Report

Dhruva Dhingra (190070020), Liza Dahiya (190050063)
& Dhananjay Singh (180050029)

Due on 23:59, 9 May 2021

Contents

1 Introduction : What is Image Inpainting?

Observed image signals are often corrupted by acquisition channel or artificial editing. The goal of image restoration techniques is to restore the original image from a noisy observation of it. Image denoising and inpainting are common image restoration problems that are both useful by themselves and important preprocessing steps of many other applications. Image denoising problems arise when an image is corrupted by additive white Gaussian noise which is common result of many acquisition channels, whereas image inpainting problems occur when some pixel values are missing or when we want to remove more sophisticated patterns, like superimposed text or other objects, from the image.

Denoising algorithms are usually of two types : either the algorithm works directly on the data in the image domain or it changes the basis to some domain where data can be easily separated from noise using statistics of natural image distribution in the new domain.

Image Inpainting methods are of 2 types : either we know where the missing pixels are located or we don't know where the missing pixels are located.
If we know where the missing pixels are located and we provide this information to the algorithm, then, this type of image inpainting is called non blind inpainting.
If we do not know where the missing pixels are, then, this type of image inpainting is called blind inpainting.

We have a plethora of non blind image inpainting algorithms which work very well. They can remove text, doodles and fill in information of large objects.

However, blind image inpainting is quite a difficult task. Existing algorithms can mostly address iid or simply structured impulse noises.

2 Algorithms known

There are several algorithms known for image inpainting such as :"MCA" (morphological component analysis)
"ASR" (adaptive sparse reconstructions)
"ECM" (expectation conditional maximization)
"KR" (kernel regression)
"FOE" (fields of experts)
"BP" (beta process)
"K-SVD"

All of these are sparse coding algorithms.

MCA and ECM compute the sparse inverse problem estimate in a dictionary that combines a curvelet frame, a wavelet frame and a local DCT basis.
ASR calculates the sparse estimate with a local DCT.
BP infers a nonparametric Bayesian model from the image under test (noise level is automatically estimated).
Using a natural image training set, FOE and K-SVD learn respectively a Markov random field model and an overcomplete dictionary that gives sparse representation for the images.

Generally, K-SVD and BP are acccepted to generate the best inpainting results in literature.

3 Avenues pursued

1. Deep Learning via SSDA
2. Context Encoders

4 Deep Learning via SSDA

4.1 An idea of the algorithm implemented in the paper

Sparse coding models work quite well for image inpainting problems. Sparse coding models say that patches of images can be expressed a linear combination of vectors from a pre-defined set of vectors (called the dictionary) and that the number of vectors required to represent the patches are small (sparse). However, note that this limits us to **linear** combinations of the dictionary vectors (also called atoms).

In the paper that we have chosen, the authors combined sparse coding with deep learning neural networks. The authors noted that sparse coding models worked well. They also noted that a proposal to denoise images using CNNs was floated a while ago. They combined these two principles to get sparse deep learning neural networks.

Autoencoders take an input image and try to reconstruct it after it passes through a low-dimensional "bottleneck" layer, with the aim of obtaining a compact fea ture representation of the scene. Unfortunately, this feature representation is likely to just compresses the image content without learning a semantically meaningful representation.
Denoising autoencoders address this issue by corrupt- ing the input image and requiring the network to undo the damage.

5 What does the algorithm do?

It uses a series of Denoiser Autoencoders i.e. a stack of Denoiser Autoencoders : SDA

5.1 Training Phase

The training phase consists of training DAs one by one to output information from the image into some smaller dimension.

5.2 What does a denoiser autoencoder do

A DA first tries to encode information from a higher dimension input vector to a lower dimension vector (aka encoder). The intuition behind this is similar as sparse coding models - the image can be represented by a small set of features.

