

Single Image Super-Resolution with EDSR, WDSR and SRGAN

A Tensorflow 2.x based implementation of

- Enhanced Deep Residual Networks for Single Image Super-Resolution (EDSR), winner of the NTIRE 2017 super-resolution challenge.
- Wide Activation for Efficient and Accurate Image Super-Resolution (WDSR), winner of the NTIRE 2018 super-resolution challenge (realistic tracks).
- Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network (SRGAN).

This is a complete re-write of the old Keras/Tensorflow 1.x based implementation available [here](#). Some parts are still work in progress but you can already train models as described in the papers via a high-level training API. Furthermore, you can also fine-tune EDSR and WDSR models in an SRGAN context. Training and usage examples are given in the notebooks

- [example-edsr.ipynb](#)
- [example-wdsr.ipynb](#)
- [example-srgan.ipynb](#)

A [DIV2K](#) data provider automatically downloads DIV2K training and validation images of given scale (2, 3, 4 or 8) and downgrade operator (“bicubic”, “unknown”, “mild” or “difficult”).

Important: if you want to evaluate the pre-trained models with a dataset other than DIV2K please read this comment (and replies) first.

Environment setup

Create a new conda environment with

```
1 conda env create -f environment.yml
```

and activate it with

```
1 conda activate sISR
```

Introduction

You can find an introduction to single-image super-resolution in this article. It also demonstrates how EDSR and WDSR models can be fine-tuned with SRGAN (see also this section).

Getting started

Examples in this section require following pre-trained weights for running (see also example notebooks):

Pre-trained weights

- `weights-edsr-16-x4.tar.gz`
 - EDSR x4 baseline as described in the EDSR paper: 16 residual blocks, 64 filters, 1.52M parameters.
 - PSNR on DIV2K validation set = 28.89 dB (images 801 - 900, 6 + 4 pixel border included).
- `weights-wdsr-b-32-x4.tar.gz`
 - WDSR B x4 custom model: 32 residual blocks, 32 filters, expansion factor 6, 0.62M parameters.
 - PSNR on DIV2K validation set = 28.91 dB (images 801 - 900, 6 + 4 pixel border included).
- `weights-srgan.tar.gz`
 - SRGAN as described in the SRGAN paper: 1.55M parameters, trained with VGG54 content loss.

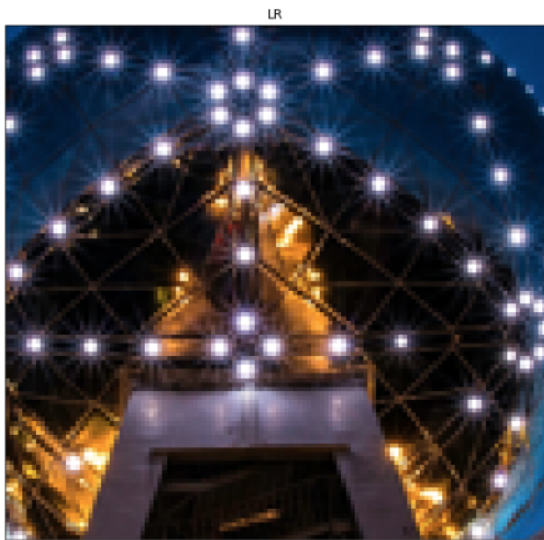
After download, extract them in the root folder of the project with

```
1 tar xvfz weights-<...>.tar.gz
```

