Motion-Aware Dynamic Architecture for Efficient Frame Interpolation

Myungsub Choi* Suyoung Lee Heewon Kim Kyoung Mu Lee
ASRI, Department of ECE, Seoul National University
{cms6539, esw0116, ghimhw, kyoungmu}@snu.ac.kr

Abstract

Video frame interpolation aims to synthesize accurate intermediate frames given a low-frame-rate video. While the quality of the generated frames is increasingly getting better, state-of-the-art models have become more and more computationally expensive. However, local regions with small or no motion can be easily interpolated with simple models and do not require such heavy compute, whereas some regions may not be correct even after inference through a large model. Thus, we propose an effective framework that assigns varying amounts of computation for different regions. Our dynamic architecture first calculates the approximate motion magnitude to use as a proxy for the difficulty levels for each region, and decides the depth of the model and the scale of the input. Experimental results show that static regions pass through a smaller number of layers, while the regions with larger motion are downscaled for better motion reasoning. In doing so, we demonstrate that the proposed framework can significantly reduce the computation cost (FLOPs) while maintaining the performance, often up to 50% when interpolating a 2K resolution video.

1. Introduction

Video frame interpolation is an increasingly popular research area in computer vision, which can be used in various real-world applications such as densely sampled light field reconstruction [14], motion blur synthesis [5], video colorization [31], and video stabilization [8]. While recent advances in frame interpolation frameworks enable high-quality synthesis of the unseen intermediate frames [10, 36], the models have become extremely complex, requiring extensive computation and memory. For example, generating a single HD resolution frame with DAIN [2] requires approximately 7GB of memory and one second of running time with a modern GPU. Additionally, the sizes of images are getting larger, increasing the computational burden. Such heavy computational overhead hinders practical usage of the existing models when the resolution of the input video is high or the computational resource is limited.

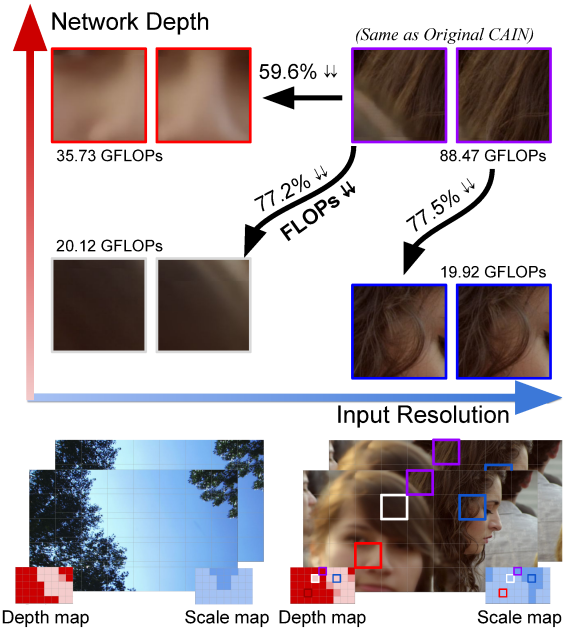

Figure 1. Motivation for this work. Purple patches represent the computational complexity of the original interpolation model, where the full depth is used for inference with the full-resolution inputs. Our method adaptively estimates the model depth and the input scale, saving excessive computations whenever possible.

However, we observe that not all regions require such heavy computation for video frame interpolation. Assume there are regions of two consecutive input frames that are completely static and has no motion between them. Then, the perfect interpolation would be a simple copy of either input frame, which does not need any convolutions. Fig. 1 shows an example where the plain regions, such as the blue sky, are interpolated with a much shallower model without losing any accuracy or visual quality. On the other hand, the regions with large motion or complex texture are relatively more complicated to interpolate.

In this work, we consider two aspects of the interpolation framework to reduce computation: input resolution and model depth. However, simply downsizing the spatial

*Currently at Google Research.

resolution of all inputs can lose the detailed textural information and considerably decrease the interpolation performance, and using a shallower model likewise reduces the average quality. To circumvent this problem, we propose a novel dynamic framework that can *adaptively* decide the specific local regions that are safe for performing such operations. Fig. 1 demonstrates the output of our model, where the "safe" regions are either downscaled, passes through a smaller number of layers, or both to reduce the computational complexity with minimal performance degradation.

For the proposed framework to dynamically allocate the appropriate amount of computation for each local region, we use the motion magnitude between two input frames as the level of complexity. Specifically, we introduce SD (Scale and Depth)-finder, which consists of two small meta-networks that estimate whether to downscale the inputs or exit the interpolation model early without passing through the full model. The input of the SD-finder is the approximate motion between the two input frames, which is calculated by a simple difference image or an optical flow estimation model. We then separate the input frames into multiple local regions, so that different amount of computation can be allocated for each region *w.r.t.* the corresponding scale and depth predicted by the SD-finder. The final interpolation is obtained by aggregating the outputs for all regions.

In summary, the proposed novel dynamic model enables locally adaptive inference for efficient frame interpolation, which effectively allocates proper computation by predicting the input scale and model depth using the amount of motion as the complexity criterion. Compared to the baseline frame interpolation model, the experimental results show that our framework is possible to save almost 50% of computation with little or no loss in performance when interpolating 2K resolution frames. Also, we analyze the accuracy-resource trade-off to investigate the missing pieces in making video frame interpolation models more practical.

2. Related Works

2.1. Video frame interpolation

Research in deep-learning-based video frame interpolation can be categorized into three directions: flow-based, kernel-based, and the others.

The standard technique for video frame interpolation aims to explicitly estimate the motion in the form of optical flow, warp the two input frames to the intermediate time step, and synthesize the occluded regions [12, 47]. These approaches introduced many novel ideas on how to better compensate for the occluded regions at motion boundaries, which include estimating flow in the voxel space [26], learning unsupervised or fine-tuned optical flow suited for frame interpolation [17, 53], using additional context for better synthesis [2, 3, 35, 40], recursively refining the estimated flow and the warped output [22], and softmax splatting for effective forward warping [36]. While being able to generate sharp and clean interpolations, the sequential process of explicit motion estimation and frame synthesis may increase the computational complexity of the model.

Kernel-based approach, also called adaptive convolution, is first proposed in frame interpolation by Niklaus *et al*. [37], where they unify motion estimation and compensation into a single convolution step with spatially-varying kernel predictions. Since then, much progress has been shown with novel ideas such as using separable convolutions [38, 39], extending the adaptive kernel predictions to be deformable [7, 21], combining with optical flow estimation [2, 3, 44], or formulating it as a loss function [43].

