i-TAP meeting

홍익대학교
노승문
2021. 5. 21

Deep High-Resolution Representation Learning for Visual Recognition

Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Mingkui Tan, Xinggang Wang, Wenyu Liu, and Bin Xiao

Abstract—High-resolution representations are essential for position-sensitive vision problems, such as human pose estimation, semantic segmentation, and object detection. Existing state-of-the-art frameworks first encode the input image as a low-resolution representation through a subnetwork that is formed by connecting high-to-low resolution convolutions *in series* (e.g., ResNet, VGGNet), and then recover the high-resolution representation from the encoded low-resolution representation. Instead, our proposed network, named as High-Resolution Network (HRNet), maintains high-resolution representations through the whole process. There are two key characteristics: (i) Connect the high-to-low resolution convolution streams *in parallel*; (ii) Repeatedly exchange the information across resolutions. The benefit is that the resulting representation is semantically richer and spatially more precise. We show the superiority of the proposed HRNet in a wide range of applications, including human pose estimation, semantic segmentation, and object detection, suggesting that the HRNet is a stronger backbone for computer vision problems. All the codes are available at https://github.com/HRNet.

HRNet: for representation learning

Replacing Mobile Camera ISP with a Single Deep Learning Model

Andrey Ignatov
andrey@vision.ee.ethz.ch

Luc Van Gool
vangool@vision.ee.ethz.ch

Radu Timofte
timofter@vision.ee.ethz.ch

ETH Zurich, Switzerland

CVPR 2020

Abstract

As the popularity of mobile photography is growing constantly, lots of efforts are being invested now into building complex hand-crafted camera ISP solutions. In this work, we demonstrate that even the most sophisticated ISP pipelines can be replaced with a single end-to-end deep learning model trained without any prior knowledge about the sensor and optics used in a particular device. For this, we present PyNET, a novel pyramidal CNN architecture designed for fine-grained image restoration that implicitly learns to perform all ISP steps such as image demosaicing, denoising, white balancing, color and contrast correction, demoireing, etc. *The model is trained to convert RAW Bayer data obtained directly from mobile camera sensor into photos captured with a professional high-end DSLR camera, making the solution independent of any particular mobile ISP implementation. To validate the proposed approach on the real data, we collected a large-scale dataset consisting of 10 thousand full-resolution RAW–RGB image pairs captured in the wild with the Huawei P20 cameraphone (12.3 MP Sony Exmor IMX380 sensor) and Canon 5D Mark IV DSLR. The experiments demonstrate that the proposed solution can easily get to the level of the embedded P20's ISP pipeline that, unlike our approach, is combining the data from two (RGB + B/W) camera sensors. The dataset, pre-trained models and codes used in this paper are available on the project website:* https://people.ee.ethz.ch/~ihnatova/pynet.html

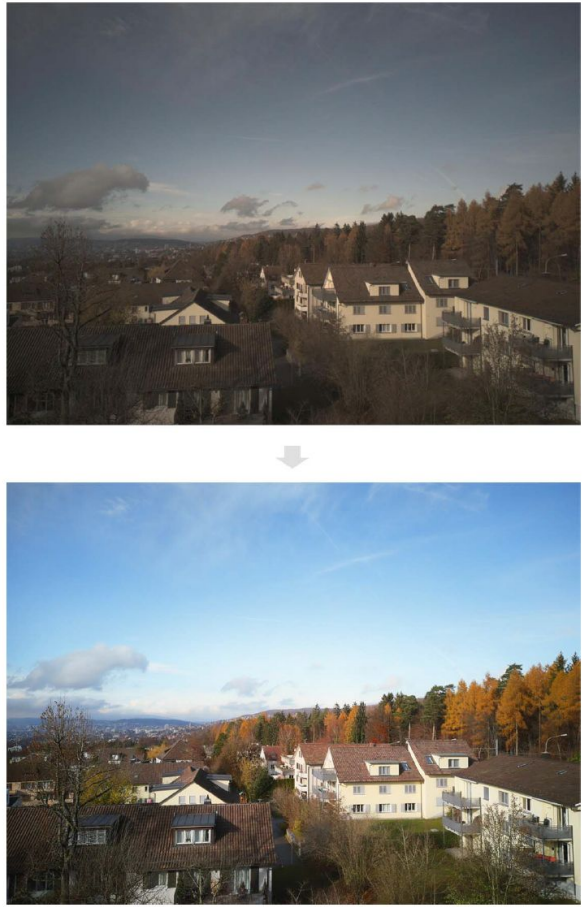
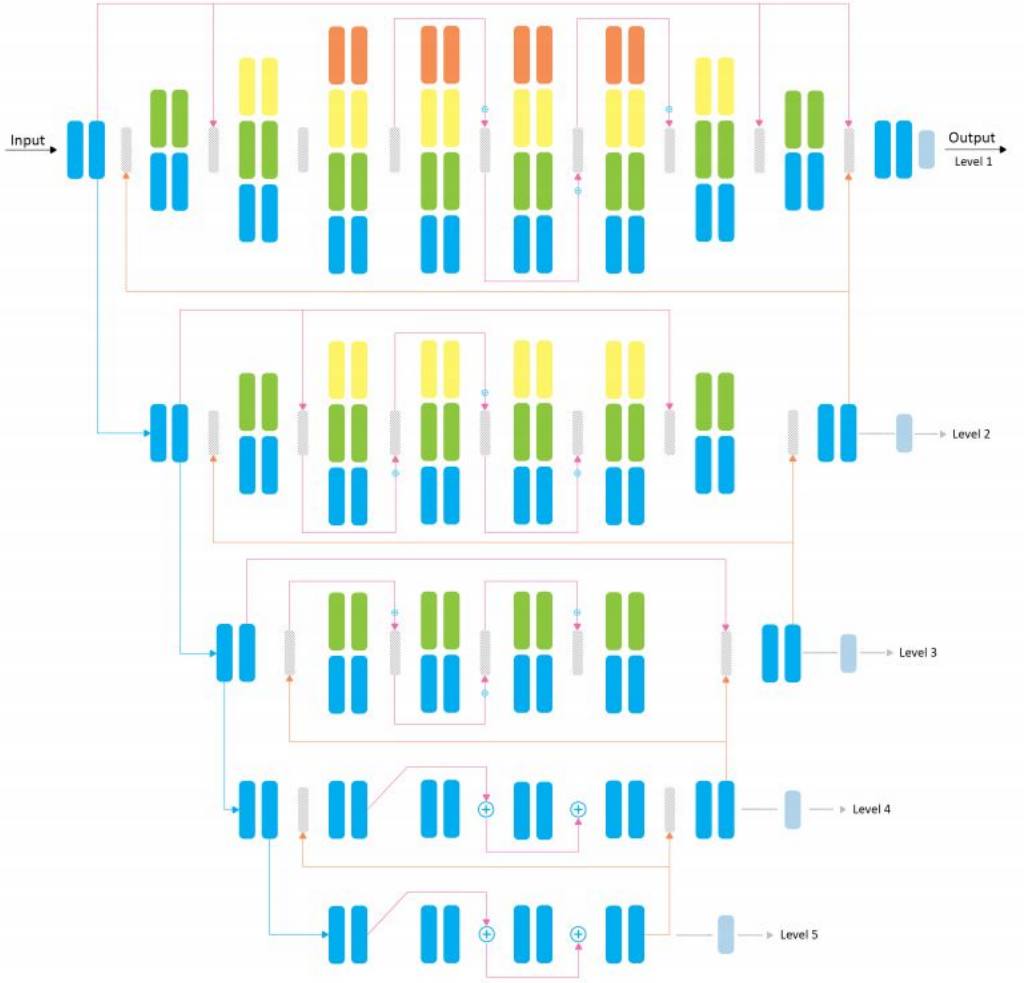
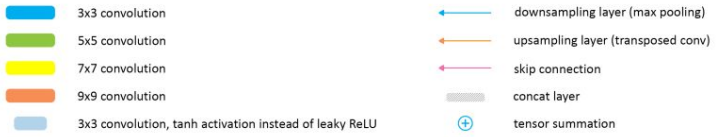


