Image Inpainting with Deep Learning

Rishab Lokray
9357-3447
University of Florida
rishab.lokray@ufl.edu

Sagnik Ghosh
3343-6044
University of Florida
sagnik.ghosh@ufl.edu

Abstract

Image Inpainting refers to the reconstruction of the missing part for the images. It's an excellent application of Computer Graphics blended with Deep Learning. This has numerous application in modern world, from recovering of old damaged photographs to modern usage of removal of photo bombers in a beautiful photo. There exist old approaches which are based on computer vision, in this project we look to explore the deep learning-based approach, more precisely the Vanilla Autoencoders and Partial convolutions.

1 Introduction

The goal of the project is to predict the missing part of an image, such that the predictions are both visually and semantically consistent. To implement that, we need to step back and think how we humans achieve it. This will help us formulate the basis of a deep learning based approach. We as humans, looks at surrounding pixels to visualize the missing pixels. We can instill this approach using Deep Learning concepts. Similar to the approach we humans take, we can harness the power of CNNs to predict the missing pixels with the knowledge it gains while training. Our approach is to use CNN to formulate Autoencoder that is expected to work well with structural regions of holes in the images. Vanilla Autoencoders heavily rely on post processing and do not work quite well with holes of irregular patterns, as mentioned in this paper. [1] Therefore, alongside the standard convolution, we look to implement partial convolutions for our CNN and compare the results of both. We have chosen the **CIFAR-10** [2] dataset to perform our experiments for the task on hand. We chose a simple dataset for our experiments as concept can be applied on a toy dataset, cutting short on computational resources and quicker implementation. The choice of dataset doesn't play a crucial role, as we can add artificial deterioration to any dataset and use that to to train our neural networks.

2 Previous Work

Before the advent of Machine Learning and Neural networks a lot of image processing was done using classical computer vision approaches and is still an active field of research. A lot of algorithms like "Object removal by exemplar based image inpainting", "Posison Equation Method", "8-pixel neighbourhood" etc were used to detect and fill in missing holes and abrasions.

The main idea behind most of these classical approaches is texture synthesis in which data regarding a missing patch is collected from its neighbourhood pixels. Some techniques use concepts from fluid mechanics and differential equations and continuous edges in objects. These techniques are good while fixing backgrounds but fail to perform when objects need to be reconstructed when the surrounding pixels do not have any information. Some repetitive objects in an image can still be fixed by traditional methods but single objects cant be fixed. [3] [4]

3 Overview

For lucid understanding, the pixels in each image can be divided into two sets, the pixels which are unchanged and the pixels which the Neural Network aims to reconstruct. The unchanged pixels are served as the context, which helps in recovering of damaged pixels.

Image Inpainting is a constrained image generation problem. The network has to take a context as an input and has to produce an image of the same dimension as the missing patch. The evaluation is based on the average element wise L_2 distance between the original missing section $Y \in \mathbb{R}^{n \times n \times 3}$ and the prediction $\hat{Y} \in \mathbb{R}^{n \times n \times 3}$. For our CIFAR10 dataset, the image size n = 32. For every sample i the sample the loss is therefore,

$$L_i = \frac{1}{n^2} \sum_{m,n,o} (Y^{(i)}_{m,n,o} - \hat{Y}^{(i)}_{m,n,o})^2$$

Image Inpainting algorithms can be categorized into two sets: blind and non-blind. Blind types are where the network does not know the position and the shape of the missing area and non-blind is where the network knows the position and shape of the missing area. From our literature review, we understand that the more work has been done in the non-blind case. We will try focusing on both the categories. For the blind approach, we test a Vanilla Convolutional AutoEncoder to recover the damaged pixels. For the non-blind approach, we decide to apply Partial Convolution layers instead of the usual Conv2D to the Autoencoder.

Our main goal is to leverage the latest and successful architectures and techniques in computer graphics to build an efficient and robust inpainting neural network. We hope to achieve acceptable results in terms of L2 loss. The approach easy approach to deteriorate images is to use patches of rectangular shapes of random sizes and at random positions. But using rectangular holes, might lead the model to over-fit to the data, and also is not a practical problem, therefore we draw lines of random length and thickness.

