ABSTRACT

Title of Dissertation: IMAGE AND VIDEO UNDERSTANDING WITH CONSTRAINED RESOURCES

Zuxuan Wu
Doctor of Philosophy, 2020

Dissertation Directed by: Professor Larry S. Davis
Department of Computer Science

Recent advances in computer vision tasks have been driven by high-capacity deep neural networks, particularly Convolutional Neural Networks (CNNs) with hundreds of layers trained in a supervised manner. However, this poses two significant challenges: (1) the increased depth in CNNs that leads to significant improvements over competitive benchmarks at the same time, limits their deployment in real-world scenarios due to high computational cost, (2) the need to collect millions of human labeled samples for training prevents such approaches to scale, especially for fine-grained image understanding like semantic segmentation, where dense annotations are extremely expensive to obtain. To mitigate these issues, we focus on image and video understanding with constrained resources, in the forms of computational resources and annotation resources. In particular, we present approaches that (1) investigate dynamic computation frameworks which adaptively allocate computing resources on-the-fly given a novel image/video to manage the trade-off between accuracy and computational complexity; (2) derive robust representations with minimal human supervision through exploring context relationships or using shared information across domains.

With this in mind, we first introduce BlockDrop, a conditional computation approach that learns to dynamically choose which layers of a deep network to execute during inference so as to best reduce total computation without degrading prediction accuracy. Next, we generalize the idea of conditional computation of images to videos by presenting AdaFrame, a framework that adaptively selects relevant frames on a per-input basis for fast video recognition. AdaFrame assumes access to all frames in videos, and hence can be only used in offline settings. To mitigate this issue, we introduce LiteEval, a simple yet effective coarse-to-fine framework for resource efficient video recognition, suitable for both online and offline scenarios.

To derive robust feature representations with limited annotation resources, we first explore the power of spatial context as a supervisory signal for learning visual representations. In addition, we also propose to learn from synthetic data rendered by modern computer graphics tools, where ground-truth labels are readily available. We propose Dual Channel-wise Alignment Networks (DCAN), a simple yet effective approach to reduce domain shift at both pixel-level and feature-level, for unsupervised scene adaptation.

IMAGE AND VIDEO UNDERSTANDING WITH CONSTRAINED RESOURCES

by

Zuxuan Wu

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2020

Acknowledgments

Larry S. Davis, my advisor, for giving me the guidance, encouragement and freedom to pursue my passion and always offering to help out when needed.

My committee members: Rama Chellappa, David Jacobs, Tom Goldstein and Abhinav Shrivastava, for providing insightful comments and valuable suggestions.

My mentors in academia and industry: Yu-Gang Jiang, Leon Sigal, Rogerio Feris, Caiming Xiong and Ser-Nam Lim for free meals at conferences and useful advice.

My AVW 3362 “dungeon” office mates: Bharat Singh, Xintong Han, Zhe Wu, Peng Zhou, Hengduo Li, and Xitong Yang, for heated discussions in a windowless yet cozy room, timely free food updates, and sweat shed in gyms.

My eating buddies when office mates are absent: Bor-Chun Chen, Mingfei Gao, Shiyi Lan, Luyu Yang, for always wondering what to eat and card/board games after meals.

My CG group mates: Lihong He, Chaoran Gao, Xin Huang, Chenghong Ma, Jie Lin, Xinyun Fan and Xieran Lin, for hundreds of messages for me to read when I wake up in the morning, and drinks we had over the years.

My parents: Jianhua Wu and Li Luo, for your trust, love and support every step along the way.

Chen Chen, for countless FaceTime calls and always standing by my side.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

The exponential growth of visual media, in the forms of images and videos, is astounding: hundreds of millions of new images are being indexed by TinEye every month [1] and a million minutes of video content will cross the network by 2021 estimated by Cisco [2]. Such growth demands systems that can perform automated understanding of visual data both effectively and efficiently. For example, online shopping websites need to automatically organize product images in order to provide recommendation services; automatic analysis of high-level semantics in videos like actions and events can facilitate web video search and management; real-time recognition of objects in complicated environments like pedestrians and obstacles is crucial in delay-sensitive applications such as autonomous driving and robotic navigation. Although significant progress has been made in recent years, algorithms developed so far still fail to generalize to real-world applications.

Recent advances in computer vision tasks like image recognition and object detection have been driven by high-capacity deep neural networks, particularly Convolutional Neural Networks (CNNs) with hundreds of layers, trained in a supervised manner requiring clean and massive human annotations. This poses two significant

[1] https://www.tineye.com/faq#count

[2] https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html

challenges: The increased depth in CNNs that leads to significant improvements over competitive benchmarks at the same time, limits their deployment in real-world scenarios due to high computational cost, especially for applications on mobile devices and delay-sensitive systems, where inputs need to be processed in real-time as they arrive. In addition, the need to collect millions of human labeled samples for training prevents such approaches to scale, especially for fine-grained image understanding tasks. For example, consider semantic segmentation—the task of assigning a class label to each pixel in an image—a critical component in autonomous driving for pedestrian, road and car detection. Existing approaches require samples with variations in lighting, seasons and camera poses, and their corresponding pixel-level annotations as labels to train a robust model. This becomes extremely expensive and unwieldy at scale when generalizing to a new application.

To address these issues, we present approaches for image and video understanding with constrained resources, in the forms of computational resources and annotation resources. In particular, we introduce methods that focus on: (1) investigating dynamic computation frameworks that manage the trade-off between accuracy and computational complexity; (2) deriving robust representations with minimal human supervision through exploring context relationships and reusing shared information across environments.

More specifically, deep neural networks have become ubiquitous in computer vision, and the ability to do faster inference has emerged as a highly desirable characteristic, particularly in real-time scenarios, such as applications on mobile devices, robotic navigation, *etc.* The rising need has motivated a plethora of work to study the acceleration of CNN models. Most methods focus on reducing the parameters of pre-trained models with model compression. However, these approaches though effective will result in a set of fixed parameters (pruned or binarized) for all images. It is worth noting that human perception system is capable of adaptively allocating time for visual recognition. For example, a single glimpse is usually sufficient to recognize most objects and scenes while more time and attention will be used to clearly understand objects with occlusion or complicated scenes. Therefore, it makes intuitive sense that computational resources should adaptively vary with inputs.

In the first part, we introduce BlockDrop, an approach for faster inference in residual networks (ResNet) by selectively choosing residual blocks to evaluate in a learned and optimized manner conditioned on input images. In particular, we trained a policy network to predict blocks to drop in a pretrained ResNet while trying to retain the prediction accuracy. The ResNet is further jointly finetuned to produce smooth feature representations tailored for block dropping behavior. Built upon a ResNet-101 model, our method achieves a speedup of 20% on average, going as high as 36% for some images, while maintaining the same 76.4% top-1 accuracy on ImageNet. Further, learned BlockDrop policies for easy images with clearly visible objects utilize fewer residual blocks compared to the difficult images that contain other occluding or background objects.

We then generalize the idea of computational computation from images to videos by presenting AdaFrame, which adaptively observes relevant frames on a per-input basis for fast video recognition. It leverages a global memory to assist a Long Short-Term Memory network to provide context information to determine relevant frames. AdaFrame is optmized with policy gradient methods, and at each time step it produces a prediction, jumps to a frame it considers informative to observe, and generates the utility, i.e., expected future rewards, of seeing more frames. During testing, AdaFrame uses predicted utilities to achieve adaptive lookahead inference such that the overall computational costs are reduced without incurring a decrease in accuracy. Extensive experiments are conducted on two large-scale video benchmarks, FCVID and AvtivityNet. AdaFrame matches the performance of using all frames with only 8.21 and 8.65 frames on FCVID and AvtivityNet, respectively. We further qualitatively demonstrate learned frame usage can indicate the difficulty of making classification decisions; easier samples need fewer frames while harder ones require more, both at instance-level within the same class and at class-level among different categories.

Since AdaFrame searches through the entire video, it assumes access to the whole video beforehand, preventing the method to be used in online settings. To address this issue, we present LiteEval, a coarse-to-fine framework for both online and offline scenarios, with an aim to achieve resource efficient video recognition. More specifically, LiteEval operates on decent yet computationally efficient features by default, and it dynamically determines whether to compute better yet computationally expensive features for incoming video frames with a fully differentiable gating module. Extensive experiments are conducted on two large-scale video benchmarks, FCVID and ActivityNet, and the results demonstrate LiteEval requires substantially less computation while offering excellent classification accuracy for both online and offline predictions.

It worth noting that one of main limitation of current fully supervised training mechanism, although widely adopted, for deep neural networks is its dependence on manually labeled ground truth annotations. Thus, the ability to scale such approaches up to thousands if not tens of thousands of categories, containing millions or billions of samples, is limited due to insurmountable annotation efforts. We tackle this problem from two aspects: learning feature representations with weak supervision; unsupervised domain adaptation using synthetic data.

To reduce the need of using manual annotation, we use spatial context in images as a supervisory signal to learn visual representations. In particular, we introduce a spatial context network by training it to predict a feature representation of one image patch from another image patch, within the same image, conditioned on their real-valued relative spatial offset. In other words, given the same input patch and different spatial offsets it learns to predict different contextual representations (*e.g.*, given a patch depicting a side-view of a car and a horizontal offset, the network may output a patch representation of another car; however, the same input patch with a vertical offset may result in a patch representation of a plane). Once the SCN is optimized, we plug them as backbone networks for different tasks. We build our spatial context networks on top of standard pre-trained deep architectures and, among other things, show that we can achieve improvements (with no additional explicit supervision) over the original models in object categorization and detection on VOC2007.

Furthermore, another alternative to mitigating the need for manual annotations is to learn from synthetic data rendered by modern computer graphics tools (*e.g.*, video game engines), where ground-truth labels are readily available. However, models trained on synthetic data usually suffer from poor generalization when exposed to novel realistic samples. Learning a discriminative model that reduces the disparity between training and testing distributions is typically known as domain adaptation; a more challenging setting is unsupervised domain adaptation that aims to bridge the gap without accessing labels of the testing domain during training. Much recent and concurrent work relies on adversarial training to align feature distributions, which is known to be difficult for optimization. We propose Dual Channel-wise Alignment Networks (DCAN), a lightweight framework to reduce domain shift for unsupervised scene adaptation. DCAN performs channel-wise feature alignment in both the image generator for synthesizing photo-realistic samples, appearing as if drawn from the target set, and the segmentation network, which simultaneously normalizes feature maps of source images. We conduct extensive experiments by transferring models learned on synthetic segmentation datasets to real urban scenes, and demonstrate the effectiveness of DCAN over state-of-the-art methods and its compatibility with modern segmentation networks.

Finally, we discuss several future directions as the continuation of thesis research.

Chapter 2: BlockDrop: Dynamic Inference Paths in Residual Networks

2.1 Introduction

Deep neural networks are now ubiquitous in computer vision owing to their recent successes in several important tasks. However, great strides in accuracy have been accompanied by increasingly complex and deep network architectures. This presents a problem for domains where fast inference is essential, particularly in delay-sensitive and real-time scenarios such as autonomous driving, robotic navigation, or user-interactive applications on mobile devices.

Most existing work pursues model compression techniques to speed up a deep network [5, 6, 7, 8, 9, 10, 11, 12, 13]. While significant speed-ups are possible, the approach yields a one-size-fits-all network that requires the same fixed set of features to be extracted for all novel images, no matter their complexity. In contrast, an important feature of the human perception system is its ability to adaptively allocate time and scrutiny for visual recognition [14]. For example, a single glimpse is sufficient to recognize some objects and scenes, whereas more time and attention is required to clearly understand occluded or complicated ones [15].

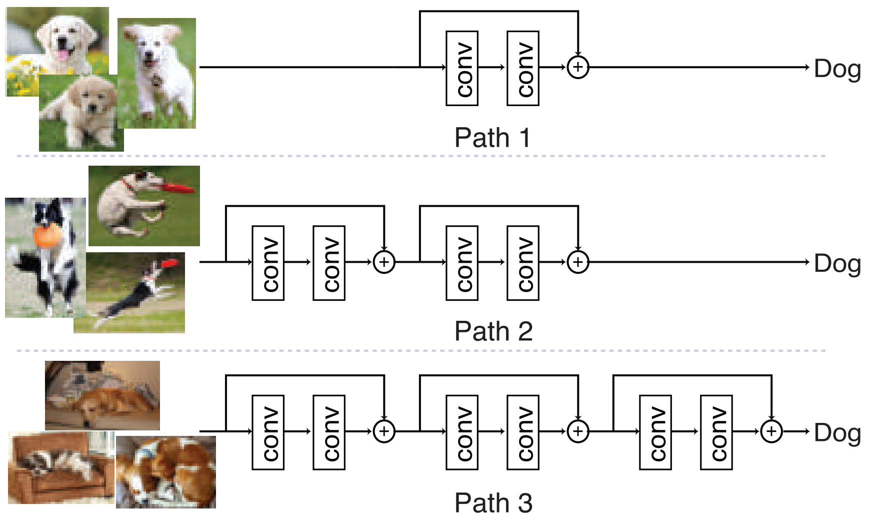


Figure 2.1: **A conceptual overview of BlockDrop.** Rather than execute all blocks of a ResNet, our approach learns a policy to select the minimal configuration of blocks that is needed to correctly classify a given input image. The resulting instance-specific paths in the network not only reflect the image's difficulty (easier samples use fewer blocks) but also encode meaningful visual information (patterns of blocks correspond to clusters of visual features).

In this spirit, we explore the problem of dynamically allocating computation across a deep network. In particular, we consider Residual Networks (ResNet) [16] both due to their strong track record for recognition tasks [16, 17, 18] as well as their tolerance to removal of layers [19]. ResNets are composed of *residual blocks*, consisting of two or more convolutional layers and skip-connections, which enable direct paths between any two residual blocks.

These skip-connections make ResNets behave like ensembles of relatively shallow networks, and hence the removal of a certain residual block generally has only a modest impact on performance [19]. However, the preliminary study of block dropping in ResNets [19] applies a global, manually defined dropping scheme (the same blocks for all images), which leads to increased errors when more blocks are dropped.

We propose to learn optimal block dropping strategies that simultaneously preserve both prediction accuracy and minimal block usage based on image-specific decisions. When a novel input is presented to the network trained for recognition, a *dynamic inference path* is followed, selectively choosing which blocks to compute for that instance. See Figure 2.1. The approach not only improves computational efficiency during inference (*i.e.*, for a similar prediction accuracy, being able to drop more residual blocks than a static global scheme), but also facilitates further insights into ResNets, *e.g.*, whether different blocks encode information about objects, whether the computation needed to classify depends on the difficulty level of the example.

To this end, we introduce *BlockDrop*, a reinforcement learning approach to derive instance-specific inference paths in ResNets. The main idea is to learn a model (referred to as the *policy network*) that, given a novel input image, outputs the posterior probabilities of all the binary decisions for dropping or keeping each block in a pretrained ResNet. The policy network is trained using curriculum learning to maximize a reward that incentivizes the use of as few blocks as possible while preserving the prediction accuracy. In addition, the pretrained ResNet is further jointly finetuned with the policy network to produce feature transformations tailored for block dropping behavior. Our approach can be seen as an instantiation of associative reinforcement learning [20] where all the decisions are taken in a single step given the context (*i.e.*, the input instance)[1]; this makes policy execution lightweight and scalable to very deep networks.

We conduct extensive experiments on CIFAR [22] and ImageNet [23]. BlockDrop achieves 93.6% and 73.7% accuracy using just 33% and 55% of blocks in a pretrained ResNet-110 on CIFAR-10 and CIFAR-100, respectively, outperforming state-of-the-art methods [10, 24, 25, 26] by clear margins. Furthermore, BlockDrop speeds up a ResNet-101 model on ImageNet by 20% while maintaining the same 76.4% top-1 accuracy [2]. Qualitatively, we observe that the dropping policies learned with BlockDrop are correlated with the visual patterns in the images, *e.g.*, within the "orange" class, images containing a pile of oranges take an inference path that is different from that taken by the close-up images of oranges. Furthermore, BlockDrop policies for *easy* images with clearly visible objects utilize fewer residual blocks compared to the *difficult* images that contain other occluding or background objects. Note that although our analysis in this chapter is focused on vanilla ResNets, our approach could also be applied to other recently proposed ResNet variants such as ResNeXt [27] or Multi-Residual Networks [28], as well as other tasks beyond image classification.

2.2 Related Work

Layer Dropping in Residual Networks. Dropping layers in residual networks has been used as a regularization mechanism, similar to Dropout [29] or DropCon-

[1]It can also be seen as contextual bandits [21] although we do not operate in an online setting which has an objective of minimizing the *regret*.

[2]`https://goo.gl/EwHQcq`

nect [30], for *training* very deep networks (*e.g.*., over 1000 layers) with stochastic depth [31]. Unlike our method, residual layer dropping in stochastic depth networks happens only during the training stage, but at *test time* the layers remain fixed. Veit *et al.* [19] show that ResNets are resilient to layer dropping at test time, which motivates our approach; however, they do not provide a way to dynamically choose which layers could be removed from a network without sacrificing accuracy. More recently, Huang and Wang [32] propose a method for selecting a subset of residual blocks to be executed based on a sparsity constraint. In contrast to these approaches, we propose an *instance-specific* residual block removal scheme to speed up ResNets during inference.

Model Compression. The need to deploy top-performing deep neural network models on mobile devices motivates techniques that can effectively reduce the storage and computational costs of such networks, including knowledge distillation [5, 6, 33], low-rank factorization [7, 8, 34], filter pruning [9, 10, 35, 36], quantization [11, 12, 13], compression with structured matrices [37, 38], network binarization [39, 40, 41], and hashing [42]. Efficient network architectures such as *SqueezeNet* [43] and *MobileNet* [44] have also been explored for training compact deep nets. In contrast to this line of work where the same amount of computation is applied to all images, we focus on efficient inference by dynamically choosing a subset of blocks to be executed *conditioned on the input image*. More importantly, our method is *complementary* to these model compression techniques: the residual blocks that are kept for evaluation can be further pruned for even greater speed up.

Conditional Computation. Several *conditional computation* methods have been proposed to dynamically execute different modules of a network model on a per-example basis [45, 46]. Sparse activations in combination with gating functions are usually adopted to selectively turn on and off a subset of modules based on the input. These gating functions can be learned with reinforcement learning [46, 47, 48]. These models typically associate a reward with a series of decisions computed after each layer/path; the resulting policy execution overhead makes it expensive to scale them up to very deep models with hundreds or thousands of layers. In contrast, our policy network makes all routing decisions in a *single step*, resulting in lower overhead cost for the routing itself and thus larger computational savings. Reinforcement learning has also been applied for dynamic feature prioritization in images [49] and video [1, 50], actively deciding which frames or image regions to visit next. These techniques could be used in tandem with our approach.

Early Prediction. Early prediction models, are a class of conditional computation models that exit once a criterion (*e.g.*, sufficient confidence for classification) is satisfied at early layers. Cascade detectors [51, 52] are among the earliest methods that exploit this idea in computer vision, often relying on handcrafted control decisions learned separately from visual features. More recently, joint learning of features and early decisions has been studied for deep neural networks. Teerapittayanon *et al.* [53] propose *BranchyNet*, a network composed of branches at each layer to make early classification decisions. Similarly, Adaptive Computation Time (ACT) [25] augments an RNN with a halting unit whose activation determines the probability that computation should continue.

Figurnov *et al.* [24] further extend this idea to the spatial domain in ResNets by applying ACT to each spatial position of multiple image blocks. Like our work, their formulation identifies instance-specific ResNet configurations, but it only allows configurations that use early, contiguous blocks in each predefined segment of the ResNet. These early blocks usually encode low-level features in high-dimensional feature maps, and may lack the discriminative power required for the task. This issue can be mitigated by using images at different scales [54, 55], but at a higher computational cost. Instead, we allow *any* block to contribute to our network, allowing for a much higher variability in potential ResNet configurations and policies.

2.3 Approach

Given a test image, our goal is to find the best configuration of computational blocks in a pretrained ResNet model, such that a minimum number of blocks is used, without incurring a decrease in classification accuracy. Treating the task of finding this configuration as a search problem quickly becomes intractable for deeper models as the number of potential configurations grows exponentially with the number of blocks. Learning a soft-attention mask over the blocks also presents problems, namely the difficulty of converting this mask into binary decisions which would require carefully handcrafted thresholds. In addition, such a thresholding operation is non-differentiable, making it non-trivial to directly adopt a supervised learning framework.

We therefore leverage *policy search methods* from reinforcement learning to derive the optimal block dropping schemes that encourage correct predictions with minimal block usage. To this end, we first revisit the architecture of ResNet in Sec. 2.3.1, and discuss why it is a good fit for block dropping. Then we introduce our policy network in Sec. 2.3.2, which learns to dynamically select inference paths conditioned on the input image. Finally, we present the training algorithm of our model in Sec. 2.3.3.

2.3.1 Pretrained Residual Networks

ResNets consist of multiple stacked *residual blocks* which are essentially regular convolutional layers that are bypassed by identity skip-connections. If we denote the input to the i-th residual block as y_i, and the function represented by its residual block as $\mathcal{F}_i$, the output of this residual block is given by: $y_{i+1} = \mathcal{F}_i(y_i) + y_i$, which is directly fed as input to the next residual block.

The presence of identity skip-connections induces direct paths between any two residual blocks, and hence top layers in the network are able to access information from bottom layers during a forward pass while gradients can be directly passed from higher layers to lower layers in the back-propagation phase. Veit *et al.* [19] demonstrated that removing (or dropping) a residual block at test time (*i.e.*, having $y_{i+1} = y_i$) does not lead to a significant accuracy drop. This behavior is due to the fact that ResNets can be viewed as an ensemble of many paths—as opposed to single-path models like AlexNet [56] and VGGNet [57]—and so information can be preserved even with the deletion of paths.

The results in [19] suggest that different blocks *do not share strong dependencies*. However, the study also shows classification errors do increase when more blocks are removed from the model during inference. We contend this is the result of their adopting a global dropping strategy for all images. We posit the best dropping schemes, which lead to correct predictions with the minimal number of blocks, must be instance-specific.

2.3.2 Policy Network for Dynamic Inference Paths

The *configurations* in the context of ResNets represent decisions to keep/drop each block, where each decision to drop a block corresponds to removing a subset of paths from the network. We refer to these decisions as our *dropping strategy*. To derive the optimal dropping strategy given an input instance, we develop a *policy network* to output a binary *policy vector*, representing the actions to keep or drop a block in a pretrained ResNet. During training, a reward is given considering both block usage and prediction accuracy, which is generated by running the ResNet with only active blocks in the policy vector. See Figure 6.2 for an overview.

Unlike standard reinforcement learning, we train the policy to predict *all actions at once*. This is essentially a single-step Markov Decision Process (MDP) given the input state and can also be viewed as contextual bandit [21] or associative reinforcement learning [20]. We examine the positive impact of this design choice on scalability in Sec. 2.4.2.

Formally, given an image $\mathbf{x}$ and a pretrained ResNet with K residual blocks, we define a policy of block-dropping behavior as a K-dimensional Bernoulli distribution:

$$\pi_{\mathbf{W}}(\mathbf{u}|\mathbf{x}) = \prod_{k=1}^{K} \mathbf{s}_k^{\mathbf{u}_k}(1-\mathbf{s}_k)^{1-\mathbf{u}_k} \tag{2.1}$$

$$\mathbf{s} = f_{pn}(\mathbf{x}; \mathbf{W}), \tag{2.2}$$

where f_{pn} denotes the *policy network* parameterized by weights $\mathbf{W}$ and $\mathbf{s}$ is the output of the network after the $\sigma(x){=}\frac{1}{1+e^{-x}}$ function. We choose the architecture of f_{pn} (details below in Sec. 2.4) such that the cost of running it is negligible compared to ResNet, *i.e.*, so that policy execution overhead remains low. The k-th entry of the vector, $\mathbf{s}^k \in [0,1]$, represents the likelihood of its corresponding residual block in the original ResNet being dropped. An action $\mathbf{u} \in \{0,1\}^K$ is selected based on $\mathbf{s}$. Here, $\mathbf{u}^k = 0$ and $\mathbf{u}^k = 1$ indicate dropping and keeping the k-th residual block, respectively.

Only the blocks that are not dropped according to $\mathbf{u}$ will be evaluated in the forward pass. To encourage both correct predictions as well as minimal block usage, we associate the actions taken with the following reward function:

$$R(\mathbf{u}) = \begin{cases} 1 - (\frac{|\mathbf{u}|_0}{K})^2 & \text{if correct} \\ \\ -\gamma & \text{otherwise.} \end{cases} \tag{2.3}$$

Here, $(\frac{|\mathbf{u}|_0}{K})^2$ measures the percentage of blocks utilized; when a correct prediction is produced, we incentivize block dropping by giving a larger reward to a policy that uses fewer blocks. We penalize incorrect predictions with γ, which controls the trade-off between efficiency (block usage) and accuracy (*i.e.*, a larger value leads to more correct, but less efficient policies). We use this parameter to vary the *operating point* of our model, allowing different models to be trained depending on the target budget constraint. Finally, to learn the optimal parameters of the policy network, we maximize the following expected reward:

$$J = \mathbb{E}_{\mathbf{u}\sim\pi_{\mathbf{W}}}[R(\mathbf{u})]. \tag{2.4}$$

In summary, our model works as follows: f_{pn} is used to decide which blocks of the ResNet to keep conditioned on the input image, a prediction is generated by running a forward pass with the ResNet using *only* these blocks, and a reward is observed based on correctness and efficiency.

2.3.3 Training the BlockDrop Policy

Expected gradient. To maximize Eqn. 2.4, we utilize policy gradient [20], one of the seminal policy search methods [58], to compute the gradients of J. In contrast to typical reinforcement learning methods where policies are sampled from a multinomial distribution [20], our policies are generated from a K-dimensional Bernoulli distribution. With $\mathbf{u}_k \in \{0, 1\}$, the gradients can be derived similarly as:

$$\begin{aligned}
\nabla_{\mathbf{W}} J &= \mathbb{E}[R(\mathbf{u})\nabla_{\mathbf{W}} \log\, \pi_{\mathbf{W}}(\mathbf{u}|\mathbf{x})] \\
&= \mathbb{E}[R(\mathbf{u})\nabla_{\mathbf{W}} \log \prod_{k=1}^{K} \mathbf{s}_k^{\mathbf{u}_k}(1-\mathbf{s}_k)^{1-\mathbf{u}_k}] \\
&= \mathbb{E}[R(\mathbf{u})\nabla_{\mathbf{W}} \sum_{k=1}^{K} \log[\mathbf{s}_k\mathbf{u}_k + (1-\mathbf{s}_k)(1-\mathbf{u}_k)]],
\end{aligned} \tag{2.5}$$

where again $\mathbf{W}$ denotes the parameters of the policy network. We approximate the expected gradient in Eqn. 2.5 with Monte-Carlo sampling using all samples in a mini-batch. These gradient estimates are unbiased, but exhibit high variance [20]. To reduce variance, we utilize a self-critical baseline $R(\tilde{\mathbf{u}})$ as in [59] , and rewrite Eqn. 2.5 as:

$$\nabla_{\mathbf{W}} J = \mathbb{E}[A\nabla_{\mathbf{W}} \sum_{k=1}^{K} \log[\mathbf{s}_k\mathbf{u}_k + (1-\mathbf{s}_k)(1-\mathbf{u}_k)]], \tag{2.6}$$

where $A = R(\mathbf{u}) - R(\tilde{\mathbf{u}})$ and $\tilde{\mathbf{u}}$ is defined as the maximally probable configuration under the current policy, $\mathbf{s}$: *i.e.*, $\mathbf{u}_i = 1$ if $\mathbf{s}_i > 0.5$, and $\mathbf{u}_i = 0$ otherwise [59].

We further encourage exploration by introducing a parameter α to bound the distribution $\mathbf{s}$ and prevent it from saturating, by creating a modified distribution $\mathbf{s}'$:

$$\mathbf{s}' = \alpha \cdot \mathbf{s} + (\mathbf{1} - \alpha) \cdot (\mathbf{1} - \mathbf{s}).$$

This bounds the distribution in the range $1 - \alpha \leq \mathbf{s}' \leq \alpha$, from which we then sample the policy vector.

Curriculum learning. Policy gradient methods are typically extremely sensitive to their initialization. Indeed, we found that starting from a randomly initialized policy and optimizing for both accuracy and block usage is not effective, due the extremely large dimension of the search space, which scales exponentially with the total number of blocks (there are 2^K possible on/off configurations of the blocks). Note that in contrast with applications such as image captioning where ground-truth action sequences (captions) can be used to train an initial policy [59], here no such "expert examples" are available, other than the standard single execution path that executes all blocks.