EDSR

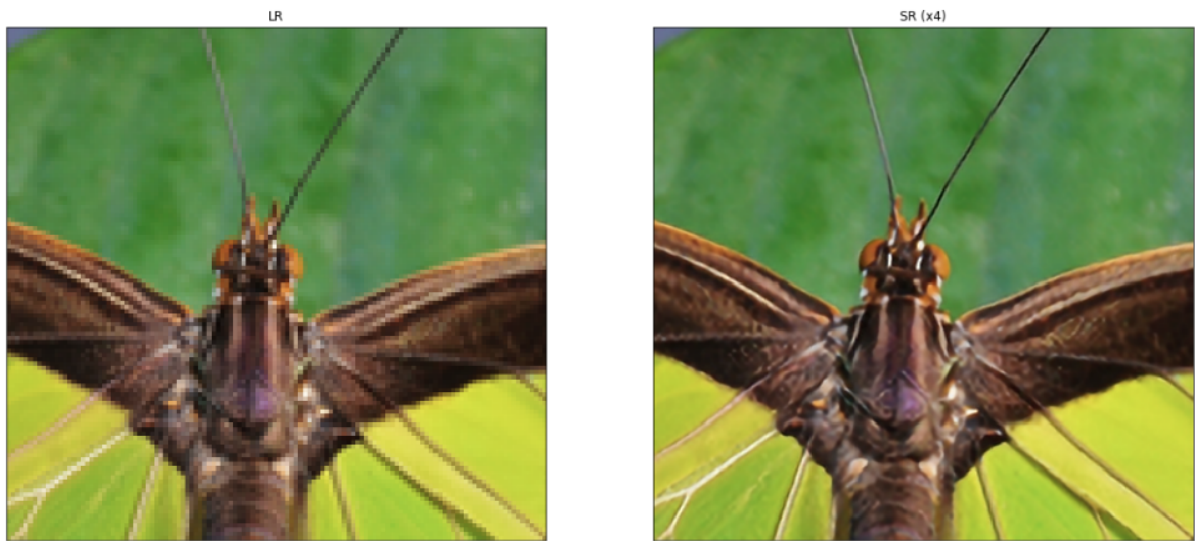
```
1 from model import resolve_single
2 from model.edsr import edsr
3
4 from utils import load_image, plot_sample
5
6 model = edsr(scale=4, num_res_blocks=16)
7 model.load_weights('weights/edsr-16-x4/weights.h5')
```

```
8
9 lr = load_image('demo/0851x4-crop.png')
10 sr = resolve_single(model, lr)
11
12 plot_sample(lr, sr)
```



WDSR

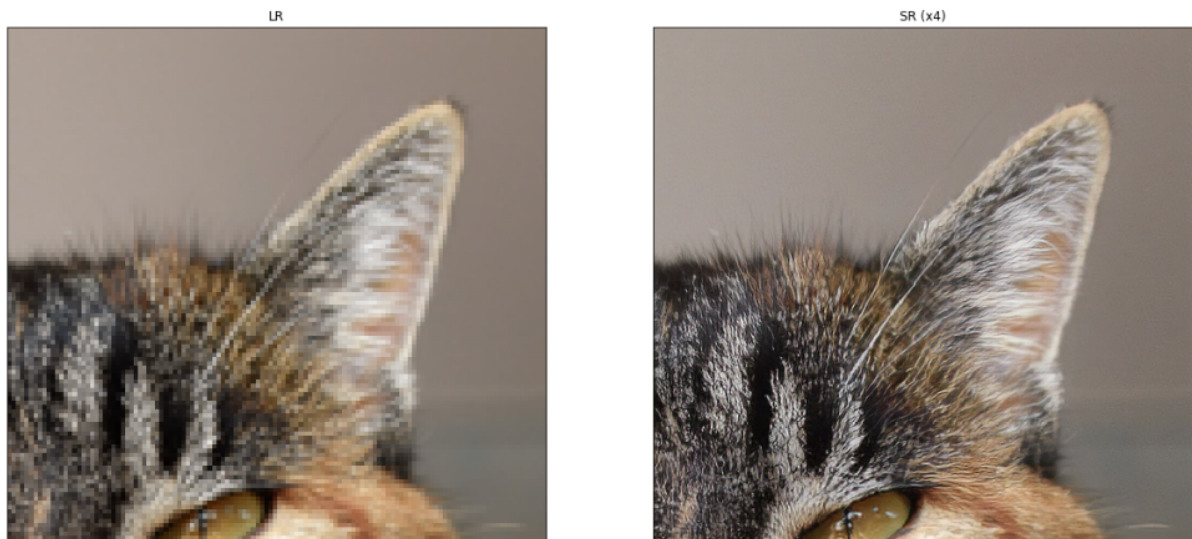
```
1 from model.wdsr import wdsr_b
2
3 model = wdsr_b(scale=4, num_res_blocks=32)
4 model.load_weights('weights/wdsr-b-32-x4/weights.h5')
5
6 lr = load_image('demo/0829x4-crop.png')
7 sr = resolve_single(model, lr)
8
9 plot_sample(lr, sr)
```



