Image In-painting using Generative Adversarial Network

Pal, Divyansh divpal@umich.edu Rastogi, Aakash arastog@umich.edu

1 Introduction

For EECS 542 final project we implemented a framework for image completion based on the paper **Globally and Locally Consistent Image Completion** and tried several improvement techniques on it. The technique allows filling in target regions with alternative contents by removing unwanted objects, or by generating occluded regions which are both locally continuous and globally consistent with the image. This is done via a generative adversarial network (GAN), in which we simultaneously train two models: a generative model that generates the occluded regions of an image and a discriminator model that estimates the probability that the image came from training data rather than the generative model. The discriminator model in our case, has two different discriminators in it. The global discriminator looks at the entire image to assess the coherence, whereas the local discriminator looks only at the coherence of the generated patches. Since the optimization consists of jointly minimizing and maximizing conflicting objectives, the training is not very stable. To avoid that, the images produced by completion network are not generated from noise. That helps the training process to be initially stable.

2 Motivation behind Computer Vision Task

Image re-construction or in-painting is the process of reconstructing lost or deteriorated parts of images and videos. In the task of Image in-painting, prediction of high-level context is required, which makes this task significantly more difficult. Numerous applications such as restoration of damaged paintings or image editing benefit from accurate in-painting methods if large regions are missing. All image in-painting methods require appropriate information to be contained in the input image, e.g., similar pixels, structures, or patches. This assumption is hard to satisfy, if the missing region is large and possibly of arbitrary shape.
One of the biggest limitations of the approaches used for image in-painting, is that the synthesized texture only comes from the input image which is an issue if the missing region is big or convincing completion texture can't be found in input image. An active (and seemingly more promising) technique for image restoration is the neural network based. The most significant difference between neural network methods and other methods is that they typically learn parameters for image restoration directly from training data (e.g., pairs of clean and corrupted images) rather than relying on pre-defined image priors. Our method involves GAN which are based on convolutional neural networks.

3 Dataset Details

The network is trained on the aligned Large-scale CelebFaces Attributes (CelebA) Dataset. We made a split of 200,000 images for training and 2,599 images for testing. The images selected for the training and evaluation were selected at random. We train using 200,000 celebrity images from Large-scale CelebFaces Attributes (CelebA) Dataset, each with 40 attribute annotations. The images in this dataset cover large pose variations and background clutter.

4 Details of Implementation

In this project we implemented an image completion framework using deep convolutional neural networks based on generative adversarial process. A single generator network was used for image completion. The discriminator is composed of two separate parts namely, global context discriminator and local context discriminator. The completion model is pitted against two adversaries: local and global context discriminator models which learn to determine whether an image is from the completion model distribution or the data distribution. A large domain of images can be realistically completed, only if all three networks are trained together for a good amount of time.

4.1 Generative Adversarial Network

The proposed framework is for estimating generative models via an adversarial process, in which two models are simultaneously trained: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake. The optimization process of GANs is a minimax game process, and the optimization goal is to reach Nash equilibrium, where the generator is considered to have captured the distribution of real samples. In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to $\frac{1}{2}$ everywhere. In the case where G and D are defined by multi-layer perceptrons, the entire system can be trained with back-propagation.
Generative adversarial networks utilize discriminative models, which are capable of mapping high dimensional, rich sensory inputs to class labels mainly because they implement back-propagation, and dropout algorithms, using piece wise linear units, which have a particularly well behaved gradient. They also overcome the disadvantages of the generative models which involve approximation of many intractable probabilistic computations that arise in maximum likelihood estimation and related strategies.

4.2 Architecture

The network consists of a generator network and discriminator network consisting of two parts - Local Discriminator and Global Discriminator. An overview of the architecture used can be seen in Fig 1.

