Project Title

Semantic Image Inpainting With context encoders

Team members:
Abhishek(2018101028)
Sreeharsha(2018102002)
Rahul (2018102037)
Nihar(2018102039)

Team:
Jaathiratnalu

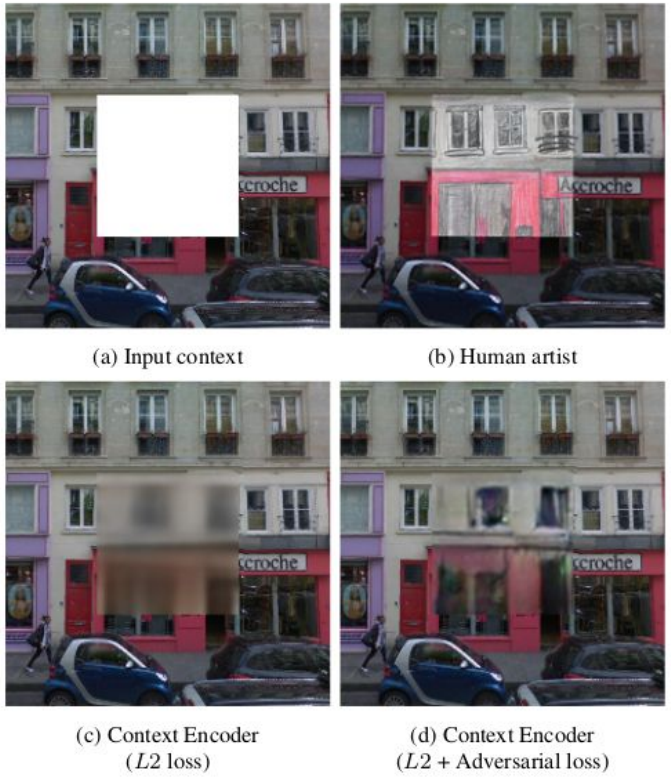
Repo URL :
https://github.com/Computer-Vision-IIITH-2021/project-jaathiratnalu

Objective

The main objective of this paper is to reconstruct the missing part of an image by learning feature representations of the scene.

Given a corrupted image with parts of image missing or distorted, our objective is to provide a seamless and plausible replacement for a specific region of pixels in the image with the help of available visual data.

This method had several use cases like restoring ancient books, art, images etc.



(a) Input context (b) Human artist

(c) Context Encoder ($L2$ loss) (d) Context Encoder ($L2$ + Adversarial loss)

Related Previous Works

- The part of the image missing is too large for **classical inpaintings.**
- **Scene completion** can’t fill arbitrary holes in the image
- Previous implementations use **hand-crafted distance metric,** like Gist

We use learned distance metric which is superior to hand crafted distance metric.

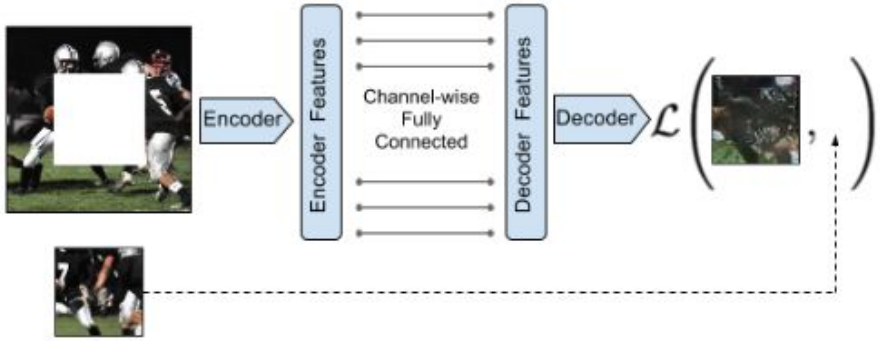
Method overview

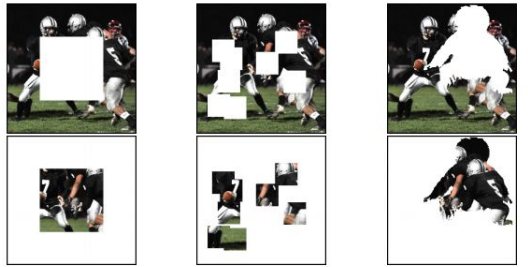
Method Overview

- We use *Context Encoders,* which is an Autoencoder based architecture used to reconstruct the missing part of the image.
- The key difference is most of the standard methods used to perform computer-aided inpainting rely on local features such as colours and textures, but the reconstruction may not be perfect in many cases.
- But extracting context of whole image will help in better reconstruction of the missing part

Method Overview

- It consists of two components, an encoder *E* and a decoder *D*. Encoder's output is a vector with a lower dimensionality than that of the input.
- The output of the context-encoders is a feature vector.
- The feature vector is channel-wise connected to the decoder
- The decoder fills in realistic image content using the encoded feature vector
- This architecture is based on the renowned AlexNet Architecture





(a) Central region (b) Random block (c) Random region

Method overview :

The pipeline consists of three major stages:

- **Context Encoder**: Our model is not trained for ImageNet classification; rather, the network is trained for context prediction “from scratch” with randomly initialized weights.
- **Channel-wise fully connected layer**.
- **Decoder**: It generates pixels of the image using the encoder features.

