Final Project

# Super Resolution Using GANs

COGNITIVE COMPUTING – TEAM 1

Trupti Gore

## Abstract

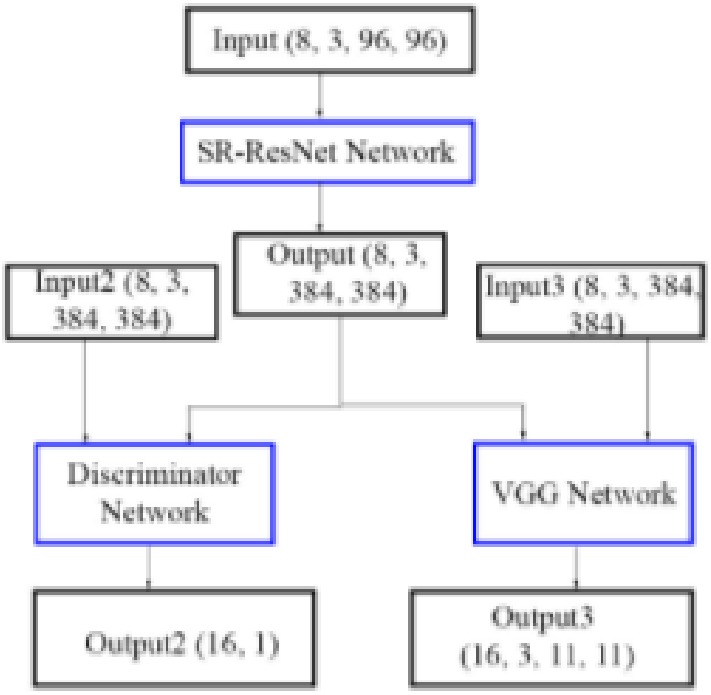
My study involved in using GANs to generate images with Super Resolution when a low-resolution image is input at the Generator with reference to this project.

## Introduction

Image super-resolution is a sought-after topic of application of Deep Learning. The GAN was first introduced by Ian Goodfellow and since then there have been varied applications of them. By the very nature of it, the GAN is very complex to build and train.

My model consists of a GAN network along with a VGG 16 network and a super-resolution net. I have used coco dataset to pre-train the model, and benchmarked it on set14, set5 and bsd100 dataset as a part of project.

## Architecture of the Model




https://github.com/truptigore17/my-projects

Initially, only the SR-ResNet model is created, to which the VGG network is appended to create the pre-training model. The VGG weights are freeze as we will not update these weights.

In the pre-train mode:

1. The discriminator model is not attached to the entire network. Therefore it is only the SR + VGG model that will be pretrained first.

2. During pretraining, the VGG perceptual losses will be used to train (using the ContentVGGRegularizer) and TotalVariation loss (using TVRegularizer). No other loss (MSE, Binary crosss entropy, Discriminator) will be applied.

3. Content Regularizer loss will be applied to the VGG Convolution 2-2 layer

4. After pre training the SR + VGG model, we will pretrain the discriminator model.

5. During discriminator pretraining, model is Generator + Discriminator. Only binary cross entropy loss is used to train the Discriminator network.

In the full train mode:

1. The discriminator model is attached to the entire network. Therefore it creates the SR + GAN + VGG model (SRGAN)
2. Discriminator loss is also added to the VGGContentLoss and TVLoss.
3. Content regularizer loss is applied to the VGG Convolution 5-3 layer. (VGG 16 is used instead of 19 for now)