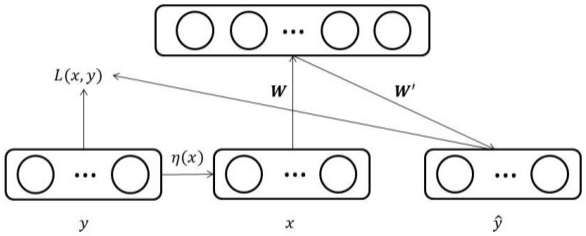
Next, the DA decodes this information (the output of the encoder). The loss function is computed over this DA and the image. Usually, the loss function is Mean Squared Error.

But, notice that this essentially amounts to learning about the image. Because, we don't want to do this, instead of passing the image itself to the encoder, we add some noise to it and give it to the encoder. We also add some regularisation and sparsity inducing terms to the cost function to maintain sparsity.

Let the image be $\mathbf{y}$.
We add some random stochastic noise to get the noisy image $\mathbf{x} = \eta(\mathbf{y})$
Now, this is fed into the encoder to get a vector of lower dimension, $\mathbf{h}(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$
This is now fed into the decoder to retrieve a non noisy image : $\hat{\mathbf{y}}(\mathbf{x}) = \sigma(\mathbf{W}'\mathbf{h}(\mathbf{x}) + \mathbf{b}')$
Now, the loss function is computed as $\mathbf{L}(\mathbf{y}, \hat{\mathbf{y}})$ where $\mathbf{L}$ is usually a mean squared error loss.

Now, the trainable parameters $\Theta = \{\mathbf{W}, \mathbf{b}, \mathbf{W}', \mathbf{b}'\}$

$\Theta = \arg\min_\Theta \sum_{i=1}^{i=N} L(y_i, \hat{y}(x_i) = \arg\min_\Theta \sum_{i=1}^{i=N} ||\mathbf{y_i} - \hat{\mathbf{y}}(\mathbf{x_i})||_2^2$



(a) Denoising auto-encoder (DA) architecture

Figure 1: DA architecture

5.3 Modification to a normal DA to get sparse DA i.e. SDA

This is a normal DA. But, we also add some sparsity inducing terms in the loss function to get the modified loss function as :

$$L_1(\mathbf{X}, \mathbf{Y}; \mathbf{\Theta}) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} ||\mathbf{y_i} - \hat{\mathbf{y}}(\mathbf{x_i})||_2^2 + \beta \, KL(\hat{\rho}||\rho) + \frac{\lambda}{2}(||\mathbf{W}||_F^2 + ||\mathbf{W}'||_F^2$$

where $\hat{\rho} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{h}(\mathbf{x_i})$

$KL(\hat{\rho}||\rho) = \sum_{j=1}^{j=|r\hat{h}o|} \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{(1-\rho)}{(1-\hat{\rho}_j)}$

What does this KL divergence term do?
We set ρ to be some small value so that the KL divergence term encourages the mean activation of hidden units to be small.
The Frobenius norm of the weight matrices is added to prevent overfitting.

By this modified loss function, we have managed to combine the virtues of a neural network with the virtues of sparse coding.

These modified DAs are the basic units of the SSDA architecture proposed in the paper.

5.4 Stacking SDAs to get SSDA

We train a single SDA. After which, the $\mathbf{h}(\mathbf{x})$ term manages to encode the information of an image into a lower dimension vector. Thus, this term carries a lot of information.

Now, this $\mathbf{h}(\mathbf{x})$ term is fed into the encoder layer of the next DA (after adding some noise). Again, the next DA is trained as the first DA.

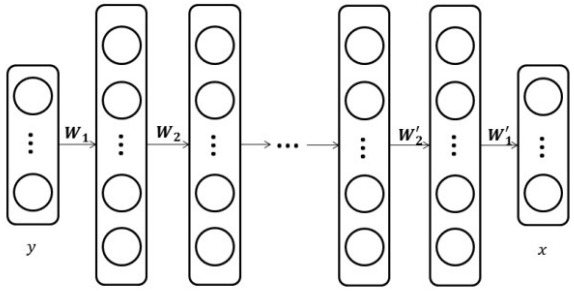
This process is then repeated so that we can manage to squeeze out a lot of information of a single image into a very small dimension vector.

5.5 Prediction architecture

The encoder layers of the DAs are stacked one after another.