Weight normalization in WDSR models is implemented with the new `WeightNormalization` layer wrapper of Tensorflow Addons. In its latest version, this wrapper seems to corrupt weights when running `model.predict(...)`. A workaround is to set `model.run_eagerly = True` or compile the model with `model.compile(loss='mae')` in advance. This issue doesn't arise when calling the model directly with `model(...)` though. To be further investigated ...

SRGAN

```
1 from model.srgan import generator
2
3 model = generator()
4 model.load_weights('weights/srgan/gan_generator.h5')
5
6 lr = load_image('demo/0869x4-crop.png')
7 sr = resolve_single(model, lr)
8
9 plot_sample(lr, sr)
```



DIV2K dataset

For training and validation on DIV2K images, applications should use the provided [DIV2K](#) data loader. It automatically downloads DIV2K images to `.div2k` directory and converts them to a different format for faster loading.

Training dataset

```
1 from data import DIV2K
2
3 train_loader = DIV2K(scale=4,          # 2, 3, 4 or 8
4                       downgrade='bicubic', # 'bicubic', 'unknown', 'mild'
5                               ' or 'difficult'
6                               subset='train') # Training dataset are images
7                                           001 - 800
8
9 # Create a tf.data.Dataset
10 train_ds = train_loader.dataset(batch_size=16, # batch size as
11                                described in the EDSR and WDSR papers
12                                random_transform=True, # random crop,
13                                flip, rotate as described in the
14                                EDSR paper
15                                repeat_count=None) # repeat
16                                iterating over training images
17                                indefinitely
18
19 # Iterate over LR/HR image pairs
20 for lr, hr in train_ds:
```

```
14      # ....
```

Crop size in HR images is 96x96.

Validation dataset

```
1  from data import DIV2K
2
3  valid_loader = DIV2K(scale=4,          # 2, 3, 4 or 8
4                        downgrade='bicubic', # 'bicubic', 'unknown', 'mild'
5                        ' or 'difficult'
6                        subset='valid')    # Validation dataset are
9                        images 801 - 900
7  # Create a tf.data.Dataset
8  valid_ds = valid_loader.dataset(batch_size=1,          # use batch
9                                  size of 1 as DIV2K images have different size
10                                 random_transform=False, # use DIV2K
11                                 images in original size
12                                 repeat_count=1)        # 1 epoch
13 # Iterate over LR/HR image pairs
14 for lr, hr in valid_ds:
15     # ....
```

Training

The following training examples use the training and validation datasets described earlier. The high-level training API is designed around *steps* (= minibatch updates) rather than *epochs* to better match the descriptions in the papers.

EDSR

```
1  from model.edsr import edsr
2  from train import EdsrTrainer
3
4  # Create a training context for an EDSR x4 model with 16
5  # residual blocks.
6  trainer = EdsrTrainer(model=edsr(scale=4, num_res_blocks=16),
7                        checkpoint_dir=f'.ckpt/edsr-16-x4')
8
9  # Train EDSR model for 300,000 steps and evaluate model
10 # every 1000 steps on the first 10 images of the DIV2K
11 # validation set. Save a checkpoint only if evaluation
```

```

12 # PSNR has improved.
13 trainer.train(train_ds,
14               valid_ds.take(10),
15               steps=300000,
16               evaluate_every=1000,
17               save_best_only=True)
18
19 # Restore from checkpoint with highest PSNR.
20 trainer.restore()
21
22 # Evaluate model on full validation set.
23 psnr = trainer.evaluate(valid_ds)
24 print(f'PSNR = {psnr.numpy():3f}')
25
26 # Save weights to separate location.
27 trainer.model.save_weights('weights/edsr-16-x4/weights.h5')

```

Interrupting training and restarting it again resumes from the latest saved checkpoint. The trained Keras model can be accessed with `trainer.model`.

