Coding Project 3: Super Resolution - Youshaa Murhij (yosha.morheg@gmail.com)

I started working on this coding project before you mention that we can choose only two out of four projects and I have already submitted two coding projects previously. But I decided to submit this one too even if it is not fully completed (Optional Tasks)

Introduction:

The goal of this project is to upscale and improve the quality of low-resolution images. This project contains implementations of Residual Dense Network for Single Image Super-Resolution (ISR) as well as scripts to train these networks using content and adversarial loss components.

The implemented networks include:

- The super-scaling Residual Dense Network described in Residual Dense Network for Image Super-Resolution (Zhang et al. 2018).

Notebooks readme file is attached to carry training and prediction steps

My comments on the 'required to read' paper:

Zhang et al proposed a Residual dense network, which makes using the hierarchical features of the original LR images through the convolutional layer more efficient. RDN proposed residual dense block (RDB) to extract abundant local features via dense connected convolutional layers.

RDN adopts local dense connections into residual dense blocks and depends on the sub-pixel convolution to aggregate global features for upscaling LR natural images. A filter size of 1×1 has been commonly employed to reduce the number of feature maps as it is not an ideal choice to boost the performance of SR reconstruction. Although gate units with filters of size 1×1×1 led to fast and stable convergence, they showed limited learning capacity as compared to 3×3×3 filters in predicting HR details.

They achieved impressive performance by exploiting the hierarchical features from all the convolutional layers. They adopted dense skip connections in their network for SISR to make full use of the hierarchical features, and achieved promising results.

In their dense architectures, features from all the previous layers are reused by the following layer.Their work has a two-step reconstruction process, a deconvolution to upscale LR features to HR space and a convolution to restore the HR image.

Installation baseline:

To install the Image Super-Resolution package from the GitHub source:

```
!git clone https://github.com/idealo/image-super-resolution
%cd image-super-resolution
!git lfs pull
!python setup.py install
```

RDN Pre-trained weights

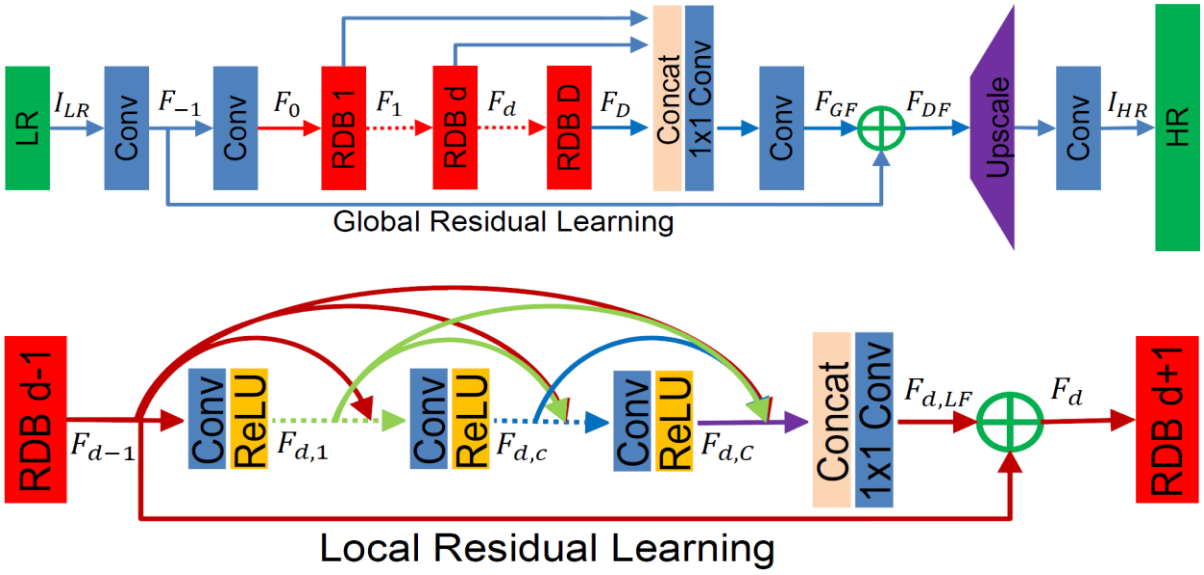
The weights of the RDN network trained on the DIV2K dataset are available in

`weights/sample_weights/rdn-C6-D20-G64-G064-x2/PSNR-driven/rdn-C6-D20-G64-G064-x2_PSNR_epoch086.hdf5`.

The model was trained using C=6, D=20, G=64, G0=64 as parameters (see architecture for details) for 86 epochs of 1000 batches of 8 32x32 augmented patches taken from LR images.

