Wide Activation for Efficient and Accurate Image Super-Resolution

Yu et al. (2018)

UVA CS 6316: Machine Learning : 2019 Fall
Course Project: Deep2Reproduce @ https://github.com/qiyanjun/deep2reproduce/tree/master/2019Fall

Machine Learning 6316 - Fall 2019
Professor Qi
Team: Skynet
Colin Price, Jacob Dineen, Kazi Ashik Islam, A S M Ahsan-Ul Haque
Motivation

• Imaging modalities are limited by resolution at some point by hardware
  – Diffraction limit in microscopy
  – Pixel count in digital cameras

• In cases where greater resolution is required than hardware allows, it is desirable to increase resolution after image acquisition
  – Can an image be reliably up-sampled using machine learning without introducing noise/false information?
  – How can model parameters and design choices impact performance through tuning?
Related Work

• Super-resolution network architecture utilizes:
  – Upsampling layers (FSRCNN showed parametric deconvolution to improve implicit $S^2$ runtime)\cite{7}
  – Deep, recursive neural networks (Generally 10 layers with 3x3 kernels, risk of overfitting)\cite{7}
  – Skip connections (Shown to preserve low-level features in images)\cite{5}

• Choice of convolutional method
  – Flattened, Group, and Depthwise Separable convolution are popular choices\cite{8,9,10}
Background

- CNN’s are state of the art and are proven to upsample\cite{1}
- Applications include security, medical, and satellite imaging\cite{2}
- Early Single Image Super Resolution (SISR) use shallow CNNs (3-5 layers)\cite{3}
  - Deep CNNs with low-level feature preserving skip-layers have been shown to be superior
  - These include SRDenseNet, RDN, and MemNet\cite{4,5,6}
- In SISR, topics of interest include activation width, normalization methods, and convolutional kernels
Main Claims:

- Non-linear Relu activations stop info propagation from shallow layers to deeper layers
- Expanding features before nonlinear activations improves SISR performance over architecture overhauls due to an increased likelihood of low level (super resolution) features flowing through network toward final layer
  - Wide activation - efficient ways to expand features before nonlinear activations
  - Using CNNs for mapping low resolution images to their high res. counterparts
To improve upon SoTA SISR performance by introducing this concept of wideness/wider activations to be computed by expanding features before activation.

Achieving higher resolution mappings without increasing model complexity (Deep NN parameters) - Real time processing - Need to keep computational overhead/parameters down.
3.4 Network Structure

Left Top: Overview of results by SotA SISR models in 2018 [12]
Left Bottom: Yu et al. proposal to simplify existing SISR network architectures [13] - This is the network we will be reproducing.
Right Top: Early work on SISR [11]
Novelty: Wider feature space before activation layer. Low level information isn’t lost as network is propagated through.

Novelty: Wider feature space before activation layer than WDSR-A + low-rank convolution stack.
Proposed Solution #1

Wide Activation: WDSR-A

a. Slim identity mapping pathway with wider channels before activation in each residual block.
   i. Wide activation - efficient ways to expand features before nonlinear activations. Need quick upsampling techniques for real time processing.
   ii. Why? Existing architectures were over-parameterized.
   iii. What methods were carried over? Skip Connections are essential. Batchnorm is neglected as accuracy was too sensitive.
Proposed Solution #2

Efficient Wider Activation: WDSR-B

a. Expands on WDSR-A.

b. Tried: additional feature expansion via group convolution and depthwise separable convolution.
   i. Group convolution: Convolving over a portion of the input channels and concatenating.
   ii. Depthwise separable convolution: Each channel is kept separate when convolving + a 1x1 spatial filter.

c. Produced low-rank convolution coupled with even wider activation
   i. Results suggested this decayed feature activations, backing hypothesis.

d. Wider activation > baselines, given different parameter budgets

e. Loosely inspired by Inverted Residuals (A parameter efficient convolution method)
Data Summary

- Model trained on DIV2K (DIVerse 2K Resolution images) dataset
- Default split of DIV2K dataset
  - 800 training images
  - 100 validation images
  - 100 testing images (not publicly available)
- Authors used 800 images for training, 10 images for validation during training
- Trained models evaluated on 100 validation images

DIV2K: [https://data.vision.ee.ethz.ch/cvl/DIV2K/](https://data.vision.ee.ethz.ch/cvl/DIV2K/)
Implementation

- Cropped 96x96 RGB input patches and bicubic downsampled image (both from HR image) used as training output-input pair
- Training data augmented with random horizontal flips and rotations
- Mean RGB values of training images subtracted from the input images
- PSNR (Peak Signal-to-Noise Ratio) is used as metric for validation
- ADAM optimizer used
- Batch size set to 16, learning rate initialized with maximum convergent value, halved every 2x10^5 iterations