## Usage

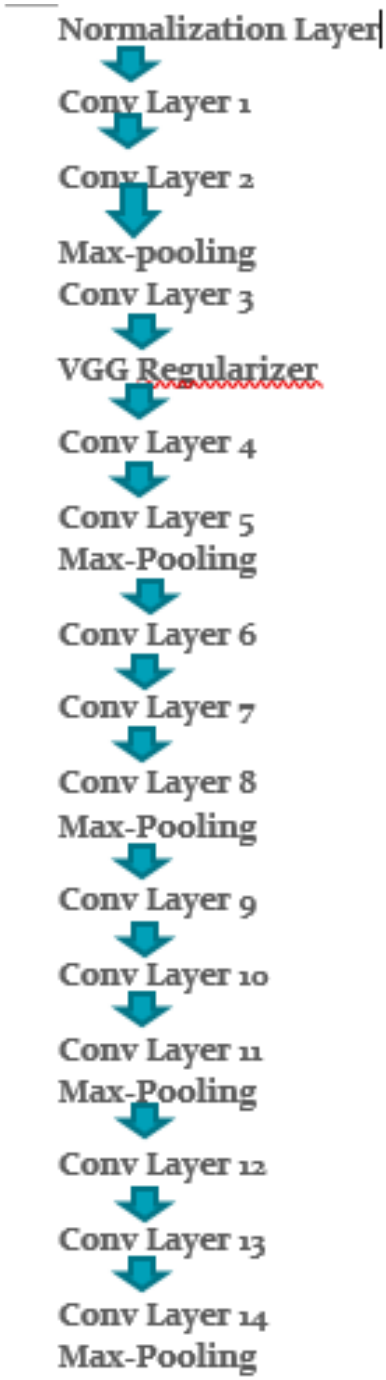
Currently, models.py contains most of the code to train and create the models. To use different modes, uncomment the parts of the code that you need.

Note the difference between the *_network objects and *_model objects.

- The *_network objects refer to the helper classes which create and manage the Keras models, load and save weights and set whether the model can be trained or not.
- The *_models objects refer to the underlying Keras model.

## VGG Model Structure

Our VGG Model is structured as below :




https://github.com/truptigore17/my-projects

## Method of Approach

I have used 2 regularizers : Content VGG regularizer and Total Variation regularizer

**Pre-training** :

Pre-trained in two stages :

1. **VGG+ Super Resolution:**

Pre-trained the VGG + Super resolution network first. The Content VGG regularizer and Total variation regularizer is used in the VGG network. This network is trained with perceptual losses.

2. **Discriminator + Generator:**

The Generator + Discriminator is trained in the second stage. Binary Cross entropy loss is used to train this GAN network.

**Full Train Mode** :

1. The two pretrained models are then combined, which creates SR+VGG+GAN model(SR-GAN)
2. This model is totally trained over Content VGG regularizer+ TV Loss regularizer+ binary cross entropy loss.

## Benchmarking

The PSNR for the three datasets used for bench marking is as follows :

| **Dataset** | **PSNR** |
|---|---|
| set14 | **20.64** |
| set5 | **19.93** |
| bsd100 | **20.21** |

## Code Snippets :

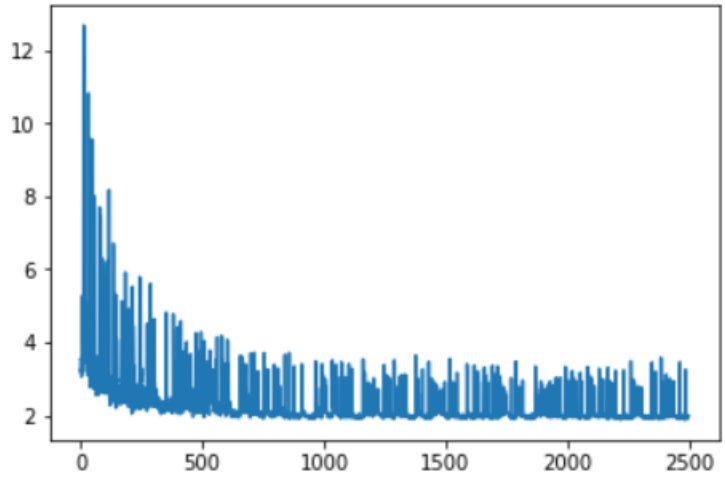
```
In [1]: from keras import backend as K
from keras.models import Model
from keras.layers import Input, add, concatenate, BatchNormalization, LeakyReLU, Flatten, Dense
from keras.layers import Conv2D, MaxPooling2D, UpSampling2D
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.utils.np_utils import to_categorical
from keras.utils.data_utils import get_file

from keras_ops import fit as bypass_fit, smooth_gan_labels

from layers import Normalize, Denormalize, SubPixelUpscaling
from loss import AdversarialLossRegularizer, ContentVGGRegularizer, TVRegularizer, psnr, dummy_loss

import os
import time
import h5py
import numpy as np
import json
from imageio import imwrite as imsave
from scipy.misc import imresize
from scipy.ndimage.filters import gaussian_filter

TF_WEIGHTS_PATH_NO_TOP = r"https://github.com/fchollet/deep-learning-models/releases/download/v0.1/vgg16_weights_tf_dim_ordering_

if not os.path.exists("weights/"):
    os.makedirs("weights/")

if not os.path.exists("val_images/"):
    os.makedirs("val_images/")

if K.image_dim_ordering() == "th":
    channel_axis = 1
else:
    channel_axis = -1
```