Figure 1. Huawei P20 RAW photo (visualized) and the corresponding image reconstructed with our method.

Figure 6. Visual results obtained with 7 different architectures. From left to right, top to bottom: visualized RAW photo, SRCNN [10], VDSR [27], SRGAN [31], Pix2Pix [25], U-Net [45], DPED [19], our PyNET architecture, Huawei ISP image and the target Canon photo.

Pynet


Input
Output
Level 1
Level 2
Level 3
Level 4
Level 5


3x3 convolution
5x5 convolution
7x7 convolution
9x9 convolution
3x3 convolution, tanh activation instead of leaky ReLU
downsampling layer (max pooling)
upsampling layer (transposed conv)
skip connection
concat layer
tensor summation

Levels 4-5 operate with images downscaled by a factor of 8 and 16, respectively, therefore they are mainly targeted at global color and brightness / gamma correction. These layers are trained to minimize the mean squared error (MSE) since the perceptual losses are not efficient at these scales.

Levels 2-3 are processing 2x / 4x downscaled images, and are mostly working on the global content domain. The goal of these layers is to refine the color / shape properties of various objects on the image, taking into account their semantic meaning. They are trained with a combination of the VGG-based [26] perceptual and MSE loss functions taken in the ratio of 4:1.

Level 1 is working on the original image scale and is primarily trained to perform local image corrections: texture enhancement, noise removal, local color processing, *etc.*, while using the results obtained from the lower layers. It is trained using the following loss function:

$$\mathscr{L}_{\text{Level 1}} = \mathscr{L}_{\text{VGG}} + 0.75 \cdot \mathscr{L}_{\text{SSIM}} + 0.05 \cdot \mathscr{L}_{\text{MSE}},$$

where the value of each loss is normalized to 1. The structural similarity (SSIM) loss [59] is used here to increase the

Perceptual Losses for Real-Time Style Transfer and Super-Resolution

Justin Johnson, Alexandre Alahi, Li Fei-Fei
{jcjohns, alahi, feifeili}@cs.stanford.edu

Department of Computer Science, Stanford University

PyNET-CA: Enhanced PyNET with Channel Attention for End-to-End Mobile Image Signal Processing

Byung-Hoon Kim[1], Joonyoung Song[1], Jong Chul Ye[1(✉)], and JaeHyun Baek[2]

[1] Korea Advanced Institute of Science and Technology, Daejeon, South Korea
{egyptdj,songjy18,jong.ye}@kaist.ac.kr
[2] Amazon Web Services, Seoul, South Korea
jakemraz100@gmail.com

~~Arxiv 2021.4~~
ECCV 2020 (2020.8)

