Image inpainting: Self supervised learning

Gopikrishna Rathinavel
rgopikrishna@vt.edu

Rajesh Kudupudi
rajeshk19@vt.edu

Abstract

We propose a Generative Adversarial Network for image in-painting task. This task does not require any labels, hence the name self-supervised. For solving the in-painting task we follow context encoder (Pathak et al [6]) like model, in which a part of the image is masked out and our network should learn to predict the contents of the masked out region, conditioned on it's surroundings. We experiment with different different network architectures and loss functions for solving this task. We compare different models and show how our skip connection based network with a local and a global discriminator is able to learn realistic finer looking details of an image. We also show that features learned by our Convolutional Neural Network, CNN (generator) by solving this in-painting task can be directly transferred to aid a classifier (with pre-trained parameters) on its classification task.

1 Introduction

Volume of data is increasing at an exponential rate in the current world. There is abundance of data already but what we consume for useful purposes is only a very small amount compared to the actual available volume. Most machine learning related applications require labels for the gathered data in order to build a model. The limitations of deep learning that are being pointed out these days are more related to the limitations of supervised learning's need for annotated data. Many deep learning methods that have been implemented and used in real world for example, image classification, segmentation, facial and speech recognition are all part of supervised family of algorithms all of them being formed from a labelled set of data. It is evident how labelled data is important in real life applications. Labeling is labor intensive and time consuming and only a very small group of people have the interest and expertise to create annotations to datasets. This renders a vast portion of data to go waste.

This is where self supervised learning comes into play. It is basically leveraging the labels that are by default present in the data to design the objective function. Effectively it is training unsupervised data in a supervised way. In computer vision, image inpainting is one such popular problem. In image inpainting, a portion of the image is cropped and the cropped portion is generated using various generative algorithms the most widely used being Generative Adversarial Networks [1]. Recently NVIDIA has developed a tool which can be used to correct portions of an image that one feels defective [2]. For example, smoothing of skin in faces. With this tool, one can erase a portion of image, and the GANs reconstruct the deleted portion of the image.

Self supervised task also usually called the pretext task, provides the supervised loss function. In most cases, the accuracy of this model itself is not very important. Rather, we hope that the task helps the model learn good semantic and structural features so that those can be utilized in a further downstream task [3]. Take various transformations for example. We can apply various transformations to an image and try and predict the kind of transformations applied to the image. This is a self supervised learning problem as the labels are coming from transformations of the image. The accuracy of this task is not very important as the transformations are made up. But it can be expected that the model learns high level details of the image and can be used for some downstream task such as object detection.

One might wonder are GANs a class of self supervised algorithm? In a broad sense this is true,

Figure 1: Original Image (Left) and Corrupted Image(Right): Corrupted in the center

however the end goal for GANs and other self supervised algorithms are quite different. GANs try to create realistic images whereas self supervised algorithms try to good features.

2 Related Work

Convolutional neural networks (CNNs) have become vital in the field of computer vision. This has further led to improvements in new areas of computer vision such as image generation. Generative adversarial networks (GANs) specifically have excelled in the task of generating real like images.

Pathak et al [6] have performed the same task as we are performing but using a different model architecture. They have used AlexNet followed by fully connected layers in the encoder, then a channel wise fully connected layer and then a series of upconvolutions in the decoder. Their pretask was visual inpainting and they performed a variety of downstream tasks such as classification, detection and segmentation.

One criticism that the above architecture has is that there is a bottleneck layer. This is disadvantageous as low level features can possibly be lost during the process. To avoid this, Ronneberger et al proposed U-net [7]. It was developed for biomedical image segmentation. U-net is characterized by something called a skip connection. In skip connections, features from each layer of the encoder is concatenated to the corresponding layer of the decoder. These skip connections pass information on top of the bottleneck layer and helps to retain features from regions outside the cropped region. We utilize this architecture in the inpainting task.