Frame interpolation approaches that do not use either adaptive kernels or optical flow include phase-based methods [32, 33], or direct pixel-level synthesis using deep networks [10, 27]. There are also a number of interesting research directions that exploit cycle consistency [25, 45], study non-linear motion models [24, 41, 52], or jointly consider frame interpolation with other video processing tasks such as deblurring [18, 46], super-resolution [20, 57], or stabilization [8]. While these methods excel in terms of performance, how to make the models computationally efficient has been less studied. Recently, a compression-driven interpolation network [11] is proposed and showed impressive performance and efficiency. We take an alternative approach and present a novel dynamic framework to reduce the computation for existing models, making them much more practical to real-world usage. Note that [11] and our method is orthogonal and can be jointly applied together.

2.2. Adaptive inference

Existing methods that make the model adaptive to its current input typically aim to enhance the computational efficiency of the network inference. Notable research directions include dynamically changing the inference path with early exits [4, 15], allocating adaptive computation time to different spatial regions [13, 48], adaptive skipping of some layers or residual blocks [49, 51], or adaptively changing the spatial resolution of the input images [30, 54]. Many of these efforts focus on strategically saving computation for "easy" samples while maintaining the overall accuracy, however, their effectiveness was only demonstrated for the standard image/video classification or detection tasks.

Recently, some techniques are employed for low-level computer vision application, including dynamic inference path selection for image denoising [55, 56] or using spatially sparse convolutions for image super-resolution [23]. However, many advancements in the classification problems are not directly applicable to low-level tasks. For instance, although changing the input resolution [30, 54] is proven to be useful for problems that consider high-level semantics of the visual data, downscaling can significantly deteriorate the performance of low-level problems due to

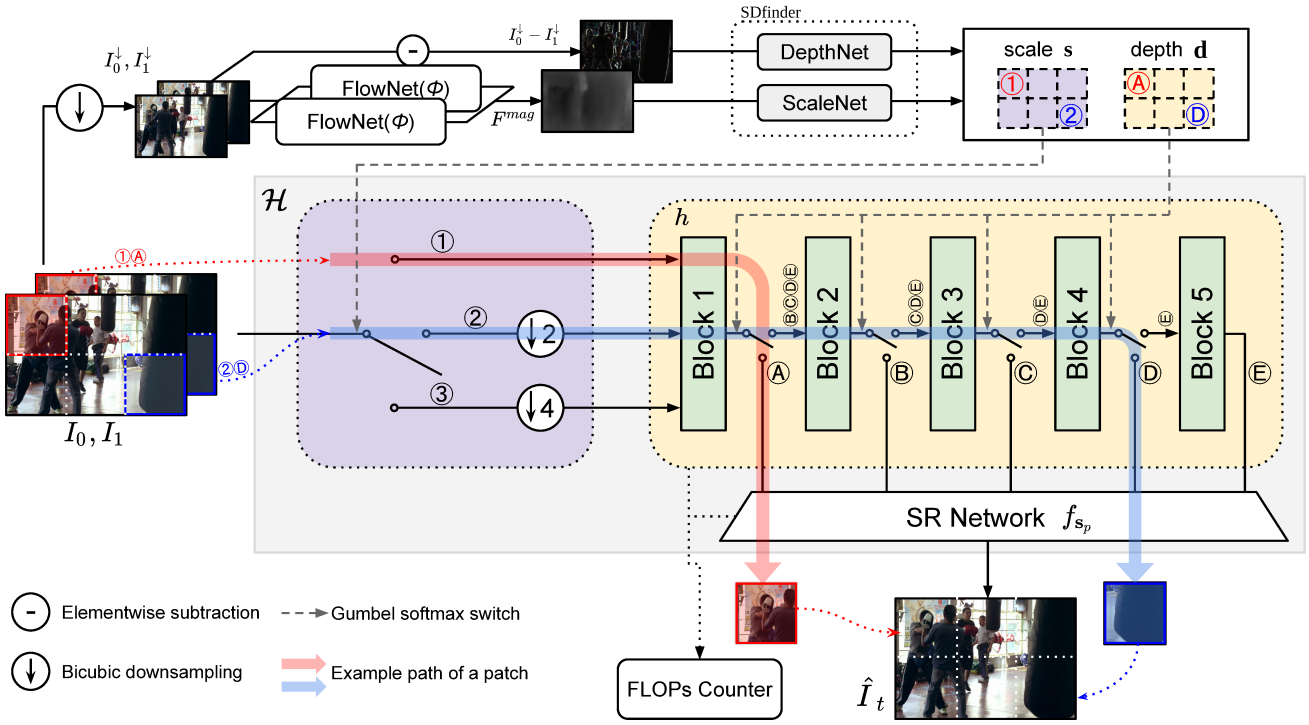

Figure 2. Illustration of the overall framework. Receiving the the flow magnitude (F^{mag}) and the difference of two downscaled frames ($I_0^{\downarrow} - I_1^{\downarrow}$), the SD-finder determines the input scaling factor and model depth for every region. Each region is put through the model $\mathcal{H}$ according to the designated scale and depth maps ($\mathbf{s}, \mathbf{d}$), which adaptively controls the inference path.

loss of textural details.

As for video frame interpolation, adapting to the input domain using cycle consistency [45] or to each input video clip using meta-learning [9] have been studied. While these approaches successfully improve the interpolation accuracy, excessive computation is needed for input-aware adaptation. On the other hand, we mainly focus on improving the computational efficiency at inference time.

3. Method

Given two consecutive frames I_0 and I_1, video frame interpolation aims to synthesize the intermediate frame $\hat{I}_t$, where $t \in (0, 1)$. In this work, we choose CAIN [10] as the baseline frame interpolation model and modify the architecture to be dynamic with respect to the input difficulty.

3.1. Dynamic framework overview

The proposed framework consists of four components: motion estimation network, scale and depth finder (SD-finder), super-resolution network, and the baseline frame interpolation model. Fig. 2 illustrates the training and inference processes. Formally, we first downscale the two input frames I_0 and I_1 to lower resolution (so that their minimum edge lengths become 192, in practice) using standard bicubic interpolation, obtaining $I_0^{\downarrow}$ and $I_1^{\downarrow}$. Then, we calculate the bidirectional optical flow $F_{0\to1}$ and $F_{1\to0}$ and compute the maximum motion magnitude for each pixel. Denoting the flow estimation network as ϕ and the magnitude of the flow map F as F^{mag},

$$F_{0\to1} = \phi\left(I_0^{\downarrow}, I_1^{\downarrow}\right), \quad F_{1\to0} = \phi\left(I_1^{\downarrow}, I_0^{\downarrow}\right), \tag{1}$$

$$F^{\text{mag}} = \max\left(F_{0\to1}^{\text{mag}}, F_{1\to0}^{\text{mag}}\right). \tag{2}$$

We take the maximum of the two magnitudes because the bidirectional flow vectors are not spatially aligned; thus, we conservatively mark the non-overlapping regions to have larger motion.