RDN Network architecture

The main parameters of the architecture structure are: - D - number of Residual Dense Blocks (RDB) - C - number of convolutional layers stacked inside a RDB - G - number of feature maps of each convolutional layers inside the RDBs - G0 - number of feature maps for convolutions outside of RDBs and of each RBD output



Source: *Residual Dense Network for Image Super-Resolution*

Main functions and Classes overview:

Class Discriminator

Implementation of the discriminator network for the adversarial component of the perceptual loss.

ARGS

- **patch_size**: integer, determines input size as (patch_size, patch_size, 3).
- **kernel_size**: size of the kernel in the conv blocks.

ATTRIBUTES

- **model**: Keras model.
- **name**: name used to identify what discriminator is used during GANs training.
- **model.name**: identifies this network as the discriminator network in the compound model built by the trainer class.
- **block_param**: dictionary, determines the number of filters and the strides for each conv block.

Class RDN

Implementation of the Residual Dense Network for image super-scaling.

ARGS

- **arch_params**: dictionary, contains the network parameters C, D, G, G0, x.
- **patch_size**: integer or None, determines the input size. Only needed at training time, for prediction is set to None.
- **c_dim**: integer, number of channels of the input image.
- **kernel_size**: integer, common kernel size for convolutions.
- **upscaling**: string, 'ups' or 'shuffle', determines which implementation of the upscaling layer to use.
- **init_extreme_val**: extreme values for the RandomUniform initializer.

ATTRIBUTES

- **C**: integer, number of conv layer inside each residual dense blocks (RDB).
- **D**: integer, number of RDBs.
- **G**: integer, number of convolution output filters inside the RDBs.
- **G0**: integer, number of output filters of each RDB.
- **x**: integer, the scaling factor.
- **model**: Keras model of the RDN.
- **name**: name used to identify what upscaling network is used during training.
- **model.name**: identifies this network as the generator network in the compound model built by the trainer class.

Class Predictor

The predictor class handles prediction, given an input model.

Loads the images in the input directory, executes training given a model and saves the results in the output directory. Can receive a path for the weights or can let the user browse through the weights directory for the desired weights.

ARGS

- **input_dir**: string, path to the input directory.
- **output_dir**: string, path to the output directory.
- **verbose**: bool.

ATTRIBUTES

- **extensions**: list of accepted image extensions.
- **img_ls**: list of image files in input_dir.

METHODS

- **get_predictions**: given a model and a string containing the weights' path, runs the predictions on the images contained in the input directory and stores the results in the output directory.

Class Trainer

Class object to setup and carry the training.

Takes as input a generator that produces SR images. Conditionally, also a discriminator network and a feature extractor to build the components of the perceptual loss. Compiles the model(s) and trains in a GANS fashion if a discriminator is provided, otherwise carries a regular ISR training.

ARGS

- **generator**: Keras model, the super-scaling, or generator, network.
- **discriminator**: Keras model, the discriminator network for the adversarial component of the perceptual loss.
- **feature_extractor**: Keras model, feature extractor network for the deep features component of perceptual loss function.
- **lr_train_dir**: path to the directory containing the Low-Res images for training.
- **hr_train_dir**: path to the directory containing the High-Res images for training.
- **lr_valid_dir**: path to the directory containing the Low-Res images for validation.
- **hr_valid_dir**: path to the directory containing the High-Res images for validation.
- **learning_rate**: float.
- **loss_weights**: dictionary, use to weigh the components of the loss function. Contains 'generator' for the generator loss component, and can contain 'discriminator' and 'feature_extractor' for the discriminator and deep features components respectively.
- **logs_dir**: path to the directory where the tensorboard logs are saved.
- **weights_dir**: path to the directory where the weights are saved.
- **dataname**: string, used to identify what dataset is used for the training session.
- **weights_generator**: path to the pre-trained generator's weights, for transfer learning.
- **weights_discriminator**: path to the pre-trained discriminator's weights, for transfer learning.
- **n_validation**: integer, number of validation samples used at training from the validation set.
- **flatness**: dictionary. Determines determines the 'flatness' threshold level for the training patches. See the TrainerHelper class for more details.
- **lr_decay_frequency**: integer, every how many epochs the learning rate is reduced.
- **lr_decay_factor**: 0 < float <1, learning rate reduction multiplicative factor.

METHODS

- **train**: combines the networks and triggers training with the specified settings.