Setting input variables

```
In [8]: # Pretrain the SRGAN network
srgan_network.pre_train_srgan(coco_path, num_images=80000, epochs=1)

Could not load generator weights.
Training SRGAN network

Epoch : 1
Found 82783 images belonging to 2 classes.

/home/ubuntu/anaconda3/envs/tensorflow_p36/lib/python3.6/site-packages/ipykernel/__main__.py:208: DeprecationWarning: `imresi
ze` is deprecated!
`imresize` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``skimage.transform.resize`` instead.

1 / 80000 | Improvement : 0.00 % | 47.38 s/steps | GLoss: 392444641280.00
2 / 80000 | Improvement : 3.49 % | 0.05 s/steps | GLoss: 378745782272.00
3 / 80000 | Improvement : 11.89 % | 0.04 s/steps | GLoss: 333696106496.00
4 / 80000 | Improvement : 32.09 % | 0.04 s/steps | GLoss: 226622046208.00
5 / 80000 | Improvement : -17.66 % | 0.04 s/steps | GLoss: 266653220864.00
6 / 80000 | Improvement : 31.99 % | 0.04 s/steps | GLoss: 181342601216.00
7 / 80000 | Improvement : 1.18 % | 0.04 s/steps | GLoss: 179209011200.00
8 / 80000 | Improvement : 6.20 % | 0.04 s/steps | GLoss: 168106967040.00
9 / 80000 | Improvement : -29.33 % | 0.04 s/steps | GLoss: 217418825728.00

In [9]: import matplotlib.pyplot as plt
```

Pre-train SR-GAN model

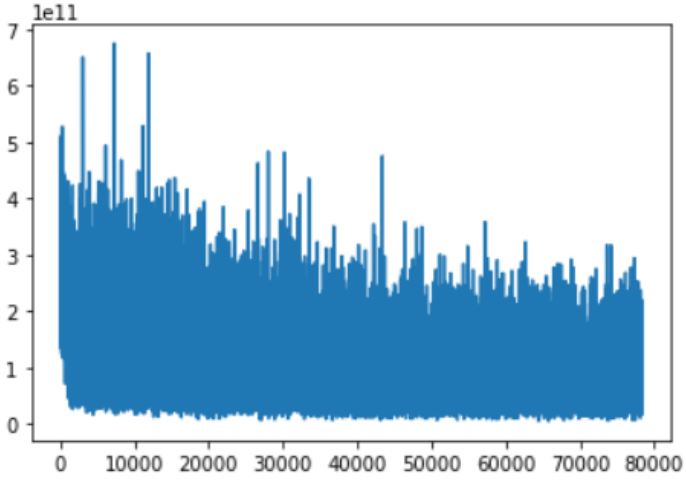
https://github.com/truptigore17/my-projects




```
Mean discriminator loss : 2.286400362801552
Std discriminator loss :  0.6954556596970372
Min discriminator loss :  1.8700445294380188
```

Discriminator Pre-train Loss Plot

```
Pretrain loss: Loaded SRGAN JSON.
Generator loss
```

