cs512 Project Report

Ayman FAHSI – A20440820

Mouhammad BAZZI – A20522180

Department of Computer Science

Illinois Institute of Technology

FACIAL IMAGE INPAINTING

I. TASK DELEGATION AND MEMBER RESPONSIBILITIES

Ayman was responsible for dataset sourcing, masking, and preprocessing. Mouhammad was responsible for implementing the convolution layers. Both worked on tweaking and fine-tunnning the model architecture for training, (implementing custom loss functions, assembling general architecture, and debugging). The work was equally split on writing this report as well as composing the presentation.

II. PROBLEM STATEMENT

Image inpainting is the process of taking images with missing pixel values, or otherwise corrupted images, and restoring them to their original form. The use of deep learning in predicting and generating the missing image regions has become the gold standard.

The task of image inpainting is laden with challenges due to the intricate nature of natural and photorealistic images, the diverse nature of missing regions, and the demand for high quality results. Moreover, conventional methods for image inpainting frequently render unauthentic and unfocused outputs, especially so in the cases of large and irregular missing regions. In recent times, deep learning techniques have demonstrated promising potential for image inpainting, but several issues and limitations still exist and need to be addressed, including the ability to accommodate irregular missing regions, and the ability to generate visually coherent and high-quality images.

III. PROPOSED SOLUTION

We divide this section into three to cover Data Sets, Deformable convolution and associated layers, the overall Model Architecture, and the metrics used for the evaluation.

A. Data Sets (QD_IMD and CelebA_HQ)

We will use a subset of high-quality images from the largescale faces and attributes dataset CelebA and apply masks from the Quick Draw Irregular Masks Dataset. The CelebA_HQ dataset is generated following the steps in the paper *Progressive Growing of GANs for Improved Quality, Stability, and Variation.* [1] The new subset will comprise of 30,000 images, and as in the paper, 27,000 will be used for training and 3,000 for testing. We will take the first 30,000 masks supplied by the QD_IMD dataset and use those to corrupt the images via OpenCV's bitwise_and() function to overlay the masks onto the faces. This is performed via custom script implemented by us. This script converted all images from xxxxx.PNG to xxxxx.JPG, which is more suitable for photorealistic images. Only then would the xxxxx.JPG images be corrupted and partitioned.

B. Deformable convolution and Implemented Layers

In order to address the inpainting image problem more efficiently, we need to use and define some special layers:

a. Deformable Convolution

Deformable convolution is an augmentation of standard convolution that allows the convolution to be spatially warped by some offset. The advantages of this method compared to standard convolution include being able to adapt to complex input data and better encapsulate spatially varying features in an image. This specific component is implemented with **OpenMMLab's DeformConv2D** layer.

b. Deformable Multi-Head Attention

The multi-head attention is the main component of the transformer architecture that revolutionized deep network training. The layer takes in input from the previous layer and generates three matrices, a Query, Key, and Value. The Query matrix is used to obtain the weights for the attention operation and represents the current position of the input sequence. The Key matrix represents the entire input sequence and is used to compute the similarity between the current position and all other positions in the sequence. Finally, the Value matrix contains information about the input sequence that is used to generate the output of the attention operation. Because each matrix is able to examine different aspects of the input from different perspectives, the model is more equipped to detect complex relationships within the data.

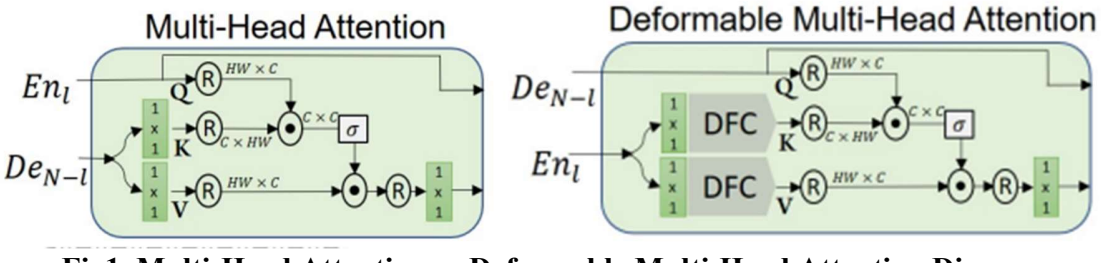


Fig1. Multi-Head Attention vs Deformable Multi-Head Attention Diagram

The above images compare the standard Multi-Head Attention to the Deformable Multi-Head Attention proposed and used in the paper. All the above-stated principles and results are the same, but note that in the proposed model, we use the aforementioned **deformable convolution layer** to further process the inputs. Doing so allows us to benefit from both the **speed** and **accuracy** of the Attention

while utilizing the full **depth of spatial complexity** extracted by Deformable Convolution. We implement this sublayer using **OpenMMLab's DeformConv2D layer, Pytorch's built-in matmul, Conv2D, and SoftMax**.