The computed motion magnitude and the subtraction of two downscaled input frames become inputs to SD-finder. Our proposed SD-finder adaptively determines the inference path by predicting the scale map $\mathbf{s}$ and depth map $\mathbf{d}$ for all local regions as follows:

$$(\mathbf{s}, \mathbf{d}) = \text{SD-finder}\left(F^{\text{mag}}, I_0^{\downarrow} - I_1^{\downarrow}\right). \tag{3}$$

Both $\mathbf{s}$ and $\mathbf{d}$ have the spatial resolution of $\mathbb{R}^{n_h \times n_w}$, where n_h and n_w are the predefined number of patches across height and width dimensions.

For the main part, we divide the input frames with the original resolution into $n_h \times n_w$ number of sub-regions. Let us denote a region index as p and its corresponding sub-regions as I_0^p and I_1^p, (estimated) scale as $\mathbf{s}_p$, and depth as

$\mathbf{d}_p$. Also, denote $(I_0^p)^{\downarrow}_{\mathbf{s}_p}$ and $(I_1^p)^{\downarrow}_{\mathbf{s}_p}$ as the I_0^p and I_1^p downscaled with a scaling factor $\mathbf{s}_p$, respectively. With $f_{\mathbf{s}_p}$ as the super-resolution model, h as our interpolation model, and $\mathcal{H}$ as the full combined model, the interpolated sub-region $\hat{I}_t^p$ is calculated as:

$$\hat{I}_t^p = \mathcal{H}\left(I_0^p, I_1^p \mid \mathbf{s}_p, \mathbf{d}_p\right) \tag{4}$$

$$= f_{\mathbf{s}_p}\left(h\left((I_0^p)^{\downarrow}_{\mathbf{s}_p}, (I_1^p)^{\downarrow}_{\mathbf{s}_p} \;\middle|\; \mathbf{d}_p\right)\right), \tag{5}$$

where $\mathcal{H}$ and h allow early exit with respect to $\mathbf{d}_p$.

Our final output is generated by merging all interpolated sub-regions. In the following sections, we describe the detailed training process and how each component is connected to be differentiable and hence end-to-end trainable with backpropagation.

3.2. Scale and depth finder (SD-finder)

The role of our proposed SD-finder is to decide the inference path in our dynamic framework, thereby allocating the appropriate amount of computation to each local region. The two components of SD-finder, ScaleNet and DepthNet, receive different inputs and do not share any parameters. Specifically, the magnitude of the flow map (F^{mag}) passes through the ScaleNet to generate the scale map $\mathbf{s}$, while the difference of image $I_0^{\downarrow} - I_1^{\downarrow}$ is used as the input to DepthNet to generate the depth map $\mathbf{d}$. The sizes of $\mathbf{s}$ and $\mathbf{d}$ are $\mathbb{R}^{n_s \times n_h \times n_w}$ and $\mathbb{R}^{n_d \times n_h \times n_w}$, respectively, where n_s is the number of scaling factors and n_d is the full depth of our interpolation model. In practice, we use $n_s = 3$ for the three scaling factors of 1, 2, and 4, and $n_d = 5$ for the number of residual groups in CAIN.

Let us first consider $\mathbf{s}_p$, the scale variable of the p-th local region. After setting the scale factor s_i ($i = 1, 2, ..., n_s$), we define π_i as the probability of choosing the scale s_i. From the categorical distribution of $\pi_1, \pi_2, ..., \pi_{n_s}$, we can draw the discrete sample m_p^s using the Gumbel-Max trick [29],

$$m_p^s = \texttt{one\_hot}\left(\arg\max_k \left[g_k + \log\pi_k\right]\right), \tag{6}$$

where $g_j \sim \text{Gumbel}(0, 1)$ is an i.i.d noise sample. However, sampling from a categorical distribution makes our framework non-differentiable. Therefore, we relax m_p^s to be continuous using the Gumbel-Softmax trick [16, 28], replacing the argmax operation in Eq. (6) with a softmax function. The i-th element of m_p^s is calculated as

$$m_{p,i}^s = \frac{\exp\left[(\log(\pi_i) + g_i)/\tau\right]}{\sum_k \exp\left[(\log(\pi_k) + g_k)/\tau\right]}. \tag{7}$$

We use the fixed temperature $\tau = 1$ in our experiments.

Although the continuous relaxation makes our model differentiable in the backward pass, choosing the scale path for the forward pass still needs to be discrete. Thus, we use the Straight-Through (ST) Gumbel-Softmax trick [16] which allows m_p^s to be discrete as in Eq. (6) for the forward pass and continuous as in Eq. (7) for the backward pass. At test time, the discrete mask is sampled, but we do not add gumbel noise g_j when applying the argmax operation to remove the randomness.

The depth variable, $\mathbf{d}_p$, follows the similar process to $\mathbf{s}_p$. Sampling the discrete depth m_p^d, a one-hot representation of d_j, can be done as in Eq. (6), and its continuous relaxation can be obtained as in Eq. (7). Training with the discrete depth also uses the ST Gumbel-Softmax trick.

Note that, the calculation of motion magnitude (Eq. (1)-(2)) and passing through SD-finder (Eq. (3)) are additional components that are not existent in the original CAIN model. Thus, to keep this extra computation minimal, we use an extremely simple 3-layer CNN as the architectures of our scale/depth networks. Since we also downscale the input frames before calculating the motion magnitude, the extra computation induced by calculating Eq. (1)-(3) is only 1-5% of the original full model for an HD-resolution input and becomes negligible for higher input resolution.

3.3. Dynamic interpolation model

In this section, we describe how to modify the baseline interpolation model (CAIN, denoted as h) to be dynamic, with multiple scale paths and allowing for early exits. As illustrated in Fig. 2, each local region of the input frame is downscaled *w.r.t.* the scale s_i. The one-hot representation of the scale sample, m_p^s (see Eq. (6)), has binary values in {0, 1} and can be used as a masking variable. Then, the output interpolation calculated using Eq. (4) can be expressed as

$$\hat{I}_t^p = \sum_i m_{p,i}^s \cdot \mathcal{H}\left(I_0^p, I_1^p \,;\, s_i, \mathbf{d}_p\right). \tag{8}$$

Using the ST Gumbel-Softmax trick described in Sec. 3.2, we can differentiate through the discrete switching of the inference path for each scale.