Therefore, to efficiently search for good action sequences, we take inspiration from the idea of curriculum learning [45]. During epoch t, for $1 \leq t < K$, we keep the first $K - t$ blocks on, and learn a policy only for the last t blocks. As t increases, the activity of more blocks are optimized, until finally all blocks are included (*i.e.*, when $t \geq K$). Using this approach, the activation of each block is first optimized according to unmodified input features in order to assess the utility of the block, and then is gradually exposed to increasingly different feature inputs as t increases and the policy for the last t blocks is jointly trained. This procedure is efficient, and it is effective at identifying and removing blocks that are redundant for the input data instance being considered. It is similar in spirit to [59, 60] that gradually exposes sequences when training with REINFORCE for text generation.

Joint finetuning. After curriculum learning, our policy network is able to identify which residual blocks in the original ResNet to drop for a given input image. Though the policy network is trained to preserve accuracy as much as possible, removing blocks from the pre-trained ResNet will inevitably result in a mismatch between training and testing conditions. We therefore jointly finetune the ResNet with the policy network, so that it can adapt itself to the learned block dropping behavior. The principle of our joint training procedure is similar to that of stochastic depth [31], with the exception that the drop rates are not fixed, but are instead controlled by the policy network. Alg. 1 presents the complete training procedure for our framework.

2.4 Experiments

2.4.1 Experimental Setup

Datasets and evaluation metrics. We evaluate our method on three benchmarks: CIFAR-10, CIFAR-100 [22], and ImageNet (ILSVRC2012) [23]. The CIFAR datasets consist of 60,000 32×32 colored images, with 50,000 images for training and 10,000 for testing. They are labeled for 10 and 100 classes for CIFAR-10 and CIFAR-100, respectively. Performance is measured by classification accuracy. ImageNet

Algorithm 1 The pseudo-code for training our network.

```
Input: An input image $\mathbf{x}$ and its label
 1: Initialize the weights of policy network $\mathbf{W}$ randomly
 2: Set epochs for curriculum learning and joint finetuning to $M^{cl}$ and $M^{ft}$, respectively;
    and set $\alpha$
 3: for $t \leftarrow 1$ to $M^{cl}$ do
 4:     $\mathbf{s} \leftarrow f_{pn}(\mathbf{x}; \mathbf{W})$
 5:     $\mathbf{s} \leftarrow \alpha \cdot \mathbf{s} + (\mathbf{1} - \alpha) \cdot (\mathbf{1} - \mathbf{s})$
 6:     if $t < K$ then
 7:         set $\mathbf{s}_{[1:K-t]} = 1$                                   ▷ curriculum training
 8:     end if
 9:     $\mathbf{u} \sim$ Bernoulli($\mathbf{s}$)
10:     Execute the ResNet according to $\mathbf{u}$
11:     Evaluate reward $R(\mathbf{u})$ with Eqn. 3.3
12:     Back-propagate gradients computed with Eqn. 2.6
13: end for
14: for $t \leftarrow 1$ to $M^{ft}$ do
15:     Jointly finetune ResNet and policy network
16: end for
```

		CIFAR-10				CIFAR-100			
		Acc	K	Acc (*ft*)	K (*ft*)	Acc	K	Acc (*ft*)	K (*ft*)
ResNet-32	FirstK	16.6	10	84.3	7	23.3	13	66.5	14
	RandomK	20.5	10	88.9	7	38.3	13	67.6	14
	DistributeK	23.4	10	90.2	7	31.9	13	66.7	14
	Ours	**88.6**	**9.4**	**91.3**	**6.9**	**58.3**	**12.4**	**68.7**	**13.1**
	Full ResNet	92.3	15	92.3	15	69.3	15	69.3	15
ResNet-110	FirstK	13.3	21	71.3	17	63.5	50	57.9	31
	RandomK	14.5	21	90.1	17	66.3	50	68.4	31
	DistributeK	13.0	21	92.7	17	49.6	50	69.9	31
	Ours	**75.4**	**20.1**	**93.6**	**16.9**	**72.1**	**49.1**	**73.7**	**30.2**
	Full ResNet	93.2	54	93.2	54	72.2	54	72.2	54

Table 2.1: **Accuracy and block usage with our policies vs. heuristic baselines**, with and without jointly finetuning (ft) for all methods. For fair comparisons, K is selected based on the average block usage of our method, and this can be different before and after finetuning. Note that the average value of K for our method is reported here for brevity. It is determined dynamically per image, and can be as low as 3 (out of 54) in ResNet-110 on CIFAR-10.

contains 1.2M training images labeled for 1,000 categories. We test on the validation set of 50,000 images and report top-1 accuracy.

Pretrained ResNet. For CIFAR-10 and CIFAR-100, we experiment with two ResNet models that achieve promising results. In particular, ResNet-32 and ResNet-110 start with a convolutional layer followed by 15 and 54 residual blocks, respectively. These residual blocks, each of which contains two convolutional layers, are evenly distributed into 3 segments with down-sampling layers in between. Finally, a fully-connected layer with 10/100 neurons is applied. See [16] for details. For ImageNet, we adopt ResNet-101 with a total of 33 residual blocks, organized into four segments (*i.e.*, $[3, 4, 20, 3]$). Here, each residual block contains three convolutional layers based on the *bottleneck* design [16] for computational efficiency. These models are pretrained to match state-of-the-art performance on the corresponding datasets when run without our policy network.

Policy network architecture. For our policy network, we use ResNets with a fraction of the depth of the base model. For CIFAR, we use a ResNet with 3 blocks (equivalently ResNet-8), while for ImageNet, we use a ResNet with 4 blocks (equivalently ResNet-10). In addition, we downsample images to 112×112 as the input of the policy network for ImageNet experiments. The computation required for the policy network is 4.8% and 3.0% of the total ResNet computation for the CIFAR (ResNet-110) and ImageNet (ResNet-101) models respectively, making policy computations negligible (it takes about 0.5 ms per image on average for ImageNet). While a recurrent model (*e.g.*, LSTM) could also serve as the policy network, we found a CNN to be more efficient with similar performance.

Implementations details. We adopt PyTorch for implementation and utilize ADAM as the optimizer. We set α to 0.8, learning rate to $1e-4$, and use a batch size of 2048 during curriculum learning. For joint finetuning, we adjust the batch size to 256 and 320 on CIFAR and ImageNet, respectively, and adjust the learning rate to $1e-5$ for ImageNet. Our code is available at `https://goo.gl/NqyNeN`.

2.4.2 Quantitative Results

Learned policies *v.s.* heuristics. We compare our block dropping strategy to the following alternative methods: (1) FirstK, which keeps only the first K residual blocks active; (2) RandomK, which keeps K randomly selected residual blocks active; (3) DistributeK, which evenly distributes K blocks across all segments. For all baselines, we choose K to match the average number of blocks used by BlockDrop, rounding up as needed. DistributeK allows us to see if feature combinations of different blocks learned by BlockDrop are better than features learned from the restricted set of early blocks of each segment. This setting resembles the allowable feature combinations from early stopping models applied to ResNets.

The results in Table 2.1 highlight the advantage of our instance-specific policy. On CIFAR-10, the learned policies give an accuracy of 88.6% and 75.4% using an average of 9.4 and 20.1 blocks from the original ResNet-32 and ResNet-110 respectively, outperforming the baselines by a large margin. Furthermore, the instance-specific nature of our method allows us to capture the inherent variance in the computational requirements of our dataset. We notice a wide distribution in block usage depending on the image. With ResNet-110, nearly 15% of the images use fewer than 10 blocks, with some images using as few as 3 blocks. This variance cannot be captured by any static policies. Similar trends are observed on CIFAR-100. This confirms that dropping residual blocks with policies computed in a learned manner is indeed significantly better than heuristic dropping behaviors. The fact that RandomK performs better than FirstK is interesting, suggesting the value of having residual blocks at different segments to learn feature representations at different scales.

Impact of joint finetuning. Next we analyze the impact of joint finetuning (cf. Sec. 2.3.3) for both our approach and the baselines, denoted *ft* in Table 2.1.

Joint finetuning further significantly improves classification accuracy using fewer (or almost the same) number of blocks. In particular, on CIFAR-10, it offers absolute performance gains of 2.7% and 18.2% using 2.5 and 3.2 *fewer* blocks with ResNet-32 and ResNet-110 respectively compared with curriculum training alone. Similarly, on CIFAR-100, joint finetuning improves accuracies and brings down block usage with ResNet-110. For ResNet-32, we observe 0.7 more blocks on average are used after finetuning, which might be due to the challenging nature of CIFAR-100 requiring more blocks to make correct predictions. Comparing ResNet-110 with ResNet-32, we observe that the computational speed-ups are more dramatic for deeper ResNets owing to the fact that there are more blocks with potentially diverse features to select from. When built upon ResNet-110, our method outperforms the pretrained model by 0.4% and 1.5% (absolute) using 31% and 55.9%

of the original blocks on CIFAR-10 and CIFAR-100, respectively. Additionally, we observe that some images use as few as 5 blocks for inference. These results confirm that joint finetuning can indeed assist the ResNet to adapt to the removal of blocks by refining its feature representations while maintaining its capacity for instance-specific variation.

BlockDrop *v.s.* state-of-the-art methods. We next compare BlockDrop to several techniques from the literature. We vary γ, which controls our algorithm's trade-off between block usage and accuracy, to get a range of models with varying computational requirements. We compute the average FLOPs utilized to classify each image in the test set; FLOPs are a hardware independent metric, allowing for fair comparisons across models. [3]

We compare to the following state-of-the-art methods [4]: (1) ACT and (2) SACT [24], (3) PFEC [10], (4) LCCL [26]. ACT and SACT learn a halting score at the end of each block, and exit the model when a high-confidence is obtained. PFEC and LCCL reduce the parameters of convolutional layers by either pruning or sparsity constraints, which is complementary to our method. Other model compression methods cited earlier do not report results on larger ResNet models, and hence are not available to compare here.

Figure 3.3 (a) presents the results on CIFAR. We observe that our best model offers 0.4% performance gain in accuracy (93.6% *v.s.* 93.2%) using 65% fewer FLOPs

[3]Note that we consider the multiply-accumulate operation as a two step process yielding two floating point operations and we only compute FLOPs for convolutional layers and linear layers as they account for most of the computation for inference.

[4]For ACT and SACT on CIFAR, we train models with the authors' code. For the rest, we compare to numbers in the respective papers.

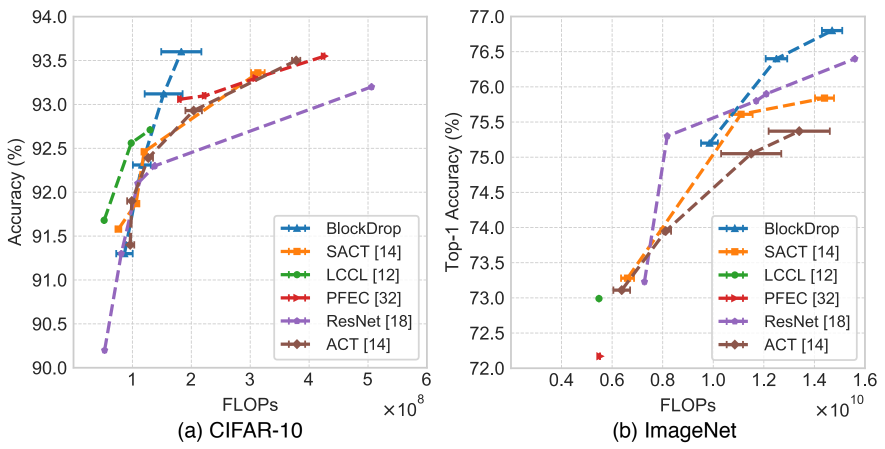


Figure 2.2: **FLOPs *v.s.* accuracy on CIFAR-10 and ImageNet**. Results compared to several state-of-the art methods. Error bars denote the standard deviation across images.

on average (1.73×10^8 *v.s.* 5.08×10^8) over the original ResNet-110 model. The performance gains might result from the regularization effect of dropping blocks when finetuning the network as in [31]. Compared to ACT and SACT, our method only requires 50% of the FLOPs to achieve the same level of precision (>93.0%). BlockDrop also exhibits a much higher variance in its FLOPs over other methods. Compared to SACT, this variance is 3 times larger, allowing some samples to achieve a speedup as high as 85% with correct predictions. Further, BlockDrop also outperforms PFEC [10] and LCCL [26], which are complementary compression techniques and can be utilized together with our framework to speed up convolution operations.

Figure 3.3 (b) presents the results for ImageNet. Compared with the original ResNet-101 model, BlockDrop again achieves slightly better results (76.8% *v.s.* 76.4%) with 6% speed up (1.47×10^{10} *v.s.* 1.56×10^{10} FLOPs). BlockDrop performs on par with the full ResNet with a 20% speed up (1.25×10^{10} *v.s.* 1.56×10^{10} FLOPs) when we relax γ slightly. This 20% acceleration without degradation in accuracy

		Time (ms)	Speed-up
ResNet-32	Full ResNet	7.71	–
	Ours-single	6.56	14.9%
	Ours-seq	9.92	-28.7%
ResNet-110	Full ResNet	24.1	–
	Ours-single	10.9	52.3%
	Ours-seq	29.1	-20.7%

Table 2.2: **Impact of our single-step policy inference on efficiency for CIFAR-10**. See text for details.

is quite promising. For example, in a high-precision image recognition service accepting 1 billion daily API calls, such a speedup would save around 1000 hours of computation on a single P6000 GPU (0.024 seconds/image).

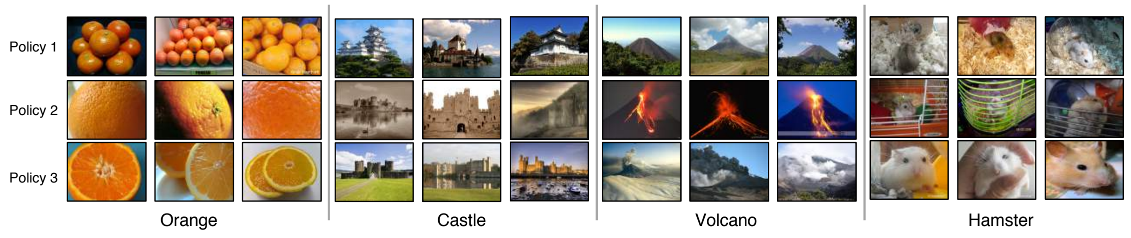


Figure 2.3: **Policies learned for four ImageNet classes, *volcano, orange, hamster* and *castle***. These policies correspond to a set of active paths in the ResNet, which seem to cater to different "states" of images of the particular class. For *volcano*, these include features like smoke, lava, *e.t.c.*, while for *orange* they include whether it is sliced/whole, quantity.

Efficiency advantage of single-step policy. The single-step design of our policy network—where the full dynamic inference path is computed without revisiting intermediate outputs of the network—has important efficiency advantages. In short, it permits lower policy execution overhead. To examine the impact empirically, we devised a variant of BlockDrop that uses traditional RL policy learning to instead make sequential decisions. In particular, the decision $\mathbf{a}_i \in \{0,1\}$ to drop or keep the i-th block is conditioned on the activations of its previous block, y_{i-1}. Unlike BlockDrop, where all the actions are predicted in one shot, this model predicts one action at a time, which is a typical reinforcement learning setting. We follow the procedure to generate the *halting scores* in [24], and arrive at an equivalent per-block *skipping score* according to:

$$\mathbf{p}_i = \texttt{softmax}(\widetilde{W}^i \texttt{pool}(y_{i-1}) + b^i),$$

where `pool` is a global average pooling operation. For fair comparisons, Ours-seq is compared to a BlockDrop model, which attains equivalent accuracy, with the same number of blocks. We select models of both variants that attain equivalent accuracy, with the same number of blocks. To ensure fair comparison, we run all three models on the same single NVIDIA P6000 GPU while disabling other processes.

Table 2.2 shows the results for CIFAR-10. We report the time per test image and the speed-up over the original ResNet run in entirety with no block dropping. This result confirms the efficiency advantage of our single-step design: to reach the same accuracy, we need much less overhead (e.g., less than 60% of the time required by the sequential variant). In fact, the sequential variant takes even *longer* to run than the original full ResNet models, yielding a negative speed-up. These results reaffirm our choice to compute all actions in one shot rather than compute them sequentially. They also stress the importance of accounting for any overhead a deep net speed-up scheme incurs to make its speed-up decisions.

2.4.3 Detailed Results on CIFAR-10 and ImageNet

We present detailed results of our method on CIFAR-10 (Table 2.3) and ImageNet (Table 2.4). We highlight the accuracy, block usage and speed up for variants of our model compared to full ResNets.

Network	FLOPs	Block Usage	Accuracy	Speed-up
ResNet-32	1.38E+08 ± 0.00E+00	15.0 ± 0.0	92.3	–
ResNet-110	5.06E+08 ± 0.00E+00	54.0 ± 0.0	93.2	–
DCAN-32 ($\gamma = 5$)	8.66E+07 ± 1.40E+07	6.9 ± 1.6	91.3	37.2%
DCAN-110 ($\gamma = 2$)	1.18E+08 ± 2.46E+07	10.3 ± 2.7	91.9	76.7%
DCAN-110 ($\gamma = 5$)	1.51E+08 ± 3.24E+07	13.8 ± 3.5	93.0	70.1%
DCAN-110 ($\gamma = 10$)	1.81E+08 ± 3.43E+07	16.9 ± 3.7	93.6	64.3%

Table 2.3: **Results of different architectures on CIFAR-10.** Depending on the base ResNet architecture, speedups ranging from 37% to 76% are observed with little to no degradation in performance.

Network	FLOPs	Block Usage	Accuracy	Speed-up
ResNet-72	1.17E+10 ± 0.00E+00	24.0 ± 0.0	75.8	–
ResNet-75	1.21E+10 ± 0.00E+00	25.0 ± 0.0	75.9	–
ResNet-84	1.34E+10 ± 0.00E+00	28.0 ± 0.0	76.1	–
ResNet-101	1.56E+10 ± 0.00E+00	33.0 ± 0.0	76.4	–
DCAN ($\gamma = 2$)	9.85E+09 ± 3.34E+08	18.8 ± 0.8	75.2	36.9%
DCAN ($\gamma = 5$)	1.25E+10 ± 4.26E+08	24.8 ± 1.0	76.4	19.9%
DCAN ($\gamma = 10$)	1.47E+10 ± 4.02E+08	29.7 ± 0.9	76.8	5.7%

Table 2.4: **Results of different architectures on ImageNet.** DCAN is built upon ResNet-101, and can achieve around 20% speedup on average with $\gamma = 5$.

2.4.4 Qualitative Results

Finally, we provide qualitative results based on our learned policies. We investigate the visual patterns encoded in these learned policies and then analyze the relation between block usage and instance difficulty.

Visual patterns in policies. Intuitively, related images can be recognized by their similar characteristics (*e.g.*, low-level clues like texture and color). Here, we analyze similarity in terms of the *policies they utilize* by sampling dominant policies for each class and visualizing samples from them. Figure 2.3 shows samples utilizing three different policies for four classes. It can be clearly seen that images under the same policy are similar, and different policies encode different styles, although they all correspond to the same semantic concept. For example, the first inference path for the "orange" class caters to images containing a pile of oranges, and close up views of oranges activate the second inference path, while images containing slices of oranges are routed through the third inference path. These results indicate that different paths encode meaningful semantic visual patterns, based on the input images. While this happens in standard ResNets as well, all images necessarily utilize all the paths, and disentangling this information is not possible.

Instance difficulty. Instance difficulty is well understood in the context of prediction confidence, where easy and difficult examples are classified with high and low probabilities, respectively. Inspired by the above analysis that revealed interesting correlations between the inference policies and the visual patterns in the images, we try to characterize instance difficulty in terms of block usage. We hypothesize that simple examples (*e.g.* images with clear objects, without occlusions) require fewer computations to be correctly recognized. To qualitatively analyze the correlations between instance difficulty and block usage, we utilize learned policies that lead to high-confidence predictions for each class.

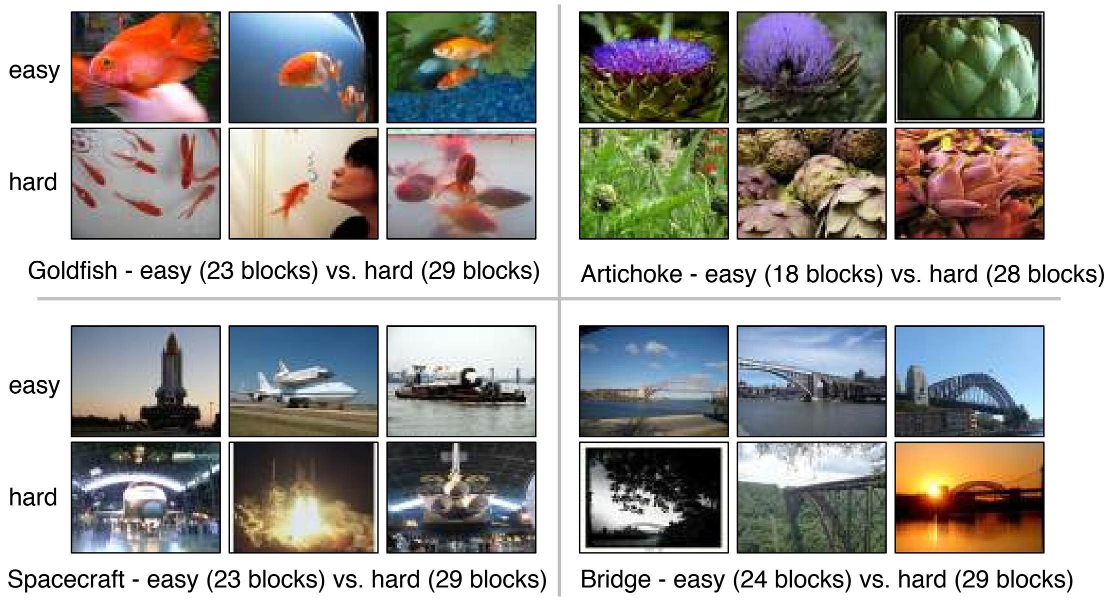

easy
hard
Goldfish - easy (23 blocks) vs. hard (29 blocks)
Artichoke - easy (18 blocks) vs. hard (28 blocks)
easy
hard
Spacecraft - easy (23 blocks) vs. hard (29 blocks)
Bridge - easy (24 blocks) vs. hard (29 blocks)

Figure 2.4: **Samples from ImageNet classes**. Easy and hard samples from *goldfish*, *artichoke*, *spacecraft* and *bridge* to illustrate how block usage translates to instance difficulty.

Figure 4.4 illustrates samples from ImageNet. The top row contains images that are correctly classified with the least number of blocks, while samples in the bottom row utilize the most blocks. We see that samples using fewer blocks are indeed easier to identify since they contain single frontal-view objects positioned in the center, while several objects, occlusion, or cluttered background occur in samples that require more blocks. This confirms our hypothesis that block usage is a function of instance difficulty. We stress that this "sorting" into easy or hard cases falls out automatically; it is learned by BlockDrop.

2.5 Conclusion

We presented BlockDrop, an approach for faster inference in ResNets by selectively choosing residual blocks to evaluate in a learned and optimized manner conditioned on inputs. In particular, we trained a policy network to predict blocks to drop in a pretrained ResNet while trying to retain the prediction accuracy. The ResNet is further jointly finetuned to produce smooth feature representations tailored for block dropping behavior. We conducted extensive experiments on CIFAR and ImageNet, observing considerable gains over existing methods in terms of the efficiency-accuracy trade-off. Further, we also observe that the policies learned encode semantic information in the images.

Chapter 3: AdaFrame: Adaptive Frame Selection for Fast Video Recognition

3.1 Introduction

The explosive increase of Internet videos, driven by the ubiquity of mobile devices and sharing activities on social networks, is phenomenal: around 300 hours of video are uploaded to YouTube every minute of every day! Such growth demands effective and scalable approaches that can recognize actions and events in videos automatically for tasks like indexing, summarization, recommendation, *etc.* Most existing work focuses on learning robust video representations to boost accuracy [61, 62, 63, 64], while limited effort has been devoted to improving efficiency [65, 66].

State-of-the-art video recognition frameworks rely on the aggregation of prediction scores from uniformly sampled frames [1], if not every single frame [67], during inference. While uniform sampling has been shown to be effective [62, 63, 64], the analysis of even a single frame is still computationally expensive due to the use of high-capacity backbone networks such as ResNet [16], ResNext [27], Inception-Net [68], *etc.* On the other hand, uniform sampling assumes information is evenly

[1]Here, we use frame as a general term, and it can be in the forms of a single RGB image, stacked RGB images (snippets), and stacked optical flow images.

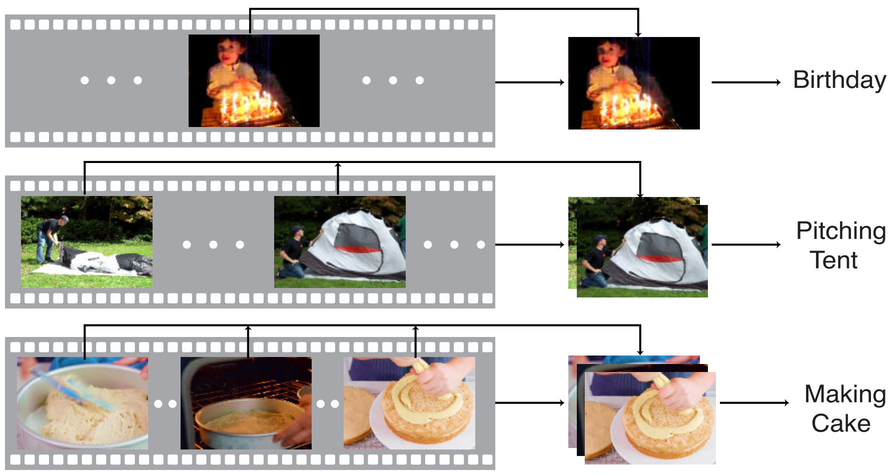


Figure 3.1: **A conceptual overview of AdaFrame.** AdaFrame aims to select a small number of frames to make correct predictions conditioned on different input videos so as to reduce the overall computational cost.

distributed over time, which could therefore incorporate noisy background frames that are not relevant to the class of interest.

It is also worth noting that the difficulty of making recognition decisions relates to the category to be classified—one frame might be sufficient to recognize most static objects (*e.g.*, "dogs" and "cats") or scenes (*e.g.*, "forests" or "sea") while more frames are required to differentiate subtle actions like "drinking coffee" and "drinking beer". This also holds for samples even within the same category due to large intra-class variations. For example, a "playing basketball" event can be captured from multiple view points (*e.g.*, different locations of a gymnasium), occur at different locations (*e.g.*, indoor or outdoor), with different players (*e.g.*, professionals or amateurs). As a result, the number of frames required to recognize the same event are different.

With this in mind, to achieve efficient video recognition, we explore how to automatically adjust computation within a network on a per-video basis such that—

conditioned on different input videos, a small number of informative frames are selected to produce correct predictions (See Figure 3.1). However, this is a particularly challenging problem, since videos are generally weakly-labeled for classification tasks, one annotation for a whole sequence, and there is no supervision informing which frames are important. Therefore, it is unclear how to effectively explore temporal information over time to choose which frames to use, and how to encode temporal dynamics in these selected frames.

In this chapter, we propose AdaFrame, a Long Short-Term Memory (LSTM) network augmented with a global memory, to learn how to adaptively select frames conditioned on inputs for fast video recognition. In particular, a global memory derived from representations computed with spatially and temporally downsampled video frames is introduced to guide the exploration over time for learning frame usage policies. The memory-augmented LSTM serves as an agent interacting with video sequences; at a time step, it examines the current frame, and with the assistance of global context information derived by querying the global memory, generates a prediction, decides which frame to look at next and calculates the utility of seeing more frames in the future. During training, AdaFrame is optimized using policy gradient methods with a fixed number of steps to maximize a reward function that encourages predictions to be more confident when observing one more frame. At testing time, AdaFrame is able to achieve adaptive inference conditioned on input videos by exploiting the predicted future utilities that indicate the advantages of going forward.

We conduct extensive experiments on two large-scale and challenging video benchmarks for generic video categorization (FCVID [69]) and activity recognition (ActivityNet [70]). AdaFrame offers similar or better accuracies measured in mean average precision over the widely adopted uniform sampling strategy, a simple yet strong baseline, on FCVID and ActivityNet respectively, while requiring 58.9% and 63.3% fewer computations on average, going as high as savings of 90.6%. AdaFrame also outperforms by clear margins alternative methods [1, 2] that learn to select frames. We further show that, among other things, frame usage is correlated with the difficulty of making predictions—different categories produce different frame usage patterns and instance-level frame usage within the same class also differs. These results corroborate that AdaFrame can effectively learn to generate frame usage policies that adaptively select a small number of relevant frames for classification for each input video.

