Total Variation Optimization Layers for Computer Vision

Raymond A. Yeh[†] Yuan-Ting Hu Zhongzheng Ren Alexander G. Schwing
Toyota Technological Institute at Chicago[†] University of Illinois at Urbana-Champaign

CVPR June 19-24 2022 New Orleans • Louisiana

Motivation

Motivation:

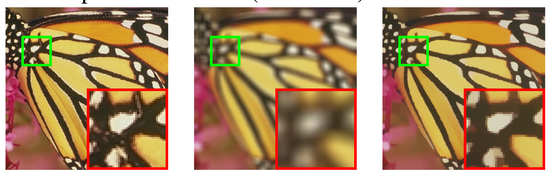
- Optimization as building blocks in deep-nets
- Total Variation (TV) is widely adopted
- Using TV as a deep-net layer?

Contribution:

- Propose total variation as a layer for deep-nets
- Develop a fast GPU-based TV solver
- Demonstrate practicality of TV layers on computer vision tasks

Introduction

Illustration of Total Variation:



- Preserves edges while smoothing

Proximal TV operator on 1D Input:

$$\arg\min_{\boldsymbol{y}} \frac{1}{2}\|\boldsymbol{y}-\boldsymbol{x}\|_2^2 + \lambda \|\boldsymbol{D}_N \boldsymbol{y}\|_1$$

- Input $\boldsymbol{x} \in \mathbb{R}^N$ and output $\boldsymbol{y} \in \mathbb{R}^N$
- $\|\boldsymbol{D}_N \boldsymbol{y}\|_1 = \sum_n |y_{n+1} - y_n|$
- $\lambda \geq 0$ controlling strength of regularization

Approach

Total Variation Optimization Layers:

- $\mathrm{Prox}_{\mathrm{TV}}^{1D}(\boldsymbol{x}, \lambda) \triangleq \arg\min_{\boldsymbol{y}} \frac{1}{2}\|\boldsymbol{y}-\boldsymbol{x}\|_2^2 + \lambda \|\boldsymbol{D}_N \boldsymbol{y}\|_1$
- $\mathrm{Prox}_{\mathrm{TV}}^{2D}(\boldsymbol{X}, \lambda) = \arg\min_{\boldsymbol{Y}} \frac{1}{2}\|\boldsymbol{Y}-\boldsymbol{X}\|_F^2 + \lambda \left(\sum_m \|\boldsymbol{D}_M \boldsymbol{Y}_{\mathtt{row(m)}}\|_1 + \sum_n \|\boldsymbol{D}_N \boldsymbol{Y}_{\mathtt{col(n)}}\|_1\right)$
- A single trainable parameter λ
- Perform "sharpening" with a residual connection, *i.e.*, $\boldsymbol{Y} = 2\boldsymbol{X} - \mathrm{Prox}_{\mathrm{TV}}^{2D}(\boldsymbol{X}, \lambda)$
- Insert into deep-net with feature dimension $C \times H \times W$
- Layer implemented as a `nn.Module` in Pytorch
- `github.com/raymondyeh07/tv_layers_for_cv`

Implementation

Custom CUDA Implementation:

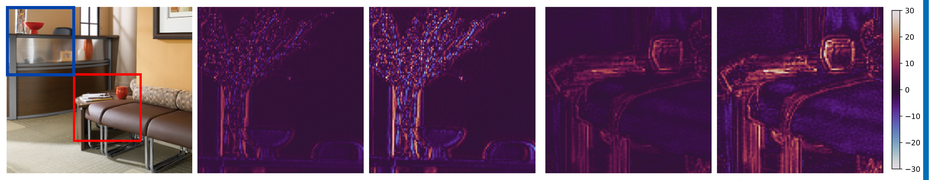
- Fast GPU TV solver (Projected-Newton) with custom CUDA kernels
- Our implementation considers the structures of the problem, *e.g.*, Tridiagonal system
- Solve 2D TV proximity problem using Proximal Dykstra method
- Alternates between solving 1D TV proximity problem per row/column

Timing Comparison:

Package	Hardware	Forward	Backward	Total
CVXPYLayers	CPU	20704 ± 32	9932 ± 41	$1770\times$
ProxTV-TS	CPU	207.8 ± 7.5	430.7 ± 7.9	$37\times$
ProxTV-PN	CPU	257.3 ± 5.8	447.0 ± 9.6	$41\times$
Ours-PN	Titan X	9.0 ± 0.7	17.7 ± 2.4	$1.5\times$
Ours-PN	A6000	10.0 ± 1.5	7.3 ± 6.0	$1\times$

Results

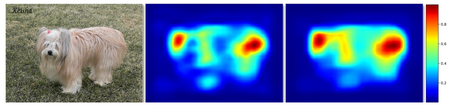
Effect of TV Layer on Features



CIFAR10 Image Classification (Out of Domain Generalization)

	Noise Type			Blur Type			
Arch.	Gaussian	Shot	Impulse	Glass	Defocus	Motion	Zoom
AllConv	36.2 ± 1.4	48.9 ± 1.5	$\mathbf{46.6} \pm 3.1$	77.9 ± 0.7	46.7 ± 2.3	72.8 ± 0.5	71.9 ± 1.1
TV-Smooth	$\mathbf{42.7} \pm 2.4$	$\mathbf{55.0} \pm 1.4$	44.8 ± 3.2	79.4 ± 0.7	49.2 ± 2.5	74.5 ± 0.8	$\mathbf{74.3} \pm 1.2$
TV-Sharp	40.6 ± 1.4	53.1 ± 1.5	46.5 ± 1.0	$\mathbf{79.8} \pm 0.3$	$\mathbf{50.3} \pm 0.6$	$\mathbf{75.7} \pm 0.7$	73.9 ± 0.4

Weakly Supervised Object Localization



Method	VGG-16	Inception-V3	ResNet-50
CAM-paper	60.02	63.40	63.65
CAM-repro.	60.13	63.51	64.09
CAM-repro.+TV	**60.35**	**63.80**	**65.36**

Edge Preserving Smoothing

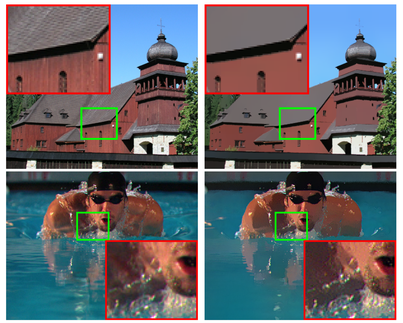
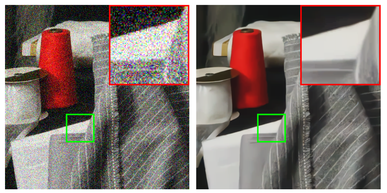


Image Denoising

Method	Kodak24		McMaster	
Noise-level σ	25	50	25	50
DnCNN	32.14	28.95	31.52	28.62
DnCNNTV	32.15	**29.09**	**32.32**	**29.35**