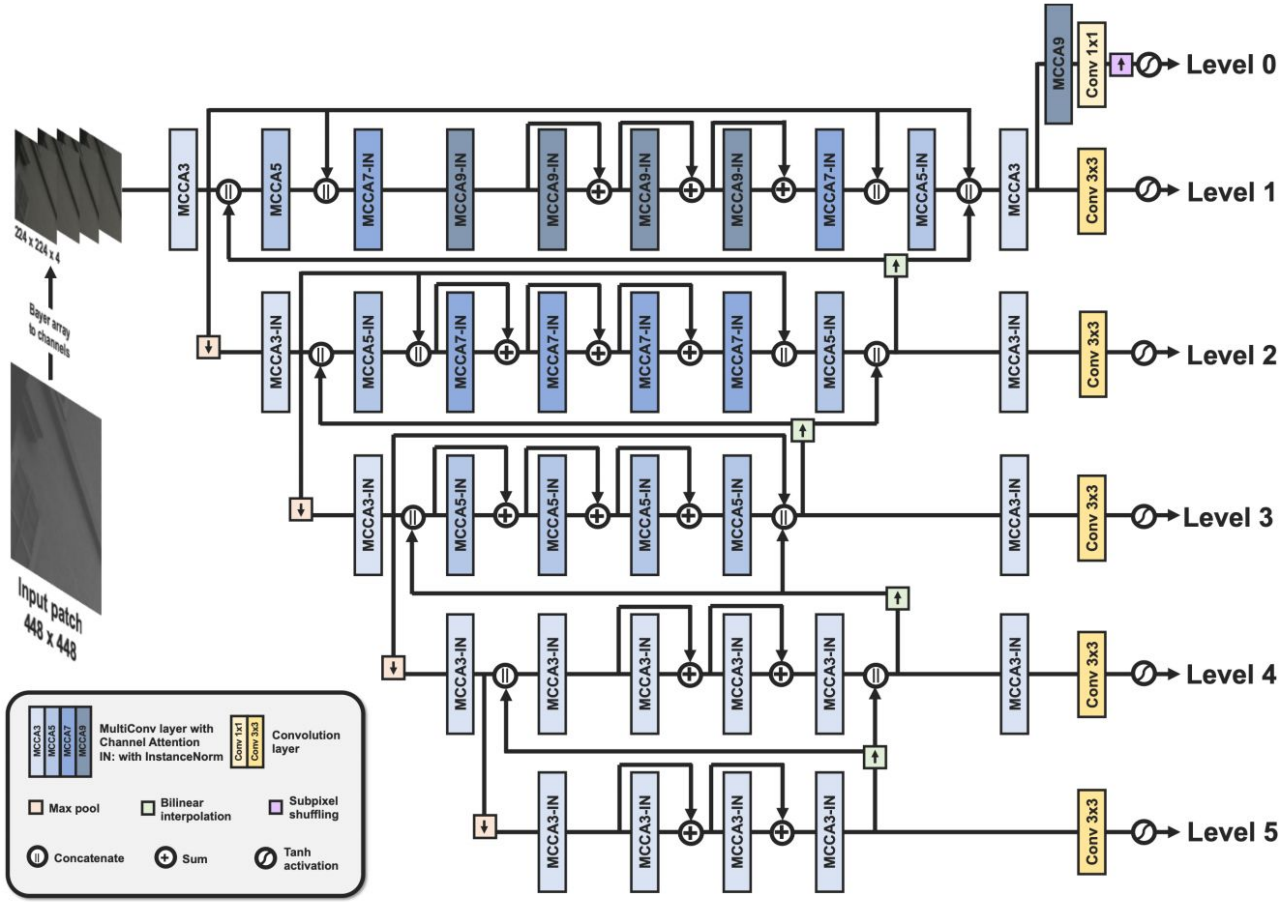


Fig. 3. Schematic illustration of the PyNET-CA model.

Progressive training The PyNET-CA model is progressively trained from the level with the lowest resolution. The target image $\mathbf{I}_{rgb}$ is downsampled with bilinear interpolation to match the resolution of the reconstructed images $\hat{\mathbf{I}}^{k}_{rgb}$ at each level. The loss function used for training the PyNET-CA is a linear combination of the mean squared error (MSE) loss, the perceptual loss using one VGG layer `relu5_4`, and the multi-scale structural similarity index measure (MS-SSIM) loss,

$$\mathcal{L} = \lambda_1 \mathcal{L}_{\texttt{MSE}} + \lambda_2 \mathcal{L}_{\texttt{VGG}} + \lambda_3 \mathcal{L}_{\texttt{MS-SSIM}}.$$


28.AUGUST
GLASGOW, UK
AIM 2020
Advances in Image Manipulation workshop
and challenges on image and video manipulation
in conjunction with ECCV 2020
CALL FOR PAPERS
IMPORTANT DATES
NTIRE2020
AIM2019

AIM 2020 image challenges

- Relighting: Track 1 Any to one relighting started!
- Relighting: Track 2 Illumination Settings estimation started!
- Relighting: Track 3 Any to any relighting started!
- Extreme Inpainting: Track 1 Classic started!
- Extreme Inpainting: Track 2 Semantic Guidance started!
- Learned Smartphone ISP (aka RAW to RGB mapping): Track 1 Fidelity started!
- Learned smartphone ISP (aka RAW to RGB mapping): Track 2 Perceptual started!
- Rendering Realistic Bokeh: Track 1 on CPU started!
- Rendering Realistic Bokeh: Track 2 on smartphone GPU started!
- Real Super-Resolution: Track 1 Upscaling x2 started!
- Real Super-Resolution: Track 2 Upscaling x3 started!
- Real Super-Resolution: Track 3 Upscaling x4 started!
- Efficient Super-Resolution started!

One needs to check the corresponding Codalab competion(s) in order to learn more about and to register to access the data and participate in the challenge(s) of interest.