To incorporate the depth variable $\mathbf{d}_p$, we set n_d as the number of exits to our interpolation model h (Ⓐ-Ⓔ in Fig. 2). Denoting the output of each computation block of h (residual group of CAIN) as $\mathcal{B}_j^p, (j = 1, 2, ..., n_d)$, we can change the expression in Eq. (5) into

$$\hat{I}_t^p = f_{\mathbf{s}_p}\left(h\left(\cdot, \cdot \mid \mathbf{d}_p\right)\right) = f_{\mathbf{s}_p}\left(\sum_j m_{p,j}^d \cdot \mathcal{B}_j^p\right). \tag{9}$$

Similar to switching the scale, we can also differentiate through the depth switches using the ST Gumbel-Softmax trick. If we consider all masking variables for the scale and depth, the final interpolation for the current local region p can be computed as

$$\hat{I}_t^p = \sum_{i=1}^{n_s} m_{p,i}^s \cdot f_{\mathbf{s}_p}\left(\sum_{j=1}^{n_d} m_{p,j}^d \cdot \mathcal{B}_j^p\right). \tag{10}$$

We can obtain the full-frame prediction $\hat{I}_t$ by merging all sub-regions without overlap.

3.4. Training

Objective We use two types of objective functions: reconstruction loss $\mathcal{L}_r$ to measure the interpolation accuracy, and the resource-aware regularization term $\mathcal{R}$ to reduce the computational complexity. We use a standard pixel-wise ℓ_1 loss as $\mathcal{L}_r$. For $\mathcal{R}$, following the approaches used in research areas for neural architecture search [6, 19, 50], we calculate the number of floating point operations (FLOPs) of each component of our dynamic model and directly use it as the regularization term. Specifically, since the total number of operations is proportional to the input size, we divide it into the spatial dimension and compute per-pixel FLOPs. Let us denote the function to calculate the FLOPs of a model as $\mathcal{C}$. Given a fixed inference path with the scale s_i and the depth d_j and the corresponding one-hot mask vectors $m^s_{p,i}$ and $m^d_{p,j}$, the computational resource for the local region p can be calculated as

$$\mathcal{R}_p = \sum_{i=1}^{n_s}\sum_{j=1}^{n_d} m^s_{p,i} \cdot m^d_{p,j} \cdot \mathcal{C}\left(\mathcal{H}(\cdot,\cdot\,;\,s_i,d_j)\right). \quad (11)$$

Combining all regions, the final per-pixel FLOPs becomes

$$\mathcal{R} = \frac{1}{H \times W}\sum_{p=1}^{n_p} |p| \cdot \mathcal{R}_p, \quad (12)$$

where H and W are the height and width of the original input resolution, $|p|$ is the area of p, and n_p $(= n_h \times n_w)$ is the number of local regions. Note that, in practice, $\sum_{p=1}^{n_p}|p|$ can be bigger than $H \times W$ if we make p overlapping with its neighboring regions to mitigate the boundary effects.

The final objective function combines both terms with a hyperparameter λ, which controls the trade-off between the accuracy and efficiency:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_r + \lambda\mathcal{R}. \quad (13)$$

Using a small λ will push the model to use the original resolution and the full depth to achieve high accuracy with more computation. The larger λ will make the model more efficient by appropriately downscaling the spatial resolution as well as exiting the interpolation model early, but performance may drop accordingly.

Curriculum training Although our framework is differentiable, end-to-end learning of all components at once leads to unstable training, which attributes to the the highly discretized FLOPS for each inference path. Therefore, we separate the training into several steps, and design the training curriculum as follows:

1. Pretrain the baseline multi-exit version of the frame interpolation model (CAIN, h), so that outputs $\mathcal{B}_j$, $j = 1, ..., n_d$ from any exit show good interpolations.

2. Using the different exits, jointly train the DepthNet of the SD-finder, the interpolation network (h), and the super-resolution network (f). We fix the scale to the original input resolution for this step.

3. Fix the parameters of the DepthNet and train the ScaleNet jointly with h and f until convergence.

In step 1, the computational complexity is fixed, so we train the model with $\mathcal{L}_r$ only. On the other hand, steps 2 and 3 incorporates the full objective including the resource-aware regularization. However, since we decompose the training of depth estimation (step 2) and scale estimation (step 3), calculation of the per-pixel resource consumption in Eq. (11) can be reduced to (see supplementary)

$$\mathcal{R}_p = \lambda_s \sum_{i=1}^{n_s} m^s_{p,i} \cdot \mathcal{C}\left(\mathcal{H}_{i,\hat{\mathbf{d}}}\right) + \lambda_d \sum_{j=1}^{n_d} m^d_{p,j} \cdot \mathcal{C}\left(\mathcal{H}_{\hat{\mathbf{s}},j}\right), \quad (14)$$

where we abuse the notation $\mathcal{H}_{i,j}$ to indicate the inference path through $\mathcal{H}(\cdot,\cdot\,;\,s_i,d_j)$, and $\hat{\mathbf{s}}$ and $\hat{\mathbf{d}}$ denotes the fixed scale and depth, respectively.

4. Experiments

4.1. Datasets

In this work, we use 3 datasets for training and evaluation.
Vimeo-90K [53] triplet is a widely-used dataset due to its clean, high-quality frames with little noise. However, its spatial resolution is small (448 × 256), making it not suitable for training the models that aims to handle high resolution frames with extremely large motion. Thus, we use Vimeo-90K only for the first step of our traning curriculum.
REDS-120fps [34] is a challenging high-fps video dataset recently made public, and all frames are of HD resolution (1280 × 720). We train the remaining steps (2 and 3) with REDS-120fps by randomly sampling three consecutive frames with one (60 → 120-fps) or two (30 → 60-fps) frame gaps. For validation, we evaluate on the first 50 frames for each sequence in the validation split in 30 → 60-fps setting since 30-fps is common in many real-world videos.
Xiph[1] videos are used for evaluation in [36], and we follow the similar settings. While the original frames are near 4K resolution (4096 × 2160), we use the downsampled version as Xiph-2K (2048 × 1080) and the center-cropped version as Xiph-"4K", where the resolution is 2K but the motion magnitude levels are of 4K.

[1] https://media.xiph.org/video/derf/

Table 1. Quantitative results for the proposed framework on Xiph videos. Computational complexity is measured in tera-FLOPs (TFLOPs) and CPU/GPU time (sec.), and the performance is measured in PSNR (dB) and SSIM.