3.2 Related Work

Video Analysis. Extensive studies have been conducted on video recognition [71]. Most existing work focuses on extending 2D convolution to the video domain and modeling motion information in videos [61, 62, 63, 64, 72]. Only a few methods consider efficient video classification [50, 65, 66, 73, 74]. However, these approaches perform mean-pooling of scores/features from multiple frames, either uniformly sampled or decided by an agent, to classify a video clip. In contrast, we focus on selecting a small number of relevant frames, whose temporal relations are modeled by an LSTM, on a per-video basis for efficient recognition. Note that our framework is also applicable to 3D CNNs; the inputs to our framework can be easily replaced with features from stacked frames. A few recent approaches attempt to reduce computation cost in videos by exploring similarities among adjacent frames [75, 76], while our goal is to selectively choose relevant frames based on inputs.

Our work is more related to [1] and [2] that choose frames with policy search methods [58]. Yeung *et al.* introduce an agent to predict whether to stop and where to look next through sampling from the whole video for action detection [1]. For detection, ground-truth temporal boundaries are available, providing strong feedback about whether viewed frames are relevant. In the context of classification, there is no such supervision, and thus directly sampling from the entire sequence is difficult. To overcome this issue, Fan *et al.* propose to sample from a predefined action set deciding how many steps to jump [2], which reduces the search space but sacrifices flexibility. In contrast, we introduce a global memory module that provides context information to guide the frame selection process. We also decouple the learning of frame selection and when to stop, exploiting predicted future returns as stop signals.

Adaptive Computation. Our work also relates to adaptive computation to achieve efficiency by deciding whether to stop inference based on the confidence of classifiers. The idea dates back to cascaded classifiers [52] that quickly reject easy negative sub-windows for fast face detection. Several recent approaches propose to add decision branches to different layers of CNNs to learn whether to exit the

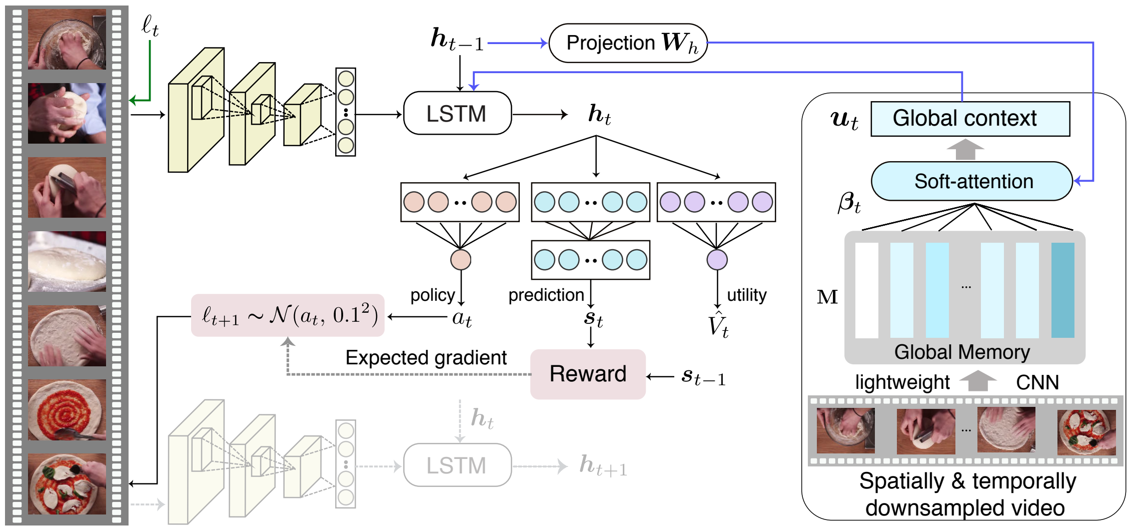


Figure 3.2: **An overview of the proposed AdaFrame framework**. A memory-agumented LSTM serves as an agent, interacting with a video sequence. At each time step, it takes features from the current frame, previous states, and a global context vector derived from a global memory to generate the current hidden states. The hidden states are used to produce a prediction, decides where to look next and calculates the utility of seeing more frames in the future. See texts for more details.

model [24, 53, 54, 55]. Graves introduce a halting unit to RNNs to decide whether computation should continue [25]. Related are also [77, 78, 79, 80, 81] that learn to drop layers in residual networks or learn where to look in images conditioned on inputs. In this chapter, we focus on adaptive computation for videos to adaptively select frames rather than layers/units in neural networks for fast inference.

3.3 Approach

Our goal is, given a testing video, to derive an effective frame selection strategy that produces a correct prediction while using as few frames as possible. To this end, we introduce AdaFrame, a memory-augmented LSTM (Section 3.3.1), to explore the temporal space of videos effectively with the guidance of context information from a global memory. AdaFrame is optimized to choose which frames to use on a per-video basis, and to capture the temporal dynamics of these selected frames. Given the learned model, we perform adaptive lookahead inference (Section 3.3.2) to accommodate different computational needs through exploring the utility of seeing more frames in the future.

3.3.1 Memory-augmented LSTM

The memory-augmented LSTM can be seen as an agent that recurrently interacts with a video sequence of T frames, whose representations are denoted as $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_T\}$. More formally, the LSTM, at the t-th time step, takes features of the current frame $\boldsymbol{v}_t$, previous hidden states $\boldsymbol{h}_{t-1}$ and cell outputs $\boldsymbol{c}_{t-1}$, as well as a global context vector $\boldsymbol{u}_t$ derived from a global memory $\mathbf{M}$ as its inputs, and produces the current hidden states $\boldsymbol{h}_t$ and cell contents $\boldsymbol{c}_t$:

$$\boldsymbol{h}_t,\, \boldsymbol{c}_t = \texttt{LSTM}([\boldsymbol{v}_t, \boldsymbol{u}_t],\, \boldsymbol{h}_{t-1},\, \boldsymbol{c}_{t-1}), \tag{3.1}$$

where $\boldsymbol{v}_t$ and $\boldsymbol{u}_t$ are concatenated. The hidden states $\boldsymbol{h}_t$ are further input into a prediction network f_p for classification, and the probabilities are used to generate a reward r_t measuring whether the transition from the last time step brings information gain. Furthermore, conditioned on the hidden states, a selection network f_s decides where to look next, and a utility network f_u calculates the advantage of seeing more frames in the future. Figure 6.2 gives an overview of the framework. In the following, we elaborate detailed components in the memory-augmented LSTM.

Global memory. The LSTM is expected to make reliable predictions and explore the temporal space to select frames guided by rewards received. However, learning where to look next is difficult due to the huge search space and limited capacity of hidden states [82, 83] to remember input history. Therefore, for each video, we introduce a global memory to provide context information, which consists of representations of spatially and temporally downsampled frames, $\mathbf{M} = [\boldsymbol{v}^s_1, \boldsymbol{v}^s_2, \ldots, \boldsymbol{v}^s_{T_d}]$. Here, T_d denotes the number of frames ($T_d < T$), and the representations are computed with a lightweight network using spatially downsampled inputs (more details in Sec. 4.3.1). This is to ensure the computational overhead of the global memory is small. As these representations are computed frame by frame without explicit order information, we further utilize positional encoding [84] to encode positions in the downsampled representations. To obtain global context information, we query the global memory with the hidden states of the LSTM to get an attention weight for each element in the memory:

$$z_{t,j} = (\boldsymbol{W}_h \boldsymbol{h}_{t-1})^\top PE(\boldsymbol{v}^s_j), \quad \boldsymbol{\beta}_t = \texttt{Softmax}(\boldsymbol{z}_t),$$

where $\boldsymbol{W}_h$ maps hidden states to the same dimension as the j-th downsampled feature $\boldsymbol{v}^s_j$ in the memory, PE denotes the operation of adding positional encoding to features, and $\boldsymbol{\beta}_t$ is the normalized attention vector over the memory. We can further derive the global context vector as the weighted average of the global memory: $\boldsymbol{u}_t = \boldsymbol{\beta}_t^\top \mathbf{M}$. The intuition of computing a global context vector with soft-attention as inputs to the LSTM is to derive a rough estimate of the current progress based on features in the memory block, serving as global context to assist the learning of which frame in the future to examine.

Prediction network. The prediction network $f_p(\boldsymbol{h}_t; \boldsymbol{W}_p)$ parameterized by weights $\boldsymbol{W}_p$ maps the hidden states $\boldsymbol{h}_t$ to outputs $\boldsymbol{s}_t \in \mathbb{R}^C$ with one fully-connected layer, where C is the number of classes. In addition, $\boldsymbol{s}_t$ is further normalized with `Softmax` to produce probability scores for each class. The network is trained with cross-entropy loss using predictions from the last time step T_e:

$$\mathcal{L}_{cls}(\boldsymbol{W}_p) = -\sum_{c=1}^{C} \boldsymbol{y}^c \log(\boldsymbol{s}^c_{T_e}), \tag{3.2}$$

where $\mathbf{y}$ is a one-hot vector encoding the label of the corresponding sample. In addition, we constrain $T_e \ll T$, since we wish to use as few frames as possible.

Reward function. Given the classification scores $\boldsymbol{s}_t$ of the t-th time step, a reward is given to evaluate whether the transition from the previous time step is useful—observing one more frame is expected to produce more accurate predictions. Inspired by [85], we introduce a reward function that forces the classifier to be more confident when seeing additional frames, taking the following form (when $t > 1$):

$$r_t = \max\{0,\, m_t - \max_{t' \in [0, t-1]} m_{t'}\}. \tag{3.3}$$

Here, $m_t = \boldsymbol{s}^{gt}_t - \max\{\boldsymbol{s}^{c'}_t | c' \neq gt\}$ is the margin between the probability of the ground-truth class (indexed by gt) and the largest probabilities from other classes, pushing the score of the ground-truth class to be higher than other classes by a margin. And the reward function in Eqn. 3.3 encourages the current margin to be larger than historical ones to receive a positive reward, which demands that the confidence of the classifier increases when seeing more frames. Such a constraint acts as a proxy to measure if the transition from the last time step brings additional information for recognizing target classes, as there is no supervision providing feedback about whether a single frame is informative.

Selection network. The selection network f_s defines a policy with a Gaussian distribution using fixed variance, to decide which frame to observe next, using hidden states $\boldsymbol{h}_t$ that contain information of current inputs and historical context. In particular, the network, parameterized by $\boldsymbol{W}_s$, transforms the hidden states to a 1-dimensional output $f_s(\boldsymbol{h}_t; \boldsymbol{W}_s) = a_t = \texttt{sigmoid}(\boldsymbol{W}_s^\top \boldsymbol{h}_t)$, as the mean of the location policy. Following [86], during training, we sample from the policy $\ell_{t+1} \sim \pi(\cdot|\boldsymbol{h}_t) = \mathcal{N}(a_t, 0.1^2)$, and at testing time, we directly use the output as the location. We also clamp ℓ_{t+1} to be in the interval of $[0, 1]$, so that it can be further transfered to a frame index multiplying by the total number of frames. It is worth noting that at the current time step, the policy searches through the entire time horizon and there is no constraint; it can not only jump forward to seek future informative frames but also go back to re-examine past information. We train the selection network to maximize the expected future reward:

$$J_{sel}(\boldsymbol{W}_s) = \mathbb{E}_{\ell_t \sim \pi(\cdot|\boldsymbol{h}_t; \boldsymbol{W}_s)} \left[\sum_{t=0}^{T_e} r_t \right]. \tag{3.4}$$

Utility network. The utility network, parameterized by $\boldsymbol{W}_u$, produces an output $f_u(\boldsymbol{h}_t; \boldsymbol{W}_u) = \hat{V}_t = \boldsymbol{W}_u^\top \boldsymbol{h}_t$ using one fully-connected layer. It serves as a critic to provide an approximation of expected future rewards from the current state, which is also known as the value function [20]:

$$V_t = \mathbb{E}_{\substack{\boldsymbol{h}_{t+1:T_e}, \\ a_{t:T_e}}} \left[\sum_{i=0}^{T_e - t} \gamma^i r_{t+i} \right], \tag{3.5}$$

where γ is the discount factor fixed to 0.9. The intuition is to estimate the value function V_t derived from empirical rollouts with the network output $\hat{V}_t$ to update policy parameters in the direction of performance improvement. More importantly, by estimating future returns, it provides the agent with the ability to look ahead, measuring the utility of subsequently observing more frames. The utility network is trained with the following regression loss:

$$\mathcal{L}_{utl}(\boldsymbol{W}_u) = \frac{1}{2} \|\hat{V}_t - V_t\|_2. \tag{3.6}$$

Optimization. Combining Eqn. 3.2, Eqn. 3.4 and Eqn. 3.6, the final objective function can be written as:

$$\underset{\Theta}{\text{minimize}}\ \mathcal{L}_{cls} + \lambda \mathcal{L}_{utl} - \lambda J_{sel},$$

where λ controls the trade off between classification and temporal exploration and Θ denotes all trainable parameters. Note that the first two terms are differentiable, and we can directly use back propagation with stochastic gradient descent to learn the optimal weights. Thus, we only discuss how to maximize the expected reward J_{sel} in Eqn. 3.4. Following [20], we derive the expected gradient of J_{sel} as:

$$\nabla_{\Theta} J_{sel} = \mathbb{E}\left[\sum_{t=0}^{T_e} (R_t - \hat{V}_t) \nabla_{\Theta} \log \pi_{\theta}(\cdot \mid \boldsymbol{h}_t)\right], \tag{3.7}$$

where R_t denotes the expected future reward, and $\hat{V}_t$ serves as a baseline function to reduce variance during training [20]. Eqn. 3.7 can be approximated with Monte-Carlo sampling using samples in a mini-batch, and further back-propagated downstream for training.

3.3.2 Adaptive Lookahead Inference

While we optimize the memory-augmented LSTM for a fixed number of steps during training, we aim to achieve adaptive inference at testing time such that a small number of informative frames are selected conditioned on input videos without incurring any degradation in classification performance. Recall that the utility network is trained to predict expected future rewards, indicating the utility/advantage of seeing more frames in the future. Therefore, we explore the outputs of the utility network to determine whether to stop inference through looking ahead. A straightforward way is to calculate the utility $\hat{V}_t$ at each time step, and exit the model once it is less than a threshold. However, it is difficult to find an optimal value that works well for all samples. Instead, we maintain a running max of utility $\hat{V}^{max}$

over time for each sample, and at each time step, we compare the current utility $\hat{V}_t$ with the max value $\hat{V}_t^{max}$; if $\hat{V}_t^{max}$ is larger than $\hat{V}_t$ by a margin μ more than p times, predictions from the current time step will be used as the final score and inference will be stopped. Here, μ controls the trade-off between computational cost and accuracy; a small μ constrains the model to make early predictions once the predicted utility begins to decrease while a large μ tolerates a drop in utility, allowing more considerations before classification. Further, we also introduce p as a patience metric, which permits the current utility to deviate from the max value for a few iterations. This is similar in spirit to reducing learning rates on plateaus, which instead of intermediately decays learning rate waits for a few more epochs when the loss does not further decrease.

Note that although the same threshold μ is used for all samples, comparisons made to decide whether to stop or not is based on the utility distribution of each sample independently, which is softer than comparing $\hat{V}_t$ with μ directly. One can add another network to predict whether to stop inference using the hidden states as in [1, 2], however coupling the training of frame selection with learning a binary policy to stop makes optimization challenging, particularly with reinforcement learning, as will be shown in experiments. In contrast, we leverage the utility network to achieve adaptive lookahead inference.

3.4 Experiments

3.4.1 Experimental Setup

Datasets and evaluation metrics. We experiment with two challenging large-scale video datasets, Fudan-Columbia Video Datasets (FCVID) [69] and ActivityNet [70], to evaluate the proposed approach. FCVID consists of $91,223$ videos from YouTube with an average duration of 167 seconds, manually annotated into 239 classes. These categories cover a wide range of topics, including scenes (*e.g.*, "river"), objects (*e.g.*, "dog"), activities (*e.g.*, "fencing"), and complicated events (*e.g.*, "making pizza"). The dataset is split evenly for training ($45,611$ videos) and testing ($45,612$ videos). ActivityNet is an activity-focused large-scale video dataset, containing YouTube videos with an average duration of 117 seconds. Here we adopt the latest release (version 1.3), which consists of around $20K$ videos belonging to 200 classes. We use the official split with a training set of $10,024$ videos, a validation set of $4,926$ videos and a testing set of $5,044$ videos. Since the testing labels are not publicly available, we report performance on the validation set. We compute average precision (AP) for each class and use mean average precision (mAP) to measure the overall performance on both datasets. It is also worth noting that videos in both datasets are *untrimmed*, for which efficient recognition is extremely critical given the redundant nature of video frames.

Implementation details. We use a one-layer LSTM with $2,048$ and $1,024$ hidden units for FCVID and ActivityNet respectively. To extract inputs for the LSTM, we decode videos at 1fps and compute features from the penultimate layer of a ResNet-101 model [16]. The ResNet model is pretrained on ImageNet with a top-1 accuracy of 77.4% and further finetuned on target datasets. To generate the global memory that provides context information, we compute features using spatially and temporally downsampled video frames with a lightweight CNN to reduce overhead. In particular, we lower the resolution of video frames to 112×112, and sample 16 frames uniformly. We use a pretrained MobileNetv2 [87] as the lightweight CNN, which achieves a top-1 accuracy of 52.3% on ImageNet with downsampled inputs. We adopt PyTorch for implementation and leverage SGD for optimization with a momentum of 0.9, a weight decay of $1e-4$ and a λ of 1. We train the network for 100 epochs with a batch size of 128 and 64 for FCVID and ActivityNet, respectively. The initial learning rate is set to $1e-3$ and decayed by a factor of 10 every 40 epochs. For the patience p during inference, it is set to 2 when $\mu < 0.7$, and $K/2+1$ when $\mu = 0.7$, where K is number of time steps the model is trained for.

3.4.2 Main Results

Effectiveness of learned frame usage. We first optimize AdaFrame with K steps during training and then at testing time we perform adaptive lookahead inference with $\mu = 0.7$, allowing each video to see K' frames on average while maintaining the same accuracy as viewing all K frames. We compare AdaFrame with the following alternative methods to produce final predictions during testing: (1) AvgPooling,

	FCVID							ActivityNet						
Method	R8	U8	R10	U10	R25	U25	All	R8	U8	R10	U10	R25	U25	All
AvgPooling	78.3	78.4	79.0	78.9	79.7	80.0	80.2	67.5	67.8	68.9	68.6	69.8	70.0	70.2
LSTM	77.8	77.9	78.7	78.1	78.0	79.8	80.0	68.7	68.8	69.8	70.4	69.9	70.8	71.0
AdaFrame	78.6		79.2		**80.2**			69.5		70.4		**71.5**		
	$5 \rightarrow 4.92$		$8 \rightarrow 6.15$		$10 \rightarrow 8.21$			$5 \rightarrow 3.8$		$8 \rightarrow 5.82$		$10 \rightarrow 8.65$		

Table 3.1: **Performance of different frame selection strategies on FCVID and ActivityNet.** R and U denote random and uniform sampling, respectively. We use $K \rightarrow K'$ to denote the frame usage for AdaFrame, which uses K frames during training and K' frames on average when performing adaptive inference. See texts for more details.

which simply computes a prediction for each sampled frame and then performs a mean pooling over frames as the video-level classification score; (2) LSTM, which generates predictions using hidden states from the last time step of an LSTM. We also experiment with different number of frames ($K + \Delta$) used as inputs for AvgPooling and LSTM, which are sampled either uniformly (U) or randomly (R). Here, we use K for AdaFrame while $K + \Delta$ for other methods to offset the additional computation cost incurred, which will be discussed later. Table 3.1 presents the results. We observe AdaFrame achieves better results than AvgPooling and LSTM whiling using fewer frames under all settings on both datasets. In particular, AdaFrame achieves an mAP of 78.6%, and 69.5% using an average of 4.92 and 3.8 frames on FCVID and ActivityNet respectively. These results, requiring 3.08 and 4.2 fewer frames, are better than AvgPooling and LSTM with 8 frames and comparable with their results with 10 frames. It is also promising to see that AdaFrame can match the performance of using all frames with only 8.21 and 8.65 frames on FCVID and ActivityNet. This verifies that AdaFrame can indeed learn to derive frame selection policies while maintaining the same accuracies.

In addition, the performance of random sampling and uniform sampling for AvgPooling and LSTM are similar and LSTM is worse than AvgPooling on FCVID, possibly due to the diverse set of categories incur significant intra-class variations. Note that although AvgPooling is simple and straightforward, it is a very strong baseline and has been widely adopted during testing for almost all CNN-based approaches due to its strong performance.

Computational savings with adaptive inference. We now discuss computational savings of AdaFrame with adaptive inference and compare with state-of-the-art-methods. We use average GFLOPs, a hardware independent metric, to measure the computation needed to classify all the videos in the testing set. We train AdaFrame with fixed K time steps to obtain different models, denoted as AdaFrame-K to accommodate different computational requirements during testing; and for each model we vary μ such that adaptive inference can be achieved within the same model.

In addition to selecting frames based on heuristics, we also compare AdaFrame with FrameGlimpse [1] and FastForward [2]. FrameGlimpse is developed for action detection with a location network to select frames and a stop network to decide whether to stop; ground-truth boundaries of actions are used as feedback to estimate the quality of selected frames. For classification, there is no such ground-truth and thus we preserve the architecture of FrameGlimpse but use our reward function. FastForward [2] samples from a predefined action set, determining how many steps to go forward. It also consists of a stop branch to decide whether to stop. In

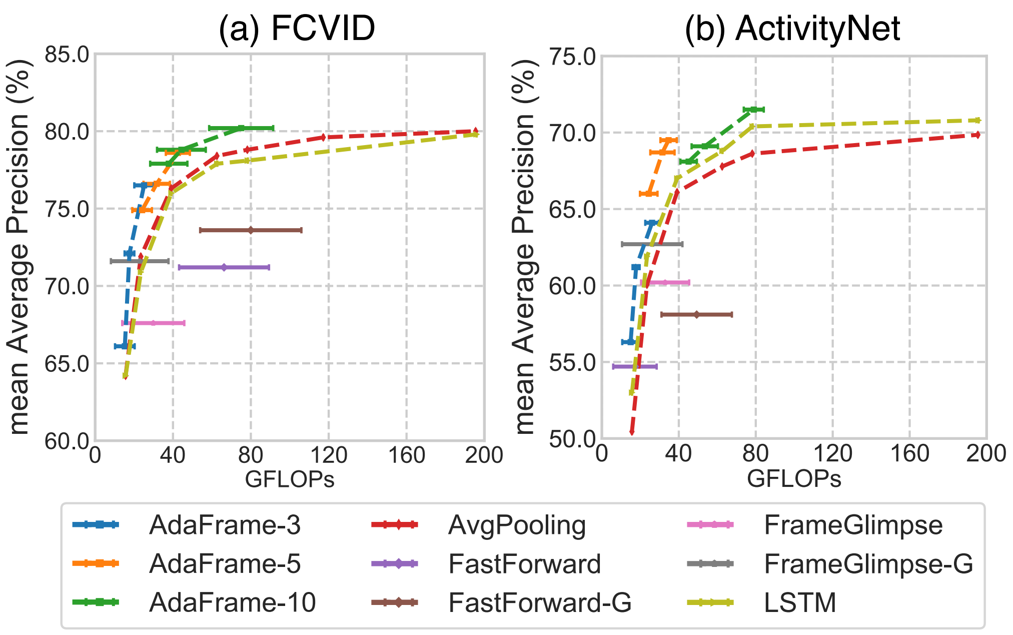


Figure 3.3: **Mean average precision *v.s.* computational cost**. Comparisons of AdaFrame with FrameGlimpse [1], FastForward [2], and alternative frame selection methods based on heuristics.

addition, we also attach the global memory to these frameworks for fair comparisons, denoted as FrameGlimpse-G and FastForward-G, respectively. Figure 3.3 presents the results. For AvgPooling and LSTM, accuracies gradually increase when more computation (frames) is used and then become saturated. Note that the computational cost for video classification grows linearly with the number of frames used, as the most expensive operation is extracting features with CNNs. For ResNet-101 it needs 7.82 GFLOPs to compute features and for AdaFrame, it takes an extra 1.32 GFLOPs due to the computation in global memory. Therefore, we expect more savings from AdaFrame when more frames are used.

Compared with AvgPooling and LSTM using 25 frames, AdaFrame-10 achieves better results while requiring 58.9% and 63.3% less computation on av-

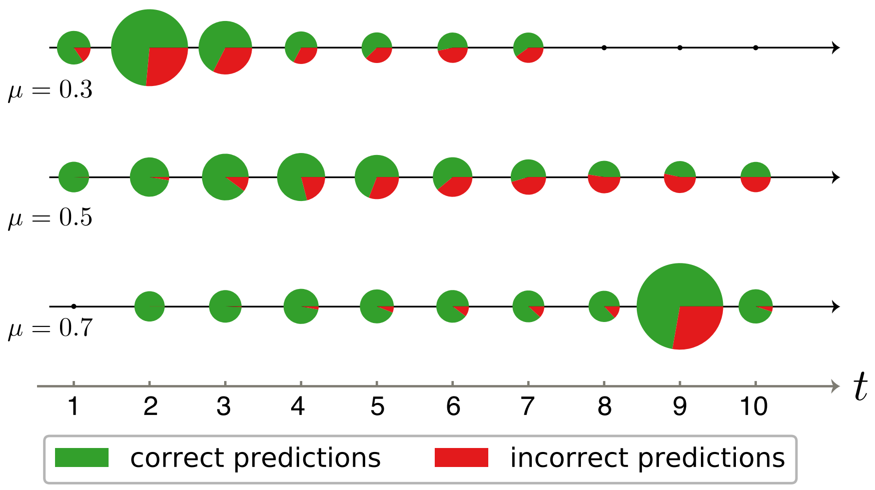


Figure 3.4: **Dataflow through AdaFrame over time**. Each circle represents, by size, the percentage of samples that are classified at the corresponding time step.

erage on FCVID (80.2 *v.s.* ∼195 GFLOPs [2]) and ActivityNet (71.5 *v.s.* ∼195 GFLOPs), respectively. Similar trends can also be found for AdaFrame-5 and AdaFrame-3 on both datasets. While the computational saving of AdaFrame over AvgPooling and LSTM reduces when fewer frames are used, accuracies of AdaFrame are still clearly better, *i.e.*, 66.1% *v.s.* 64.2% on FCVID, and 56.3% *v.s.* 53.0% on ActivityNet. Further, AdaFrame also outperforms FrameGlimpse [1] and Fast-Forward [2] that aim to learn frame usage by clear margins, demonstrating that coupling the training of frame selection and learning to stop with reinforcement learning on large-scale datasets without sufficient background videos is difficult. In addition, the use of a global memory, providing context information improves accuracies of the original model in both frameworks.

We can also see that changing the threshold μ within the same model can also adjust computation needed; the performance and average frame usage declines

[2]195.5 GFLOPS for AvgPooling and 195.8 GFLOPs for LSTM.

simultaneously as the threshold becomes smaller, forcing the model to make predictions as early as possible. But the resulting policies with different thresholds still outperform alternative counterparts in both accuracy and computation required.

Comparing across different models of AdaFrame, we observe that the best model of AdaFrame trained with a smaller K achieves better or comparable results over AdaFrame optimized with a large K using a smaller threshold. For example, AdaFrame-3 with $\mu = 0.7$ achieves an mAP of 76.5% using 25.1 GFLOPs on FCVID, which is better than AdaFrame-5 with $\mu = 0.5$ that produces an mAP of 76.6% with 31.6 GFLOPs on average. This possibly results from the discrepancies between training and testing—during training a large K allows the model to "ponder" before emitting predictions. While computation can be adjusted with varying thresholds at test time, AdaFrame-10 is not fully optimized for classification with extremely limited information as is AdaFrame-3. This highlights the need to use different models based on computational requirements.

Analyses of learned policies. To gain a better understanding of what is learned in AdaFrame, we take the trained AdaFrame-10 model and vary the threshold to accommodate different computational needs. And we visualize in Figure 3.4, at each time step, how many samples are classified, and the prediction accuracies of these samples. We can see high prediction accuracies tend to appear in early time steps, pushing difficult decisions that require more scrutiny downstream. And more samples emit predictions at later time steps when computational budget increases (larger μ).