AIM 2020 Learned Smartphone ISP Challenge - Track 2: Perceptual

Organized by Radu - Current server time: May 18, 2021, 1:40 a.m. UTC

Previous	▶ Current	End
Testing	Development	Competition Ends
July 10, 2020, 11:59 p.m. UTC	May 9, 2020, 11:59 p.m. UTC	July 17, 2020, 11:59 p.m. UTC

Learn the Details | Phases | Participate | Results | Forums

Development | Testing

Phase description

None

Max submissions per day: 10

Max submissions total: 20

Download CSV

LearnedISP								
#	User	Entries	Date of Last Entry	PSNR ▲	SSIM ▲	Runtime per image [s] ▲	CPU [1] / GPU [0] ▲	Extra Data [1] / No Extra Data [0] ▲
1	itb202d	1	07/17/20	21.954 (3)	0.7788 (4)	0.0150 (7)	0.0 (2)	0.0 (1)
2	mo_ming	22	07/17/20	22.247 (1)	0.7893 (1)	0.0100 (8)	1.0 (1)	0.0 (1)
3	egyptdj	12	07/17/20	21.709 (4)	0.7868 (2)	0.1000 (3)	0.0 (2)	0.0 (1)

AIM 2020 Challenge on Learned Image Signal Processing Pipeline

Andrey Ignatov, Radu Timofte, Zhilu Zhang, Ming Liu, Haolin Wang, Wangmeng Zuo, Jiawei Zhang, Ruimao Zhang, Zhanglin Peng, Sijie Ren, Linhui Dai, Xiaohong Liu, Chengqi Li, Jun Chen, Yuichi Ito, Bhavya Vasudeva, Puneesh Deora, Umapada Pal, Zhenyu Guo, Yu Zhu, Tian Liang, Chenghua Li, Cong Leng, Zhihong Pan, Baopu Li, Byung-Hoon Kim, Joonyoung Song, Jong Chul Ye, JaeHyun Baek, Magauiya Zhussip, Yeskendir Koishekenov, Hwechul Cho Ye, Xin Liu, Xueying Hu, Jun Jiang, Jinwei Gu, Kai Li, Pengliang Tan, and Bingxin Hou *

Abstract. This paper reviews the second AIM learned ISP challenge and provides the description of the proposed solutions and results. The participating teams were solving a real-world RAW-to-RGB mapping problem, where to goal was to map the original low-quality RAW images captured by the Huawei P20 device to the same photos obtained with the Canon 5D DSLR camera. The considered task embraced a number of complex computer vision subtasks, such as image demosaicing, denoising, white balancing, color and contrast correction, demoireing, etc. The target metric used in this challenge combined fidelity scores (PSNR and SSIM) with solutions' perceptual results measured in a user study. The proposed solutions significantly improved the baseline results, defining the state-of-the-art for practical image signal processing pipeline modeling.

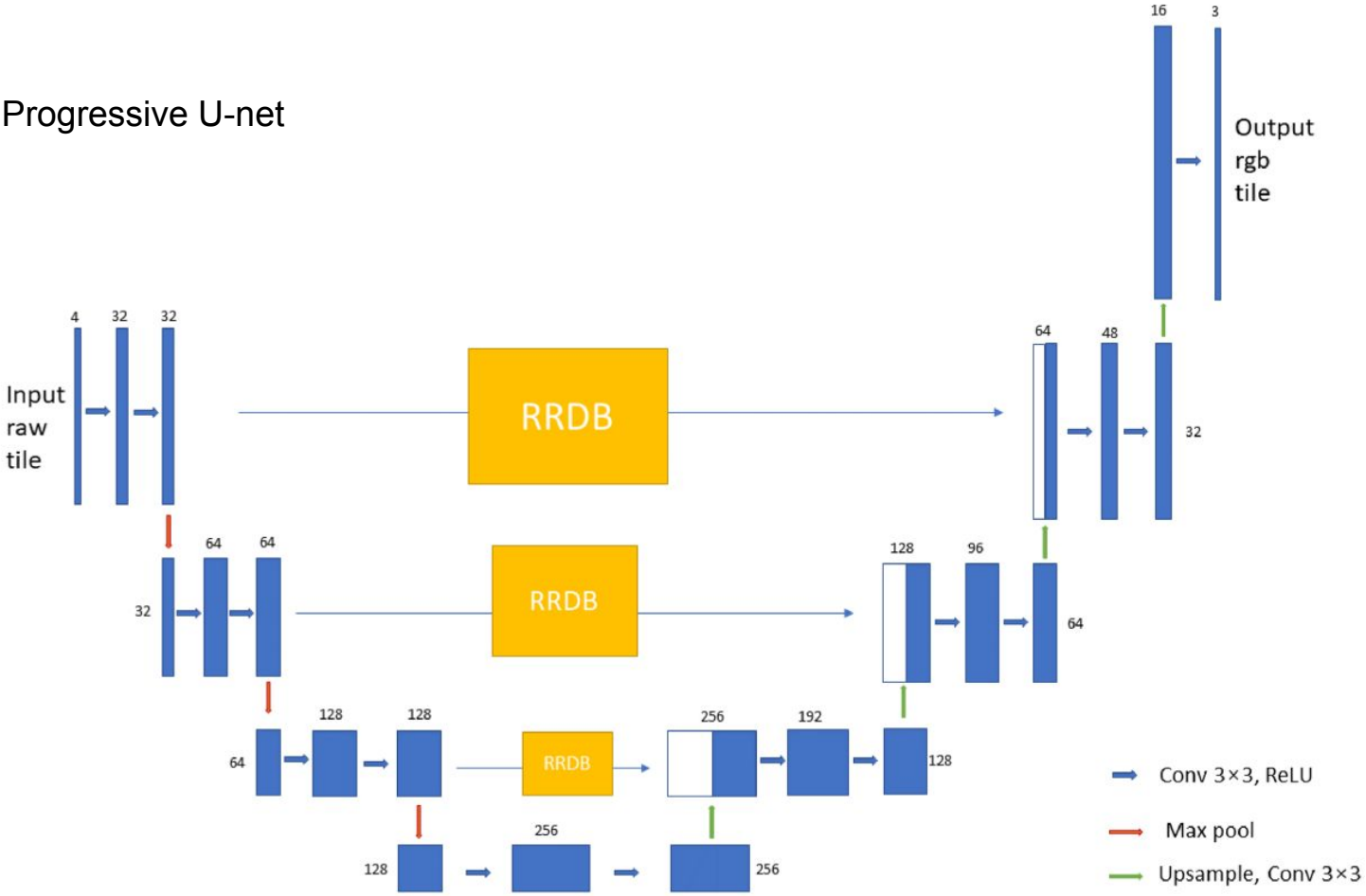
		Factsheet Info			Track 1: Fidelity		Track 2: Perceptual		
Team	Author	Framework	Hardware, GPU	Runtime, s	PSNR↑	SSIM↑	PSNR↑	SSIM↑	MOS↑
MW-ISPNet	zhangzhilu	PyTorch	4 × GeForce GTX 1080 Ti	∼1	21.91	0.7842	21.57	0.7770	**4.7**
MacAI	itb202d	PyTorch	2 × GeForce RTX 2080 Ti	0.83	21.86	0.7807	21.86	0.7807	4.5
Vermilion Vision	wataridori2010	TensorFlow	GeForce RTX 2080 Ti	0.062	21.40	0.7834	21.40	0.7834	4.2
Eureka	bhavya_vasudeva	Keras (TF)	4 × GeForce GTX 1080 Ti	0.078	21.18	0.7794	21.18	0.7794	4.1
Airia_CG	mo_ming	PyTorch	8 × Nvidia TITAN Xp	-	**22.26**	**0.7913**	21.01	0.7691	4
Baidu	zhihongp	PyTorch	GeForce GTX 2080 Ti	1.2	21.91	0.7829	**21.91**	0.7829	4
skyb	egyptdj	PyTorch	Nvidia Tesla V100	0.2	21.93	0.7865	21.73	**0.7891**	3.8
STAIR	dark_1im1ess	PyTorch	4 × GeForce GTX 1080	0.59	21.57	0.7846	21.57	0.7846	3.5
Sensebrainer	acehu	PyTorch	4 × Nvidia Tesla V100	0.075	21.14	0.7729	21.14	0.7729	3.2
bupt-mtc206	TheClearwind	PyTorch	GeForce GTX 1080 Ti	0.03	20.19	0.7622	20.19	0.7622	2.4
BingSoda	houbingxin	PyTorch	Nvidia TITAN RTX	0.04	20.14	0.7438	20.14	0.7438	2.2

