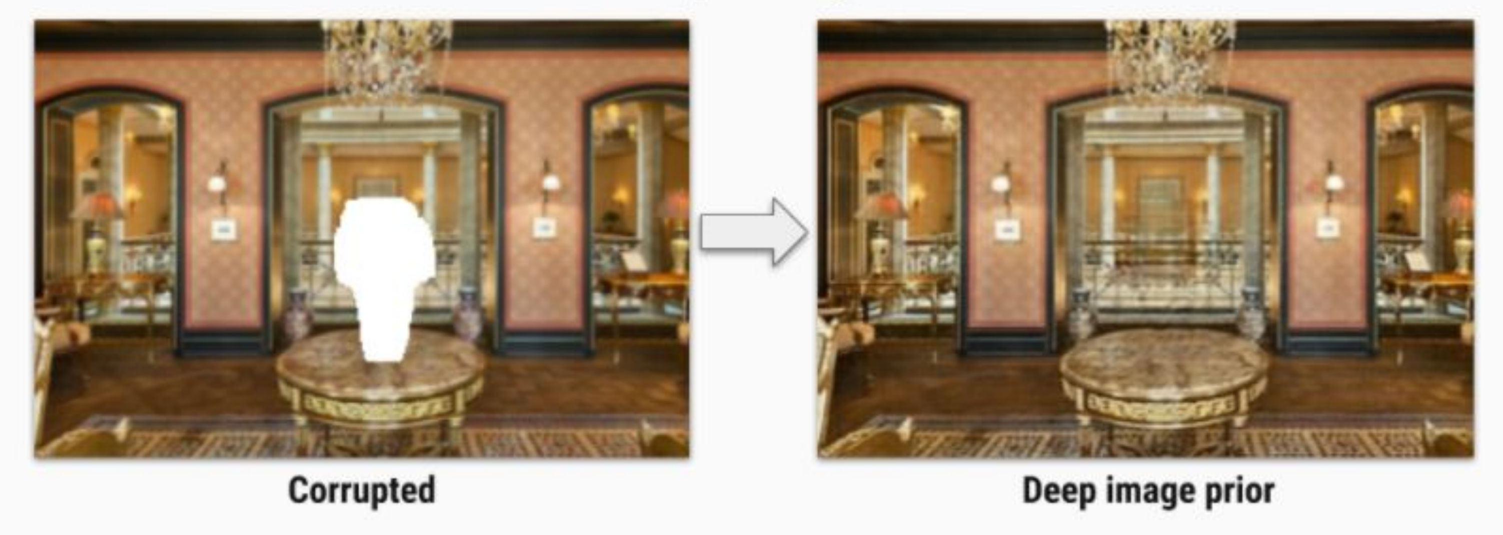
Tech4Heritage

Sample Phase 1 Submission

Team Name : Coffee Addicts
Team Id : 91
Team Leader Name : Aman Gupta

Problem Statement

Planning of art restoration using deep learning to preserve our Heritage. Image below represents one aspect of problem :-