	Xiph-2K					Xiph-"4K"				
	TFLOPs$_\downarrow$	CPU time$_\downarrow$	GPU time$_\downarrow$	PSNR$_\uparrow$	SSIM$_\uparrow$	TFLOPs$_\downarrow$	CPU time$_\downarrow$	GPU time$_\downarrow$	PSNR$_\uparrow$	SSIM$_\uparrow$
SepConv [38]	2.078	11.07	**0.218**	34.85	0.9308	2.078	11.10	**0.219**	32.10	0.8861
SuperSloMo [17]	2.957	17.86	0.337	33.88	0.9247	2.957	18.12	0.352	31.99	0.8800
AdaCoF + [21]	5.433	26.70	0.518	35.09	0.9309	5.433	26.72	0.522	32.19	0.8818
DAIN [2]	13.22	66.43	4.619	**35.97**	**0.9400**	13.22	66.45	4.620	**33.51**	0.8983
CAIN [10]	3.133	10.64	0.225	35.21	0.9366	3.133	10.35	0.239	32.56	**0.9005**
CAIN-SD (Ours)	**1.598**	**8.83**	0.237	34.68	0.9235	**1.983**	**9.25**	0.242	32.92	0.8934

Table 2. Validation set performance on REDS-120fps.

	TFLOPs$_\downarrow$	CPU time$_\downarrow$	GPU time$_\downarrow$	PSNR$_\uparrow$	SSIM$_\uparrow$
CAIN [10]	1.305	4.15	0.103	28.62	0.8303
CAIN-S	1.507	4.76	0.127	28.58	0.8281
CAIN-D	1.008	3.67	0.101	28.29	0.8214
CAIN-SD	0.882	3.82	0.122	28.32	0.8212

4.2. Implementation details

In general, we follow the details from the original CAIN [10]. We use PWC-Net [47] for optical flow estimation, and the mini version of CARN, CARN-M [1], is used for the image super-resolution (SR) model. We use the patch size of 256×256 To train the interpolation model. For calculating the SD-finder inputs, however, we downscale the input frames to have the minimum edge length of 192 to keep the additional computation minimal. We use PyTorch [42] framework for all of our implementation. We use a single NVIDIA Quadro RTX6000 for training, and also when measuring the test time. The code and the pretrained models will be made public for reproducibility and further research. For the additional training details, please refer to the supplementary materials.

4.3. Quantitative comparison

Metrics. For the computational complexity, we calculate the number of floating-point operations (FLOPs) and the actual running time (latency) in CPU and GPU. For the performance measures, we use the standard peak signal-to-noise ratio (PSNR) and structural similarity index (SSIM).

Compared models. We report three variants of the proposed method, using CAIN as the baseline. The *scale-only* version only considers the ScaleNet of SD-finder by skipping step 2 of our training curriculum, and we denote it as **CAIN-S**. For this setting, we do not use multi-exits and fix the depth of the interpolation model to maximum (original). The *depth-only* version skips the third training step and only consider the DepthNet of SD-finder, which is denoted as **CAIN-D**. Lastly, both scale and depth are adaptively reduced for **CAIN-SD**, which is our final version. We also report the performance and the computational complexity of the existing frame interpolation models: SepConv [38], SuperSloMo [17], AdaCoF + [21], DAIN [2], and CAIN [10].

Results. The quantitative results for Xiph-2K and Xiph-"4K" videos are shown in Table 1. For Xiph-2K, our final CAIN-SD model can reduce the FLOP of the original CAIN both by 49%, with small loss in the performance measures. However, for the actual CPU/GPU latency, which includes all parts (optical flow estimation, SD-finder, interpolation, and super-resolution) is not improved as much; while the CPU runtime is improved by 17%, CAIN-SD runs slightly slower than the original CAIN on GPU. This is mainly due to the slow latency in SR model; out of 236.7 ms average GPU latency, SR model alone consumes 71.3 ms. The remaining parts are indeed made efficient by running in 165.4 ms. We believe that jointly learning a more efficient SR model with our framework can further reduce the CPU/GPU latency, which remains as our future work.

For Xiph-"4K", CAIN-SD can save 36.7% FLOPs compared to the baseline, with even higher PSNR. We believe the higher performance is due to the scaling capability of CAIN-SD. Since 4K videos usually contain extremely large motions (sometimes over 100 pixels), it is hard to interpolate or compensate for using the existing models. However, by downscaling the inputs, the effective amount of motion is reduced by the scaling factor, and our models can find the correct intermediate position better, as shown in the visual results (Sec. 4.4). We believe that further exploiting the scale space is crucial for handling high-resolution frames for future research. Note that, although both Xiph-2K and "4K" have the same spatial resolution and the FLOPs for the existing models stay the same, the proposed method automatically saves the computation *w.r.t.* the inherent motion in order to improve the efficiency.

Compared to the existing frame interpolation models other than the baseline CAIN, our CAIN-SD always require the smallest FLOPs and CPU runtime. SepConv shows the fastest GPU latency but with more FLOPs; also, the performance of CAIN-SD for Xiph-"4K" is significantly better than SepConv, with more than 0.8dB PSNR gain. DAIN shows the best performance, but it is extremely slow with heavy compute. Compared to SuperSloMo and AdaCoF +, CAIN-SD is computationally more efficient in all measures