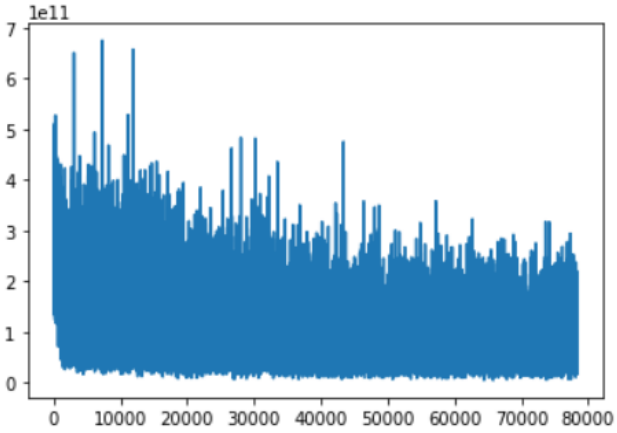



```
Mean gan loss : 105233211341.21861
Std gan loss :  55777692427.159485
Min gan loss :  4423798272.0
PSNR loss
```

Pre-train GAN Loss

https://github.com/truptigore17/my-projects

```
Pretrain loss: Loaded SRGAN JSON.
Generator loss
```



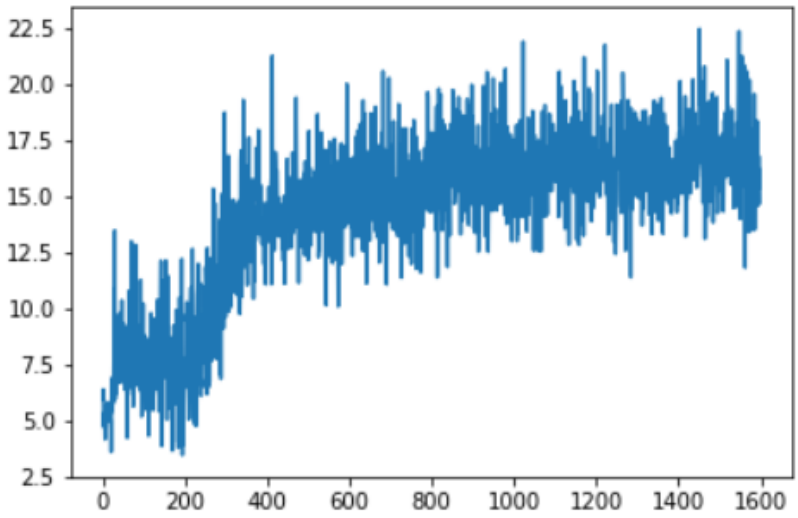

```
Mean gan loss : 105233211341.21861
Std gan loss :  55777692427.159485
Min gan loss :  4423798272.0
PSNR loss
```

Pre-train GAN Loss

```python
from keras.layers import Input
from keras.models import Model
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator

import sys
sys.path.append("..")

#import models
from loss import PSNRLoss, psnr

import os
import time
import numpy as np
from imageio import imwrite as imsave
from scipy.misc import imresize
from scipy.ndimage.filters import gaussian_filter

base_weights_path = "weights/"
base_val_images_path = "val_images/"
base_test_images = "test_images/"

set5_path = "tests/set5"
set14_path = "tests/set14"
bsd100_path = "tests/bsd100"

if not os.path.exists(base_weights_path):
    os.makedirs(base_weights_path)

if not os.path.exists(base_val_images_path):
    os.makedirs(base_val_images_path)

if not os.path.exists(base_test_images):
    os.makedirs(base_test_images)
```

Bench-Marking

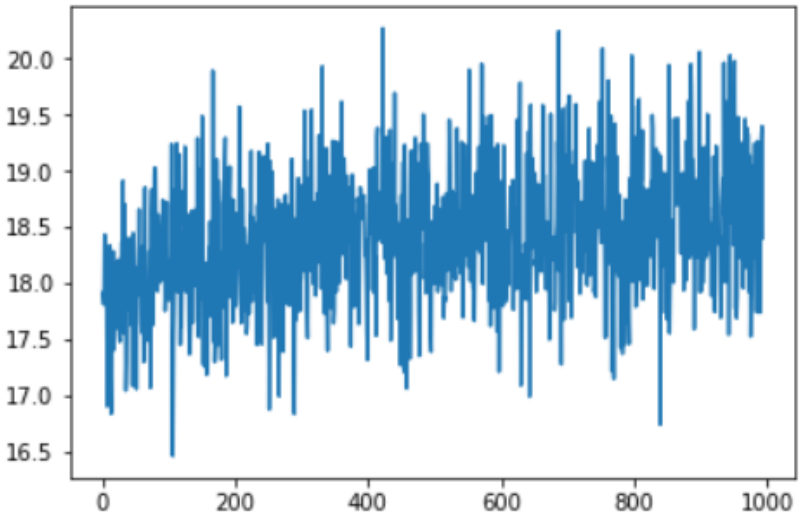
https://github.com/truptigore17/my-projects




```
Mean psnr loss : 14.321495875147747
Std psnr loss :  3.6173389147162807
Min psnr loss :  3.449070155620575
```