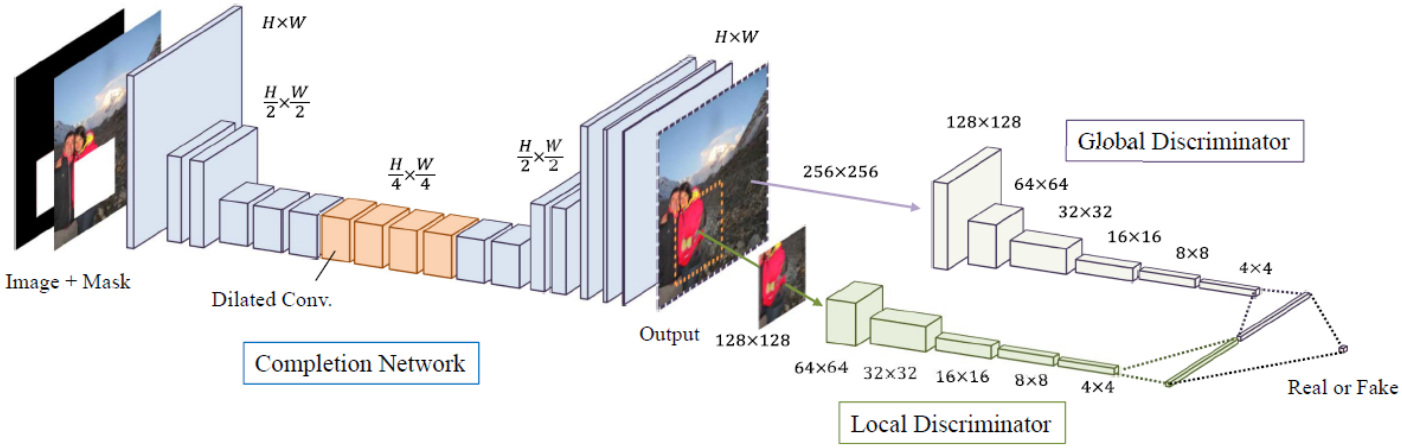

Figure 1: Overview of the architecture for Image completion. It consists of one completion network, one global context discriminator and one local context discriminator.The Global discriminator takes the whole image as input whereas the Local discriminator takes a small region around the completed area as input.

4.2.1 Generator

The generator network in the paper is based on a fully convolutional network. The model architecture can be seen in Fig 2. The completion network takes RGB images along with a binary channel as input. The channel indicates the image completion mask, where 1 is taken if a pixel is to be completed. The output for the network is also an RGB image. Since, we do not want any changes in regions outside the patch, the pixels outside the completed region are restored to the input RGB values. An encoder- decoder architecture is used in order to reduce memory usage and computational time by reducing the image resolution before processing. Then deconvolution layers (convolutional layers with fractional strides) are used to recover the original resolution. In the architecture, the resolution was reduced to $\frac{1}{4}$ of the original size. In order to generate non - blurred texture in the missing regions, the dilated convolution layers are also used (with $l > 1$), so as to consider a larger input area as compared to standard convolution. The resulting network model computes each output pixel by taking a 160 X 160 pixel region as input. The masks are computed randomly, with size ranging from 46 to 92 pixels.

4.2.2 Discriminator

The context discriminator network follows the generator network. Overview of discriminator network can be seen in Fig 3. Globally and locally aware context discriminators are responsible for identifying if an image is real or has been

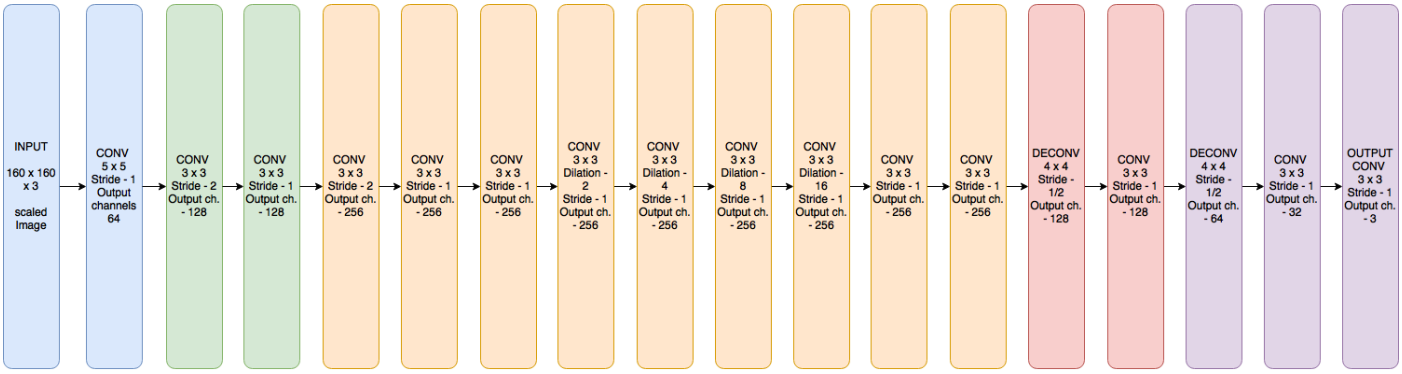

Figure 2: Architecture of Generator. It consists of convolution, dilated convolution and deconvolution layers.

