results, about 1.5% improvement compared to 1x1 convolution kernels. The run time hardly increases for deeper layers, and is still 0.04 sec per frame. The # param. are 26.85, 26.92, 27.01 and 27.19 M respectively of conv1, conv2, conv3 and conv4. Considering the trade-off between the results and parameters, we believe conv2 provides the best accuracy/efficiency trade-off.

What flow? As we leverage flow information as prior conditions, we wonder how the model is influenced by flow images. Here we adopt flow images generated by three different methods (seen in Figure 4.6) and evaluate how our strategies work. We use BroxFlow [14] (accurate flow method), Flownet [46] (deep network method) and a real-time but less accurate optical flow method [118] (RealTimeFlow). From

	BroxFlow	FlowNet	RealTimeFlow
flow-stream	11.60	7.13	3.58
RGB-stream	18.49	18.49	18.49
two-stream	19.79	19.75	18.53
two-in-one stream	**21.51**	19.97	19.16

TABLE 4.2: **What flow?** No matter what flow images are applied on *UCF101-24*, our two-in-one stream outperforms the corresponding flow-, RGB- and two-stream. We obtain the best result with BroxFlow.

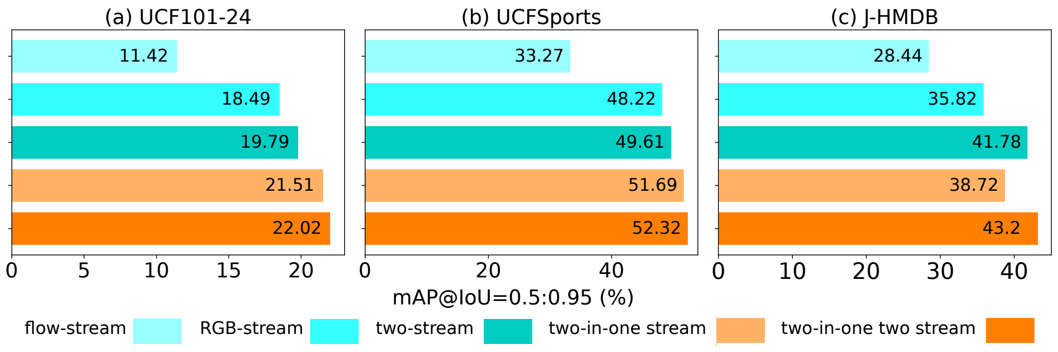


FIGURE 4.7: **Generalization ability.** Accuracy comparison on: (a) *UCF101-24*, (b) *UCFSports*, (c) *J-HMDB*, with different methods. Two-in-one stream even outperforms two-stream on *UCF101-24* and *UCFSports*. Two-in-one stream fused with a flow-stream obtains the best accuracy on all three datasets.

Table 5.4, it is concluded that no matter which kind of flow images are applied, our two-in-one stream outperforms RGB-streams and corresponding two-streams. We also note that the more accurate the flow images, the more improvement the two-in-one stream obtains. Even when using the somewhat noisy RealTimeFlow images, the two-in-one stream still improves the RGB-stream. However, a two-stream based on RealTimeFlow obtains almost the same accuracy as the RGB-stream, which illustrates that two-stream depends on the the quality of flow images. Our two-in-one stream is more robust to the quality of flow images. Moreover, we report the flow computation in seconds/frame for the three kinds of flow methods: BroxFlow (0.098), FlowNet (0.183) and RealTimeFlow (0.014). RealTimeFlow only needs 0.014 seconds to generate one flow image, at the expense of a slightly lower *mAP*.

Generalization ability. To stress the generalization ability of our proposal, we compare the results on three different datasets. Following the conclusions of our ablation so far, we use the BroxFlow image for generating condition and apply a 3x3 kernel to the last layer of the motion condition layer. The motion modulation layer is only leveraged for the conv2 layer of the appearance stream. We report results in Figure 4.7.

Obviously, the proposed two-in-one stream performs better than other one-stream networks. It is noteworthy that our two-in-one stream even outperforms traditional two-stream networks on *UCF101-24* and *UCFSport* by 2% with only half the parameters of a two-stream network. On *J-HMDB*, two-in-one is 3% higher than RGB-stream but 3% lower than two-stream. We look into *J-HMDB* and find that most videos in the dataset have neighbouring repeated frames. For fair comparison, we just download the BroxFlow images used in [184, 106]. However, the provided BroxFlow image between the two repeated RGB frames is not 0, as it should be, but similar to the last flow frame. The issue affects our two-in-one stream due to the fact that we need correct flow image as the condition of the corresponding RGB frame. We expect that two-in-one will present better results on *J-HMDB* after correcting the flow images. As expected, adding a separate flow-stream to our two-in-one stream gives the best accuracy on all datasets.

4.4.3 Qualitative Analysis

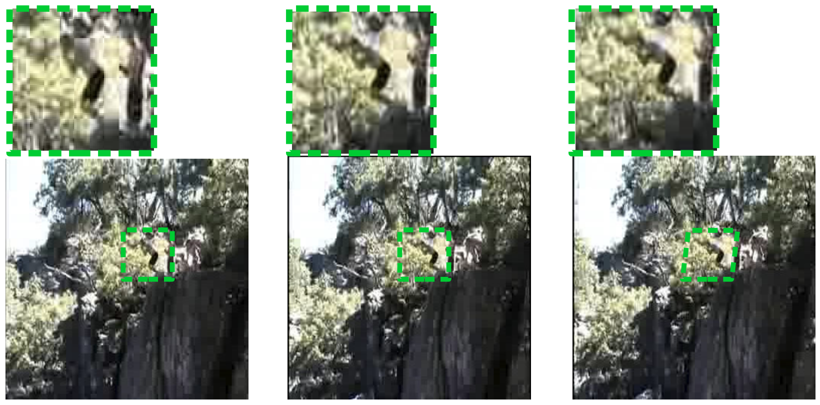
The motion condition layer and the motion modulation layer are beneficial to generate better video representations for spatio-temporal action detection. But how do the layers make a difference to the appearance network? To understand this behavior, we visualize in Figure 4.8, the detection results of an RGB-stream network and a two-in-one network. Also, we visualize the gradient-weighted class activation heatmaps [247] for better understanding how the motion conditions influence the behavior of the appearance network. We choose a challenging case of cliff diving here. The image resolution is low and the actor is quite tiny. The cluttered background obviously increases the difficulty to detect actions. We manually overlay green dashed boxes to indicate the locations of the actor and zoom in to highlight where the action is happening. The second row shows that the RGB-stream fails to detect any actions. From the corresponding heatmaps, it is apparent that the appearance network pays more attention to the background than to the actions. There are only weak responses on the action positions. We manually overlay red dashed boxes to highlight the position of the actor on the heatmaps. From the heatmaps for the two-in-one network in the last row, we clearly see it is capable to balance the activation on actions and background. The responses on action positions are strengthened. As expected, the two-in-one stream performs better than the RGB-stream. It outputs correct detections for cliff diving on all the frames (forth row).

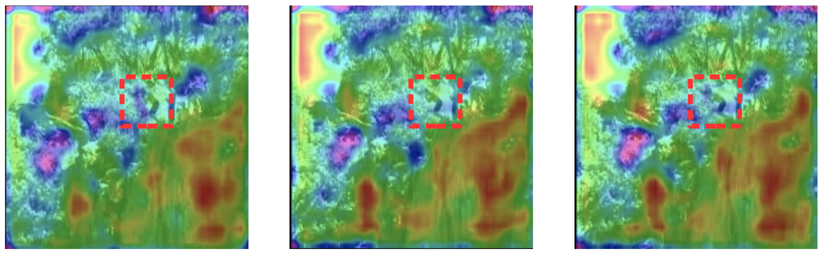
4.4.4 Comparison to the State-of-the-art

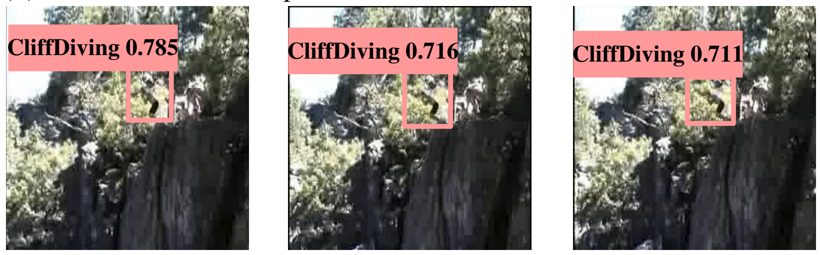
Accuracy. For fair comparisons, we just use the original images as in all the state-of-the-arts, without camera motion removal. We compare the mAP at variable IoU thresholds in Table 4.3. Considering the most challenging high IoU thresholds 0.5:0.95, we observe that for the single-frame setting, our two-in-one stream achieves even better results than existing two-stream methods on *UCF101-24* and *UCFSports*. For instance, two-in-one stream outperforms Singh *et al.* [184] with the same SSD

	UCF101-24				*UCFSports*			*J-HMDB*		
	0.20	0.50	0.75	0.50:0.95	0.50	0.75	0.50:0.95	0.50	0.75	0.50:0.95
Single-frame										
Peng & Schmid [154]	71.80	35.90	1.60	8.80	**94.80**	47.30	51.00	70.60	*48.20*	*42.20*
Saha *et al.* [172]	66.70	35.90	7.90	14.40	–	–	–	*71.50*	43.30	40.00
Behl *et al.* [8]	71.53	40.07	13.91	17.90	–	–	–	–	–	–
Singh *et al.* [184]	73.50	46.30	15.00	20.40	–	–	–	**72.00**	44.50	41.60
Two-in-one	*75.13*	*47.47*	*17.21*	*21.51*	87.46	*57.81*	*51.69*	60.99	47.23	38.72
Two-in-one two stream	**77.49**	**49.54**	**17.62**	**22.02**	*87.81*	**62.67**	**52.32**	70.00	**52.00**	**43.20**
Multi-frame										
Saha *et al.* [171]	63.06	33.06	0.52	10.72	–	–	–	57.31	–	–
Kalogeiton *et al.* [106]	76.50	49.20	19.70	23.40	92.70	78.40	58.80	*73.70*	*52.10*	*44.80*
Singh *et al.* [183]	**79.00**	**50.90**	20.10	*23.90*	–	–	–	–	–	–
Two-in-one	75.48	48.31	*22.12*	*23.90*	*92.74*	*83.64*	*59.60*	57.96	42.78	34.56
Two-in-one two stream	*78.48*	*50.30*	**22.18**	**24.47**	**96.52**	**90.41**	**63.59**	**74.74**	**53.28**	**45.01**

TABLE 4.3: **Accuracy comparison to the state-of-the-art.** Bold means top accuracy and italic means second top accuracy. For the high overlap setting of $mAP@IoU$=0.5:0.95, our two-in-one stream works well in both a single-frame and multiple-frame network for all three datasets. When we add an additional flow-stream to obtain a two-in-one two stream we further improve accuracy.


(a) RGB-stream Results: no detections (confidence scores < 0.5)


(b) RGB-stream Heatmaps: low activation on actor



(c) Two-in-one Results: correct detections (cliff diving scores > 0.5)

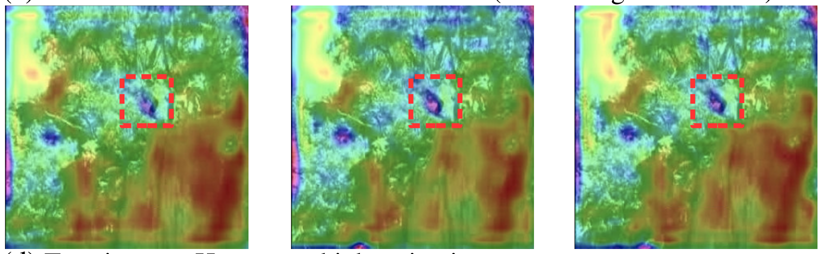

(d) Two-in-one Heatmaps: high activation on actor

FIGURE 4.8: **Visualization of detection and heatmaps** on conv4 layers from RGB-stream network in (a) (b) and two-in-one stream network in (c) (d). We add the green dashed boxes to indicate the action. The two-in-one stream has higher activation on actions, resulting in correct detection.

detector by more than 1% and Peng and Schmid [154] with a Faster-RCNN detector by an absolute 12% on *UCF101-24*. As analyzed previously, two-in-one stream performs modest on *J-HMDB* because of the data issue of the provided BroxFlow images.

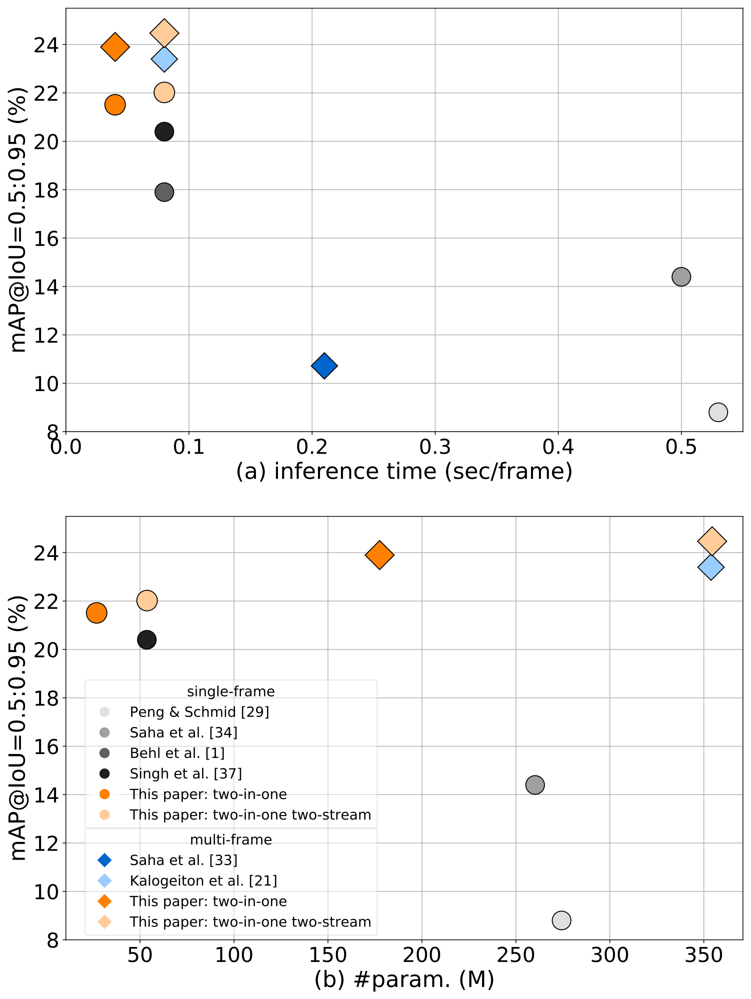


FIGURE 4.9: **Efficiency comparison to the state-of-the-art.** Accuracy *vs.* (a) inference time (second per frame) and (b) # param. (M) on *UCF101-24*. Our two-in-one stream best balances accuracy and efficiency.

When we combine two-in-one into a regular two-stream network by fusing with a flow-stream, it produces good results on all three datasets. Compared to two-in-one stream, it gets about 5% improvement on *J-HMDB*. Moreover, when feeding our two-in-one network variants with multiple frames, as suggested by Kalogeiton *et al.* [106], our two-in-one stream outperforms the two-stream [106] a little on *UCF101-24* and *UCFSports* with only half computation and the number of parameters. Our two-in-one stream fused with a flow stream further boosts the results, outperforming the very recent work of Singh *et al.* [183].

Efficiency. Besides good detection accuracy, our method has the advantage of a reduced inference time and less # param.. Here we compare our methods from the efficiency aspect to the state-of-the-art on *UCF101-24*. We test our models on one NVIDIA GTX 1080 GPU. The trade-off between accuracy and inference time, as well as parameters are visualized in Figure 4.9. Among the single-frame methods, our two-in-one stream has the fastest run time with 0.04s per frame, two times faster than [8] and [184] and much faster than [172] and [154] (about 0.5s per frame). Moreover, the # param. of our two-in-one stream is smallest, about 26.93 M. While our two-in-one accuracy is even better than the two-stream methods by [8, 172, 154, 184]. Combining our two-in-one stream with a standard flow-stream gains an accuracy improvement at the expense of more computation and parameters. Our two-in-one alternative even outperforms [106] a little in accuracy with only half the parameters. The two-in-one two stream further improves the result with almost similar inference time, but slightly more parameters. We conclude that two-in-one stream networks provide a good accuracy/efficiency trade-off.

4.5 Conclusion

We propose an effective and efficient two-in-one stream network for spatio-temporal action detection. It takes flow images as prior motion condition when training an RGB-stream network. The network's motion condition layer and motion modulation layer address two issues in action detection: frame-level RGB images lack motion information and (static) background-context may dominant the learned representation. Our two-in-one stream achieves state-of-the-art accuracy at high IoU thresholds, using only half of the parameters and computation of two-stream alternatives. Besides motion, we believe that other information such as depth-maps or infrared images may help locate the actors, and can be exploited as additional prior conditions for training two-in-one streams.

Acknowledgments Supported by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior/Interior Business Center (DOI/IBC) contract number D17PC00343. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing endorsements, either expressed or implied, of IARPA, DOI/IBC, or the U.S. Government.

Chapter 5

Go with the Flow: Aligned 3D Convolutions for Action Recognition

5.1 Introduction

Models for video tasks such as action classification and detection, need to learn how to represent space and time. The classical two-stream regime [181, 54] achieves good accuracy at the expense of optical flow pre-computation, a duplicated learning effort, and double memory consumption. Instead, the 3D convolution [104, 202] is a compact design for considering the time dimension. However, without considering the object dislocations between successive frames, it gathers the information from the same spatial locations along the time dimension. As shown in Figure 5.1 (a), the existing 3D convolution may not integrate valuable information from the moving parts. In this chapter, we propose aligned 3D convolutions (Figure 5.1 (b-d)) to address the issue by calibrating feature points and collecting motion information for action classification.

We are inspired by the idea of dense trajectories [209] that are local space-time features for action recognition. Specifically, the idea is to densely sample feature points in each frame, which are described by multiple hand-crafted features including HOG [35], HOF [122] and MBH [36], and track them in a video using optical flow. Thanks to the big success of deep convolutions, Wang *et al*. [210] improve the method by extracting deep features instead of hand-crafted features. Recently, Zhao *et al*. [245] propose a 1D trajectory convolution to aggregate the features from 2D convolutions along the trajectories represented by optical flow. All these existing methods require pre-computed dense trajectories and/or optical flow. We propose aligned 3D convolutions, which learn spatial offsets to match feature points between successive frames, only driven by the action classification task. Furthermore, stimulated by merits of the deformable convolution [33] for static images, we develop a generalized point-to-point aligned convolution with enhanced transformation abilities capable to learn finer motion representations from videos.

We make the following contributions in this chapter. We design three variants of aligned 3D convolutions for generating motion representations. The patch-to-patch convolution is able to maintain local consistency. The patch-to-point convolution is lighter and requires less computation. The point-to-point convolution is capable to handle the geometric transformation of moving objects. We rethink how the length

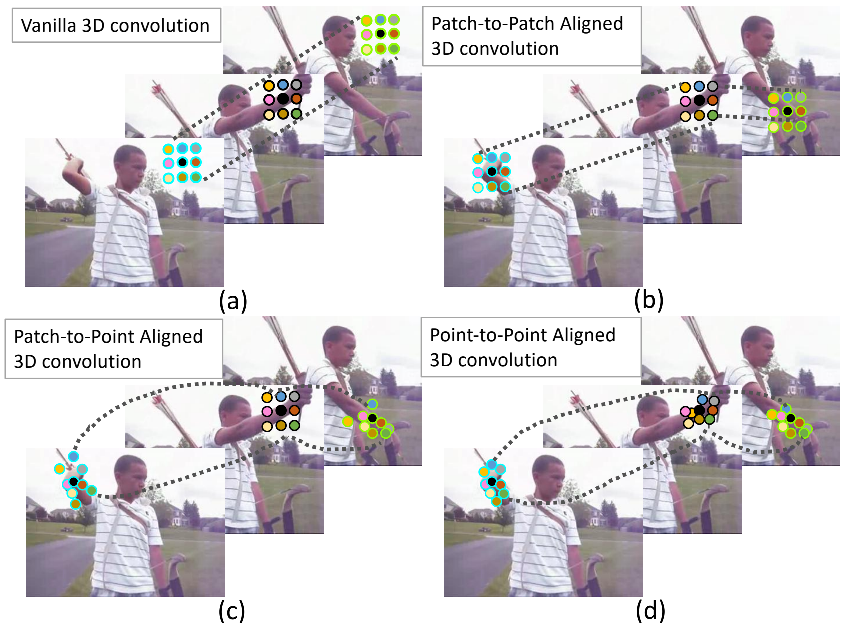

Vanilla 3D convolution
(a)
Patch-to-Patch Aligned 3D convolution
(b)
Patch-to-Point Aligned 3D convolution
(c)
Point-to-Point Aligned 3D convolution
(d)

FIGURE 5.1: **Aligned 3D convolutions.** (a) Vanilla 3D convolution has a receptive field corresponding to a regular cubic grid. (b) Patch-to-patch aligned 3D convolution calibrates the centers of spatial receptive windows between successive frames. (c) Patch-to-point aligned 3D convolution has independent offsets for each spatial location in the local window. (d) Point-to-point aligned 3D convolution tracks the deformed features on each successive frame. Different filling colors of the points represent different spatial positions and edge colors of the points represent different temporal positions.

and sampling rate of inputs to a 3D network affect the action classification accuracy. The proposed aligned convolutions maintain good performance on low-frame-rate videos. We observe the learned offsets contain motion information, which makes us develop a dual learning strategy to mimic the two-stream regime without the need to pre-compute optical flow. The aligned 3D convolution blocks are easy to embed into many existing backbones.

5.2 Related Work

Action recognition using 3D convolution variants. The 3D convolution [6, 104] is a natural solution for action analysis as it is able to model appearance and motion information simultaneously. Tran *et al.* [202] proposed the C3D network to learn spatio-temporal features on large-scale training datasets and revealed 3D ConvNet outperformed 2D ConvNet. Carreira and Zisserman [21] introduced I3D based on 2D ConvNet inflation to copy the success of ImageNet training. Following these, both [201] and [221] observed factorizing the 3D convolutional filters into a 2D spatial convolution and a 1D temporal convolution yielded significant gains in accuracy. For efficient action recognition, Zolfaghari *et al.* [249] proposed to employ 3D convolutions on top layers only and Lin *et al.* [133] developed a temporal shift module inserted to 2D convolutions to mimic 3D convolutions. We develop aligned 3D convolutions to learn more precise motion information.

Action recognition using trajectories. As a powerful motion modeling, trajectory was leveraged to improve action recognition in many previous works [145, 192, 209, 208, 210, 245]. Sun *et al.* [192] modeled the spatio-temporal context information residing with the SIFT-based trajectories of sparse salient points. Wang *et al.* [209] extracted hand-crafted features aligned with the trajectories to characterize shape, appearance and motion. They stated dense trajectories guaranteed better performance than sparse sampling. Furthermore, in [208] the authors further improved the dense trajectories by taking into account camera-motion compensation. In [210], Wang *et al.* utilized trajectories to pool and integrate off-the-shelf deep features.

In order to learn better representations in conjunction with feature tracking, Zhao *et al.* [245] proposed a trajectory convolution to trace the features extracted from 2D convolutions along the trajectories represented by pre-computed optical flow. They also combined a dense flow prediction network jointed trained with the action classification. Compared to the previous trajectory-based methods, it built an end-to-end learning. There are still two limitations: It required supervision from optical flow; Both forward- and backward-optical flow needs to be computed. Thus, all the trajectories-based methods suffer from a heavy computation and memory requirement. Our aligned convolutions automatically learn offsets between successive frames. It is driven only by the classification task, without the need for any pre-computed optical flow or optical flow supervision.

Action recognition without optical flow. Optical flow is helpful for action analysis, as demonstrated by the successful two-stream regime [181, 211, 240]. However, the requirement of optical flow pre-computation makes action recognition less practical. Some works [213, 26, 231, 93] focus on learning to capture long range dependencies from RGB inputs only. Both [190] and [32] train a network from RGB inputs, while distilling knowledge from a network that recognized actions from optical flow. In this way, inference only needs RGB inputs, but training still requires optical flow. Feichtenhofer *et al.* [55] observed the 3D convolution focused more on spatial semantics than temporal information on low frame-rate videos. Hence, they proposed two separate pathways for semantic and temporal information. We prefer to redesign the 3D convolution to address the inability of 3D convolutions to capture motion changes.