4 Technical Description

4.1 Vanilla AutoEncoder

Auto Encoders are a special type of convolutional neural network whose job is to copy the input to the output. The working behind these is that they compress or flatten the input vector to a one dimensional vector. The first part of the system is an encoder that compresses the image and then comes the decoder that reconstructs the image from the latent space representation. Autoencoders come handy when they are restricted to learn some features of the input. This can be done by making the latent space smaller than the input vector space. This forces the system to learn more salient features. Auto encoders are mainly used to reduce dimensionality and to denoise images. The autoencoder we use is basically an undercomplete autoencoders.

4.2 Partial Convolution

The majority of deep learning frameworks which deal with image inpainting, treat missing and valid pixels the same sense. The missing pixel values are filled with a fixed pixel value and apply standard convolutions to the input image. The problem with this approach is, setting the missing pixels to a pre-defined value may not be a good idea and also the idea of operating the convolution operation to the input image regardless of the validness of the pixels. We get the idea of Partial Convolution from authors of the paper [1].

The idea in partial convolutions is to employ a U-net like network with skip connections in which all standard convolutional layers are replaced by proposed partial convolutional layers. Moreover, separation of missing pixels from the valid pixels during the convolution makes the result depend only on valid pixels. This is the reason this method is called partial convolution. The convolution is partially performed on the input based on a binary mask image that can be updated automatically.

Let us define W and b be the weights and bias of the convolution filter. X represents the pixel values (or feature activation values) being convolved and M is the corresponding binary mask which indicates the validness of each pixel/feature value (0 for missing pixels and 1 for valid pixels). The proposed partial convolution is computed,

$$x' = \begin{cases} \mathbf{W}^T(\mathbf{X} \odot \mathbf{M})\frac{\text{sum}(\mathbf{1})}{\text{sum}(\mathbf{M})} + b, & \text{if sum}(\mathbf{M}) > 0 \\ 0, & \text{otherwise} \end{cases}$$

where $\odot$ means element-wise multiplication and 1 is a matrix of ones that has the same shape as M. From this equation, you can see that the results of the partial convolution only depend on the valid input values (as $X \odot M$). sum($\mathbf{1}$)/sum($\mathbf{M}$) is a scaling factor to adjust the results as the number of valid input values for each convolution is varying.

Update the binary mask after each partial convolutional layer. The proposed rule to update the binary mask is quite easy. If the result of the current convolution is conditioned on at least one valid input value, the corresponding location will be regarded as valid for the next partial convolutional layer.

$$m' = \begin{cases} 1, & \text{if sum}(\mathbf{M}) > 0 \\ 0, & \text{otherwise} \end{cases}$$

5 Implementation

5.1 Data Generation

To generate the training and testing data set and the targets, we use the images from the CIFAR dataset and apply a mask creaked using OpenCV to the image. We draw black lines of random length and thinckness on a white background and impose it on the image to produce the input images and the fresh images serve as the target variables.

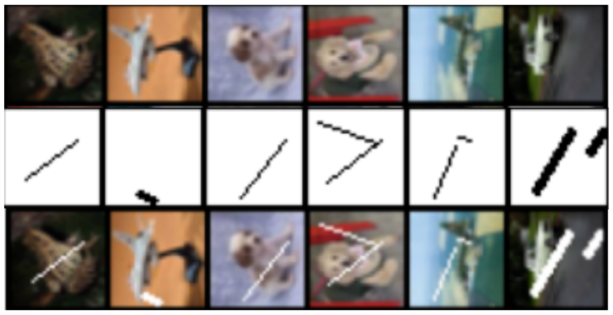


Figure 1: Mask Generation and Superposing over original images

5.2 Vanilla AutoEncoder

To implement the encoder part of the autoencoder we used Conv2D layers with an incremental channel of 3,32,64,128,256,512. We use a padding and stride of 1 each and a kernel of size (3,3). This is followed by maxpooling and Relu activation function. The decoder contains contranspose2D layers that reduce the channels and increase the size back to 32*32*3 channels. The input data is normalised so the final layer has an activation function of sigmoid.