Finally, the decoder layers are applied in reverse (decoder of the last DA first).

And an output layer is applied at the end.



(b) Stacked sparse denoising auto-encoder architecture

Figure 2: SSDA architecture

5.6 Results

Unfortunately, our implementation couldn't give satisafactory results. We have included the code but as it doesn't give satisfatory results, we have not included the results.

6 Context Encoders

6.1 Intuition behind the algorithm

Again, we exploit the fact that natural images are highly structured.

Given an image with some missing pixels, we train a convolutional neural network to regress to the missing pixel values. This model is called **Context Encoder** by the authors of the paper.
These context encoders are very closely related to autoencoders as they share a very similar encoding-decoding architecture.

Note that the encoders and decoders in this model are fully connected to each other. This is because standard CNNs connet different features together, but never diretly connect all of them together. Thus, a normal CNN model cannot completely predict an image because no part of the model gets information about the complete image.

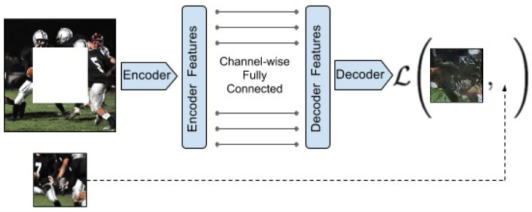


Figure 3: Context Encoder Architecture

6.1.1 Then what is the difference between SSDA and Context Encoders?

Autoencoders take an input image and try to reconstruct it after it passes through a low-dimensional "bottleneck" layer, with the aim of obtaining a compact fea ture representation of the scene. Unfortunately, this feature representation is likely to just compresses the image content without learning a semantically meaningful representation.
Denoising autoencoders address this issue by corrupt- ing the input image and requiring the network to undo the damage.

However, this corruption process is typically very localized and low-level, and does not require much semantic information to undo. In contrast, context encoders need to solve a much harder task: to fill in large missing areas of the image, where they can't get "hints" from nearby pixels. This requires a much deeper semantic understanding of the scene, and the ability to synthesize high-level features over large spatial extents.
This is similar in spirit to word2vec which learns word rep- resentation from natural language sentences by predicting a word given its context.

6.1.2 vs Texture learning models

This is different from texture learning algorithms. Texture learning algorithms can learn the texture and predict some missing / corrupted image pixels when this pixels are far apart (that is, this set of pixels forms holes).

Whereas, in this model, an entire rectangular portion of the image is missing. This is a much more difficult task.

6.2 Encoder Decoder Pipeline

We do not describe the encoder decoder pipeline in detail because this pipeline is the same as that in SSDA with a single notable difference.

6.2.1 Fully Connected Layers

Note that the encoders and decoders in this model are fully connected to each other. This is because standard CNNs connet different features together, but never diretly connect all of them together. Thus, a normal CNN model cannot completely predict an image because no part of the model gets information about the complete image.

6.3 Loss function

A satisfactory model needs to both understand the content of an image as well as produce a plausible hypothesis for the missing parts.

6.3.1 Masked L2 loss is not sufficient

L2 loss is not sufficient to train the models. This is because L2 loss encourages the neural network to set weights such that most of the pixels in the missing region converge to the mean of the image.
But, we don't want such an output. We want the output to be able to predict high frequency changes in the image too.

(c) Context Encoder
($L2$ loss)

Figure 4: L2 Loss predictions : blurry image

Thus, we add another loss - adversarial loss.

6.4 Adversarial Loss

The adversarial loss is based on Generative Adversarial Networks (GANs).