completed. The global context discriminator takes as input the image re scaled to 160 X 160 pixels. The global discriminator network consists of six convolutional layers and one fully connected layer which outputs a 1024 dimensional vector. The convolutional layers have a stride of 2x2 pixels to decrease the image resolution while increasing the number of output filters. The kernel size is 5x5 for the global context discriminator. The local context discriminator takes as input a 80 x 80 pixel image patch centered around the completed patch generated by generator network in the case of completed images and a random patch in the case of real images. This outputs a 1024-dimensional vector representing the local context around the completed region. The output of the global and local context discriminators are concatenated together into one 2048 dimensional vector which is operated on by one fully connected layer followed by a sigmoid non-linearity to give us the probability of the image being real or fake.

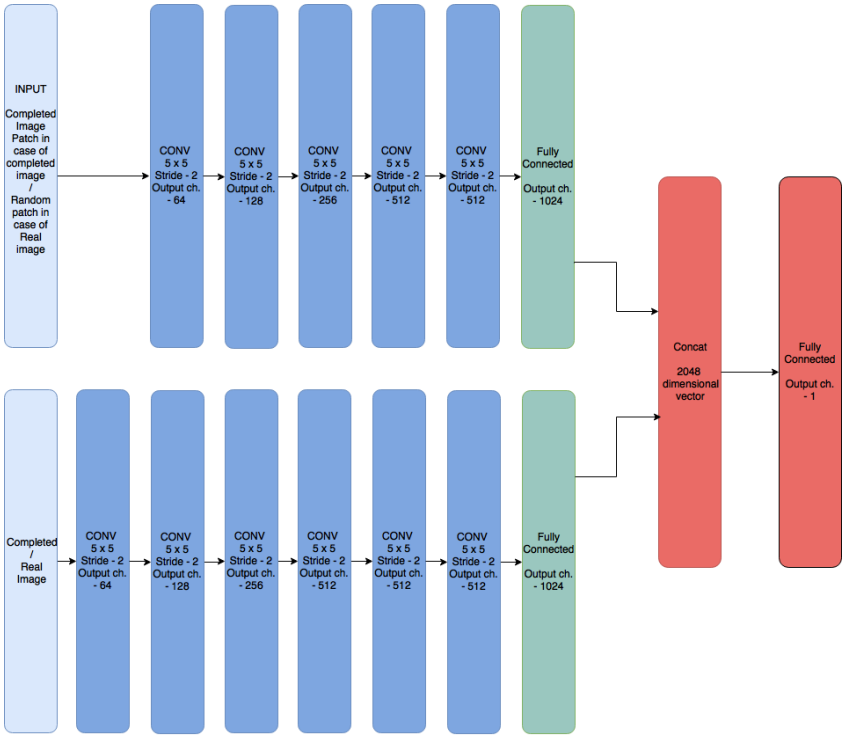

Figure 3: Architecture of Generator. It consists of convolution, dilated convolution and deconvolution layers.

4.3 Training

Let $C(x, M_c)$ denote the completion network and $D(x, M_d)$ denote the combined discriminator network. x denotes the input image, Mc denotes the binary completion region mask and Md denotes the binary mask for the discriminator network. The network uses a weighted Mean Squared Error (MSE) loss for the reconstruction network and a Generative Adversarial Network (GAN) loss to improve the sharpness and realism of the results. The MSE loss for the network is defined as:

$$L(x, M_c) = \|M_c \odot (C(x, M_c) - x)\|^2, \tag{1}$$

where, $\odot$ represents pixel wise multiplication.
The context discriminator works with the GAN loss. This training process involves turning the standard optimization of the neural network into a min-max optimization problem in which at each iteration the discriminator networks are jointly updated with the completion network. The GAN loss is given by:

$$\min_C \max_D \mathbb{E}[log(D(x, M_d)) + log(1 - D(C(x, M_c), M_c))], \tag{2}$$

where, Md is a random mask, Mc is input mask and the expectation value is the average over the training images x. Combining the two loss functions we get,

$$\min_C \max_D \mathbb{E}[L(x, M_c) + \alpha log(D(x, M_d)) + \alpha log(1 - D(C(x, M_c), M_c))], \tag{3}$$

where, α is a weighing hyper parameter.
The optimization procedure involves backpropagating the gradients for the Generator and Discriminator network using Standard gradient descent. Let θ_c denote the generator network parameters. In the standard stochastic gradient descent method, the loss is minimized by taking the gradient of the loss function with respect to the completion network parameters θ_c and updating the network parameters. The gradient of the loss function with respect to θ_c is given by:

$$\mathbb{E}[\nabla_{\theta_C} L(x, M_c) + \alpha \nabla_{\theta_C} log(1 - D(C(x, M_c), M_c))], \tag{4}$$