5.3 Aligned 3D Convolution Blocks

We consider a 3D convolution with temporal kernel size equaling 3 and an arbitrary spatial kernel size. We define F ($\in \mathbb{R}^{C\times T\times H\times W}$) as input feature maps with temporal

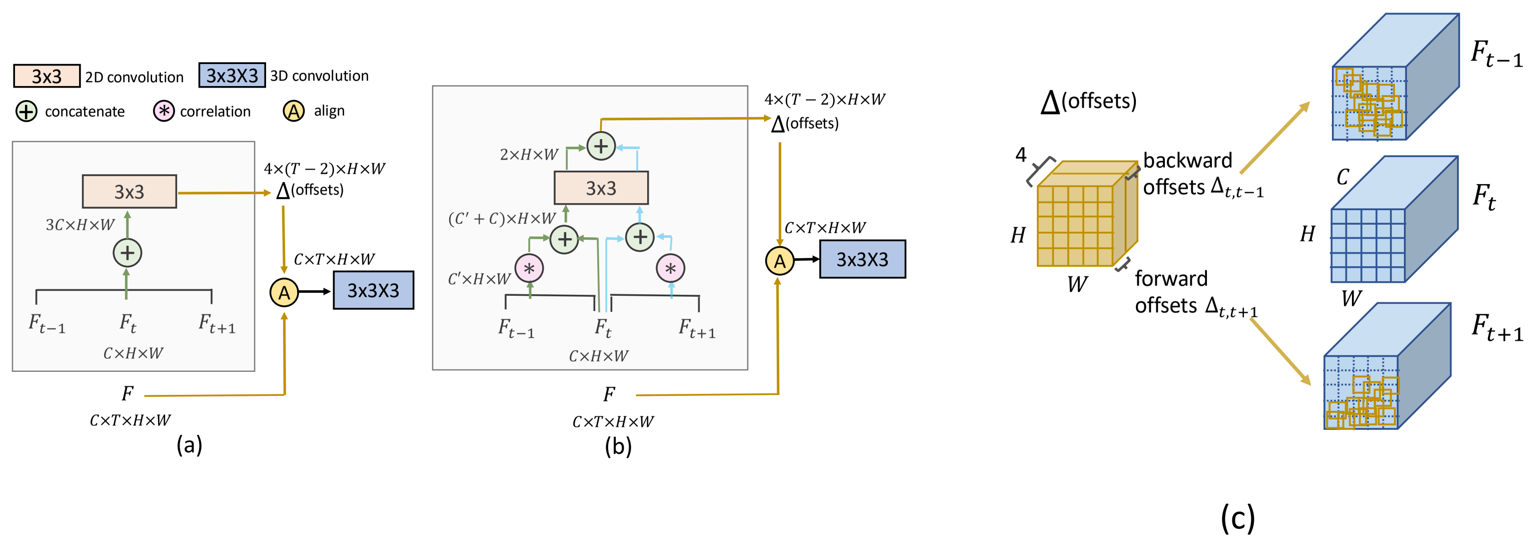


TABLE 5.1: **Ailgned 3D convolution blocks** (a) based on concatenation features; (b) based on correlation features. The feature maps are shown as the shape of their tensors, *e.g.*, $C \times T \times H \times W$ for C channels. (c) Alignment operation in patch-to-point aligned convolution.

dimension T. C denotes the number of channels, and H and W denote the spatial dimensions. An aligned 3D convolution block contains two modules. First, it utilizes the correlation between successive temporal feature maps to learn robust displacements of the feature points. Second, it applies the displacements to rectify the spatial locations for convolution. Intuitively, the aligned 3D convolution block is able to better utilize the inherent motion in videos. The details of the building block are visualized in Figure 5.1 and discussed next.

5.3.1 Generating Offsets

The key idea of the aligned 3D convolution block is to automatically generate feature offsets for each spatial position between successive temporal feature maps. As shown in Figure 5.1, we take a 3D convolution kernel with size of $3 \times 3 \times\ 3$ as an example. F_t, F_{t-1} and F_{t+1} ($\in \mathbb{R}^{C \times H \times W}$) are feature slices in F at the t-th, $t-1$-th and $t+1$-th time steps.

Concatenation features. In Figure 5.1 (a), we naively concatenate the three successive feature slices along the time dimension and then use a 2D convolution to produce the spatial offsets among them. The generated offsets Δ includes backward offsets $\Delta_{t,t-1} \in \mathbb{R}^{2 \times H \times W}$ between F_t, F_{t-1} and forward offsets $\Delta_{t,t+1} \in \mathbb{R}^{2 \times H \times W}$ between F_t, F_{t+1}. Intuitively, this way is insufficient to explore relationships between successive temporal features due to absence of frame interactions. Instead, we use correlation features [46], which were previously successful for tracking [10] and optical flow estimation [95].

Correlation features. Inspired by [46], correlation operations are used to match feature points between successive frames. Taking the forward process as an example, we calculate the correlation features between F_t and F_{t+1}. The forward offsets are produced from the concatenation of the correlation features and F_t. In a similar fashion we obtain the backward offsets between F_t and F_{t-1}. The aligned 3D convolution block based on correlation features is illustrated in Figure 5.1 (b). Specifically, the correlation operation performs point-wise feature comparisons of two feature maps. Considering that the displacements between two frames are not too large, we restrict the correlation in a local square window $[-d, d]$. Compared to considering all possible circular shifts in a feature map, this reduces the computation and output dimensionality. It also avoids matching features at large distance. Equation 5.1 shows a correlation of two feature maps F_t and F_{t+1} with largest displacement d.

$$F^{corr}_{t,t+1}(i,j,p,q) = \langle F_t(i,j), F_{t+1}(i+p, j+q) \rangle \tag{5.1}$$

Where $-d < p < d$ and $-d < q < d$. Each location (i,j) on F_t will be compared to all locations in a neighborhood $[-d, d]$ centered at (i,j) on F_{t+1}. Thus, the correlation feature has $(2d+1) \times (2d+1)$ channels.

5.3.2 Aligning Features

After acquiring the offsets between successive temporal feature maps, we use them to rectify the locations before a convolutional operation. The vanilla 3D convolution with kernel size $(2\tau + 1) \times (2\rho + 1) \times (2\sigma + 1)$ calculates the feature on each spacetime position $(t, x, y) \in [0, T) \times [0, H) \times [0, W)$ as:

$$O_{t,x,y} = \sum_{dt=-\tau}^{\tau} \sum_{dx=-\rho}^{\rho} \sum_{dy=-\sigma}^{\sigma} w^{dt,dx,dy} \cdot F_{t,x,y}^{dt,dx,dy}. \tag{5.2}$$

where $w^{dt,dx,dy}$ are the 3D convolution weights. As shown in Figure 5.1 (a), the information is integrated from the same spatial locations on the successive feature maps. In order to gather information from related locations on the successive feature maps, we introduce the aligned convolutions with a general expression as:

$$O_{t,x,y} = \sum_{dt=-\tau}^{\tau} \sum_{dx=-\rho}^{\rho} \sum_{dy=-\sigma}^{\sigma} w^{dt,dx,dy} \cdot F_{t,x,y}^{dt,dx+\Delta^x,dy+\Delta^y}. \tag{5.3}$$

Where Δ^x and Δ^y are spatial offsets between F_t and F_{t+dt} along the $x-$direction and $y-$direction. They have variable expressions for different version of the aligned convolution.

Patch-to-patch aligned convolution. We first define a patch-to-patch aligned 3D convolution. The Δ^x and Δ^y in Equation 5.3 are:

$$\Delta^x = \Delta_{t,t+dt}(x),\ \Delta^y = \Delta_{t,t+dt}(y). \tag{5.4}$$

For $dt = 0$, $\Delta_{t,t}$ means the offsets between F_t and F_t, are zeros. As shown in Figure 5.1 (b), the receptive fields of a $(2\rho + 1) \times (2\sigma + 1)$ spatial kernel share the same offsets produced on center position (x, y). As the learned offsets are real-values, we apply bilinear interpolation to calculate the features on the fractional location $(x, y) + \Delta_{t,t+dt}(x, y)$ of F_{t+dt}. The patch-to-patch aligned convolution helps maintain local structures but at the expense of additional computation. It needs to perform $(2\rho + 1) \times (2\sigma + 1)$ bilinear interpolations for each position (x, y).

Patch-to-point aligned convolution. In order to reduce the computation of the patch-to-patch aligned convolution, we propose a patch-to-point aligned convolution by replacing Δ^x and Δ^y in Equation 5.3 as:

$$\Delta^x = \Delta_{t,t+dt}(dx),\ \Delta^y = \Delta_{t,t+dt}(dy). \tag{5.5}$$

In this way, each position in the spatial window $(2\rho + 1) \times (2\sigma + 1)$ can use its own offsets recorded in Δ when the 3D convolution is performed on (x, y). It is reasonable, as the local structures do not change much between two neighboring frames. As a result, it only needs one bilinear interpolation on (x, y). Hence, there is a 88.8% computation reduction compared to the patch-to-patch aligned convolution. We show patch-to-point aligned convolution in Figure 5.1 (c) and Figure 5.1 (c).

Relation with existing work. It is of interest to note the implementation of our aligned convolution can be partially borrowed from the 2D deformable convolution [33], even though the two methods have different motivation. The 2D deformable convolution was designed to enhance the transformation modeling capacity of a convolution operation. However, the 3D aligned convolution focuses on tracking temporal information in videos. Specifically, the 2D deformable convolution yields the output features on spatial location (x, y):

$$O_{x,y} = \sum_{dx=-\rho}^{\rho} \sum_{dy=-\sigma}^{\sigma} w^{dx,dy} \cdot F_{x,y}^{dx+\Delta^{2D}(dx),dy+\Delta^{2D}(dy)}. \tag{5.6}$$

Where $\Delta^{2D} \in \mathbb{R}^{2\times(2\rho+1)\times(2\sigma+1)\times H\times W}$ are the generated offsets used to augment a regular grid covered by a normal 2D convolution.

Furthermore, a 3D deformable convolution can be extended from Equation 5.6. However, it was found to be ineffective for action recognition in [245]. Instead, they use a 2D normal convolution plus a 1D trajectory convolution to process videos:

$$1D : O_{t,x,y} = \sum_{dt=-\tau}^{\tau} w^{dt} \cdot F_{t,x,y}^{dt,dx+\Delta'_{t,t+dt}(x),dy+\Delta'_{t,t+dt}(y)} \tag{5.7}$$

Where $\Delta'_{t,t+dt}$ are the spatial offsets represented by optical flow. For each frame, both forward and backward optical flow are needed, which requires heavy pre-computation. Our aligned convolution automatically learn the offsets from the task objective, *e.g.* action classification, without any optical flow supervision. Besides, from all the equations above, one can clearly see the difference between our methods and the existing work.

Point-to-point aligned convolution. Enlightened from the deformable spirit, we further propose a point-to-point aligned convolution. The deformed features of the successive frames are calibrated by the aligned convolution (as shown in Figure 5.1 (d)). Specifically, Δ^x and Δ^y in Equation 5.3 are expressed as:

$$\begin{aligned} \Delta^x &= \Delta_t(dx) + \Delta_{t,t+dt}(dx) \\ \Delta^y &= \Delta_t(dy) + \Delta_{t,t+dt}(dy). \end{aligned} \tag{5.8}$$

Where Δ_t are the irregular sampling locations on F_t. It is a generalized case of the patch-to-point aligned convolution, where all Δ_t are zeros. With the enhanced transformation ability, the point-to-point aligned convolution is able to gather finer motion information.

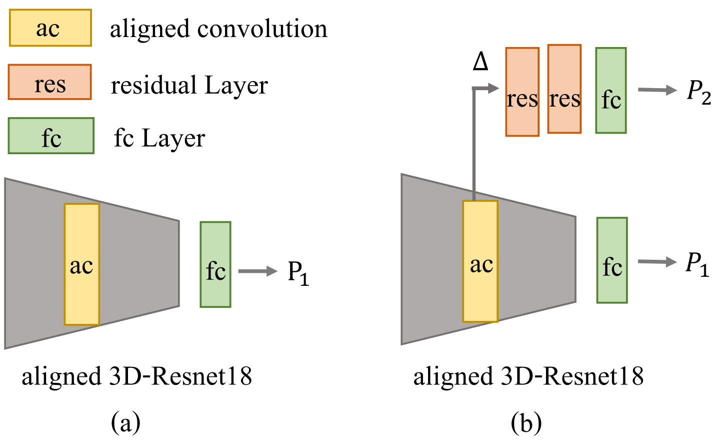


FIGURE 5.2: **Dual learning.** (a) The normal learning structure only provides predictions from the main branch. (b) The dual learning structure also provides predictions from the offset branch.

5.3.3 Dual Learning with Offsets

It is known that motion is important for action classification. Most previous works [211, 54] utilize pre-computed optical flow in a two-stream training regime [181]. Our learned offsets between successive feature maps capture motion information as shown in Figure 5.5 (e). We explore how much the learned offsets can contribute to action classification. Besides the normal learning shown in Figure 5.2 (a), we also introduce a dual learning strategy as shown in Figure 5.2 (b) to learn the same action classification task simultaneously. One learning takes the predictions P_1 from the main branch and the other takes the predictions P_2 from the offset branch. The total loss is:

$$\mathcal{L}_{total} = \mathcal{L}_1(P_1, Y) + \mathcal{L}_2(P_2, Y). \tag{5.9}$$

where $\mathcal{L}_1(\cdot)$ and $\mathcal{L}_2(\cdot)$ represent the softmax Cross Entropy Loss. Y denotes the label.

5.4 Experiments

5.4.1 Implementation

Datasets. We perform our experiments on three action classification datasets. UCF101 [187] includes 13,320 videos in 101 classes and has three train/test splits. *We report on split1 for our ablation studies.* HMDB51 [119] contains 6,766 video clips from 51 action categories. Three train/test splits are provided. Kinetics400 [107] consists of 400 categories. Because of the expiration of some YouTube links, we can only download 234,4910 out of 246,535 videos for training and 19,328 for validation. The *top-1* and *top-5* accuracy are reported.

	top-1	*top-5*
3D baseline	47.59	69.59
Trajectory convolution [245]	49.60	73.29
Patch-to-patch aligned convolution	49.76	**75.99**
Patch-to-point aligned convolution	48.76	72.26
Point-to-point aligned convolution	**51.37**	74.83

TABLE 5.2: **Aligned *vs.* Vanilla 3D convolution.** The aligned convolutions perform better than the vanilla 3D convolution on UCF101. Note that the trajectory convolution requires pre-computed optical flow for its alignment.

Training and testing. In order to demonstrate the effectiveness of our strategies, we chose a simple and light 3D-Resnet18 [82] backbone, without any bells and whistles. When training, we perform a 112×112 multi-scale crop for each frame [211]. We apply the same training and testing regimes as in [78]. For ablation studies on UCF101 split1, the network is trained **from scratch** with an initial learning rate of 0.1, which is reduced by 0.1 after the validation loss saturates. On Kinetics400, the training process takes 150 epochs with the initial learning rate of 0.01 and batch size of 64. For HMDB51, the network is initialized by a pre-trained mondel on Kinetics400. For testing, the frames are rescaled and then center cropped to 112×112. We take every 16 frames as an input sequence and average the scores of all short sequences to get the final score for one video. The code and models will be released.

5.4.2 Ablation Studies

Aligned *vs.* Vanilla 3D convolution. We replace the normal 3D convolutions in the first residual block, res2, with the aligned 3D convolution blocks using correlation features. Input length is 16. We also compare with the trajectory convolution proposed in [245]. For the trajectory convolution, we pre-compute forward and backward optical flow using FlowNet2 [95] and take them as the offsets between successive frames. The results are shown in Table C.1. All the aligned convolution variants perform better than the vanilla 3D convolution. The patch-to-patch and point-to-point aligned convolution are even better than the trajectory convolution. Due to their mechanism of automatically learning offsets, aligned convolution blocks, without using pre-computed optical flow, are more effective and practical.

Correlation *vs.* Concatenation. Intuitively, learning more precise offsets between successive frames improves the alignment. In Figure 5.3, we compare two ways to generate the offsets. Aligned convolutions are applied to the first residual block res2 and input length is 32. All aligned convolution methods outperform the baseline. Using correlation features to produce offsets improves the accuracy more than using

	16×1		32×1		64×1		16×2		16×4		32×2		32-rand	
	top-1	*top-5*	*top-1*	*top-5*	*top-1*	*top-5*	*top-1*	*top-5*	*top-1*	*top-5*	*top-1*	*top-5*	*top-1*	*top-5*
3D baseline	47.59	69.59	48.57	71.87	51.93	75.89	43.57	67.03	45.24	70.15	47.54	71.31	53.31	77.45
Patch-to-point	48.76	72.26	52.25	74.17	63.80	83.42	49.05	71.92	50.19	73.03	55.26	77.76	56.90	78.34
<Improvement>↑	+1.17	+2.67	+3.68	+2.30	+11.87	+7.53	+5.48	+4.89	+4.95	+2.88	+7.72	+6.45	+3.59	+0.89

TABLE 5.3: **Impact of input length and sampling rate.** For group comparisons, same input length is marked with same color and same sampling rate shares same color. The accuracy gains of our method are shown in the last row. Our patch-to-point aligned convolution performs better than the 3D baseline for all the cases on UCF101.

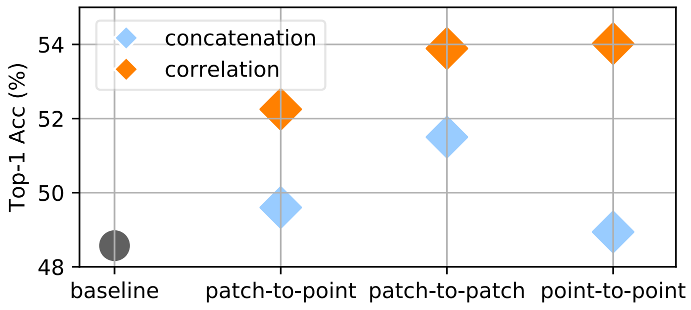


FIGURE 5.3: **Correlation *vs.* Concatenation.** For aligned convolution blocks generating offsets from correlation features is more effective than concatenation features on UCF101.

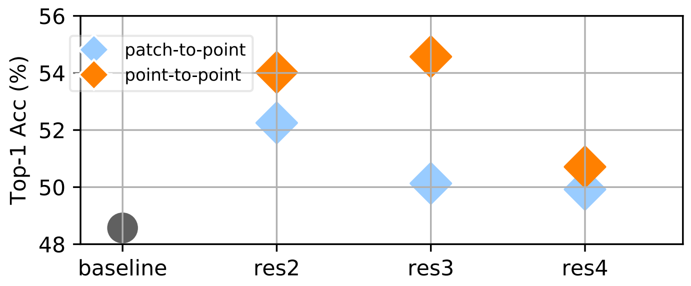
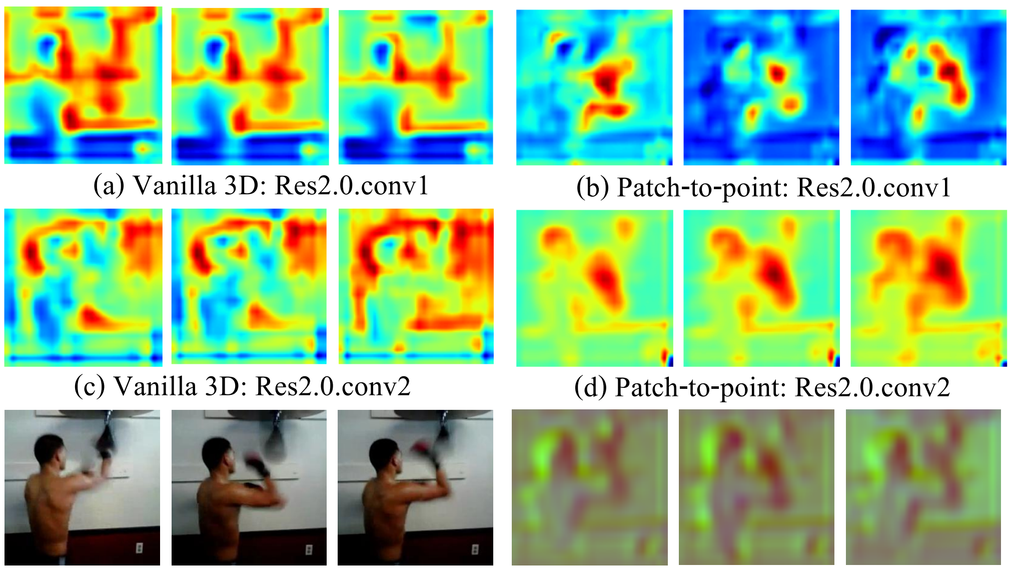


FIGURE 5.4: **Where to apply aligned convolution?** Adding the aligned convolution blocks to bottom layers gives better performance on UCF101.

concatenation features. It is predicable as the correlation operation models the relationship between successive frames better. Unless specified, otherwise, correlation features are utilized in all the following experiments.

Where to apply aligned convolution? We plug aligned convolution blocks into different residual layers res2, res3 and res4. Each residual layer has two blocks. We conduct experiments using the patch-to-point aligned convolution which is the lightest version and the point-to-point aligned convolution which is the generalized version. Clips length is 32. Seen from Figure 5.4, res2 is the best layer to add the patch-to-point aligned convolution. It is reasonable as finer matching is performed on bottom layers. The point-to-point aligned convolution gets highest *top-1* accuracy at the layer res3. Applying the aligned convolution to top layer decreases the improvement. We recommend to add the aligned convolution to res2.

Impact of input length and sampling rate. The input length and video sampling rate decide how much temporal information in one training sample is seen by a network, which affects action classification accuracy. Intuitively, the longer and denser the input is, the better the performance may be. However, a network can only process limited data per iteration. Thus, it is interesting to seek a balance between the input length and sampling rate. In Table B.2, we present results of using inputs with different length $\times$ sampling rate. For example, 16×2 means the input length is 16



(a) Vanilla 3D: Res2.0.conv1 (b) Patch-to-point: Res2.0.conv1

(c) Vanilla 3D: Res2.0.conv2 (d) Patch-to-point: Res2.0.conv2

(e) Input sequence (f) Patch-to-point: Res2.0.conv2.offsets

FIGURE 5.5: **Feature visualization.** The feature maps of res2.0.conv1 and res2.0.conv2 are compared between the vanilla 3D convolution ((a) and (c)) and the patch-to-point aligned 3D convolution ((b) and (d)). (e) shows the input sequence. The learned offsets are shown in (f). The patch-to-point aligned convolution makes the model focus more on actions.

but it covers 32 frames in the raw view. Besides, a random sampling from [249] is also applied. In this way, the video is split into N subsections of equal length. And in each subsection, exactly one frame is sampled randomly. As a result, totally N frames are taken as input. We use N-rand to represent this sampling. Here, just the patch-to-point aligned convolution is compared.

From Table B.2, it is concluded that: (1) With the same sampling rate, a longer input performs better for both the normal 3D convolution and the aligned 3D convolution. It is not surprising as longer input brings more temporal information. (2) However, with a same input length, a bigger sampling rate may not be beneficial to normal 3D convolution although it also covers a longer temporal view. For instance, inputs 16×2 and 16×4 for normal 3D convolution network obtain worse results than 16×1. We suppose there are two reasons. A big sampling rate not only means a longer temporal window but also losing details as the displacements between successive frames are larger. The normal 3D convolution is unable to handle them well. In contrast, due to the function of matching features, the aligned convolution suffers less from these issues. The patch-to-point aligned convolution achieves a *top-1* accuracy of 48.76%, 49.05% and 50.09% respectively when input are 16×1, 16×2 and 16×4. (3) With the same temporal view, a bigger sampling rate reduces the accuracy of both the two convolutions. It is expected as temporal information becomes sparse and displacements becomes larger. For the patch-to-point aligned

	Resnet18	Resnet50	I3D
3D baseline	56.11	62.72	79.96
Patch-to-point	58.34	65.07	81.36

TABLE 5.4: **Generalization on backbones.** *top-1* accuracy comparisons using different backbones. All models are trained from ImageNet pretrained models. The aligned convolutions performs better on UCF101.

convolution, it is influenced more than the normal 3D convolutions. For example, for inputs 64×1 and 32×2, there is 8.54% decrease of *top-1* accuracy using the aligned convolution compared to 4.39% decrease using normal convolution. We infer that the aligned convolution has a tolerance of the dislocations between two frames. (4) Another interesting point is that input with 32-rand outperforms 32×1 and 32×2. It is because this random sampling covers a whole video even with uncertain displacements between successive frames. It can be concluded that the length of temporal information is more important than the density of temporal information for action classification. Moreover, N-rand sampling leads to much faster testing. (6) Whatever the input, the patch-to-point aligned convolution leads to obvious improvements from what is shown in the last row. Taking input 64×1, it even increases 11.87% with regard to *top-1* accuracy. We believe the proposed aligned convolution is capable to capture long-range dependencies.

Generalization on backbones. The aligned convolution blocks are easy to plug into many existing backbones. We report the *top-1* accuracy on UCF101 split1 using Resnet18, Resnet50 and I3D [21] in Table 5.4. Resnet18 takes as input 16 frames and the two others take as input 32 frames. All the model weights are initialized with ImageNet [40] pre-trained models in the way introduced in [21]. Seen from Table 5.4, the patch-to-point aligned convolution works better than vanilla 3D convolution in any backbone.

5.4.3 Analysis

Feature visualization. By visualizing the feature maps from the learned models, it is clear to see how the aligned convolution influences the behavior of the network. One example of 'boxing speed bag' is shown in Figure 5.5, where (e) presents the input sequences and (a) and (c) respectively show the feature maps of the layer res2.0.conv1 and res2.0.conv2 of the vanilla 3D network. There is high activation on the background. When replacing a normal 3D convolution by the patch-to-point aligned convolution for res2.0.conv2, the network focuses more on the moving objects, which is shown in (d). Moreover, this effect will improves the whole network and implies the network learns better features, like feature maps of res2.0.conv1 in (b). We also visualize the learned offsets in (f). Different with optical flow, the offset is a feature-level representation. We can see the activation for moving objects, especially

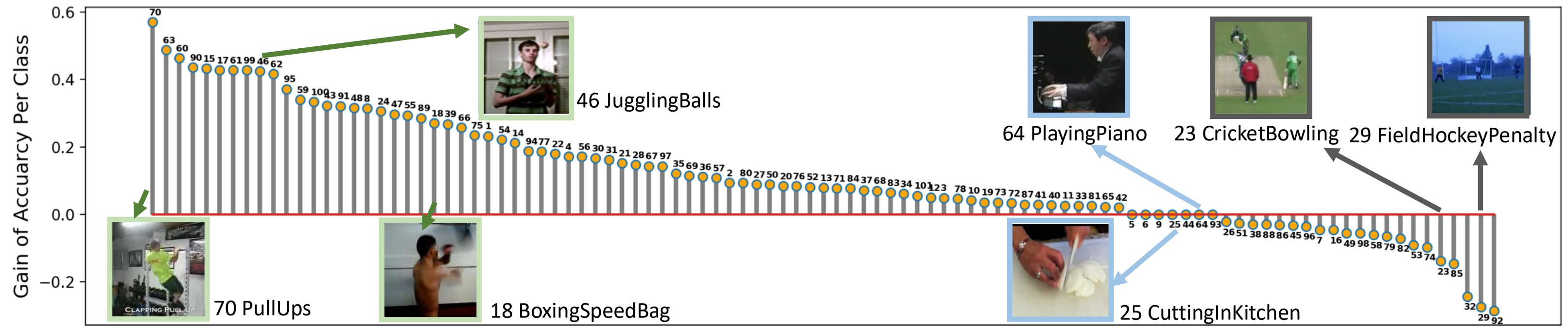


TABLE 5.5: **Result analysis.** Improvements of *top-1* accuracy per class of UCF101 using patch-to-point aligned convolution compared to normal 3D convolution. The index of the label is marked on the bar of the respective class. The aligned convolution performs better for most classes.

	top-1	*top-5*
normal learning		
3D baseline (P_1)	48.57	71.87
Patch-to-point (P_1)	52.25	74.17
dual learning		
Patch-to-point (P_1)	52.91	76.06
Patch-to-point (P_2)	50.45	76.10
Patch-to-point ($P_1 + P_2$)	54.63	78.16

TABLE 5.6: **Learning from offsets.** Result comparisons of normal learning and the dual learning. Dual learning helps to further improve the performance on UCF101.

for the edges of the objects. We utilize the offsets in a dual learning to further improve action classification.