train

ARGS

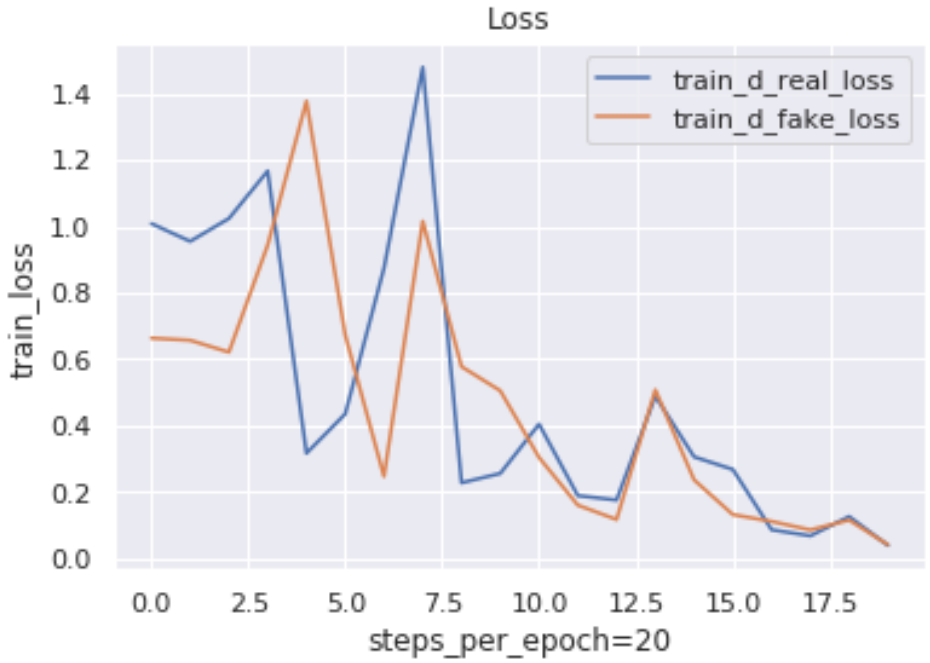
- epochs: how many epochs to train for.
- steps_per_epoch: how many batches epoch.
- batch_size: amount of images per batch.
- monitored_metrics: dictionary, the keys are the metrics that are monitored for the weights saving logic. The values are the mode that trigger the weights saving ('min' vs 'max').

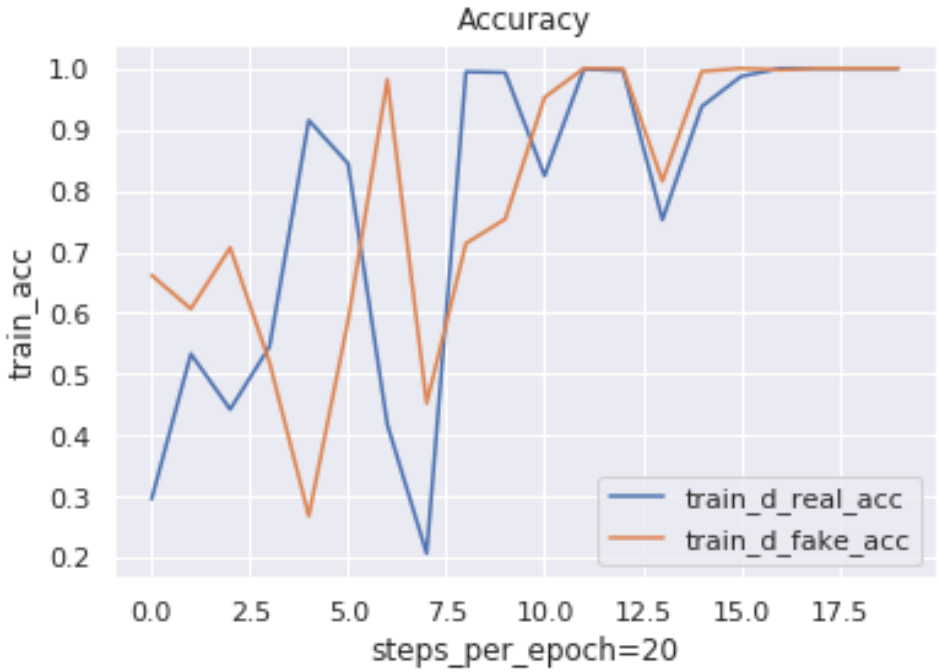
Pasting Training Results from console:

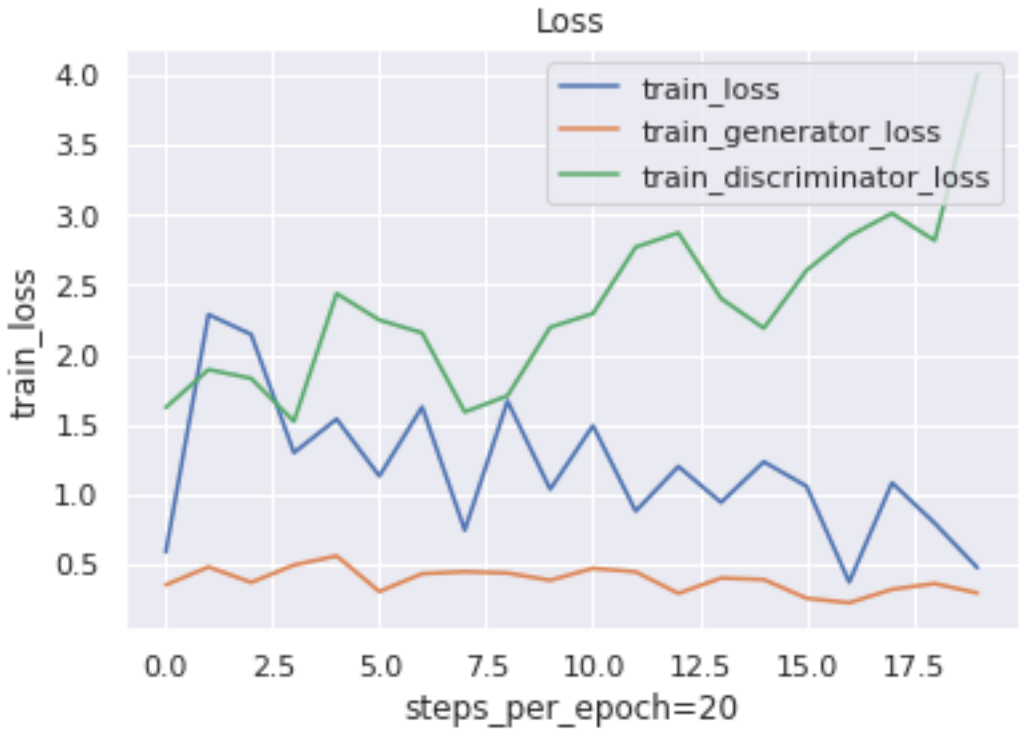
```
Training details:
  training_parameters:
    metrics: {'generator': <function PSNR_Y at 0x7f75e83a4950>}
    losses: {'generator': 'mae', 'discriminator': 'binary_crossentropy',
'feature_extractor': 'mse'}
    adam_optimizer: {'beta1': 0.9, 'beta2': 0.999, 'epsilon': None}
    learning_rate: {'initial_value': 0.0004, 'decay_factor': 0.5,
'decay_frequency': 30}
    flatness: {'min': 0.0, 'max': 0.15, 'increase': 0.01,
'increase_frequency': 5}
    n_validation: 40
    dataname: div2k
    fallback_save_every_n_epochs: 2
    log_dirs: {'logs': './logs', 'weights': './weights'}
    loss_weights: {'generator': 0.0, 'feature_extractor': 0.0833,
'discriminator': 0.01}
    hr_valid_dir: div2k/DIV2K_train_HR/
    lr_valid_dir: div2k/DIV2K_train_LR_bicubic/X2/
    hr_train_dir: div2k/DIV2K_train_HR/
    lr_train_dir: div2k/DIV2K_train_LR_bicubic/X2/
    lr_patch_size: 40
    steps_per_epoch: 20
    batch_size: 4
    starting_epoch: 0
  generator:
    name: rrdn
    parameters: {'C': 4, 'D': 3, 'G': 64, 'G0': 64, 'T': 10, 'x': 2}
    weights_generator: None
  discriminator:
    name: srgan-large
    weights_discriminator: None
  feature_extractor:
    name: vgg19
    layers: [5, 9]
Epoch 0/1
Current learning rate: 0.00039999998989515007
  0%|          | 0/20 [00:00<?, ?it/s]
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.cast instead.
```

```
/usr/local/lib/python3.6/dist-packages/keras/engine/training.py:490:
  'Discrepancy between trainable weights and collected trainable'
100%|██████████| 20/20 [12:55<00:00, 38.05s/it]
Epoch 0 took      775.5s
160/160 [==============================] - 289s 2s/step
```

Training, validation and curves:







Comparing my results of training and validation procedure with the results obtained in the paper, I can say that they are far worse, because this model need more than one epochs to converge but we I loaded the pretrained weights from the repository and trained my model using the same parameters they used I got almost the same results they got as seen in the figures above (I plot the figures using my own code after recoding the log data during the training process to a text log file)

Set5

Eval. Mat	Scale	RDN (Paper)	RDN (mine)
PSNR	2	38.24	38.11

```
%cd drive/My\ Drive/image-super-resolution
```

Install required packages

```
!pip install folium==0.2.1
!pip install imgaug==0.2.6
```

```
!pip install ISR
```

Train

Get the training data

Geting data to train the model. The div2k dataset linked here is for a scaling factor of 2.

```
!wget http://data.vision.ee.ethz.ch/cvl/DIV2K/DIV2K_train_LR_bicubic_X2.zip
!wget http://data.vision.ee.ethz.ch/cvl/DIV2K/DIV2K_valid_LR_bicubic_X2.zip
!wget http://data.vision.ee.ethz.ch/cvl/DIV2K/DIV2K_train_HR.zip
!wget http://data.vision.ee.ethz.ch/cvl/DIV2K/DIV2K_valid_HR.zip
```

