Outpainting using Context Encoders and GAN

Shiwangi Singh
Electronic Visualization Laboratory University of Illinois at Chicago
ssingh79@uic.edu

Shi Yin
Electronic Visualization Laboratory
University of Illinois at Chicago
syin8@uic.edu

ABSTRACT

Image Extrapolation is necessary when parts of the image is missing and requires filling those missing gaps, we call this process as Outpainting. Semantics of a painting can be captured by neighbourhood pixel values which are basically contexts in an image. We build a model similar to semantic Inpainting except that we expand the image instead of filling in the missing gaps. We developed an extension of semantic Inpainting where we can retrieve image content around the borders and thus expand the image. Additionally, we built an interactive web interface where users can upload an image and test the model's capability to expand an image.

Categories and Subject Descriptors

I.4.9 [**Image Processing and Computer Vision**]: Application; I.5.1 [**Pattern Recognition**]: Models—*Neural Nets*

General Terms

Algorithms, Design, Theory

Keywords

Deep Learning, Image Inpainting, Outpainting, Generative Adversarial Networks, Context Encoders

1. INTRODUCTION

We have copious amount of historical images, black and white images and even digital images which have been deteriorated because of mishandling and corruption. Inpainting techniques are used to restore these deteriorated images by filling gaps. There are parametric and non-parameters methods in which one perform inpainting. Techniques that use *Convolutional Neural Network* (CNN) are categorized as parametric approaches to inpainting. Due to the recent advances in the field of Image Generation [1], it only makes sense to use CNN and its variates to image inpainting.

Unsupervised deep learning has recently made it possible to answer questions pertaining image processing and image analysis such image denoising, image synthesis, image inpainting and so on. *Generative Adversarial Networks* (GANs) were developed by Goodfellow et al. that allowed image generation in an unsupervised manner [2]. Pathak et al. used the advantages of Context Encoders and GANs for semantic inpainting [4]. Yeh et al. used contextual and perceptual loss in a Deep Convolutional GAN (DCGAN) like architecture for semantic inpainting further validating the importance of generative networks for inpainting [5]. This implementation was based on Pathak's work except that the use case is image expansion instead of hole filling inside the image.

2. CONTEXT ENCODER AND GAN

2.1 Architecture

2.1.1 *Context Encoders*

There are two models used together for context inpainting. The first is the context encoder which is like a variation of denoising autoencoder for filling the missing region using the contextual information. Context encoder is an encoder-decoder model where encoder is used to learn high level representation of the data and the decoder is used to reconstruct the image from this abstract representation. As seen in Figure 1 we have fully connected layers between encoders and decoders which takes a shape of 4000 units long vector. The input to the encoder is an image with missing borders and the output of the decoder is the filled borders. This output is regressed against the actual border cut outs. A binary mask is used for identifying the border regions which are dropped from the actual input to the model.