Quoting directly from the paper,

> To learn a generative model G of a data distribution, GAN proposes to jointly learn an adversarial discriminative model D to provide loss gradients to the generative model. G and D are parametric functions (e.g., deep networks) where $G : Z \rightarrow X$ maps samples from noise

distribution Z to data distribution X. The learning procedure is a two-player game where an adversarial discriminator D takes in both the prediction of G and ground truth samples, and tries to distinguish them, while G tries to confuse D by producing samples that appear as "real" as possible. The objective for discriminator is logistic likelihood indicating whether the input is real sample or predicted one :

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

6.4.1 Adapting this to our needs

We model the generator by the context encoder $G \equiv F$. We replace the z term with a masked x term. But, if we do so, the discriminator D can easily exploit this discontinuity to predict that the image is fake. Thus, we condition only the generator on the context.

$$\mathcal{L}_{adv} = \max_D \mathbb{E}_{x \in \mathcal{X}}[\log(D(x)) \log(1 - D(F((1 - \hat{M}) \odot x))]$$

6.5 Joint Loss

$$\mathcal{L} = \lambda_{rec}\mathcal{L}_{rec} + \lambda_{adv}\mathcal{L}_{adv}$$

where
$\mathcal{L}_{rec}$ is the masked L2 loss
$\mathcal{L}_{adv}$ is the adverserial loss