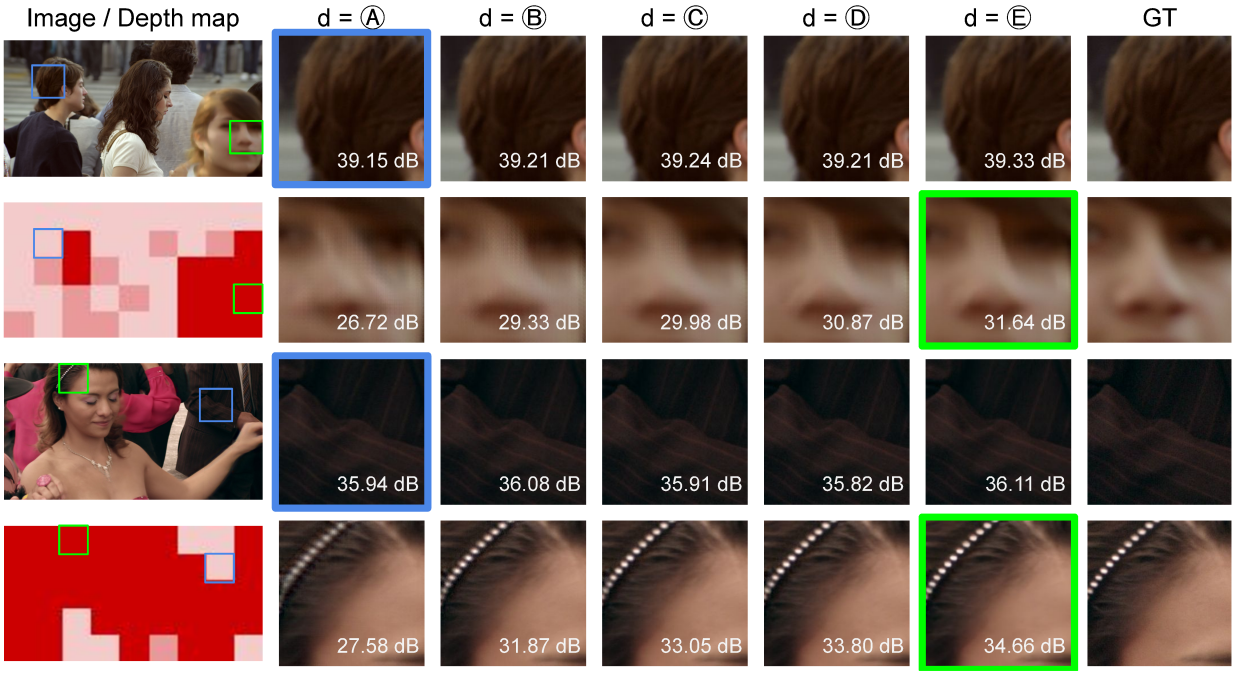

Figure 3. Comparison between the local regions with different depth predictions. d =Ⓔ represents the full model inference (original CAIN), while d =Ⓐ shows the output at the first early exit. Blue boxes show the stationary regions where the model already performs well on its first exit and saves excessive computations. On the other hand, deeper layers are needed to maintain the performance for regions with large motions depicted in the green boxes, where our model decides to pass through the full interpolation model.

with competitive or better performance.

Table 2 shows the results for the variants of our proposed model on REDS-120fps dataset. The results of CAIN-S shows that only reducing the scale is not very effective in improving the computational complexity, showing even more FLOPs due to the usage of overlapping patches. This tells us that the ScaleNet of CAIN-S mostly do not downscale the input frames to maintain the performance. CAIN-D, on the other hand, successfully saves 23% of FLOPs and 12% of CPU time, but with more decrease in performance. Our final model, CAIN-SD, also saves 32.4% FLOPs with 0.3 dB PSNR loss, similar to Xiph-2K and Xiph-"4K" results. However, since REDS-120fps dataset has lower resolution than Xiph, the effects of the constant additional computation including the flow estimation and the SD-finder becomes relatively larger than the savings obtainable by FLOPs reduction, and the overall running speed becomes relatively slower. For more detailed analysis on FLOPs and latency, please refer to our supplementary document.

4.4. Visual comparison

We visualize the depth and scale masks predicted by our SD-finder and analyze the results. Fig. 3 illustrates the depth predictions made by our DepthNet. The regions that already perform well on the first exit of our multi-exit CAIN stops proceeding to further layers and save significant amount of computations, while more difficult regions that need further processing passes through the later exits. We can observe that, in general, the local regions with large motion and complex, non-uniform texture require more depth.

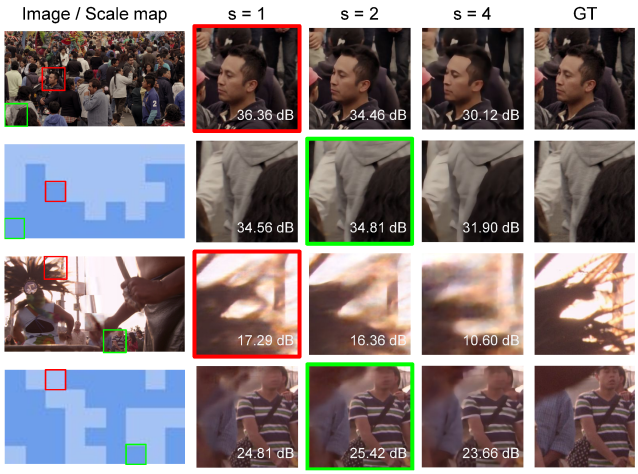

Figure 4. Comparison between the regions with different scale predictions. Regions with the red box significantly lose the textural details when the input frames are downscaled, so the original scale is used to maintain the performance. Green box shows the regions where our SD-finder chooses the scaling factor of 2, where we can save computations compared to $s = 1$ with even increased PSNR.

Fig. 4 depicts the predicted scales made by the ScaleNet. Downsizing the input resolution and upscaling back with our SR model loses some precise textural details, but it can sometimes increase the performance while improving the efficiency at the same time, especially for the regions

Table 3. Effects of resource-aware regularization hyperparameters. We report the quantitative results for Xiph-2K videos.

λ_s	λ_d	TFLOPs$_\downarrow$	PSNR$_\uparrow$	SSIM$_\uparrow$
5×10^{-9}	5×10^{-10}	2.154	33.94	0.9178
	5×10^{-11}	1.598	34.68	0.9235
	5×10^{-12}	3.804	35.33	0.9334
5×10^{-8}	5×10^{-11}	1.827	31.10	0.8599
5×10^{-9}		1.598	34.68	0.9235
5×10^{-10}		1.699	34.71	0.9244

with large motion. For additional qualitative comparison in different scenes, please refer to our project page: https://myungsub.github.io/adaptive-int.

4.5. Ablation study

We analyze how each aspects of our proposed framework affects the trade-off between performance and efficiency.

Accuracy-resource trade-off. In Table 3, we vary λ_s and λ_d to investigate the changes in performance and FLOPs. First, we fix λ_s to the value for our final CAIN-SD model, 5×10^{-9}, and modify λ_d. We conjecture that as λ_d gets smaller, the resource-aware regularization term decreases and the loss is more dominated by the pixel-wise reconstruction errors, leading to higher performance. However, FLOPs for $\lambda_d = 5 \times 10^{-10}$ is higher than $\lambda_d = 5 \times 10^{-9}$ with a large margin. We attribute this phenomenon to the mismatch between the ratio between λ_s and λ_d. Though we fix each hyperparameter to examine its effects separately for this analysis, we found $\frac{\lambda_d}{\lambda_s} = 0.01$ to be a good ratio to control the accuracy-resource trade-off in practice.