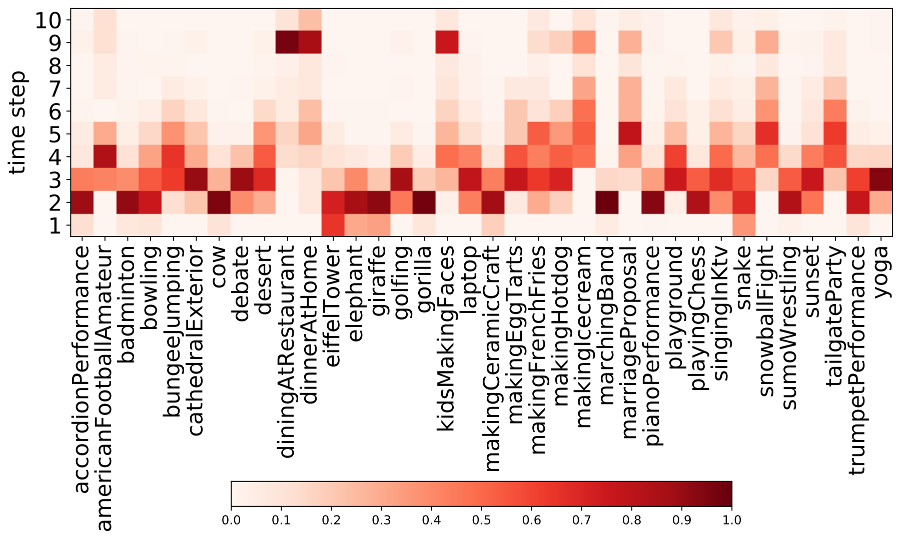


Figure 3.5: **Learned inference policies for different classes over time.** Each square, by density, indicates the fraction of samples that are classified at the corresponding time step from a certain class in FCVID.

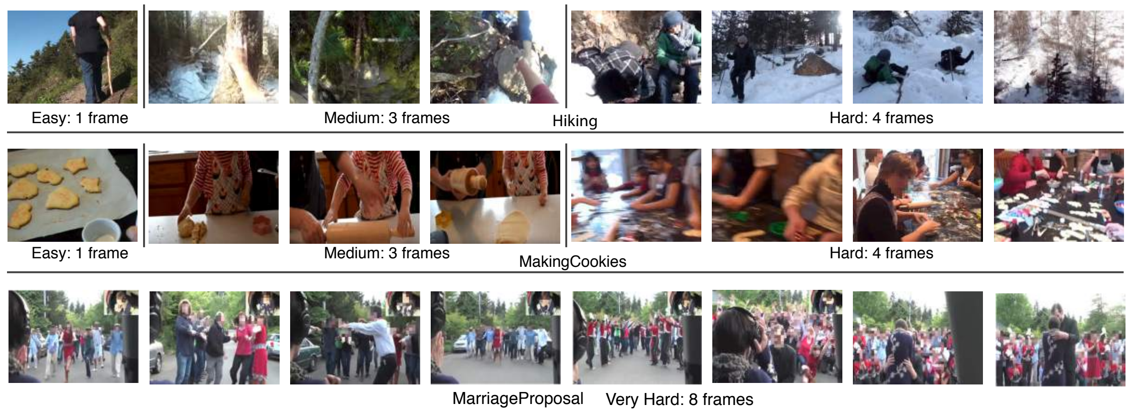


Figure 3.6: **Validation videos from FCVID using different number of frames for inference.** Frame usage differs not only among different categories but also within the same class (*e.g.*, "making cookies" and "hiking").

We further investigate whether computations vary for different categories. To this end, we show the fraction of samples from a subset of classes in FCVID that are classified at each time step in Figure 3.5. We observe that, for simple classes like objects (*e.g.*, "gorilla" and "elephants") and scenes ("Eiffel tower" and "cathedral exterior"), AdaFrame makes predictions for most of the samples in the first three steps; while for some complicated DIY categories (*e.g.*, "making ice cream" and "making egg tarts"), it tends to classify in the middle of the entire time horizon. In addition, AdaFrame takes additional time steps to differentiate very confusing classes like "dining at restaurant" and "dining at home". Figure 4.4 further illustrates samples using different numbers of frames for inference. We can see that frame usage varies not only across different classes but also within the same category (see the top two rows of Figure 4.4) due to large intra-class variations. For example, for the "making cookies" category, it takes AdaFrame four steps to make correct predictions when the video contains severe camera motions and cluttered backgrounds.

In addition, we also examine where the model jumps at each step; for AdaFrame-10 with $\mu = 0.7$, we found that it goes backward at least once for 42.8% of videos on FCVID to re-examine past information instead of always going forward, confirming the flexibility AdaFrame enjoys when searching over time.

3.4.3 Discussions

In this section, we conduct a set of experiments to justify our design choices of AdaFrame.

Global memory. We perform an ablation study to see how many frames are needed in the global memory. Table 3.2 presents the results. The use of a global memory module improves the non-memory model with clear margins. In addition,

Global Memory		Inference	
# Frames	Overhead	mAP	# Frames
0	0	77.9	8.40
12	0.98	79.2	8.53
32	2.61	80.2	8.24
16	1.32	**80.2**	**8.21**

Table 3.2: **Results of using different global memories on FCVID**. Different number of frames are used to generate different global memories. The overhead is measured for each frame compared to a standard ResNet-101.

we observe using 16 frames offers the best trade-off between computational overheads and accuracies.

Reward function. Our reward function forces the model to increase its confidence when seeing more frames, to measure the transition from the last time step. We further compare with two reward functions: (1) PREDICTION REWARD, that uses the prediction confidence of the ground-truth class p_t^{gt} as reward; (2) PREDICTION TRANSITION REWARD, that uses $p_t^{gt} - p_{t-1}^{gt}$ as reward. The results are summarized in Table 3.3. We can see that our reward function and PREDICTION TRANSITION REWARD, both modeling prediction differences over time, outperform PREDICTION REWARD that is simply based on predictions from the current step. This verifies that forcing the model to increase its confidence when viewing more frames can provide feedback about the quality of selected frames. Our result is also better than PREDICTION TRANSITION REWARD by further introducing a margin between predictions from the ground-truth class and other classes.

Stop criterion. In our framework, we use the predicted utility, measuring future rewards of seeing more frames, to decide whether to continue inference or not. An

Reward function	mAP	# Frames
Prediction Reward	78.7	8.34
Prediction Transition Reward	78.9	8.31
Ours	**80.2**	**8.21**

Table 3.3: **Comparisons of different reward functions on FCVID**. Frames used on average and the resulting mAP.

alternative is to simply rely on the entropy of predictions, as a proxy to measure the confidence of classifiers. We also experimented with entropy to stop inference, however we found that it cannot enable adaptive inference based on different thresholds. We observed that predictions over time are not as smooth as predicted utilities, *i.e.*, high entropies in early steps and extremely low entropies in the last few steps. In contrast, utilities are computed to measure future rewards, explicitly considering future information from the very first step, which leads to smooth transitions over time.

3.5 Conclusion

In this chapter, we presented AdaFrame, an approach that derives an effective frame usage policy so as to use a small number of frames on a per-video basis with an aim to reduce the overall computational cost. It contains an LSTM network augmented with a global memory to inject global context information. AdaFrame is trained with policy gradient methods to predict which frame to use and calculate future utilities. During testing, we leverage the predicted utility for adaptive inference. Extensive results provide strong qualitative and quantitative evidence that AdaFrame can derive strong frame usage policies based on inputs.

Chapter 4: LiteEval: A Coarse-to-Fine Framework for Resource Efficient Video Recognition

4.1 Introduction

Convolutional neural networks (CNNs) have demonstrated stunning progress in several computer vision tasks like image classification [27, 88, 89], object detection [18, 90], video classification [62, 64], *etc.*, sometimes even surpassing human-level performance [88] when recognizing fine-grained categories. The astounding performance of CNN models, while making them appealing for deployment in many practical applications such as autonomous vehicles, navigation robots and image recognition services, results from complicated model design, which in turn limits their use in real-world scenarios that are often resource-constrained. To remedy this, extensive studies have been conducted to compress neural networks [10, 39, 42] and design compact architectures suitable for mobile devices [43, 44]. However, they produce one-size-fits-all models that require the same amount of computation for all samples.

Although computationally efficient models usually exhibit good accuracy when recognizing the majority of samples, computationally expensive models, if not ensembles of models, are needed to additionally recognize corner cases that lie in the tail of the data distribution, offering top-notch performance on standard benchmarks like ImageNet [23] and COCO [91]. In addition to network design, the computational cost of CNNs is directly affected by input resolution—74% of computation can be saved (measured by floating point operations) when evaluating a ResNet-101 model on images with half of the original resolution, while still offering reasonable accuracy. Motivated by these observations, a natural question arises: can we have a network with components of different complexity operating on different scales and derive policies conditioned on inputs to switch among these components to save computation? Intuitively, during inference, lightweight modules are run by default to recognize easy samples (*e.g.*, images with canonical views) with coarse scale inputs and high-precision components will be activated to further obtain finer details to recognize hard samples (*e.g.*, images with occlusion). This is conceptually similar to human perception systems where we pay more attention to complicated scenes while a glance would suffice for most objects.

In this spirit, we explore the problem of dynamically allocating computational resources for video recognition. We consider resource-constrained video recognition for two reasons: (1) Videos are more computationally demanding compared to images. Thus, video recognition systems should be resource efficient, since computation is a direct indicator of energy consumption, which should be minimized to be cost-effective and eco-friendly; additionally, power assumption directly affects battery life of embedded systems. (2) Videos exhibit large variations in computation required to be correctly labeled. For instance, for videos that depict static scenes (*e.g.*, "river" or "desert") or centered objects (*e.g.*, "gorilla" or "panda"), viewing a single frame already gives high confidence, while one needs to see more frames in order to distinguish "making latte" from "making cappuccino". Further, frames needed to predict the label of a video clip not only differ among different classes but also within the same category. For example, for many sports actions like "running" and "playing football", professionally recorded videos with less camera motion are more easily recognized compared to user-generated videos using hand-held devices or wearable cameras.

We introduce LiteEval, a resource-efficient framework suitable for both online and offline video classification, which adaptively assigns computational resources to incoming video frames. In particular, LiteEval is a coarse-to-fine framework that uses coarse information for economical evaluation while only requiring fine clues when necessary. It consists of a coarse LSTM operating on features extracted from downsampled video frames using a lightweight CNN, a fine LSTM whose inputs are features from images of a finer scale using a more powerful CNN, as well as a gating module to dynamically decide the granularity of features to use. Given a stream of video frames, at each time step, LiteEval computes coarse features from the current frame and updates the coarse LSTM to accumulate information over time. Then, conditioned on the coarse features and historical information, the gating module determines whether to further compute fine features to obtain more details from the current frame. If further analysis is needed, fine features are computed and input into the fine LSTM for temporal modeling; otherwise hidden states from the coarse LSTM are synchronized with those of the fine LSTM such that the fine

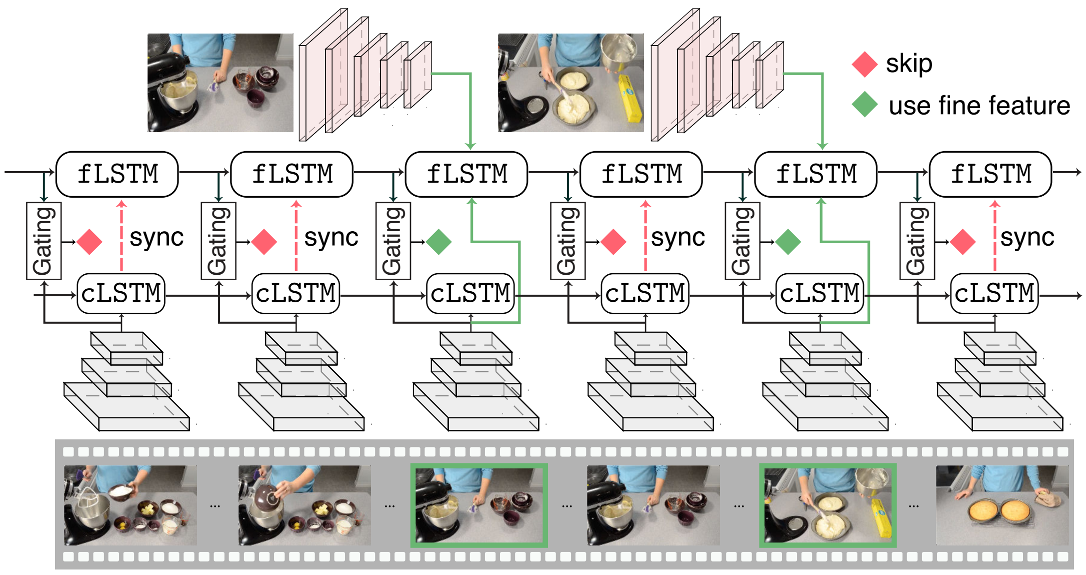


Figure 4.1: **An overview of LiteEval.** At each time step, coarse features, computed with a lightweight CNN, together with historical information are used to determine whether to examine the current frame more carefully. If further inspection is needed, fine features are derived to update the fine LSTM; otherwise the two LSTMs are synchronized. See texts for more details.

LSTM contains all information seen so far to be readily used for prediction. Finally, LiteEval proceeds to the next frame. Such a recurrent and efficient way of processing video frames allows LiteEval to be used in both online and offline scenarios. See Figure 6.2 for an overview of the framework.

We conduct extensive experiments on two large-scale video datasets for generic video classification (FCVID [69]) and activity recognition (ActivityNet [70]) under both online and offline settings. For offline predictions, we demonstrate that LiteEval achieves accuracies that are on par with the strong and popular uniform sampling strategy while requiring 51.8% and 51.3% less computation, and it also outperforms efficient video recognition approaches in recent literature [1, 2]. We also show that LiteEval can be effectively used for online video predictions to accommodate different computational budgets. Furthermore, qualitative results suggest the learned fine feature usage policies not only correspond to the difficulty to make predictions (*i.e.*, easier samples require fewer fine features) but also can reflect salient parts in videos when recognizing a class of interest.

4.2 Approach

LiteEval consists of a coarse LSTM and a fine LSTM that are organized hierarchically taking in visual information at different granularities, as well as a conditional gating module governing the switching between different feature scales. In particular, given a stream of video frames, the goal of LiteEval is to learn a policy that determines at each time step whether to examine the incoming video frame carefully with discriminative yet computationally expensive features, conditioned on a quick glance of the frame with economical features computed at a coarse scale and historical information. LiteEval operates on coarse information by default and is expected to take in fine details infrequently, reducing overall computational cost while maintaining recognition accuracy. In the following, we introduce each component in our framework in detail, and present the optimization of the model.

4.2.1 A Coarse-to-Fine Framework

Coarse LSTM. Operating on features computed at a coarse image scale using a lightweight CNN model (see Sec. 4.3.1 for details), the coarse LSTM quickly glimpses over video frames to get an overview of the current inputs in a computationally efficient manner. More formally, at the t-th time step, the coarse LSTM takes in the coarse features $\boldsymbol{v}_t^c$ of the current frame, previous hidden states $\boldsymbol{h}_{t-1}^c$ and cell outputs $\boldsymbol{c}_{t-1}^c$ to compute the current hidden states $\boldsymbol{h}_t^c$ and cell states $\boldsymbol{c}_t^c$:

$$\boldsymbol{h}_t^c,\, \boldsymbol{c}_t^c = \texttt{cLSTM}(\boldsymbol{v}_t^c,\, \boldsymbol{h}_{t-1}^c,\, \boldsymbol{c}_{t-1}^c). \tag{4.1}$$

Conditional gating module. The coarse LSTM skims video frames efficiently without allocating too much computation; however, fast processing with coarse features will inevitably overlook important details needed to differentiate subtle actions/events (*e.g.*, it is much easier to separate "drinking coffee" from "drinking beer" with larger video frames). Therefore, LiteEval incorporates a conditional gating module to decide whether to examine the incoming video frame more carefully to obtain finer details. The gating module is a one-layer MLP that outputs the probability (unnormalized) to compute fine features with a more powerful CNN:

$$\boldsymbol{b}_t \in \mathbb{R}^2 = \boldsymbol{W}_g^\top [\boldsymbol{v}_t^c, \boldsymbol{h}_{t-1}^f, \boldsymbol{c}_{t-1}^f], \tag{4.2}$$

where $\boldsymbol{W}_g$ are the weights for the conditional gate, $\boldsymbol{h}_{t-1}^f$ and $\boldsymbol{c}_{t-1}^f$ are the hidden and cell states of the fine LSTM (discussed below) from the previous time step, and $[\,,\,]$ denotes the concatenation of features. Since the gating module aims to make a discrete decision whether to compute features at a finer scale based on $\boldsymbol{b}_t$, a straightforward way is choose a higher value in $\boldsymbol{b}_t$, which, however, is not differentiable. Instead, we define a random variable B_t to make the decision through sampling from $\boldsymbol{b}_t$. Learning such a parameterized gating function by sampling can be achieved in different ways, as will be discussed below in Section 4.2.2.

Fine LSTM. If the gating module selects to pay more attention to the current frame (*i.e.*, $B_t = 1$), features at a finer scale will be computed with a computationally intensive CNN, and will be sent to the fine LSTM for temporal modeling. In particular, the fine LSTM takes as inputs—fine features $\boldsymbol{v}_t^f$ concatenated with coarse features $\boldsymbol{v}_t^c$, previous hidden states $\boldsymbol{h}_{t-1}^f$ and cell states $\boldsymbol{c}_{t-1}^f$—to produce hidden states $\boldsymbol{h}_t^f$ and cells outputs $\boldsymbol{c}_t^f$ of the current time step:

$$\widetilde{\boldsymbol{h}_t^f},\, \widetilde{\boldsymbol{c}_t^f} = \texttt{fLSTM}([\boldsymbol{v}_t^c, \boldsymbol{v}_t^f],\, \boldsymbol{h}_{t-1}^f,\, \boldsymbol{c}_{t-1}^f) \tag{4.3}$$

$$\boldsymbol{h}_t^f = (1 - B_t)\boldsymbol{h}_{t-1}^f + B_t\widetilde{\boldsymbol{h}_t^f}, \quad \boldsymbol{c}_t^f = (1 - B_t)\boldsymbol{c}_{t-1}^f + B_t\widetilde{\boldsymbol{h}_t^f}. \tag{4.4}$$

When the gating module opts out of the computation of fine features (*i.e.*, $B_t = 0$), hidden states from the previous time step are reused.

Synchronizing the `cLSTM` with the `fLSTM`. It worth noting that the coarse LSTM contains information from all frames seen so far, while hidden states in the fine LSTM only consist of accumulated knowledge from frames selected by the gating module. While fine-grained details are stored in `fLSTM`, `cLSTM` provides context information from the remaining frames that might be beneficial for recognition. To obtain improved performance, a straightforward way is to concatenate their hidden states before classification, yet they are asynchronous (the coarse LSTM is always ahead of the fine LSTM, seeing more frames), making it difficult to know when to perform fusion. Therefore, we synchronize these two LSTMs by simply copying.

In particular, at the t-th step, if the gating module decides not to compute fine features (*i.e.*, $B_t = 0$ in Equation 4.4), instead of using $\boldsymbol{h}_{t-1}^f$ directly, we update $\boldsymbol{h}_t^f = [\boldsymbol{h}_t^c, \boldsymbol{h}_{t-1}(D^c + 1 : D^f)]$, where D^c and D^fdenote the dimension of $\boldsymbol{h}^c$ and $\boldsymbol{h}^f$, respectively. Similar modifications are performed to $\boldsymbol{c}_t^f$. Now the hidden states in the fine LSTM contains all information seen so far and can be readily used to derive predictions at any time: $\boldsymbol{p}_t = \texttt{softmax}(\boldsymbol{W}_p^\top \boldsymbol{h}_t^f)$, where $\boldsymbol{W}_p$ denotes the weights for the classifier.

4.2.2 Optimization

Let $\Theta = \{\Theta_{\texttt{cLSTM}}, \Theta_{\texttt{fLSTM}}, \Theta_g\}$ denote the trainable parameters in the framework, where $\Theta_{\texttt{cLSTM}}$ and $\Theta_{\texttt{fLSTM}}$ represent the parameters in the coarse and fine LSTMs, respectively and Θ_g are weights for the gating module [1]. During training, we use predictions from the last time step T as the video-level predictions, and optimize the following loss function:

$$\underset{\Theta}{\text{minimize}}\ \mathbb{E}^{B_t \sim \texttt{Bernoulli}(\boldsymbol{b}_t;\Theta_g)}_{(\boldsymbol{x},\boldsymbol{y})\sim D_{train}} -\boldsymbol{y}\,\texttt{log}(\boldsymbol{p}_T(\boldsymbol{x};\Theta)) \ +\ \lambda(\frac{1}{T}\sum_{t=1}^{T} B_t - \gamma)^2. \tag{4.5}$$

Here $\boldsymbol{x}$ and $\boldsymbol{y}$ denote a sampled video and its corresponding one-hot label vector from the training set D_{train} and the first term is a standard cross-entropy loss. The second term limits the usage of fine features to a predefined target γ with $\frac{1}{T}\sum_{t=1}^{T} B_t$ being the fraction of the number of times fine features are used over the entire time

[1]We absorb the weights of the classifier $\boldsymbol{W}_p$ into $\Theta_{\texttt{fLSTM}}$.

horizon. In addition, λ balances the trade-off between recognition accuracy and computational cost.

However, optimizing Equation 4.5 is not trivial as the decision whether to compute fine features is binary and requires sampling from a Bernoulli distribution parameterized by Θ_g. One way to solve this is to convert the optimization in Equation 4.5 to a reinforcement learning problem and then derive the optimal parameters of the gating module with policy gradient methods [20] by associating each action taken with a reward. However, training with policy gradient requires techniques to reduce variance during training as well as carefully selected reward functions. Instead, we use a Gumbel-Max trick to make the framework fully differentiable. More specifically, given a discrete categorical variable $\hat{B}$ with class probabilities $P(\hat{B} = k) \propto b_k$, where $b_k \in (0, \infty)$ and $k \leq K$ (K denotes the total number of classes; in our framework $K = 2$), the Gumbel-Max [92, 93] trick indicates the sampling from a categorical distribution can be performed in the following way:

$$\hat{B} = \arg\max_k (\texttt{log}\, b_k + G_k), \tag{4.6}$$

where $G_k = -\texttt{log}\,(-\texttt{log}\,(U_k))$ denotes the Gumbel noise and U_k are i.i.d samples drawn from $\texttt{Uniform}\,(0, 1)$. Although the arg max operation in Equation 4.6 is not differentiable, we can use `softmax` as as a continuous relaxation of arg max [93, 94]:

$$B_i = \frac{\texttt{exp}((\texttt{log}\, b_i + G_i)/\tau)}{\sum_{j=1}^{K} \texttt{exp}((\texttt{log}\, b_j + G_j)/\tau)} \quad \text{for } i = 1, .., K \tag{4.7}$$

where τ is a temperature parameter controlling discreteness in the output vector B. Consider the extreme case when $\tau \to 0$, Equation 4.7 produces the same samples as Equation 4.6.

In our framework, at each time step, we are sampling from a Gumbel-Softmax distribution parameterized by the weights of of the gating module Θ_g. This facilitates the learning of binary decisions in a fully differentiable framework. Following [94], we anneal the temperature from a high value to encourage exploration to a smaller positive value.

4.3 Experiments

4.3.1 Experimental Setup

Datasets and evaluation metrics. We adopt two large-scale video classification benchmarks to evaluate the performance of LiteEval, *i.e.*, FCVID and ActivityNet. FCVID (Fudan-Columbia Video Dataset) [69] contains $91,223$ videos collected from YouTube belonging to 239 classes that are selected to cover popular topics in our daily lives like "graduation", "baby shower", "making cookies", *etc.* The average duration of videos in FCVID is 167 seconds and the dataset is split into a training set with $45,611$ videos and a testing set with $45,612$ videos. While FCVID contains generic video classes, ActivityNet [70] consists of videos that are action/activity-oriented like "drinking beer", "drinking coffee", "fencing", *etc.* There are around $20K$ videos in ActivityNet with an average duration of 117 seconds, manually annotated into 200 categories. Here, we use the $v1.3$ split with a training set of $10,024$ videos, a validation set of $4,926$ videos and a testing set of $5,044$ videos. We report performance on the validation set since labels in the testing set are withheld by the authors. For offline prediction, we compute average precision (AP) for each video category and use mean AP across all classes to measure the overall performance following [69, 70]. For online recognition, we compute top-1 accuracy when evaluating the performance of LiteEval since average precision is a ranking-based metric based on all testing videos, which is not suitable for online prediction (we do observe similar trends with both metrics). We measure computational cost with giga floating point operations (GFLOPs), which is a hardware independent metric.

Implementation details. We extract coarse features with a MobileNetv2 [87] model using spatially downsampled video frames (*i.e.*, 112×112). The MobileNetv2 is lightweight model and achieves a top-1 accuracy of 52.3% on ImageNet operating on images with a resolution of 112×112. To extract features from high-resolution images (*i.e.*, 224×224) as inputs to the fine LSTM, we use a ResNet-101 model and obtain features from its penultimate layer. The ResNet-101 model offers a top-1 accuracy of 77.4% on ImageNet and it is further finetuned on target datasets to give better performance. We implement the framework using PyTorch on one NVIDIA P6000 GPU and adopts Adam [95] as the optimizer with a fixed learning rate of $1e-4$ and set λ to 2. For ActivityNet, we train with a batch size of 128 and the coarse LSTM and the fine LSTM respectively contain 64 and 512 hidden units, while for FCVID, there are 512 and $2,048$ hidden units in the coarse and fine LSTM respectively and the batch size is 256. The computational cost for MobileNetv2 (112×112) ResNet-101 (224×224) is 0.08 and 7.82 GFLOPs, respectively.

4.3.2 Main Results

Offline recognition. We first report the results of LiteEval for offline prediction and compare with the following alternatives: (1) Uniform, which computes predictions from 25 uniformly sampled frames and then averages these frame-level results as video-level classification scores; (2) LSTM, which produces predictions with hidden states from the last time step of an LSTM; (3) FrameGlimpse [1], which employs an agent trained with REINFORCE [20] to select a small number of frames for efficient recognition; (4) FastForward [2], which at each time step learns how many steps to jump forward by training an agent to select from a predefined action set; (5) LiteEval-RL, which is a variant of LiteEval using REINFORCE for learning binary decisions. The first two methods are widely used baselines for video recognition, particularly the strong uniform testing strategy which is adopted by almost all CNN-based approaches, while the remaining approaches focus on efficient video understanding.

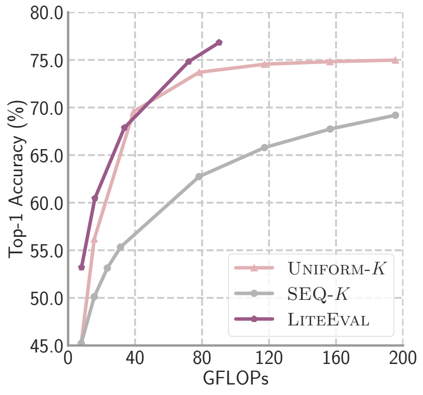
Table 4.1 summarizes the results and comparisons. LiteEval offers 51.8% (94.3 *v.s.* 195.5) and 51.3% (95.1 *v.s.* 195.5) computational savings measured by GFLOPs compared to the uniform baseline while achieving similar or better accuracies on FCVID and ActivityNet, respectively. The confirms that LiteEval can save computation by computing expensive features as infrequently as possible while operating

Table 4.1: **Results of different methods for offline video recognition.** We compare LiteEval with alternative methods on FCVID and ActivityNet.

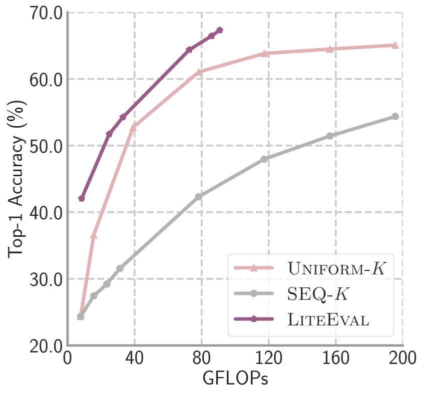
	FCVID		ActivityNet	
Method	mAP	GFLOPs	mAP	GFLOPs
Uniform	80.0%	195.5	70.0%	195.5
LSTM	79.8%	196.0	70.8%	195.8
FrameGlimpse [1]	71.2%	29.9	60.2%	32.9
FastForward [2]	67.6%	66.2	54.7%	17.2
LiteEval-RL	74.2%	245.9	65.2%	269.3
LiteEval	**80.0%**	94.3	**72.7%**	95.1

on economical features by default. The reason that LiteEval requires more computation on average on ActivityNet than FCVID is that categories in ActivityNet are action-focused whereas FCVID also contains classes that are relatively static with fewer motion like scenes and objects. Further, compared to FrameGlimpse and FastForward that also learn frame usage policies, LiteEval achieves significantly better accuracy although it requires more computation. Note that the low computation of FrameGlimpse and FastForward results from their access to future frames (*i.e.*, jumping to a future time step), while we simply make decisions whether to compute fine features for the current frame, making the framework suitable not only for offline prediction but also in online settings, as will be discussed below. In addition, we also compare with LiteEval-RL, which instead of using Gumbel-Softmax leverages policy search methods, to learn binary decisions. LiteEval is clearly better than LiteEval-RL in terms of both accuracy and computational cost, and it is also easier to optimize.