WDSR

```

1 from model.wdsr import wdsr_b
2 from train import WdsrTrainer
3
4 # Create a training context for a WDSR B x4 model with 32
5 # residual blocks.
6 trainer = WdsrTrainer(model=wdsr_b(scale=4, num_res_blocks=32),
7                       checkpoint_dir=f'.ckpt/wdsr-b-8-x4')
8
9 # Train WDSR B model for 300,000 steps and evaluate model
10 # every 1000 steps on the first 10 images of the DIV2K
11 # validation set. Save a checkpoint only if evaluation
12 # PSNR has improved.
13 trainer.train(train_ds,
14               valid_ds.take(10),
15               steps=300000,
16               evaluate_every=1000,
17               save_best_only=True)
18
19 # Restore from checkpoint with highest PSNR.
20 trainer.restore()
21
22 # Evaluate model on full validation set.
23 psnr = trainer.evaluate(valid_ds)
24 print(f'PSNR = {psnr.numpy():3f}')
25
26 # Save weights to separate location.
27 trainer.model.save_weights('weights/wdsr-b-32-x4/weights.h5')

```

SRGAN

Generator pre-training

```
1 from model.srgan import generator
2 from train import SrganGeneratorTrainer
3
4 # Create a training context for the generator (SRResNet) alone.
5 pre_trainer = SrganGeneratorTrainer(model=generator(), checkpoint_dir=f
    '.ckpt/pre_generator')
6
7 # Pre-train the generator with 1,000,000 steps (100,000 works fine too)
8 pre_trainer.train(train_ds, valid_ds.take(10), steps=1000000,
    evaluate_every=1000)
9
10 # Save weights of pre-trained generator (needed for fine-tuning with
    GAN).
11 pre_trainer.model.save_weights('weights/srgan/pre_generator.h5')
```

Generator fine-tuning (GAN)

```
1 from model.srgan import generator, discriminator
2 from train import SrganTrainer
3
4 # Create a new generator and init it with pre-trained weights.
5 gan_generator = generator()
6 gan_generator.load_weights('weights/srgan/pre_generator.h5')
7
8 # Create a training context for the GAN (generator + discriminator).
9 gan_trainer = SrganTrainer(generator=gan_generator, discriminator=
    discriminator())
10
11 # Train the GAN with 200,000 steps.
12 gan_trainer.train(train_ds, steps=200000)
13
14 # Save weights of generator and discriminator.
15 gan_trainer.generator.save_weights('weights/srgan/gan_generator.h5')
16 gan_trainer.discriminator.save_weights('weights/srgan/gan_discriminator
    .h5')
```

SRGAN for fine-tuning EDSR and WDSR models

It is also possible to fine-tune EDSR and WDSR x4 models with SRGAN. They can be used as drop-in replacement for the original SRGAN generator. More details in this article.

```
1 # Create EDSR generator and init with pre-trained weights
2 generator = edsr(scale=4, num_res_blocks=16)
3 generator.load_weights('weights/edsr-16-x4/weights.h5')
4
5 # Fine-tune EDSR model via SRGAN training.
6 gan_trainer = SrganTrainer(generator=generator, discriminator=
    discriminator())
7 gan_trainer.train(train_ds, steps=200000)
```

```
1 # Create WDSR B generator and init with pre-trained weights
2 generator = wdsr_b(scale=4, num_res_blocks=32)
3 generator.load_weights('weights/wdsr-b-16-32/weights.h5')
4
5 # Fine-tune WDSR B model via SRGAN training.
6 gan_trainer = SrganTrainer(generator=generator, discriminator=
    discriminator())
7 gan_trainer.train(train_ds, steps=200000)
```