The training procedure of the complete network is as follows:

Algorithm 1 Training procedure of image completion network

```
while iterations t < T_train do
  Sample a minibatch of images x from training data.
  Generate masks M_c with random holes for each image x in the minibatch.
  if t < T_c then
    Update the generator network C with weighted MSE loss using (x, M_c).
  else
    Generate masks M_d with random holes for each image x in the minibatch.
    Update the discriminators D with binary cross entropy loss with both (C(x, M_c), M_c) and (x, M_d).
    if t > T_c + T_D then
      Update the generator network C with joint loss gradients using (x, M_c) and D.
    end if
  end if
end while
```

In the generator training procedure, initially when generator is poor, discriminator can reject samples with high confidence. This necessitates that initially rather than minimizing $log(1 - D(G(z)))$, maximize $log(D(G(z)))$. The discriminator network is updated by updating in the opposite direction so that the loss increases. As the discriminator network consists of both the local and global context discriminators, the gradients splits into the two networks and then merges into the generator network. ADADELTA algorithm is used to update and optimize the network.

4.4 Design Changes and Improvements made in the Implementation

1. Generator architecture modification - Implemented Fully Convolutional Network based on the paper **Fully Convolutional Networks for Semantic Segmentation**, instead of the architecture proposed in paper. The generator architecture now consists of convolution layers and dilated convolution layers. Instead of pooling, convolution layer with stride 2 is used. Then deconvolution layers (convolutional layers with fractional strides) are used to recover the original resolution. Links that combine the final prediction layer with lower layers with finer strides are added. This is done to predict finer details while retaining high-level semantic information. This should help combine coarse, high layer information with fine, low layer information which should ideally help in generating much finer images than simple architecture used in the paper.

2. Semi-Supervised Learning - We output a K-dimensional vector of logits using a standard classifier. We can thus use supervised learning by training the model by minimizing the cross-entropy between real and generated image vectors. In sum, the main idea is training a discriminator playing both the roles of a classifier performing image

classification task as well as trained to distinguish generated samples produced by a generator from the real data. To be more specific, the discriminator/classifier takes an image as input and classified it into $n+1$ classes, where n is the number of classes of a classification task. True samples are classified into the first n classes and generated samples are classified into the $(n+1)^{th}$ class.
The training function for discriminator will now become

$$L = L_{supervised} + L_{unsupervised}, \tag{5}$$

where,
$L_{supervised}$ = cross entropy loss,
$L_{unsupervised}$ = normal GAN loss,

For the CelebA dataset, we used the annotation points for the supervised learning of the discriminator. The training procedure remains same as before with discriminator loss now getting changed.

3. Normalizing the input - We normalized the input to be from -1 to 1 rather than 0 to 255. This significantly reduced the reconstruction loss as output of generator is through tanh layer.

4. Modified Loss Function - In GAN papers, the loss function to optimize G is $min(log(1-D))$, but in practice and as mentioned in the paper Goodfellow et. al (2014), $max(logD)$ is used for training as the first formulation usually has vanishing gradients early on.

5. Batch Normalization - In order to facilitate the propagation of gradients through the network during training, batch normalization layers are used after all convolutional layers except for the last layers of both the completion and the discriminator networks. This normalizes the output of each layer using output statistics that are updated online.

6. Single Sided Label Smoothing - In this technique, 0 and 1 targets for a classifier are replaced with smoothed values, like .9, as it reduce the vulnerability of neural networks to adversarial examples Smoothing 0 is problematic because, in areas where data is approximately zero, erroneous samples have no incentive to move nearer to the data. Thus, only positive labels are smoothed to 0.75, leaving negative labels set to 0.