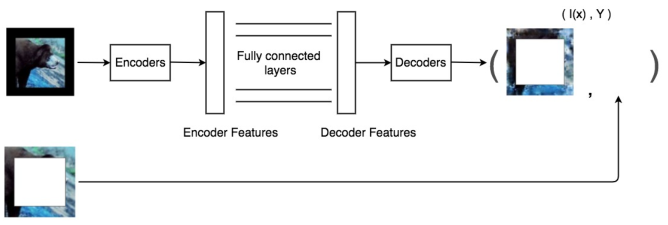


Figure 1: Encoder-decoder model showing reconstruction loss

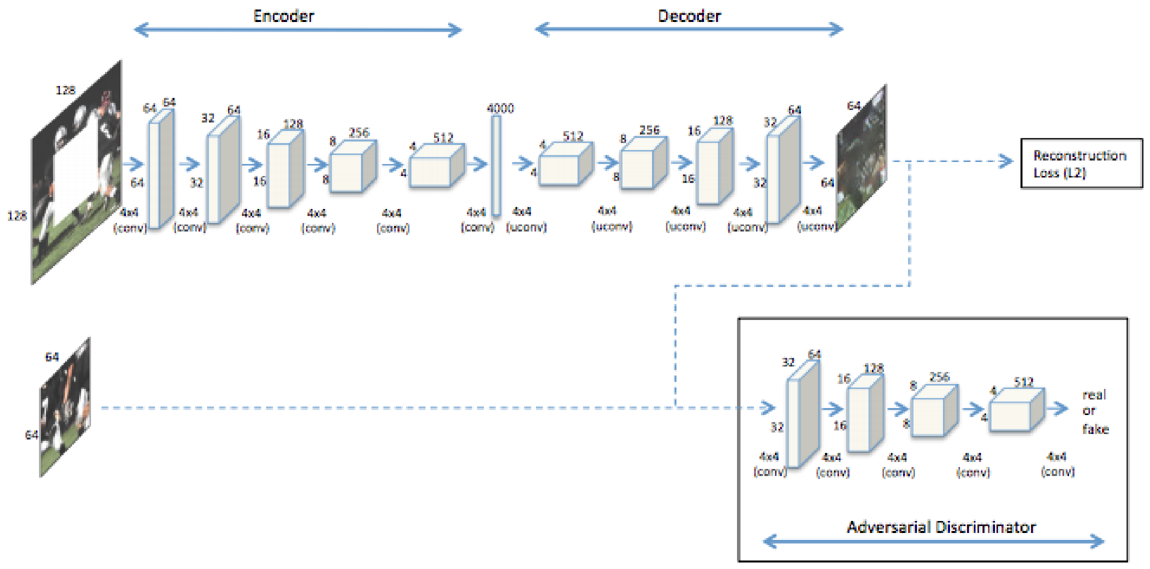


Figure 2: Model architecture as in [4] except our inputs were as explained in Section 3

2.1.2 *Generative Adversarial Networks (GANs)*

The output obtained from the context encoder model is not as sharp as we would like to achieve. To overcome the blurriness of the output and get a sharper result, we make use of GANs as seen in [2]. GANs consists of a Discriminator D and a Generator G. The role of the Generator is to map a noisy distribution Z to a data distribution X, this can be written as $G : Z \to X$. The role of the discriminator is to distinguish between the actual ground truth data and the predicted data by G. They basically play a game where D tries to correct the model output and G tries to confuse D by showing the 'fake' data as 'real'. The following equation shows this mini-max game between G and D:

$$\min_G \max_D \mathbb{E}_{x \in \mathcal{X}}[\log(D(x))] + \mathbb{E}_{z \in \mathcal{Z}}[\log(1 - D(G(z)))]. \quad (1)$$

For modeling the GANs, we use Generator as the Context Encoder model, and a adversarial Discriminator to identify between the real (ground truth) and the fake (decoder output) data. This can be seen in the bottom model in Figure 2.

The architecture is adapted from AlexNet architecture [3] with five convolutional layers with stride equal to two and five corresponding deconvolutional layers of hidden sizes (64, 32, 16, 8, 4). The filters used are of size 4*4. Figure 1 shows the encoder and decoder shapes for the context encoder. The second type of model used for outpainting is GAN, where the input for the Discriminator comes from the context encoder model which is the predicted output borders and the ground truth borders, see Figure **??**. Again, the shape of the adversarial network consists of four convolutional layers taking a hidden shape of (32, 16, 8, 4) with 4*4 filters convolving the image.

2.2 Loss Functions

They use two kinds of losses used in the context encoder plus GANs model. The first is the reconstruction loss λ_{rec} which is the L2 loss obtained from the Context encoder while regressing the context encoder model output to the actual missing borders. These losses are applied separately to the inputs and the missing regions. Thus a binary mask is used in the loss functions. The second kind of loss is the adversarial loss which is obtained from the discriminator. L2 tends to average together multiple modes in prediction whereas adversarial picks the mode from the distribution making the prediction look more real. Loss equations can be given as follows:

$$L_{rec}(x) = ||M(x - F(1 - M)x))||, \quad (2)$$

$$L_{adv} = \max_D \mathbb{E}_{x \in \mathcal{X}}[\log(D(x)) + \log(1 - D(F((1 - M)x)))], \quad (3)$$

and

$$L = \lambda_{rec} L_{rec} + \lambda_{adv} L_{adv}, \quad (4)$$

where L_{rec} is the total reconstruction loss, L_{adv} is the adversarial loss and L is the final joint loss.

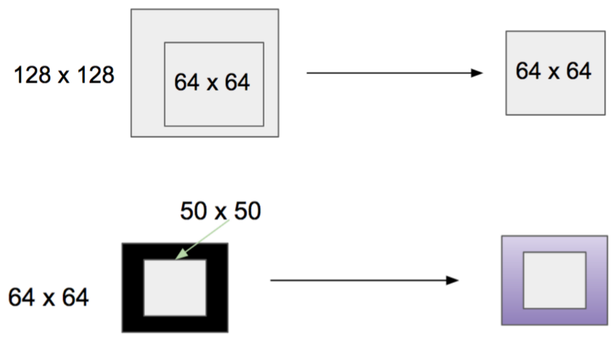


Figure 3: Image cropping: (a) the random 64*64 selection from the resized image; (b) the missing borders (black) and the filled borders (purple)

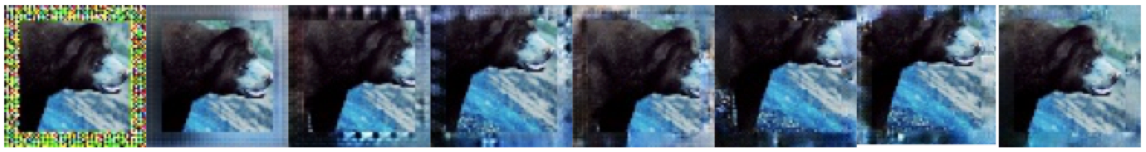


Figure 4: Results obtained at various timestamps during the training process

3. IMPLEMENTATION

Our system, written in Python and TensorFlow, consists of the following four modules: Image Loader module, Image Cropping module, GAN module, and UI module.