Setting λ_d fixed with varying λ_s also shows a similar phenomenon, where the performance increases with smaller value of λ_s. When $\lambda_s = 5 \times 10^{-8}$, however, we observe that the scales predicted by SD-finder is highly polarized to either extremely downscale with a scaling factor of 4 or to remain in its original resolution. We believe that downscaling with such a big scaling factor excessively lose the image details, making the regions not able to recover the performance. Since the other regions have to make up for the low performance, they remain in the original scale and contribute to make FLOPs larger.

Effects on patch size. Although the training patch size for our method is fixed to 256, we can change the patch size at the inference stage, so we study its effects in Table 4. For the 'full frame', we regard the whole input frame as a single large patch. However, in full-frame test setting, the performance deteriorates significantly since downscaling the input or exitting the model early is conducted as a whole, and the regions with detailed texture get over-smoothed. The large patch size of 512 shows higher performance with more computation due to the bigger overlap size of the neighboring patches. When we use a small patch size of 128, the accuracy is considerably lower than 256, even with more FLOPs. We believe this is due to the high resolution of the input frames, and the SD-finder may not be able to decide the appropriate scale and depth with such limited spatial context. The result of patch size 256 is the closest to the optimum in the accuracy-resource trade-off, which explains why we chose 256 for all other experiments.

Table 4. Effects of patch size at inference stage for Xiph-2K.

Test	TFLOPs$_\downarrow$	GPU time$_\downarrow$	PSNR$_\uparrow$	SSIM$_\uparrow$
Full frame	1.530	222.2	33.24	0.8972
512	2.190	304.0	34.88	0.9248
256	1.598	236.7	34.68	0.9243
128	1.712	266.6	33.89	0.9184

Varying the inputs and weights of SD-finder. For our SD-finder, ScaleNet receives the magnitude of the optical flow estimation as its input, and DepthNet receives the difference image. Using the other combinations of input modalities for SD-finder did not show any improvements; *e.g.* 1) using both modalities resulted in the same accuracy with slightly more FLOPs, 2) ScaleNet without flow magnitude input or DepthNet without difference image input resulted in significantly worse accuracy, and 3) Using RGB images as additional inputs also worsened the accuracy and the computational complexity. For more detailed results and analyses, please refer to our project page.

5. Conclusion

In this work, we exploit the accuracy-resource trade-off of the existing video frame interpolation model and present a dynamic framework to improve its computational efficiency. The proposed SD-finder adaptively estimates the input resolution (scale) and the model depth, allocating the proper amount of computation for each local region by deciding the inference path *w.r.t.* the level of motion between the two input frames. This is achieved by allowing our interpolation model to have multiple exits and incorporating a super-resolution model to restore the textural details of the downscaled inputs. Consequently, the experimental results show that the proposed framework can save almost 50% of the FLOPs on average compared to the baseline while maintaining the interpolation quality. Our framework is especially more effective in high-resolution scenarios, where many existing approaches fail to generate accurate interpolations even with high computational cost.

Acknowledgements This work was supported in part by IITP grant funded by the Korea government [No. 2021-0-01343, Artificial Intelligence Graduate School Program (Seoul National University)], and in part by AIRS Company in Hyundai Motor and Kia through HMC/KIA-SNU AI Consortium Fund.

References

[1] Namhyuk Ahn, Byungkon Kang, and Kyung-Ah Sohn. Fast, accurate, and lightweight super-resolution with cascading residual network. In *ECCV*, 2018. 6

[2] Wenbo Bao, Wei-Sheng Lai, Chao Ma, Xiaoyun Zhang, Zhiyong Gao, and Ming-Hsuan Yang. Depth-aware video frame interpolation. In *CVPR*, 2019. 1, 2, 6

[3] Wenbo Bao, Wei-Sheng Lai, Xiaoyun Zhang, Zhiyong Gao, and Ming-Hsuan Yang. Memc-net: Motion estimation and motion compensation driven neural network for video interpolation and enhancement. *arXiv preprint arXiv:1810.08768*, 2018. 2

[4] Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. Adaptive neural networks for efficient inference. In *ICML*, 2017. 2

[5] Tim Brooks and Jonathan T Barron. Learning to synthesize motion blur. In *CVPR*, 2019. 1

[6] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *ICLR*, 2019. 5

[7] Xianhang Cheng and Zhenzhong Chen. Video frame interpolation via deformable separable convolution. In *AAAI*, 2020. 2

[8] Jinsoo Choi and In So Kweon. Deep iterative frame interpolation for full-frame video stabilization. *ACM Transactions on Graphics (TOG)*, 39(1):1–9, 2020. 1, 2

[9] Myungsub Choi, Janghoon Choi, Sungyong Baik, Tae Hyun Kim, and Kyoung Mu Lee. Scene-adaptive video frame interpolation via meta-learning. In *CVPR*, 2020. 3

[10] Myungsub Choi, Heewon Kim, Bohyung Han, Ning Xu, and Kyoung Mu Lee. Channel attention is all you need for video frame interpolation. In *AAAI*, 2020. 1, 2, 3, 6

[11] Tianyu Ding1, Luming Liang, Zhihui Zhu, and Ilya Zharkov. Cdfi: Compression-driven network design for frame interpolation. In *CVPR*, 2021. 2

[12] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick Van Der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. In *ICCV*, 2015. 2

[13] 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. 2

[14] Yuan Gao, Reinhard Koch, Robert Bregovic, and Atanas Gotchev. Fast: Flow-assisted shearlet transform for densely-sampled light field reconstruction. In *ICIP*, 2019. 1

[15] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q Weinberger. Multi-scale dense networks for resource efficient image classification. In *ICLR*, 2017. 2

[16] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *ICLR*, 2017. 4

[17] Huaizu Jiang, Deqing Sun, Varun Jampani, Ming-Hsuan Yang, Erik Learned-Miller, and Jan Kautz. Super slomo: High quality estimation of multiple intermediate frames for video interpolation. In *CVPR*, 2018. 2, 6

[18] Meiguang Jin, Zhe Hu, and Paolo Favaro. Learning to extract flawless slow motion from blurry videos. In *CVPR*, 2019. 2

[19] Heewon Kim, Seokil Hong, Bohyung Han, Heesoo Myeong, and Kyoung Mu Lee. Fine-grained neural architecture search. *arXiv preprint arXiv:1911.07478*, 2019. 5

[20] Soo Ye Kim, Jihyong Oh, and Munchurl Kim. Fisr: Deep joint frame interpolation and super-resolution with a multi-scale temporal loss. In *AAAI*, 2020. 2