Loss function:

- **Reconstruction Loss:** We use a normalized masked L2 as our reconstruction loss function
- **Adversarial Loss:** Loss based on GAN’s

Details of each component of the Architecture

Context Encoder

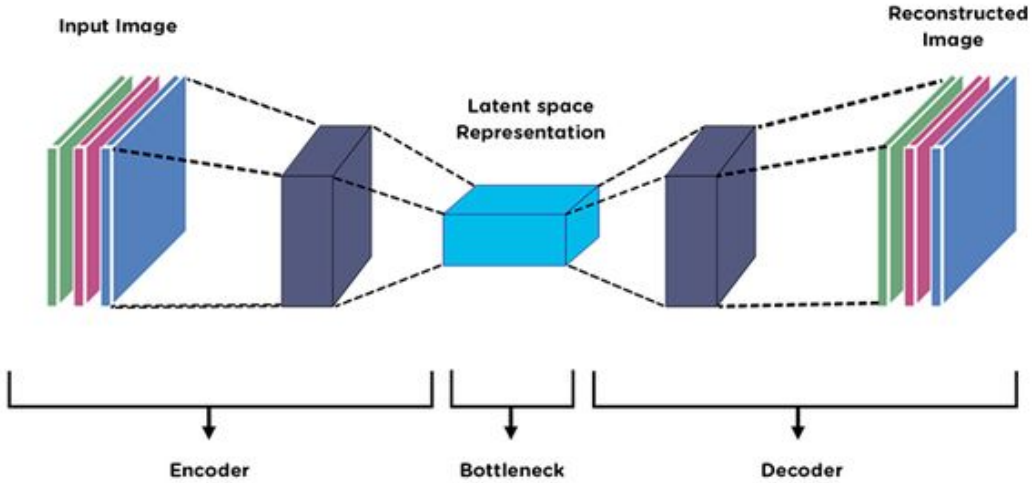
- The encoder is inspired from the AlexNet architecture with five convolutional layers followed by max pooling. The encoder, upon being provided an image of 227*227, provides a 6*6*256 dimensional feature representation.
- However, unlike AlexNet our model is not trained for ImageNet classification; rather, the network is trained for context prediction “from scratch” with randomly initialized weights.

Channel-wise fully connected layers

- The propagation of information within the feature representation is handled by channel-wise fully-connected layer.
- The fully-connected layer with groups has all the nodes connected to each other. This layer joins the output of the encoder to the input of the decoder.
- If the input layer has m feature maps of size n × n, this layer will output m feature maps of dimension n × n.
- This layer allows each unit of the decoder to perceive the complete image content.

Decoder

- The output from the channel-wise fully-connected layer is fed to the decoder, which reconstructs the image and the missing content.
- The decoder has five up-convolution layers each followed by a ReLU activation function so as to reach the target image size.



Loss function formulation

The loss function has two components to it;

L2 reconstruction loss: Responsible for capturing the structure of the missing region and coherence with regards to its context, but tends to average together the multiple modes in predictions. It is given by:

$$\mathcal{L}_{rec}(x) = \|\hat{M} \odot (x - F((1 - \hat{M}) \odot x))\|_2$$

Adversarial loss: aims to make prediction look real, and has the effect of picking a particular mode from the distribution. It's given by:

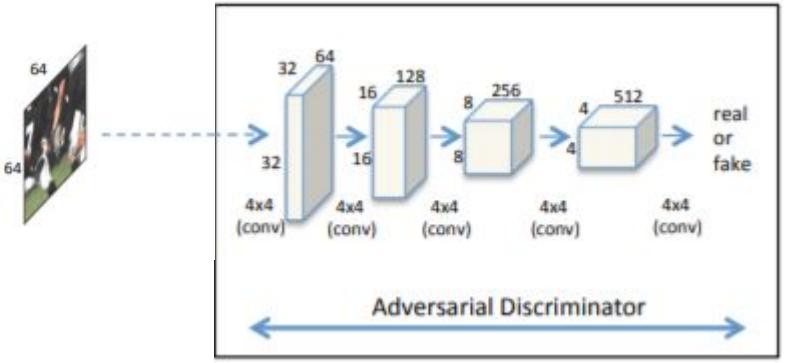
$$\mathcal{L}_{adv} = \max_{D} \quad \mathbb{E}_{x \in \mathcal{X}}[\log(D(x)) + \log(1 - D(F((1 - \hat{M}) \odot x)))]$$

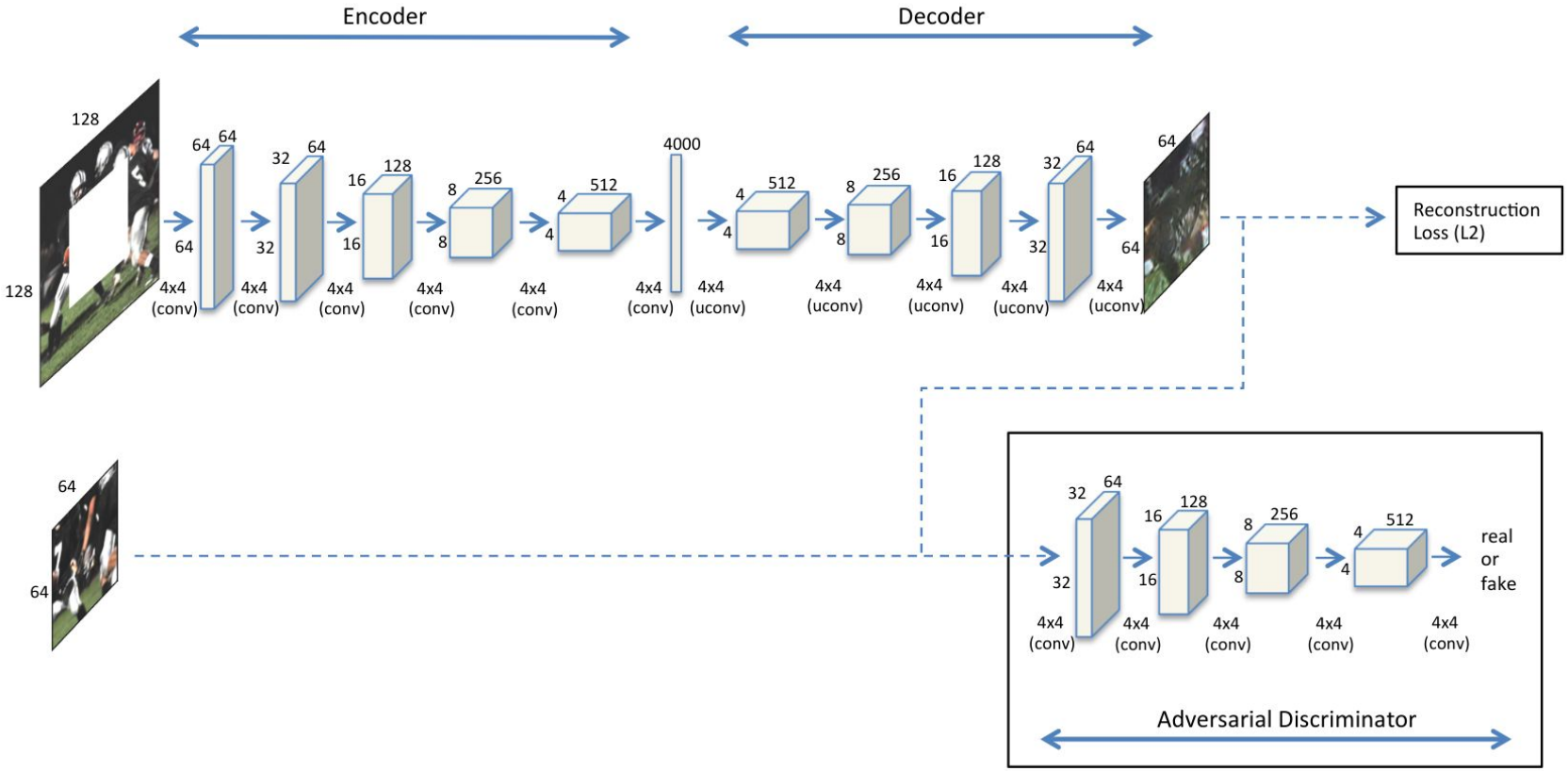
We experimentally weigh the contribution of each of the loss terms given by the below expression:

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

Discriminator

- The discriminator has an image input and it is tasked with classifying it either as genuine data or an inpainting result.
- The network outputs its opinion in a single scalar representing a probability. For arbitrary (or random) region damage, an image of the same size as the original undamaged sample is on the Discriminator's input.
- One of the reasons for this architectural design was the fact that the discriminator could fail to learn useful features and only manage to learn to recognize a boundary of the area where inpainted data were inserted.
- Overall patch-only evaluation results in lower computational requirements and in turn a shorter training time.



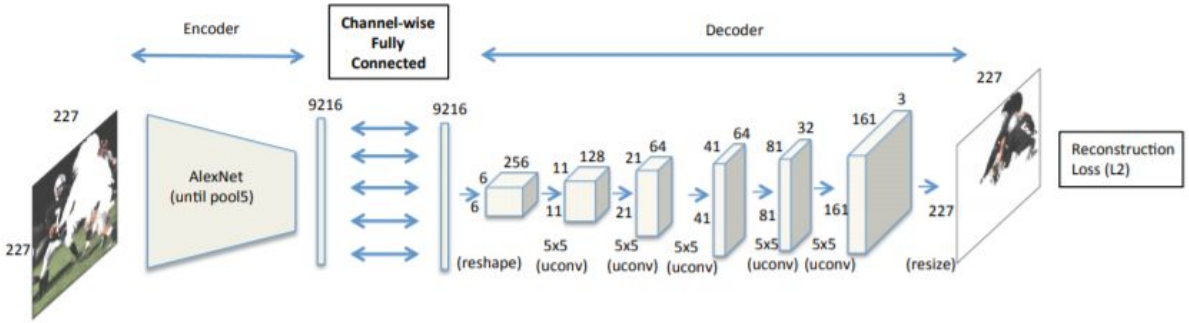

Encoder
Decoder
128
128
64
64
64
64
32
64
32
16
128
16
8
256
8
4
512
4
4000
4
512
4
8
256
8
16
128
16
32
64
32
64
64
4x4
(conv)
4x4
(uconv)
Reconstruction
Loss (L2)
64
64
32
64
32
16
128
16
8
256
8
4
512
4
4
real
or
fake
4x4
(conv)
Adversarial Discriminator

AlexNet

Significance of AlexNet

Alex net is an Architecture that consists

- 5 Convolutional Layers
- 3 Fully Connected Layers



What makes this architecture special is its outstanding performance on the ImageNet image classification dataset.

This architecture shows good performance in other tasks involving extracting features from scenes

Unlike the actual AlexNet, our model isn't trained for the ImageNet dataset.

Dataset

Dataset

- We have used Paris Street view dataset which consists of about 6412 images collected from Flickr.
- The following queries were used to collect images from Flickr:

La Defense Paris,Eiffel Tower Paris,Hotel des Invalides Paris,Louvre Paris,Moulin Rouge Paris,Munsee d'Orsay Paris,Notre Dame Paris,Pantheon Paris,Pompidou Paris,Sacre Couer Paris,Arc de Triomphe Paris,Paris

Results

Different Variations Tested

- Changes in L2 weightages
- Effect of Adversarial in the image
- Alexnet (with L2 only)
- Fully connected layer
- Using L1
- Random patches