Extracting the training and validation datasets

```
!mkdir div2k
!unzip -q DIV2K_valid_LR_bicubic_X2.zip -d div2k
!unzip -q DIV2K_train_LR_bicubic_X2.zip -d div2k
!unzip -q DIV2K_train_HR.zip -d div2k
!unzip -q DIV2K_valid_HR.zip -d div2k
```

Create the models

Importing the models from the ISR package and create

- a RRDN super scaling network
- a discriminator network for GANs training
- a VGG19 feature extractor to train with a perceptual loss function

Carefully selecting

- 'x': this is the upscaling factor (2 by default)
- 'layers_to_extract': these are the layers from the VGG19 that will be used in the perceptual loss (leave the default if you're not familiar with it)
- 'lr_patch_size': this is the size of the patches that will be extracted from the LR images and fed to the ISR network during training time

We can play around with the other architecture parameters

```
from ISR.models import RRDN
from ISR.models import Discriminator
from ISR.models import Cut_VGG19
```

```
lr_train_patch_size = 40
layers_to_extract = [5, 9]
scale = 2
hr_train_patch_size = lr_train_patch_size * scale

rrdn  = RRDN(arch_params={'C':4, 'D':3, 'G':64, 'G0':64, 'T':10, 'x':scale}, patch_size=lr_train_patch_size)
f_ext = Cut_VGG19(patch_size=hr_train_patch_size, layers_to_extract=layers_to_extract)
discr = Discriminator(patch_size=hr_train_patch_size, kernel_size=3)
```

Give the models to the Trainer

The Trainer object will combine the networks, manage your training data and keep you up-to-date with the training progress through Tensorboard and the command line.

Here we do not use the pixel-wise MSE but only the perceptual loss by specifying the respective weights in `loss_weights`

```
from ISR.train import Trainer
loss_weights = {
  'generator': 0.0,
  'feature_extractor': 0.0833,
  'discriminator': 0.01
```

```
}
losses = {
  'generator': 'mae',
  'feature_extractor': 'mse',
  'discriminator': 'binary_crossentropy'
}

log_dirs = {'logs': './logs', 'weights': './weights'}

learning_rate = {'initial_value': 0.0004, 'decay_factor': 0.5, 'decay_frequency': 30}

flatness = {'min': 0.0, 'max': 0.15, 'increase': 0.01, 'increase_frequency': 5}

trainer = Trainer(
    generator=rrdn,
    discriminator=discr,
    feature_extractor=f_ext,
    lr_train_dir='div2k/DIV2K_train_LR_bicubic/X2/',
    hr_train_dir='div2k/DIV2K_train_HR/',
    lr_valid_dir='div2k/DIV2K_train_LR_bicubic/X2/',
    hr_valid_dir='div2k/DIV2K_train_HR/',
    loss_weights=loss_weights,
    learning_rate=learning_rate,
    flatness=flatness,
    dataname='div2k',
    log_dirs=log_dirs,
    weights_generator=None,
    weights_discriminator=None,
    n_validation=40,
)

```

Choosing epoch number, steps and batch size and start training

```
trainer.train(
    epochs=1,
    steps_per_epoch=20,
    batch_size=4,
    monitored_metrics={'val_PSNR_Y': 'max'}
)
```

```
Training details:
  training_parameters:
    metrics: {'generator': <function PSNR_Y at 0x7f75e83a4950>}
    losses: {'generator': 'mae', 'discriminator': 'binary_crossentropy', 'feature_extractor': 'mse'}
    adam_optimizer: {'beta1': 0.9, 'beta2': 0.999, 'epsilon': None}
    learning_rate: {'initial_value': 0.0004, 'decay_factor': 0.5, 'decay_frequency': 30}
    flatness: {'min': 0.0, 'max': 0.15, 'increase': 0.01, 'increase_frequency': 5}
    n_validation: 40
    dataname: div2k
    fallback_save_every_n_epochs: 2
    log_dirs: {'logs': './logs', 'weights': './weights'}
    loss_weights: {'generator': 0.0, 'feature_extractor': 0.0833, 'discriminator': 0.01}
    hr_valid_dir: div2k/DIV2K_train_HR/
    lr_valid_dir: div2k/DIV2K_train_LR_bicubic/X2/
    hr_train_dir: div2k/DIV2K_train_HR/
    lr_train_dir: div2k/DIV2K_train_LR_bicubic/X2/
    lr_patch_size: 40
    steps_per_epoch: 20
    batch_size: 4
    starting_epoch: 0
  generator:
    name: rrdn
    parameters: {'C': 4, 'D': 3, 'G': 64, 'G0': 64, 'T': 10, 'x': 2}
    weights_generator: None
  discriminator:
    name: srgan-large
    weights_discriminator: None
  feature_extractor:
    name: vgg19
    layers: [5, 9]
Epoch 0/1
Current learning rate: 0.00039999998989515007
  0%|          | 0/20 [00:00<?, ?it/s]WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/math_ops.py:306
Instructions for updating:
Use tf.cast instead.
/usr/local/lib/python3.6/dist-packages/keras/engine/training.py:490: UserWarning: Discrepancy between trainable weights and collected trai
  'Discrepancy between trainable weights and collected trainable'
100%|██████████| 20/20 [12:55<00:00, 38.05s/it]
Epoch 0 took      775.5s
160/160 [==============================] - 289s 2s/step
val_PSNR_Y is NOT among the model metrics, removing it.
{'val_loss': 0.9109238140285015, 'val_generator_loss': 0.3806507588829845, 'val_discriminator_loss': 0.6993054270744323, 'val_feature_extr
```

```
%matplotlib inline
import pandas as pd
import matplotlib.pyplot as plt
import io
import seaborn as sns
sns.set()
df= pd.read_csv('test.csv')
```