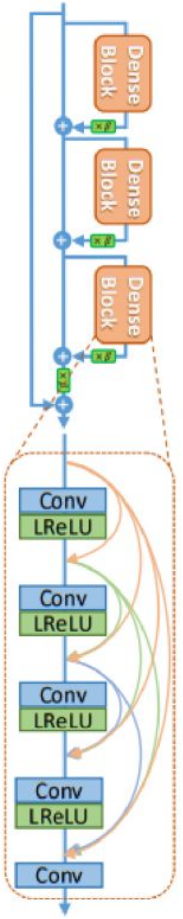
4.5 Airia_CG

Team Airia_CG used two different approaches in this challenge. In the Perceptual track, it proposed a Progressive U-Net (PU-Net) architecture (Fig. 6, bottom) that is essentially a U-Net model augmented with Contrast-Aware Channel Attention modules [13], switchable normalization layers [29] and pixel shuffle layers for upsampling the images. The authors have additionally cleaned the provided ZRR dataset by removing all blurred photos, and used the obtained image subset for training the model.

In the fidelity track, the authors used an ensemble of six different models: the PU-Net model described above, PyNET [24] and four EEDNets [44] (Fig. 6, top) with slightly different architectures: EEDNetv1 uses simple copy and crop rather than RRDB modules compared to EEDNetv2, and EEDNetv4 adds a contrast-aware channel attention module after the RRDB module. The considered ensemble was able to improve the PSNR on the validation dataset by 0.61dB compared to its best single model.

Progressive U-net





European Conference on Computer Vision
ECCV 2020: Computer Vision – ECCV 2020 Workshops pp 171-184 | Cite as

EEDNet: Enhanced Encoder-Decoder Network for AutoISP

Authors | Authors and affiliations

Yu Zhu, Zhenyu Guo, Tian Liang, Xiangyu He, Chenghua Li, Cong Leng, Bo Jiang, Yifan Zhang, Jian Cheng

Conference paper
First Online: 30 January 2021

Part of the Lecture Notes in Computer Science book series (LNCS, volume 12537)

4.6 Baidu Research Vision

Baidu team based their solution on a mosaic-adaptive dense residual network (Fig. 7). A mosaic stride convolution layer at the beginning of the model is used to extract mosaic-adaptive shallow features. The model is enhanced with additional channel-attention modules as in RCAN [43]. The network was trained with a combination of the L_1 and SSIM loss functions, a self-assemble strategy was additionally used for generating the final results.