Miscellaneous Tests

- Alexnet (with discriminator)
- Using SGD

Variations in L2 weightages

L2=0

PSNR=8.5760

L2=0.5

PSNR=9.7919

L2=0.9

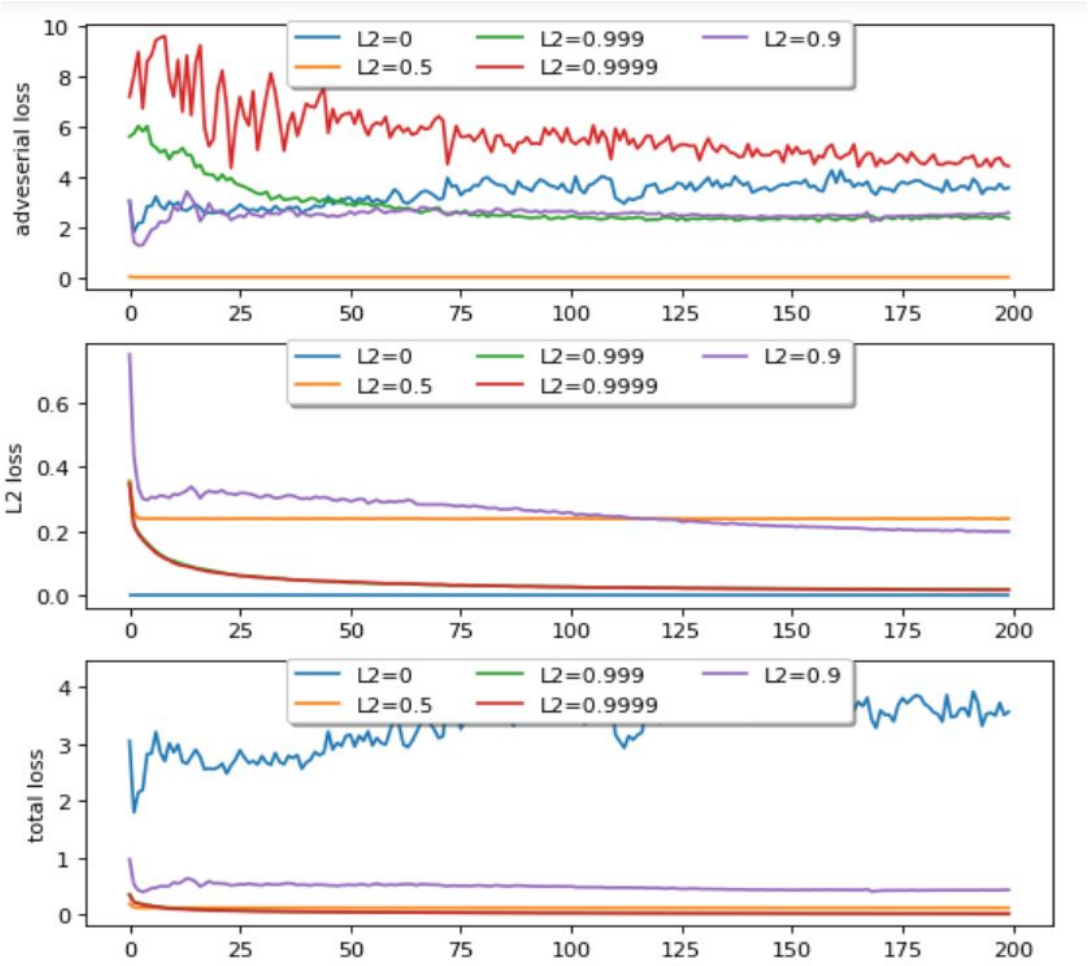
PSNR=15.91

L2=0.999

PSNR=20.39

L2=0.9999

PSNR=20.81


adveserial loss
L2=0
L2=0.5
L2=0.999
L2=0.9999
L2=0.9
L2 loss
total loss

Effect of Adversarial

L2=0.999

PSNR=18.15

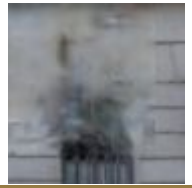
PSNR=20.39

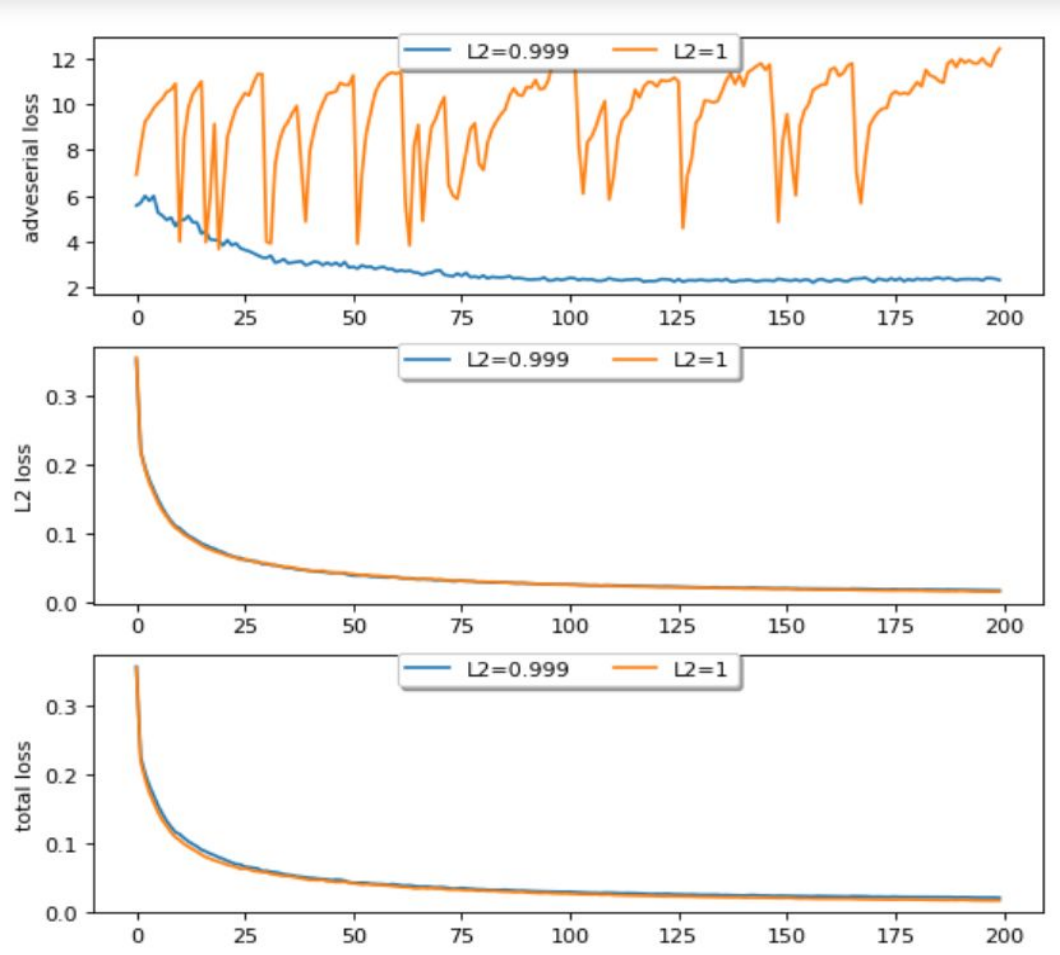
L2=1

PSNR=19.08

PSNR=20.70

In this case we can see that when we use the Adversarial component the black patches in between which are visible in only L2 case are rearranged in more proper fashion to which suits the image




L2=0.999
L2=1
adveserial loss
L2 loss
total loss

AlexNet (L2 only)

Paper model (Only L2)

PSNR=20.3954

AlexNet(Only L2)