Corrupted

Deep image prior

Objective 1: Data Set Selection

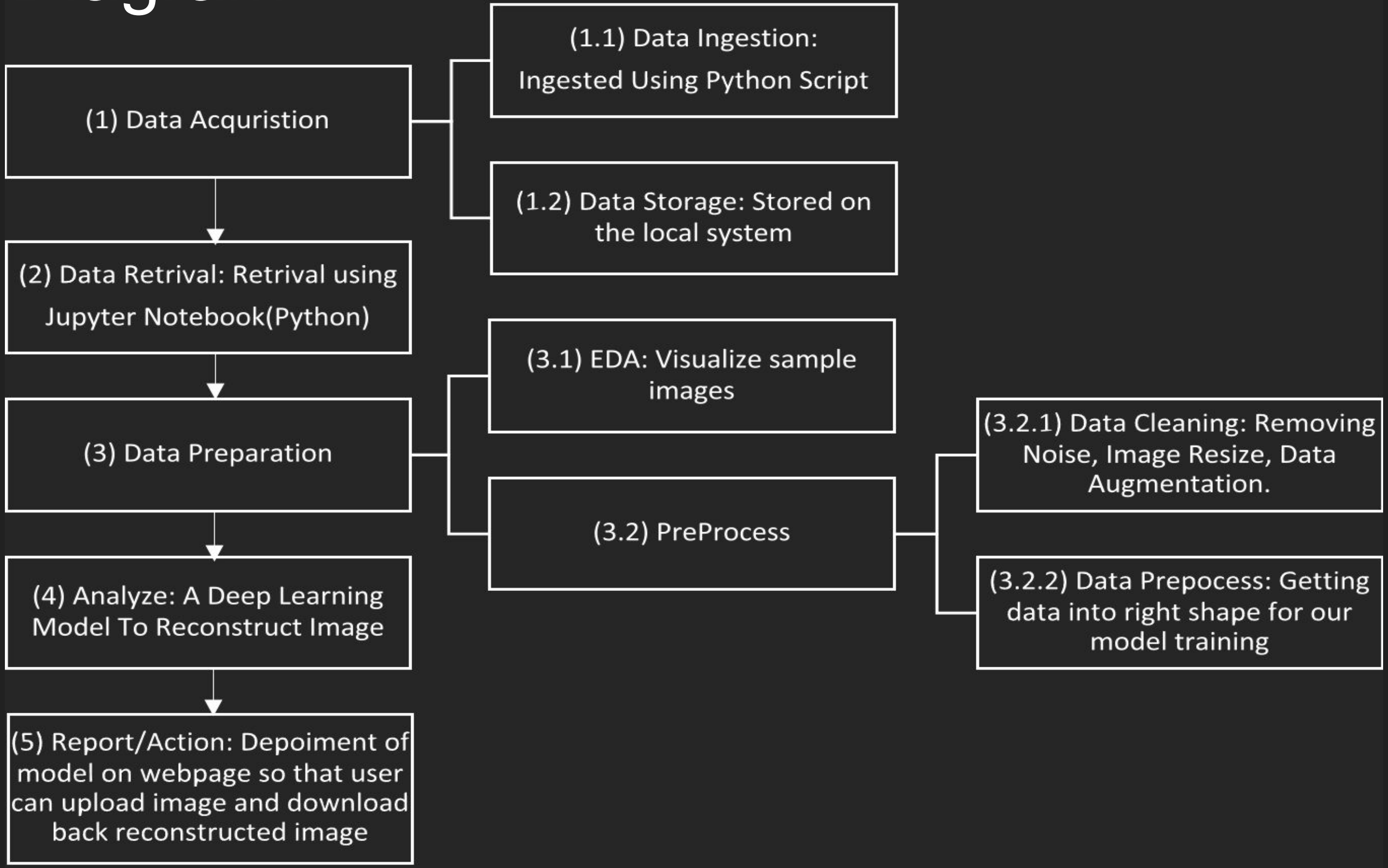
We have selected this dataset because it is required to train our model for the solution to the given problem. We need data that have some discontinuity or similar to the given image examples and thus we ingested these images to our Data Folder using the sources given below with the help of Python Script:-

1. Google Images

Data Inside Folder :-

1. Martand Sun Temple Damaged Sculptures
2. Ancient Damaged Wall Painting of India
3. Ellora Caves Damaged Sculptures
4. Ajanta Caves Wall Painting
5. Khajuraho Damaged Sculptures

Flow Diagram



(1) Data Acquristion

(1.1) Data Ingestion: Ingested Using Python Script

(1.2) Data Storage: Stored on the local system

(2) Data Retrival: Retrival using Jupyter Notebook(Python)

(3) Data Preparation

(3.1) EDA: Visualize sample images

(3.2) PreProcess

(3.2.1) Data Cleaning: Removing Noise, Image Resize, Data Augmentation.