Result analysis. For further evaluation, the *top-1* accuracies for each class of UCF101 are considered. In Figure 5.5, we show how much the aligned convolution improves per class compared to the normal 3D convolution. The input length is 64. The largest improvement happening on 'pull up' is even about 60%. For some speedy actions like 'juggling balls' and 'boxing speed bag', which have large movements, the aligned convolution aids recognition. For the actions with modest motion such as 'playing piano' and 'cutting in kitchen', the aligned convolution performs on par with the vanilla 3D convolution. If the actors are tiny, like in 'cricket bowling' and 'field hockey penalty' shown in the figure, the aligned convolution may not work well. It is observed that the aligned convolution obtains improvements for about 73% of 101 classes, equal performance on very few classes and worse performance on 20% of the classes. Its gains on fast actions profit from the function of tracing changes.

5.4.4 Learning from Offsets

From the offset feature maps shown in Figure 5.5 (e), it is expected the offsets resemble a feature-level motion representations. Hence, the offsets could help to further improve action classification. As described in Section 3.3, we train a network taking as input 32 frames using our dual learning strategy. The results are illustrated in Table 5.6. It is interesting that dual learning even improves the predictions P_1 from the main branch, with 0.66% of *top-1* and 1.99% of *top-5* gains. We guess the dual learning helps to learn more precise offsets. The predictions P_2 from the offset branch obtains good results, even better than the 3D baseline. It supports our statement that the learned offsets contain valuable information. Moreover, combining P_1 and P_2 brings further improvements. It illustrates that the offset branch learns additional motion information which are not captured by the main branch.

	Kinetics400		UCF101		HMDB51	
	top-1	*top-5*	*top-1*	*top-5*	*top-1*	*top-5*
Hara *et al.* [78]	54.2	78.1	84.4	-	56.4	-
3D baseline	62.2	84.2	88.0	98.6	63.3	90.2
Patch-to-point	64.2	87.3	90.8	99.3	65.7	91.9

TABLE 5.7: **Action classification** on Kinetics400, UCF101 and HMDB51. The models for UCF101 and HMDB51 are initialized by a pre-trained model on Kinetics. The patch-to-point aligned convolution achieves better results.

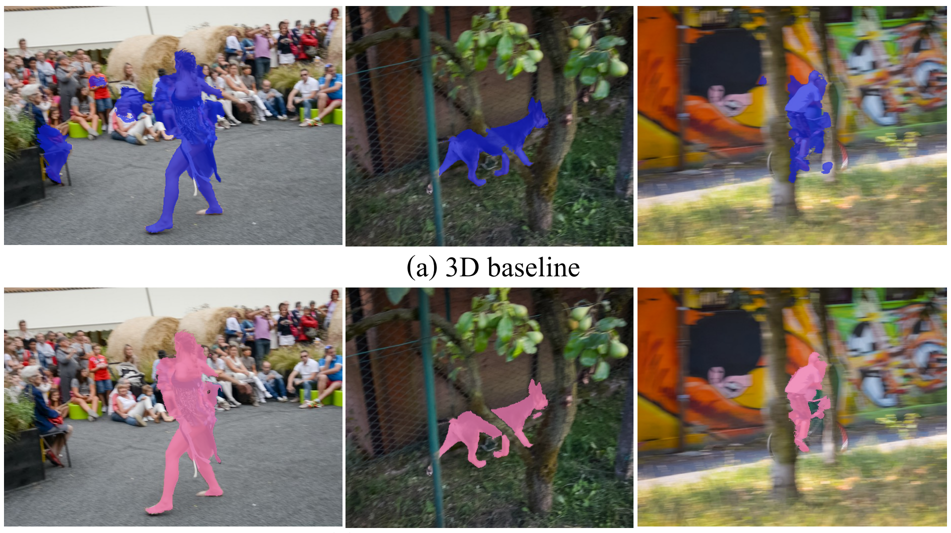
5.4.5 Do We Have the State-of-the-art?

Action classification. Recently, pursuing the state-of-the-art on a huge video dataset like Kinetics400, has become a battle of computation power. It all depends on the availability of multiple GPUs, deeper networks, bigger input length, larger batch size, and pre-computed optical flow. Limited by GPU usage, we only adopt a lighter network Resnet18 with the patch-to-point aligned convolution on res2 for fast training. The input is $16 \times 112 \times 112$ and batch size is set to 64. For fair apple-to-apple comparisons, the results of [78] with the identical settings are reported in Table 5.7. Only RGB images are used. We are well aware that some recent works [21, 213, 55, 32, 105, 143, 245] achieved higher accuracy in the range of 72.0%-78.0% on the Kinetics validation set. However, all of them adopted deeper networks *e.g.* Resnet50 or Resnet101. They used $32 \times 224 \times 224$ or $64 \times 224 \times 224$ as input and larger batch sizes. Our objective is to show that the aligned convolution improves the vanilla 3D convolution for action recognition. In Table 5.7, UCF101 and HMDB51 are trained from the Kinetics pre-trained model. For all datasets the patch-to-point convolution improves over the baseline. In Table 5.4, we already showed the improvement generalizes over deep backbones as well. We believe it has achieved an improvement over the commonly used 3D convolution, that is of community interest.

Moving object segmentation. We also investigate the patch-to-point aligned convolution block on moving object segmentation in videos. We build a 3D convolution network based on the DeepLab [23] model with Resnet18 as an encoder. The details of the network are provided in the supplementary material. We evaluate on ***DAVIS2016*** (Densely Annotated Video Segmentation 2016) following [155]. Region similarity $\mathcal{J}$ and contour accuracy $\mathcal{F}$ are reported in Table 5.8. The aligned convolution also improves the performance of moving object segmentation in videos. We present the qualitative results on three video sequences in Figure 5.6. The patch-to-point aligned convolution is more robust to crowded background and occlusion cases.

	$\mathcal{J}$		$\mathcal{F}$	
	Mean↑	Recall↑	Mean↑	Recall↑
3D baseline	67.5	82.0	63.0	73.4
Patch-to-point	69.1	83.0	65.0	74.4

TABLE 5.8: **Moving object segmentation** on DAVIS2016. Region similarity $\mathcal{J}$ and contour accuracy $\mathcal{F}$ are reported. Higher values are better. The aligned convolution outperforms the normal 3D convolution in this setting.


(a) 3D baseline

(b) patch-to-point aligned 3D

FIGURE 5.6: **Moving object segmentation visualizations.** The aligned convolution leads to less wrong detection and better contours for crowded background and occlusion cases.

5.5 Conclusion

In this chapter, we propose aligned 3D convolutions for video action recognition. It generates offsets to align the feature points between successive frames. The mechanism allows to trace the motion changes and conduct convolution operators on rectified locations of successive features. The aligned convolution makes it possible to exploit motion information using RGB inputs only for action recognition, without the need to pre-compute any forward or backward optical flow. Moreover, due to its ability to track motion at feature level, it performs well on variable frame-rate videos. The aligned convolution also presents a better ability on another video challenge, namely moving object segmentation. We expect the aligned 3D convolution is useful for other tasks as well, most notably action detection, which we aim to address in future work.

Chapter 6

TubeR: Tubelet Transformer for Video Action Detection

6.1 Introduction

This chapter tackles the problem of spatio-temporal action detection in videos [18, 199, 101], which plays a central role in advanced video search engines, robotics, and self-driving cars. Action detection is a compound task, requiring the localization of per-frame person instances, the linking of these detected person instances into action tubes and the prediction of their action class labels. Two approaches for spatio-temporal action detection are prevalent in the literature: frame-level detection and tubelet-level detection. Frame-level detection approaches detect and classify the action independently on each frame[71, 154, 184], and then link per-frame detections together into coherent action tubes. To compensate for the lack of temporal information, several methods simply repeat 2D proposals [74, 191, 67] or offline person detections [55, 219, 197, 152] over time to obtain spatio-temporal features (Figure 6.1 top left). Alternatively, tubelet-level detection approaches [88, 106, 240, 185, 224, 131], directly generate spatio-temporal volumes from a video clip to capture the coherence and dynamic nature of actions. They typically predict action localization and classification jointly over spatial-temporal hypotheses, like 3D cuboid proposals [88, 106] (Figure 6.1 top right). Unfortunately, these 3D cuboids can only capture a short period of time as the spatial location of a person changes as soon as they move, or due to camera motion. Ideally, this family of models would use flexible spatial-temporal tubelets that can track the person over a longer time, but the large configuration space of such a parameterization has restricted previous methods to short cuboids. In this work we present a tubelet-level detection approach that is able to simultaneously localize and recognize action tubelets in a flexible manner, which allows tubelets to change in size and location over time (Figure 6.1 bottom). This allows our system to leverage longer tubelets, which aggregate visual information of a person and their actions over longer periods of time.

We draw inspiration from sequence-to-sequence modelling in Natural Language Processing, particularly machine translation [206, 130, 193, 108], and its application to object detection, DETR [20]. Being a detection framework, DETR can be applied as a frame-level action detection approach trivially, but the power of the transformer

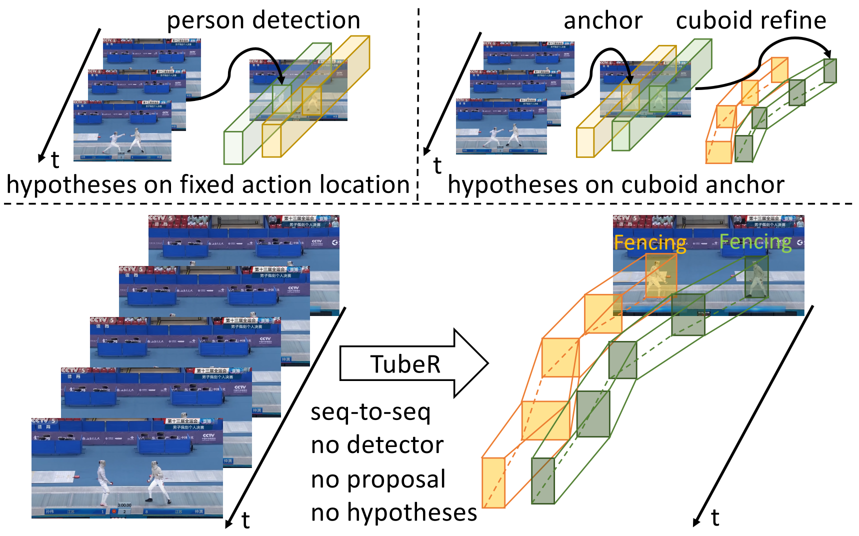


FIGURE 6.1: **TubeR** takes as input a video clip and directly outputs tubelets: sequences of bounding boxes and their action labels. TubeR runs end-to-end without person detectors, anchors or proposals.

framework, on which DETR is built, is its ability to generate complex structured outputs over sequences. In NLP, this typically takes the form of sentences but in this work we use the notion of decoder queries to represent people and their actions over video sequences, without having to restrict tubelets to fixed cuboids.

We propose a tubelet-transformer, we call TubeR, for localizing and recognizing actions from a single representation. Building on the DETR framework [20], TubeR learns a set of tubelet queries to pull action-specific tubelet-level features from a spatio-temporal video representation. Our TubeR design includes a specialized spatial and temporal tubelet attention to allow our tubelets to be unrestricted in their spatial location and scale over time, thus overcoming previous limitations of methods restricted to cuboids. TubeR regresses bounding boxes within a tubelet jointly across time, considering temporal correlations between tubelets, and aggregates visual features over the tubelet to classify actions. This core design already performs well, outperforming many previous model designs, but still does not improve upon frame-level approaches using offline person detectors. We hypothesize that this is partially due to the lack of more global context in our query based feature as it is hard to classify actions referring to relationships such as 'listening-to' and 'talking-to' by only looking at one person. To this end we introduce a context aware classification head that, along with the tubelet feature, takes the full clip feature from which our classification head can draw contextual information. This design allows the network to effectively relate a person tubelet to the full scene context where the tubelet appears and is shown to be effective on its own in our results section. One limitation of this design is the context feature is only drawn from the same clip our tubelet occupies. It has been shown [219] to be important to also include long term contextual features for the final action classification. Thus, we introduce a memory system inspired by [223] to compress and store contextual features from video content around the tubelet. We feed this long term contextual memory to our classification head using the same feature injection strategy and again show this gives an important improvement over the short term context alone. We test our full system on three popular action detection datasets (AVA [74], UCF101-24 [187] and JHMDB51-21 [103]) and show our method can outperform other state-of-the-art results.

In summary our contributions are as follows:

1. We propose TubeR: a tubelet-level transformer framework for action detection.
2. Our tubelet query and attention based formulation is able to generate tubelets of arbitrary location and scale.
3. Our context aware classification head is able to aggregate short-term and long-term contextual information.
4. We present state-of-the-art results on three challenging action detection datasets.

6.2 Related Work

Frame-level action detection. Spatio-temporal action detection in video has a long tradition, *e.g.* [18, 199, 101, 216, 154, 74, 197, 152]. Inspired by object detection using deep convolution neural networks, action detection in video has been considerably improved by frame-level methods [216, 154, 172, 184]. These methods perform localization and recognition per-frame and then link frame-wise predictions to action tubes. Specifically, they apply 2D positional hypotheses (anchors) or an offline person detector on a keyframe for localizing actors, and then focus more on improving action recognition. They incorporate temporal patterns by an extra stream utilizing optical flow. Others [74, 67, 191] apply 3D convolution networks to capture temporal information for recognizing actions. Feichtenhofer *et al.* [55] present a slowfast network to even better capture spatio-temporal information. Both Tang *et al.* [197] and Pan *et al.* [152] propose to explicitly model relations between actors and objects. Recently, Chen *et al.* [25] propose to train actor localization and action classification end-to-end from a single backbone. Different from these frame-level approaches, we target on tubelet-level video action detection, with an unified configuration to simultaneously perform localization and recognition.

Tubelet-level action detection. Detecting actions by taking a tubelet as a representation unit [129, 240, 185, 224, 131] has been popular since it was proposed by Jain *et al.* [101]. Kalogeiton *et al.* [106] repeat 2D anchors on per-frame for pooling ROI features and then stack the frame-wise features for predicting action labels. Hou *et al.* [88] and Yang *et al.* [224] depend on carefully-designed 3D cuboid proposals. The former directly detects tubelets and the later progressively refines 3D cuboid proposals across time. Besides using these box/cuboid anchors, Li *et al.* [131] detect tubelet instances relying on center position hypotheses. Hypotheses-based methods have difficulties to process long video clips, as discussed in the Introduction. We add to the tubelet tradition by learning a small set of tubelet queries to represent the dynamic

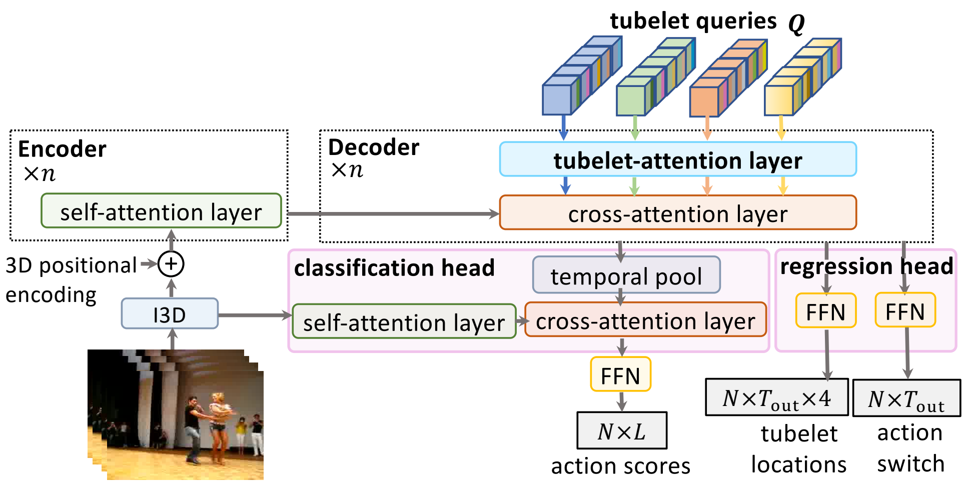


FIGURE 6.2: **The overall structure of TubeR**. Both encoder and decoder contain n stacked modules. We only show the key components in the encoder and decoder modules. Encoder models the spatio-temporal features from the backbone by self-attention layers. The decoder transforms a set of tubelet queries and finally predicts action tubelets (see Section 6.3.1). We utilize tubelet-attention layers (see Section 6.3.2) to model the relations between box query embeddings within a tubelet. Finally, we apply the context aware classification head and action switch regression head to predict tubelet labels and coordinates (see Section 6.3.3).

nature of tubelets. We reformulate the action detection task as a sequence-to-sequence learning problem and explicitly model the temporal correlations within a tubelet. Our method is capable to handle long video clips.
Transformer-based action detection. Vaswani *et al.* [206] proposed the transformer for machine translation, and soon after it became the most popular backbone on sequence-to-sequence tasks, *e.g.*, [130, 193, 108]. Recently, it has also demonstrated impressive advances in object detection [20, 248], image classification [45, 230] and video recognition [61, 239, 51]. Girdhar *et al.* [68] propose a video action transformer network for detecting actions. They apply a region-proposal-network for localization. The transformer is utilized for further improving action recognition by aggregating features from the spatio-temporal context around actors. We propose an unified solution to simultaneously localize and recognize actions.

6.3 Action Detection by TubeR

In this section, we present our TubeR that takes as input a video clip and directly outputs a tubelet: a sequence of bounding boxes and the action label. The TubeR design takes inspiration from the image-based DETR [20] but reformulates the transformer architecture for sequence-to-sequence(s) modeling in video (Figure 6.2).

Given a video clip $I \in \mathbb{R}^{T_{\text{in}} \times H \times W \times C}$ where T_{in}, H, W, C denote the number of frames, height, width, and colour channels, TubeR first applies a 3D backbone to extract video features $F_{\text{b}} \in \mathbb{R}^{T' \times H' \times W' \times C'}$, where T' is the temporal dimension and C' is the feature dimension. A transformer encoder-decoder is then utilized to transform the video features into a set of tubelet-specific features $F_{\text{tub}} \in \mathbb{R}^{N \times T_{\text{out}} \times C'}$, with T_{out} the output temporal dimension and N the number of tubelets. In order to process long video clips, we use temporal down-sampling to make $T_{\text{out}} < T' < T_{\text{in}}$, which reduces our memory requirement. In this case, TubeR generates sparse tubelets. For short video clips we remove the temporal down-sampling to make sure $T_{\text{out}}{=}T'{=}T_{\text{in}}$, which results in dense tubelets. Tubelet regression and associated action classification can be achieved simultaneously with separated task heads as:

$$y_{\text{coor}} = f(F_{\text{tub}}); y_{\text{class}} = g(F_{\text{tub}}). \tag{6.1}$$

where f denotes the tubelet regression head and $y_{\text{coor}} \in \mathbb{R}^{N \times T_{\text{out}} \times 4}$ stands for the coordinates of N tubelets, each of which is across T_{out} frames (or T_{out} sampled frames for long video clips). g denotes the action classification head, and $y_{\text{class}} \in \mathbb{R}^{N \times L}$ stands for the action classification for N tubelets with L possible labels.

6.3.1 TubeR Encoder-Decoder

Encoder. Different from the vanilla transformer encoder, the TubeR encoder is designed for processing information in the 3D spatio-temporal space. Each encoder layer is made up of a self-attention layer (SA), two normalization layers and a feed forward network (FFN), following [206]. We only put the core attention layers in all equations below.

$$F_{\text{en}} = \text{Encoder}(F_{\text{b}}), \tag{6.2}$$

$$\text{SA}(F_{\text{b}}) = \text{softmax}(\frac{\sigma_q(F_{\text{b}}) \times \sigma_k(F_{\text{b}})^T}{\sqrt{C'}}) \times \sigma_v(F_{\text{b}}), \tag{6.3}$$

$$\sigma(*) = \text{Linear}(*) + \text{Emb}_{\text{pos}}. \tag{6.4}$$

where F_{b} is the backbone feature and $F_{\text{en}} \in \mathbb{R}^{T'H'W' \times C'}$ denotes the C' dimensional encoded feature embedding. The $\sigma(*)$ is the linear transformation plus positional embedding. Emb_{pos} is the 3D positional embedding [239]. The optional temporal down-sampling can be applied to the backbone feature to shrink the input sequence length to the transformer for better memory efficiency.

Decoder. After having the encoded feature embedding F_{en}, the decoder with tubelet-attention (TA) and cross-attention (CA) layers is used to decode tubelet-specific features F_{tub} from F_{en} given a set of tubelet query Q:

$$F_{\text{tub}} = \text{Decoder}(F_q, F_{\text{en}}), \tag{6.5}$$

$$\mathrm{CA}(F_q, F_{\mathrm{en}}) = \mathrm{softmax}(\frac{F_q \times \sigma_k(F_{\mathrm{en}})^T}{\sqrt{C'}}) \times \sigma_v(F_{\mathrm{en}}), \tag{6.6}$$

$$F_q = \mathrm{TA}(Q). \tag{6.7}$$

where TA is the tubelet-attention for modeling relations between tubelet queries Q and outputs query features $F_q \in \mathbb{R}^{N \times T_{\mathrm{out}} \times C'}$, which will be described in section 6.3.2. $F_{\mathrm{tub}} \in \mathbb{R}^{N \times T_{\mathrm{out}} \times C'}$ is the tubelet specific feature. Note that with temporal pooling, $T_{\mathrm{out}} < T_{\mathrm{in}}$, TubeR produces sparse tubelets; For T_{out}=T_{in}, TubeR produces dense tubelets.

6.3.2 Tubelet Modeling

In order to model tubelets, we propose tubelet queries which are used to query tubelet-specific features from encoded features F_{en}, and a tubelet attention module for reasoning over relations between boxes within a tubelet.

Tubelet query. Directly detecting tubelets is quite challenging based on anchor hypotheses. The tubelet space along the spatio-temporal dimension is huge compared to the single-frame bounding box space. Consider for example Faster-rcnn [162] for object detection, which requires for each position in a feature map with spatial size $H' \times W'$, $K(=9)$ anchors. There are in total $KH'W'$ anchors. For a tubelet across T_{out} frames, it would require $(KH'W')^{T_{\mathrm{out}}}$ anchors to maintain the same sampling in space-time. To reduce the tubelet space, several methods [88, 224] adopt 3D cuboids to approximate tubelets by ignoring the spatial action displacements in a short video clip. However, the longer the video clip is, the less accurately a 3D cuboid hypotheses represents a tubelet. We propose to learn a small set of tubelet queries Q=$\{Q_1, ..., Q_N\}$ driven by the video data. N is the number of queries. The i-th tubelet query Q_i=$\{q_{i,1}, ..., q_{i,T_{\mathrm{out}}}\}$ contains T_{out} box query embeddings $q_{i,t} \in \mathbb{R}^{C'}$ across T_{out} frames. We learn a tubelet query to represent the dynamics of a tubelet, instead of hand-designing 3D anchors. We initialize the box embeddings identically for a tubelet query.

Tubelet attention. In order to model relations in the tubelet queries, we propose a tubelet attention module which contains two self-attention layers (shown in Figure 6.2). First we have a *spatial self-attention layer* that processes the spatial relations between box query embeddings within a frame *i.e.* $\{q_{1,t}, ..., q_{N,t}\}$, t=$\{1, ..., T_{\mathrm{out}}\}$. The intuition of this layer is that recognizing actions benefits from the interactions between actors, or between actors and objects in the same frame. Next we have our *temporal self-attention layer* that models the correlations between box query embeddings across time within the same tubelet, *i.e.* $\{q_{i,1}, ..., q_{i,T_{\mathrm{out}}}\}$, i=$\{1, ..., N\}$. This layer facilitates a TubeR query to track actors and generate action tubelets that focus on single actors instead of a fixed area in the frame. TubeR applies the tubelet attention module to tubelet queries Q for generating tubelet query features F_{q}.

6.3.3 Task-Specific Heads

The bounding boxes and action classification for each tubelet can be done simultaneously with independent task-specific heads. Such design maximally reduces the computational overheads and makes our system expandable.
Context aware classification head. The classification is achieved with a simple linear projection.

$$y_{\text{class}} = \text{Linear}_{\text{c}}(F_{\text{tub}}). \tag{6.8}$$

where $y_{\text{class}} \in \mathbb{R}^{N \times L}$ denotes the classification score on L possible labels, one for each tubelet.

It is known that context is important for understanding sequences [206]. We propose to leverage spatio-temporal context from the backbone features to further help video sequence understanding. Furthermore, inspired by [213, 239, 219] which explore long-range temporal information for video understanding, we propose to pull features with long-term context for classifying actions.