c. Gated Feed Forward Layer

The purpose of the GFFL is to utilize a gating mechanism to selectively allow features to pass through. It allows to suppression of any undesired features if present. To do so we take our input, convolve it, and pass it through a Gelu (Gaussian Error Linear Unit) and we add to this the same input but convolved with another convolution layer.

This is implemented via Pytorch's Gelu (Gaussian Error) and Conv2D layers.

d. Nested Deformable Multi-Head Attention Layer

The final proposed layer is a culmination of the previously explained sublayers as described in the following diagram:

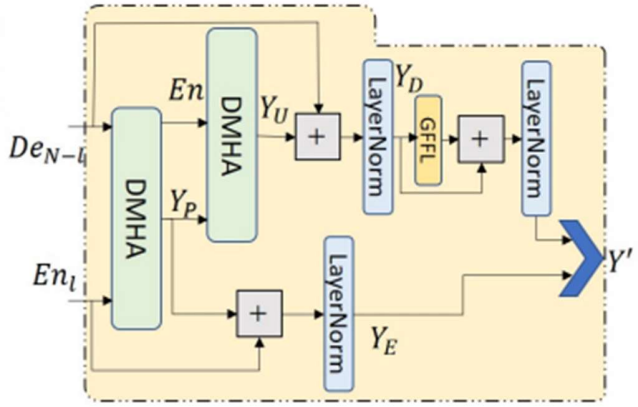


Fig2. Nested Deformable Multi-Head Attention Layer Diagram

This NDMHL is designed to capture long-term dependencies and extract valid features from maximum receptive fields. It achieves this by using a multi-head attention mechanism that allows the network to focus on different parts of the image and selectively attend to relevant information.

By using this approach, we are able to generate globally and locally consistent realistic images that closely resemble the original target images.

e. Gated Convolution Layer

This layer combines convolutions and gating mechanisms to selectively **control the flow of information**. It performs two convolutions on the input tensor: one for the feature map and another for the gating map. The gating map goes through a sigmoid activation function. The outputs of both functions are element-wise multiplied, enabling the model to retain or suppress features as needed. This layer helps improve model performance by handling long-range dependencies and focusing on relevant features

This is implemented via Pytorch's built in Conv2D and Sigmoid Modules.

f. Gated Deconvolution Layer

This layer performs standard Deconvolution (transposed convolution), in that it up samples the feature maps from a lower resolution to a higher resolution. The gating mechanism is used here to selectively

allow up-sampled features to pass through. In essence, it performs the inverse convolution operation reversing the effects of the gated convolution.
This is implemented via Pytorch's ConvTranspose2d and Sigmoid Modules.

C. Model Architecture & Hyperparameters

Since all the previous classes are implemented, building the model comes down to assembling together the layers to form the network described in the following diagram:

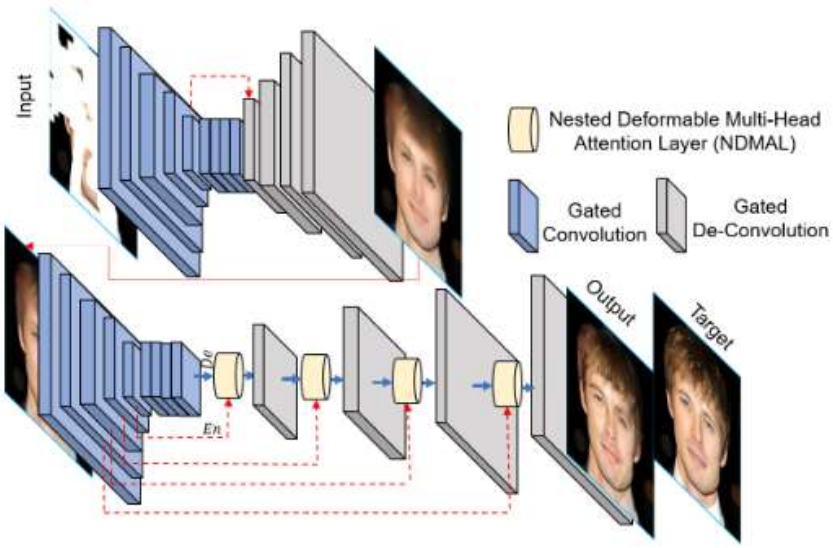


Fig3. Model Architecture

The proposed approach uses a coarse-to-fine architecture with a nested deformable multi-head attention layer (NDMAL) to efficiently capture long-term dependencies in image inpainting tasks. The coarse stage's encoder and decoder layers employ gated convolution and Leaky ReLu activation. The fine stage utilizes NDMAL to process encoder layer outputs and decoder feature keys/values while preserving valid content. The coarse output provides contextual information to the fine stage.