Online recognition with varying computational budgets. Once trained, LiteEval can be readily deployed in an online setting where frames arrive sequen-



(a) FCVID



(b) ACTIVITYNET

Figure 4.2: **Computational cost *v.s.* recognition accuracy on FCVID and ActivityNet.** Results of LiteEval and comparisons with alternative methods for online video prediction.

tially. Since computing fine features is the most expensive operation in the framework, given a video clip (7.82 GFLOPs per frame), we vary the number of times fine features are read in (denoted by K) such that different computational budgets can be accommodated, *i.e.* forcing early predictions after the model has computed fine features for the K-th time. This is similar in spirit to any time prediction [55] where there is a budget for each testing sample. We then report the average computational cost with respect to the achieved top-1 recognition accuracy on the testing set. We compare with (1) UNIFORM-K, which, for a testing video, averages predictions from K frames sampled uniformly from a total of K' frames as its final prediction scores (K' is the location where LiteEval produces predictions after having seen the fine features for the K-th time); (2) SEQ-K, which performs a mean-pooling of K consecutive frames.

The results are summarized in Figure 4.2. We observe the LiteEval offers the best trade-off between computational cost and recognition accuracy in the online setting on both FCVID and ACTIVITYNET. It is worth noting while UNIFORM-K is a powerful baseline, it is not practical in the online setting as there is no prior about how many frames are seen so far and yet to arrive. Further, LiteEval outperforms the straightforward frame-by-frame computation strategy SEQ-K by clear margins. This confirms the effectiveness when LiteEval is deployed online.

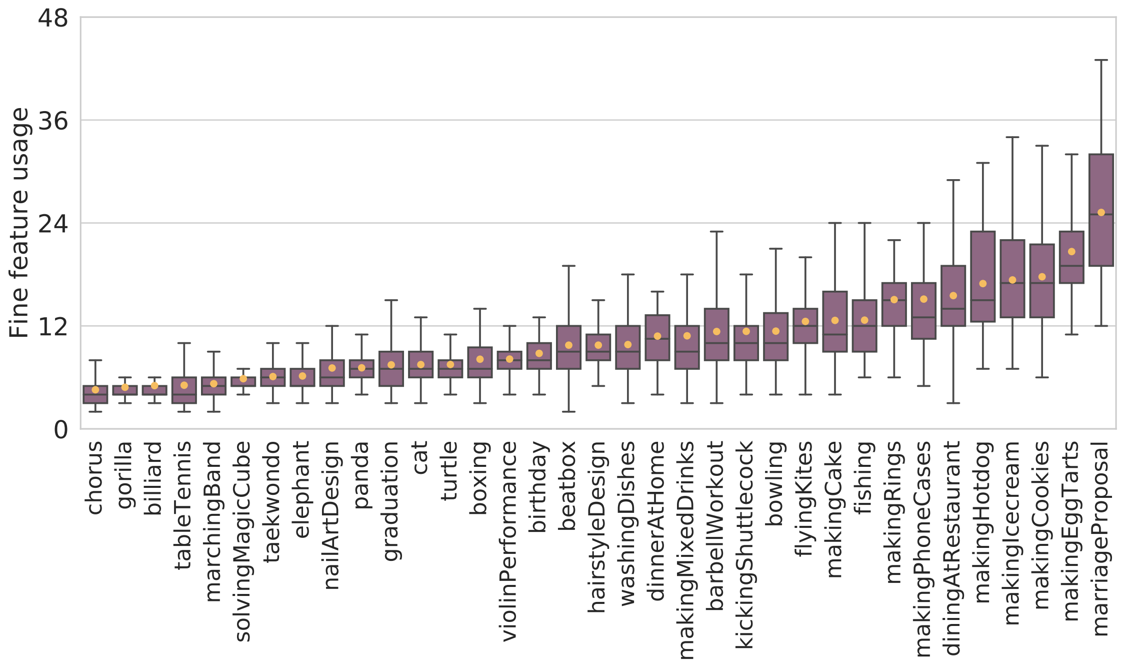


Figure 4.3: **The distribution of fine feature usage for sampled classes on FCVID.** In addition to quartiles and medians, mean usage, denoted as yellow dots, is also presented.

Learned policies for fine feature usage. We now analyze the policies learned by the gating module whether to compute fine features or not. Figure 4.3 visualizes the distribution of fine feature usage for sampled video categories in FCVID. We can see that the number of times fine features are computed not only varies across different categories but also within the same class. Since fine feature usage is proportional to the overall computation required, this verifies our hypothesis that computation

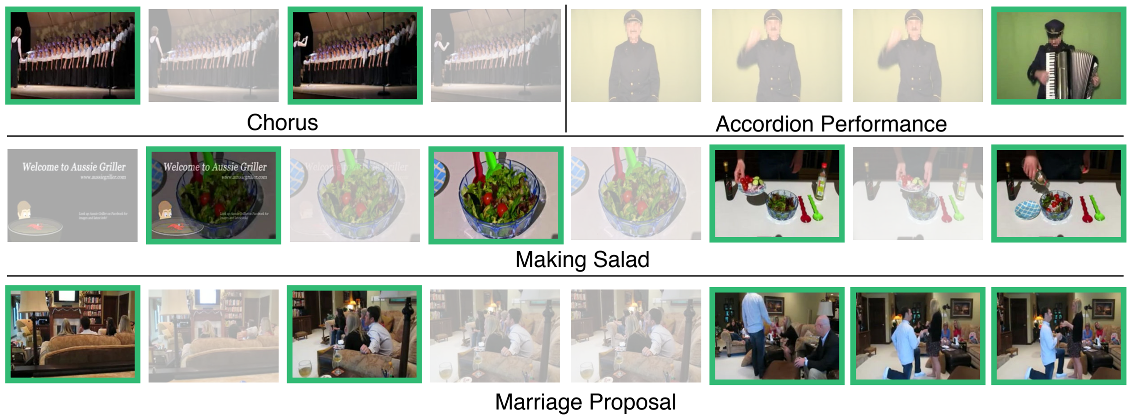


Figure 4.4: **Frame selected (indicated by green borders) by LiteEval of sampled videos to compute fine features in FCVID.**

Table 4.2: **The effectiveness of syncing LSTMs on FCVID.**

Method	mAP
w/o. sync	65.7%
LiteEval	80.0%

Table 4.3: **Results of different γ in LiteEval on FCVID.**

γ	mAP	GFLOPs
0.01	78.8%	75.4
0.03	79.7%	82.1
0.10	80.1%	139.0
0.05	80.0%	94.3

Table 4.4: **Results of different sizes of LSTMs on FCVID.**

# units in `cLSTM`	mAP
64	76.9%
128	77.3%
256	78.3%
512	80.0%

required to make correct predictions is different conditioned on input samples. We further visualize, in Figure 4.4, selected frames by LiteEval to compute fine features of certain videos. We observe that redundant frames without additional information are ignored and those selected frames provide salient information for recognizing the class of interest.

4.3.3 Ablation Studies

Fine feature usage. Table 4.3 presents the results of using γ to control fine feature usage in LiteEval. We observe that setting γ to 0.05 offers the best trade-off between computational cost and accuracies while using a extremely small γ

(*e.g.*, 0.01) achieves worse results, since it forces the model to compute fine features as seldom as possible to save computation and could possibly overlook important information. It is also worth mentioning that using relatively small values (*i.e.*, less or equal than 0.1) produces decent results, demonstrating there exists a high level of redundancy in video frames.

The synchronization of the fine LSTM with the coarse LSTM. We also investigate the effectiveness of synchronization of the two LSTMs. We can see in Table 4.2 that, without updating the hidden states of the `fLSTM` with those of the `cLSTM`, the performance degrades to 65.7%. This confirms that synchronization by transferring information from the `cLSTM` to `fLSTM` is critical for good performance as it makes the fine LSTM aware of all useful information seen so far.

Number of hidden units in the LSTMs. We experiment with different number of hidden units in the coarse LSTM and present the results in Table 4.4. We can see that using a small LSTM with fewer hidden units degrades performance due to limited capacity. As mentioned earlier, the most expensive operation in the framework is to compute CNN features from video frames, while LSTMs are much more computationally efficient—only 0.06% of GFLOPs needed to extract features with a ResNet-101 model. For the fine LSTM, we found that a size of $2,048$ offers the best results.

4.4 Related Work

Conditional Computation. Our work relates to conditional computation that aims to achieve decent recognition accuracy while accommodating varying computational budgets. Cascaded classifiers [52] are among the earliest work to save computation by quickly rejecting easy negative windows for fast face detection. Recently, the idea of conditional computation has also been investigated in deep neural networks [24, 46, 47, 53, 54, 55] through learning when to exit CNNs with attached decision branches. Graves [25] add a halting unit to RNNs to associate a ponder cost for computation. Several recent approaches learn to choose which layers in a large network to use [77, 78, 79] or select regions to attend to in images [80, 81], conditioned on inputs, to achieve fast inference. In contrast, we focus on conditional computation in videos, where we learn a fine feature usage strategy to determine whether to use computationally expensive components in a network.

Efficient Video Analysis. While there is plethora of work focusing on designing robust models for video classification, limited efforts have been made on efficient video recognition [1, 2, 65, 66, 73, 96, 97, 98]. Yeung *et al.* use an agent trained with policy gradient methods to select informative frames and predict when to stop inference for action detection [1]. Fan *et al.* further introduce a fast forward agent that decides how many frames to jump forward at a certain time step [2]. While they are conceptually similar to our approach, which also aims to skip redundant frames, our framework is fully differentiable, and thus is easier to train than policy search methods [1, 2]. More importantly, without assuming access to future frames, our framework is not only suitable for offline predictions but also can be deployed in an online setting where a stream of video frames arrive sequentially. A few recent approaches explore lightweight 3D CNNs to save computation [73, 97], but they use the same set of parameters for all videos regardless of their complexity. In contrast, LiteEval is a general dynamic inference framework for resource-efficient recognition, leveraging LSTMs to aggregate temporal information and making feature usage decisions over time; it is complementary to 3D CNNs, as we can replace the inputs to the fine LSTM with features from 3D CNNs, dynamically determining whether to compute powerful features from incoming video snippets.

4.5 Conclusion

We presented LiteEval, a simple yet effective framework for resource-efficient video prediction in both online and offline settings. LiteEval is a coarse-to-fine framework that contains a coarse LSTM and a fine LSTM organized hierarchically, as well as a gating module. In particular, LiteEval operates on compact features computed at a coarse scale and dynamically decides whether to compute more powerful features for incoming video frames to obtain more details with a gating module. The two LSTMs are further synchronized such that the fine LSTM always contains all information seen so far that can be readily used for predictions. Extensive experiments are conducted on FCVID and ActivityNet and the results demonstrate the effectiveness of the proposed approach.

Chapter 5: Weakly-Supervised Spatial Context Networks

5.1 Introduction

Recent successful advances in object categorization, detection and segmentation have been fueled by high capacity deep learning models (*e.g.*, CNNs) learned from massive labeled corpora of data (*e.g.*, ImageNet [23], COCO [91]). However, the large-scale human supervision that makes these methods effective at the same time, limits their use; especially for fine-grained object-level tasks such as detection or segmentation, where annotation efforts become costly and unwieldily at scale. One popular solution is to use a pre-trained model (*e.g.*, VGG_19 trained on ImageNet) for other, potentially unrelated, image tasks. Such pre-trained models produce effective and highly generic feature representations [99, 100]. However, it has also been shown that fine-tuning with task-specific labeled samples is often necessary [101].

Unsupervised learning is one way to potentially address some of these challenges. Unfortunately, despite significant research efforts unsupervised models such as auto-encoders [102, 103] and, more recently, context encoders [104] have not produced representations that can rival pre-trained models (let alone beat them). Among the biggest challenges is how to encourage a representation that captures

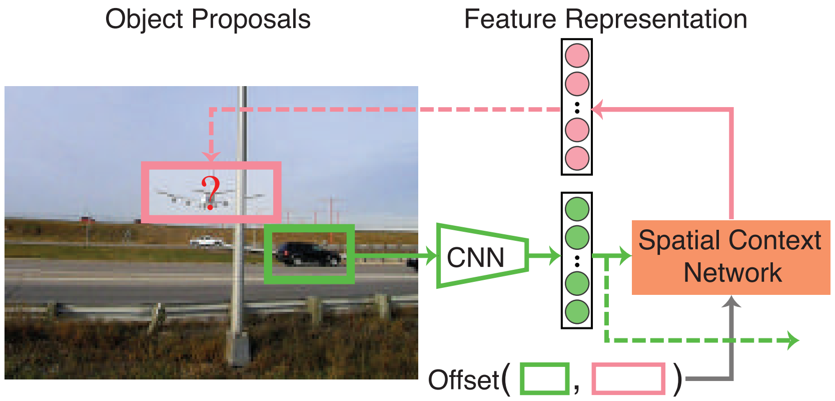


Figure 5.1: **Illustration of the proposed spatial context network.** A CNN used to compute feature representation of the green patch is fine-tuned to predict feature representation of the red patch using the proposed spatial context module, conditioned on their relative offset. Pairs of patches used to train the network are obtained from object proposal mechanisms. Once the network is trained, the green CNN can be used as a generic feature extractor for other tasks (dotted green line).

semantic-level (*e.g.*, object-level) information without having access to explicit annotations for object extent or class labels.

In the text domain, the idea of local spatial context within a sentence, proved to be an effective supervisory signal for learning distributed word vector representations (*e.g.*, continuous bag-of-words (CBOW) [105] and skip-gram models [105]). The idea is conceptually simple; given a word tokenized corpus of text, learn a representation for a target word that allows it to predict representations of contextual words around it; or vice versa, given contextual words to predict a representation of the target word. Generalizing this idea to images, while appealing, is also challenging as it is not clear how to 1) *tokenize* the image (*i.e.*, what is an elementary entity between which context supervision should be applied) and 2) apply the notion of context effectively in a 2-dimensional real-valued domain.

Recent attempts to use spatial context as supervision in vision, resulted in models that used (regularly sampled) image patches as *tokens* and either learned a representation that is useful for classifying contextual relationships between them [106] or attempted to learn representations that fill in an image patch based on the larger surrounding pixels [104]. In both cases, the resulting feature representations fail to perform at the level of the pre-trained ImageNet models. This could be attributed to a number of reasons: 1) spatial context may indeed not be a good supervisory signal; 2) generic and neighboring image patches may not be an effective *tokenization* scheme; and/or 3) it may be difficult to train a model with a contextual loss from scratch.

Our motivation is similar to [104, 106]; however, we posit that image *tokenization* is important and should be done at the level of objects. By working with patches at object scale, our network can focus on more object-centric features and potentially ignore some of the texture and color detail that are likely less important for semantic tasks. Further, instead of looking at immediate regions around the patch for context [104] and encoding the relationship between the contextual and target regions implicitly, we look at potentially non-overlapping patches with longer spatial contextual dependencies and explicitly condition the predicted representation on the relative spatial offset between the two regions. In addition, when training our network, we make use of a pre-trained model to extract intermediate representations. Since lower levels of CNNs have been shown to be task independent, this allows us to learn a better representation.

Specifically, we propose a novel architecture – Spatial Context Network (SCN) – which is built on top of existing CNN networks and is designed to predict a representation of one (object-like) image patch from another (object-like) image patch, conditioned on their relative spatial offset. As a result, the network learns a spatially conditioned contextual representation of image patches. In other words, given the same input patch and different spatial offsets it learns to predict different contextual representations (*e.g.*, given a patch depicting a side-view of a car and a horizontal offset, the network may output a patch representation of another car; however, the same input patch with a vertical offset may result in a patch representation of a plane). We also make use of ImageNet pre-trained model as both an initialization and to define intermediate representations.

Once an SCN model is trained (on pairs of patches), we can use one of its two streams as an image representation that can be used for a variety of tasks, including object categorization or localization (*e.g.*, as part of Faster R-CNN [107]). This setting allows us to definitively answer the question of whether spatial context can be an effective supervisory signal – it can, improving on the original ImageNet pre-trained models.

Contributions: Our main contribution is the spatial context network (SCN), which differs from other models in that it uses two offset patches as a form of contextual supervision. Further, we explore a variety of tokenization schemes for mining training patch pairs, and show that an object proposal mechanism is the most effective. This observation validates the intuition that for semantic tasks, context is most useful at the object scale. Finally, we conduct extensive experiments to investigate the capacity of the proposed SCN for capturing context information in images, and demonstrate its ability to improve, in an unsupervised manner, on ImageNet pre-trained CNN models for both categorization (on VOC2007 and VOC2012) and detection (on VOC2007), where the bottom stream of the trained SCN is used as a generic feature extractor (see Figure 6.2 (bottom)).

5.2 Related Work

Unsupervised Learning. Auto-encoders [108] are among the earliest works in unsupervised deep learning. Auto-encoders typically learn a representation by employing an encoder-decoder architecture; the encoder encodes the image (or patch) into a compact hidden state representation and the decoder reconstructs it back to a full image. As such, auto-encoders learn a representation that attempts to preserve as much image content as possible in the compact hidden state. De-noising auto-encoders [103] reconstruct images (or patches) subject to local corruptions. The most extreme variant of de-noising auto-encoders are the context encoders [104], which aim to reconstruct a large hole (patch) given its surrounding spatial context.

A number of papers proposed to learn representations by converting the generative auto-encoder-like objectives to discriminative classification counterparts, where CNNs have been shown to learn effectively. For example, [109] proposed an idea of surrogate classes that are formed by applying a variety of transformations to randomly sampled image patches. Classification into these surrogate classes is used as a supervisory signal to learn image representations. Alternatively, in [106], neighboring patches are used in Siamese-like networks to predict the relative *discrete* (*e.g.*, to the top-right, bottom-left, *etc.*) location of patches. Related, is also [110] that attempts to learn a similarity function across patches using various deep learning architectures, including center-surround (similar to [104]) and forms of Siamese networks. Goodfellow *et al.* [111] proposed Generative Adversarial Networks (GAN) that contain a generative model and discriminative model. Pathak *et al.* [104] built upon GANs to model context through inpainting missing patches.

Our model is related to auto-encoders [108], and particularly context encoders [104], however, it is conceptually somewhere between the discriminative and generative forms discussed above. We have encoder and decoder components, but instead of decoding the hidden state all way to an image, our decoder decodes it to an intermediate discriminatively trained representation. Further, unlike previous methods, our decoder takes real-valued patch offset as input, in addition to the representation of the patch itself.

Pre-trained Models. Pre-trained CNN models have been shown to generalize to a large number of different tasks [99, 100]. However, their transferability, as was noted in [112], is affected by specialization of higher layer neurons to the original task (often ImageNet categorization). By taking a network pre-trained on the ImageNet task and using its intermediate representation as target for our decoder, we make use of the knowledge distilled in the network [113] while attempting to improve it using spatial context. Works like [114] and [115] attempt to similarly re-use lower layers [114] of the pre-trained network and fine-tune, typically, fully-connected layers to specific tasks (*e.g.*, object detection). However, such models assume some labeled data in the target domain, if not for classes of interest [114], then for related ones [115]. In our case, we assume no supervision of this form. Instead, we just assume that there exists a process that can generate category agnostic object-like proposal patches. Our work is similar to [116] that also attempts to improve the performance of pre-trained models. While they augment existing networks with reconstructive decoding pathways for image reconstruction, our model focuses on exploiting contextual cues in images.

Weakly-supervised and Self-supervised Learning. Recent years have witnessed a growing trend in weakly-supervised and self-supervised learning, which attempt to achieve similar performance to fully supervised models with limited use of annotated labels. A typical setting is to, for example, use image-level annotations to learn an object detection model [117, 118, 119, 120, 121, 122]. However, such models typically rely on latent variables and appearance regularities present within individual object class. In addition, researchers also utilized motion coherence (tracked patches [123] or ego-motion from sensors [124]) in videos as supervisory signals to train network. Zhang *et al.* [125] proposed to generate a color version of a grayscale photo through a CNN model, which could further serve as an auxiliary task for feature learning. Different from these works, we experiment with (category-independent) object proposals as a way to *tokenize* an image into more

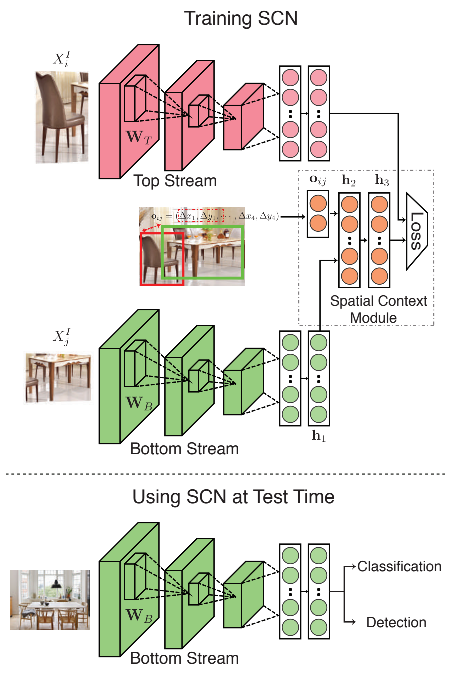


Figure 5.2: **Overview of the proposed spatial context network architecture.** See texts for complete description and discussion.

semantically meaningful parts. This can be thought of as (perhaps) a very weak form of supervision, but unlike any that we are aware has been used before.

5.3 Approach

We now introduce the proposed spatial context network (see Figure 6.2 (top)), which consists of a top stream and a bottom stream operating on a pair of patches cropped from the same image. The goal is to utilize their spatial layout information as contextual clues for feature representation learning. Once the spatial context network is learned, the bottom stream can be used as a feature extractor (see Figure 6.2 (bottom)) for a variety of image recognition tasks, specifically, object categorization and detection.

More formally, given a patch $\mathbf{X}_i^I$ extracted from an image $I \in \mathbb{I}$, where $\mathbb{I}$ is the training set, we denote the patch bounding box $\mathbf{b}_i^I$ as an eight-tuple consisting of (x, y) positions of top-left, top-right, bottom-left and bottom-right corners. We can then denote the training samples for the network as 3-tuples ($\mathbf{X}_i^I$, $\mathbf{X}_j^I$, $\mathbf{o}_{ij}^I$), where $\mathbf{o}_{ij}^I = \mathbf{b}_i^I - \mathbf{b}_j^I$ is the relative offset between two patches computed by subtracting locations of their respective four corners.

Top stream. The goal of the top stream is to provide a feature representation for patch $\mathbf{X}_i^I$ that will be used as soft *target* for contextual prediction by the *learned* representation of the patch $\mathbf{X}_j^I$. This stream consists of an ImageNet pre-trained state-of-the-art CNN such as VGG_19, GoogleNet or ResNet (any pre-trained CNN model can be used). More specifically, the output of the top stream is the representation from the fully-connected layer (fc_7) obtained by propagating patch $\mathbf{X}_i^I$ through the original pre-trained ImageNet model (here we remove the softmax layer). More formally, let $g(\mathbf{X}_i^I; \mathbf{W}_T)$ denote the non-linear function approximated by the CNN model and parameterized by weights $\mathbf{W}_T$. Note that one can also utilize representation of other layers; we use fc_7 for simplicity and because of its superior performance in most high-level visual tasks [100].

Bottom stream. The bottom stream consists of an identical CNN model to the top stream which feeds into the spatial context module. The spatial context module then accounts for spatial offset between the input pair of patches. The network first maps the input patch to a feature representation $\mathbf{h}_1 = g(\mathbf{X}_j^I; \mathbf{W}_B)$ and then the resulting $\mathbf{h}_1$ (fc_7 representation) is used as input for the spatial context module.

We initialize the bottom stream with the ImageNet pre-trained model as well, so initially, $\mathbf{W}_B = \mathbf{W}_T$. However, while $\mathbf{W}_T$ remains fixed, $\mathbf{W}_B$ is optimized during training.

Spatial Context Module. The role of the spatial context module is to take the feature representation of the patch $\mathbf{X}_j^I$ produced by the bottom stream and, given the offset to patch $\mathbf{X}_i^I$, predict the representation of patch $\mathbf{X}_i^I$ that would be produced by the top stream. The spatial context module is represented by a non-linear function $f([\mathbf{h}_1, \mathbf{o}_{ij}^I]; \mathbf{V})$, parameterized by weight matrix $\mathbf{V} = \{\mathbf{V}_1, \mathbf{V}_{loc}, \mathbf{V}_2\}$.

In particular, the spatial context module first takes the feature vector $\mathbf{h}_1$ (computed from patch $\mathbf{X}_j^I$) together with the offset vector $\mathbf{o}_{ij}$ between $\mathbf{X}_j^I$ and $\mathbf{X}_i^I$ to derive an encoded representation:

$$\mathbf{h}_2 = \sigma(\mathbf{V}_1\mathbf{h}_1 + \mathbf{V}_{loc}\mathbf{o}_{ij}), \tag{5.1}$$

where $\mathbf{V_1}$ denotes the weights for $\mathbf{h}_1$; $\mathbf{V}_{loc}$ is the weight matrix for the input offset, and $\sigma(x) = 1/(1+e^{-x})$. (Note that we absorb the bias term in the weight matrix for convenience). Finally, $\mathbf{h}_2$ is mapped to $\mathbf{h}_3$ with a linear transformation to reconstruct the fc_7 feature vector computed by the top stream on the patch $\mathbf{X}_i^I$.

Loss Function. Given the output feature representations from the aforementioned two streams, we train the network by regressing the features from the bottom stream to those from the top stream. We use a squared loss function:

$$\min_{\mathbf{V},\mathbf{W}_B} \sum_{I\in\mathbb{I};i\neq j} \left\| g(\mathbf{X}_i^I;\mathbf{W}_T) - f([g(\mathbf{X}_j^I;\mathbf{W}_B),\mathbf{o}_{ij}];\mathbf{V}) \right\|^2 . \tag{5.2}$$

The model is essentially an encoder-decoder framework with the bottom stream encoding the input image patch into a fixed representation and spatial context module decoding it to representation of another, spatially offset, patch. The intuition comes from the skip-gram model [126] that attempts to predict the context given a word, which has been demonstrated to be effective for a number of NLP tasks. Since objects often co-occur in images in particular relative locations, it makes intuitive sense to explore such relations as contextual supervision.

The network can be easily trained using back-propagation with stochastic gradient descent. Note that for the top stream, rather than predicting raw pixels in images, we utilize the features extracted from off-the-shelf CNN architecture as *ground truth*, to which the features constructed by the bottom stream regress. This is because the pre-trained CNN model contains valuable semantic information (*e.g.*, referred to as dark knowledge [113]) to differentiate objects and the extracted off-the-shelf features have achieved great success on various tasks [127, 128].

One alternative to formulating the problem as a regression task would be to turn it into a classification problem by appending a softmax layer on top of the two streams and predicting whether a pair of features is likely given the spatial offset. However, this would require a large number of negative samples (*e.g.*, a *car* is not likely to be in a *lake*), making training difficult. Further, our regression loss also builds on intuitions explored in [113], where it is shown that soft real-valued targets are often better than discrete labels.

Implementation Details. We adopt two off-the-shelf CNN architectures, CNN_M and VGG_19 [57], to train the spatial context network. CNN_M is an AlexNet [56] style CNN with five convolutional layers topped by three fully-connected layers (the dimension for fc_6 and fc_7 is $2,048$), but contains more convolutional filters. VGG_19 network consists of 16 convolutional layers followed by three fully-connected layers, possessing stronger discriminative power.

The pipeline was implemented in Torch and we apply mini-batch stochastic gradient descent in training with the batch size of 64. The weights for the spatial context module are initialized randomly. We fine-tune the fully-connected layers in the bottom stream CNN model with convolutional layers fixed, unless otherwise specified. The input patches are resized to 224×224. We set the initial learning rate to $1e^{-3}$, which is decreased to $1e^{-4}$ after 100 epochs; we fix weight decay to $5e{-4}$ and the maximum number of epochs to 200. We will discuss patch selection in Experiements.

Using SCN for Classification and Detection. Once the SCN is trained, we use $\mathbf{h}_1$ from the bottom stream as a feature representation for other tasks (Figure 6.2 (bottom)). As we will show, these feature representations are better than those obtained from the original ImageNet pre-trained model for object detection and classification.

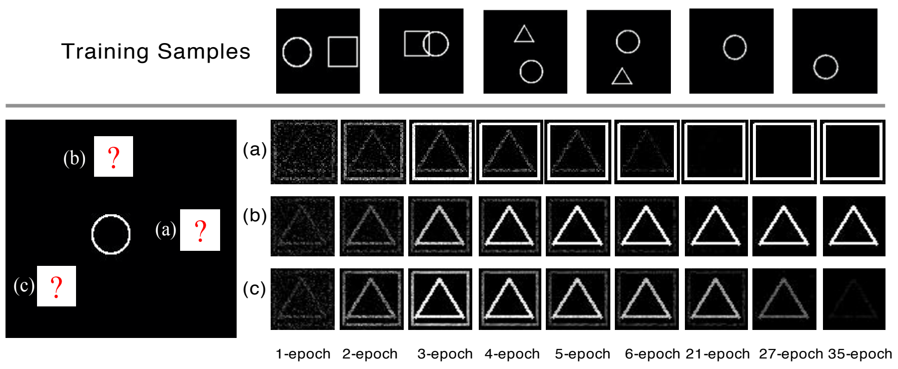


Figure 5.3: **Experiments with synthetic dataset.** Training samples are shown in top row. Bottom rows show predicted patches for the labeled regions on the left, after 1–35 epochs of training. Predicted patches are obtained by treating the circle in the middle and an appropriate spatial offset to (a), (b), or (c) as input to an SCN and visualizing the output $\mathbf{h}_3$ layer.