Short-term context head. We first propose to query the action specific feature from the context in the backbone features F_{b} to strengthen our tubelet-level features F_{tub}:

$$F_{\text{c}} = \text{CA}(\text{Pool}_t(F_{\text{tub}}), \text{SA}(F_{\text{b}})) + \text{Pool}_t(F_{\text{tub}}). \tag{6.9}$$

A self-attention layer is first applied to the backbone feature F_b, then a cross-attention layer utilizes tubelet-level features F_{tub} to query from F_b. $F_{\text{c}} \in \mathbb{R}^{N \times C'}$ is the final feature for action classification. A linear layer follows to predict action class scores.

Long-term context head. To utilize long-range temporal information but under certain memory budget, we adopt a two-stage decoder for long-term context compression as described in [223]:

$$\text{Emb}_{\text{long}} = \text{Decoder}(\text{Emn}_{n1}, \text{Decoder}(\text{Emb}_{n0}, F_{\text{long}}). \tag{6.10}$$

The long-term context $F_{\text{long}} \in \mathbb{R}^{(2w+1)T' \times H'W' \times C'}$ is a buffer that contains backbone features extracted from a long $2w$ adjacent clips concatenated along time. In order to compress the long-term context, we apply two stacked decoders with two token embedding Emn_{n0} and Emn_{n1} following [223]. In this sense, the long-term context F_{long} with original temporal dimension $(2w+1)T'$ is firstly compressed to an intermediate feature with a temporal dimension $n0$, and further to the final compressed feature Emb_{long} with a lower temporal dimension $n1$. Then we adopt a cross-attention layer to F_{b} and Emb_{long} to generate a long-term context feature $F_{\text{lt}} \in \mathbb{R}^{T' \times H' \times W' \times C'}$ as:

$$F_{\text{lt}} = \text{CA}(F_{\text{b}}, \text{Emb}_{\text{long}}). \tag{6.11}$$

Finally, we query the action specific features from F_{lt} instead of F_{b} to get the final feature for classification using Eq. 6.9.

Action switch regression head. The T_{out} bounding boxes in a tubelet are simultaneously regressed with an FC layer as:

$$y_{\text{coor}} = \text{Linear}_{\text{b}}(F_{\text{tub}}). \tag{6.12}$$

where $y_{\text{coor}} \in \mathbb{R}^{N \times T_{\text{out}} \times 4}$, N is the number of action tubelets, and T_{out} is the temporal length of an action tubelet. To remove non-action boxes in a tubelet, we further include an FC layer for deciding whether a box prediction depicts the actor performing the action(s) of the tubelet, we call action switch. The action switch allows our method to generate action tubelets with a more precise temporal extent. The probabilities of the T_{out} predicted boxes in a tubelet being visible are:

$$y_{\text{switch}} = \text{Linear}_{\text{s}}(F_{\text{tub}}). \tag{6.13}$$

where $y_{\text{switch}} \in \mathbb{R}^{N \times T_{\text{out}}}$. For each predicted tubelet, each of its T_{out} bounding boxes obtain an action switch score.

6.3.4 Losses

The total loss is a linear combination of four losses:

$$\begin{aligned} \mathcal{L} = \lambda_1 \mathcal{L}_{\text{switch}}(y_{\text{switch}}, Y_{\text{switch}}) + \lambda_2 \mathcal{L}_{\text{class}}(y_{\text{class}}, Y_{\text{class}}) \\ + \lambda_3 \mathcal{L}_{\text{box}}(y_{\text{coor}}, Y_{\text{coor}}) + \lambda_4 \mathcal{L}_{\text{iou}}(y_{\text{coor}}, Y_{\text{coor}}). \end{aligned} \tag{6.14}$$

where y is the model output and Y denotes the ground truth. The action switch loss $\mathcal{L}_{\text{switch}}$ is a binary cross entropy loss. The classification loss $\mathcal{L}_{\text{class}}$ is a cross entropy loss. The $\mathcal{L}_{\text{box}}$ and $\mathcal{L}_{\text{iou}}$ denotes the per-frame bounding box matching error. It is noted when $T_{\text{out}} < T_{\text{in}}$, the tubelet is sparse and the coordinate ground truth Y_{coor} are from the corresponding temporally down-sampled frame sequence. We used the Hungarian matching similar to [20] and more details can be found in the supplementary. We empirically set the scale parameter as λ_1=1, λ_2=5, λ_3=2, λ_4=2.

6.4 Experiments

6.4.1 Experimental Setup

Datasets. We report experiments on three commonly used video datasets for action detection. **UCF101-24** [187] is a subset of UCF101. It contains 24 sport classes in 3207 untrimmed videos. Each video contains a single action class. Multiple instances of the same class can occur in a single video but with different spatial and temporal boundaries. We use the revised annotations for UCF101-24 from [184] and report the performance on split-1. **JHMDB51-21** [103] contains 21 action categories in 928 trimmed videos. We report the average results over all three splits. **AVA** [74] is larger-scale and includes 299 15-minute movies, 235 for training, and the remaining

	UCF101-24	*AVA*
single query	48.8	26.2
tubelet query set	52.9	27.4

(A) **Analysis on tubelet query.** Our tubelet query set design allows for each query to focus on the spatial location of the action on a specific frame.

	UCF101-24	*AVA*
self-attention	52.9	27.4
tubelet attention	53.8	27.7

(B) **Effect of tubelet attention.** With tubelet attention reasoning about the relations between bounding boxes within a tubelet and across tubelets improves.

	UCF101-24	*AVA*
w/o switch	53.8	27.7
w/ switch	57.7	27.7

(C) **Benefit of action switch.** Action switch produces a more precise temporal extent, which can only be shown by video-mAP.

	UCF101-24	*AVA*
FC head	57.8	23.4
+ short-term context	58.4	27.7
+ long-term context	-	28.8

(D) **Effectiveness of short- and long-term context.** The short-term context and long-term context helps with performance, more noticeable on AVA.

	UCF101-24	*AVA*
8	53.9	24.4
16	58.2	26.9
32	58.4	27.7

(E) **Length of input clip.** Longer input video leads to a better performance on both UCF101-24 and AVA.

w	# of clips	duration (s)	mAP
-	1	2.1	27.7
2	5	10.6	28.4
3	7	14.9	28.8
5	11	23.5	28.6

(F) **Long-term context length** analysis on AVA. The right amount of long-term context helps improve frame-mAP on AVA.

TABLE 6.1: Ablation studies on UCF101-24 and AVA 2.1. The proposed tubelet query, tubelet attention, the action switch and context-aware generally helps with the model performance. The proposed TubeR works well on long clips with shot changes. We report video-mAP@IoU=0.5 for UCF101-24 and frame-mAP@IoU=0.5 for AVA.

64 for validating. Box and label annotations are provided on per-second sampled keyframes. We evaluate on AVA with both annotation versions v2.1 and v2.2.
Evaluation criteria. We report the video-mAP at different IoUs on UCF101-24 and JHMDB51-21. As AVA only has keyframe annotations, we report frame-mAP@IoU=0.5 following [74] using a single, center-crop inference protocol.
Implementation details. We pre-train the backbone on Kinetics-400 [107]. The encoder and decoder contain 6 blocks on AVA. For the smaller UCF101-24 and JHMDB51-21, we reduce the numbers of blocks to 3 to avoid overfitting. We empirically set the number of tubelet query N equaling 15. During **training**, we use the bipartite matching [63] based on the Hungarian algorithm [120] between predictions and the ground truth. We use the AdamW [140] optimizer with an initial learning rate $1e{-}5$ for the backbone and $1e{-}4$ for the transformers. We decrease the learning rate $10\times$ when the validation loss saturates. We set $1e{-}4$ as the weight decay. Scale

jittering in the range of (288, 320) and color jittering are used for data augmentation. During **inference**, we always resize the short edge to 256 and use a single center-crop (1-view). We also tested the horizontal flip trick to create 2-view inference. For fair comparisons with previous methods on UCF101-24 and JHMDB51-21, we also test a two-stream setting with optical flow following [240].

6.4.2 Ablations

We perform our ablations on both UCF101-24 and AVA 2.1 to demonstrate the effectiveness of our designs on different evaluation protocols. Only RGB inputs are considered. For UCF101-24 with per-frame annotations, we report **video-mAP** at IoU=0.5. A standard backbone I3D-VGG [74] is utilized and the input length is set to 7 frames if not specified. For AVA 2.1 with 1-fps annotation, we only take the model prediction on keyframes and report **frame-mAP** at IoU=0.5. We use a CSN-50 backbone [203] with a single view evaluation protocol if not specified.

Benefit of tubelet queries. We first show the benefit of the proposed tubelet query sets. Each query set is composed of T_{out} per-frame query embeddings (see section 6.3.2), which predict the spatial location of the action on their respective frames. We compare this to using a single query embedding that represents a whole tubelet and must regress T_{out} box location for all frames in the clip. Our results are shown in Table 6.1a. Compared to using a single query embedding, our tubelets query set improves performance by +4.1% video mAP on UCF101-24, showing that modeling action detection as a sequence-to-sequence task effectively leverages the capabilities of transformer architectures.

Effect of tubelet attention. In Table 6.1b, we show using our tubelet attention module helps improve video-*mAP* on UCF101-24 by 0.9%, 0.3% on AVA. The tubelet attention saves about 10% memory (4,414 MB) than the typical self-attention implementation (5,026 MB) during training (16 frames input with batch size of 1).

Benefit of action switch. We report the effectiveness of our action switch head in Table 6.1c. On UCF101-24 the action switch increases the video-mAP from 53.8% to 57.7% by precisely determine the starting and ending point of actions temporally. Without action switch, TubeR mis-classifies transitional states as actions, like an example shown in Figure 6.3 (bottom row). As only the frame-level evaluation can be done on AVA, the advantage of the action switch is not shown by the frame-mAP. Instead, we demonstrate its effect in Figure 6.4 and Figure 6.5. The action switch produces tubelets with precise temporal extent for videos with shot changes.

Effect of short and long term context head. We report the impact of our context aware classification head with both short and long-term features in Table 6.1d. The context head brings a decent performance gain (+4.3%) on AVA. This is probably because the movie clips in AVA contain shot changes and so the network benefits from seeing the full context of the clip. On UCF101-24, the videos are usually short and without shot changes. The context does not bring a significant improvement on UCF101-24.

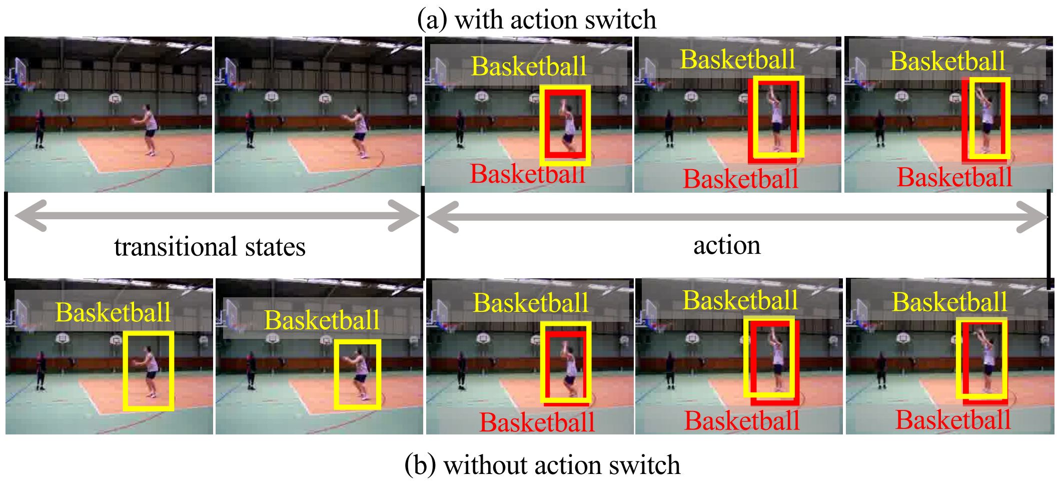


FIGURE 6.3: **Visualizations of action switch on UCF101-24**. Best view in color. The red box and label represent the ground truth. Yellow are for the detected tubelets. With the action switch (top row), TubeR avoids mis-classification for the transitional states.

Length of input clip. We report results with variable input lengths in Table 6.1e. We compare with input length of 8, 16 and 32 on both UCF101-24 and AVA with CSN-152 as backbone. TubeR is able to handle long video clips as expected. We notice that our performance on UCF101-24 saturates faster than on AVA, probably because the UCF101-24 does not contain shot changes that requires longer temporal context for classification.

Length of long-term context. This ablation is only conducted on AVA as videos on UCF101-24 are too short to use long-term context. (Table 6.1f). It shows that the right amount of long-term context helps performance, but overwhelming the amount of long-term context harms performance. This is probably because the long-term feature contains both useful information and noise. The experiments show that about 15s context serves best. Note that the context length varies for datasets, but can be easily determined empirically.

6.4.3 Frame-level State-of-the-Art

AVA 2.1 Comparison. We first compare our results with previously proposed methods on AVA 2.1 in Table 6.2. Compared to previous end-to-end models, with comparable backbone (I3D-Res50) and the same inference protocol, the proposed TubeR outperforms all. TubeR outperforms the most recent end-to-end works WOO [25] by 0.9% and VTr [68] by 1.2%. This demonstrates the effectiveness of our designs.

Compared to previous work using an offline person detector, the proposed TubeR is also more effective under the same inference protocols. This is because TubeR generates tubelet-specific features without assumptions on location, while the two-stage methods have to assume the actions occur at a fixed location. It is also worth mentioning that the TubeR with CSN backbones outperforms the two-stage model

Model	**Detector**	**Input**	**Backbone**	**Pre-train**	**Inference**	**GFLOPs**	**mAP**
Comparison to end-to-end models							
I3D [74]	✗	32 × 2	I3D-VGG	K400	1 view	NA	14.5
ACRN [191]	✗	32 × 2	S3D-G	K400	1 view	NA	17.4
STEP [224]	✗	32 × 2	I3D-VGG	K400	1 view	NA	18.6
VTr [68]	✗	64 × 1	I3D-VGG	K400	1 view	NA	24.9
WOO [25]	✗	8 × 8	SF-50	K400	1 view	142	25.2
TubeR	✗	16 × 4	I3D-Res50	K400	1 view	132	**26.1**
TubeR	✗	16 × 4	I3D-Res101	K400	1 view	246	**28.6**
Comparison to two-stage models							
Slowfast-50 [55]	F-RCNN	16 × 4	SF-50	K400	1 view	308	24.2
X3D-XL [53]	F-RCNN	16 × 5	X3D-XL	k400	1 view	290	26.1
CSN-152*	F-RCNN	32 × 2	CSN-152	IG + K400	1 views	342	27.3
LFB [219]	F-RCNN	32 × 2	I3D-101-NL	k400	18 views	NA	27.7
ACAR-NET [152]	F-RCNN	32 × 2	SF-50	K400	6 views	NA	28.3
TubeR	✗	32 × 2	CSN-50	K400	1 view	78	**28.8**
TubeR	✗	32 × 2	CSN-152	IG + K400	1 view	120	**31.7**
Comparison to best reported results							
WOO [25]	✗	8 × 8	SF-101	K400+K600	1 view	246	28.0
SF-101-NL [55]	F-RCNN	32 × 2	SF-101+NL	K400+K600	6 views	962	28.2
ACAR-NET [152]	F-RCNN	32 × 2	SF-101	K400+K600	6 views	NA	30.0
AIA [197]	F-RCNN	32 × 2	SF-101	K400+K700	18 views	NA	31.2
TubeR	✗	32 × 2	CSN-152	IG + K400	2 view	240	**32.0**

TABLE 6.2: **Comparison on AVA v2.1** validation set. Detector shows if additional detector is required; * denotes the results we tested. IG denotes the IG-65M dataset, SF denotes the slowfast network. The FLOPs for two-stage models are the sum of Faster RCNN-R101-FPN FLOPs (246 GFLOPs [20]) plus classifier FLOPs multiplied by view number. TubeR performs more effectively and efficiently.

with the same backbone by +4.4%, demonstrating that the gain is not from the backbone but our TubeR design. TubeR even outperforms the methods with multi-view augmentations (horizontal flip, multiple spatio crop and multi-scale). TubeR is also significantly faster than previous models, we have attempted to collect the reported FLOPs from previous works (Table 6.2). Our TubeR has 8% fewer FLOPs than the most recently published end-to-end model [25] with higher accuracy. Tuber is also 4× more efficient than the two-stage model [55] with noticeable better performance. Thanks to our sequence-to-sequence design, the heavy backbone is shared and we do not need temporal iteration for tubelet regression.

We finally present the highest number reported in the literature, regardless of the inference protocol, pre-training dataset and additional information used. TubeR still achieves the best performance, even better than the model using additional

Model	**backbone**	**pre-train**	**inference**	**mAP**
Single-view				
X3D-XL [53]	X3D-XL	K600+ K400	1 view	27.4
CSN-152 [238]	CSN-152	IG + K400	1 view	27.9
WOO [25]	SF-101	K600+ K400	1 view	28.3
M-ViT-B-24 [51]	MViT-B-24	K600+ K400	1 view	28.7
TubeR	CSN-152	IG + K400	1 view	33.4
Multi-view				
SlowFast-101 [55]	SF-101	K600+ K400	6 views	29.8
ACAR-Net [152]	SF-101	K700+ K400	6 views	33.3
AIA (obj) [197]	SF-101	K700+ K400	18 views	32.2
TubeR	CSN-152	IG + K400	2 views	**33.6**

TABLE 6.3: **Comparison on AVA v2.2** validation set. IG denotes the IG-65M dataset, SF denotes the slowfast network. TubeR achieves the best result.

object bounding-boxes as input [197].The results show that the proposed sequence-to-sequence model with tubelet specific feature is a promising direction for action detection.

AVA 2.2 Comparison. The results are shown in Table 6.3. Under the same single-view protocol, TubeR is significantly better than previous methods, including the most recent work with an end-to-end design (WOO [25] +5.1%) and the two-stage work with strong backbones (MViT [51] +4.7%). A fair comparison between TubeR and a two-stage model [238] with the same backbone CSN-152, shows TubeR gains +5.5% frame-mAP. It demonstrates TubeR's superior performance comes from our design rather than the backbone.

UCF101-24 Comparison. We also compare TubeR with the state-of-the-art using frame-mAP@IoU=0.5 on the UCF101-24 dataset (see the first column with numbers in Table 6.4). Compared to existing methods, TubeR acquires better results with comparable backbones, for both RGB-stream and two-stream settings. Further with a CSN-152 backbone, TubeR get 83.2 frame-mAP, even better than two-stream methods. Though TubeR targets on tubelet-level detection, it performs well on frame-level evaluation on both AVA and UCF101-24.

6.4.4 Video-level State-of-the-Art

6.4.5 Visualization

We also compare TubeR with various settings to state-of-the-art reporting video-mAP on UCF101-24 and JHMDB51-21 in Table 6.4. TubeR acquires better results with and without optical flow as inputs. For fair comparisons, TubeR with a 2D backbone gains +4.4% video-mAP@IoU=0.5 compared to the recent start-of-the-art [131] on UCF101-24 without using optical flow, which demonstrates that TubeR learning tubelet queries is more effective compared to using positional hypotheses. Compared

		UCF101-24				*JHMDB51-21*	
	Backbone	f-mAP	0.20	0.50	0.50:0.95	0.20	0.50
RGB-stream							
MOC [131]	DLA34	72.1	78.2	50.7	26.2	-	-
TubeR	Res50	79.5	81.2	55.1	28.1	-	-
T-CNN [88]	C3D	41.4	47.1	-	-	78.4	76.9
TubeR	I3D	80.1	**82.8**	**57.7**	**28.6**	**79.7**	**78.3**
TubeR	CSN-152	**83.2**	**83.3**	**58.4**	**28.9**	-	-
Two-stream							
TacNet [185]	VGG	72.1	77.5	52.9	24.1	-	-
2in1 [240]	VGG		78.5	50.3	24.5	-	74.7
ACT [106]	VGG	67.1	77.2	51.4	25.0	74.2	73.7
MOC [131]	DLA34	78.0	82.8	53.8	28.3	77.3	77.2
STEP [224]	I3D	75.0	76.6	-	-	-	-
I3D [74]	I3D	76.3	-	59.9	-	-	78.6
TubeR	I3D	**81.3**	**85.3**	**60.2**	**29.7**	**81.8**	**80.7**

TABLE 6.4: **Comparison on UCF101-24 and JHMDB51-21** with video-*mAP*. TubeR achieves better results under both settings with and without optical flow inputs. f-mAP denotes the frame mAP@IoU=0.5.

to TacNet [185] which proposes a transition-aware context network to distinguish transitional states, TubeR with action switch performs better even with a one-stream setting. We further conduct an experiment with CSN-152 backbone. The results show that the TubeR generalizes well by using different backbones. When incorporating optical flow inputs, the TubeR further boosts the video-level results and achieves the best among the two-stream approaches. The result illustrates our design is effective for video-level action detection.

We first provide visualizations (Figure 6.4) of the tubelet-specific features by overlaying the tubelet-specific feature activation over the input frames using attention rollout [1]. The example in Figure 6.4 is challenging as it contains multiple people and concurrent actions. The visualization show that: 1. Our proposed TubeR is able to generate highly discriminative tubelet-specific features. Different actions in this case are clearly separated in different tubelets. 2. Our action switch works as expected and initiates/cuts the tubelets when the action starts/stops. 3. Our TubeR generalizes well to scale changes (the brown tubelet). 4. The generated tubelets are tightly associated with tubelet specific feature as expected.

We further show our TubeR performs well in various scenarios. TubeR works well on videos with shot changes (Figure 6.5 top); TubeR is able to detect an actor moving with distance (Figure 6.5 middle); and TubeR is robust to action detection even for small people (Figure 6.5 bottom).

6.5 Discussion and Conclusion

Limitations. Although proposed for long videos, we noticed two potential limitations that stop us from feeding in very long videos in one shot.
1. We observe that 90% of computation (FLOPs) and 67% of memory usage was used by our 3D backbone. This heavy backbone restricts us from applying TubeR on long videos. Recent works show that transformer encoders can be used for video embedding [239, 51, 4] and are less memory and computationally hungry. We will explore these transformer based embeddings in future work.
2. If we were to process a long video in one pass we'd need enough queries to cover the maximum number of different actions per-person in that video. This would likely require a large number of queries which would cause memeory issues in our self attention layers. A possible solution is to generate person tubelets, instead of action tubelets, so that we do not need to split tubelets when a new action happens. Then we would only need a query for each person instance.
Potential negative impact. There are real-world applications of action detection technology such as patient or elderly health monitoring, public safety, Augmented/Virtual Reality, and collaborative robots. However, there could be unintended usages and we advocate responsible usage and complying with applicable laws and regulations.
Conclusion. This chapter introduces TubeR, a unified solution for spatio-temporal video action detection in a sequence-to-sequence manner. Our design of tubelet-specific features allows TubeR to generate tubelets (a set of linked bounding boxes) with action predictions for each of the tubelets. TubeR does not rely on positional hypotheses and therefore scales well to longer video clips. TubeR achieves state-of-the-art performance and better efficiency compared with previous works.

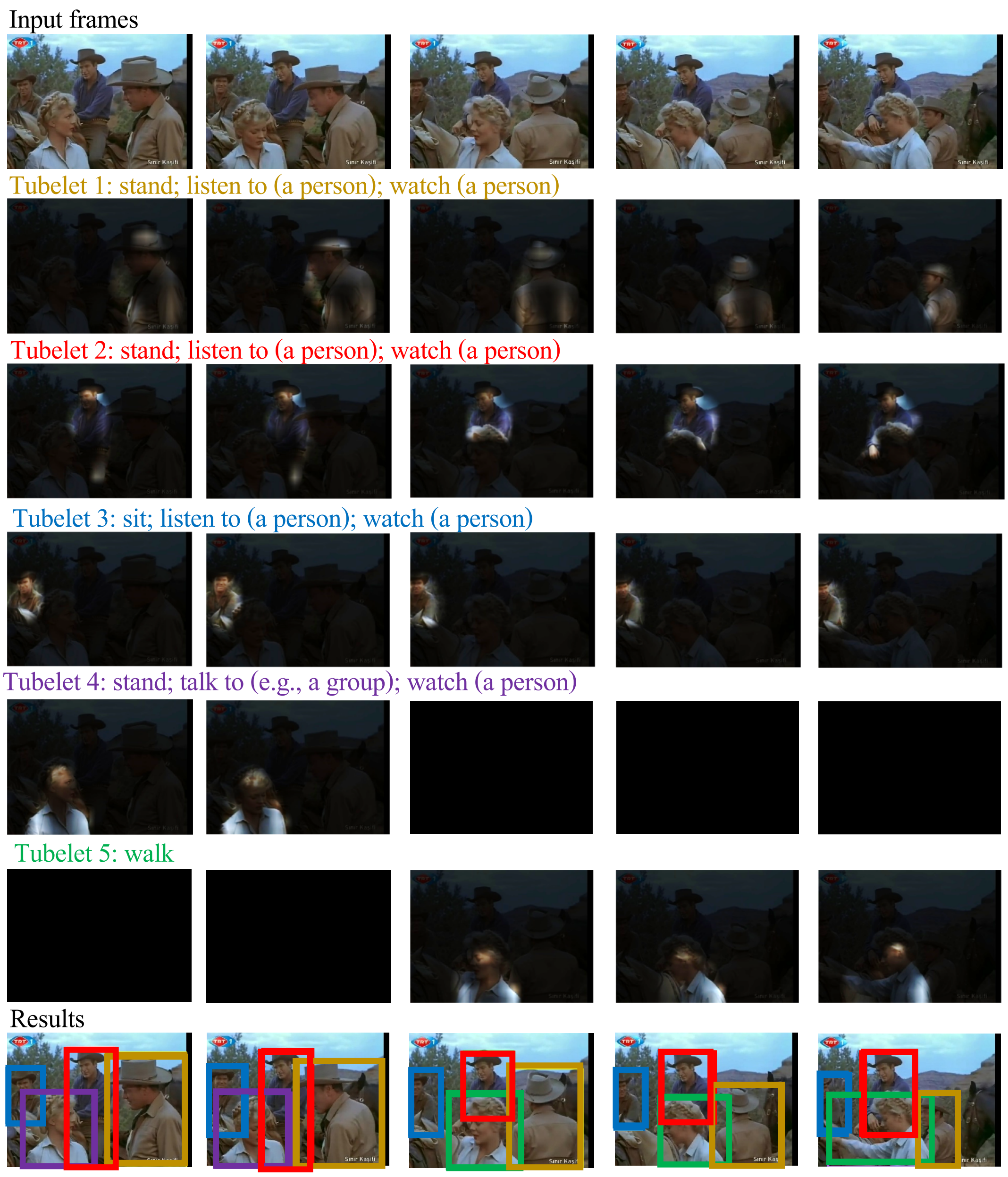

Input frames
Tubelet 1: stand; listen to (a person); watch (a person)
Tubelet 2: stand; listen to (a person); watch (a person)
Tubelet 3: sit; listen to (a person); watch (a person)
Tubelet 4: stand; talk to (e.g., a group); watch (a person)
Tubelet 5: walk
Results

FIGURE 6.4: **Visualization of tubelet specific feature with attention rollout**. Best view in color. Each tubelet covers a separated action instance.