3.1 Image Loader

Our system firstly takes a set of images as input, and feeds it to *Image Loader* module.

Image Loader module crops the maximum square region from the center of each input image. In other words, it ensures 1) the width (and height) of the cropped image is either the width or height of the original image, whichever is smaller, and 2) the center of the cropped image is also the center of the original image.

The cropped image then is scaled to 146*146. It randomly selects an 128*128 region from this 146*146 image. The RGB values are then mapped from $(0, 1)$ to $(-1, 1)$ linearly.

The set of these 128*128 images, noted as $TrainImage_{ori}$, is the output of Image Loader module, and is fed to Image Cropper module.

3.2 Image Cropper

Image Cropper module's main functionality is two-fold. First, it produces input image set ($TrainImage$) for the generator by randomly picking a 64*64 square region in each image, and leaving the central 50*50 square of the 64*64 region intact while assigning black ((117, 104, 123) is used in our case) to the rest. Note that there is a 7-pixel wide 'border' area in the 64*64 square region that is also assigned black color. This process is illustrated in Figure 3. Second, it also produces input image set ($TrainCrop$) for the discriminator. The input set is simply the 64*64 square region with original values.

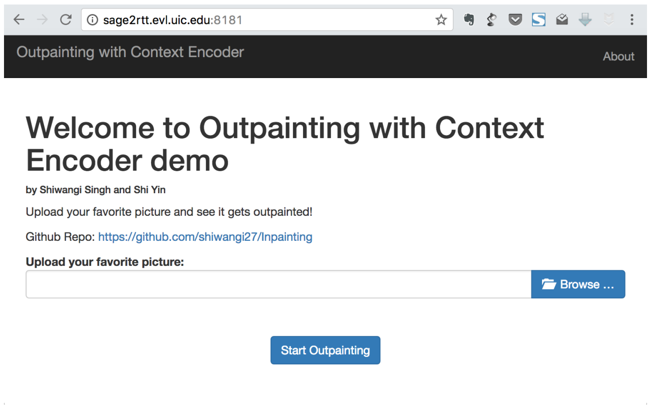
3.3 GAN module

GAN module is where our core generative adversarial network resides, where our Generator is nothing but the Context Encoder model. The generator has five convolutional layers, with a Leaky ReLU layer after each of them. There there are five deconvolutional layers, with a ReLU layer after each one. The Discriminator, on the other hand, has four convolutional layers, also with a Leaky ReLU layer after each.

The output of each one of the models is used for computing the generative and the adversarial loss. Original reconstruction loss is calculated using the L2 loss function. Then using binary masks, the losses are calculated separately for the overlapping and the borders regions which are then summed together to obtain total Reconstruction loss. The objective function used in the GANs are sigmoid cross entropy with logits both for generative and adversarial parts of the model. The final loss is the joint loss making use of reconstruction loss and adversarial loss at the same time. The lambda values used for the reconstruction loss and for the adversarial loss are 0.9 and 0.1, respectively. The update function used is the Adam Optimizer with a learning rate of 0.0003 and a weight decay rate of 0.00001. The model is trained for 2000 epochs for approximately 48 hours.

3.4 User Interface

All the above-mentioned modules are for back-end computation. UI module is for front-end user interaction. The purpose of having a user interface is to allow users to upload their own images and view the output so that they have an intuitive way to evaluate our model.



sage2rtt.evl.uic.edu:8181

Outpainting with Context Encoder

About

Welcome to Outpainting with Context Encoder demo

by Shiwangi Singh and Shi Yin

Upload your favorite picture and see it gets outpainted!

Github Repo: https://github.com/shiwangi27/Inpainting

Upload your favorite picture:

Figure 5: User Interface of our system

Figure 5 is an overview of our user interface. Users are allowed to upload any image file that has an extension of 'png', 'jpg' or 'jpeg' from their local machine. Upon clicking on 'Start Outpainting' button, the selected file will be uploaded to the system.