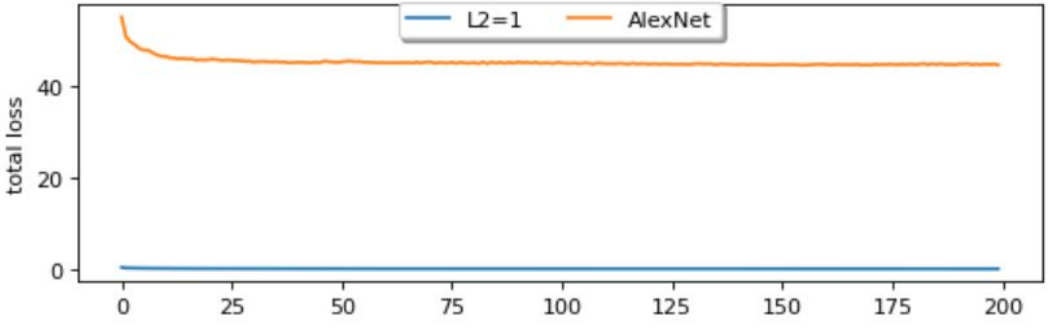
PSNR=18.8796

Paper model (Only L2)

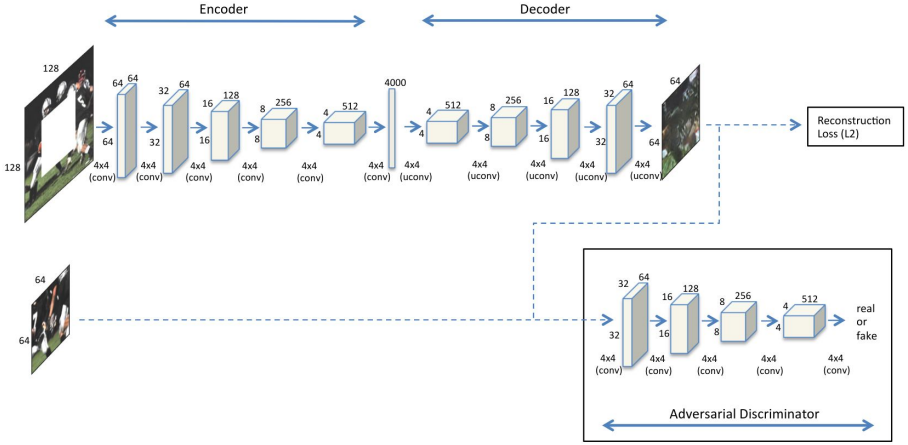
PSNR=19.087

AlexNet(Only L2)

PSNR=15.006



Channel-wise connected layer



In the above architecture a channel wise connected layer is added in between the encoder and decoder

With channel-wise connection

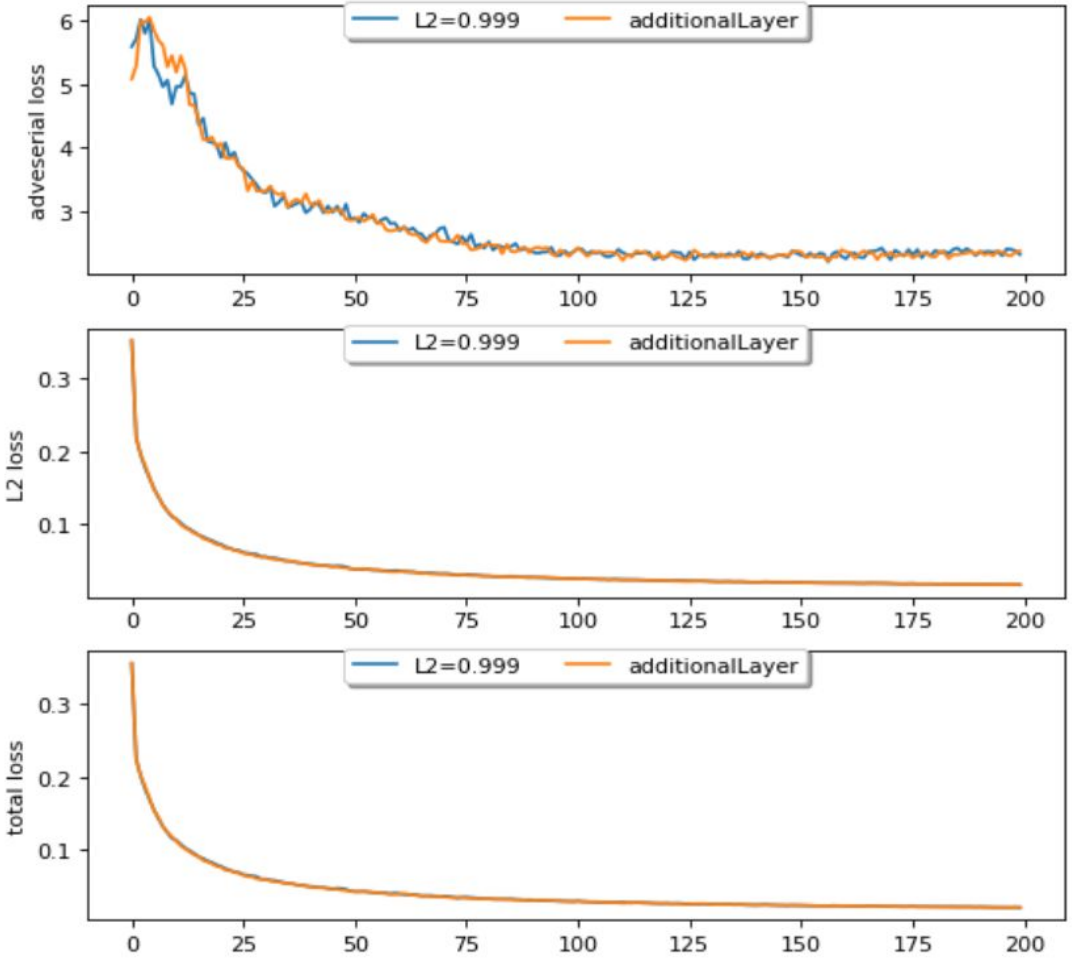
PSNR=19.11

PSNR=21.232

Without channel-wise connection

PSNR=19.087

PSNR=20.3954


L2=0.999
additionalLayer
adveserial loss
6
5
4
3
0
25
50
75
100
125
150
175
200
L2=0.999
additionalLayer
L2 loss
0.3
0.2
0.1
0
25
50
75
100
125
150
175
200
L2=0.999
additionalLayer
total loss
0.3
0.2
0.1
0
25
50
75
100
125
150
175
200

Using L1

- No significant difference
- Both the simple loss encourages the decoder to produce a rough outline of the predicted object, it often fails to capture any high frequency detail
- loss often prefer a blurry solution, over highly accurate textures.