[21] Hyeongmin Lee, Taeoh Kim, Tae young Chung, Daehyun Pak, Yuseok Ban, and Sangyoun Lee. Adacof: Adaptive collaboration of flows for video frame interpolation. In *CVPR*, 2020. 2, 6

[22] Haopeng Li, Yuan Yuan, and Qi Wang. Video frame interpolation via residue refinement. In *ICASSP*, 2020. 2

[23] Ming Liu, Zhilu Zhang, Liya Hou, Wangmeng Zuo, and Lei Zhang. Deep adaptive inference networks for single image super-resolution. In *ECCV Workshops*, 2020. 2

[24] Yihao Liu, Liangbin Xie, Li Siyao, Wenxiu Sun, Yu Qiao, and Chao Dong. Enhanced quadratic video interpolation. *arXiv preprint arXiv:2009.04642*, 2020. 2

[25] Yu-Lun Liu, Yi-Tung Liao, Yen-Yu Lin, and Yung-Yu Chuang. Deep video frame interpolation using cyclic frame generation. In *AAAI*, 2019. 2

[26] Ziwei Liu, Raymond A Yeh, Xiaoou Tang, Yiming Liu, and Aseem Agarwala. Video frame synthesis using deep voxel flow. In *ICCV*, 2017. 2

[27] Gucan Long, Laurent Kneip, Jose M Alvarez, Hongdong Li, Xiaohu Zhang, and Qifeng Yu. Learning image matching by simply watching video. In *ECCV*, 2016. 2

[28] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *ICLR*, 2017. 4

[29] Chris J Maddison, Daniel Tarlow, and Tom Minka. A* sampling. In *NIPS*, 2014. 4

[30] Yue Meng, Chung-Ching Lin, Rameswar Panda, Prasanna Sattigeri, Leonid Karlinsky, Aude Oliva, Kate Saenko, and Rogerio Feris. Ar-net: Adaptive frame resolution for efficient action recognition. In *ECCV*, 2020. 2

[31] Simone Meyer, Victor Cornillère, Abdelaziz Djelouah, Christopher Schroers, and Markus Gross. Deep video color propagation. In *BMVC*, 2018. 1

[32] Simone Meyer, Abdelaziz Djelouah, Brian McWilliams, Alexander Sorkine-Hornung, Markus Gross, and Christopher Schroers. Phasenet for video frame interpolation. In *CVPR*, 2018. 2

[33] Simone Meyer, Oliver Wang, Henning Zimmer, Max Grosse, and Alexander Sorkine-Hornung. Phase-based frame interpolation for video. In *CVPR*, 2015. 2

[34] Seungjun Nah, Sanghyun Son, Radu Timofte, and Kyoung Mu Lee. Aim 2019 challenge on video temporal super-resolution: Methods and results. In *ICCV Workshops*, Oct 2019. 5

[35] Simon Niklaus and Feng Liu. Context-aware synthesis for video frame interpolation. In *CVPR*, 2018. 2

[36] Simon Niklaus and Feng Liu. Softmax splatting for video frame interpolation. In *CVPR*, 2020. 1, 2, 5

[37] Simon Niklaus, Long Mai, and Feng Liu. Video frame interpolation via adaptive convolution. In *CVPR*, 2017. 2

[38] Simon Niklaus, Long Mai, and Feng Liu. Video frame interpolation via adaptive separable convolution. In *ICCV*, 2017. 2, 6

[39] Simon Niklaus, Long Mai, and Oliver Wang. Revisiting adaptive convolutions for video frame interpolation. *arXiv preprint arXiv:2011.01280*, 2020. 2

[40] Junheum Park, Keunsoo Ko, Chul Lee, and Chang-Su Kim. Bmbc: Bilateral motion estimation with bilateral cost volume for video interpolation. In *ECCV*, 2020. 2

[41] Sunghyun Park, Kangyeol Kim, Junsoo Lee, Jaegul Choo, Joonseok Lee, Sookyung Kim, and Edward Choi. Vid-ode: Continuous-time video generation with neural ordinary differential equation. *arXiv preprint arXiv:2010.08188*, 2020. 2

[42] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017. 6

[43] Tomer Peleg, Pablo Szekely, Doron Sabo, and Omry Sendik. Im-net for high resolution video frame interpolation. In *CVPR*, 2019. 2

[44] Fitsum A Reda, Guilin Liu, Kevin J Shih, Robert Kirby, Jon Barker, David Tarjan, Andrew Tao, and Bryan Catanzaro. Sdc-net: Video prediction using spatially-displaced convolution. In *ECCV*, 2018. 2

[45] Fitsum A Reda, Deqing Sun, Aysegul Dundar, Mohammad Shoeybi, Guilin Liu, Kevin J Shih, Andrew Tao, Jan Kautz, and Bryan Catanzaro. Unsupervised video interpolation using cycle consistency. In *ICCV*, 2019. 2, 3

[46] Wang Shen, Wenbo Bao, Guangtao Zhai, Li Chen, Xiongkuo Min, and Zhiyong Gao. Blurry video frame interpolation. In *CVPR*, 2020. 2

[47] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume. In *CVPR*, 2018. 2, 6

[48] Thomas Verelst and Tinne Tuytelaars. Dynamic convolutions: Exploiting spatial sparsity for faster inference. In *CVPR*, 2020. 2

[49] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *ECCV*, 2018. 2

[50] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR*, 2019. 5

[51] 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. 2

[52] Xiangyu Xu, Siyao Li, Wenxiu Sun, Qian Yin, and Ming-Hsuan Yang. Quadratic video interpolation. In *NeurIPS*, 2019. 2

[53] Tianfan Xue, Baian Chen, Jiajun Wu, Donglai Wei, and William T Freeman. Video enhancement with task-oriented flow. In *CVPR*, 2018. 2, 5

[54] Le Yang, Yizeng Han, Xi Chen, Shiji Song, Jifeng Dai, and Gao Huang. Resolution adaptive networks for efficient inference. In *CVPR*, 2020. 2

[55] Ke Yu, Chao Dong, Liang Lin, and Chen Change Loy. Crafting a toolchain for image restoration by deep reinforcement learning. In *CVPR*, 2018. 2

[56] Ke Yu, Xintao Wang, Chao Dong, Xiaoou Tang, and Chen Change Loy. Path-restore: Learning network path selection for image restoration. *arXiv preprint arXiv:1904.10343*, 2019. 2

[57] Liad Pollak Zuckerman, Shai Bagon, Eyal Naor, George Pisha, and Michal Irani. Across scales & across dimensions: Temporal super-resolution using deep internal learning. In *ECCV*, 2020. 2