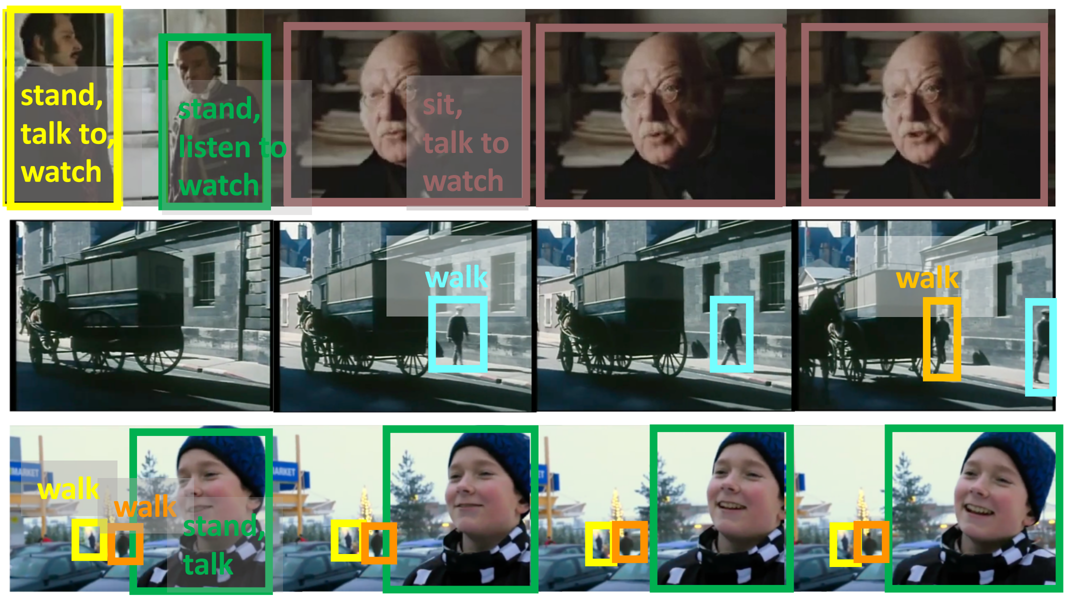


FIGURE 6.5: **Results visualization**. Best view in color. We use different colors to label different tubelets. Each action tubelet contains its action labels and boxes on each frame. We only show the action labels on the first frame of an action tubelet. Some challenging cases are shown. Top: shot changes; Middle: actors moving with distance; Bottom: multiple actors with small and large scales.

Appendix A

Supplementary Materials for LiftPool

We show additional analysis and results for robustness and semantic segmentation in this Appendix.

LiftDownPool vs. MaxPool We provide a schematic diagram in Figure A.1 to further illustrate the difference between MaxPool and LiftDownPool, MaxUpPool and LiftUpPool. Taking kernel size 2, stride 2 as an example, MaxPool selects the maximum activations in a local neighbourhood. Hence, it looses 75% information. The lost details could be important for image recognition. LiftDownPool decomposes a feature map into LL, LH, HL and HH. LL containing low-pass coefficients is an approximation of the input. It is designed for capturing correlated structures of the input. Other sub-bands contain detail coefficients along different directions. The pooling is implemented by summing up all the sub-bands. The final pooled result containing both the approximation and details is expected to be more effective for image classification.

LiftUpPool vs. MaxUpPool The pooling function in MaxPool is not invertible. MaxPool records the maximum indices for performing the corresponding MaxUpPool. MaxUpPool takes the activations at the corresponding positions for the recorded maximum indices on the output. For other indices, there will be zeros. The final up-sampled output has many ‘holes’. By contrast, the pooling functions in LiftDownPool are invertible. Leveraging the property by performing a LiftDownPool backwards, LiftUpPool is able to generate a refined output from an input, including the recorded details.

Experiment Settings The VGG13 [182] network trained on CIFAR-100 is optimized by SGD with a batch size of 100, weight decay of 0.0005, momentum of 0.9. The learning rate starts from 0.1 and is reduced by multiplying 0.1 after 80 and 120 epochs for a total of 160 epochs. We train ResNets for 100 epochs and MobileNet for 150 epochs on ImageNet, following the standard training recipe from the public PyTorch [153] repository.

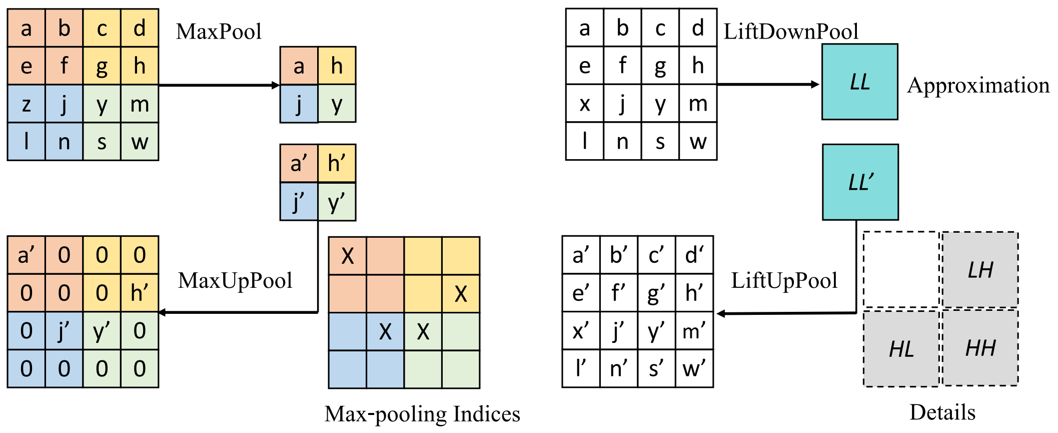


FIGURE A.1: **Comparisons between MaxPool and LiftDownPool, MaxUpPool and LiftUpPool.** MaxPool looses details. With the recorded maximum indices, MaxUpPool generates a very sparse output. LiftDownPool decomposes the input into an approximation and several details sub-bands. It realizes a pooling by summing up all sub-bands. LiftUpPool produces a refined output by performing LiftDownPool backwards.

High-resolution Feature Maps Visualization By using ResNet50 with input size 224×224, we extract the feature maps of an image from the first pooling layer. We show the high-resolution feature maps in Figure A.2. We only show the LL sub-band from LiftDownPool. Compared to MaxPool, LiftDownPool better maintains the local structure.

Anti-aliasing LiftDownPool effectively reduces aliasing following the Lifting Scheme [195] compared to naive downsizing. Figure A.3(b) provides a simple illustration of LiftDownPool. The dashed line is an original signal x. According to Eq 3.1,

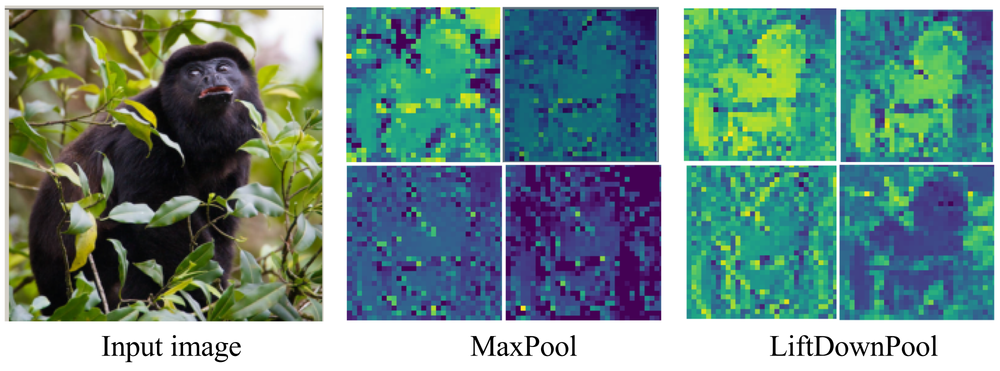
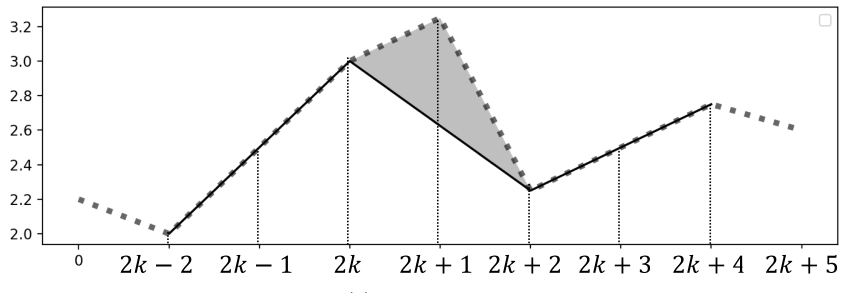
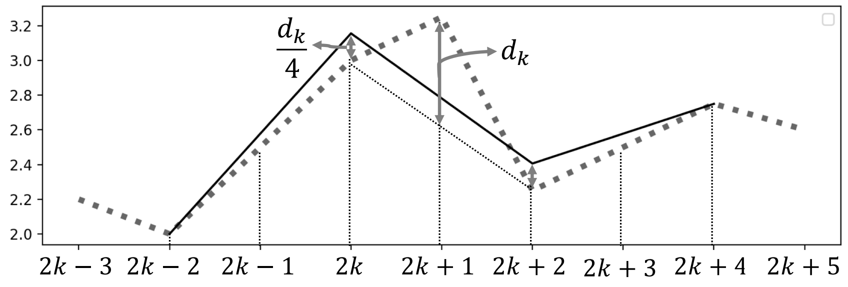


FIGURE A.2: **High-resolution feature maps visualization.** LiftDownPool better maintains local structure.



(a) Downsizing



(b) LiftDownPool

FIGURE A.3: **Illustration how LiftDownPool reduces aliasing compared to downsizing** [195]. Dashed line means original signal. (a) solid line is after downsizing. (b) solid line is after LiftDownPool. The solid and dashed lines cover the same area in (b).

the predictor $\mathcal{P}(\cdot)$ for the odd part $\boldsymbol{x}_{2k+1}$ could easily take the average of its two even neighbors:

$$\boldsymbol{d}_k = \boldsymbol{x}_{2k+1} - (\boldsymbol{x}_{2k} + \boldsymbol{x}_{2k+2})/2 \tag{A.1}$$

Thus, if $\boldsymbol{x}$ is linear in a local area, the detail $\boldsymbol{d}_k$ is zero. The prediction step takes care of some of the spatial correlation. If an approximation $\boldsymbol{s}$ of the original signal $\boldsymbol{x}$ is simply taken from the even part $\boldsymbol{x}^e$, it is really downsizing the signal shaped in the red line. There is serious aliasing. The running average of $\boldsymbol{x}^e$ is not the same as that of the original signal $\boldsymbol{x}$. The updater $\mathcal{U}(\cdot)$ in Eq 3.3 corrects this by replacing $\boldsymbol{x}^e$ with smoothed values $\boldsymbol{s}$. Specifically, $\mathcal{U}(\cdot)$ restores the correct running average and thus reduces aliasing:

$$\boldsymbol{s}_k = \boldsymbol{x}_{2k} + (\boldsymbol{d}_{k-1} + \boldsymbol{d}_k)/4 \tag{A.2}$$

As shown in Figure A.3, $\boldsymbol{d}_k$ is the difference between the odd sample $\boldsymbol{x}_{2k+1}$ and the average of two even samples. This causes a loss $\boldsymbol{d}_k/2$ in the area with the red shade. To preserve the running average, this area is redistributed to the two neighbouring even samples $\boldsymbol{x}_{2k}$ and $\boldsymbol{x}_{2k+2}$, which shapes a coarser piecewise linear signal $\boldsymbol{s}$ in the solid line. The signal after LiftDownPool, drawn as solid line, covers the same area with the original signal dashed line. LiftDownPool reduces aliasing compared to the downsizing drawn in the solid line in (a).

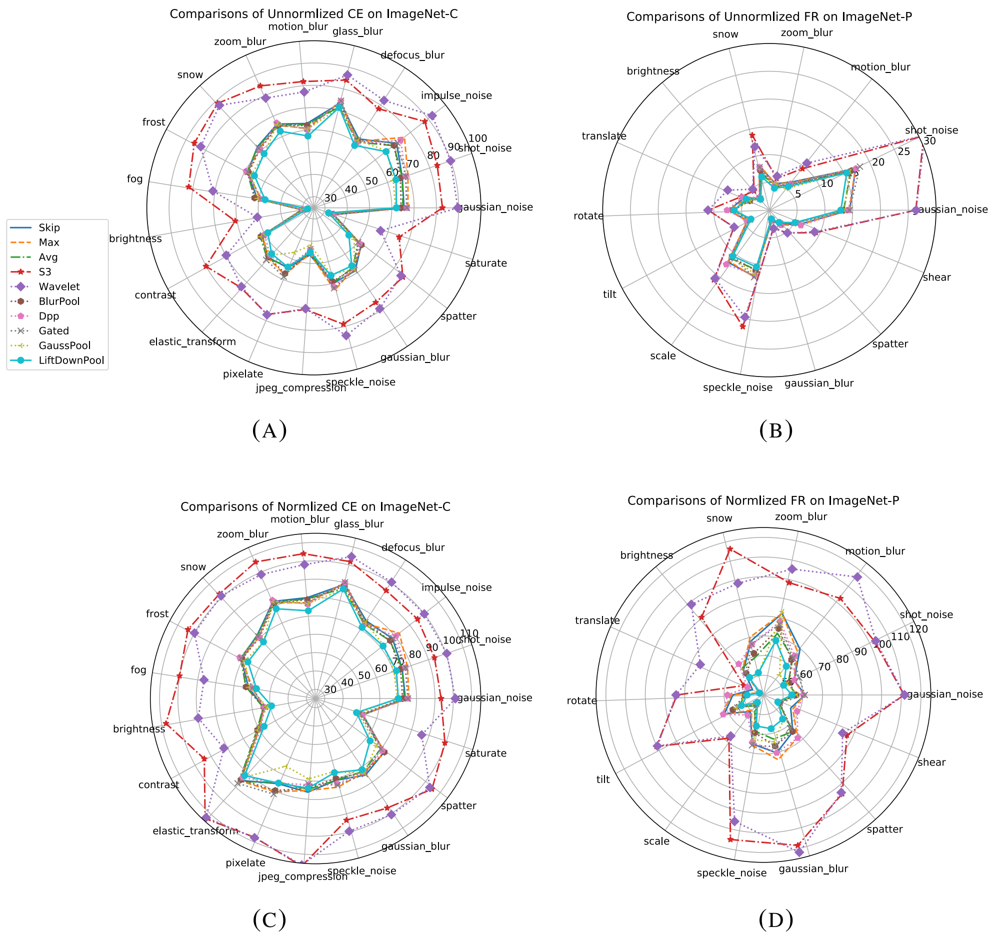


FIGURE A.4: **Comparisons between the robustness of various pooling methods to per kind of corruption on ImageNet-C and perturbation on ImageNet-P.** LiftDownPool presents stronger robustness to almost all the corruptions and perturbations.

Out-of-distribution Robustness We show the robustness of pooling methods for each corruption and perturbation type in Figure A.4. Corruption Error (*CE*) is the metric of the robustness to corruptions on ImageNet-C. And Flip Rate (*FR*) is reported for the robustness to perturbation on ImageNet-P. Following [84], we report both unnormalized raw values and normalized values by AlexNet's *CE* and *FR*. Lower values are better. As seen in Figure A.4(a) and (c), LiftDownPool gets the lowest *CE* for most of the "high frequency" corruptions including gaussian noise and spatter, as well as the "low frequency" corruptions such as motion blur, zoom blur. In Figure A.4(b) and (d), it clearly shows LiftDownPool has less sensitivity to most of the perturbations such as speckle noise and gaussian blur.

Visualization of Up-pooling In Figure A.5, we show the feature map for each predicted category from the last layer of SegNet using varying up-pooling methods.

Using MaxUpPool, the feature maps look noisy and less continuous due to the fact that MaxUpPool generates the output with many ‘zeros’, where there is no information. By applying a BlurPool following the MaxUpPool, the feature maps turn more smooth, while still with less details. LiftUpPool, benefiting from the recorded details during LiftDownPool, produces finer feature maps for each category. It has smooth edges, continuous segmentation maps and less aliasing.

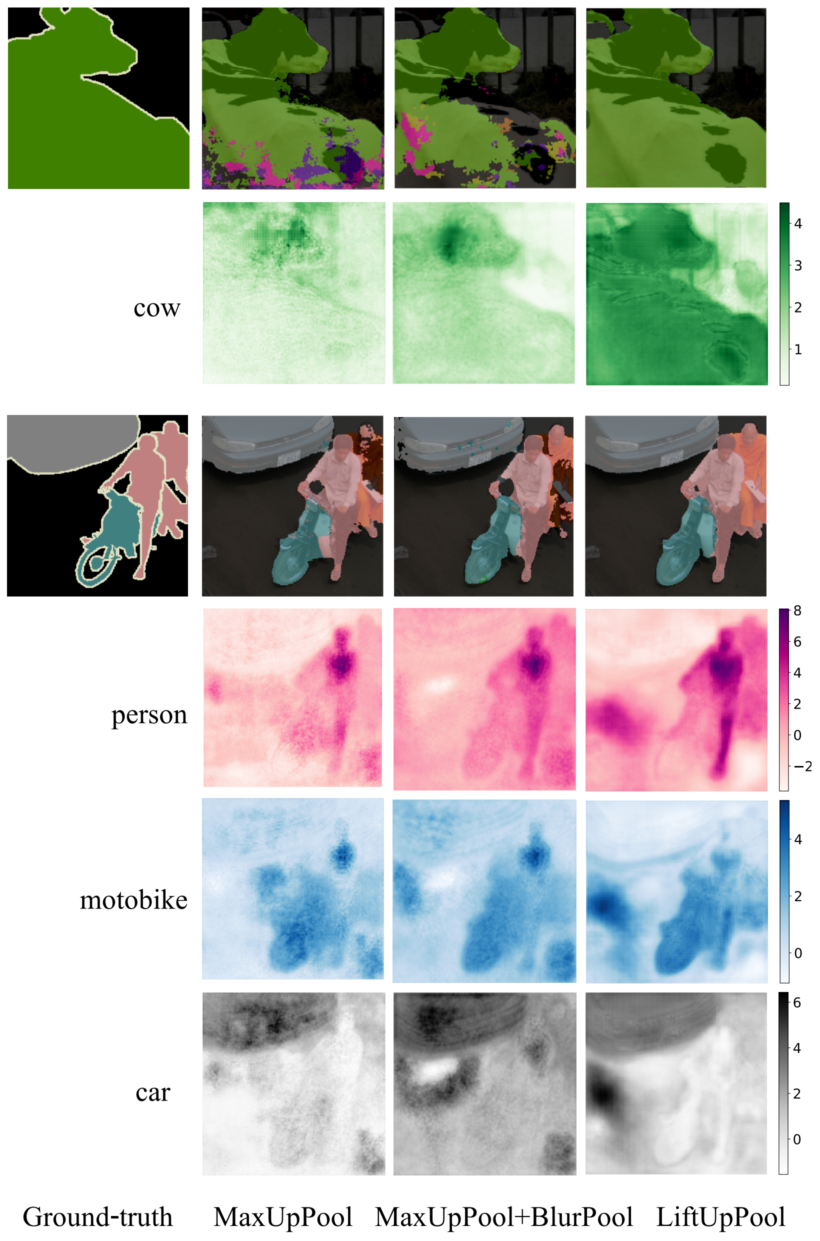


FIGURE A.5: **Visualization of feature maps per-predicted-category from the last layer of SegNet.** Lift-UpPool generates more precise predictions for each category.

Appendix B

Supplementary Materials for Dance with Flow

B.1 Extra Experiments and Results

All the results in this supplementary material are from a single-frame network trained on *UCF101-24*. We report mAP at the high IoU threshold of 0.5:0.95. BroxFlow is applied here.

Influence of Fusion Performance of different fusion methods are seen in Table C.1. Besides mean fusion, we use another two fusion methods to fuse RGB-stream and flow-stream following [184]. To conduct boost fusion, we perform L-1 normalization on the detection boxes' scores after fusion and then retain any flow detection boxes for which an associated appearance based box was not found. The other way is retaining the union $\{b_i^a\} \cup \{b_j^f\}$ of the two sets of RGB-stream $\{b_i^a\}$ and flow-stream $\{b_j^f\}$ detection boxes, respectively. All of the three fusion methods applied to two stream help to improve results of single stream. Mean fusion is the best way to fuse RGB- and flow- stream. Our two-in-one stream beats all of them with only half runtime and model size.

Quantitative Results per Action We report deteciton results per action from *UCF101-24* in Table B.2. For some challenging cases such as basketball dunk, cricket bowling, pole vault and volleyball spiking, which have crowded and cluttered backgrounds, our two-in-one stream achieves better results than other methods. For instance two-in-one stream outperforms two-stream by 6% and RGB-stream by 9% for pole vault. For the cases where multiple instances may occur, such as fencing, ice dancing and salsa spin, our two-in-one stream also boosts the accuracy. Notably, fusing RGB and optical flow, improves results in most cases except for skiing. Both two-stream and two-in-one stream perform worse than the RGB-stream for skiing. The flow images are very noisy and even make results worse. Overall, the proposed two-in-one stream outperforms alternatives for 16 out of 24 action classes.

Qualitative Results per Action Some successful detected results of challenging cases using our two-in-one stream are visualized in Figure B.1 and Figure B.2. The green boxes represent the ground truth boxes. The yellow boxes with the labels are detected boxes with the classification scores.

	Accuracy	**Efficiency**	
		sec/frame	parameters (MB)
flow-stream	11.60	0.04	102.32
RGB-stream	18.49	0.04	102.32
two-stream (boost-fusion)	18.97	0.09	204.64
two-stream (union-set)	19.42	0.09	204.64
two-stream (mean-fusion)	19.79	0.09	204.64
two-in-one stream	**21.51**	0.04	102.44

TABLE B.1: **Influence of Fusion** Performance ($mAP@IoU =$ 0.5:0.95, runtime and model size) comparison on *UCF101-24*. Three different fusion methods are used in two-stream method. Mean fusion achieves better results than the two others. Our two-in-one stream outperforms all of them.

Basketball dunk is difficult as there are many interfering actors. An RGB-stream cannot detect any actions for the scenes shown in the Figure B.1. For cliff diving, when the actor reaches the surface of the water, the action is mistaken as surfing due to the sea context captured by the RGB-stream. The RGB-stream model may pay more attention to backgrounds. However, our two-in-one stream using flow condition to modulate RGB features, focuses more on actions and improves the results. For pole vault, it is easily mistaken as cliff diving when the actor falls down from up using the RGB stream. Our two-in-one strem performs better. In Figure B.2, we also show some multi-instance cases such as ice dancing and salsa spin. In these cases only one ground-truth box is given for each image. However, it is reasonable that our two-in-one stream is capable to detect multiple instances. Thus, the results are actually better than the AP values of these cases shown in Table B.2. It is worth to be mentioned that the detection boxes of our two-in-one stream have high overlap with the ground-truth boxes.

Failure Cases We show three kinds of failure cases in Figure B.3. It is difficult to define whether the frames at the bound of an action are action or not. In the first row, these frames follow an action tennis swing. The model still takes them as tennis swing. Without considering ground-truth, we think it is reasonable. In the second row, when the actor appears blurry, our model still gives correct detection in the last three frames. However, there are no actions in the ground-truth for these frames. In the third row, the model successfully locates the actors, but assigns the wrong action label. The real action is pole vault, which has a similar run-up with floor gymnastics in the beginning of the action.

video-mAP(%)	mAP	Basketball	BasketballDunk	Biking	CliffDiving	CricketBowling	Diving	Fencing	FloorGymnastics
flow-stream	11.60	0.03	0.06	9.09	2.57	0.11	1.40	33.21	31.79
RGB-stream	18.49	0.00	0.27	23.25	8.15	0.38	5.90	46.90	45.30
two-stream	19.79	0.01	0.12	**24.81**	**11.04**	0.20	4.95	45.03	43.45
two-in-one stream	**21.51**	**0.17**	**1.16**	23.13	9.13	**0.89**	**7.30**	**53.70**	**54.31**
		GolfSwing	HorseRiding	IceDancing	LongJump	PoleVault	RopeClimbing	SalsaSpin	SkateBoarding
flow-stream		4.64	35.69	15.63	14.75	2.81	23.69	0.34	28.17
RGB-stream		10.45	42.35	17.11	20.70	4.10	31.00	0.75	41.10
two-stream		12.38	40.77	16.87	**25.53**	7.94	36.46	0.44	44.35
two-in-one stream		**14.37**	**45.30**	**19.00**	23.44	**13.70**	**46.40**	**1.70**	**46.40**
		Skiing	Skijet	SoccerJuggling	Surfing	TennisSwing	TrampolineJumping	VolleyballSpiking	WalkingWithDog
flow-stream		6.08	2.84	44.29	4.13	0.20	5.14	0.00	11.71
RGB-stream		**37.03**	26.66	25.41	18.12	0.15	5.72	0.00	26.85
two-stream		34.25	**28.84**	**45.98**	16.02	0.17	**6.60**	0.00	**28.61**
two-in-one stream		35.57	28.11	41.02	**18.56**	**0.46**	5.65	**0.06**	26.80

TABLE B.2: Per action class video $AP@IoU = 0.5{:}0.95$ on *UCF101-24*. Our two-in-one stream achieves better results for 16 action classes. Especially for diving, fencing, floor gymnastics, horse riding, ice dancing, pole vault and rope climbing, there are obvious improvements. For fencing, ice dancing and salsa spin, where multiple instances may occur, two-in-one stream also boosts the accuracy. For skiing, both two-stream and two-in-one stream are lower than RGB-stream in accuracy. But our two-in-one stream is still better than two stream in this case.