3 Problem Statement

We know that CNNs are data hungry and tend to do well on any task when provided with a lot of labelled data for that task. But in cases like medical and other related fields large labelled datasets are not always available. Annotating images is labour intensive, time consuming and might be expensive. Often we have access to a lot of unlabelled data available to us. How do we make use of unlabelled data such that our CNNs learn something useful from it? The answer is self-supervised learning Lilian [3]. In self-supervised learning data provides supervision.
In self-supervised learning approach we have a pretext task and a downstream task. The pretext task is to learn good feature representations, both structural and semantic. We then transfer the parameters learned on the pretext task in a downstream task which is image classification in our case . Pretext task is framed such that we do not need any labels for it. The pretext task which we choose to solve is inpainting.
In inpainting, given an image we hold back some portion of the image and train our network such that it predicts this part of the image. Here, the hidden portion of the images serves as a supervision signal. Humans generally can guess the missing region because we understand the context of the image. In order for CNN to solve this task effectively our network should learn good features. We then transfer the parameters learned by training on this pretext task to image classification task. So, we will be providing our algorithm's performance on inpainting and image classification tasks.

4 About the Dataset

We have used two datasets for the two tasks respectively.

- COCO image dataset [4]
- Pascal VOC 2012 [5]

COCO image dataset was released by Microsoft in 2015. It is usually used for object detection and segmentation tasks. It has 300k images with more than 200k annotated images. We however are using only the training set of year 2014 which consists of >100k images. We use this dataset for the image inpainting task.
PASCAL VOC 2012 is another standardized image dataset for object class recognition. It consists of 17k images which fall into 20 classes. These 20 classes can be categorized into four main classes which are person, animal, vehicle and indoor. This dataset was used for the classification task.

5 Methodology

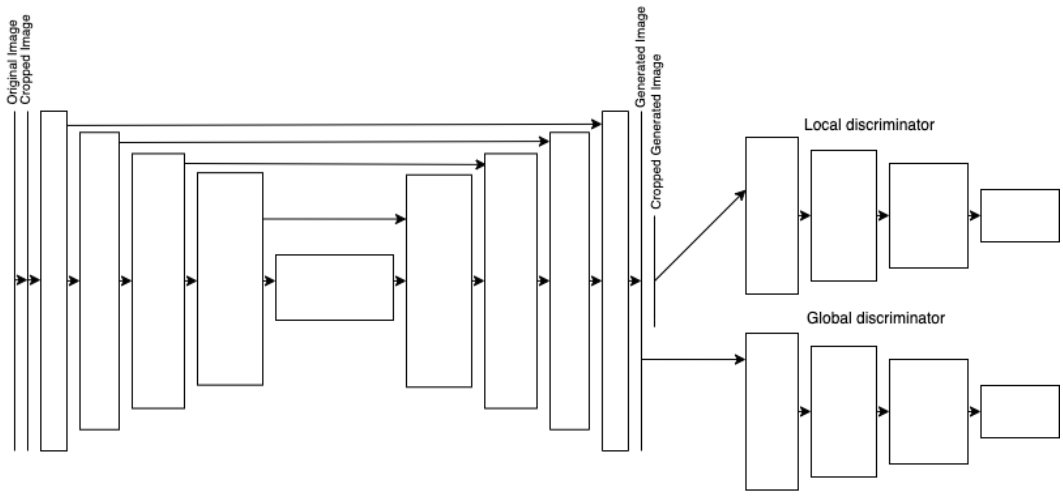

Figure 2: Model architecture (Model-M1)

We first give an overview of our model's general architecture for in-painting task, which consists generator, discriminator and different loss functions. In the next section we comment on how the pre-trained network parameters which we learn by solving the in-painting task can be used to aid classifiers. Our proposed model for in-painting task uses DCGAN architecture with skip connections, with a global and a local discriminator. We present these components in great detail below.