5.4 Experiments

We first validate the ability of the proposed SCN to learn context information on a synthetic dataset and with the real images from VOC2012. We then evaluate the effectiveness of features extracted from the spatial context framework in classification and detection tasks, as compared with original pre-trained ImageNet features, and competing state-of-the-art feature learning methods.

5.4.1 Synthetic Dataset Experiments

.

We construct a synthetic dataset containing *circles*, *squares* and *triangles* to verify whether the proposed spatial context framework is able to learn correlations in spatial layout patterns of these objects. More specifically, we create 300 (circle, square) pairs where circles are always horizontally offset (see Figure 5.3 (top)) from the squares (vertical difference is within 30 pixels); and 300 (circle, triangle) pairs

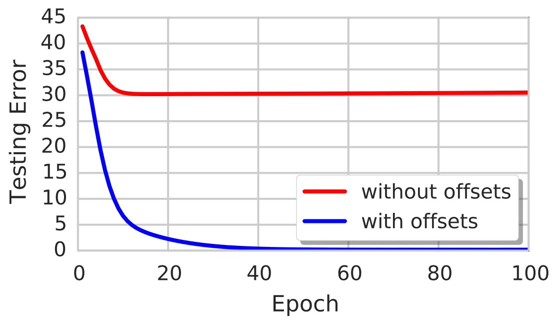


Figure 5.4: **Testing error on the synthetic dataset.** Illustrated is the testing error with and without offset vector in the input.

where circles are vertically offset from the triangles (horizontal difference is within 30 pixels); as well as 200 (circles, black image) pairs where the offset vector is randomly sampled. We randomly split the dataset into 600 training and 200 testing pairs. We assume perfect proposals and crop patches tightly around the objects (circles, squares and triangles). Here, we adopt the CNN_M model only.

The testing error loss (mean squared error) on this dataset is visualized in Figure 5.4. As we can see from the figure, the testing error of the spatial context network steadily decreases for the first 20 epochs and nearly reaches zero after 25 epochs. To investigate the role offset vectors play in the learning process, we remove the offset vector from the input and retrain the network. The loss of this network stabilizes to 30 after 10 epochs; this is significantly higher than the error of the spatial context network. Figure 5.4 confirms that the proposed spatial context network can make effective use of the spatial context information between objects.

To gain further insights into the learning process, we replace the *target* feature representation of the top stream with raw ground truth image patches (see Figure 5.5). After each epoch, given an input bottom stream object patch (depicting

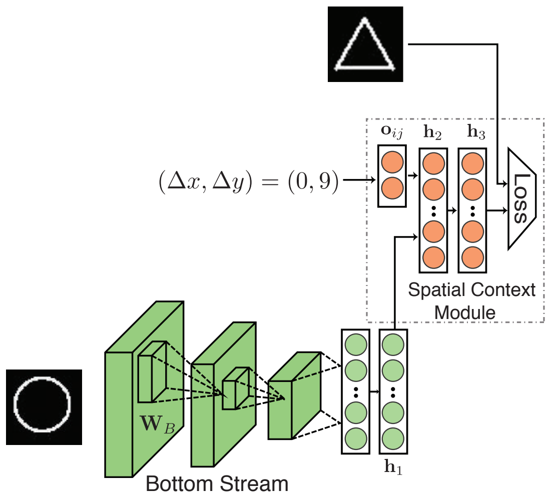


Figure 5.5: **Reconstruction.** Illustrated is the framework for reconstructing images in the top stream.

circle) and an offset vector from the testing set, we adopt the output of the last layer $\mathbf{h}_3$ in the SCN to reconstruct images for the top stream. The results are visualized in Figure 5.3 (bottom) for three different spacial offsets (a), (b) and (c).

When circles are combined with either horizontal or vertical offsets, the network is able to reconstruct square and triangle patches (respectively) after about five epochs of training. For the first few epochs, both triangles and squares co-occur in the constructed images, but clear square and triangle patterns emerge as the training proceeds. It took longer for the network to learn that conditioned on an off-axis offset vector and a circle patch it should produce an empty (black) patch image. This experiment validates that our spatial context network is able to learn correct spatially varying contextual representation based on (identical) input patch (circle) and varying offsets. Without providing location offset information, the network overfits and simply generates a patch containing overlapping triangles and squares (which explains the poor convergence in Figure 5.4).

Imagining a circle is a car, a square a tree and the triangle (which is above circle) to be sky, this synthetic dataset provides a simplified version of spatial context information in real-world scenarios. The experiments indicate that the varying spatial contextual information among multiple objects can be learned by the SCN.

5.4.2 Modeling Context in Real Images

We now discuss context modeling in real images and validate the capability of the network to capture such real-world contextual clues. To this end, we use the PASCAL VOC 2012 [129] dataset, which consists of a training set with 5,717 images and a validation set with 5,823 images, totaling 20 object categories (denoted by VOC2012-Img). We first crop objects from the original images on both subsets using the provided annotations of bounding boxes, which leads to 15,774 objects for training and 15,787 objects for testing (denoted by VOC2012-Obj[1]). Objects from the same image are further paired and are used as inputs for the spatial context network (SCN) together with their offset vector. In total, we obtain 34,378 training and 34,722 testing paired samples (VOC2012-Pairs).

We first train the spatial context network using paired images. Given the trained network, we compute the outputs of the last layer from the spatial context module (*i.e.*, $\mathbf{h}_3$) as the synthesized/predicted feature representations for a single patch in the top stream (on both training and test set).

[1]The difference between VOC2012-Obj and VOC2012-Img is that in the former the objects are cropped, where as in the latter they are not.

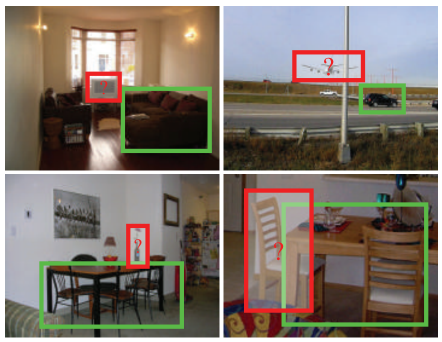
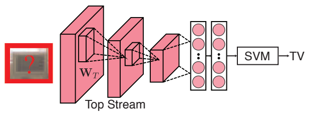
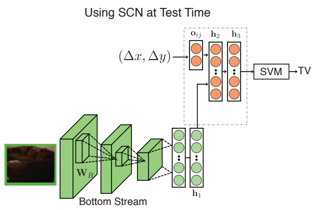


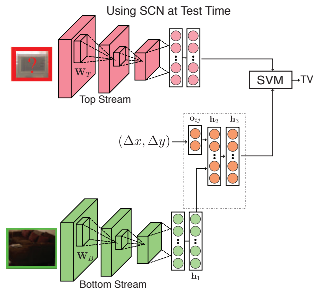
Figure 5.6: **SCN contextual classification.** Features of the top stream (red boxes) are predicted using patches from bottom stream (green boxes) and offset vector as inputs to the trained SCN. A classifier is then trained to predict the label of the red patch based on the *predicted* features from the training set. Performance on testing set is 56.3% (Table 5.1).



(a) Classification with original VGG fc7 top stream features.



(b) Classification using SCN predicted features of the top stream.



(c) Classification using combination of VGG fc7 top stream features and SCN predicted features

Figure 5.7: **Models used to compare classification.** Illustrated models are those evaluated in Table 5.1; in the same respective order, where (a) is the baseline model that uses original VGG features of the top stream; (b) is model that is using SCN predicted features for the patches instead, and (c) combines the two.

Then we train a linear classifier with the extracted features using all training patches in the top stream (See Figure 5.6 for illustration and Figure 5.7 (b) for model design). To establish a baseline, for all patches in the top stream, we compute the raw fc_7 features from the original VGG_19 network and similarly train a linear SVM classifier (See Figure 5.7 (a)). The results are summarized in Table 5.1.

features	VOC2012-Pairs (%)
VGG_19 fc_7	78.3
SCN predicted ($\mathbf{h}_3$) features	56.3
VGG_19 fc_7 + SCN predicted	79.5

Table 5.1: **Performance comparisons of classification.** Different feature representations for the *top* patch classification are compared. SCN predicted features are obtained by regressing *top* stream features from the contextual *bottom* stream patch.

It is surprising to see that the predicted features achieve a 56.3% accuracy in object classification given the fact that these features are *predicted* from nearby objects within the same image (from the bottom stream) using the trained spatial context network (SCN). In other words we are able to recognize objects at 56.3% accuracy without ever seeing the real image features contained in the corresponding image patches; the recognition is done purely based on the *contextual* predictions of those features from other patches (note that 92.6% of patches do not or minimally overlap (< 0.2 IoU)). This indicates *very strong* contextual information that our network was able to learn.

To eliminate the possibility that accuracy comes from images containing multiple instances of the same object, we analyzed the dataset and found only 45% of training and 42% of testing image patch pairs correspond to the same objects. Further, using pairs that do not contain same objects produces an accuracy of 52.8%, and 63.2% with pairs only from the same objects.

To investigate whether the synthesized features $\mathbf{h}_3$ contain contextual information that might be complementary to the original fc_7 features, we perform feature fusion by concatenating the two representations into a 8,192-dimensional vector and training a linear SVM for classification (See Figure 5.7 (c)). We observe 1.2% performance gain compared with raw VGG fc_7 features, which again confirms that context is beneficial.

5.4.3 Feature Learning with SCN for Classification

In the last two sections, to verify the effectiveness of spatial contextual learning, we assumed knowledge of object bounding boxes (but, importantly, not their categorical identity); in other words, we assumed existence of a perfect object proposal mechanism; this is clearly unrealistic. In this section, we explore the importance/significance of the quality of the object proposal mechanism on the performance of features learned using SCN. We do so in the context of classification, where once SCN is trained, we use SVM on top of generic SCN features (see Figure 6.2 (bottom)).

We use ground truth bounding boxes, provided by the dataset, as a baseline (*SCN-BBox*). In addition, we test the following object proposal methods:

- **Random Patches** (*SCN-Random*): We randomly crop 5 patches of size of 64×64 in each image (consistent with [104]) to generate 10 patch pairs per image. In total, we collect 28K cropped patches and 57K pairs.[2]

- **Edge Box** [130] (*SCN-EdgeBox*): EdgeBox is a generic method to generate object bounding box proposals based on edge responses. We filter out the

[2]Note that in the pairing process one could simply swap the inputs of the top and bottom stream to double the number of pairs for the network, however, empirically, we found it not to be helpful.

	features-fc_7	VOC2012-Obj	VOC2012-Img
CNN_M	Original	75.3	68.5
	SCN-BBox	78.7	70.8
	SCN-YOLO	79.2	70.7
	SCN-EdgeBox	**79.9**	**72.8**
	SCN-Random	78.8	70.0
VGG_19	Original	81.4	78.1
	SCN-BBox	82.6	78.8
	SCN-YOLO	83.0	79.0
	SCN-EdgeBox	**83.6**	**79.5**
	SCN-Random	83.2	79.2

Table 5.2: **Performance with various object proposals.** Comparison of classification with features obtained using SCN trained with different patch selection mechanisms is illustrated on VOC2012-Obj and VOC2012-Img, using two CNN architectures.

bounding boxes with confidence lower than 0.1 and those with irregular aspect ratio, leading to 43K *object* patches and 160K pairs for training.

- **YOLO** [131] (*SCN-YOLO*): YOLO is a recently introduced end-to-end framework trained on VOC for object detection. We use YOLO as an object proposal mechanism, by taking patches from detection regions but ignoring the detected labels. We collect 13K objects forming 17K image patch pairs.

We expect the quality of object proposal methods (from least object-like to most object-like) on VOC to roughly follow the following pattern:

$$\textit{Random} < \textit{EdgeBox} < \textit{YOLO} < \text{ground-truth } \textit{BBox}.$$

Given a trained SCN model, we utilize the bottom stream (see Fig. 6.2 (bottom)) to test generalization of the learned feature representations, by performing

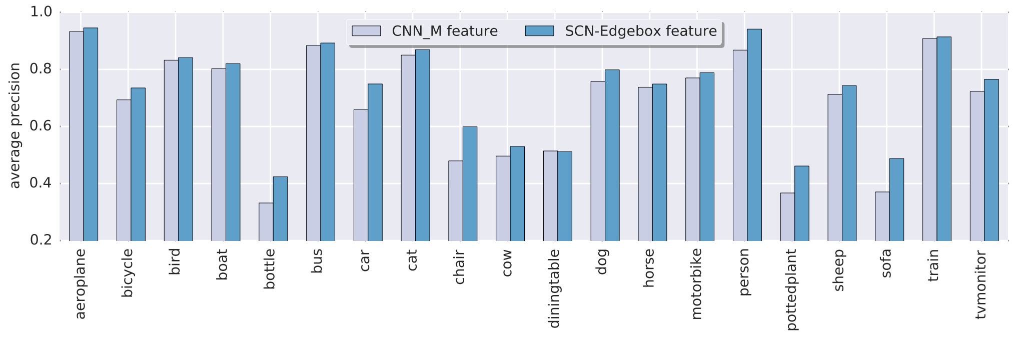


Figure 5.8: **Classification per class performance.** Reported is average precision obtained using original CNN_M features and SCN-EdgeBox features on VOC2012-Img.

classification with linear SVMs on VOC2012-Obj and VOC2012-Img (see footnote 1 for explanation) with the outputs from the first hidden layer ($\mathbf{h}_1$, *i.e.*, fine-tuned version of fc_7) in the bottom stream of SCN. The results are measured in mAP. We compare the different patch selection mechanisms discussed above and also to the original ImageNet pre-trained models. The results are summarized in Table 5.2. We observe that SCN-BBox and SCN-YOLO achieve better results compared with the original fc_7 features. It is also surprising to see that SCN-EdgeBox obtains the best performance, even higher than models trained with ground-truth bounding boxes. It is 4.6 and 4.3 percentage points better than the original fc_7 features on VOC2012-Obj and VOC2012-Img respectively.

We believe that better performance of the SCN-EdgeBox stems from Edge-Box's ability to select object-like regions that go beyond the 20 object classes labeled in ground truth and detected by YOLO. We also note that while Random patch sampling also improves the performance, with respect to the original ImageNet pre-trained network, it is doing so by a much smaller margin than EdgeBox patch sampling.

The original fc_7 features are trained using labels from ImageNet; our spatial context network is appealing in that it learns a better feature representation by exploiting contextual cues without any additional explicit supervision. Figure 5.8 compares the per-class performance of SCN-EdgeBox and the original fc_7 features on VOC2012-Img, where we can see that SCN-EdgeBox features outperform the original fc_7 features for all classes. It is also interesting to see that, for small objects, such as "bottle" and "potted plant", the performance gain of SCN-EdgeBox is more significant.

	VOC2012-Obj
VGG_19 fc_7	81.4
SCN-EdgeBox (fc_6, fc_7)	83.6
SCN-EdgeBox (fc_6, fc_7, $conv_5$)	84.3
SCN-EdgeBox (all layers)	82.5

Table 5.3: **Exploring SCN learning strategies.** Classification performance based on features obtained using different fine-tuning strategies. See text for more details.

Fine-tuning Convolutional Layers. In addition to only fine-tuning the fully-connected layers of the bottom stream CNN model, we also explore whether joint training with VGG_19 network could further improve the performance of the extracted features. More specifically, for the top stream we fix the weights since computing features dynamically poses challenges for network convergence. Further, this avoids *trivial* solutions of both streams learning, for example, to predict zero features for all patches. In addition, this makes use of transferability of lower levels of pre-trained CNN models as targets for the bottom stream decoding. The results are summarized in Table 5.3. By back-propagating the error through deeper layers we observe a significant performance gain (2.9 percentage points) over the original features of VGG_19 network, which confirms the fact that SCN is effective and VGG layers could be fine-tuned jointly for specific tasks in order to gain better performance using our formulation. When fine-tuning all layers in the network, the performance of SCN degrades slightly to 82.5%.

	Initialization	Supervision	Pretraining time	Classification	Detection
Random Gaussian	random	N/A	$<$ 1 minute	53.3	43.4
Wang *et al.* [123]	random	motion	1 week	58.4	44.0
Doersch *et al.* [106]	random	context	4 weeks	55.3	46.6
*Doersch *et al.* [106]	1000 class labels	context	–	65.4	50.4
Pathak *et al.* [104]	random	context inpainting	14 hours	56.5	44.5
Zhang *et al.* [125]	random	color	–	65.6	46.9
ImageNet [104]	random	1000 class labels	3 days	78.2	56.8
*ImageNet	random	1000 class labels	3 days	76.9	58.7
SCN-EdgeBox	1000 class labels	context	10 hours	79.0	**59.4**

Table 5.4: **Quantitative comparison for classification and detection on VOC 2007.** Classification and Fast-RCNN detection results are on the PASCAL VOC 2007 test set. The baselines labeled with * are based on our experiments, rest taken from original papers.

5.4.4 Feature Learning with SCN for Detection

We also explore the applicability of SCN features for object detection tasks to verify generic feature effectiveness. To make fair comparisons with prior work, we adopt the experimental setting of [104] and fine-tune the SCN-EdgeBox model (based on CNN_M architecture) on Pascal VOC2007, which is then applied in the *Fast R-CNN* [107] framework. More precisely, we replace the ImageNet pre-trained

CNN_M model with the fine-tuned bottom stream in SCN (See Figure 6.2 (bottom)). The weights for final classification and bounding box regression layers are initialized from scratch. Following the training and testing protocol defined in [107], we finetune layers conv2 and up and report detector performance in mAP.

The results and comparisons with existing state-of-the-art methods are summarized in Table 5.4. SCN-EdgeBox model improves on the original ImageNet pre-trained model by 0.7 percentage points. Further, compared with alternative unsupervised learning methods, our approach achieves significantly better performance. We also significantly outperform other feature training methods on classification (including our fine-tuned ImageNet model) and Doersch *et al.* [106] model initialized with ImageNet.

Figure 5.9 visualizes some sample images where SCN-EdgeBox outperforms the pre-trained ImageNet model. Our model is better at detecting relatively small objects (*e.g.*, airplane in the first row and chair in the second row).

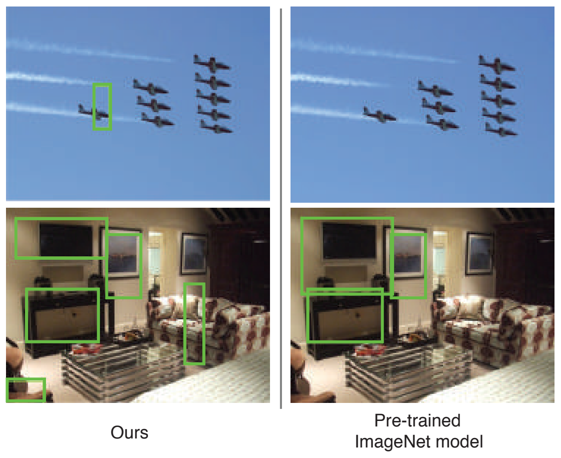


Figure 5.9: **Sample detection results.** Illustrated are results obtained using SCN-EdgeBox model and the original pre-trained ImageNet model, respectively, on VOC2007.

5.5 Conclusion

In this chapter, we presented a novel spatial context network built on top of existing CNN architectures. The SCN network exploits implicit contextual layout cues in images as a supervisory signal. More specifically, the network is trained to predict the intermediate representation of one (object-like) image patch from another (object-like) image patch, within the same image, conditioned on their relative spatial offset. Consequently, the network learns a spatially conditioned contextual representation of image patches. Extensive experiments are conducted to validate the effectiveness of the proposed spatial context network in modeling context information in images. We show that the proposed spatial context network can achieve improvements (with no additional explicit supervision) over the original ImageNet pre-trained models in object categorization on VOC2007 / VOC2012 and detection on VOC2007.

Chapter 6: DCAN: Dual Channel-wise Alignment Networks for Unsupervised Scene Adaptation

6.1 Introduction

Deep neural networks have driven recent advances in computer vision. However, significant boosts in accuracy achieved by high-capacity deep models require large corpora of manually labeled data such as ImageNet [23] and COCO [132]. The need to harvest clean and massive annotations limits the ability of these approaches to scale, especially for fine-grained understanding tasks like semantic segmentation, where dense annotations are extremely costly and time-consuming to obtain. One possible solution is to learn from synthetic images rendered by modern computer graphics tools (*e.g.*, video game engines), such that ground-truth labels are readily available. While synthetic data have been exploited to train deep networks for a multitude of tasks like depth estimation [133], object detection [134], *etc.*, the resulting models usually suffer from poor generalization when exposed to novel realistic samples. The reasons are mainly two-folds: (1) the realism of synthesized images is limited—inducing an inherent gap between synthetic and real image distributions; (2) deep networks are prone to overfitting in the training stage, which leads to limited generalization ability.

Learning a discriminative model that reduces the disparity between training and testing distributions is typically known as domain adaptation; a more challenging setting is unsupervised domain adaptation that aims to bridge the gap without accessing labels of the testing domain during training. Most existing work seeks to align features in a deep network of the *source* domain (training sets) and the *target* domain (testing sets) by either explicitly matching feature statistics [135, 136, 137] or implicitly making features domain invariant [138, 139]. Recent work also attempts to minimize domain shift in the pixel space to make raw images look alike [140, 141, 142] with adversarial training. While good progress has been made for classification, generalizing these ideas to semantic segmentation has been shown to be less effective [143], possibly due to the fact that high-dimensional feature maps are more challenging to align compared to features used for classification from fully-connected layers.

In this chapter, we study unsupervised domain adaptation for semantic segmentation, which we refer as unsupervised scene adaptation. We posit that channel-wise alignment of high-level feature maps is important for adapting segmentation models, as it is able to preserve spatial structures and consider semantic information like attributes and concepts encoded in different channels [144] independently, which implicitly helps transfer feature distributions between the corresponding concepts across domains. In particular, we build upon recent advances of instance normalization [145] due to its effectiveness and simplicity for style transfer [145, 146, 147].

Instance normalization is motivated by the fact that mean and standard deviation in each channel of CNN feature maps contain the style information of an image, and hence they are used to translate feature maps of a source image into a normalized version based on a reference image for each channel. In addition to being able to match feature statistics, the ability to maintain spatial structures in feature maps with channel-wise normalization makes it appealing for tasks like segmentation.

Motivated by these observations, we propose to reduce domain differences at both low-level and high-level through channel-wise alignment. In particular, we normalize features of images from the source domain with those of images from the target domain by matching their channel-wise feature statistics. Nevertheless, such alignment is on a per image basis with each target sample serving as a reference for calibration. When multiple images exist in the target domain, a straightforward way is to enumerate all of them to cover all possible variations, which is computationally expensive. In contrast, we stochastically sample from the target domain for alignment. The randomization strategy is not only efficient, but more importantly, provides a form of regularization for training in similar spirit to stochastic depth [31], data transformation [148, 149], and dropout [29].

To this end, we present, Dual Channel-wise Alignment Networks (DCAN), a simple yet effective framework optimized in an end-to-end manner. The main idea is leveraging images from the target domain for channel-wise alignment, which not only enables minimizing the low-level domain discrepancies in pixel space (*e.g.*, color, texture, lighting conditions, *etc.*), but also, simultaneously normalizes high-level feature maps of source images specific to those of target images for improved segmentation. Figure 6.2 gives an overview of the framework. In particular, we utilize an image generator to map an image from the source domain to multiple representations with the same content as the input but in different styles, determined by unlabeled images randomly selected from the target set. These synthesized images, resembling samples from the target domain, together with sampled target images, are further input into a segmentation network, in which channel-wise feature alignment is performed once more to refine features for the final segmentation task.

Contributions: The key contributions of DCAN are summarized as follows: (1) we present an end-to-end learning framework, guided by feature statistics of images from the target domain, to synthesize new images as well as normalize features on-the-fly for unsupervised scene adaptation; (2) we demonstrate that channel-wise feature alignment, preserving spatial structures and semantic concepts, is a simple yet effective way to reduce domain shift in high-level feature maps. With this, our method departs from much recent and concurrent work, which uses adversarial training for distribution alignment; (3) we conduct extensive experiments by transferring models trained on synthetic segmentation benchmarks, *i.e.*, SYNTHIA [150] and GTA5 [151], to real urban scenes, CITYSCAPES [152], and demonstrate DCAN outperforms state-of-the-art methods with clear margins and it is compatible with several modern segmentation networks.

6.2 Related Work

Unsupervised Domain Adaptation. Most existing work focuses on classification problems and falls into two categories: feature-level and pixel-level adaptation. Feature-level adaptation seeks to align features by either explicitly minimizing the distance measured by Maximum Mean Discrepancies (MMD) [137, 153], covariances [136], *etc.*, between source and target distributions or implicitly optimizing adversarial loss functions in the forms of reversed gradient [154, 155], domain confusion [156], or Generative Adversarial Network [138, 139, 157, 158], such that features are domain-invariant. In contrast, pixel-level domain adaptation attempts to remove low-level differences like color and texture by stylizing source images to resemble target images [140, 159, 160, 161]. Compared to a large amount of work on classification problems, limited effort has been made for semantic segmentation. In [139], adversarial training is utilized to align features in fully convolutional networks for segmentation, and the idea is further extended for both pixel-level and feature-level adaptation jointly using cycle consistency [141]. A curriculum learning strategy is proposed in [143] by leveraging information from global label distributions and local super-pixel distributions. Our work differs from previous work in two aspects: (1) we introduce channel-wise alignment for unsupervised scene adaption, which preserves spatial information and semantic information of each channel when normalizing high-level feature maps for alignment; (2) we avoid adversarial training, which "remains remarkably difficult to train" [162], yet achieves better performance.

Image Synthesis. GANs [111], consisting of a generator and a discriminator optimized to compete with each other, are one of the most popular deep generative models for image synthesis [161, 163, 164]. Various prior information, including labels [165], text [166], attributes [167], images [168, 169] has been explored to condition the generation process. GANs have also been further extended to the problem of image-to-image translation, which maps a given image to another one in a different style, using cycle consistency [3] or a shared latent space [4]. This line of work aims to learn a joint distribution of images from two domains using images from the marginal distributions of each domain. As previously mentioned, adversarial loss functions are hard to train, and hence generating high resolution images is still a challenging problem that could take days [170]. A different direction of image-to-image translation is neural style transfer [147, 171, 172, 173, 174]. Though style transfer can be seen as a special domain adaptation problem with each style as a domain [175], our goal in this work is different: we focus on unsupervised scene adaption, by jointly synthesizing images and performing segmentation with the help of images from the target domain for channel-wise distribution alignment.

6.3 Approach

Given labeled images from a source domain and unlabeled samples from a target domain, our goal is to reduce domain discrepancies at both pixel-level and feature-level. In particular, we leverage unlabeled target images for channel-wise alignment—synthesizing photo-realistic samples to appear as if from the target set,

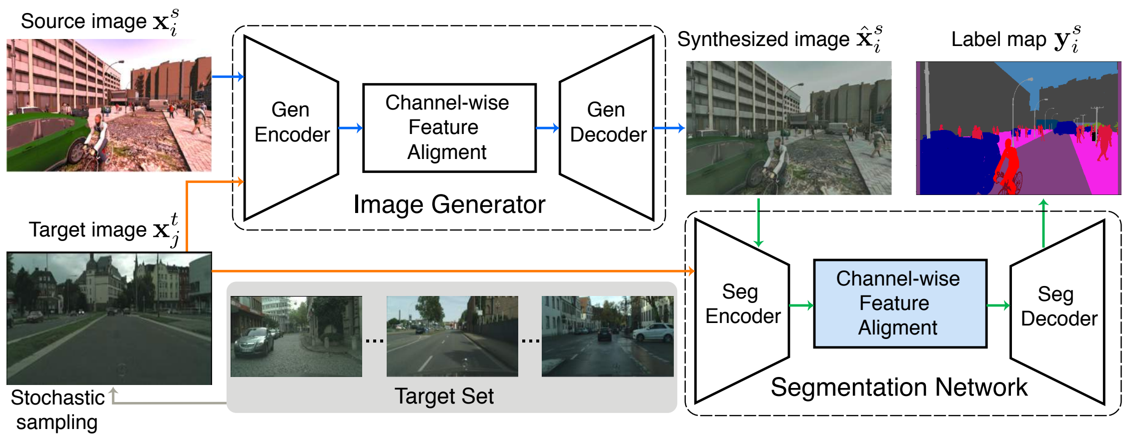

Source image $\mathbf{x}_i^s$
Gen Encoder
Channel-wise Feature Aligment
Gen Decoder
Image Generator
Synthesized image $\hat{\mathbf{x}}_i^s$
Label map $\mathbf{y}_i^s$
Target image $\mathbf{x}_j^t$
Stochastic sampling
Target Set
Seg Encoder
Channel-wise Feature Aligment
Seg Decoder
Segmentation Network

Figure 6.1: **An overview of the proposed DCANs.** It contains an image generator and a segmentation network, in both of which channel-wise alignment is performed. The generator synthesizes a new image, reducing low-level appearance differences, which is further input to the semantic segmentation network. Features directly used for segmentation are refined before producing prediction maps. During testing, we turn off the alignment (shaped in blue) and the segmentation network can be readily applied.

and simultaneously normalizing feature maps of source images, upon which segmentation classifiers directly rely. The resulting segmentation model can then be readily applied to the novel target domain. To this end, we consider each image from the target domain as a unique reference sample, whose feature representations are used to normalize those of images from the source domain. In addition, given an image from the source domain, instead of considering every single target image, we sample from the target set for alignment stochastically, serving as regularization to improve generalization. Figure 6.2 gives an overview of this framework.