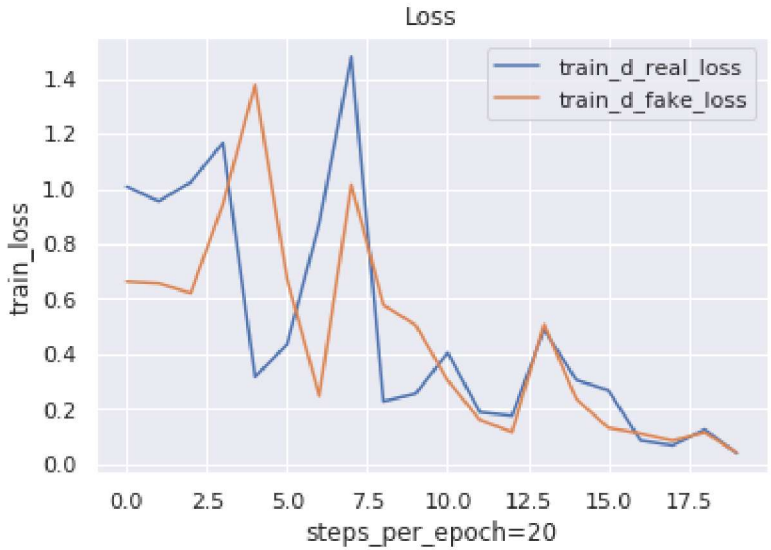
```
df.head()
```

	train_d_real_loss	train_d_real_acc	train_d_fake_loss	train_d_fake_acc	train_loss	train_generator_loss	train_discriminator_loss	t
0	1.008249	0.295625	0.662678	0.661875	0.587981	0.353752	1.621550	
1	0.954688	0.533125	0.656268	0.606875	2.286234	0.482036	1.892887	
2	1.023664	0.442500	0.620428	0.706875	2.144126	0.371839	1.829795	
3	1.166968	0.545000	0.943410	0.519375	1.298555	0.495433	1.522776	
4	0.315486	0.915000	1.377564	0.268125	1.539065	0.561159	2.436161	