(3.2.2) Data Prepocess: Getting data into right shape for our model training

(4) Analyze: A Deep Learning Model To Reconstruct Image

(5) Report/Action: Depoiment of model on webpage so that user can upload image and download back reconstructed image

Data Acquisition & Storage

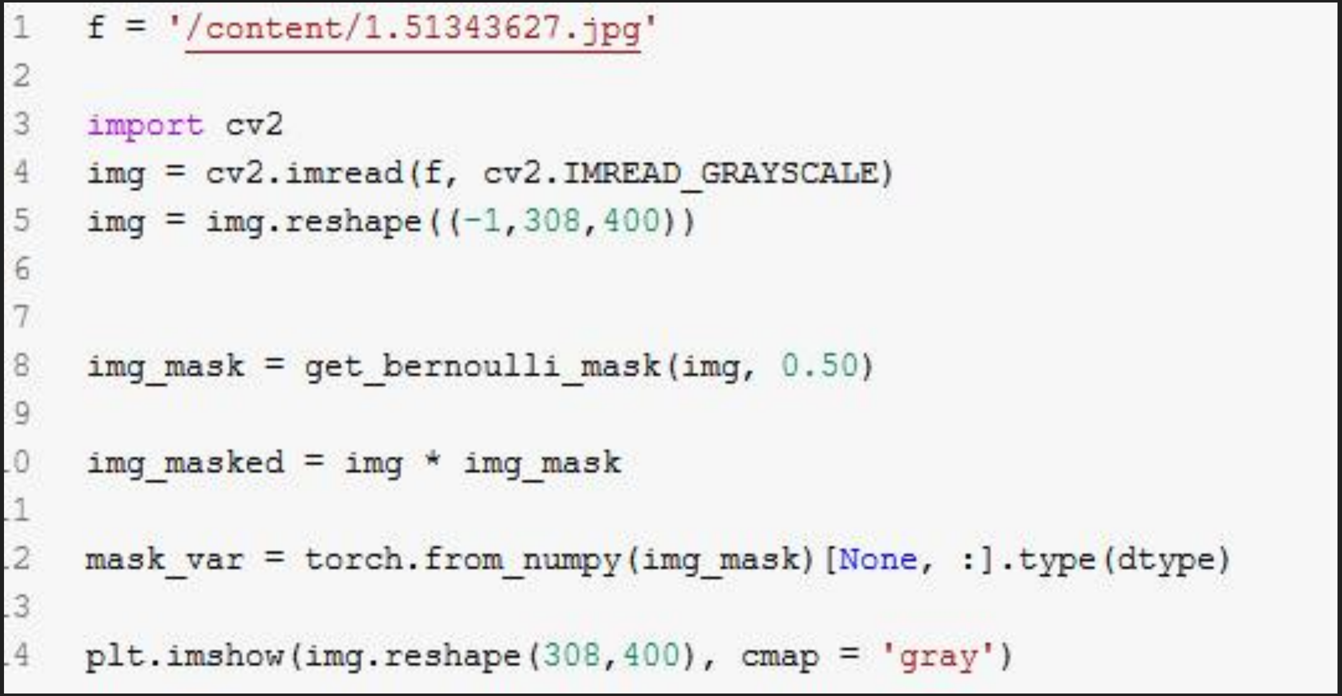
1. We ingested data to our local system using Python Script from Google Images.
2. After Ingestion, Generally we store data on some database , But for phase 1 and smaller set of problem we used our local system as storage.

Link to google_image_download : https://pypi.org/project/google_images_download/

Acquired Data : https://github.com/ravichaubey/Hack4Heritage-Hackathon/tree/master/Data

Data Retrieval

We have retrieved our data from local using Python (OpenCV) :-



```
f = '/content/1.51343627.jpg'

import cv2
img = cv2.imread(f, cv2.IMREAD_GRAYSCALE)
img = img.reshape((-1,308,400))


img_mask = get_bernoulli_mask(img, 0.50)

img_masked = img * img_mask

mask_var = torch.from_numpy(img_mask)[None, :].type(dtype)

plt.imshow(img.reshape(308,400), cmap = 'gray')
```

Data Preparation

For phase 1, We are just creating a simpler solution so we have resized image to a particular pixel. Later for complete solution we are going to build a Pipeline to read image, resize image image and augment image.