This image is fed into *Image Loader* module and then *Image Cropper* module to be processed, before GAN module finally produces a generated border to the intact central region of the uploaded image.

4. RESULTS

We tested our GAN model on a data set that consists of 82,783 images retrieved from ImageNet database. We used 90% of the downloaded images as training set, while the rest 10% was used as test set.

We trained the data set for over 2000 epochs, with a learning rate of 0.0003 and a decay rate of 0.00001. Figure 4 shows the results obtained at various timestamps during the training process.

Figure 6 shows the results obtained on the test set. The borders painted are 7*7 in height and width in the 64*64

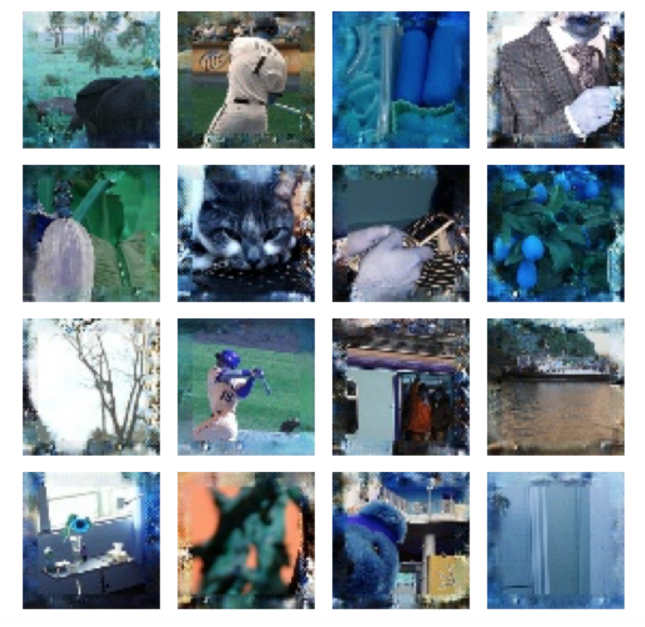


Figure 6: Results on test set after 2000 epochs

image. These borders were originally hidden or passed as black pixel values of 64*64 and the image content inside was 50*50.

Figure 7 shows the model input and output of 8 randomly chosen images. The size of the input image is 50*50 (left) and the size of the output image is 64*64 (right). This result is obtained from a saved training model trained for 370 epochs.

5. CONCLUSION

We used a context encoder model as a replacement for Generative network. This work required us to know a-priori about the ground truth which makes this model somewhat supervised. We needed an approach that was completely unsupervised for extrapolating images to a desired extent, shape and size, instead we were only able to outpaint image of fixed borders. In the future, we would like to use a GAN which is modeled to extrapolate the images and predict the neighborhood pixels using contextual information. Although, the results in that case wouldn't be as sharp as the one obtained by using context encoders. Sophisticated loss functions and trained models could be used to obtain competitive results for image outpainting. Furthermore, the interface allows us to interactively explore the capabilities of the model to generate outpainted images. This interface can be further extended to obtain outpainting of specified border shapes and sizes.

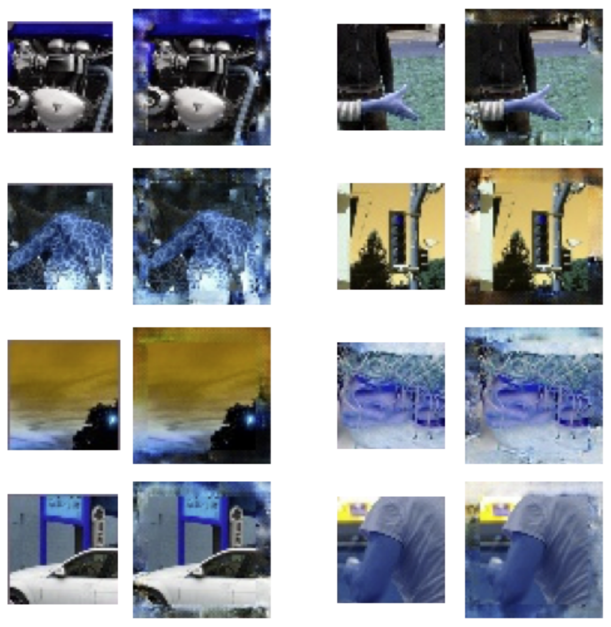


Figure 7: Comparison of model inputs and outputs

6. REFERENCES

[1] C. Doersch, A. Gupta, and A. A. Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1422–1430, 2015.

[2] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[4] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2536–2544, 2016.

[5] R. Yeh, C. Chen, T. Y. Lim, M. Hasegawa-Johnson, and M. N. Do. Semantic image inpainting with perceptual and contextual losses. *arXiv preprint arXiv:1607.07539*, 2016.