5.1 Generator

The overall architecture of the generator is a simple encoder decoder pipeline.The encoder mainly consists of several convolutional blocks with batch normalization and leaky ReLU. The encoder takes in an input image with masked regions and computes it's latent feature representation. The decoder takes in this latent feature vector and tries to produce the entire image along with the missing region. Decoder consists of several transpose convolutional blocks with batch norm and leaky ReLU units. The size of the input taken in by the encoder and the output of decoder are of same size. We hypothesize here that encoder learns a good latent representation of the image along with the missing regions from the context of pixels around the missing regions. Then the decoder tries to reconstruct the original image back from the latent representation. But there is a small problem with this encoder decoder pipeline. Since all the network information between the encoder and decoder has to pass through the latent feature representation or the bottleneck layer, there is a chance that a great deal

of low level image information might be lost (like edges, color and so on). So, we include skip connections between the encoder and decoder to circumvent this problem. We follow the general shape of "U-Net" architecture [x]. So, if a generator has totally n layers then we add skip connections between layer *i* and layer *n-i*. By skip connection we mean that we would append all the output channels of layer *i* with those at layer *n-i*. Table 1 below summarizes the architecture of our generator.

Layer	Kernel size	Stride	Filter size	Activation
Conv2d	4	2	64	Leaky ReLu
Conv2d	4	2	64	Leaky ReLu
Conv2d	4	2	128	Leaky ReLu
Conv2d	4	2	256	Leaky ReLu
Conv2d	4	2	512	Leaky ReLu
Conv2d	4	1	4000	Leaky ReLu
T.Conv2d	4	1	512	ReLu
T.Conv2d	4	2	256	Leaky ReLu
T.Conv2d	4	2	128	Leaky ReLu
T.Conv2d	4	2	64	Leaky ReLu
T.Conv2d	4	2	64	Leaky ReLu
T.Conv2d	4	2	3	tanh

Table 1: Generator architecture.

As we mentioned earlier each layer is followed by a batch normalization layer and a Leaky ReLu layer of $\alpha = 0.2$. T.Conv2d stands for transpose convolutions.

5.2 Discriminators

Our model consists of 2 discriminators, a local and a global discriminator. Local discriminator is to distinguish real patch or masked region from the generated one. And the global discriminator differentiates between the entire real and generated images. We have a global discriminator because we wanted our generator to generate good realistic images, in an attempt that it might help with the predictions in masked region also. We had a local discriminator to give feedback to the generator only about the predictions in masked region, as this is what we really care about. In Table 2 below we have provided the architecture of global and local discriminators.

Layer	KernelSize	Stride	FilterSize	Activation
Conv2d	4	2	64	Leaky ReLu
Conv2d	4	2	128	Leaky ReLu
Conv2d	4	2	256	Leaky ReLu
Conv2d	4	2	512	Leaky ReLu
Conv2d	4	2	512	Leaky ReLu
Conv2d*	4	2	1	Sigmoid

Table 2: Global and local discriminator architecture.

The layer $Conv2d^*$ will not be present in the local discriminator.

5.3 Loss Function

We train our generator by regressing on ground truth of the missing region, original image (reconstruction loss) and adversarial loss. The reconstruction loss which is an L2 loss in our case is responsible for replicating the overall structure of missing region with respect to its context. But as explained in [1] using only reconstruction loss tends to produce blurry patches (this fails to capture high frequency components). The adversarial loss is responsible for the high frequency components, which makes

Figure 3: In each block, Leftmost image corresponds to real image, Center one is corrupted image, Right one is reconstructed image (central patch there is predicted by our generator). These images are generated by training Model-M1 shown in Figure [2].