REF: https://github.com/JiahuiYu/wdsr_ntire2018
Experimental Results

**Efficiency** and **Accuracy** comparison in terms of **NO. of parameters** and validation **PSNR** respectively, among baseline model EDSR and proposed models (with same NO. of residual blocks)

<table>
<thead>
<tr>
<th>Residual Blocks</th>
<th>1</th>
<th>3</th>
<th>5</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Networks</strong></td>
<td>EDSR</td>
<td>WDSR-A</td>
<td>WDSR-B</td>
<td>EDSR</td>
</tr>
<tr>
<td>Parameters</td>
<td>0.26M</td>
<td>0.08M</td>
<td>0.08M</td>
<td>0.41M</td>
</tr>
<tr>
<td>DIV2K (val) PSNR</td>
<td>33.210</td>
<td>33.323</td>
<td>33.434</td>
<td>34.043</td>
</tr>
</tbody>
</table>

*Higher PSNR value is better*
Experimental Results (Continued)

Effect of weight normalization compared to batch and no normalization (during training)
Experimental Analysis

- Weight normalization has faster convergence.
- Batch normalization is unstable during testing.
  - Possibly due to different mean, variance in test and batch-train data.
Experimental Analysis (Cont’d)

- Batch normalization is unstable during testing
  - Not because of Learning Rate
    - Tried a variety of LR
    - Unstable PSNR for every LR
Conclusion

● Proposed 2 SR networks:
  ○ \textit{WDSR-A}: Image features expanded before ReLU
  ○ \textit{WDSR-B}: Image channels expanded using 1x1 convolution

● Experimented on DIV2K dataset
  ○ Weight Normalization works better than Batch-Norm or no norm

● Achieved better accuracy, keeping the same:
  ○ Parameters
    ■ Model complexity
Paper Reconstruction

CODE ADAPTED FROM: https://github.com/krasserm/super-resolution
Data - DIV2K Dataset

- Bicubic-Downsample each image in data set
  - Produces the highest quality downsample through weighted averaging of neighboring pixels
- Compare the super-resolved down-sampled image to the original image using peak signal-to-noise ratio (PSNR) as loss function
- 800 train, 100 test, 100 validation
Training Pipeline

- Calculate loss through mapping downsampled low resolution to high resolution image (x4 upscaled)
- Training images randomly flipped, rotated, and cropped
- Loss Function: PSNR
  - Related to MSE
- Train EDSR, WSDR-A, WSDR-B models
- Adam Optimizer with learning rate schedule (PiecewiseConstantDecay)

\[
\text{MSE} = \sum_{\{\text{all } i\}} \sum_{\{\text{all } j\}} (\text{Orig}(i, j) - \text{Rcvd}(i, j))^2 \\
\text{PSNR} = 10 \log_{10} \left( \frac{255^2}{\text{MSE}} \right)
\]
def edsr(scale, num_filters=64, num_res_blocks=8, res_block_scaling=None):
    x_in = Input(shape=(None, None, 3))
    x = Lambda(normalize)(x_in)

    x = b = Conv2D(num_filters, 3, padding='same')(x)
    for i in range(num_res_blocks):
        b = res_block(b, num_filters, res_block_scaling)
    b = Conv2D(num_filters, 3, padding='same')(b)
    x = Add()([x, b])

    x = upsample(x, scale, num_filters)
    x = Conv2D(3, 3, padding='same')(x)

    x = Lambda(denormalize)(x)
    return Model(x_in, x, name="edsr")
def wdsr(scale, num_filters, num_res_blocks, res_block_expansion, res_block_scaling, res_block):
    x_in = Input(shape=(None, None, 3))
    x = Lambda(normalize)(x_in)

    # main branch
    m = Conv2D(num_filters, 3, padding='same')(x)
    for i in range(num_res_blocks):
        m = res_block(m, num_filters, res_block_expansion, kernel_size=3, scaling=res_block_scaling)(m)
    m = Conv2D(3 * scale ** 2, 3, padding='same', name=f'conv2d_main_scale_{scale}')(m)
    m = Lambda(pixel_shuffle(scale))(m)

    # skip branch
    s = Conv2D(3 * scale ** 2, 5, padding='same', name=f'conv2d_skip_scale_{scale}')(x)
    s = Lambda(pixel_shuffle(scale))(s)

    x = Add()([m, s])
    x = Lambda(denormalize)(x)

    return Model(x_in, x, name="wdsr")
def res_block_a(x_in, num_filters, expansion, kernel_size, scaling):
    x = Conv2D(num_filters * expansion, kernel_size, padding='same', activation='relu')(x_in)
    x = Conv2D(num_filters, kernel_size, padding='same')(x)
    if scaling:
        x = Lambda(lambda t: t * scaling)(x)
    x = Add()([x_in, x])
    return x
def res_block_b(x_in, num_filters, expansion, kernel_size, scaling):
    linear = 0.8
    x = Conv2D(num_filters * expansion, 1, padding='same', activation='relu')(x_in)
    x = Conv2D(int(num_filters * linear), 1, padding='same')(x)
    x = Conv2D(num_filters, kernel_size, padding='same')(x)
    if scaling:
        x = Lambda(lambda t: t * scaling)(x)
    x = Add()([x_in, x])
    return x
Model Architecture Samples