6.6 Final Network

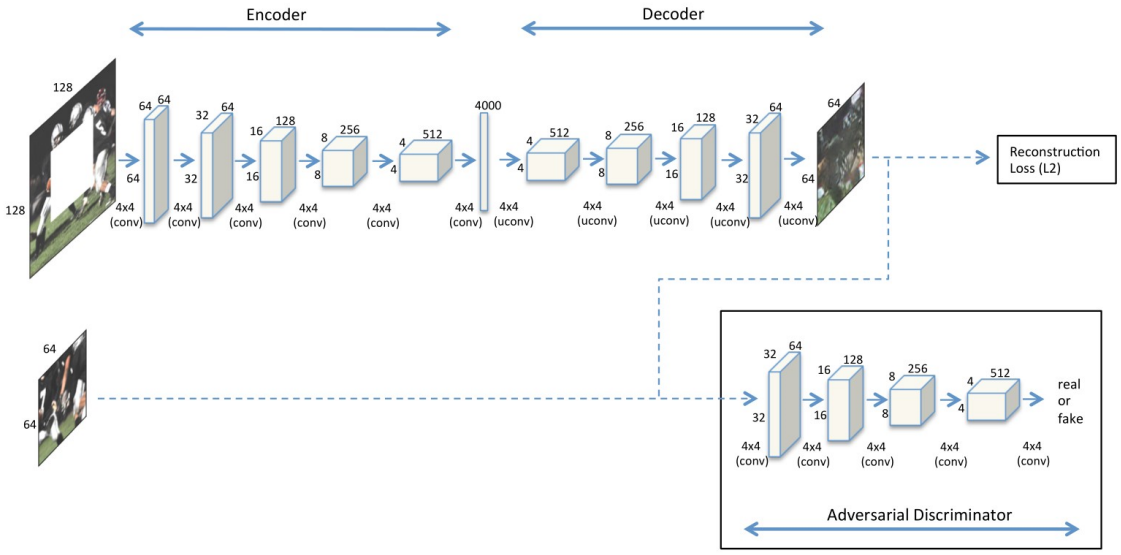


Figure 5: The complete architecture

6.7 Results

6.7.1 Results of the paper

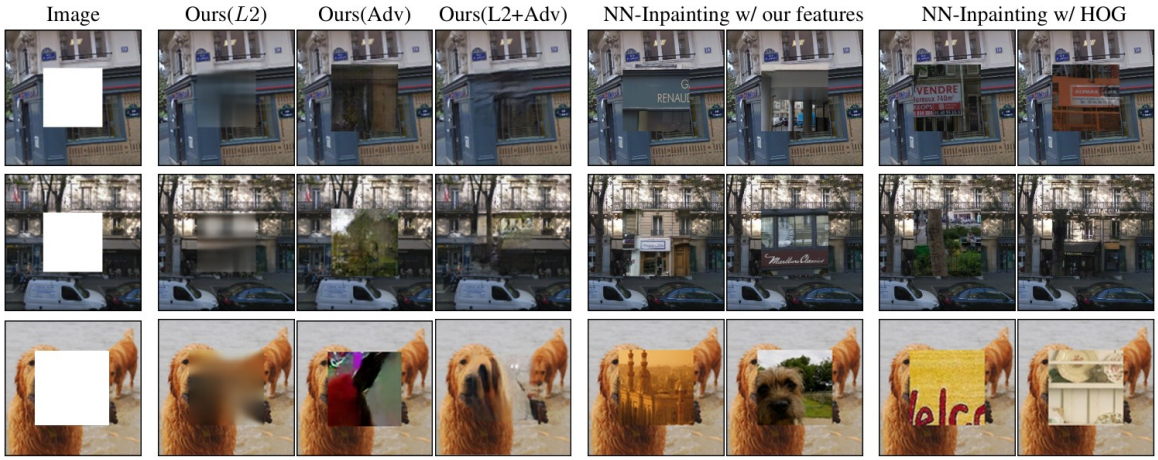


Figure 6: Semantic Inpainting using different methods on *held-out* images. Context Encoder with just L2 are well aligned, but not sharp. Using adversarial loss, results are sharp but not coherent. Joint loss alleviate the weaknesses of each of them. The last two columns are the results if we plug-in the best nearest neighbor (NN) patch in the masked region.

Figure 6: Results presented by the authors of the papers

6.7.2 Our Results

We trained our model on images from the `ImageNet` database. Our results are satifactory. Here are our results :

Figure 7: First row = original image, second row = masked image, third row = inpainted image

Figure 8: First row = original image, second row = masked image, third row = inpainted image

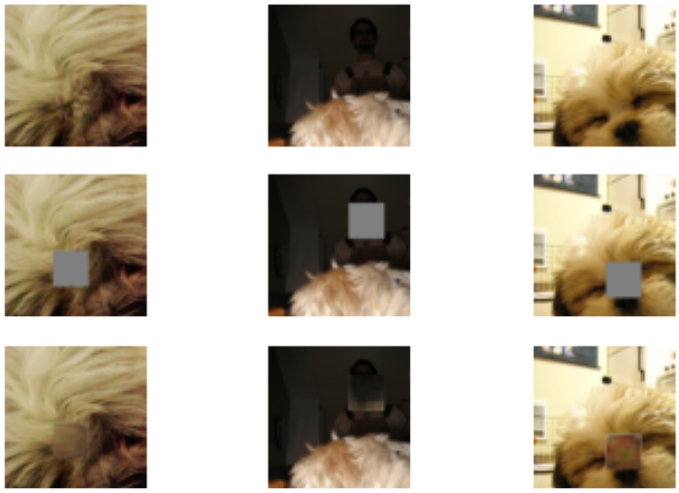


Figure 9: First row = original image, second row = masked image, third row = inpainted image

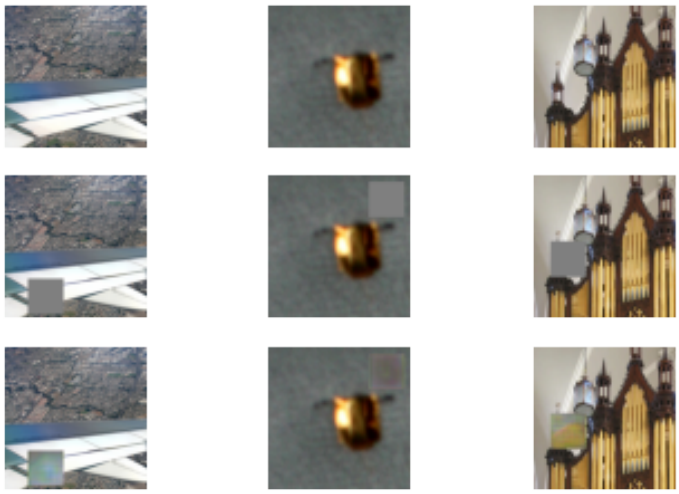


Figure 10: First row = original image, second row = masked image, third row = inpainted image

6.8 Conclusion

This model gives us quite good solutions. Further paper have been written extending this model.

1. After the context encoder model, some models have been developed to transfer the style of the most similar valid pixels to the generated pixels to enhance local texture details
2. Some papers have tried to take into account both the local and global information of an image to enforce locally and globally consistency (by having both global and local discriminators)