the predictions look real. There are several components in our loss which are listed below:

$$\mathcal{L}_{MSE}^{full} = \frac{1}{N}\|(I_{gen} - I_{orig}) \cdot (1 - M)\|^2$$

$$\mathcal{L}_{MSE}^{masked} = \frac{1}{N}\|(I_{gen} - I_{orig}) \cdot M\|^2$$

$$\mathcal{L}_{MSE}^{total} = w_1\mathcal{L}_{MSE}^{full} + w_2\mathcal{L}_{MSE}^{masked}$$

$$\mathcal{L}_{D_l} = -\log D_l(I_{orig} \cdot M) - \log\Big(1 - D_l(I_{gen} \cdot M)\Big)$$

$$\mathcal{L}_{D_g} = -\log D_g(I_{orig}) - \log\Big(1 - D_g(I_{gen})\Big)$$

$$\mathcal{L}_G = \mathcal{L}_{MSE}^{total} - w_3 \log(D_g(I_{gen})) - w_4 \log(D_l(I_{gen} \cdot M))$$

Here, N denotes the total number of elements in that corresponding norm. It is used to compute the mean. The "$\cdot$" is the dot operator. I_{gen} is the generated image, which is produced by the generator G, such that $I_{gen} = G(I_{orig} \cdot (1 - M))$. M is the mask used for cropping the image, and I_{orig} is the original image. $\mathcal{L}_{MSE}^{full}$, $\mathcal{L}_{MSE}^{masked}$ and $\mathcal{L}_{MSE}^{total}$ correspond to mean squared errors for total image (excluding the masked portion), masked portion and their weighted sum respectively. Here w_1, w_2, w_3, w_4 are the weights given to each of these loss functions. We experiment with many different values of these weights. D_l is the local discriminator and D_g is the global discriminator. $\mathcal{L}_{D_l}, \mathcal{L}_{D_g}$ correspond to the adversarial losses for local and global discriminator respectively. $\mathcal{L}_G$ is the generator loss.

5.4 Other explored Architectures

We also experiment with 2 different architecture, you can think of this as an ablation study. One, of these architecture is the same as the in shown in Figure 2, but without global discriminator. Lets call this (Model-**M2**). Another model architecture (Model-**M3**) does not have any skip connections or global discriminator, its generator also just predicts the patch instead of the entire image. (Model-**M1**) is our original model Figure 2.

6 Results and Experimental Analysis

6.1 In-Painting

We have run all our experiments for image in-painting on COCO-2014 dataset. It had >100000k images. We ran our model proposed in Figure 2 (Model-M1). While training we used an Adam

Figure 4: Leftmost image corresponds to real image, center one is corrupted image, right one is reconstructed image. Behind the central person in the reconstructed image we can see an artifact.

Figure 5: Inpainting: Rightmost image generated by Model-M2. We can see the blurry patch.

optimizer for both the discriminators and the generator with a learning rate of 1e-3 for discriminators and 2e-3 for generator. We experiment with many values for w's for our loss and finally settle on $w_1 = 0.9, w_2 = 0.1, w_3 = 0.00001, w_4 = 0.00001$. We gave very less weight to adversarial loss because when we tried higher weights, we were getting random artifacts which are unrelated to the context of the image Figure 4. We ran the entire training of M1 for 13 hours on Nvidea 2070 SUPER GPU. The results of our model-M1 run on in-painting task are provided in Figure 3. As we can infer from the images our model-M1 was able to reproduce good crisp predictions in the masked region. As discussed by Yu et.al.[8] there is no clear candidate for qualitatively evaluating our generated images. So, looking at the results generated we feel that our model performs reasonably well.

In Figure 5 we show the results generated by model-M2 (same as our model but without global discriminator). As we can see the image is too blurry. We find this trend that images without global discriminator are blurry. We hypothesized that without global discriminator our generator will not have global adversarial loss, which might be causing the lack of high frequency components in the generated image patches.