The NDMAL utilizes a single 8x8 (8x8 is the window size) attention block to capture long-term dependencies while minimizing computational costs. This architecture focuses on different receptive fields, generating high-quality inpainted results, and maintaining linear complexity by applying attention channel-wise instead of spatially.

It is important to note that this model takes as input the **mask** along with the **corrupted image** with holes to guide the inpainting process during training and testing. The images are **normalized** to be between 0 and 1.

We use the Adam optimizer with a learning rate of 0.0002, beta_1 = 0.5, and beta_2 = 0.99. The ideal number of epochs should be equal to 200 with a batch size of 1.

We will also try to use a loss function that contains 4 terms: the **L1 loss** (to optimize the network for better reconstruction by minimizing the absolute difference between the inpainted and target images), the **adversarial loss** (to enforce global and local consistency in the generated image by solving a min-max problem between the generator and discriminator), the **perceptual loss** (to capture textural and structural information by comparing deep feature maps of ground-truth and inpainted images using a pre-trained VGG19 model) and finally the **edge**

loss (to emphasize edge enhancement during training by comparing the edge maps of the target and inpainted images, and this using the Sobel operator).

D. Baseline Model

We implement our own autoencoder-based model with Unet-like skip connections to cerate a baseline level of confidence in the efficiency of the proposed model.

1. Model architecture
   The architecture of the model is that of an Encoder-Decoder structure. The Encoder section consists of 4 convolution blocks, which are built using a Conv2D layer, a Batch normalization layer, and an activation function. Each block is followed up by a MaxPooling layer. The blocks up sample the input by increasing the number of filters by powers of two; 32, 64, 128, 256. A fifth and final conv block is implemented and applies a bottleneck effect on the inputs and has 512 filters.
   The Decoder section consists of 4 transpose convolution blocks. Each block begins with a standard Conv2DTranspose layer with filters descending in powers of two; 256, 128, 64, 32. We then implement the skip connections by concatenating the transpose convolution output with the corresponding convolution block output from the encoder; final encoder layer with first decoder layer, first encoder layer with final decoder layer, all with matching filters. The resulting concatenation is then convoluted before passing it down to the next Conv2DTranspose layer. The final layer uses the sigmoid activation function to output values between 0-1, which when scaled, result in pixel values.
2. Model Parameters
   We have 4 convolution blocks in the encoder and 4 in the decoder. They increase and decrease in powers of 2 respectively. The convolution blocks in both encoder and decoder use a 3x3 kernel size, stride of 1, ReLu activation, and batch normalization. The decoder transpose convolution layers use a 2x2 kernel size, a stride of 2 in order to up sample, and are then simply concatenated
   The encoder and decoder sections are joined together by a bottleneck convolution block, with 512 filters, kernel size of 3x3, stride of 1, ReLu activation, and batch normalization. Below are the parameters for our baseline model:

```
Total params: 3,838,851
Trainable params: 3,835,907
Non-trainable params: 2,944
```

E. Model Evaluation: Metrics

In order to evaluate the models, we will use 5 different metrics:

1. **PSNR (Peak Signal-to-Noise Ratio):** It measures the quality of reconstructed images by comparing them to the original images. Higher PSNR indicates better image quality. We aim to maximize PSNR. It is unbounded.
2. **SSIM (Structural Similarity Index Measure)**: It evaluates the similarity between two images by considering luminance, contrast, and structure. A value closer to 1 indicates higher similarity. We aim to maximize SSIM. It is bounded between -1 and 1.
3. **L1 Loss**: It calculates the absolute differences between the target and predicted values. Lower L1 loss indicates better model performance. We aim to minimize L1 Loss. It is bounded (lower bound) at 0 (perfect prediction) but unbounded otherwise.