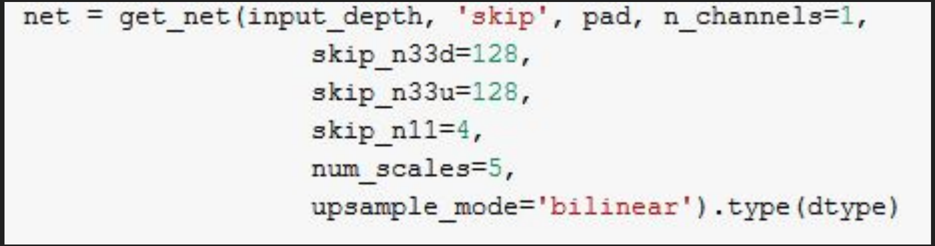
Total Trainable Params: 2217573

Data Analyse

We have uses encoder-decoder with skip connections . Encoder downsample our image and decoder generate a new image by using upsample the output of encoder.

Parameters of Network are below:

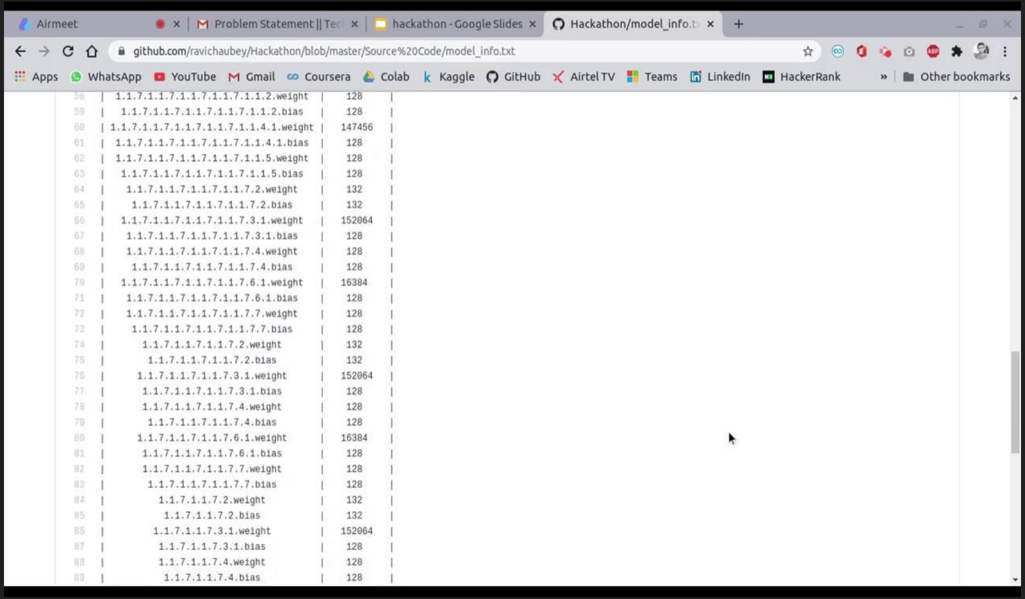
1. Activation Function : LeakyReLU
2. Upsample Mode: bilinear
3. Downsample Mode : stride
4. Total Trainable Params: 2217573



```
net = get_net(input_depth, 'skip', pad, n_channels=1,
                           skip_n33d=128,
                           skip_n33u=128,
                           skip_n11=4,
                           num_scales=5,
                           upsample_mode='bilinear').type(dtype)
```