L1 =0.999

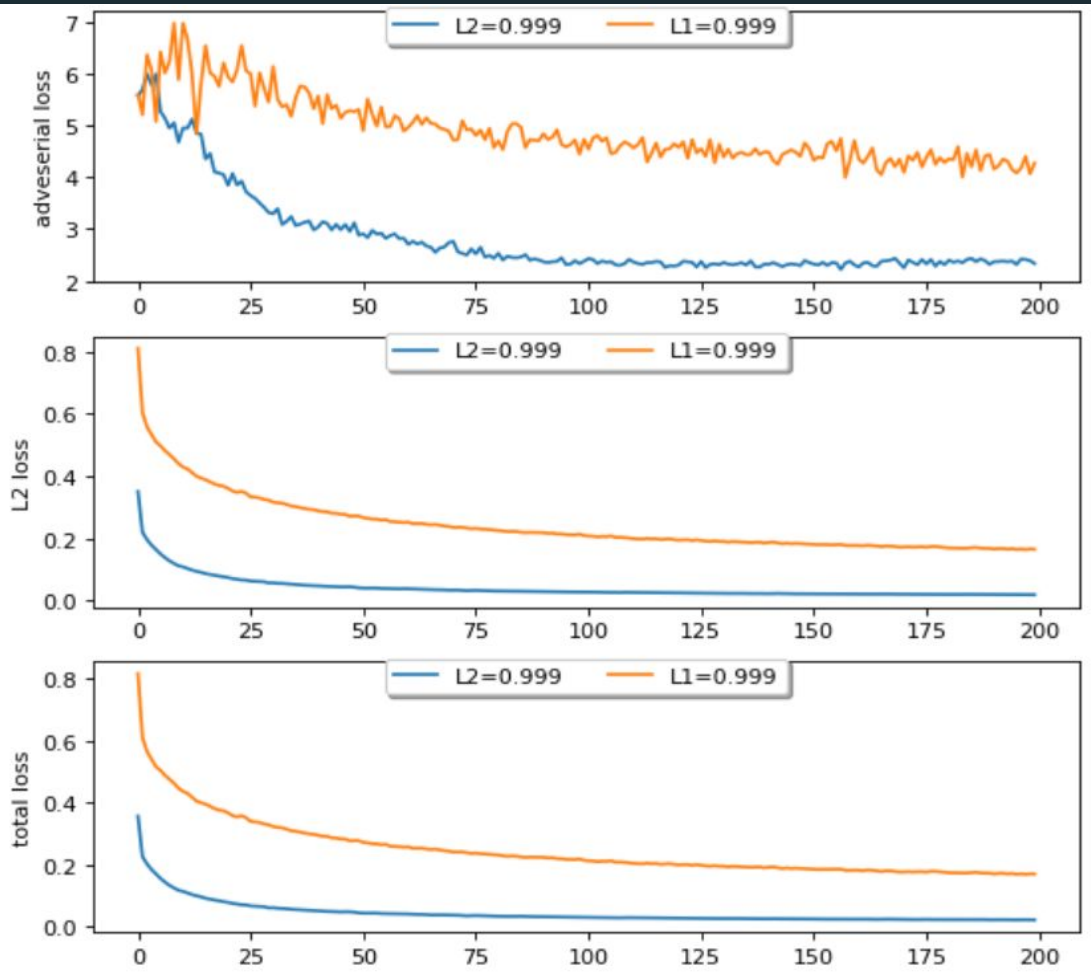
PSNR=18.93

L2 =0.999

PSNR=18.15

PSNR=21.2045

PSNR=20.3954


L2=0.999
L1=0.999
adveserial loss
L2 loss
total loss
0
25
50
75
100
125
150
175
200

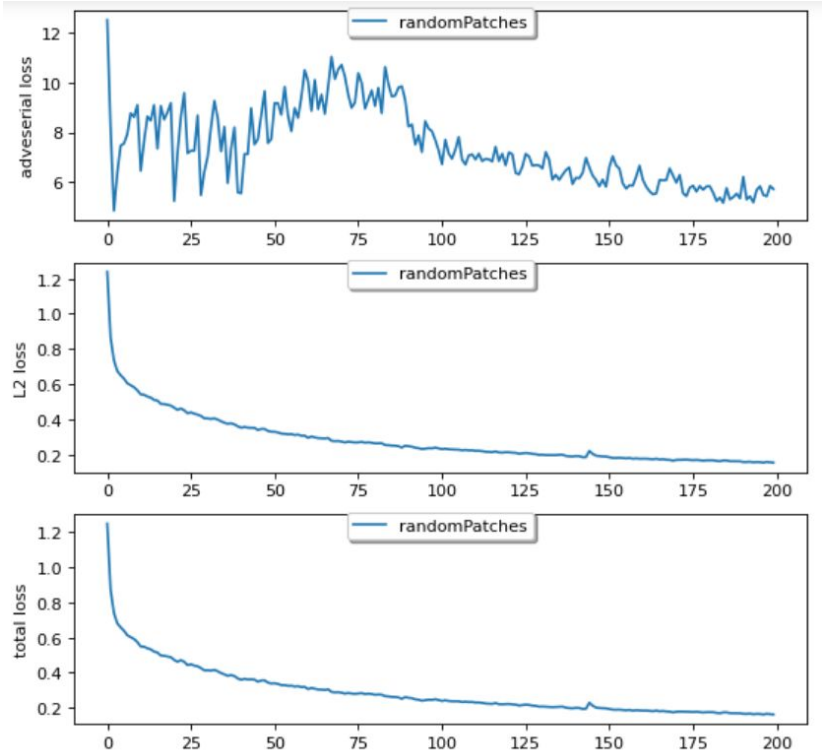
Random patches

Real Cropped Reconstructed

PSNR value of reconstructed image after 200 epochs =20.57

Real Cropped Reconstructed

PSNR value of reconstructed image after 200 epochs = 18.694



Miscellaneous Tests

AlexNet (with Discriminator)