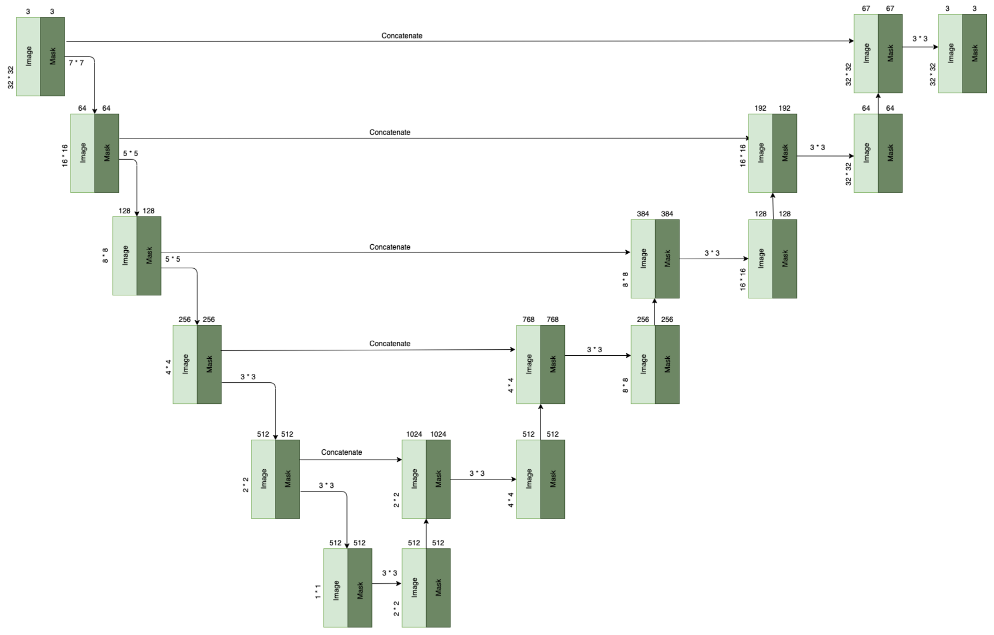


Figure 2: Network Architecture of AutoEncoder with Partial Convolution

5.3 Partial Convolution

There is no official implementation of Partial Convolution in TensorFlow and Pytorch, therefore we implemented the Partial Convolution Layer. As this is an Autoencoder, the architecture has two components - encoder and decoder. We created the partial convolution layer too. The decoder and encoder uses the partial convolution layer followed by Batch Normalization and ReLU activation functions. We have used 5 layers of decoder and 5 layers of encoder. The encoder takes the images of size 32×32 and downsamples it to 1×1. The decoder concatenates data from previous layer with output of corresponding layer of encoder. It first performs upsampling, then partial convolution and batch normalization. We have used LeakyReLU as activation function. We compiled the model with the Adam optimizer with default parameters, mean_square_error as the loss function. The authors of the paper [1] have suggested use of loss functions to target both per pixel reconstruction loss as well as composition loss, i.e. how smoothly the predicted values transition into the surrounding the context. For simplicity and as we are dealing with simple data set, we didn't include that in our experimentation.

6 Experiments

Both the networks is able to reconstruct the images from the masked image. We represent the images below. Even though the images generated by two models aren't that different visually, the pixel wise L2 loss is less while using Partial Convolution.

We trained the model with partial convolution for 100 epochs, the model with VanillaCNN for 20 epochs. We found that the VanillaCNN performs good after 10 epochs of training. However, the model with partial convolution keeps getting better with epochs. Below we represent the losses and the graph of validation loss for our models.

	Partial Convolution	Vanilla CNN
Training Loss	.0003	.0009
Validation Loss	.0003	.0013
Testing Loss	.0005	.0010

From the results, we can say both the models generalize well and provides good accuracy in train, validation and test.