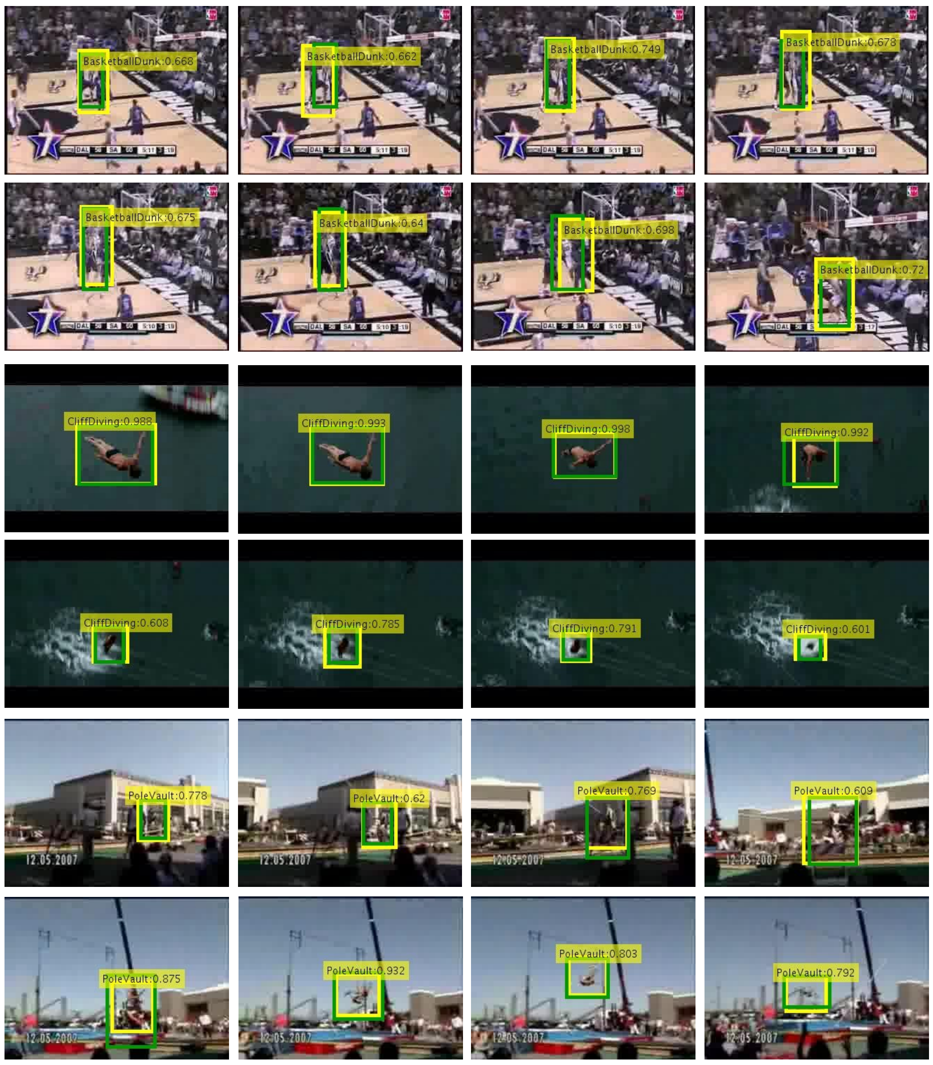


FIGURE B.1: Examples of successful detected results of some challenging cases using our two-in-one stream. The green boxes represent ground-truth boxes. The yellow boxes with labels mean detection boxes with classification scores. Our two-in-one stream performs well at high IoU thresholds.

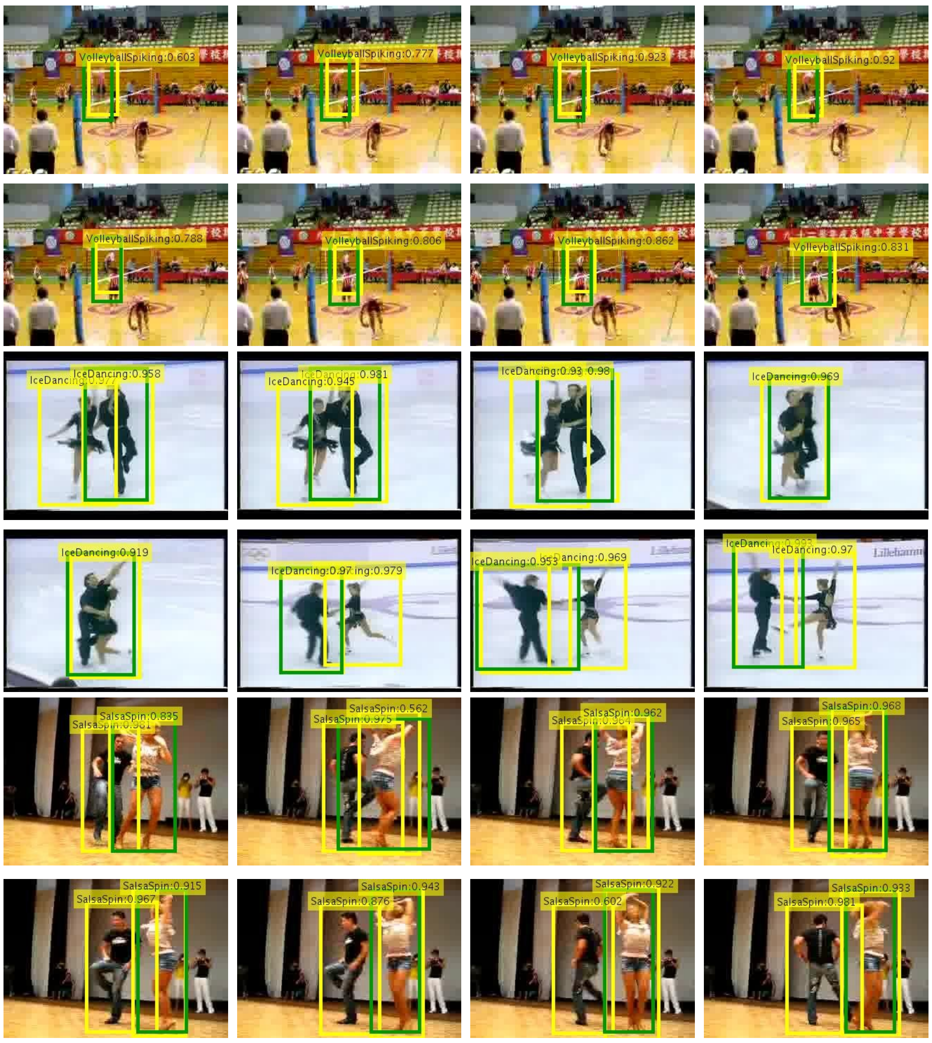


FIGURE B.2: Examples of successful detected results. Our two-in-one stream is able to detect multiple instances for ice dancing and salsa spin. It is reasonable even only one actor is labeled in the ground-truth.

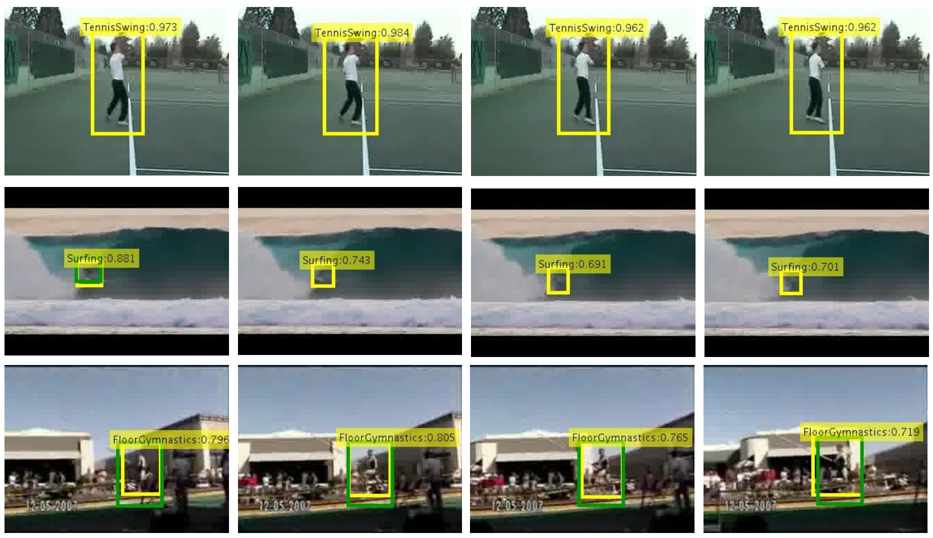


FIGURE B.3: Failure cases. It is reasonable to assign tennis swing to these actions in first row even there is no action for these frames in the ground-truth. In the second row, our model still gives correct detections for the blurry actions even no actions are labeled in the ground-truth. In the last row, our model successfully locates the actors. But pole vault is mistaken as floor gymnastics as the two actions have the similar run-up in the beginning.

Appendix C

Supplementary Materials for Go with the Flow

C.1 Action Classification

Backbone. For fast training, we use Resnet18-3D as backbone for the ablation study. We suggest to plug the aligned 3D convolution into the layer res2. The network structure is shown in Table C.1.

Applying the aligned convolution to multiple layers. In the ablation study, we plug the patch-to-point aligned 3D convolution into different individual layers. We find it is better to apply the aligned 3D convolution to the bottom layer. Besides, we also add it to multiple residual layers. We use input length 32. There are modest improvements when applying the patch-to-point aligned convolution to multiple layers compared to a single layer. The results are shown in Table C.2. Thus, we prefer to utilize the aligned 3D convolution in the single res2 layer.

C.2 Moving Object Segmentation.

Network Architecture. We extend a Deeplab [23] model to a 3D convolution network for moving object segmentation in videos. An encoder-decoder structure shown in Figure C.1 is adopted. An input clip includes eight successive frames from a video. We take a light network Resnet18 as the encoder. A pretrained model for action classification on Kinetics400 is used for initializing the weights. We apply Atrous Spatial Pyramid Pooling for multiple scale representations before the decoder. In order to generate segmentation maps with the same spatial and temporal dimensions per input clip, 3D upsampling is utilized in the decoder. To verify the effectiveness of the aligned 3D convolution for moving object segmentation, we only plug the patch-to-point aligned convolution into the residual layers in the encoder.

	layer	output size
conv_1	$7 \times 7 \times 7$, 64, stride 1,2,2	$16 \times 56 \times 56$
pool_1	$3 \times 3 \times 3$, max, stride 2,2,2	$8 \times 28 \times 28$
res_2	$\begin{bmatrix} 3 \times 3 \times 3, 64 \\ 3 \times 3 \times 3, 64 \end{bmatrix} \times 2$	$8 \times 28 \times 28$
res_3	$\begin{bmatrix} 3 \times 3 \times 3, 128 \\ 3 \times 3 \times 3, 128 \end{bmatrix} \times 2$	$4 \times 14 \times 14$
res_4	$\begin{bmatrix} 3 \times 3 \times 3, 256 \\ 3 \times 3 \times 3, 256 \end{bmatrix} \times 2$	$2 \times 7 \times 7$
res_5	$\begin{bmatrix} 3 \times 3 \times 3, 512 \\ 3 \times 3 \times 3, 512 \end{bmatrix} \times 2$	$1 \times 4 \times 4$
global average pool, fc		$1 \times 1 \times 1$

TABLE C.1: **Resnet18-3D baseline model** used in our ablation studies. The dimensions of 3D output maps and filter kernels are in $T \times H \times W$, with the number of channels following. The input size $16 \times 112 \times 112$ is taken as an example. The details of a Residual block is shown in brackets.

	top-1	*top-5*
3D baseline	48.57	71.87
patch-to-point		
res2	52.25	74.17
res2, res3	52.38	75.67
res2, res3, res4	52.70	75.07

TABLE C.2: **Applying the patch-to-point aligned convolution to multiple layers** achieves slight improvements compared to a single layer. All aligned convolution models outperform the 3D baseline on UCF101.

Implementation. We conduct the experiments on Densely Annotated Video Segmentation 2016 (DAVIS2016) [155]. There are 50 videos with 3455 annotated frames, including 30 videos for training and 20 videos for test. Following the previous works [89], we use 480p videos with a resolution of 854×480 pixels. For training and test, we resize the frames to 530×300. The training leverages the Adam optimizer with initial learning rate 0.001.

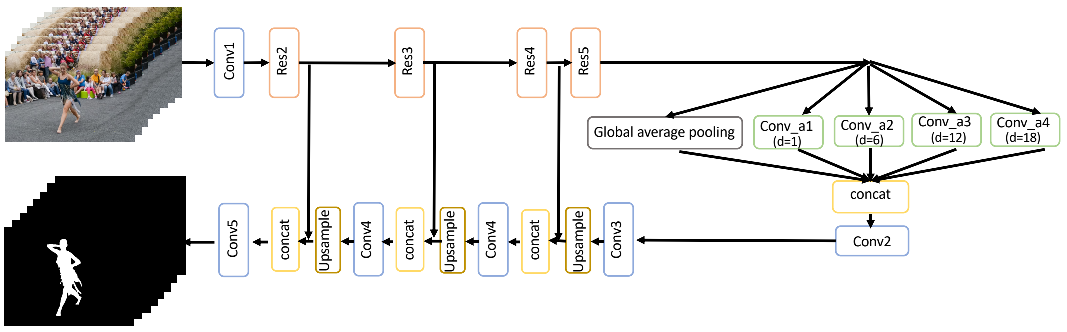


FIGURE C.1: **Network architecture for moving object segmentation.** We use Resnet18-3D as the encoder. For multiple scale representations, Atrous Spatial Pyramid Pooling is applied. *d* means the dilation rate for the atrous convolution. In the decoder, the up-sampling layer is a tri-linear interpolation layer.

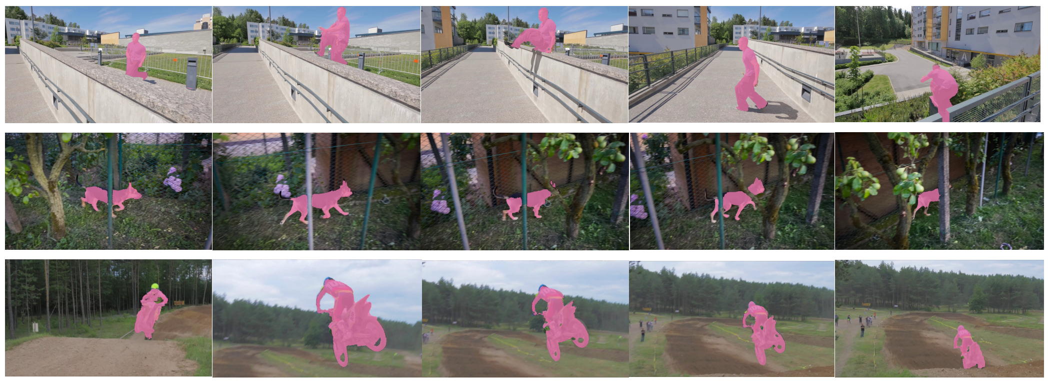


FIGURE C.2: **Sequence visualization of moving object segmentation.** We show segmentation results of some video sequences in DAVIS2016 generated by the patch-to-point aligned 3D convolution network. The output on the pixel level are indicated by the pink mask. It works well for the occlusion and deformable cases.

Visualization. We visualize the segmentation results of some sequences using the patch-to-point aligned convolution network in Figure C.2. The patch-to-point aligned convolution performs well for the occlusion and deformable cases. Some video examples are also uploaded with the file.

Appendix D

Supplementary Materials for TubeR

Losses. During TubeR training, we first produce an optimal bipartite matching δ between predictions and ground truth tubelets. $\delta(i)$ is the index of the prediction matched with the i-th ground-truth tubelet. We need to calculate the losses between a set of ground-truth tubelets $\mathbf{Y}{=}(Y_{\text{coor}}, Y_{\text{switch}}, Y_{\text{class}})$ and the matched predictions $\mathbf{y}{=}(y_{\text{coor}}, y_{\text{switch}}, y_{\text{class}})$.

We utilize four losses: an action classification loss, a box matching loss, a generalized IoU [163] loss and an action switch loss to train TubeR. The total loss is a linear combination of the four losses:

$$\begin{aligned}\mathcal{L} = \lambda_1 \mathcal{L}_{\text{switch}}(y_{\text{switch}}, Y_{\text{switch}}) + \lambda_2 \mathcal{L}_{\text{class}}(y_{\text{class}}, Y_{\text{class}}) \\ + \lambda_3 \mathcal{L}_{\text{box}}(y_{\text{coor}}, Y_{\text{coor}}) + \lambda_4 \mathcal{L}_{\text{iou}}(y_{\text{coor}}, Y_{\text{coor}}).\end{aligned} \tag{D.1}$$

$$\begin{aligned}\mathcal{L}_{\text{class}} = -\sum_{i=1}^{N}\sum_{j=1}^{L} [Y_{\text{class}}(i,j) \log y_{\text{class}}(\delta(i),j) \\ +(1 - Y_{\text{class}}(i,j)) \log(1 - y_{\text{class}}(\delta(i),j))].\end{aligned} \tag{D.2}$$

$$\begin{aligned}\mathcal{L}_{\text{switch}} = -\sum_{i=1}^{N}\sum_{j=1}^{T_{\text{out}}} [Y_{\text{switch}}(i,j) \log y_{\text{switch}}(\delta(i),j) \\ +(1 - Y_{\text{switch}}(i,j)) \log(1 - y_{\text{switch}}(\delta(i),j))].\end{aligned} \tag{D.3}$$

$$\mathcal{L}_{\text{box}} = \sum_{i=1}^{N}\sum_{j=1}^{T_{\text{out}}} \|Y_{\text{coor}}(i,j) - y_{\text{coor}}(\delta(i),j)\|_1. \tag{D.4}$$

$$\mathcal{L}_{\text{iou}} = \sum_{i=1}^{N}\sum_{j=1}^{T_{\text{out}}} \mathfrak{G}_{\text{iou}}(Y_{\text{coor}}(i,j), y_{\text{coor}}(\delta(i),j)), \tag{D.5}$$

$$\mathfrak{G}_{\text{iou}}(b,\hat{b}) = 1 - \left(\frac{|b \bigcap \hat{b}|}{|b \bigcup \hat{b}|} - \frac{|B(b,\hat{b}) \backslash b \bigcup \hat{b}|}{B(b,\hat{b})}\right). \tag{D.6}$$

Here $\mathfrak{G}_{\text{iou}}(b,\hat{b})$ is the generalized IoU [163] loss between two given boxes b and $\hat{b}$. We empirically set the scale parameter as $\lambda_1{=}1, \lambda_2{=}5, \lambda_3{=}2, \lambda_4{=}2$.

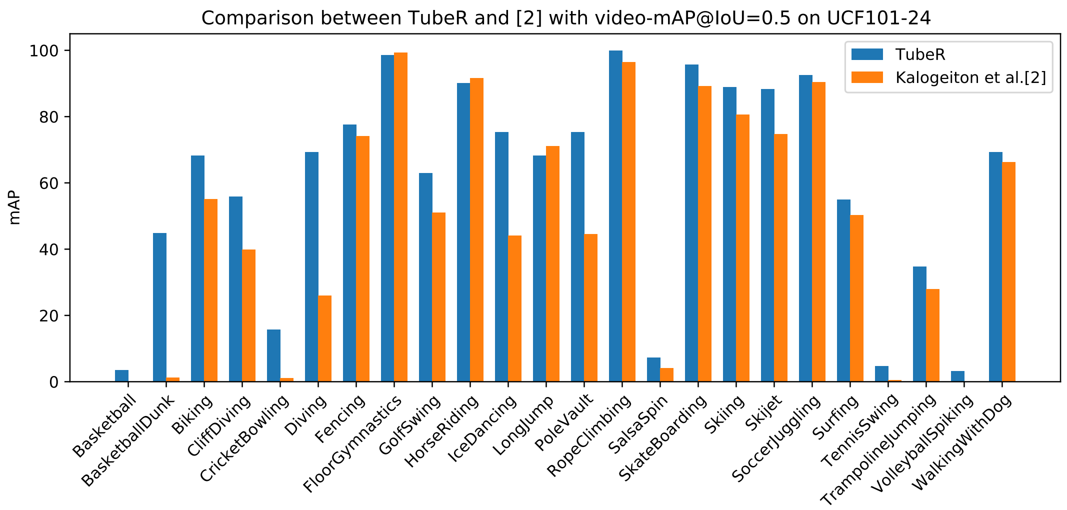


FIGURE D.1: **Comparison between TubeR and a hypotheses-base detector on UCF101-24.** TubeR performs better on most of the action classes.

TubeR vs. hypotheses-based method on UCF101-24. We compare TubeR and [106], which depends on positional hypotheses to do detection on UCF101-24, with per-class *Video-mAP*@0.5 in Figure D.1. For actions with multiple people, TubeR detects the action more precisely and produces higher video-mAP, like 44.83% for 'Basketball-Dunk' compared to [106] with video-mAP 1.19%. The tubelet attention mechanism better models the relations between the real action tubelets and surroundings. We note that [106] hardly works for 'Basketball' and 'TennisSwing' which have many transitional states. TubeR improves significantly for theses action categories. TubeR performs slightly worse for 'LongJump' in which actors may change scales along time. As [106] applies multiple scale anchors and multiple level features, it is more robust in this case. Incorporating multiple level features into TubeR will further help improve TubeR results.
TubeR vs. two-stage method on AVA. We use CSN-50 [203] as backbone with 1-view evaluation protocol unless specified otherwise. We report frame-mAP@IoU=0.5 for AVA v2.1. We compare the performance between a baseline [74] using offline person detection rather than a Region-Proposal-Network, and our TubeR. We used the same input (32 × 2) and we did not use long-term context feature for TubeR. To make a fair comparison, the baseline is evaluated using bounding boxes generated by TubeR (93.3% AP for person localization). The results are shown in Table D.1. TubeR achieves +5% frame-mAP compared to the baseline. TubeR is a unified approach which performs localization and classification from a single tubelet-level feature, rather than separated features used in the baseline. It further demonstrates the effectiveness of tubelet-level features.

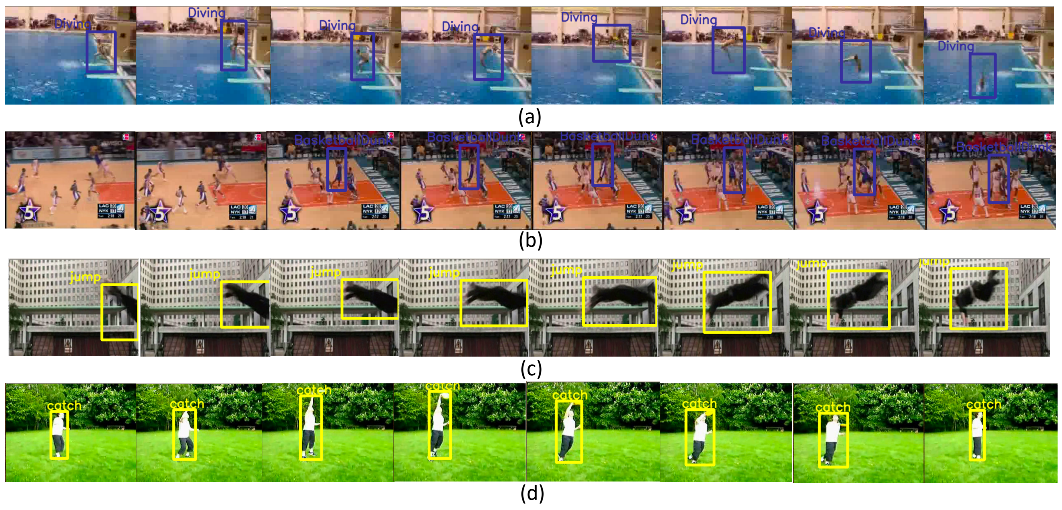


FIGURE D.2: **Action tubelets visualization on UCF101-24 and JHMDB51-21.** Each action tubelet contains its action labels and boxes on each frame. (a-b) are from UCF101-24 to show the cases with deformable actors and crowded people. (c-d) are from JHMDB51-21 to show the fast action and interacted action.

Model	frame-mAP@IoU=0.5
Baseline [74]	22.8
TubeR	**27.7**

TABLE D.1: **Comparison between a two-stage baseline and TubeR**. The TubeR only use short-term context feature. TubeR performs significantly better than the baseline.

Visualization. We show more action tubelets generated by TubeR in Figure D.2. TubeR performs well in various cases. In Figure D.2 (a-b), we show the cases with deformable actors and crowded people from UCF101-24. Figure D.2 (c-d) present the fast action and interacted action from JHMDB51-21. Moreover, some challenging cases on AVA are visualized in Figure D.3. All these cases show our TubeR is able to generate precise tubelets with various length.

We also generate a demo on AVA and some random movie clips. Different colours are used for labeling different action tubelet. One can clearly see TubeR detects action tubelets well.

We will release all TubeR code.