- In this case we can see clearly from the graphs that discriminator does not converge
- The Discriminator instantly recognizes the artefacts that are created in this method and latches onto it
- This makes it impossible to train the Discriminator

Paper model

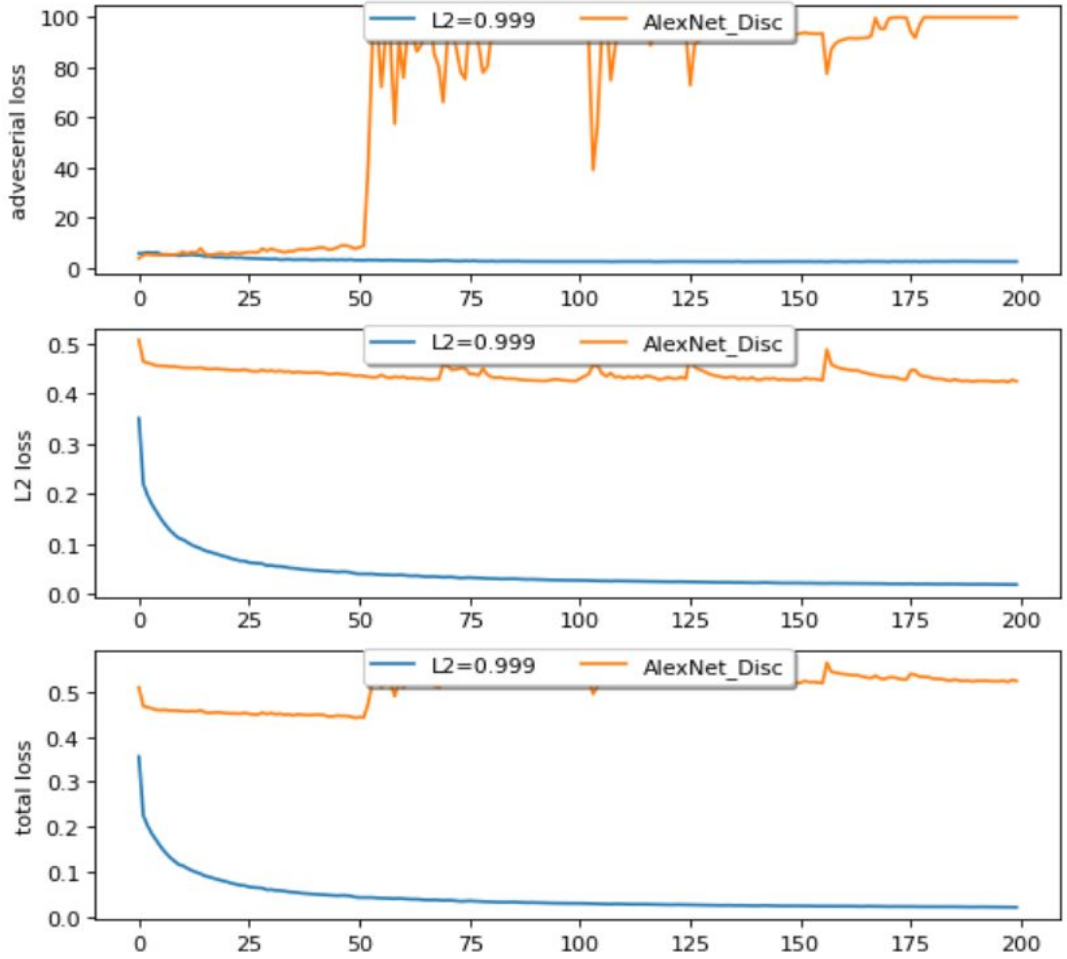
PSNR=19.087

PSNR=18.1575

Alexnet with Discriminator

PSNR=18.6565

PSNR=14.7734


L2=0.999
AlexNet_Disc
adveserial loss
L2 loss
total loss

Using SGD

Adam

PSNR :18.1575

PSNR=19.087

SGD

PSNR=13.9196

PSNR=16.2267

Epoch Wise Results for each Test

Inference on the effect of epochs in training(L2=0)

output				
Epoch	50	100	150	200
L1 loss	0.3721	0.5003	0.6306	0.3835
L2 loss	0.2288	0.4360	0.4958	0.2119
PSNR	12.4255	9.6260	9.0677	12.7581

Inference on the effect of epochs in training(L2=0)

output				
Epoch	50	100	150	200
L1 loss	0.4537	0.6062	0.8499	0.7058
L2 loss	0.3114	0.5370	0.8531	0.5552
PSNR	11.0873	8.7206	6.7102	8.5760

Inference on the effect of epochs in training(L2=0.5)

output				
Epoch	50	100	150	200
L1 loss	0.3515	0.7220	0.3920	0.5631
L2 loss	0.2245	0.6403	0.2415	0.4283
PSNR	12.5082	7.9566	12.1905	9.7030

Inference on the effect of epochs in training (L2=0.9)

output				
Epoch	50	100	150	200
L1 loss	0.2861	0.2374	0.1917	0.1997
L2 loss	0.1367	0.1033	0.0598	0.0663
PSNR	14.6618	15.8774	18.2481	17.8027

Inference on the effect of epochs in training (L2=0.9)

output				
Epoch	50	100	150	200
L1 loss	0.3434	0.29377326	0.2138	0.2636
L2 loss	0.1830	0.14155917	0.0747	0.1025
PSNR	13.3939	14.5112	17.2857	15.9154

Inference on the effect of epochs in training (L2=0.999)

output				
Epoch	50	100	150	200
L1 loss	0.14012	0.1326	0.1307	0.1381
L2 loss	0.0337	0.0312	0.0314	0.0365
PSNR	20.74345	21.0768	21.0487	20.3954