5 Evaluation

The code has been done using Tensorflow from scratch as no implementation has been provided for this paper. The weights have been trained from scratch. The parameters used for training are as follows:

1. Learning rates - (i) Discriminator Optimizer - 0.001, (ii) Reconstruction Loss Optimizer - 0.001, (iii) Generator + Reconstruction Loss Optimizer - 0.001

2. Train periods - (i) T_{train} - 50000, (ii) T_C - 9000, (iii) T_D - 1000

3. $\alpha = 0.0004$

4. Batch Size - 64

5. Optimizer - ADADELTA optimizer

6. Weight decay - 0.0004

5.1 Dataset Preparation

The dataset has images of 178*218 pixels, but we used 160*160 centrally-cropped image patches for training instead of using 256*256 image patches. For dataset pre-processing, the images in the dataset are de-meaned by calculating the mean of all the 200k images and then subtracting it from each image. Also, the generator network overwrites the completion region of the training input image by a constant colour, which is the mean pixel value of the training dataset, before putting it into the network in order to make the training easier.

6 Results

The author of the paper trained the net for 2 months on a single machine equipped with four K80 GPUs. Thus this net requires excessive training. There are no performance metric defined for Generative Adversarial Networks. The authors of the paper compared the output of their network with existing approaches to demonstrate the performance of their approach. They also conducted a user study with 10 users to calculate the percentage of images deemed to be real by users. In our case, as we could not train the model a lot due to debugging issues, our results are successfully able to localize the facial features but the results are very blurry. The results of the paper are shown in Fig 4 and result of our implementation are shown in Fig 9.

Figure 4: Image showing the result of the paper on Faces.

The first time we trained the network for 5 days on Nvidia Titan X GPU, generator loss first dipped and then kept increasing, while discriminator's loss kept decreasing. We think the discriminator got too strong relative to the generator. Beyond this point, the generator finds it almost impossible to fool the discriminator, hence the increase in it's loss. This can be seen in Fig 5.

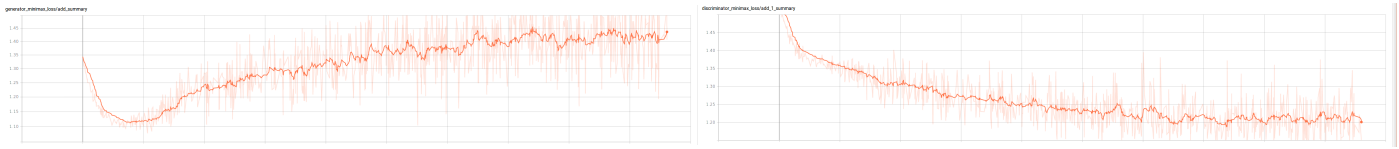

Figure 5: Generator and Discriminator Loss for the first round of training of our network. The Generator Loss first decreased and then kept increasing, while discriminator loss kept decreasing.

The final network has been trained for 4 days on Nvidia Titan X GPU. We require a lot more training on the network as the results are blurry. The loss plots for the final run are shown below in Fig 6, 7, 8:

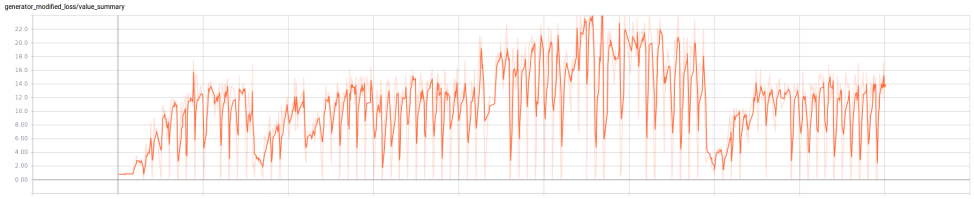

Figure 6: Snapshot of plot showing generator loss. Generator loss can't decrease or increase steadily. In the first case the generator becomes strong relative to discriminator while in the later discriminator becomes stronger than generator. This is the reason loss oscillates.

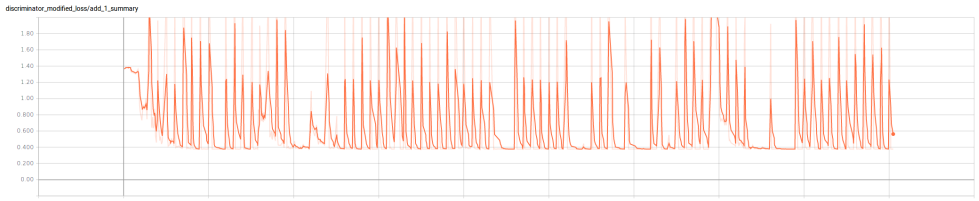

Figure 7: Snapshot of plot showing discriminator loss. Discriminator loss can't decrease or increase steadily. In the first case the discriminator becomes strong relative to generator while in the later generator becomes stronger than discriminator. This is the reason loss oscillates.