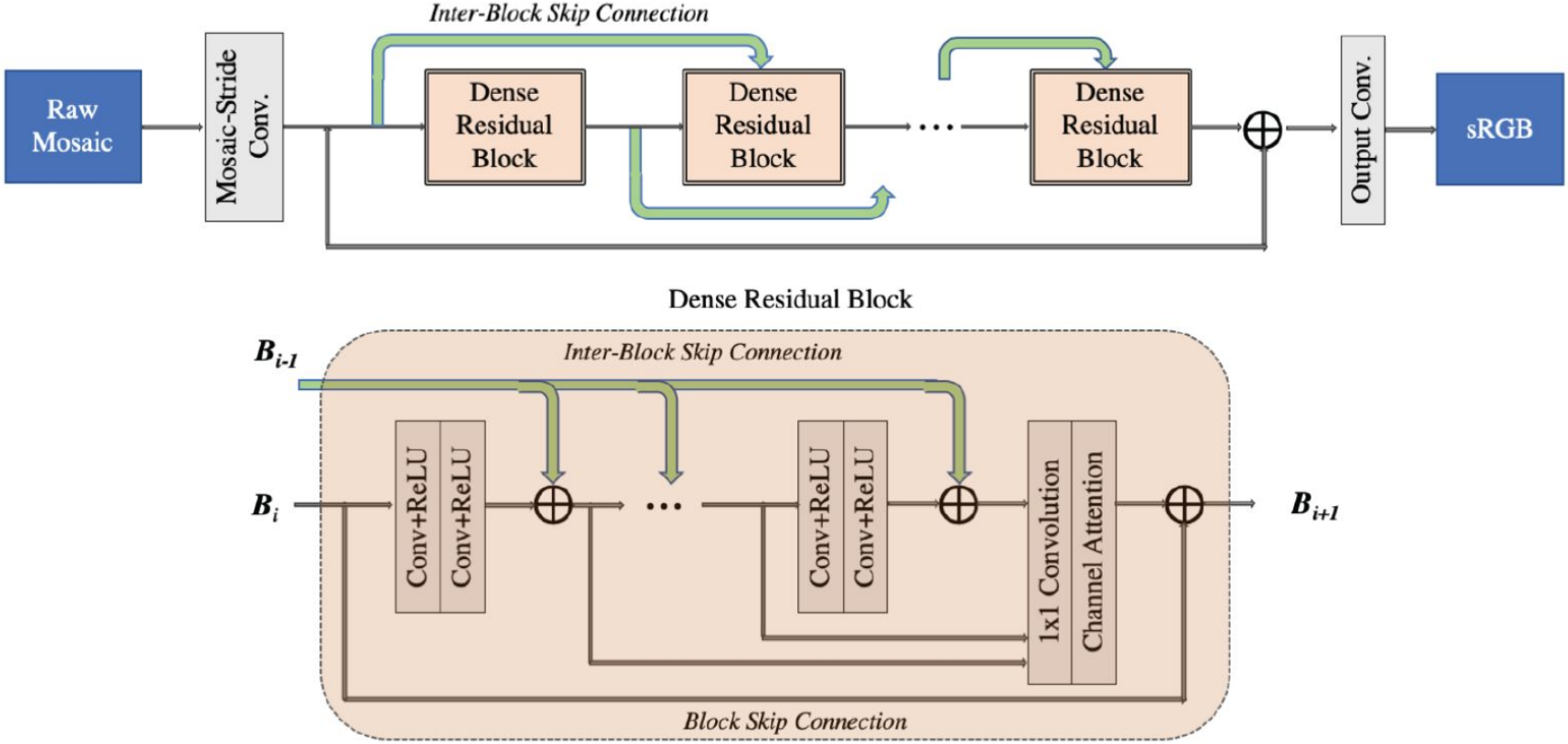


Fig. 7. Mosaic-adaptive dense residual network proposed by Baidu.

4.7 Skyb

Skyb presented a PyNet-CA model [25] (Fig. 8) that adds several enhancements on top of the standard PyNET [24] architecture. In particular, the authors added RCAN style channel attention [43] on top of the outputs from multi-convolutional layers. Besides that, some instance normalization ops were removed, an additional multi-convolutional layer was used for upscaling the final image, and a different one-cycle learning rate policy [35] was used for training each level of the model. The network was trained with a combination of the MSE, VGG-based and SSIM loss function taken in different combinations depending on the track and PyNET level, a self-ensemble strategy was additionally applied for producing the final outputs.

11.October Montreal, Canada

AIM 2021

Advances in Image Manipulation workshop

and challenges on image and video manipulation

in conjunction with ICCV 2021

https://data.vision.ee.ethz.ch/cvl/aim21/

CVPR VIRTUAL JUNE 19-25

HOME ORGANIZERS SPONSORS SUBMISSION ATTEND PROGRAM REGISTRATION

ABOUT

CVPR is the premier annual computer vision event comprising the main conference and several co-located workshops and short courses. With its high quality and low cost, it provides an exceptional value for students, academics and industry researchers.

Due to the continuing impact and

PyNet Run

aiff22 / PyNET-PyTorch

Watch 13 | Star 181 | Fork 35

Code | Issues 4 | Pull requests | Actions | Projects | Wiki | Security | Insights

master | 1 branch | 0 tags | Go to file | Add file | Code

aiff22 Google Drive quota — b05d817 on 7 Mar 2020 — 8 commits

models/original	Original commit	15 months ago
raw_images	Original commit	15 months ago
results/full-resolution	Original commit	15 months ago
LICENSE.md	Original commit	15 months ago
README.md	Google Drive quota	15 months ago
dng_to_png.py	Original commit	15 months ago
evaluate_accuracy.py	Original commit	15 months ago
load_data.py	Fixing LoadVisualData	15 months ago

About

Generating RGB photos from RAW image files with PyNET (PyTorch)

www.vision.ee.ethz.ch/~ihnatova/py...

photos | mobile | computer-vision | deep-learning | camera | image-reconstruction | photography | image-processing | isp | raw | image-to-image-translation | image-enhancement | pynet | pytorch-implementation | raw-to-rgb

Readme

View license

Pretrained weights: 182Mb

3. First steps

- Download the pre-trained PyNET model *(PSNR: 21.17, MS-SSIM: 0.8623)* and put it into `models/original/` folder.
- Download Zurich RAW to RGB mapping dataset and extract it into `raw_images/` folder.

  This folder should contain three subfolders: `train/`, `test/` and `full_resolution/`

  Please note that Google Drive has a quota limiting the number of downloads per day. To avoid it, you can login to your Google account and press "Add to My Drive" button instead of a direct download. Please check this issue for more information.