More formally, let $\mathbf{X}^s = \{\mathbf{x}_i^s, \mathbf{y}_i^s\}_{i \in [N^s]}$ denote the source domain with N^s images $\mathbf{x}_i^s \in \mathbb{R}^{3 \times H \times W}$ and the corresponding label maps $\mathbf{y}_i^s \in \{0,1\}^{C \times H \times W}$, where H and W represent the height and width of the image, respectively and C denotes the number of classes. The target domain, on the other hand, has N^t images $\mathbf{X}^t =$

$\{\mathbf{x}_j^t\}_{j\in[N^t]}$ of the same resolution without labels. For each image $\mathbf{x}_i^s$ in the source domain, we randomly select one sample $\mathbf{x}_j^t$ from the target domain (we use one image here for the ease of description, but it can be a set of images as will be shown in experiments). A synthesized image $\hat{\mathbf{x}}_i^s$ is generated with the content of $\mathbf{x}_i^s$ and style of $\mathbf{x}_j^t$ by channel-wise alignment of feature statistics. This image is then fed into a segmentation network, where domain shift in high-level feature maps is further minimized for segmentation.

In the following, we first revisit channel-wise alignment (Sec 6.3.1), and then we present DCAN (Sec 6.3.2), which contains an image generator, synthesizing new images to minimize low-level differences like color and texture, and a segmentation network, refining high-level feature maps that are critical in the final segmentation task. Finally, we introduce the learning strategy (Sec. 6.3.3).

6.3.1 Channel-wise Feature Alignment

The mean and standard deviation of each channel in CNN feature maps have been shown to capture the style information of an image [145, 146, 147], and hence channel-wise alignment of feature maps is adopted for fast style transfer with a simple instance normalization step. Here, due to its effectiveness and simplicity, we use adaptive instance normalization [147], to match the mean and standard deviation of images from two different domains. In particular, given feature maps F_i^s and F_j^t of the same size $\mathbb{R}^{\hat{C}\times\hat{H}\times\hat{W}}$ ($\hat{C}$, $\hat{H}$, $\hat{W}$ represents the channel, height and width respectively) from the source and target domain, adaptive instance normalization h

produces a new representation of the source image as:

$$\begin{aligned}\hat{F_i^s} =& h(F_i^s, F_j^t) = \sigma(F_j^t)\left(\frac{F_i^s - \mu(F_i^s)}{\sigma(F_i^s)}\right) + \mu(F_j^t), \\ \mu_c(F) =& \frac{1}{\hat{H}\hat{W}}\sum_{h=1}^{\hat{H}}\sum_{w=1}^{\hat{W}} F_{chw},\ \sigma_c^2(F) = \frac{1}{\hat{H}\hat{W}}\sum_{h=1}^{\hat{H}}\sum_{w=1}^{\hat{W}}(F_{chw} - \mu_c(F))^2,\end{aligned} \tag{6.1}$$

where μ_c and σ_c denotes mean and variance across spatial dimensions for the c-th channel. This simple operation normalizes features of a source image to have similar statistics with those of a target image for each channel, which is appealing for segmentation tasks, since it is spatially invariant, *i.e.*, relative locations of pixels are fixed. In addition, such channel-wise alignment ensures semantic information like attributes encoded in different channels [144] is processed independently. In our work, we adopt channel-wise feature alignment in both our image generator for synthesizing photo-realistic samples, and segmentation network to refine features used for segmentation. Note that channel-wise feature alignment is generic and can be plugged into different layers of networks.

6.3.2 Dual Channel-wise Alignment Networks

Image generator. Our image generator contains an encoder and a decoder with channel-wise alignment in between. More specifically, the encoder, denoted as f_{gen}, is truncated from a pre-trained VGG19 network [57] by taking layers up till `relu4`. We fix the weights of the encoder, following [147, 176], to map images $\mathbf{x}_i^s$ and $\mathbf{x}_j^t$ into fixed representations: $F_i^s = f_{gen}(\mathbf{x}_i^s)$ and $F_j^t = f_{gen}(\mathbf{x}_j^t)$, respectively. F_i^s is further normalized to produce a new representation $\hat{F_i^s}$ according to Eqn. (6.1). Given the aligned source representation, a decoder, represented by g_{gen}, is applied to synthesize a new image $\hat{\mathbf{x}}_i^s = g_{gen}(\hat{F_i^s})$, in the style of samples from the target set. This is achieved by minimizing the following image generation loss function:

$$\ell_{gen} = \|f_{gen}(\hat{\mathbf{x}}_i^s) - \hat{F_i^s}\|_2 + \sum_{l=1}^{4} ||G(f_{gen}^l(\hat{\mathbf{x}}_i^s)) - G(f_{gen}^l(\mathbf{x}_j^t))||_2. \tag{6.2}$$

Here, the first term is the content loss measuring the discrepancies between features from the stylized image $\hat{\mathbf{x}}_i^s$ and the aligned features of the source image (weights of f_{gen} are fixed), forcing the synthesized image to contain the same contents as the original one. The second term matches the style information, by penalizing the differences of Gram matrices between $\hat{\mathbf{x}}_i^s$ and the target image $\mathbf{x}_j^t$ using features from the first four layers (with l denoting the layer index) in the encoder [171]. More specifically, given a reshaped feature map F with its original channel, height and width being $\hat{C}$, $\hat{H}$, $\hat{W}$ respectively, the gram matrix can be computed as:

$$G(F) = \sum_{k=1}^{\hat{H}\hat{W}} F_{ik}F_{jk} \in \mathbb{R}^{\hat{C}\times\hat{C}},\ F \in \mathbb{R}^{\hat{C}\times\hat{H}\hat{W}}.$$

Segmentation network. A new image $\hat{\mathbf{x}}_i^s$ synthesized with our generator, resembling target samples with similar low-level details like color, texture, lighting, *etc.*, is ready for semantic segmentation. Instead of sending $\hat{\mathbf{x}}_i^s$ to any off-the-shelf segmentation engine for the task, we leverage the target style image $\mathbf{x}_j^t$ once more to calibrate features of $\hat{\mathbf{x}}_i^s$ with channel-wise alignment, such that they possess similar statistics and its spatial information is preserved for segmentation. Here, the intuition is to remove undesired mismatches in higher-level feature maps that might still exist after minimizing low-level differences in the first stage. Therefore, DCAN explicitly performs another round of alignment in the segmentation network, refining features tailored for pixel-level segmentation. To this end, we divide a fully convolutional network (FCN) based model into an encoder f_{seg} and a decoder g_{seg}, with alignment in between. In particular, the segmentation decoder produces a prediction map: $\mathbf{p}_i^s = g_{seg}(h(f_{seg}(\hat{\mathbf{x}}_i^s), f_{seg}(\mathbf{x}_j^t)))$ and the segmentation loss ℓ_{seg} takes the form:

$$\ell_{seg} = -\sum_{m=1}^{H\times W}\sum_{c=1}^{C} \mathbf{y}_i^{mc}\texttt{log}(\mathbf{p}_i^{mc}), \tag{6.3}$$

which is essentially a multi-class cross-entropy loss summed over all pixels (superscript s denoting the source domain is omitted here). Note that state-of-the-art segmentation networks like DeepLab [177], FCN [178], PSPNet [179], GCN[180], *etc.*, are usually built upon top-performing models on ImageNet like VGG [63] or ResNet [16]; these networks differ in depth but have similar configurations, *i.e.*, five groups of convolution. In this case, we utilize the first three convolution groups from a segmentation model as our encoder and the remaining part as the decoder. For encoder-decoder based segmentation networks like SegNet [181], the simple idea could be directly applied.

In summary, DCAN works in the following way: given a source image, a target image is randomly selected whose style information is used for dual channel-wise alignment in both image synthesis and segmentation phases. The image generator first synthesizes a new image on-the-fly to appear similar as samples from the target domain, reducing low-level domain discrepancies in pixel space (*e.g.*, color, texture, lighting conditions, *etc.*), which is further input into the segmentation network. In the segmentation model, features from the synthesized image are further normalized specific to the sampled target image while preserving spatial structures and semantic information before producing label maps.

At test time, a novel image from the target domain is input into the segmentation network (segmentation encoder and then decoder) to predict its semantic map. The channel-wise feature alignment in the segmentation network is turned off since the network is already trained to match the feature statistics between two domains and thus can be directly applied for testing as shown in Figure 6.2.

6.3.3 Optimization

One could train the framework by selecting each sample in the source domain and normalizing it with the style information of each image in the target domain, which leads to N^t copies of the original image; the new dataset $\hat{\mathbf{X}}^s$ with the size of N^sN^t can then be used for training by minimizing:

$$\mathcal{L} = \frac{1}{N^s}\sum_{i=1}^{N^s}\frac{1}{N^t}\sum_{j=1}^{N^t}(\ell_{seg}(\mathbf{x}_i^s, \mathbf{x}_j^t, \mathbf{y}_i^s; \Theta_{seg}) + \lambda\ell_{gen}(\mathbf{x}_i^s, \mathbf{x}_j^t; \Theta_{gen})), \tag{6.4}$$

where Θ_{seg} and Θ_{gen} denote the parameters for the segmentation network and the image generator, respectively, and λ balances the two losses. However, enumerating all targets would be computationally expensive, as the cost grows linearly with the number of images in the target domain. It is worth noting that when there are infinite target images, Eqn (6.4) can be re-written as:

$$\mathcal{L} = \frac{1}{N^s} \sum_{i=1}^{N^s} \mathbb{E}_{\mathbf{x}_j^t \sim \mathbf{X}^t} [\ell_{seg}(\mathbf{x}_i^s, \mathbf{x}_j^t, \mathbf{y}_i^s; \Theta_{seg}) + \lambda \ell_{gen}(\mathbf{x}_i^s, \mathbf{x}_j^t; \Theta_{gen})]. \tag{6.5}$$

Here, the expected mean can be computed by stochastic sampling during training. The intuition is to introduce "uncertainties" to the learning processes as opposed to summing over all target styles deterministically, making the derived model more robust to noise and to generalize better on the target domain. It is a type of regularization similar in spirit to SGD for fast convergence [182], stochastic depth [31] and dropout [29, 183, 184]. Another way to view this is randomized data augmentation to improve generalization ability [56, 57]. Unlike PixelDA [140] which generates new samples conditioned on a noise vector, we augment data using feature statistics of images randomly sampled from the target domain. It is also worth noting that the idea of sampling is in line with stochastic gradient descent, which loops over the training set by sampling batches of images, and hence can be easily implemented in current deep learning frameworks.

6.4 Experiments

In this section, we first introduce the experimental setup and implementation details. Then, extensive experimental results are presented to demonstrate the effectiveness of our method. Finally, an ablation study is conducted to evaluate the contribution of different components of DCAN.

6.4.1 Experimental Setup

Datasets and evaluation metrics. We train DCAN on two source datasets, SYNTHIA [150] and GTA5 [151] respectively, and then evaluate the models on CITYSCAPES [152]. CITYSCAPES is a real-world dataset, capturing street scenes of 50 different cities, totaling $5,000$ images with pixel-level labels. The dataset is divided into a training set with $2,975$ images, a validation set with 500 images and a testing set with $1,525$ images. SYNTHIA is a large-scale synthetic dataset automatically generated for semantic segmentation of urban scenes. As in [139, 143], we utilize SYNTHIA-RAND-CITYSCAPES, a subset that contains $9,400$ images paired with CITYSCAPES, sharing 16 common classes. We randomly select 100 images for validation and use the remaining $9,300$ images for training. GTA5 contains $24,966$ high-resolution images, automatically annotated into 19 classes. The dataset is rendered from a modern computer game, Grand Theft Auto V, with labels fully compatible with those of CITYSCAPES. We randomly pick $1,000$ images for validation and use the remaining $23,966$ images for training.

Following [139, 143], to train our model, we utilize *labeled* images from the training set of either SYNTHIA or GTA5, as well as *unlabeled* images from the training set of CITYSCAPES serving as references for distribution alignment. Then we evaluate the segmentation model on the validation set of CITYSCAPES, and report mean intersection-over-union (mIoU) to measure the performance. These two adaptation settings are denoted as SYNTHIA $\rightarrow$ CITYSCAPES and GTA5 $\rightarrow$ CITYSCAPES, respectively.

Network architectures. For the image generator, its encoder is based on a VGG19 network. The architecture of the decoder is illustrated below. It takes an aligned feature representation to synthesize a new image. The architecture gradually reduces the number of channels while increasing the resolution of feature maps layer by layer, *i.e.*, the reverse process of the encoder. Therefore, the number of convolutional layers used here is similar to that in the encoder (truncated from VGG19 till the `relu4` layer).

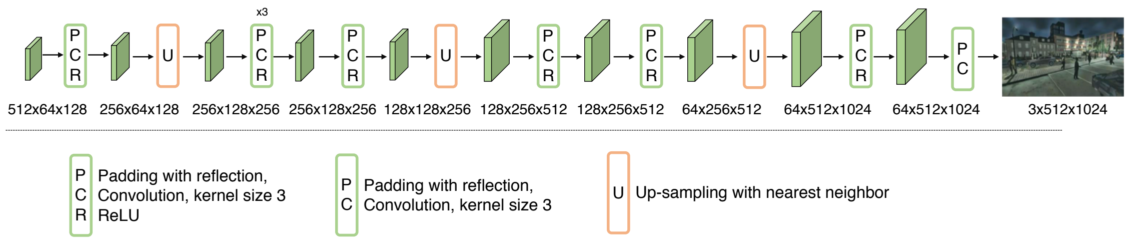

x3
P C R
U
P C
512x64x128
256x64x128
256x128x256
256x128x256
128x128x256
128x256x512
128x256x512
64x256x512
64x512x1024
64x512x1024
3x512x1024
Padding with reflection, Convolution, kernel size 3 ReLU
Padding with reflection, Convolution, kernel size 3
Up-sampling with nearest neighbor

Figure 6.2: **Architecture of the decoder in the image generator.**

To verify the effectiveness of DCAN in state-of-the-art segmentation networks, we experiment with three top-performing architectures, FCN-8s-VGG16 [178], FCN-8s-ResNet101, and PSPNet [179]. In particular, FCN8s-VGG16 and FCN8s-ResNet101 respectively adapt a pre-trained VGG16 and a ResNet101 network into fully convolutional networks and use skip connections for detailed segmentations. PSPNet is built upon a ResNet50 model with a novel pyramid pooling module to obtain representations of multiple sub-regions for per-pixel prediction [179]. These networks are pre-trained on ImageNet.

Implementation details. We adopt PyTorch for implementation and use SGD as the optimizer with a momentum of 0.99. The learning rate is fixed to $1e-3$ for both FCN8s-ResNet101 and PSPNet, and $1e-5$ for FCN8s-VGG16. We adopt a batch size of three and optimize for $100,000$ iterations, and we fix λ to 0.1. Given each sample in the training set, we randomly sample 2 images and 1 image from the target image set for experiments on SYNTHIA and GTA5 respectively. This is to achieve efficient training on GTA5, for its size is three times larger than SYNTHIA, and we will analyze the effect of the number of sampled images below. We use a crop of 512×1024 during training, and for evaluation we upsample the prediction map by a factor of 2 and then evaluate mIoU.

6.4.2 Main Results

We compare DCAN to state-of-the-art methods on unsupervised domain adaptation for semantic segmentation, including "FCN in the wild" [139] and "Curriculum Adaptation" [143]. In particular, FCN in the wild uses an adversarial loss to align fully connected layers (adapted to convolution layers) of a VGG16 model, and additionally leverages multiple instance learning to transfer spatial layout [139]. Curriculum Adaptation infers properties of the target domain using label distributions of images and superpixels [143]. The results of SYNTHIA $\rightarrow$ CITYSCAPES and GTA5 $\rightarrow$ CITYSCAPES are summarized in Table 6.1.

SYNTHIA → CITYSCAPES

Method	network	road	sidewalk	building	wall	fence	pole	traffic light	traffic sign	vegetation	sky	person	rider	car	bus	motorbike	bike	mIOU	mIOU gain
Source [139]	A/d	6.40	17.7	29.7	1.20	0.00	15.1	0.00	7.20	30.3	66.8	51.1	1.50	47.3	3.90	0.10	0.00	17.4	
[139]	A/d	11.5	19.6	30.8	4.40	0.00	20.3	0.10	11.7	42.3	68.7	51.2	3.80	54.0	3.20	0.20	0.60	20.2	2.80
Source [143]	A	5.60	11.2	59.6	8.00	0.50	21.5	8.00	5.30	72.4	75.6	35.1	9.00	23.6	4.50	0.50	18.0	22.0	
[143]	A	65.2	26.1	74.9	0.10	0.50	10.7	3.50	3.00	76.1	70.6	47.1	8.20	43.2	20.7	0.70	13.1	29.0	7.00
Source	A	10.8	11.4	66.6	1.60	0.10	16.9	5.50	14.1	74.2	76.2	46.0	11.5	45.4	15.1	6.00	13.4	25.9	-
DCAN	A	79.9	30.4	70.8	1.60	0.60	22.3	6.70	23.0	76.9	73.9	41.9	16.7	61.7	11.5	10.3	38.6	**35.4**	**9.5**
Source	B	57.9	17.0	72.7	0.20	0.00	10.4	0.00	0.00	73.5	75.4	37.8	9.30	59.3	21.7	0.40	12.3	28.0	
DCAN	B	81.5	33.4	72.4	7.90	0.20	20.0	8.60	10.5	71.0	68.7	51.5	18.7	75.3	22.7	12.8	28.1	**36.5**	**8.5**
Source	C	56.0	24.6	76.5	5.00	0.20	19.0	5.70	7.80	77.5	78.9	44.7	7.70	35.3	7.90	1.50	24.0	29.5	
DCAN	C	82.8	36.4	75.7	5.08	0.06	25.8	8.04	18.7	74.7	76.9	51.1	15.9	77.7	24.8	4.11	37.3	**38.4**	**8.9**
Oracle	A	96.4	70.3	85.9	44.4	35.8	31.5	41.5	54.2	87.5	88.9	64.1	40.8	88.5	66.1	35.5	60.3	62.0	-
	B	97.3	76.7	88.1	44.4	46.9	35.3	44.5	55.9	88.6	91.2	67.7	41.6	89.9	73.3	44.7	63.1	65.6	-
	C	97.8	78.6	89.6	56.7	57.8	39.9	61.3	65.2	89.9	91.5	73.4	56.0	89.9	84.1	54.2	69.5	72.2	-

GTA5 → CITYSCAPES

Method	network	road	sidewalk	building	wall	fence	pole	traffic light	traffic sign	vegetation	terrain	sky	person	rider	car	truck	bus	train	motorbike	bike	mIOU	mIOU gain
Source [139]	A/d	31.9	18.9	47.7	7.40	3.10	16.0	10.4	1.00	76.5	13.0	58.9	36.0	1.00	67.1	9.50	3.70	0.00	0.00	0.00	21.2	
[139]	A/d	70.4	32.4	62.1	14.9	5.40	10.9	14.2	2.70	79.2	21.3	64.6	44.1	4.20	70.4	8.00	7.30	0.00	3.50	0.00	27.1	5.90
Source [143]	A	18.1	6.80	64.1	7.30	8.70	21.0	14.9	16.8	45.9	2.40	64.4	41.6	17.5	55.3	8.40	5.0	6.90	4.30	13.8	22.3	
[143]	A	74.9	22.0	71.7	6.00	11.9	8.40	16.3	11.1	75.7	13.3	66.5	38.0	9.30	55.2	18.8	18.9	0.00	16.8	16.6	28.9	6.6
Source	A	72.5	25.1	71.2	6.60	13.4	12.3	11.0	4.70	76.1	16.4	67.7	43.1	8.00	70.4	11.3	4.80	0.00	13.9	0.40	27.8	
DCAN	A	82.3	26.7	77.4	23.7	20.5	20.4	30.3	15.9	80.9	25.4	69.5	52.6	11.1	79.6	24.9	21.2	1.30	17.0	6.70	**36.2**	**8.4**
Source	B	44.5	12.7	71.1	9.40	17.7	15.3	24.3	11.9	80.5	14.3	80.0	50.3	7.70	45.4	30.5	30.8	5.50	9.80	3.50	29.8	
DCAN	B	88.5	37.4	79.3	24.8	16.5	21.3	26.3	17.4	80.8	30.9	77.6	50.2	19.2	77.7	21.6	27.1	2.70	14.3	18.1	**38.5**	**8.7**
Source	C	69.9	22.3	75.6	15.8	20.1	18.8	28.2	17.1	75.6	8.00	73.5	55.0	2.90	66.9	34.4	30.8	0.00	18.4	0.00	33.3	
DCAN	C	85.0	30.8	81.3	25.8	21.2	22.2	25.4	26.6	83.4	36.7	76.2	58.9	24.9	80.7	29.5	42.9	2.50	26.9	11.6	**41.7**	**8.4**
Oracle	A	96.4	70.3	85.9	44.4	35.8	31.5	41.5	54.2	87.5	51.9	88.9	64.1	40.8	88.5	55.8	66.1	44.9	35.5	60.3	60.2	-
	B	97.3	76.7	88.1	44.4	46.9	35.3	44.5	55.9	88.6	55.9	91.2	67.7	41.6	89.9	60.1	73.3	54.4	44.7	63.1	64.2	-
	C	97.8	78.6	89.6	56.7	57.8	39.9	61.3	65.2	89.9	58.9	91.5	73.4	56.0	89.9	75.8	84.1	78.8	54.2	69.5	72.0	-

Table 6.1: **Results and comparisons on Cityscapes when adapted from Synthia and Gta5, respectively.** Here, "Source" denotes source only methods, "Oracle" denotes results from supervised training, and A, B, C represent FCN8s-VGG16, FCN8s-ResNet101 and PSPNet. A/d uses dilation in VGG16 for segmentation.

We observe that these domain adaptation methods, although different in design, can indeed lead to improvements over the source only method (denoted as source), which simply trains a model on the source domain and then directly applies it to the target domain. In particular, DCAN outperforms its corresponding source only baseline with clear margins, around 8 and 9 absolute percentage points, using all three different networks on both datasets. This confirms the effectiveness of DCAN, which not only reduces domain differences for improved performance but also is general for multiple network architectures. Furthermore, with PSPNet, DCAN achieves 41.7% and 38.4% on CITYSCAPES when adapted from GTA5 and SYNTHIA, respectively. Compared to [139, 143], with the same backbone VGG16 architecture, DCAN offers the best mIoU value as well the largest relative mIoU gain (9.5% and 8.4% trained from SYNTHIA and GTA5 respectively). Note that although the backbone network is the same, source only baselines are different due to different experimental settings. A dilated VGG16 network is adopted in [139] and the network is additionally pre-trained on PASCAL-CONTEXT in [143]. In addition, it uses a crop size of 320×640 during training. Our model is initialized on ImageNet and we choose 512×1024 for training since large resolution offers better performance as observed in [179], which is also consistent with state-of-the-art supervised methods on CITYSCAPES [152]. It is worth noting that DCAN improves a stronger baseline by 36% relatively (25.9% to 35.4%). With the same image size as in [143], DCAN improves the source only baseline from 23.6% to 33.0% (*v.s.*, 22.0% to 29.0% in [143]; see Table 6.2).

Among three different networks, PSPNet gives the best results on both datasets, mainly resulting from the pyramid pooling module that considers difference scales. Figure 6.3 illustrates sampled results of PSPNet under the GTA5 $\rightarrow$ CITYSCAPES setting, and its comparison with the source only method. Comparing across datasets, models trained on GTA5 produce better accuracies than those learned from SYN-

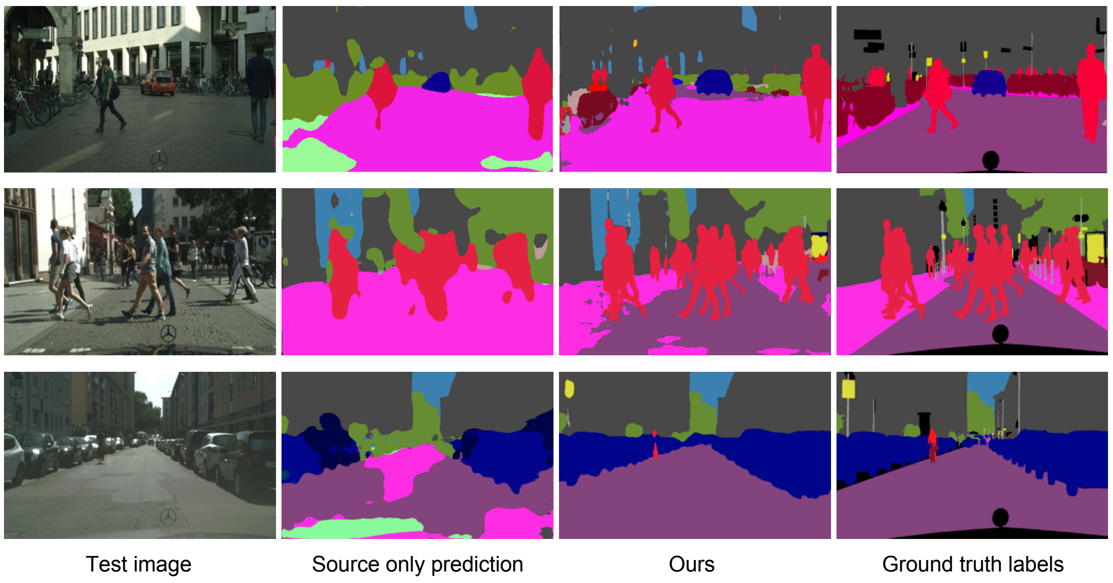


Figure 6.3: **Sampled prediction results** of PSPNet and its corresponding source only model under the GTA5 $\rightarrow$ CITYSCAPES setting using testing images from CITYSCAPES. Our model effectively improves the generalization ability of the trained segmentation network.

THIA. The reasons are two-folds: (1) a large number of images from SYNTHIA are rendered at night, incurring significant domain differences since images from CITYSCAPES are captured during day time; (2) there are more training samples in GTA5. In addition, oracle results, which are produced with traditional supervised training using annotations from the target domain, are also listed for reference. We can see there is still significant performance gaps between domain adaptation methods and oracle supervised training, highlighting the challenging nature of this problem.

SYNTHIA → CITYSCAPES			
Resolution	Method	mIoU	gain
256×512	Source	21.2	
	DCAN	29.6	8.4
320×640	Source	23.6	
	DCAN	33.0	9.4
512×1024	Source	25.9	
	DCAN	35.4	9.5

Table 6.2: **Results of FCN8s-VGG16 using three different image resolutions.**

SYNTHIA → CITYSCAPES	
Method	mIoU
CycleGAN [3]	30.4
CycleGAN w. FeatureAlignment	31.7
UNIT [176]	31.6
UNIT w. FeatureAlignment	32.7
DCAN w/o FeatureAlignment	33.8
DCAN (two stage)	33.7
DCAN (end-to-end)	35.4

Table 6.3: **Training with and without feature alignment in FCN8s-VGG16 using different image synthesis methods.**

6.4.3 Discussions

In this section, we run a number of experiments to analyze DCAN in the SYNTHIA → CITYSCAPES setting, and provide corresponding results and discussions.

Image resolution. As previously mentioned, top performing approaches on CITYSCAPES typically use a high resolution for improved performance [152]. For example, GCN and FRRN utilize a resolution of 800×800 [180] and 512×1024 [185], respectively. Here, we report the results of DCAN adapted from SYNTHIA using FCN8s-VGG16 with three different resolutions, and compare with the corresponding source only method in Table 6.2. DCAN offers significant performance gains for all resolutions, and a larger resolution is indeed better for unsupervised domain adaptation.