Inference on the effect of epochs in training (L2=0.999)

output				
Epoch	50	100	150	200
L1 loss	0.1583	0.1541	0.1503	0.1630
L2 loss	0.0482	0.0473	0.0547	0.0611
PSNR	19.1883	19.2699	18.6377	18.1575

Inference on the effect of epochs in training (L2=0.9999)

output				
Epoch	50	100	150	200
L1 loss	0.1526	0.1555	0.1509	0.1538
L2 loss	0.0430	0.0523	0.0479	0.0495
PSNR	19.6824	18.8363	19.2097	19.0735

Inference on the effect of epochs in training (L2=0.9999)

output				
Epoch	50	100	150	200
L1 loss	0.1350	0.1420	0.1353	0.1368
L2 loss	0.0317	0.0349	0.0320	0.0331
PSNR	21.0097	20.5832	20.9659	20.8146

Inference on the effect of epochs in training(L2=1)

output				
Epoch	50	100	150	200
L1 loss	0.1766	0.1563	0.1522	0.1548
L2 loss	0.0598	0.0503	0.04958	0.0493
PSNR	18.2537	19.0042	19.0676	19.0879

Inference on the effect of epochs in training(L2=1)

output				
Epoch	50	100	150	200
L1 loss	0.1469	0.1441	0.1374	0.1404
L2 loss	0.0383	0.0353	0.0318	0.0340
PSNR	20.1805	20.5337	20.9956	20.7012

With a learning rate scheduler (lr = 0.01, gamma = 0.5)

output				
Epoch	50	100	150	200
L1 loss	0.2438	0.2423	0.2409	0.2412
L2 loss	0.0888	0.0888	0.0872	0.0878
PSNR	16.5357	16.5360	16.6120	16.5852

Using L1 loss

output				
Epoch	50	100	150	200
L1 loss	0.1456	0.1455	0.1454	0.1475
L2 loss	0.0531	0.0568	0.0575	0.0511
PSNR	18.7694	18.4785	18.4197	18.9325

Adding an additional layer

output				
Epoch	60	100	160	200
L1 loss	0.1469	0.1614	0.1578	0.146
L2 loss	0.0525	0.0605	0.065	0.0490
PSNR	18.8211	18.2055	17.8916	19.1160

Alexnet with only L2 (updated)

output				
Epoch	50	100	150	200
L1 loss	0.3112	0.3202	0.3076	0.2986
L2 loss	0.1296	0.1347	0.1303	0.1262
PSNR	14.8913	14.7248	14.8705	15.0067

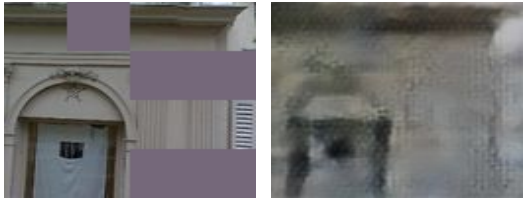
Alexnet with Discriminator

output				
Epoch	50	100	150	200
L1 loss	0.1865	0.2136	0.1891	0.1955
L2 loss	0.0541	0.0610	0.0549	0.0545
PSNR	18.6854	18.1643	18.6272	18.6565

Random Patches

50 Epochs

100 Epochs



150 Epochs

200 Epochs

Random Patches

output				
Epoch	50	100	150	200
L1 loss	0.1535	0.1451	0.16136	0.16364
L2 loss	0.0467	0.0438	0.05131	0.0540
PSNR	19.3182	19.6036	18.9181	18.694

Random Patches

output				
Epoch	50	100	150	200
L1 loss	0.1372	0.1247	0.121	0.13395
L2 loss	0.03444	0.03326	0.028	0.0350
PSNR	20.6495	20.800	21.4886	20.5784

Best Result obtained

Final evaluation training results

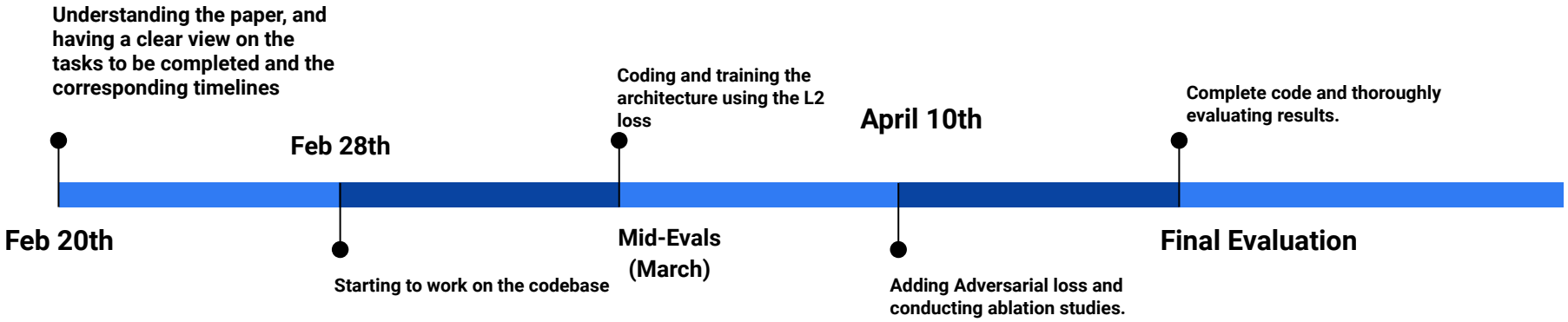
Cropped Image

Real Image

Output after 200 epochs

Paper implementation

Timeline


Understanding the paper, and having a clear view on the tasks to be completed and the corresponding timelines
Feb 20th
Feb 28th
Starting to work on the codebase
Coding and training the architecture using the L2 loss
Mid-Evals (March)
April 10th
Adding Adversarial loss and conducting ablation studies.
Complete code and thoroughly evaluating results.
Final Evaluation

References:

- https://people.eecs.berkeley.edu/~pathak/context_encoder/
- https://heartbeat.fritz.ai/semantic-image-inpainting-with-context-encoders-b1b1b01b90c7
- https://medium.com/knowledge-engineering-seminar/context-encoder-image-inpainting-using-gan-ccd6a1ea5fb7
- https://github.com/pathak22/context-encoder

Thank you