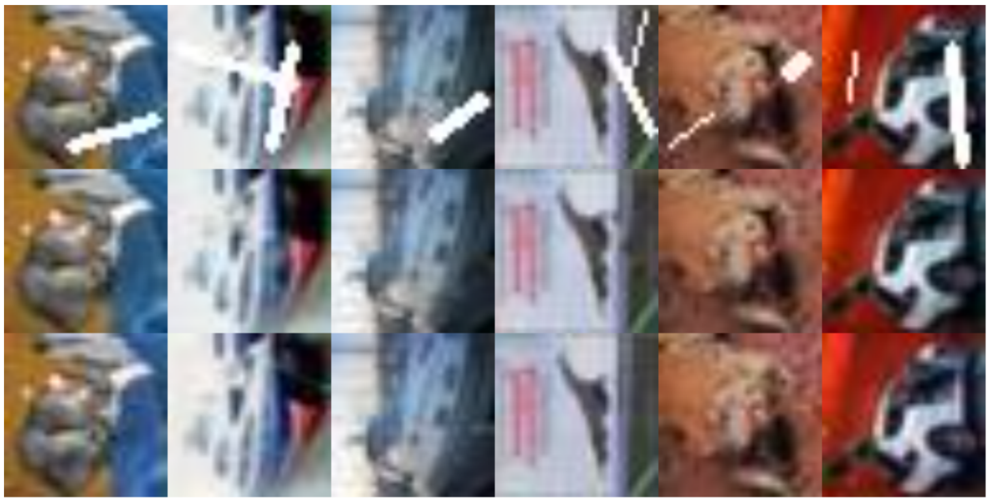
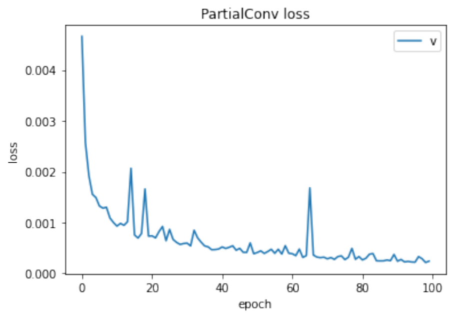
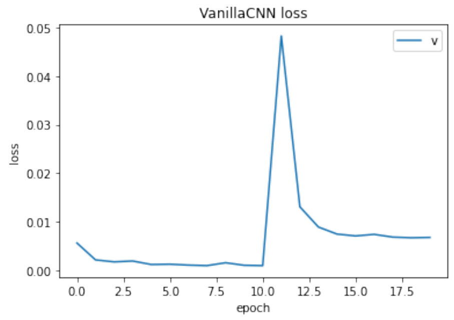


Figure 3: Results [Top: Masked Image, Middle: Partial Convolution, Bottom: VanillaCNN]



(a) Partial Convolution



(b) vanillaCNN

Figure 4: Loss Minimization over Epochs

7 Running our Code

The code can be ran in two modes. Additional argument needs to be passed if we want to test on pretrained model. We need to unzip the package.

```
# To run from pretrained model
python3 main.py 1

# To train the model
python3 main.py 0
```

8 Conclusions

In this report, we have proposed two models that are used for recover a distorted image. The VanillaCNN approach which is part of blind set of image inpainting algorithms is working upto the standard, but if we compare losses, the Autoencoder with partial convolution feature minimizes the loss to the scale of 0.0002. As, we are dealing with images of lower resolution, the VanillaCNN approach is working. With higher resolution images, we would see the Autoencoder with Partial Convolution work much better. We could only touch the basics of Image Inpainting algorithms. Generative Adversarial Nets can be used for this task and has proved to be a huge success in recent works.

Future improvements could be incorporating GANs in this project and doing a comparative study. Also, training the models in a high resolution datasets. We would like to thank the teaching team for providing us this great learning opportunities and University Of Florida for providing the precious GPUs in HiPerGator to train the models.

References

[1] G. Liu, F. A. Reda, K. J. Shih, T. Wang, A. Tao, and B. Catanzaro, “Image inpainting for irregular holes using partial convolutions,” *CoRR*, vol. abs/1804.07723, 2018.

[2] A. Krizhevsky, V. Nair, and G. Hinton, “Cifar-10 (canadian institute for advanced research),”

[3] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester, “Image inpainting,” in *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH ’00, (USA), p. 417–424, ACM Press/Addison-Wesley Publishing Co., 2000.

[4] S. Chhabra and S. Saxena, “An analytical study of different image inpainting techniques,” *Indian Journal of Computer Science and Engineering*, vol. 3, 07 2012.