Model Information

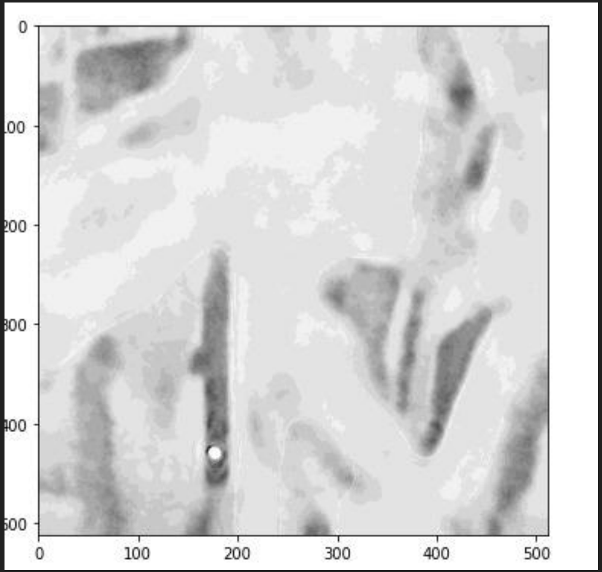
We are providing video to show the architecture and parameters for phase 1 model :-



```
|  1.1.7.1.1.7.1.1.7.1.1.7.1.1.2.weight  |   128   |
|   1.1.7.1.1.7.1.1.7.1.1.7.1.1.2.bias   |   128   |
| 1.1.7.1.1.7.1.1.7.1.1.7.1.1.4.1.weight |  147456 |
|  1.1.7.1.1.7.1.1.7.1.1.7.1.1.4.1.bias  |   128   |
|  1.1.7.1.1.7.1.1.7.1.1.7.1.1.5.weight  |   128   |
|   1.1.7.1.1.7.1.1.7.1.1.7.1.1.5.bias   |   128   |
|    1.1.7.1.1.7.1.1.7.1.1.7.2.weight    |   132   |
|     1.1.7.1.1.7.1.1.7.1.1.7.2.bias     |   132   |
|   1.1.7.1.1.7.1.1.7.1.1.7.3.1.weight   |  152064 |
|    1.1.7.1.1.7.1.1.7.1.1.7.3.1.bias    |   128   |
|    1.1.7.1.1.7.1.1.7.1.1.7.4.weight    |   128   |
|     1.1.7.1.1.7.1.1.7.1.1.7.4.bias     |   128   |
|   1.1.7.1.1.7.1.1.7.1.1.7.6.1.weight   |  16384  |
|    1.1.7.1.1.7.1.1.7.1.1.7.6.1.bias    |   128   |
|    1.1.7.1.1.7.1.1.7.1.1.7.7.weight    |   128   |
|     1.1.7.1.1.7.1.1.7.1.1.7.7.bias     |   128   |
|       1.1.7.1.1.7.1.1.7.2.weight       |   132   |
|        1.1.7.1.1.7.1.1.7.2.bias        |   132   |
|      1.1.7.1.1.7.1.1.7.3.1.weight      |  152064 |
|       1.1.7.1.1.7.1.1.7.3.1.bias       |   128   |
|       1.1.7.1.1.7.1.1.7.4.weight       |   128   |
|        1.1.7.1.1.7.1.1.7.4.bias        |   128   |
|      1.1.7.1.1.7.1.1.7.6.1.weight      |  16384  |
|       1.1.7.1.1.7.1.1.7.6.1.bias       |   128   |
|       1.1.7.1.1.7.1.1.7.7.weight       |   128   |
|        1.1.7.1.1.7.1.1.7.7.bias        |   128   |
|          1.1.7.1.1.7.2.weight          |   132   |
|           1.1.7.1.1.7.2.bias           |   132   |
|         1.1.7.1.1.7.3.1.weight         |  152064 |
|          1.1.7.1.1.7.3.1.bias          |   128   |
|          1.1.7.1.1.7.4.weight          |   128   |
|           1.1.7.1.1.7.4.bias           |   128   |
```

Model Output

We are sharing output, Please remember we need lot of iteration to get actual regenerated image because problem is complex. Due to lack resources we can not afford such large training in this short time. So we are sharing snap of training number 11000 approx , We will optimize this in Phase 2.



Reporting and Action

Basically reporting our action of our deep learning model is to bring system to work for users. Reporting is either reporting visuals or reporting information like metrics or some relationship. In this case we are reducing _loss_function_name_.

Action :-

We are going to create a webpage to deploy our model. User can upload an image and get back re-generated image on same webpage. ‘Please note that we are not going to show this webpage in Phase 1, due to lack of time. But we are prepared for phase 2.’

Thank you !!!