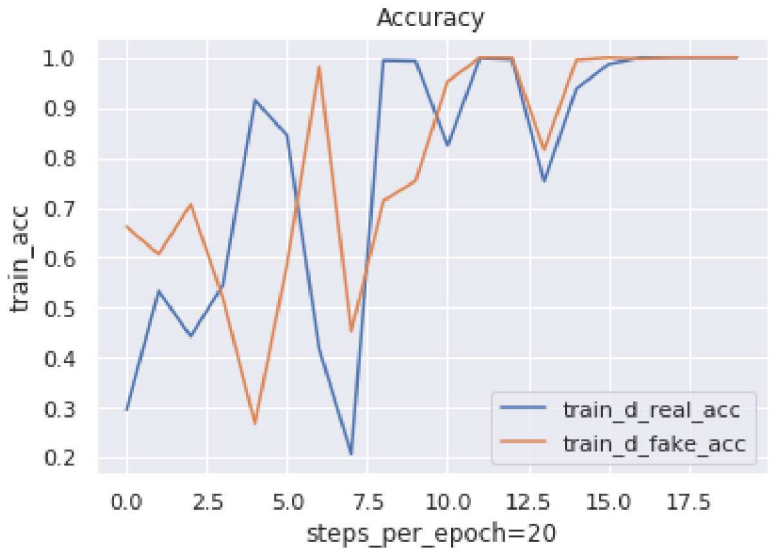
```
df.info()
```

```
df.columns
```

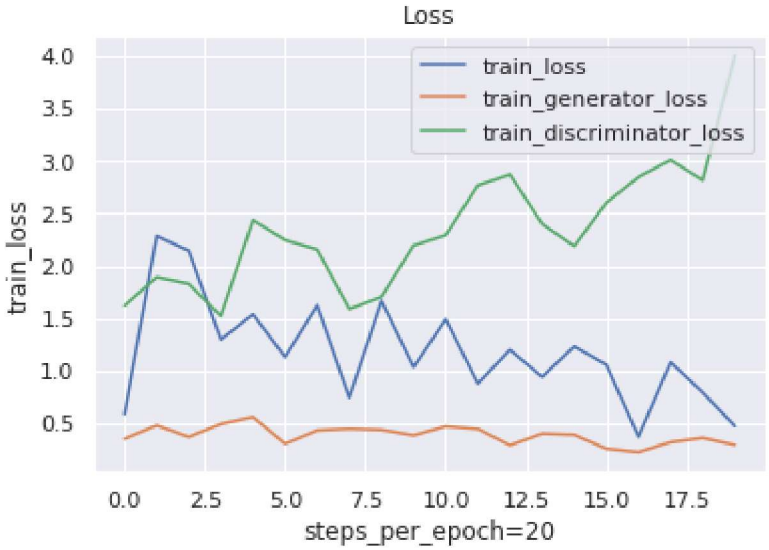
```
plt.xlabel('steps_per_epoch=20')
plt.ylabel('train_loss')
plt.title('Loss')
plt.plot(df['  train_d_real_loss'])
plt.plot(df[' train_d_fake_loss'])
plt.legend(['train_d_real_loss','train_d_fake_loss'],loc='upper right');

plt.show()
```



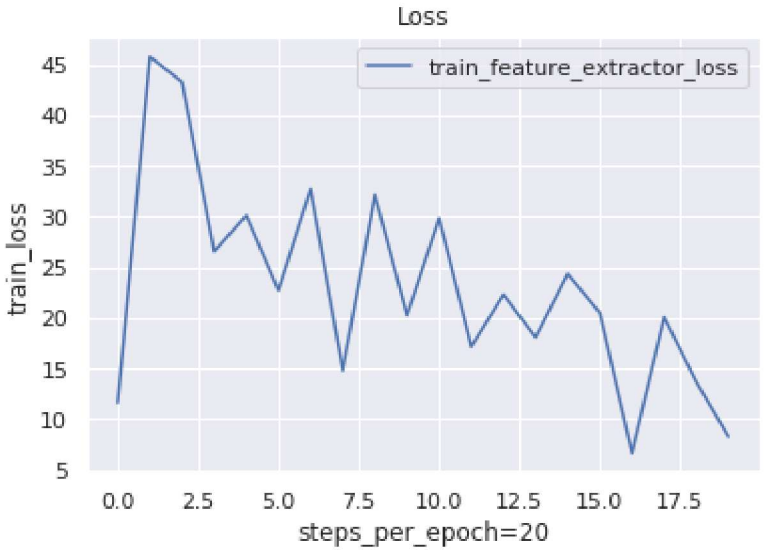

```
plt.xlabel('steps_per_epoch=20')
plt.ylabel('train_acc')
plt.title('Accuracy')
plt.plot(df['train_d_real_acc'])
plt.plot(df[' train_d_fake_acc'])
plt.legend(['train_d_real_acc','train_d_fake_acc'],loc='lower right');
plt.show()
```



```
plt.xlabel('steps_per_epoch=20')
plt.ylabel('train_loss')
plt.title('Loss')
plt.plot(df[' train_loss'])
plt.plot(df[' train_generator_loss'])
plt.plot(df[' train_discriminator_loss'])
plt.legend(['train_loss','train_generator_loss','train_discriminator_loss'],loc='upper right');
plt.show()
```




```
plt.xlabel('steps_per_epoch=20')
plt.ylabel('train_loss')
plt.title('Loss')
plt.plot(df[' train_feature_extractor_loss'])
plt.legend(['train_feature_extractor_loss'],loc='upper right');
plt.show()
```



Downloading testing datasets:

```
1 !wget http://vllab.ucmerced.edu/wlai24/LapSRN/results/SR_testing_datasets.zip
```

```
1 !mkdir SR
2 !unzip -q SR_testing_datasets.zip -d SR
```

```
1 !ls
```

```
1 !python tests/predict/test_predict.py
```

Predict

Get the pre-trained weights and data

Get the weights with

```
%cd /content/drive/My\ Drive/image-super-resolution
```

```
/content/drive/My Drive/image-super-resolution
```

```
!wget https://media.githubusercontent.com/media/idealo/image-super-resolution/master/weights/
--sample_weights/rdn-C6-D20-G64-G064-x2/ArtefactCancelling/rdn-C6-D20-G64-G064-x2_ArtefactCancelling_epoch219.hdf5
!wget https://media.githubusercontent.com/media/idealo/image-super-resolution/master/weights/
--sample_weights/rdn-C6-D20-G64-G064-x2/PSNR-driven/rdn-C6-D20-G64-G064-x2_PSNR_epoch086.hdf5
!wget https://media.githubusercontent.com/media/idealo/image-super-resolution/master/weights/
--sample_weights/rdn-C3-D10-G64-G064-x2/PSNR-driven/rdn-C3-D10-G64-G064-x2_PSNR_epoch134.hdf5
!wget https://media.githubusercontent.com/media/idealo/image-super-resolution/master/weights/
--sample_weights/rrdn-C4-D3-G32-G032-T10-x4/Perceptual/rrdn-C4-D3-G32-G032-T10-x4_epoch299.hdf5
#!mkdir weights
#!mv *.hdf5 weights
```

```
% cd weights
```

```
/content/drive/My Drive/image-super-resolution/weights
```