Full Train PSNR Loss

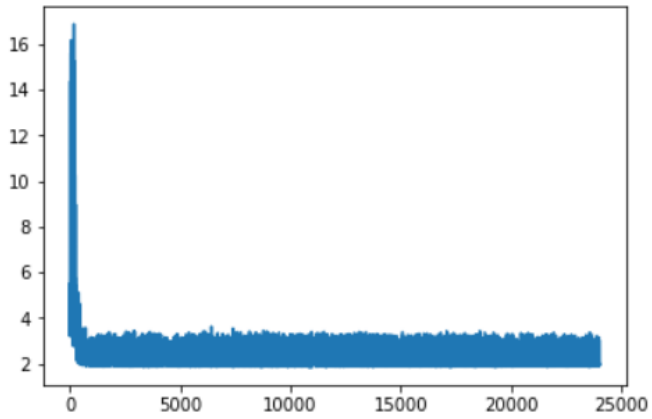
```
Fulltrain loss: Loaded fulltrain JSON
PSNR loss
```




```
Mean psnr loss : 18.413078985696462
Std psnr loss :  0.595642967460921
Min psnr loss :  16.45867645740509
Discriminator accuracy
```

Full Train Discriminator Accuracy

https://github.com/truptigore17/my-projects




```
Mean discriminator loss : 2.104387870296243
Std discriminator loss :  0.5005519708440574
Min discriminator loss :  1.8318455815315247
Generator loss
```

Full Train Generator Loss

## Challenges

Listed below are the difficulties faced while training the model :

- To understand the existing benchmarks and deciding what models to fit into our GAN setting
- Understand the existing work on super resolution. There is a variety of super resolution contesting against each other and I had to do extensive research while finalizing a model
- The datasets used (Coco dataset) were massive (14gb). This took a lot of computational power and time even after using AWS instance with high bandwidth to load the data on the filesystem and training the model on the data.
- Our model is a very deep CNN net with a complex architecture involving VGG-16 . This again consumed computational power and time. It took an average of 10 hours of training time to train our model on AWS p3 2xlarge instance having NVIDIA Volta GPUs.

https://github.com/truptigore17/my-projects

- The configuration of the structure that took some time to converge. There were times when the mean square error of the generated image value and the validation image were close (which was expected). However, for most times, the value for mse between generated and validation image kept fluctuating with vast differences. To overcome this, I inserted the VGG regularizer in the VGG architecture. This helped to stabilize the model.
- Used plots to understand how the model was converging and the plots were very discrete and this made it difficult to understand how the model was performing. To handle this challenge, I used transposed images in our generator and then recorded loss and used data augmentation before comparing it from the discriminator.

## Discussion

The model achieved a commendable benchmark for the three datasets as below

| **Dataset** | **PSNR** |
|---|---|
| set14 | **20.64** |
| set5 | **19.93** |
| bsd100 | **20.21** |

The introduction of the regularizer helped to achieve the anticipated benchmarking. The methodology of structuring and pre-training the model in patches helps to overcome the challenges faced during the earlier stage of training the model.

## Future Scope

I believe that with the given time frame, I have implemented VGG-16 with regularizer helped us successfully achieve a good result. A model with VGG-19 of deeper Conv net and apt regularizers could be a better performer. I am looking forward to working on this sooner.

## Conclusion

The model trained faster than the earlier instance of the model training (without regularizer) and succeeded in establishing higher benchmarks with the 3 datasets.

| Dataset | PSNR |
|---|---|
| set14 | **20.64** |
| set5 | **19.93** |
| bsd100 | **20.21** |