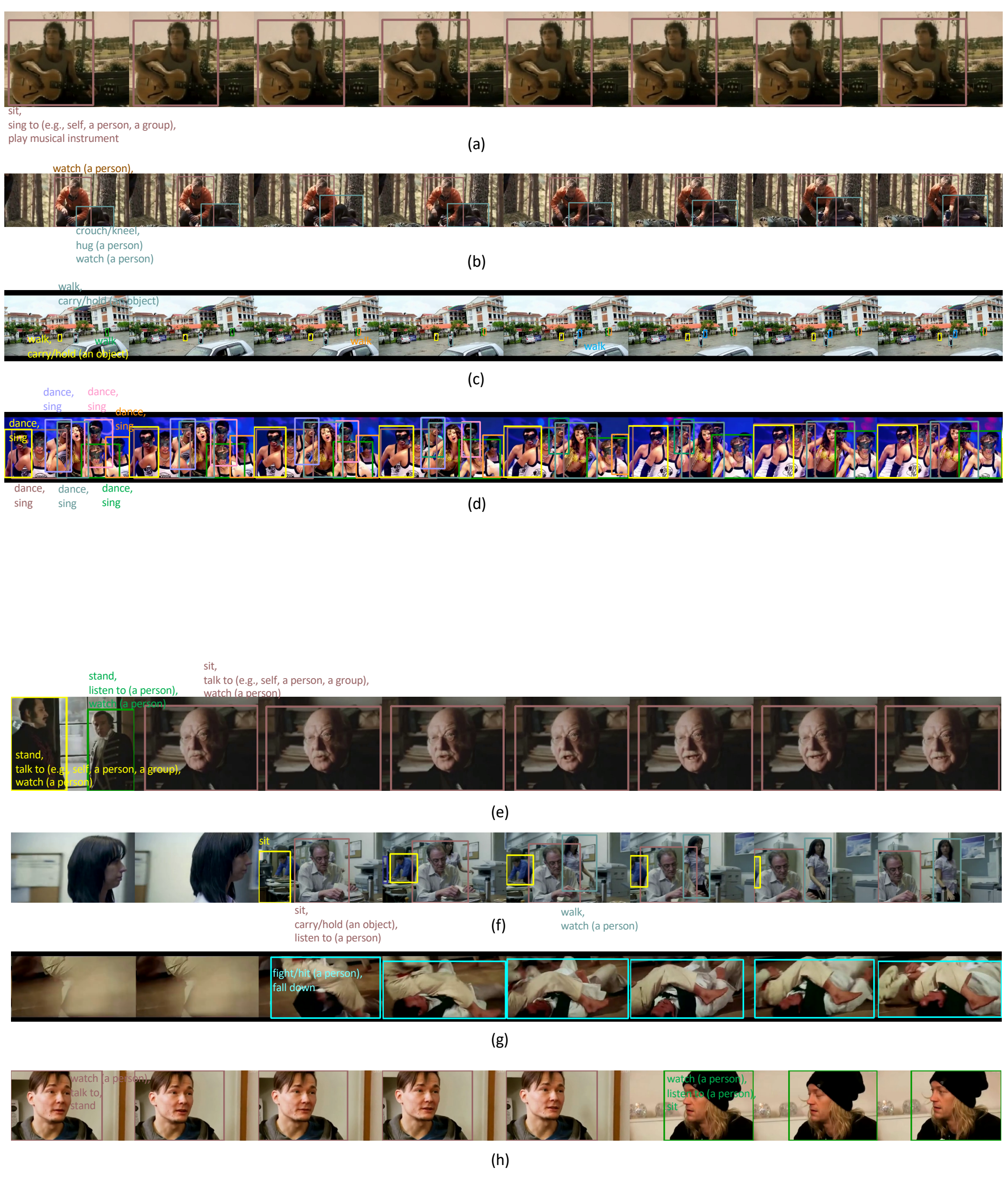


FIGURE D.3: **Action tubelets visualization on AVA.** We use different colors to mark different tubelets. Each action tubelet contains its action labels and boxes on each frame. We only show the action labels on the first frame of an action tube. We show some challenging cases here. (a) and (b) Raw actions: “play musical instrument”, “hug (a person)” . (c) Tiny actions. The actors are very tiny. (d) Crowded cases. (e-h) Shot cuts. All these cases show our TubeR is able to generate precise tubelets with various length.

Bibliography

[1] Samira Abnar and Willem H Zuidema. “Quantifying attention flow in transformers”. In: *ACL*. 2020.

[2] Alexander Alshin, Elena Alshina, and Tammy Lee. “Bi-directional optical flow for improving motion compensation”. In: *PCS*. 2010.

[3] Pablo Arbeláez et al. “Semantic segmentation using regions and parts”. In: *CVPR*. 2012.

[4] Anurag Arnab et al. “Vivit: A video vision transformer”. In: *arXiv preprint arXiv:2103.15691* (2021).

[5] Andrei Atanov et al. “Semi-conditional normalizing flows for semi-supervised learning”. In: *INNFw*. 2019.

[6] Moez Baccouche et al. “Sequential deep learning for human action recognition”. In: *International Workshop on Human Behavior Understanding*. 2011.

[7] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. “SegNet: A deep convolutional encoder-decoder architecture for image segmentation”. In: *TPAMI* (2017).

[8] Harkirat S Behl et al. “Incremental tube construction for human action detection”. In: *BMVC*. 2018.

[9] Gedas Bertasius, Lorenzo Torresani, and Jianbo Shi. “Object detection in video with spatiotemporal sampling networks”. In: *ECCV*. 2018.

[10] David S Bolme et al. “Visual object tracking using adaptive correlation filters”. In: *CVPR*. 2010.

[11] Y-Lan Boureau, Jean Ponce, and Yann LeCun. “A theoretical analysis of feature pooling in visual recognition”. In: *ICML*. 2010.

[12] William Brendel and Sinisa Todorovic. “Video object segmentation by tracking regions”. In: *ICCV*. 2009.

[13] Thomas Brox and Jitendra Malik. “Object segmentation by long term analysis of point trajectories”. In: *ECCV*. 2010.

[14] Thomas Brox et al. “High accuracy optical flow estimation based on a theory for warping”. In: *ECCV*. 2004.

[15] Antoni Buades, Bartomeu Coll, and Jean Michel Morel. “A non-local algorithm for image denoising”. In: *CVPR*. 2005.

[16] Aurélie Bugeau, Vinh-Thong Ta, and Nicolas Papadakis. “Variational exemplar-based image colorization”. In: *TIP* (2014).

[17] Holger Caesar, Jasper Uijlings, and Vittorio Ferrari. “COCO-Stuff: Thing and stuff classes in context”. In: *CVPR*. 2018.

[18] Liangliang Cao, Zicheng Liu, and Thomas S Huang. “Cross-dataset action detection”. In: *CVPR*. 2010.

[19] Yun Cao et al. “Unsupervised diverse colorization via generative adversarial networks”. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. 2017.

[20] Nicolas Carion et al. “End-to-end object detection with transformers”. In: *ECCV*. 2020.

[21] Joao Carreira and Andrew Zisserman. “Quo vadis, action recognition? a new model and the kinetics dataset”. In: *CVPR*. 2017.

[22] Guillaume Charpiat, Matthias Hofmann, and Bernhard Schölkopf. “Automatic image colorization via multimodal predictions”. In: *ECCV*. 2008.

[23] Liang-Chieh Chen et al. “Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs”. In: *TPAMI* (2018).

[24] Liang-Chieh Chen et al. “Encoder-Decoder with atrous separable convolution for semantic image segmentation”. In: *ECCV*. 2018.

[25] Shoufa Chen et al. “Watch only once: An End-to-end video action detection framework”. In: *ICCV*. 2021.

[26] Yunpeng Chen et al. “Aˆ 2-Nets: double attention networks”. In: *NeurIPS*. 2018.

[27] Heng-Da Cheng et al. “Color image segmentation: advances and prospects”. In: *PR* (2001).

[28] Zezhou Cheng, Qingxiong Yang, and Bin Sheng. “Deep colorization”. In: *ICCV*. 2015.

[29] Alex Yong-Sang Chia et al. “Semantic colorization with internet images”. In: *TOG* (2011).

[30] Ronald R Coifman and David L Donoho. “Translation-invariant de-noising”. In: *Wavelets and Statistics*. 1995.

[31] Dorin Comaniciu and Peter Meer. “Robust analysis of feature spaces: color image segmentation”. In: *CVPR*. 1997.

[32] Nieves Crasto et al. “MARS: Motion-Augmented RGB stream for action recognition”. In: *CVPR*. 2019.

[33] Jifeng Dai et al. “Deformable convolutional networks”. In: *ICCV*. 2017.

[34] Jifeng Dai et al. “R-fcn: Object detection via region-based fully convolutional networks”. In: *NIPS*. 2016.

[35] Navneet Dalal and Bill Triggs. “Histograms of oriented gradients for human detection”. In: *CVPR*. 2005.

[36] Navneet Dalal, Bill Triggs, and Cordelia Schmid. “Human detection using oriented histograms of flow and appearance”. In: *ECCV*. 2006.

[37] Martin Danelljan et al. “Adaptive color attributes for real-time visual tracking”. In: *CVPR*. 2014.

[38] Ingrid Daubechies and Wim Sweldens. “Factoring wavelet transforms into lifting steps”. In: *Journal of Fourier Analysis and Applications* (1998).

[39] Harm De Vries et al. “Modulating early visual processing by language”. In: *NIPS*. 2017.

[40] Jia Deng et al. “ImageNet: A large-scale hierarchical image database”. In: *CVPR*. 2009.

[41] Aditya Deshpande, Jason Rock, and David Forsyth. “Learning large-scale automatic image colorization”. In: *ICCV*. 2015.

[42] Aditya Deshpande et al. “Learning diverse image colorization”. In: *CVPR*. 2017.

[43] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. “Density estimation using real NVP”. In: *ICLR*. 2017.

[44] Sanwta Ram Dogiwal, YS Shishodia, and Abhay Upadhyaya. “Efficient lifting scheme based super resolution image reconstruction using low resolution images”. In: *Advanced Computing, Networking and Informatics*. 2014.

[45] Alexey Dosovitskiy et al. “An image is worth 16x16 words: Transformers for image recognition at scale”. In: *ICLR*. 2021.

[46] Alexey Dosovitskiy et al. “Flownet: Learning optical flow with convolutional networks”. In: *ICCV*. 2015.

[47] Kevin Duarte, Yogesh Rawat, and Mubarak Shah. “Videocapsulenet: A simplified network for action detection”. In: *NeurIPS*. 2018.

[48] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. “A learned representation for artistic style”. In: *ICLR*. 2017.

[49] Hugh Durrant-Whyte and Tim Bailey. “Simultaneous localization and mapping: part I”. In: *IEEE Robotics & Automation Magazine* (2006).

[50] Mark Everingham et al. “The pascal visual object classes challenge: A retrospective”. In: *IJCV* (2015).

[51] Haoqi Fan et al. “Multiscale vision transformers”. In: *arXiv preprint arXiv:2104.11227* (2021).

[52] Liang Fan, Siwei Ma, and Feng Wu. “Overview of AVS video standard”. In: *ICME*. 2004.

[53] Christoph Feichtenhofer. “X3D: Expanding architectures for efficient video recognition”. In: *CVPR*. 2020.

[54] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. “Convolutional two-stream network fusion for video action recognition”. In: *CVPR*. 2016.

[55] Christoph Feichtenhofer et al. “Slowfast networks for video recognition”. In: *ICCV*. 2019.

[56] Pedro F Felzenszwalb and Daniel P Huttenlocher. “Efficient graph-based image segmentation”. In: *IJCV* (2004).

[57] Kevin Frans. “Outline colorization through tandem adversarial networks”. In: *arXiv preprint arXiv:1704.08834* (2017).

[58] Kunihiko Fukushima. “A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position”. In: *Biol. Cybern.* (1980).

[59] Kunihiko Fukushima. “Neural network model for a mechanism of pattern recognition unaffected by shift in position-Neocognitron”. In: *IEICE Technical Report, A* (1979).

[60] Alberto Garcia-Garcia et al. “A review on deep learning techniques applied to semantic segmentation”. In: *arXiv preprint arXiv:1704.06857* (2017).

[61] Kirill Gavrilyuk et al. “Actor-transformers for group activity recognition”. In: *CVPR*. 2020.

[62] Andreas Geiger, Julius Ziegler, and Christoph Stiller. “Stereoscan: Dense 3d reconstruction in real-time”. In: *IEEE Intelligent Vehicles Symposium*. 2011.

[63] Dobrik Georgiev and Pietro Lió. “Neural bipartite matching”. In: *arXiv preprint arXiv:2005.11304* (2020).

[64] Charles J Geyer. “Practical markov chain monte carlo”. In: *Statistical Science* (1992).

[65] Golnaz Ghiasi et al. “Exploring the structure of a real-time, arbitrary neural artistic stylization network”. In: *BMVC*. 2017.

[66] Arjan Gijsenij, Theo Gevers, and Joost van de Weijer. “Generalized gamut mapping using image derivative structures for color constancy”. In: *IJCV* (2010).

[67] Rohit Girdhar et al. “A better baseline for AVA”. In: *arXiv preprint arXiv:1807.10066* (2018).

[68] Rohit Girdhar et al. “Video action transformer network”. In: *CVPR*. 2019.

[69] Ross Girshick. “Fast r-cnn”. In: *ICCV*. 2015.

[70] Ross Girshick et al. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *CVPR*. 2014.

[71] Georgia Gkioxari and Jitendra Malik. “Finding action tubes”. In: *CVPR*. 2015.

[72] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep sparse rectifier neural networks”. In: *AISTATS*. 2011.

[73] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[74] Chunhui Gu et al. “Ava: A video dataset of spatio-temporally localized atomic visual actions”. In: *CVPR*. 2018.

[75] Sergio Guadarrama et al. “Pixcolor: Pixel recursive colorization”. In: *BMVC*. 2017.

[76] Raj Kumar Gupta et al. “Image colorization using similar images”. In: *Multimedia*. 2012.

[77] Maureen T Hallinan. “Tracking: From theory to practice”. In: *Sociology of Education* (1994).

[78] Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. “Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet?” In: *CVPR*. 2018.

[79] Robert M Haralick and Linda G Shapiro. “Image segmentation techniques”. In: *Computer Vision, Graphics, and Image Processing* (1985).

[80] Susan Hayward. *Key concepts in cinema studies.* Routledge, 1996.

[81] Jiawei He et al. “Generic tubelet proposals for action localization”. In: *WACV*. 2018.

[82] Kaiming He et al. “Deep residual learning for image recognition”. In: *CVPR*. 2016.

[83] Mingming He et al. “Deep exemplar-based colorization”. In: *TOG* (2018).

[84] Dan Hendrycks and Thomas G Dietterich. “Benchmarking neural network robustness to common corruptions and surface variations”. In: *ICLR*. 2019.

[85] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* (1997).

[86] Berthold KP Horn and Brian G Schunck. “Determining optical flow”. In: *Artificial Intelligence* (1981).

[87] Gerard JC Van der Horst and Maarten A Bouman. “Spatiotemporal chromaticity discrimination”. In: *JOSA* (1969).

[88] Rui Hou, Chen Chen, and Mubarak Shah. “Tube convolutional neural network (T-CNN) for action detection in videos”. In: *ICCV*. 2017.

[89] Rui Hou et al. “An efficient 3D CNN for action/object segmentation in video”. In: *arXiv preprint arXiv:1907.08895* (2019).

[90] Ping Hu et al. “Temporally distributed networks for fast video semantic segmentation”. In: *CVPR*. 2020.

[91] Xun Huang and Serge J Belongie. “Arbitrary style transfer in real-Time with adaptive instance normalization.” In: *ICCV*. 2017.

[92] Yi-Chin Huang et al. “An adaptive edge detection based colorization algorithm and its applications”. In: *Multimedia*. 2005.

[93] Noureldien Hussein, Efstratios Gavves, and Arnold WM Smeulders. “Timeception for Complex Action Recognition”. In: *CVPR*. 2019.

[94] Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. “Let there be color!: joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification”. In: *TOG* (2016).

[95] Eddy Ilg et al. “Flownet 2.0: Evolution of optical flow estimation with deep networks”. In: *CVPR*. 2017.

[96] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *arXiv preprint arXiv:1502.03167* (2015).

[97] Revital Ironi, Daniel Cohen-Or, and Dani Lischinski. “Colorization by example.” In: *Rendering Techniques*. 2005.

[98] Phillip Isola et al. “Image-to-image translation with conditional adversarial networks”. In: *CVPR*. 2017.

[99] Pavel Izmailov et al. “Semi-supervised learning with normalizing flows”. In: *ICML*. 2020.

[100] Jörn-Henrik Jacobsen, Arnold Smeulders, and Edouard Oyallon. “i-RevNet: Deep invertible networks”. In: *ICLR*. 2018.

[101] Mihir Jain et al. “Action localization with tubelets from motion”. In: *CVPR*. 2014.

[102] Mihir Jain et al. “Tubelets: Unsupervised action proposals from spatiotemporal super-voxels”. In: *IJCV* (2017).

[103] Hueihan Jhuang et al. “Towards understanding action recognition”. In: *ICCV*. 2013.

[104] Shuiwang Ji et al. “3D convolutional neural networks for human action recognition”. In: *TPAMI* (2012).

[105] Boyuan Jiang et al. “STM: SpatioTemporal and motion encoding for action recognition”. In: *ICCV*. 2019.

[106] Vicky Kalogeiton et al. “Action tubelet detector for spatio-temporal action localization”. In: *ICCV*. 2017.

[107] Will Kay et al. “The kinetics human action video dataset”. In: *CVPR*. 2017.

[108] Aisha Urooj Khan et al. “MMFT-BERT: Multimodal fusion transformer with BERT encodings for visual question answering”. In: *arXiv preprint arXiv:2010.14095* (2020).

[109] Fahad Shahbaz Khan, Joost Van De Weijer, and Maria Vanrell. “Top-down color attention for object recognition”. In: *ICCV*. 2009.

[110] Fahad Shahbaz Khan et al. “Color attributes for object detection”. In: *CVPR*. 2012.

[111] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *ICLR*. 2015.

[112] Diederik P Kingma and Max Welling. “Auto-encoding variational bayes”. In: *ICLR* (2014).

[113] Alexander Klaser, Marcin Marszałek, and Cordelia Schmid. “A spatio-temporal descriptor based on 3d-gradients”. In: *BMVC*. 2008.

[114] Takumi Kobayashi. “Gaussian-Based pooling for convolutional neural networks”. In: *NeurIPS*. 2019.

[115] Takumi Kobayashi. “Global feature guided local pooling”. In: *ICCV*. 2019.

[116] Alex Krizhevsky and Geoffrey Hinton. “Learning multiple layers of features from tiny images”. In: *Technical Report* (2009).

[117] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *NIPS*. 2012.

[118] Till Kroeger et al. “Fast optical flow using dense inverse search”. In: *ECCV*. 2016.

[119] Hildegard Kuehne et al. “HMDB: a large video database for human motion recognition”. In: *ICCV*. 2011.

[120] Harold W Kuhn. “The Hungarian method for the assignment problem”. In: *Naval Research Logistics Quarterly* (1955).

[121] Tian Lan, Yang Wang, and Greg Mori. “Discriminative figure-centric models for joint action localization and recognition”. In: *ICCV*. 2011.

[122] Ivan Laptev et al. “Learning realistic human actions from movies”. In: *CVPR*. 2008.

[123] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. “Learning representations for automatic colorization”. In: *ECCV*. 2016.

[124] Yann LeCun et al. “Handwritten digit recognition with a back-propagation network”. In: *NIPS*. 1990.

[125] Chen-Yu Lee, Patrick W Gallagher, and Zhuowen Tu. “Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree”. In: *AISTATS*. 2016, pp. 464–472.

[126] Yong Jae Lee, Jaechul Kim, and Kristen Grauman. “Key-segments for video object segmentation”. In: *ICCV*. 2011.

[127] Thomas Leung and Jitendra Malik. “Contour continuity in region based image segmentation”. In: *ECCV*. 1998.

[128] Anat Levin, Dani Lischinski, and Yair Weiss. “Colorization using optimization”. In: *TOG* 23.3 (2004), pp. 689–694.

[129] Dong Li et al. “Recurrent tubelet proposal and recognition Networks for action detection”. In: *ECCV*. 2018.

[130] Guang Li et al. “Entangled transformer for image captioning”. In: *ICCV*. 2019.

[131] Yixuan Li et al. “Actions as moving points”. In: *ECCV*. 2020.

[132] Zhenyang Li et al. “VideoLSTM convolves, attends and flows for action recognition”. In: *CVIU* (2018).

[133] Ji Lin, Chuang Gan, and Song Han. “Tsm: Temporal shift module for efficient video understanding”. In: *ICCV*. 2019.

[134] Tsung-Yi Lin et al. “Microsoft coco: Common objects in context”. In: *ECCV*. 2014.

[135] Michael Lindenbaum, Marc Fischer, and Alfred Bruckstein. “On Gabor’s contribution to image enhancement”. In: *PR* (1994).

[136] Wei Liu et al. “Ssd: Single shot multibox detector”. In: *ECCV*. 2016.

[137] Xiaopei Liu et al. “Intrinsic colorization”. In: *TOG* (2008).

[138] Ze Liu et al. “Video swin transformer”. In: *arXiv preprint arXiv:2106.13230* (2021).

[139] Jonathan Long, Evan Shelhamer, and Trevor Darrell. “Fully convolutional networks for semantic segmentation”. In: *CVPR*. 2015.

[140] Ilya Loshchilov and Frank Hutter. “Decoupled weight decay regularization”. In: *ICLR* (2017).

[141] Zhongyu Lou et al. “Color constancy by deep learning.” In: *BMVC*. 2015.

[142] Qing Luan et al. “Natural image colorization”. In: *Proceedings of the 18th Eurographics conference on Rendering Techniques*. 2007.

[143] Brais Martinez et al. “Action recognition with spatial-temporal discriminative filter banks”. In: *ICCV*. 2019.

[144] Larry Medsker and Lakhmi Jain. “Recurrent neural networks”. In: *Design and Applications* (2001).

[145] Ross Messing, Chris Pal, and Henry Kautz. “Activity recognition using the velocity histories of tracked keypoints”. In: *ICCV*. 2009.

[146] Etienne Mouragnon et al. “Real time localization and 3d reconstruction”. In: *CVPR*. 2006.

[147] Walter Murch. *In the Blink of an Eye*. Silman-James Press Los Angeles, 2001.

[148] Romi Nijhawan and Beena Khurana. *Space and time in perception and action*. Cambridge University Press, 2010.

[149] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. “Learning deconvolution network for semantic segmentation”. In: *ICCV*. 2015.

[150] Dan Oneata, Jakob Verbeek, and Cordelia Schmid. “Efficient action localization with approximately normalized fisher vectors”. In: *CVPR*. 2014.

[151] Aaron van den Oord et al. “Conditional image generation with pixelcnn decoders”. In: *NIPS*. 2016.

[152] Junting Pan et al. “Actor-context-actor relation network for spatio-temporal action localization”. In: *CVPR*. 2021.

[153] Adam Paszke et al. “Automatic differentiation in PyTorch”. In: (2017).

[154] Xiaojiang Peng and Cordelia Schmid. “Multi-region two-stream R-CNN for action detection”. In: *ECCV*. 2016.

[155] Federico Perazzi et al. “A benchmark dataset and evaluation methodology for video object segmentation”. In: *CVPR*. 2016.

[156] Ethan Perez et al. “Film: Visual reasoning with a general conditioning layer”. In: *AAAI*. 2018.

[157] Patrick Pérez et al. “Color-based probabilistic tracking”. In: *ECCV*. 2002.

[158] Béatrice Pesquet-Popescu and Vincent Bottreau. “Three-dimensional lifting schemes for motion compensated video compression”. In: *ICASSP*. 2001.

[159] Boris T Polyak and Anatoli B Juditsky. “Acceleration of stochastic approximation by averaging”. In: *SIAM Journal on Control and Optimization* (1992).

[160] Yingge Qu, Tien-Tsin Wong, and Pheng-Ann Heng. “Manga colorization”. In: *TOG* (2006).

[161] Alec Radford, Luke Metz, and Soumith Chintala. “Unsupervised representation learning with deep convolutional generative adversarial networks”. In: *ICLR*. 2016.

[162] Shaoqing Ren et al. “Faster r-cnn: Towards real-time object detection with region proposal networks”. In: *NIPS*. 2015.

[163] Hamid Rezatofighi et al. “Generalized intersection over union: A metric and a loss for bounding box regression”. In: *CVPR*. 2019.

[164] Oren Rippel, Jasper Snoek, and Ryan P Adams. “Spectral representations for convolutional neural networks”. In: *NIPS*. 2015.

[165] Maria Ximena Bastidas Rodriguez et al. “Deep adaptive wavelet network”. In: *WACV*. 2020.

[166] Mikel D Rodriguez, Javed Ahmed, and Mubarak Shah. “Action mach a spatio-temporal maximum average correlation height filter for action recognition”. In: *CVPR*. 2008.

[167] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-Net: Convolutional networks for biomedical image segmentation”. In: *MICCAI*. 2015.

[168] John Rosenberg. *The healthy edit: Creative techniques for perfecting your movie*. Routledge, 2013.

[169] Amélie Royer, Alexander Kolesnikov, and Christoph H Lampert. “Probabilistic Image Colorization”. In: *BMVC*. 2017.

[170] Faraz Saeedan et al. “Detail-preserving pooling in deep networks”. In: *CVPR*. 2018.

[171] Suman Saha, Gurkirt Singh, and Fabio Cuzzolin. “AMTnet: Action-Micro-Tube regression by end-to-end trainable deep architecture”. In: *ICCV*. 2017.

[172] Suman Saha et al. “Deep learning for detecting multiple space-time action tubes in videos”. In: *BMVC*. 2016.

[173] Tim Salimans et al. “Pixelcnn++: A pixelcnn implementation with discretized logistic mixture likelihood and other modifications”. In: *ICLR*. 2017.

[174] Juan María Sanchez and Xavier Binefa. “Improving visual recognition using color normalization in digital video applications”. In: *ICME*. 2000.

[175] Koen E. A. van de Sande, Theo Gevers, and Cees G. M. Snoek. “Evaluating color descriptors for object and scene recognition”. In: *TPAMI* (2010).