Download a sample image, in this case

```
!wget http://images.math.cnrs.fr/IMG/png/section8-image.png
!mkdir -p data/input/test_images
!mv *.png data/input/test_images
```

Load the image with PIL

```
import numpy as np
from PIL import Image

img = Image.open('data/input/test_images/section8-image.png')
img
```

Get predictions

Create the model and run prediction

Create the RDN model, for which we provide pre-trained weights, and load them

Choose amongst the available model weights, compare the output if you wish.

```
from ISR.models import RDN, RRDN
```

RRDN GANS model

```
#rrdn = RRDN(arch_params={'C': 4, 'D':3, 'G':32, 'G0':32, 'x':4, 'T': 10})
#rrdn.model.load_weights('weights/rrdn-C4-D3-G32-G032-T10-x4_epoch299.hdf5')
```

```
rrdn = RRDN(arch_params={'C':4, 'D':3, 'G':32, 'G0':32, 'T':10, 'x':4})
rrdn.model.load_weights('weights/rrdn-C4-D3-G32-G032-T10-x4_epoch299.hdf5')
```

Large RDN model

```
rdn = RDN(arch_params={'C': 6, 'D':20, 'G':64, 'G0':64, 'x':2})
rdn.model.load_weights('weights/rdn-C6-D20-G64-G064-x2_PSNR_epoch086.hdf5')
```

Small RDN model

```
rdn = RDN(arch_params={'C': 3, 'D':10, 'G':64, 'G0':64, 'x':2})
rdn.model.load_weights('weights/rdn-C3-D10-G64-G064-x2_PSNR_epoch134.hdf5')
```

Large RDN noise cancelling, detail enchancing model

```
rdn = RDN(arch_params={'C': 6, 'D':20, 'G':64, 'G0':64, 'x':2})
rdn.model.load_weights('weights/rdn-C6-D20-G64-G064-x2_ArtefactCancelling_epoch219.hdf5')
```

Run prediction

```
sr_img = rdn.predict(np.array(img))
Image.fromarray(sr_img)
```


```
sr_img = rdn.predict(np.array(img))
Image.fromarray(sr_img)
```


Usecase: upscaling noisy images

Now, for science, let's make it harder for the networks.

We compress the image into the jpeg format to introduce compression artefact and lose some information.

We will compare:

- the baseline bicubic scaling
- the basic model - Add Hyperlink
- a model trained to remove noise using perceptual loss with deep features and GANs training

So let's first compress the image

```
img.save('data/input/test_images/compressed.jpeg','JPEG', dpi=[300, 300], quality=50)
compressed_img = Image.open('data/input/test_images/compressed.jpeg')

compressed_img
```

(open the image in a new tab and zoom in to inspect it)

Baseline

Bicubic scaling

```
compressed_img.resize(size=(compressed_img.size[0]*2, compressed_img.size[1]*2), resample=Image.BICUBIC)
```

Large RDN model (PSNR trained)

```
rdn = RDN(arch_params={'C': 6, 'D':20, 'G':64, 'G0':64, 'x':2})
rdn.model.load_weights('weights/rdn-C6-D20-G64-G064-x2_PSNR_epoch086.hdf5')
sr_img = rdn.predict(np.array(compressed_img))
Image.fromarray(sr_img)
```

Small RDN model (PSNR trained)

```
rdn = RDN(arch_params={'C': 3, 'D':10, 'G':64, 'G0':64, 'x':2})
rdn.model.load_weights('weights/rdn-C3-D10-G64-G064-x2_PSNR_epoch134.hdf5')
sr_img = rdn.predict(np.array(compressed_img))
Image.fromarray(sr_img)
```

Large RDN noise cancelling, detail enhancing model

```
rdn = RDN(arch_params={'C': 6, 'D':20, 'G':64, 'G0':64, 'x':2})
rdn.model.load_weights('weights/rdn-C6-D20-G64-G064-x2_ArtefactCancelling_epoch219.hdf5')
sr_img = rdn.predict(np.array(compressed_img))
Image.fromarray(sr_img)
```

Predictor Class

You can also use the predictor class to run the model on entire folders:

```
from ISR.predict import Predictor
!mkdir -p data/output
predictor = Predictor(input_dir='data/input/test_images/')
predictor.get_predictions(model=rdn, weights_path='weights/rdn-C6-D20-G64-G064-x2_ArtefactCancelling_epoch219.hdf5')
```