4. **LPIPS (Learned Perceptual Image Patch Similarity):** It measures the perceptual similarity between two images by comparing their feature representations from a pre-trained deep network. Lower LPIPS values indicate higher similarity. We aim to minimize LPIPS. It is bounded (lower bound) at 0 (perfect prediction) but unbounded otherwise.
5. **FID (Fréchet Inception Distance):** is a metric used to evaluate the quality of generated images, such as those produced by Generative Adversarial Networks (GANs). It compares the distribution of features from real images and generated images using a pre-trained Inception network. Lower FID values indicate that the generated images are closer to the real images in terms of quality and diversity. We aim to minimize FID. It is unbounded.

IV. IMPLEMENTATION DETAILS

We divide this section into four to cover *data sets importations issues*, *class implementation issues*, *model architecture & loss function issues*, and *hyperparameters selection and training*.

A. Data Sets importations issues

The original paper uses four distinct datasets, NVIDIA Irregular masks, QD_IMD, CelebA_HQ, and Places2. NVIDIA Irregular masks is a mask dataset with sharper edges, corners, and more box-like structures. This dataset was not publicly available nor were the instructions on how to produce a dataset of similar nature. For this reason, we were unable to use this mask dataset.

Fig 4. NVIDIA masks applied onto sample Places2 image

Places2 is the second iteration of the Places365 dataset, which amounts to roughly 10 million images originating from 365 different locations. We were unable to use this dataset for two main reasons. Firstly, the paper applied the NVIDIA masks onto the place's dataset, but as previously mentioned, those masks were not public. Secondly, the sheer volume of data was not manageable by our team due to hardware constraints.

Our team was, however, able to use the other two datasets mentioned in the paper as intended. The prime reasoning behind applying QD_IMD onto CelebA_HQ and NVIDIA onto Places2 was due to the nature of the masks. NVIDIA's masks had no "human" aspect and the resulting inpainted regions would be produced more roughly. Quick Draw on the other hand is much more reflective of the curvature of the human face, and for that specific case, would result in more accurately inpainted regions.

B. Custom Class Implementations

In order to build the model architecture, we had to implement several custom Pytorch subclasses, a very demanding task considering the multitude of issues and technical difficulties. In this section we discuss the key problems we faced while trying to implement these classes:

Our first task was to implement the Deformable Multi-Head as described below:

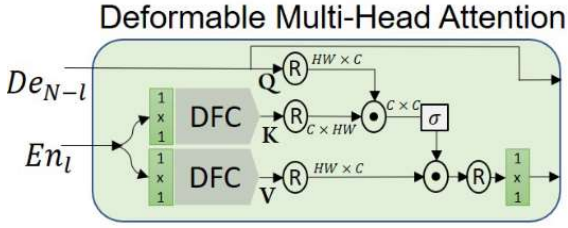


Fig 5. Diagram of the Deformable Multi-Head Attention

But to do so we need to have a Deformable Convolution layer. The first thing we did is to check if this layer is built in to the TensorFlow framework that we familiar with. Unfortunately, the TensorFlow documentation had not trace of a deformable convolution layer. We then explored alternative solutions to implement it within TensorFlow and found several public resources available on GitHub or other libraries that have already implemented this deformable convolution layer.

While we were able to find several implementations of this layer, none of them functioned as expected. We consistently encountered compilation errors, deprecated warnings, and library version inconsistencies. In order to address these issues, we attempted to modify the class to fix the compilations issues but we failed and had no more ideas to get our deformable convolution ready for use.

As a result, we decided to explore other alternatives for implementing deformable convolution. But after facing several more failures, we ultimately chose to switch to the Pytorch framework. This presented an additional challenge as we had limited prior experience with Pytorch, requiring us to *discover* and *learn* how to use it from scratch.

After familiarizing ourselves with Pytorch, we successfully found a library that implemented deformable convolution as expected. With this critical component in place, we proceeded to implement the Deformable Multi-Head Attention (DMHA) and, subsequently, the Nested Deformable Multi-Head Attention Layer (NDMHAL).

Also, in order to implement the NDMHAL we had to create a Gated Feed Forward Layer (GGFL), which was not fully addressed by the paper. It was a quite unclear on its behavior with relation to the model and we are still unsure of our implementation because there is no basis for comparison between our work and that which is discussed in the paper.

Finally, creating the gated convolution and deconvolution class was fairly straightforward, especially since a lot of resources on the internet explain how to implement it.

C. Baseline model

We implemented the model using the TensorFlow framework and had minimal difficultly in implementing and training the model. However, we would like to note that training the model was very time consuming on our local devices and required us to train through Google Colab. We still experienced difficulties in training due to low system and GPU RAM, so we purchased Colab Pro and were able to train our models. This only allowed us to train with 750 target images, 750, masks, and 750 corrupted images.