Different image synthesis methods. We compare with two different image synthesis methods: (1) CycleGAN [3] and (2) UNIT [4], both of which attempt to learn a distribution mapping function between two domains. Once the mapping

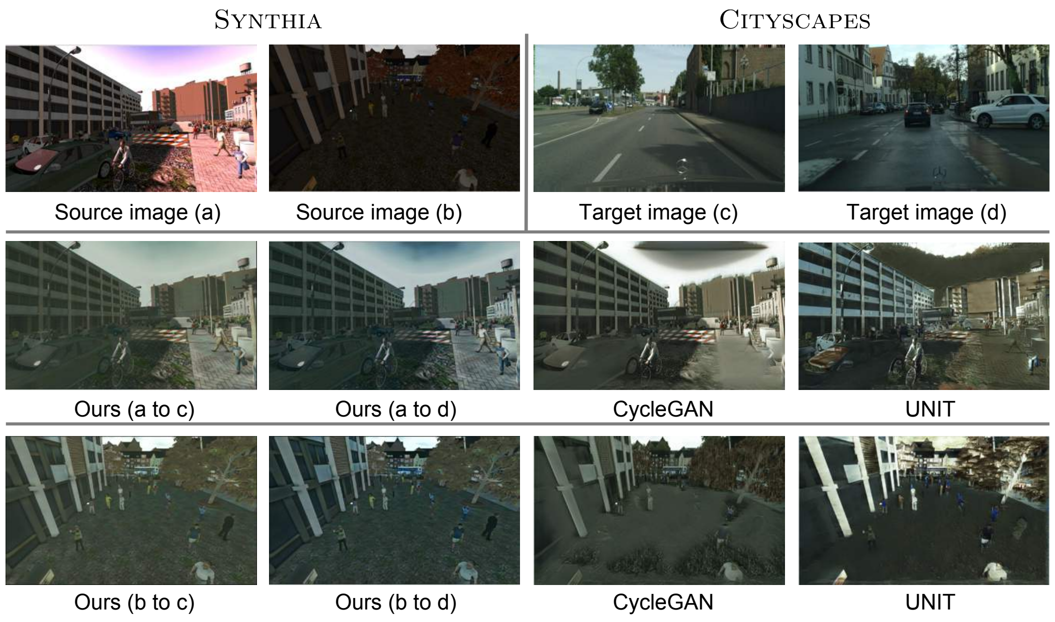


Figure 6.4: **Images from Synthia synthesized in the style of Cityscapes with CycleGAN [3], UNIT [4] and DCAN.**

function is learned, images from the source domain can be translated to the style of the target domain. Therefore, we use the translated images from the source domain to train the segmentation network. Table 6.3 presents the results. For fair comparisons, we compare them under two settings, with and without the channel-wise feature alignment in the segmentation network. DCAN achieves better results than both GAN-based image synthesis methods in both scenarios. To justify the advantage of an end-to-end framework, we also compare with a two-stage training strategy, which simply trains a segmentation network using pre-synthesized images without end-to-end training. In this case, image synthesis is not optimized using gradients from the segmentation network. DCAN improves the two-stage training by 1.7% mIOU, demonstrating the importance of guiding the synthesis process with useful information from the final task.

Figure 6.4 further compares images produced by different synthesis methods. DCAN is able to generate images that conform to the style of images from the target set, containing fewer artifacts than CycleGAN and UNIT. In addition, both CycleGAN and UNIT seek to align distributions at a dataset level, and once the mapping is learned, the translation from the source to the target is fixed (a fixed output given an input image). Learning such a transformation function on high resolution images is a non-trivial task and might not perfectly cover all possible variations. Instead, DCAN performs image translation at an instance level, and in the regime of stochastic sampling, it is able to cover sufficient styles from the target set for distribution alignment. It is also worth noting that feature alignment can improve segmentation results regardless of synthesis methods. We also experimented with other GAN-based approaches like PixelDA [140] for image synthesis; however, conditioning on a noise vector rather than label maps [186] fails to produce photo-realistic images in high resolution.

Channel-wise feature alignment for segmentation. We now analyze the effect of channel-wise alignment in the segmentation network (Table 6.4) with FCN8s-VGG16. We compare with Adversarial Discriminative Domain Adaptation [138], which leverages an adversarial loss to make features from two domains indistinguishable without considering spatial structures explicitly. DCAN outperforms ADDA by 1.4%, and also converges faster during training. We also implemented MMD [137] and CORAL [136] loss to align features, but their results are worse than source only methods. This is consistent with observations in [143]. We further investigate where to align in the segmentation network and found that alignment after the `Conv3` layer gives the best results, possibly due to it contains both sufficient number of channels and relatively large feature maps. In addition, aligning features maps with more detailed spatial information (`Conv2` and `Conv4`) is also better than `Conv6` and `Conv7` (convolution layers adapted from fully connected layers, whose feature maps are smaller). This confirms the importance to consider detailed spatial information explicitly for alignment.

SYNTHIA → CITYSCAPES	
Alignment Method	mIoU
ADDA [138]	34.0
Ours-w/o alignment	33.8
Ours–`Conv2`	34.0
Ours–`Conv4`	34.4
Ours–`Conv6`	33.2
Ours–`Conv7`	32.7
Ours–`Conv3`	35.4

Table 6.4: **Comparisons of different feature alignment methods in the segmentation network.**

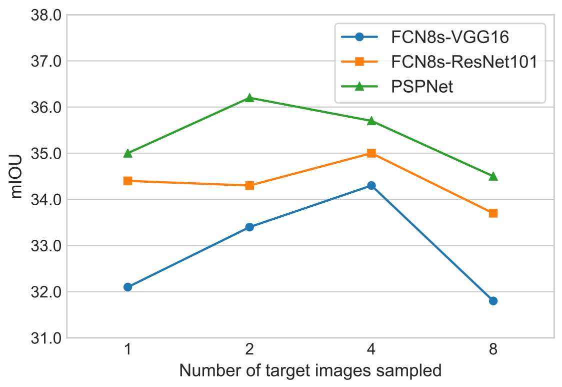


Figure 6.5: **Effect of using different number of target images for each training sample.**

Number of target images sampled. We also evaluate how the number of sampled target images affects the performance. Since enumerating around $3,000$ samples for each image in the training set is computationally prohibitive, we create a pseudo-target set with 8 images randomly selected from 8 cities in CITYSCAPES. This is to ensure there are variations among the targets and it is computationally feasible for enumerating all targets. We then analyze the effect of the number of target images used during training by randomly selecting 1, 2, 4 samples from SYNTHIA. Figure 6.5 presents the results. We observe that stochastically selecting from the target set is better than using all of them for all three networks. This might result from two reasons: (1) translating one image to multiple different representations in one-shot is hard to optimize; (2) stochastic sampling acts as regularization to improve generalization, which is similar to the case that stochastic gradient is better than full batch gradient descent. Interestingly, for PSPNet and FCN8s-ResNet101, sampling one image achieves competitive results, and this is very appealing when the number of samples in the target domain is limited.

6.5 Conclusion

In this chapter, we have presented, DCAN, a simple yet effective approach to reduce domain shift at both pixel-level and feature-level for unsupervised scene adaptation. In particular, our framework leverages channel-wise feature alignment in both the image generator for synthesizing photo-realistic samples, appearing as if drawn from the target set, and the segmentation network, which simultaneously normalizes feature maps of source images. In contrast to recent work that makes extensive use of adversarial training, our framework is lightweight and easy to train. We conducted extensive experiments by transferring models learned on synthetic segmentation datasets to real urban scenes, and demonstrated the effectiveness of DCAN over state-of-the-art methods and its compatibility with modern segmentation networks.

Chapter 7: Conclusions and Future Directions

We presented approaches for image and video understanding with constrained computational resources and annotation resources, given that very deep networks are resource intensive for real-world deployment and they require substantial amount of annotations to train.

To achieve accurate recognition when computational budget is limited, we draw inspiration from human perception systems, where glances of scenes and easy objects are sufficient to get an overview and more attention will be paid to objects with occlusion. We developed BlockDrop, AdaFrame, LiteEval, all of which are conditional computation frameworks that dynamically allocate computational resources conditioned on inputs. We believe it is therefore important to incorporate dynamic computation as a modularized component in modern deep models such that computing resources can be adaptively allocated on a per-sample basis. With the dynamic computation component, the trade-off between accuracy and computational cost can be balanced, offering flexibilities in state-of-the-art frameworks conditioned on computing resources. It is a desirable property for a variety of tasks, including image classification, object detection, semantic segmentation, *etc.*, when deploying deep models to real-world applications. In addition, dynamic computation selects different paths (modules) conditioned on inputs for efficient inference and hence is complementary to model compression techniques that prune parameters.

Furthermore, videos capture appearance of objects (and people) and how those evolve over time, and are naturally multimodal. In addition to adaptively selecting relevant frames for fast recognition, it would be interesting to study conditional computation methods for multiple modalities. For instance, if RGB information in static frames is not sufficient, one can also dynamically extract information from the audio modality on-the-fly based on input videos as well as computing resources.

In addition, to mitigate the need of collecting manual annotations, we demonstrated context information in images and shared information across domains are powerful for deriving robust feature representations. A promising future direction is to make use of class hierarchy as a knowledge base, whether defined or learned, to enrich information when learning representations. For example, both "Husky" and "Chihuahua" share not only commonalities in the "dog" class, but also generic information with other types animals like the "cat" class. Class hierarchy is essentially a graph, in which information can be propagated among related classes as a way to reduce the number of samples needed for training, and can possibly scale up current approaches to tens of thousands of categories in real-world. The idea of using a graph to measure proximity can be further extended to unsupervised domain adaptation. One can construct a graph estimating the distances of synthetic and realistic samples at the image-level or super-pixel level, and then the graph could be used to guide the adaptation process by dynamically selecting the near regions or images of a target to transfer.

In addition, we, as humans, are able to gain knowledge and perform new tasks by watching videos. It would be intriguing for computer vision systems to learn from the massive amounts of online videos that contain abundant multimodal information, with an aim to derive robust and generic feature representations for other tasks, and ultimately to acquire spatial-temporal structures and commonsense knowledge.

Bibliography

[1] Serena Yeung, Olga Russakovsky, Greg Mori, and Li Fei-Fei. End-to-end learning of action detection from frame glimpses in videos. In *CVPR*, 2016.

[2] Hehe Fan, Zhongwen Xu, Linchao Zhu, Chenggang Yan, Jianjun Ge, and Yi Yang. Watching a small portion could be as good as watching all: Towards efficient video classification. In *IJCAI*, 2018. doi: 10.24963/ijcai.2018/98.

[3] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *ICCV*, 2017.

[4] Ming-Yu Liu, Thomas Breuel, and Jan Kautz. Unsupervised image-to-image translation networks. In *NIPS*, 2017.

[5] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. In *arXiv preprint arXiv:1503.02531*, 2015.

[6] G. Chen, M. Chandraker, T. Han, W. Choi, and X. Yu. Learning efficient object detection models with knowledge distillation. In *NIPS*, 2017.

[7] Yani Ioannou, Duncan Robertson, Jamie Shotton, Roberto Cipolla, and Antonio Criminisi. Training cnns with low-rank filters for efficient image classification. In *ICLR*, 2016.

[8] Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *ICASSP*, 2013.

[9] Adam Polyak and Lior Wolf. Channel-level acceleration of deep face representations. *IEEE Access*, 3:2163–2175, 2015.

[10] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *ICLR*, 2017.

[11] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *ICLR*, 2016.

[12] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *CVPR*, 2016.

[13] Hao Li, Soham De, Zheng Xu, Christoph Studer, Hanan Samet, and Tom Goldstein. Training quantized nets: A deeper understanding. In *NIPS*, 2017.

[14] Ales Ude. Integrating visual perception and manipulation for autonomous learning of object representations. *Can developmental robotics yield human-like cognitive abilities?*, 2012.

[15] Dirk B Walther, Barry Chai, Eamon Caddigan, Diane M Beck, and Li Fei-Fei. Simple line drawings suffice for functional mri decoding of natural scene categories. *PNAS*, 2011.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[17] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *NIPS*, 2016.

[18] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *ICCV*, 2017.

[19] Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *NIPS*, 2016.

[20] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* MIT press Cambridge, 1998.

[21] John Langford and Tong Zhang. The epoch-greedy algorithm for multi-armed bandits with side information. In *NIPS*, 2008.

[22] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[23] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[24] Michael Figurnov, Maxwell D Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry Vetrov, and Ruslan Salakhutdinov. Spatially adaptive computation time for residual networks. In *CVPR*, 2017.

[25] Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

[26] Xuanyi Dong, Junshi Huang, Yi Yang, and Shuicheng Yan. More is less: A more complicated network with less inference complexity. In *CVPR*, 2017.

[27] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, 2017.

[28] Masoud Abdi and Saeid Nahavandi. Multi-residual networks: Improving the speed and accuracy of residual networks. *arXiv preprint arXiv:1609.05672*, 2016.

[29] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 2014.

[30] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *ICML*, 2013.

[31] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *ECCV*, 2016.

[32] Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. *arXiv preprint arXiv:1707.01213*, 2017.

[33] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. In *arXiv preprint arXiv:1412.6550*, 2014.

[34] Cheng Tai, Tong Xiao, Xiaogang Wang, et al. Convolutional neural networks with low-rank regularization. In *ICLR*, 2016.

[35] Y. LeCun, J. Denker, and S. Solla. Optimal brain damage. In *NIPS*, 1989.

[36] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *CVPR*, 2018.

[37] Y. Cheng, F. Yu, R. Feris, S. Kumar, A. Choudhary, and S. F. Chang. An exploration of parameter redundancy in deep networks with circulant projections. In *ICCV*, 2015.

[38] Vikas Sindhwani, Tara Sainath, and Sanjiv Kumar. Structured transforms for small-footprint deep learning. In *NIPS*, 2015.

[39] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016.

[40] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[41] Zhe Li, Xiaoyu Wang, Xutao Lv, and Tianbao Yang. Sep-nets: Small and effective pattern networks. *arXiv preprint arXiv:1706.03912*, 2017.

[42] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *ICML*, 2015.

[43] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size. In *arXiv:1602.07360*, 2016.

[44] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. In *CVPR*, 2017.

[45] Yoshua Bengio. Deep learning of representations: Looking forward. In *SLCP*, 2013.

[46] Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. In *ICML Workshop on Abstraction in Reinforcement Learning*, 2016.

[47] Lanlan Liu and Jia Deng. Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. *arXiv preprint arXiv:1701.00299*, 2017.

[48] Ludovic Denoyer and Patrick Gallinari. Deep sequential neural network. *arXiv preprint arXiv:1410.0510*, 2014.

[49] Sergey Karayev, Mario Fritz, and Trevor Darrell. Anytime recognition of objects and scenes. In *CVPR*, 2014.

[50] Yu-Chuan Su and Kristen Grauman. Leaving some stones unturned: dynamic feature prioritization for activity detection in streaming video. In *ECCV*, 2016.

[51] P. Felzenszwalb, R. Girshick, and D. McAllester. Cascade object detection with deformable part models. In *CVPR*, 2010.

[52] Paul Viola and Michael J Jones. Robust real-time face detection. *IJCV*, 2004.

[53] Surat Teerapittayanon, Bradley McDanel, and HT Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *ICPR*, 2016.

[54] Mason McGill and Pietro Perona. Deciding how to decide: Dynamic routing in artificial neural networks. In *ICML*, 2017.

[55] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q Weinberger. Multi-scale dense convolutional networks for efficient prediction. In *ICLR*, 2018.

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

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

[58] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2013.

[59] Steven J Rennie, Etienne Marcheret, Youssef Mroueh, Jarret Ross, and Vaibhava Goel. Self-critical sequence training for image captioning. In *CVPR*, 2017.

[60] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. In *ICLR*, 2016.

[61] Du Tran, Lubomir D Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. C3d: Generic features for video analysis. In *ICCV*, 2015.

[62] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *CVPR*, 2018.

[63] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *NIPS*, 2014.

[64] Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. Temporal segment networks: Towards good practices for deep action recognition. In *ECCV*, 2016.

[65] Chao-Yuan Wu, Manzil Zaheer, Hexiang Hu, R Manmatha, Alexander J Smola, and Philipp Krähenbühl. Compressed video action recognition. In *CVPR*, 2018.

[66] Bowen Zhang, Limin Wang, Zhe Wang, Yu Qiao, and Hanli Wang. Real-time action recognition with enhanced motion vector cnns. In *CVPR*, 2016.

[67] Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. In *CVPR*, 2015.

[68] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, 2017.

[69] Y.-G. Jiang, Z. Wu, J. Wang, X. Xue, and S.-F. Chang. Exploiting feature and class relationships in video categorization with regularized deep neural networks. *IEEE TPAMI*, 2018.

[70] Fabian Caba Heilbron, Victor Escorcia, Bernard Ghanem, and Juan Carlos Niebles. Activitynet: A large-scale video benchmark for human activity understanding. In *CVPR*, 2015.

[71] Zuxuan Wu, Ting Yao, Yanwei Fu, and Yu-Gang Jiang. Frontiers of multimedia research. chapter Deep Learning for Video Classification and Captioning. 2018.

[72] Saining Xie, Chen Sun, Jonathan Huang, Zhuowen Tu, and Kevin Murphy. Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification. In *ECCV*, 2018.

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

[74] Y.-G. Jiang, Q. Dai, T. Mei, Y. Rui, and S.-F. Chang. Super fast event recognition in internet videos. *IEEE TMM*, 2015.

[75] Xizhou Zhu, Yuwen Xiong, Jifeng Dai, Lu Yuan, and Yichen Wei. Deep feature flow for video recognition. In *CVPR*, 2017.

[76] Bowen Pan, Wuwei Lin, Xiaolin Fang, Chaoqin Huang, Bolei Zhou, and Cewu Lu. Recurrent residual module for fast inference in videos. In *CVPR*, 2018.

[77] Xin Wang, Fisher Yu, Zi-Yi Dou, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *ECCV*, 2018.

[78] Andreas Veit and Serge Belongie. Convolutional networks with adaptive inference graphs. In *ECCV*, 2018.

[79] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. In *CVPR*, 2018.

[80] Mahyar Najibi, Bharat Singh, and Larry S Davis. Autofocus: Efficient multiscale inference. In *ICCV*, 2019.

[81] Mingfei Gao, Ruichi Yu, Ang Li, Vlad I Morariu, and Larry S Davis. Dynamic zoom-in network for fast object detection in large images. In *CVPR*, 2018.

[82] Jasmine Collins, Jascha Sohl-Dickstein, and David Sussillo. Capacity and trainability in recurrent neural networks. In *ICLR*, 2017.

[83] Dani Yogatama, Yishu Miao, Gabor Melis, Wang Ling, Adhiguna Kuncoro, Chris Dyer, and Phil Blunsom. Memory architectures in recurrent neural network language models. In *ICLR*, 2018.

[84] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017.

[85] Shugao Ma, Leonid Sigal, and Stan Sclaroff. Learning activity progression in lstms for activity detection and early detection. In *CVPR*, 2016.

[86] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *NIPS*, 2014.

[87] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.

[88] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015.

[89] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *CVPR*, 2018.

[90] Bharat Singh, Mahyar Najibi, and Larry S Davis. Sniper: Efficient multi-scale training. In *NIPS*, 2018.

[91] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context. In *ECCV*, 2014.

[92] Tamir Hazan and Tommi S. Jaakkola. On the partition function and random maximum a-posteriori perturbations. In *ICML*, 2012.

[93] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. In *ICLR*, 2017.

[94] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *ICLR*, 2017.

[95] Ting Yao, Chong-Wah Ngo, and Shiai Zhu. Predicting domain adaptivity: Redo or recycle? In *ACM Multimedia*, 2012.

[96] Zuxuan Wu, Caiming Xiong, Chih-Yao Ma, Richard Socher, and Larry S Davis. Adaframe: Adaptive frame selection for fast video recognition. In *CVPR*, 2019.

[97] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. Slowfast networks for video recognition. In *ICCV*, 2019.

[98] Bruno Korbar, Du Tran, and Lorenzo Torresani. Scsampler: Sampling salient clips from video for efficient action recognition. In *ICCV*, 2019.

[99] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *ICML*, 2014.

[100] Ali Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *CVPR workshop of DeepVision*, 2014.

[101] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

[102] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 2006.

[103] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML*, 2008.

[104] Deepak Pathak, Philipp Krähenbühl, Jeff Donahue, Trevor Darrell, and Alexei A. Efros. Context encoders: Feature learning by inpainting. In *CVPR*, 2016.

[105] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013.

[106] Carl Doersch, Abhinav Gupta, and Alexei A. Efros. Unsupervised visual representation learning by context prediction. In *ICCV*, 2015.

[107] Ross Girshick. Fast r-cnn. In *ICCV*, 2015.

[108] Geoffrey E Hinton. Learning multiple layers of representation. *Trends in cognitive sciences*, 2007.

[109] Alexey Dosovitskiy, Jost Tobias Springenberg, Martin Riedmiller, and Thomas Brox. Discriminative unsupervised feature learning with convolutional neural networks. *TPAMI*, 2015.

[110] Sergey Zagoruyko and Nikos Komodakis. Learning to compare image patches via convolutional neural networks. In *CVPR*, 2015.

[111] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014.

[112] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *NIPS*, 2014.

[113] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *CoRR*, 2015.

[114] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *CVPR*, 2014.

[115] Judy Hoffman, Sergio Guadarrama, Eric Tzeng, Ronghang Hu, Jeff Donahue, Ross Girshick, Trevor Darrell, and Kate Saenko. Lsda: Large scale detection through adaptation. In *NIPS*, 2014.

[116] Yuting Zhang, Kibok Lee, and Honglak Lee. Augmenting supervised neural networks with unsupervised objectives for large-scale image classification. In *ICML*, 2016.

[117] R. Cinbis, J. Verbeek, and C. Schmid. Multi-fold mil training for weakly supervised object localization. In *CVPR*, 2014.

[118] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Is object localization for free? – weakly-supervised learning with convolutional neural networks. In *CVPR*, 2015.

[119] Z. Shi, T. M. Hospedales, and T. Xiang. Bayesian joint topic modelling for weakly supervised object localisation. In *ICCV*, 2013.

[120] H. O. Song, R. Girshick, S. Jegelka, J. Mairal, Z. Harchaoui, and T. Darrell. On learning to localize objects with minimal supervision. In *ICML*, 2014.

[121] H. O. Song, Y. J. Lee, S. Jegelka, and T. Darrell. Weakly supervised discovery of visual pattern configurations. In *NIPS*, 2014.

[122] C. Wang, W. Ren, K. Huang, and T. Tan. Weakly supervised object localization with latent category learning. In *ECCV*, 2014.

[123] Xiaolong Wang and Abhinav Gupta. Unsupervised learning of visual representations using videos. In *ICCV*, 2015.

[124] Pulkit Agrawal, Joao Carreira, and Jitendra Malik. Learning to see by moving. In *ICCV*, 2015.

[125] Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *ECCV*, 2016.

[126] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[127] Shengxin Zha, Florian Luisier, Walter Andrews, Nitish Srivastava, and Ruslan Salakhutdinov. Exploiting image-trained cnn architectures for unconstrained video classification. In *BMVC*, 2015.

[128] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. In *NIPS*, 2014.

[129] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *IJCV*, 2010.

[130] C Lawrence Zitnick and Piotr Dollár. Edge boxes: Locating object proposals from edges. In *ECCV*. 2014.

[131] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, 2016.

[132] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014.

[133] G. Varol, J. Romero, X. Martin, N. Mahmood, M. J. Black, I. Laptev, and C. Schmid. Learning from synthetic humans. In *CVPR*, 2017.

[134] Xingchao Peng, Baochen Sun, Karim Ali, and Kate Saenko. Learning deep object detectors from 3d models. In *ICCV*, 2015.

[135] Baochen Sun, Jiashi Feng, and Kate Saenko. Return of frustratingly easy domain adaptation. In *AAAI*, 2016.

[136] Baochen Sun and Kate Saenko. Deep CORAL - correlation alignment for deep domain adaptation. In *ECCV*, 2016.

[137] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael I. Jordan. Learning transferable features with deep adaptation networks. In *ICML*, 2015.

[138] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *CVPR*, 2017.

[139] Judy Hoffman, Dequan Wang, Fisher Yu, and Trevor Darrell. Fcns in the wild: Pixel-level adversarial and constraint-based adaptation. *CoRR*, 2016.

[140] Konstantinos Bousmalis, Nathan Silberman, David Dohan, Dumitru Erhan, and Dilip Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *CVPR*, 2017.

[141] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei A Efros, and Trevor Darrell. Cycada: Cycle-consistent adversarial domain adaptation. In *ICML*, 2018.

[142] Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige, et al. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. *arXiv preprint arXiv:1709.07857*, 2017.

[143] Yang Zhang, Philip David, and Boqing Gong. Curriculum domain adaptation for semantic segmentation of urban scenes. In *ICCV*, 2017.

[144] Long Chen, Hanwang Zhang, Jun Xiao, Liqiang Nie, Jian Shao, Wei Liu, and Tat-Seng Chua. Sca-cnn: Spatial and channel-wise attention in convolutional networks for image captioning. In *CVPR*, 2017.

[145] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *CoRR*, 2016.

[146] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. A learned representation for artistic style. In *ICLR*, 2016.

[147] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *ICCV*, 2017.

[148] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *CVPR*, 2012.

[149] Mehdi Sajjadi, Mehran Javanmardi, and Tolga Tasdizen. Regularization with stochastic transformations and perturbations for deep semi-supervised learning. In *NIPS*, 2016.

[150] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M. Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *CVPR*, 2016.

[151] Stephan R Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In *ECCV*, 2016.

[152] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 2016.

[153] Konstantinos Bousmalis, George Trigeorgis, Nathan Silberman, Dilip Krishnan, and Dumitru Erhan. Domain separation networks. In *NIPS*, 2016.

[154] Yaroslav Ganin and Victor S. Lempitsky. Unsupervised domain adaptation by backpropagation. In *ICML*, 2015.

[155] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *JMLR*, 2016.

[156] Eric Tzeng, Judy Hoffman, Trevor Darrell, and Kate Saenko. Simultaneous deep transfer across domains and tasks. In *ICCV*, 2015.

[157] Swami Sankaranarayanan, Yogesh Balaji, Arpit Jain, Ser Nam Lim, and Rama Chellappa. Unsupervised domain adaptation for semantic segmentation with gans. In *CVPR*, 2018.

[158] Yiheng Zhang, Zhaofan Qiu, Ting Yao, Dong Liu, and Tao Mei. Fully convolutional adaptation networks for semantic segmentation. In *CVPR*, 2018.

[159] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Joshua Susskind, Wenda Wang, and Russell Webb. Learning from simulated and unsupervised images through adversarial training. In *CVPR*, 2017.

[160] Yaniv Taigman, Adam Polyak, and Lior Wolf. Unsupervised cross-domain image generation. In *ICLR*, 2017.

[161] Ming-Yu Liu Liu and Oncel Tuzel. Coupled generative adversarial networks. In *NIPS*, 2016.

[162] Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. In *ICLR*, 2017.

[163] Donggeun Yoo, Namil Kim, Sunggyun Park, Anthony S. Paek, and In-So Kweon. Pixel-level domain transfer. In *ECCV*, 2016.

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

[165] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans. In *ICML*, 2017.

[166] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. In *ICML*, 2016.

[167] Wei Shen and Rujie Liu. Learning residual images for face attribute manipulation. In *CVPR*, 2017.

[168] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017.

[169] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *CVPR*, 2017.

[170] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *ICLR*, 2018.

[171] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *CVPR*, 2016.

[172] Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor S Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. In *ICML*, 2016.

[173] Guim Perarnau, Joost van de Weijer, Bogdan Raducanu, and Jose M Álvarez. Invertible conditional gans for image editing. In *NIPS Workshop*, 2016.

[174] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, 2016.

[175] Yanghao Li, Naiyan Wang, Jiaying Liu, and Xiaodi Hou. Demystifying neural style transfer. In *IJCAI*, 2018.

[176] Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming-Hsuan Yang. Universal style transfer via feature transforms. In *NIPS*, 2017.

[177] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE TPAMI*, 2018.

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

[179] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *CVPR*, 2017.

[180] Chao Peng, Xiangyu Zhang, Gang Yu, Guiming Luo, and Jian Sun. Large kernel matters – improve semantic segmentation by global convolutional network. In *CVPR*, 2017.

[181] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE TPAMI*, 2017.

[182] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Compstat*. 2010.

[183] Bohyung Han, Jack Sim, and Hartwig Adam. Branchout: Regularization for online ensemble tracking with convolutional neural networks. In *CVPR*, 2017.

[184] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus. Regularization of neural networks using dropconnect. In *ICML*, 2013.

[185] T. Pohlen, A. Hermans, M. Mathias, and B. Leibe. Full-resolution residual networks for semantic segmentation in street scenes. In *CVPR*, 2017.

[186] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *CVPR*, 2018.