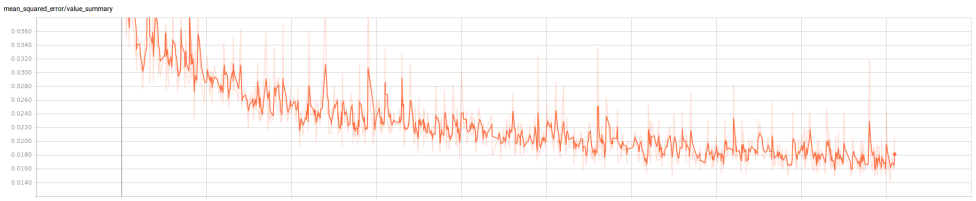

Figure 8: Snapshot of plot showing reconstruction loss. This is mean squared error in constructed and real image and decreases steadily with iterations as per expectation.

Figure 9: Image showing our results. First row is the input image to the network. Second row is the generated image from the generator and third row is the ground truth. The images are blurry and require more training but are able to estimate the position of facial features correctly. The completed regions have subtle inconsistencies with surrounding regions and requires blending as post processing.

7 Analysis and Conclusion

GAN are extremely tough to train due to stability issues. This was seen when we first trained the network in Fig 5. The discriminator got too strong relative to the generator. Beyond this point, the generator finds it almost impossible to fool the discriminator, hence the increase in it's loss. When successfully trained, GAN's can be used successfully for image in-painting. The net needs to be trained much further to get accurate results as in paper.
Significantly large holes cannot be filled in due to lack of spatial support of the model. Thus we have limited the maximum size of mask on an image. Also, the masked region should have spatial support on all sides. If the mask is on one side, the generated image does not have realistic feature locations. Thirdly, to use this network on an image other than faces, the network needs to be trained again on a similar dataset to get good results.
Blending techniques have not been used on the finally generated images and thus the completed regions have subtle colour inconsistencies from the surrounding regions. To avoid this post-processing can be done by blending the completed region with the color of the surrounding pixels by using fast marching method or Poisson image blending techniques. Secondly, the architecture of the generator can be made more deeper and more dilated convolutions and links from lower layers can be added to get more richer completed region.
The implementation of the code can be found at `https://github.com/akashr050/img_inpaint.git`.

References

[1] Satoshi Iizuka,Edgar Simo-Serra, Hiroshi Ishikawa. *Globally and Locally consistent Image Completion.* ACM Transactions on Graphics, 10.1145/3072959.3073659, 2017.

[2] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio *Generative Adversarial Nets.* arXiv, 1406.2661, 2014

[3] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen. *Improved Techniques for Training GANs.* arXiv, 1606.03498v1, 2016

[4] Jonathan Long, Evan Shelhamer, Trevor Darrell. *Fully Convolutional Networks for Semantic Segmentation.* arXiv, 1411.4038, 2014

[5] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, Alexei A. Efros. *Context Encoders: Feature Learning by Inpainting.* arXiv, 1604.07379, 2016.

[6] Alec Radford, Luke Metz, Soumith Chintala *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks.* arXiv, 1511.06434, 2016

[7] Globally and Locally Consistent Image Completion,
`http://hi.cs.waseda.ac.jp/ iizuka/projects/completion/en/`

[8] How to Train a GAN? Tips and tricks to make GANs work,
`https://github.com/soumith/ganhacks`

[9] Large-scale CelebFaces Attributes (CelebA) Dataset,
`http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html`

[10] [CVPR 2016] Unsupervised Feature Learning by Image Inpainting using GANs,
`https://github.com/pathak22/context-encoder`

[11] A tensorflow implementation of "Deep Convolutional Generative Adversarial Networks,
`https://github.com/carpedm20/DCGAN-tensorflow`

[12] TensorFlow-Slim,
`https://github.com/tensorflow/tensorflow/tree/master/tensorflow/contrib/slim`

[13] CS231n Convolutional Neural Networks for Visual Recognition,
`http://cs231n.github.io/convolutional-networks/`

[14] Dilated Convolutions and Kronecker Factored Convolutions,
`http://www.inference.vc/dilated-convolutions-and-kronecker-factorisation/`