**EDSR**

- `input_10: InputLayer`
- `lambda_96: Lambda`
- `conv2d_98: Conv2D`
- `conv2d_99: Conv2D`
- `conv2d_100: Conv2D`
- `add_30: Add`
- `conv2d_102: Conv2D`
- `lambda_3: Lambda`
- `conv2d_2_scale_2: Conv2D`
- `lambda_3: Lambda`
- `conv2d_104: Conv2D`
- `lambda_36: Lambda`
- `conv2d_10: Conv2D`
- `lambda_39: Lambda`

Redundant Convolutional stack.

**WDSRa**

- `input_9: InputLayer`
- `lambda_32: Lambda`
- `conv2d_95: Conv2D`
- `conv2d_96: Conv2D`
- `add_37: Add`
- `conv2d_main_scale_4: Conv2D`
- `lambda_33: Lambda`
- `conv2d_92: Conv2D`
- `conv2d_93: Conv2D`
- `add_38: Add`
- `lambda_35: Lambda`

Architecture the same as WDSRa. Wider channel activation before skip connection.

**WDSRb**

- `input_8: InputLayer`
- `lambda_28: Lambda`
- `conv2d_91: Conv2D`
- `conv2d_92: Conv2D`
- `add_36: Add`
- `lambda_30: Lambda`
- `conv2d_94: Conv2D`
- `conv2d_95: Conv2D`
- `add_38: Add`
- `lambda_31: Lambda`
Example Visual Results
Comparison of Models

PSNR on 10 Validation Images During Training
## Comparison of Models

<table>
<thead>
<tr>
<th>Residual Blocks</th>
<th>1</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Networks</td>
<td>EDSR</td>
<td>WDSR-A</td>
</tr>
<tr>
<td>Parameters</td>
<td>409731.0</td>
<td>92304.0</td>
</tr>
<tr>
<td>DIV2k PSNR (validation)</td>
<td>27.5</td>
<td>27.8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Residual Blocks</th>
<th>5</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Networks</td>
<td>EDSR</td>
<td>WDSR-A</td>
</tr>
<tr>
<td>Parameters</td>
<td>705155.0</td>
<td>387856.0</td>
</tr>
<tr>
<td>DIV2k PSNR (validation)</td>
<td>27.9</td>
<td>28.2</td>
</tr>
</tbody>
</table>
## Comparison of Models

<table>
<thead>
<tr>
<th>Residual Blocks</th>
<th>1</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Networks</td>
<td>EDSR</td>
<td>WDSR-A</td>
</tr>
<tr>
<td>% Parameter Increase</td>
<td>-</td>
<td>22.53%</td>
</tr>
<tr>
<td>% Val PSNR Improvement</td>
<td>-</td>
<td>1.05%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Residual Blocks</th>
<th>5</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Networks</td>
<td>EDSR</td>
<td>WDSR-A</td>
</tr>
<tr>
<td>% Parameter Increase</td>
<td>-</td>
<td>55.00%</td>
</tr>
<tr>
<td>% Val PSNR Improvement</td>
<td>-</td>
<td>1.18%</td>
</tr>
</tbody>
</table>
Key Takeaways

- Models likely trained far shorter (no reference in paper).
- WDSRa or WDSRb always outperform EDSR.
- WDSRa and B always run at a fraction of the total number of params.
- WDSRb with 8 resblocks runs @ the highest Validation PSNR with 18% of the number of trainable parameters of EDSR.
• More Results

- PSNR mostly increases as a function of model depth (# of residual blocks)
- WDSRb parameter growth is much flatter than EDSR and WDSRa, even moreso than described in paper.
• Normalization Effects

- No drastic difference between utilization of weight normalization layers against no normalization.
- Tests ran with Batchnorm proved to be unstable (PSNR ~16 throughout training)
References

2. Sharon Peled and Yehezkel Yeshurun. “Superresolution in MRI: application to human white matter fiber tract visualization by diffusion tensor imaging”.
3. Chao Dong et al. “Learning a deep convolutional network for image super-resolution”.
4. Tong Tong et al. “Image Super-Resolution Using Dense Skip Connections”.
5. Y. Zhang et al. “Residual Dense Network for Image Super-Resolution”.
Assignments

https://docs.google.com/spreadsheets/d/1CoJuoJgLHKp-m18c3TeofyVIXqRbnKhy0Aq3JYwUds8/edit#gid=1386834576