Dataset: 23.8Gb

Data Format

- Hynix
  - Raw: imgs/train_folder/msc/ : 256*256*3
  - Rgb: imgs/train_folder/gt/ : 256*256*3
- PyNet
  - Raw: raw_images/train/huawei_raw/ : 448*448
  - Rgb: raw_images/train/canon/ : 448*448*3

Fixes

- scipy .misc -> PIL
  - imread
  - imresize
- train_model
  - Parameters: TRAIN_SIZE(46839->800), etc
  - Comment out visualization
  - Send tensors to float
- To run
  - Use batch size 1 for level 2 (GTX 1080, 8gb)

Run: In progress

```
The following parameters will be applied for CNN training:
Training level: 1
Batch size: 1
Learning rate: 5e-05
Training epochs: 25
Restore epoch: 16
Path to the dataset: raw_images/
CUDA visible devices: 1
CUDA Device Name: GeForce GTX 1080
Epoch 0, mse: 0.0918, psnr: 10.9960, vgg: 0.3188, ms-ssim: nan
Epoch 1, mse: 0.0335, psnr: 16.1024, vgg: 0.2437, ms-ssim: 0.7608
Epoch 2, mse: 0.0293, psnr: 16.4678, vgg: 0.2300, ms-ssim: 0.7587
Epoch 3, mse: 0.0279, psnr: 16.5064, vgg: 0.2251, ms-ssim: 0.7465
Epoch 4, mse: 0.0261, psnr: 16.8856, vgg: 0.2270, ms-ssim: 0.7657
Epoch 5, mse: 0.0307, psnr: 15.8042, vgg: 0.2476, ms-ssim: 0.7221
Epoch 6, mse: 0.0276, psnr: 16.5112, vgg: 0.2380, ms-ssim: 0.7331
Epoch 7, mse: 0.0261, psnr: 16.7333, vgg: 0.2362, ms-ssim: 0.7538
Epoch 8, mse: 0.0266, psnr: 16.5761, vgg: 0.2433, ms-ssim: 0.7353
Epoch 9, mse: 0.0262, psnr: 16.5609, vgg: 0.2338, ms-ssim: nan
Epoch 10, mse: 0.0277, psnr: 16.3059, vgg: 0.2225, ms-ssim: nan
Epoch 11, mse: 0.0268, psnr: 16.4939, vgg: 0.2310, ms-ssim: nan
Epoch 12, mse: 0.0265, psnr: 16.5741, vgg: 0.2311, ms-ssim: 0.7349
Epoch 13, mse: 0.0259, psnr: 16.6673, vgg: 0.2284, ms-ssim: 0.7391
Epoch 14, mse: 0.0262, psnr: 16.4734, vgg: 0.2292, ms-ssim: nan
Epoch 15, mse: 0.0264, psnr: 16.4502, vgg: 0.2288, ms-ssim: 0.7176
Epoch 16, mse: 0.0254, psnr: 16.6640, vgg: 0.2209, ms-ssim: 0.7302
Epoch 17, mse: 0.0252, psnr: 16.7663, vgg: 0.2168, ms-ssim: 0.7412
Epoch 18, mse: 0.0263, psnr: 16.4252, vgg: 0.2277, ms-ssim: nan
Epoch 19, mse: 0.0249, psnr: 16.7048, vgg: 0.2242, ms-ssim: 0.7330
Epoch 20, mse: 0.0240, psnr: 16.9826, vgg: 0.2164, ms-ssim: 0.7561
Epoch 21, mse: 0.0241, psnr: 16.8932, vgg: 0.2109, ms-ssim: 0.7464
Epoch 22, mse: 0.0244, psnr: 16.8212, vgg: 0.2202, ms-ssim: 0.7401
Epoch 23, mse: 0.0253, psnr: 16.6675, vgg: 0.2209, ms-ssim: nan
Epoch 24, mse: 0.0245, psnr: 16.8318, vgg: 0.2134, ms-ssim: 0.7398
```

```
The following parameters will be applied for CNN training:
Training level: 0
Batch size: 1
Learning rate: 5e-05
Training epochs: 50
Restore epoch: 24
Path to the dataset: raw_images/
CUDA visible devices: 1
CUDA Device Name: GeForce GTX 1080
Epoch 0, mse: 0.0869, psnr: 11.1659, vgg: 0.1531, ms-ssim: 0.4675
Epoch 1, mse: 0.0273, psnr: 16.1914, vgg: 0.1312, ms-ssim: 0.6878
Epoch 2, mse: 0.0274, psnr: 16.1434, vgg: 0.1300, ms-ssim: nan
Epoch 3, mse: 0.0273, psnr: 16.2140, vgg: 0.1291, ms-ssim: 0.6876
Epoch 4, mse: 0.0264, psnr: 16.3893, vgg: 0.1299, ms-ssim: 0.6921
Epoch 5, mse: 0.0286, psnr: 15.9877, vgg: 0.1360, ms-ssim: nan
Epoch 6, mse: 0.0273, psnr: 16.2517, vgg: 0.1299, ms-ssim: 0.6835
Epoch 7, mse: 0.0269, psnr: 16.3053, vgg: 0.1329, ms-ssim: 0.6842
Epoch 8, mse: 0.0287, psnr: 16.0411, vgg: 0.1332, ms-ssim: 0.6648
Epoch 9, mse: 0.0268, psnr: 16.3512, vgg: 0.1323, ms-ssim: 0.6926
Epoch 10, mse: 0.0276, psnr: 16.2422, vgg: 0.1333, ms-ssim: 0.6795
Epoch 11, mse: 0.0268, psnr: 16.3752, vgg: 0.1309, ms-ssim: 0.6882
Epoch 12, mse: 0.0270, psnr: 16.3541, vgg: 0.1341, ms-ssim: 0.6807
Epoch 13, mse: 0.0277, psnr: 16.1887, vgg: 0.1351, ms-ssim: 0.6789
Epoch 14, mse: 0.0265, psnr: 16.4117, vgg: 0.1339, ms-ssim: 0.6901
Epoch 15, mse: 0.0268, psnr: 16.3685, vgg: 0.1337, ms-ssim: 0.6841
Epoch 16, mse: 0.0269, psnr: 16.3665, vgg: 0.1354, ms-ssim: 0.6894
Epoch 17, mse: 0.0273, psnr: 16.2877, vgg: 0.1338, ms-ssim: 0.6850
Epoch 18, mse: 0.0269, psnr: 16.3631, vgg: 0.1330, ms-ssim: 0.6899
Epoch 19, mse: 0.0265, psnr: 16.4263, vgg: 0.1315, ms-ssim: 0.6953
Epoch 20, mse: 0.0266, psnr: 16.4243, vgg: 0.1339, ms-ssim: 0.6889
Epoch 21, mse: 0.0269, psnr: 16.3725, vgg: 0.1328, ms-ssim: 0.6901
Epoch 22, mse: 0.0275, psnr: 16.2718, vgg: 0.1361, ms-ssim: 0.6828
Epoch 23, mse: 0.0268, psnr: 16.3987, vgg: 0.1358, ms-ssim: 0.6880
Epoch 24, mse: 0.0271, psnr: 16.3347, vgg: 0.1338, ms-ssim: 0.6864
Epoch 25, mse: 0.0270, psnr: 16.3539, vgg: 0.1349, ms-ssim: 0.6891
Epoch 26, mse: 0.0276, psnr: 16.2414, vgg: 0.1373, ms-ssim: 0.6837
Epoch 27, mse: 0.0274, psnr: 16.2920, vgg: 0.1364, ms-ssim: 0.6812
Epoch 28, mse: 0.0274, psnr: 16.3216, vgg: 0.1348, ms-ssim: 0.6821
Epoch 29, mse: 0.0274, psnr: 16.2749, vgg: 0.1352, ms-ssim: 0.6847
Epoch 30, mse: 0.0276, psnr: 16.2383, vgg: 0.1351, ms-ssim: 0.6878
Epoch 31, mse: 0.0271, psnr: 16.3459, vgg: 0.1355, ms-ssim: 0.6848
Epoch 32, mse: 0.0276, psnr: 16.2629, vgg: 0.1357, ms-ssim: 0.6844
Epoch 33, mse: 0.0267, psnr: 16.4229, vgg: 0.1327, ms-ssim: 0.6937
```