In Figure 6 we show the results generated by model-M3, which does not have any skip connections and generator here predicts only patch instead of the entire image. As we can see by comparing Figure 6 and Figure 3, our model-M1 generates better crisper and realistic looking patches. This may be due to the lack to skip connections in Model-M3

Figure 6: Inpainting: Rightmost image generated by Model-M3. We can see the image quality is bad in comparison to images generated by Model-M1

6.2 Classification

Model	Training Accuracy (%)	Testing Accuracy (%)
With weight initialization from pretrained model	75.5%	57.4%
With random weight initialization	75.8%	55.1%

Table 3: Accuracy scores

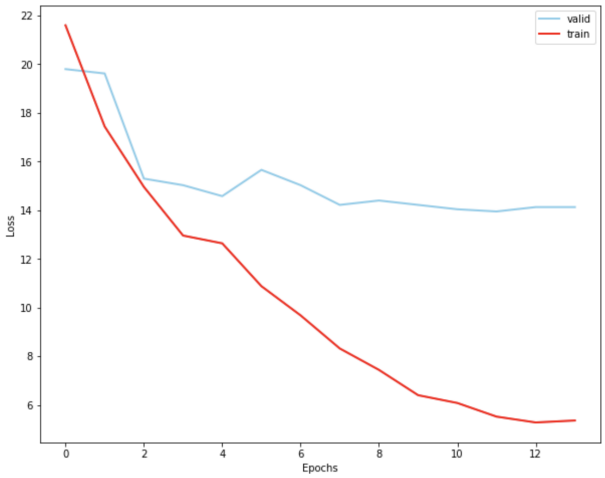

Figure 7: Training history for classification (with weight initialization from generator trained on in-painting task)

Now we plan on transferring the weights of the generator (encoder inside generator) to a classifier, and fine tune it on Pascal VOC 2012 dataset. The architecture of the classification model that we have implemented is very similar to the generator (only until encoder), but with final few fully connected layers at the end of the model for fine tuning to make classification possible. Pooling layers are added in between the fully connected layers to help with over-fitting. The model was run twice, one with weights initialized from the trained generator (encoder only) model obtained from the inpainting task and the other time with random weight initialization. The input image size was 128x128 pixels. Fixing the hyperparameter was challenging as we run into overfitting issues. We introduced pooling layers along with fully connected layers and we add regularization also. The batch size used was 64. We used stochastic gradient descent along with momentum of 0.9 as the optimizer. Learning rate of 0.005 was used. Total number of epochs was 14.
The accuracy scores we achieved are shown in Table 3. Training history is shown in Figure 7. Weight initialization from the model learnt from the inpainting task did not have any effect on the performance of the model on the training set. However, there was a small increase in performance in the testing set.

7 Conclusion and Future work

We were able to use various GAN architectures including Unet for the generator, to perform visual inpainting. The images we generated are real like and one would find it hard to notice many differences between the real and generated images.
We also were able to achieve better performance on classification after weight initialization using pretrained weights from the generator from the inpainting task.
In the future, partial convolution architecture could be tried in the inpainting task. Also Wasserstain GAN which overcomes some of the problems faced in general GANs can be used in the inpainting task.

References

[1] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... Bengio, Y. (2014). Generative adversarial nets. Advances in Neural Information Processing Systems, 3(January). https://doi.org/10.3156/jsoft.29.5.177.2

[2] NVIDIA Inpainting: https://www.nvidia.com/research/inpainting/

[3] https://lilianweng.github.io/lil-log/2019/11/10/self-supervised-learning.html

[4] Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... Zitnick, C. L. (2014). Microsoft COCO: Common objects in context. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 8693 LNCS(PART 5), 740–755. https://doi.org/10.1007/978-3-319-10602-1.48

[5] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL visual object classes (VOC) challenge," IJCV, vol. 88, no. 2, pp. 303–338, Jun. 2010

[6] Pathak, D., Krahenbuhl, P., Donahue, J., Darrell, T., Efros, A. A. (2016). Context Encoders: Feature Learning by Inpainting. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2016-December, 2536–2544. https://doi.org/10.1109/CVPR.2016.278

[7] Ronneberger, O., Fischer, P., Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 9351, 234–241. https://doi.org/10.1007/978-3-319-24574-4.28

[8] Yu, J., Lin, Z., Yang, J., Shen, X., Lu, X., Huang, T. S. (2018). Generative Image Inpainting with Contextual Attention. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 5505–5514. https://doi.org/10.1109/CVPR.2018.00577