[176] Mark Sandler et al. “MobileNet-V2: Inverted residuals and linear bottlenecks”. In: *CVPR*. 2018.

[177] Patsorn Sangkloy et al. “Scribbler: Controlling deep image synthesis with sketch and color”. In: *CVPR*. 2017.

[178] Dominik Scherer, Andreas Müller, and Sven Behnke. “Evaluation of pooling operations in convolutional architectures for object recognition”. In: *ICANN*. 2010.

[179] Wenzhe Shi et al. “Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network”. In: *CVPR*. 2016.

[180] Thomas Sikora. “The MPEG-4 video standard verification model”. In: *TCSVT* (1997).

[181] Karen Simonyan and Andrew Zisserman. “Two-stream convolutional networks for action recognition in videos”. In: *NIPS*. 2014.

[182] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *ICLR*. 2015.

[183] Gurkirt Singh, Suman Saha, and Fabio Cuzzolin. “TraMNet-Transition matrix network for efficient action tube proposals”. In: *ACCV*. 2018.

[184] Gurkirt Singh et al. “Online real-Time multiple spatiotemporal action localisation and prediction.” In: *ICCV*. 2017.

[185] Lin Song et al. “Tacnet: Transition-aware context network for spatio-temporal action detection”. In: *CVPR*. 2019.

[186] Khurram Soomro, Haroon Idrees, and Mubarak Shah. “Predicting the where and what of actors and actions through online action localization”. In: *CVPR*. 2016.

[187] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. “UCF101: A dataset of 101 human actions classes from videos in the wild”. In: *arXiv preprint arXiv:1212.0402* (2012).

[188] Jost Tobias Springenberg et al. “Striving for simplicity: The all convolutional net”. In: *ICLR*. 2015.

[189] Rastislav Šrámek, Broňa Brejová, and Tomáš Vinař. “On-line Viterbi algorithm and its relationship to random walks”. In: *arXiv preprint arXiv:0704.0062* (2007).

[190] Jonathan C Stroud et al. “D3D: Distilled 3D networks for video action recognition”. In: *arXiv preprint arXiv:1812.08249* (2018).

[191] Chen Sun et al. “Actor-centric relation network”. In: *ECCV*. 2018.

[192] Ju Sun et al. “Hierarchical spatio-temporal context modeling for action recognition”. In: *CVPR*. 2009.

[193] Chiranjib Sur. “Self-segregating and coordinated-segregating transformer for focused deep multi-modular network for visual question answering”. In: *arXiv preprint arXiv:2006.14264* (2020).

[194] M. J. Swain and D. H. Ballard. “Color indexing”. In: *IJCV* (1991).

[195] Wim Sweldens. “The lifting scheme: A construction of second generation wavelets”. In: *SIAM Journal on Mathematical Analysis* 29.2 (1998), pp. 511–546.

[196] Yu-Wing Tai, Jia-Ya Jia, and Chi-Keung Tang. “Local color transfer via probabilistic segmentation by expectation-maximization”. In: *CVPR*. 2005.

[197] Jiajun Tang et al. “Asynchronous interaction aggregation for action detection”. In: *ECCV*. 2020.

[198] Yicong Tian, Rahul Sukthankar, and Mubarak Shah. “Spatiotemporal deformable part models for action detection”. In: *CVPR*. 2013.

[199] Du Tran and Junsong Yuan. “Max-margin structured output regression for spatio-temporal action localization”. In: *NIPS*. 2012.

[200] Du Tran, Junsong Yuan, and David Forsyth. “Video event detection: From subvolume localization to spatio-temporal path search”. In: *TPAMI* (2014).

[201] Du Tran et al. “A closer look at spatiotemporal convolutions for action recognition”. In: *CVPR*. 2018.

[202] Du Tran et al. “Learning spatiotemporal features with 3d convolutional networks”. In: *ICCV*. 2015.

[203] Du Tran et al. “Video classification with channel-separated convolutional networks”. In: *ICCV*. 2019.

[204] Shimon Ullman. “The interpretation of structure from motion”. In: *Proceedings of the Royal Society of London. Series B. Biological Sciences* (1979).

[205] Jan C Van Gemert et al. “APT: Action localization proposals from dense trajectories.” In: *BMVC*. 2015.

[206] Ashish Vaswani et al. “Attention is all you need”. In: *NIPS*. 2017.

[207] Carl Vondrick et al. “Tracking emerges by colorizing videos”. In: *ECCV*. 2018.

[208] Heng Wang and Cordelia Schmid. “Action recognition with improved trajectories”. In: *ICCV*. 2013.

[209] Heng Wang et al. “Dense trajectories and motion boundary descriptors for action recognition”. In: *IJCV* (2013).

[210] Limin Wang, Yu Qiao, and Xiaoou Tang. “Action recognition with trajectory-pooled deep-convolutional descriptors”. In: *CVPR*. 2015.

[211] Limin Wang et al. “Towards good practices for very deep two-stream convnets”. In: *arXiv preprint arXiv:1507.02159* (2015).

[212] Shiyao Wang et al. “Fully motion-aware network for video object detection”. In: *ECCV*. 2018.

[213] Xiaolong Wang et al. “Non-local neural networks”. In: *CVPR*. 2018.

[214] Xintao Wang et al. “Recovering realistic texture in image super-resolution by deep spatial feature transform”. In: *CVPR*. 2018.

[215] Zhou Wang, Eero P Simoncelli, and Alan C Bovik. “Multiscale structural similarity for image quality assessment”. In: *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers*. 2003.

[216] Philippe Weinzaepfel, Zaid Harchaoui, and Cordelia Schmid. “Learning to track for spatio-temporal action localization”. In: *ICCV*. 2015.

[217] Tomihisa Welsh, Michael Ashikhmin, and Klaus Mueller. “Transferring color to greyscale images”. In: *TOG* (2002).

[218] Travis Williams and Robert Li. “Wavelet pooling for convolutional neural networks”. In: *ICLR*. 2018.

[219] Chao-Yuan Wu et al. “Long-term feature banks for detailed video understanding”. In: *CVPR*. 2019.

[220] Yonghong Wu et al. “Adaptive denoising based on lifting scheme”. In: *ICSP*. 2004.

[221] Saining Xie et al. “Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification”. In: *ECCV*. 2018.

[222] Kai Xu et al. “Spatiotemporal CNN for video object segmentation”. In: *CVPR*. 2019.

[223] Mingze Xu et al. “Long short-Term transformer for online action detection”. In: *NeurIPS*. 2021.

[224] Xitong Yang et al. “Step: Spatio-temporal progressive learning for video action detection”. In: *CVPR*. 2019.

[225] Zhenheng Yang, Jiyang Gao, and Ram Nevatia. “Spatio-temporal action detection with cascade proposal and location anticipation”. In: *BMVC*. 2017.

[226] Liron Yatziv and Guillermo Sapiro. “Fast image and video colorization using chrominance blending”. In: *TIP* (2006).

[227] Alper Yilmaz, Omar Javed, and Mubarak Shah. “Object tracking: A survey”. In: *CSUR* (2006).

[228] Gang Yu and Junsong Yuan. “Fast action proposals for human action detection and search”. In: *CVPR*. 2015.

[229] Junsong Yuan, Zicheng Liu, and Ying Wu. “Discriminative video pattern search for efficient action detection”. In: *TPAMI* (2011).

[230] Li Yuan et al. “Tokens-to-token vit: Training vision transformers from scratch on imagenet”. In: *arXiv preprint arXiv:2101.11986* (2021).

[231] Kaiyu Yue et al. “Compact generalized non-local network”. In: *NeurIPS*. 2018.

[232] Matthew D Zeiler and Rob Fergus. “Stochastic pooling for regularization of deep convolutional neural networks”. In: *arXiv preprint arXiv:1301.3557* (2013).

[233] Shuangfei Zhai et al. “S3Pool: Pooling with stochastic spatial sampling”. In: *CVPR*. 2017.

[234] Hang Zhang et al. “Context encoding for semantic segmentation”. In: *CVPR*. 2018.

[235] Richard Zhang. “Making convolutional networks shift-invariant again”. In: *ICML*. 2019.

[236] Richard Zhang, Phillip Isola, and Alexei A Efros. “Colorful image colorization”. In: *ECCV*. 2016.

[237] Richard Zhang et al. “Real-time user-guided image colorization with learned deep priors”. In: *SIGGRAPH*. 2017.

[238] Yanyi Zhang, Xinyu Li, and Ivan Marsic. "Multi-Label Activity Recognition Using Activity-Specific Features and Activity Correlations". In: *CVPR*. 2021.

[239] Yanyi Zhang et al. "Vidtr: Video transformer without convolutions". In: *ICCV*. 2021.

[240] Jiaojiao Zhao and Cees GM Snoek. "Dance with flow: Two-in-one stream action detection". In: *CVPR*. 2019.

[241] Jiaojiao Zhao and Cees GM Snoek. "LiftPool: Bidirectional Convnet pooling". In: *ICLR*. 2021.

[242] Jiaojiao Zhao et al. "Pixel-level semantics guided image colorization". In: *BMVC*. 2018.

[243] Jiaojiao Zhao et al. "Pixelated semantic colorization". In: *IJCV* (2020).

[244] Jiaojiao Zhao et al. "TubeR: Tube-transformer for action detection". In: *arXiv preprint arXiv:2104.00969* (2021).

[245] Yue Zhao, Yuanjun Xiong, and Dahua Lin. "Trajectory convolution for action recognition". In: *NeurIPS*. 2018.

[246] Yi Zheng, Ruijin Wang, and Jianping Li. "Nonlinear wavelets and BP neural networks adaptive lifting scheme". In: *ICACIA*. 2010.

[247] Bolei Zhou et al. "Learning deep features for discriminative localization". In: *CVPR*. 2016.

[248] Xizhou Zhu et al. "Deformable DETR: Deformable transformers for end-to-end object detection". In: *ICLR*. 2021.

[249] Mohammadreza Zolfaghari, Kamaljeet Singh, and Thomas Brox. "Eco: Efficient convolutional network for online video understanding". In: *ECCV*. 2018.

Summary

This thesis aims at learning continuity for visual recognition. As a natural property of images and videos, continuity is important for many computer vision tasks. The thesis strives to answer the research question "*What is the benefit of continuity for image and video recognition?*" Therefore, the thesis includes two parts, respectively looking into spatial continuity of images and spatio-temporal continuity of videos. Our contributions are:

Part I: Learning Continuity for Image Recognition

In Chapter 2, we explore spatial continuity for image colorization. While many image colorization algorithms have recently shown the capability of producing plausible color versions from gray-scale photographs, they still suffer from limited semantic understanding. To address this shortcoming, we propose to exploit pixelated object semantics to guide image colorization. The rationale is that human beings perceive and distinguish colors based on the semantic categories of objects. Starting from an autoregressive model, we generate image color distributions, from which diverse colored results are sampled. We propose two ways to incorporate object semantics into the colorization model: through a pixelated semantic embedding and a pixelated semantic generator. Our network, when trained with semantic segmentation labels, produces more realistic and finer results compared to the colorization state-of-the-art.

Chapter 3 presents a new pooling method maintaining better spatial continuity. Most existing pooling operations downsample the feature maps, which is a lossy process. Moreover, they are not invertible: upsampling a downscaled feature map can not recover the lost information in the downsampling. By adopting the philosophy of the classical Lifting Scheme from signal processing, we propose LiftPool for bidirectional pooling layers, including LiftDownPool and LiftUpPool. LiftDownPool decomposes a feature map into various downsized sub-bands, each of which contains information with different frequencies. LiftUpPool is able to generate a refined upsampled feature map using the detail sub-bands, which is useful for image-to-image translation challenges. Experiments show the proposed methods achieve better results on image classification and semantic segmentation, using various backbones. Moreover, LiftDownPool offers better robustness to input corruptions and perturbations.

Part II: Learning Continuity for Video Recognition

The goal of Chapter 4 is to utilize temporal continuity for action detection. The two-stream detection network based on RGB and flow provides state-of-the-art accuracy at the expense of a large model-size and heavy computation. We propose to embed RGB and optical-flow into a single two-in-one stream network with new layers. A motion condition layer extracts motion information from flow images, which is leveraged by the motion modulation layer to generate transformation parameters for modulating the low-level RGB features. The method is easily embedded in existing appearance- or two-stream action detection networks, and trained end-to-end. Leveraging the motion condition to modulate RGB features improves detection accuracy and is more efficient.

Chapter 5 targets on endowing a 3D-Convnet with spatio-temporal continuity. Effectively capturing spatio-temporal information is critical for video analysis. The 3D convolution is designed for this objective. It aggregates the information from the same spatial grids across the time steps, thereby ignoring the object dislocations caused by motion. In this chapter, we present aligned 3D convolution blocks, which collect the valuable information from the locations aligned by the learned offsets rather than the original dislocated positions. Furthermore, we propose three variants: patch-to-patch aligned convolution, patch-to-point aligned convolution and point-to-point aligned convolution. Thanks to their ability to align spatial video changes over time, without the need to pre-compute optical flow, aligned 3D convolution blocks are effective for different frame-rate videos. Additionally, the building blocks are easily plugged into many existing network architectures. Experiments illustrate the effectiveness and robustness of the aligned 3D convolutions.

In Chapter 6, we propose TubeR: a simple solution for spatio-temporal video action detection. Different from existing methods that depend on either an off-line actor detector or hand-designed actor-positional hypotheses like proposals or anchors, we propose to directly detect an action tubelet in video by simultaneously performing action localization and recognition from a single representation. TubeR learns a set of tubelet-queries and utilizes a tubelet-attention module to model the dynamic spatio-temporal nature of a video clip, which effectively reinforces the model capacity compared to using actor-positional hypotheses in the spatio-temporal space. For videos containing transitional states or scene changes, we propose a context aware classification head to utilize short-term and long-term context to strengthen action classification, and an action switch regression head for detecting the precise temporal action extent. TubeR directly produces action tubelets with variable lengths and even maintains good results for long video clips.

This thesis has explored potential benefits of continuity for image and video recognition. Various fundamental topics have been covered, including image colorization, image classification, semantic segmentation, video action detection, action recognition and video object segmentation. Finally, our journey through exploring continuity in visual recognition has come to an end, but the quest will never be over. With speedy developments in computer vision by learning, more and more attention is being paid to the benefit of continuity, especially for video tasks. Future work should target on even more flexible spatio-temporal continuity modelling to further boost video understanding.

Samenvatting

Dit proefschrift is gericht op het leren van continuïteit voor visuele herkenning. Als een natuurlijke eigenschap van afbeeldingen en video's is continuïteit belangrijk voor veel beeldverwerkingstaken. Het proefschrift streeft naar het beantwoorden van de volgende onderzoeksvraag: "Wat is het nut van continuïteit voor beeld- en videoherkenning?" Het proefschrift omvat twee delen: het kijken naar ruimtelijke continuïteit van afbeeldingen en spatio-temporele continuïteit van video's. Onze bijdragen zijn:

Deel I: Continuïteit leren voor beeldherkenning

In hoofdstuk 2 onderzoeken we de ruimtelijke continuïteit voor beeld inkleuring. Hoewel veel algoritmen voor het inkleuren van beelden onlangs het vermogen hebben laten zien om geloofwaardige kleurenversies van grijswaardenfoto's te produceren, is er nog steeds een beperkt semantisch begrip. Om deze tekortkomingen te adresseren stellen we voor om gepixelde object semantiek te exploiteren om beeld inkleuring te bewerkstelligen. De grondgedachte is dat mensen waarnemen en onderscheid maken tussen kleuren op basis van de semantische categorieën van objecten. Beginnend vanuit een autoregressief model genereren we beeld-kleur verdelingen van waaruit diverse gekleurde resultaten worden gesampled. We stellen twee manieren voor om object semantiek te verwerken in het kleuringsmodel: via een gepixelde semantische representatie en een gepixelde semantische generator. Ons netwerk, wanneer het getraind wordt met semantische segmentatielabels, produceert meer realistische en fijnere resultaten in vergelijking met de state-of-the-art kleuring.

Hoofdstuk 3 presenteert een nieuwe pooling methode die een betere ruimtelijke continuïteit behoudt. De meeste bestaande pooling operaties downsamplen de feature maps, wat een verliesgevend proces is. Bovendien zijn ze niet omkeerbaar: het upsamplen van een verkleinde feature map kan niet de verloren informatie in de downsampling herstellen. Door de filosofie van het klassieke 'Lifting Scheme' van signaalverwerking aan te passen, stellen we LiftPool voor voor bidirectionele pooling lagen, inclusief LiftDownPool en LiftUpPool. LiftDownPool ontleedt een feature map in verschillende verkleinde sub-banden, waarvan elk informatie bevat met verschillende frequenties. LiftUpPool kan een verfijnde ge-upsamplede feature map genereren door gedetailleerde sub-banden te gebruiken, wat handig is voor uitdagingen in beeld-naar-beeld vertaling. Experimenten tonen aan dat de voorgestelde methoden betere resultaten opleveren op beeldclassificatie en semantische segmentatie, met behulp van verschillende backbones. Bovendien biedt LiftDownPool een betere robuustheid voor invoercorrupties en verstoringen.

Deel II: Continuïteit leren voor videoherkenning

Het doel van hoofdstuk 4 is om temporele continuïteit te gebruiken voor actiedetectie. Het tweestromige detectie netwerk gebaseerd op RGB en stroom biedt state-of-the-art nauwkeurigheid ten koste van een grote modelgrootte en zware berekeningen. We stellen voor om RGB en optische stroom in een enkel twee-in-een stroom netwerk te embedden met nieuwe lagen. Een bewegingscondidtielaag extraheert bewegingsinformatie uit stroombeelden, die wordt benut door de bewegingsmodulatielaag om transformatieparameters te genereren om de low-level RGB-functies te moduleren. De methode kan eenvoudig worden ingebed in een bestaande verschijning of tweestrooms actiedetectienetwerken en kan end-to-end worden getraind. Het gebruik maken van de bewegingsconditie om RGB-functies te moduleren verbetert de detectienauwkeurigheid en is efficiënter.

Hoofdstuk 5 richt zich op het geven van een 3D-Convnet met spatio-temporele continuïteit. Het effectief vastleggen van spatio-temporele informatie is van cruciaal belang voor video-analyse. De 3D convolutie is ontworpen voor dit doel. Het aggregeert de informatie van dezelfde ruimtelijke rasters over tijd, waarbij de objectdislocaties worden genegeerd die worden veroorzaakt door beweging. In dit hoofdstuk presenteren we uitgelijnde 3D-convolutieblokken die de waardevolle informatie verzamelen van locaties die uitgelijnd zijn door de geleerde offsets in plaats van de oorspronkelijke ontwrichte posities. Verder stellen we drie varianten voor: patch-naar-patch uitgelijnde convolutie, patch-naar-punt uitgelijnde convolutie en punt-naar-punt uitgelijnde convolutie. Dankzij hun vermogen om ruimtelijke videoveranderingen in de loop van de tijd op elkaar af te stemmen, zonder de noodzaak om de optische stroom vooraf te berekenen, zijn uitgelijnde 3D-convolutieblokken effectief voor video's met verschillende frame snelheden. Bovendien zijn de bouwstenen eenvoudig in te pluggen in veel bestaande netwerkarchitecturen. Experimenten illustreren de effectiviteit en robuustheid van de uitgelijnde 3D-convoluties.

In hoofstuk 6 stellen we TubeR voor: een eenvoudige oplossing voor spatio-temporele video actiedetectie. Anders dan bestaande methoden die afhankelijk zijn van een offline actor detector of met de hand ontworpen actor-positionele hypothesen zoals voorstellen of ankers, stellen we voor om een actie tubelet in video te detecteren door gelijktijdig actielokalisatie en herkenning vanuit een enkele representatie uit te voeren. TubeR leert een verzameling van tubelet query's en maakt gebruik van een tubelet-aandacht-module om de dynamische spatio-temporele aard van een video clip te modelleren, waardoor de modelcapaciteit effectief wordt versterkt vergeleken met het gebruik van actor-positionele hypothesen in de spatio-temporele ruimte. Voor video's met overgangstoestanden of scèneveranderingen stellen we een contextbewuste classificatie kop voor om korte- en lange termijn context te gebruiken om actieclassificatie te versterken en een actie schakelaar regressie kop om precieze temporele actie omvang te detecteren. TubeR produceert direct actie tubelets met variabele lengtes en behoudt zelfs goede resultaten voor lange videoclips.

Dit proefschrift heeft de potentiële voordelen van continuïteit voor beeld- en video-herkenning onderzocht. Er zijn verschillende fundamentele onderwerpen behandeld, waaronder het inkleuren van afbeeldingen, beeldclassificatie, semantische segmentatie, video-actiedetectie, actieherkenning en video-object segmentatie. Uiteindelijk is onze reis door het verkennen van continuïteit in visuele herkenning tot een einde gekomen, maar de zoektocht zal nooit voorbij zijn. Met snelle ontwikkelingen in beeldverwerking door te leren, komt er steeds meer aandacht ten behoeve van de continuïteit, met name voor videotaken. Toekomstig werk moet gericht zijn op nog flexibelere spatio-temporele continuïteitsmodellering om begrip over video verder te stimuleren.

Acknowledgments

Five chapters, four years and one PhD thesis, these simple numbers carry tons of efforts from many people. The story of my PhD can never be ended up without these lovable people.

As my supervisor, Cees plays the most important role in my PhD journey. He provided a free research environment to me, where I did not get much pressure from graduation, but drove myself to the interesting research. I got professional guides from him, not only on research, but also academic concepts. His commitment to meticulous scientific inquiry influenced me deeply. I enjoyed all submissions we worked together and learned the new from him every time on writing, presentation, reviewing, communication and collaboration. We shared the happiness and sadness of each submission. A flood of memories rushes into my mind when I am writing these words. How can I express my appreciation to him in a few words! I feel so lucky to have him as my supervisor for the PhD.

I deeply thank Pascal who is my co-promotor. Even before becoming my co-promotor, Pascal is always helpful for junior PhD students. He is really professional in all aspects. I got precious advice from him at the beginning of my PhD. Working with him for the DIVA project is a wonderful starting point. I also absorbed lots of skills from his professional presentation course. I expect more collaboration with him in the future.

I would like to express my respect to Arnold who is the most senior scientist in our lab. His questions and suggestions on SOOS talks are always impressive and inspired. Discussing with him is like a brain-storm. His spirit of tracing to the source in research deserves respect. Arnold also supervised me in the transformer project. He stimulated me to think and dig deep. I owe a deep sense of gratitude to Arnold.

Sincere gratitude is expressed to the committee members for my Ph.D. defense: Prof. C. Sánchez Gutiérrez, Prof. A.W.M. Smeulders, Prof. dr. Th. Gevers, Prof. dr. D. Damen and Dr. ir. R.W. Poppe. It is my honor to have you as my esteemed opponent. I appreciate your time in reading and helping to improve my thesis.

It is so great to know the guys in Kepler when I did an internship there. I thank Harro Stokman, Marc van Oldenborgh, Jaap Veerhoek, Koen van de Sande, Fares Alnajar, Per John and Rob van der Leek. Special thanks go to Ran Tao who was my mentor and good friend. I learned much from them and I pretty enjoyed the time in Kepler.

Another unforgettable experience was my internship in Amazon Rekognition Team Seattle. Even I did it remotely, I had a great time with Xinyu Li, Joe, Chunhui Liu, Zhe Wang, Hengduo Li, Shuai Bing, Hao Chen and Yuanjun Xiong. Thank all

you guys for helping me during my internship, especially Xinyu, who is so nice and we built firm friendships.

I would like to thank my old lab mates, Gjorgji Strezoski, William Thong, Devanshu Arya, Yunlu Chen, Shuo Chen, Mehmet Altinkaya, David Zhang and Fida Thoker. I enjoyed times in bars with you guys, and plank competitions in the lab. Special thanks should be given to Sarah Ibrahimi, who is one of my best friends and helped me to translate my summary in English to a Dutch version. I never forget the happy moments we spent together. The movies, activities, dance shows, museum exhibitions, the summer school, and every meal we had together are just like happening yesterday. You guys lead me to diverse cultures. I also thank Shihan Wang, Zenglin Shi, Yunhua Zhang and Sadaf Gulshad. You guys make my PhD more colorful. I learned a lot from all of you, not limited to research, but also life. Simple words 'Friendship Forever' can not express my feelings.

My PhD gets indispensable favors from Dennis and Virginie. Dennis supplied all the hardware, software, and technical support during my PhD. He is one of the most important people in the lab. I can not forget his warm words and efforts to fix my PC which was 'planted' by me. He always teaches me a lot without awareness. Virginie is a meek lady who brings sunshine to me. She is so gentle and willing to help. I appreciated her emails for caring for me when I was in quarantine in China. I feel she is like a family from her warm greetings.

I thank all the VisLab colleagues: Efstratios Gavves, Iris Groen, Xiantong Zhen, Yuki M. Asano, Hazel Doughty, Ivan Sosnovik, Artem Moskalev, Mert Kilickaya, Riaan Zoetmulder, Noureldien Hussein, Tom Runia, Kirill Gavrilyuk, Shuai Liao, Berkay Kicanaoglu, Deepak Gupta, Amber Brands, Mina Ghadimiatigh, Melika Ayoughi, Sarah Rastegar and Nguyen Duy Kien. And so many names are not listed here but in my heart. I borrow one sentence from a very famous Chinese poem 'Hai Nei Cun Zhi Ji, Tian Ya Ruo Bi Lin (A bosom friend afar brings a distant land near.)' to depict our friendship.

Lastly, I must thank my big family. Thank my parents for their unlimited support and deep understanding. Thank my sisters and brothers for taking on the responsibility for the family that should have been borne by me. I own my further thanks to my third sister Nana Zhao, who designed the beautiful cover for my thesis. Thank all my nieces and nephews, Jiahui Tang, Yi Zhao, Jiayu Tang, Xinzhu Li, Hexin Chen, Xingchen Li and Xiu'an Chen, who bring me so much happiness. No matter how far apart we are, our hearts are like the sweet 'sugar-coated haws on the stick'.