Attention Compression

An Survey of Neural Network Compression

James T. O' Neill
Department of Computer Science
University of Liverpool
Liverpool, England, L69 3BX
james.o-neill@liverpool.ac.uk

2.3 Weight Sharing in Large Architectures

Applications in Transformers Dehghani et al. (2018) propose Universal Transformers (UT) to combine the benefits of recurrent neural networks ((RNNs) Rumelhart et al., 1985; Hochreiter and Schmidhuber, 1997) (recurrent inductive bias) with Transformers (Vaswani

4.2 Applications of Tensor Decomposition to Self-Attention and Recurrent Layers

4.2.1 Block-Term Tensor Decomposition (BTD)

Block-Term Tensor Decomposition ((BTD) De Lathauwer, 2008) combines CP decomposition and Tucker decomposition. Consider an $n-th$ order tensor as $\mathcal{A} \in \mathbb{R}^{A_1 x \ldots x A_n}$ that can be decomposed into N block terms and each block consist of k elements between a core tensor $\mathcal{G}_n \in \mathbb{R}^{G_1 \times \ldots G_d}$ and d and factor matrices $\mathcal{C}_n^{(k)} \in \mathbb{R}^{A_k \times G_k}$ along the k-th dimension where $n \in [1, N]$ and $k \in [1, d]$ (De Lathauwer, 2008). BTD can then be defined as,

$$\mathcal{A} = N \sum_{n=1} \mathcal{G}_n \otimes \mathcal{C}_n^{(1)} \otimes \mathcal{C}_n^{(2)} \ldots \mathcal{C}_n^{(d)} \tag{51}$$

The N here is the CP-rank, G_1, G_2, G_3 is the Tucker-rank and d the core-order.

5.4 Distilling Transformer-based (Non-Autoregressive) Networks

Knowledge distillation has also been applied to very large transformer networks, predominantly on BERT (Devlin et al., 2018) given its wide success in NLP. Thus, there has been a lot of recent work towards reducing the size of BERT and related models using knowledge distillation.

DistilBERT Sanh et al. (2019) achieves distillation by training a smaller BERT on very large batches using gradient accumulation, uses dynamic masking, initializes the student weights with teacher weights and removes the next sentence prediction objective. They train the smaller BERT model on the original data BERT was trained on and fine that DistilBERT is within 3% of the original BERT accuracy while being 60% faster when evaluated on the GLUE (Wang et al., 2018a) benchmark dataset.

BERT Patient Knowledge Distillation Instead of minimizing the soft probabilities between the student and teacher network outputs, Sun et al. (2019) propose to also learn from the intermediate layers of the BERT teacher network by minimizing the mean squared error between adjacent and normalized hidden states. This loss is combined with the original objective proposed by Hinton et al. (2015) which showed further improves in distilling BERT on the GLUE benchmark datasets (Wang et al., 2018a).

TinyBERT TinyBERT (Jiao et al., 2019) combines multiple Mean Squared Error (MSE) losses between embeddings, hidden layers, attention layers and prediction outputs between S

Questions?