D. Model Architecture & Loss function

Another significant challenge we encountered was implementing the model architecture. The difficulty, however, did not stem from a lack of knowledge on how to create the architecture, especially after gaining experience with Pytorch through the implementation of the previous classes.

Rather, the uncertainty was about the specific details of the model posed the main challenge. The description of the network in the source material was quite vague, which left us to interpret the model architecture figures and estimate the number of layers, layer dimensions, and other key parameters. Unfortunately, there was no code repository available for reference to clarify the exact architecture. Despite these challenges, we believe that our implementation closely resembles the intended design.

We also ran into difficulties implementing the loss function, because it was composed of 4 distinct terms. We successfully implemented 3 of the 4 terms, but the final loss term was related to some unexplained adversarial loss. We were unable to understand what the discriminator and generator mentioned in the paper were and were then unable to define this part of the loss. Due to the lack of information offered in the paper, we decided to remove this part from the loss formula. We believe this not to be super impactful because their results indicate varying metrics were collected by the authors when training their model, and they did so without the adversarial term and achieved similar results (but obviously not as good as with this adversarial loss).

Similar to training the baseline model, we experienced difficulties in training the model due to hardware constraints and resolved them through Google Colab Pro. We trained on 750 target images, masks, and corrupted images.

E. Hyperparameters selection and training

Ultimately, we faced challenges in determining the hyperparameters of our model and the training process of the model. Since our primary goal was to replicate the paper's results, we opted to use the hyperparameters provided by the authors. However, some of the hyperparameters were not given, so we had to rely on our own inferences. But we encountered three main issues:

1. **Insufficient computational power**: The paper mentioned training on 30,000 images for 200 epochs, which took the authors three days. Of course, our available computational resources are significantly less. Our plan was to train for fewer epochs using a subset of images. However, even training a single image took on average 15 seconds for training, which is super slow (that means that training on 500 images for 20 epochs would take more than a day and a half for training). We can clearly see that it's impossible to train efficiently and in good conditions and this, even when using cloud services such as Google Colab. (Note that 500 images represent only 1.6% of the entire data set and we are not even able to train for more than one day without interruption on Google Colab).
2. **Model divergence**: The second main problem is that our model diverged super quickly: after training on only one image, it diverges (we know that because the loss function is equal to NaN after fitting one image). We suspected that the learning rate might be too high, and after reducing it, the model diverged after six images instead of one. Although this remained to be a significant issue, we were at least able to identify some potential causes, one being the complexity of the model itself. After a few iterations of reducing complexity through the hyperparameters, such as filter count, image size, etc, we decided to reduce the size of the proposed model such that only 2 NDMHA layers exist in the decoder section of the model. This itself resolved many issues, but we still occasionally found the model diverging after the first 2 images. After several trial runs, we concluded this to be an effect of the random initialization of the weights, and were not able to resolve them due to not knowing how to fix the initialization. Another component that added to the model divergence issue was that our implementation of the edge loss term would occasionally result in a division by 0, thus leading the model to diverge. We were able to resolve this by adding an arbitrarily small epsilon value such that division by 0 would not occur. However, it was concerning that the learning rate provided in the paper did not suit our implementation and this may suggest that there might have been aspects of the model not adequately explained in the paper or some imperfections in our implementation.
3. **Failed metric and loss term implementations:** The paper mentioned 5 metrics, L1, PSNR, LPIPS, FID, SSIM, and 4 loss terms, L1, Edge loss, Perceptual loss, and Adversarial loss. We were not able to implement the Adversarial loss, due to a lack of information provided form the paper regarding the generator and discriminator. The FID metric is based on GAN, so we also disregarded this metric. We failed to implement the LPIPS metric despite attempting multiple libraries across both frameworks. Manual implementation also failed due to a clash of tensor importation mechanisms. Libraries such as piq

and LPIPS also failed due to similar tensor mismatches between NumPy, Tensor, and torch.Tensor. We were unable to resolve these issues so we removed them from consideration.

These challenges highlighted the difficulties in reproducing the results presented in the paper, mainly due to the lack of detailed explanations, unavailability of the authors' implementation, and limitations in computational resources.

Some instructions on how to use the code are provided in the project folder in the README file.

V. RESULTS AND DISCUSSION

As discussed in the implementation details section, we encountered multiple challenges that prevented us from successfully running the code and training the model. Consequently, our results are not up to our own standards. Below, we will discuss the results of the baseline model, our implementation of the proposed model, and the findings established in the paper

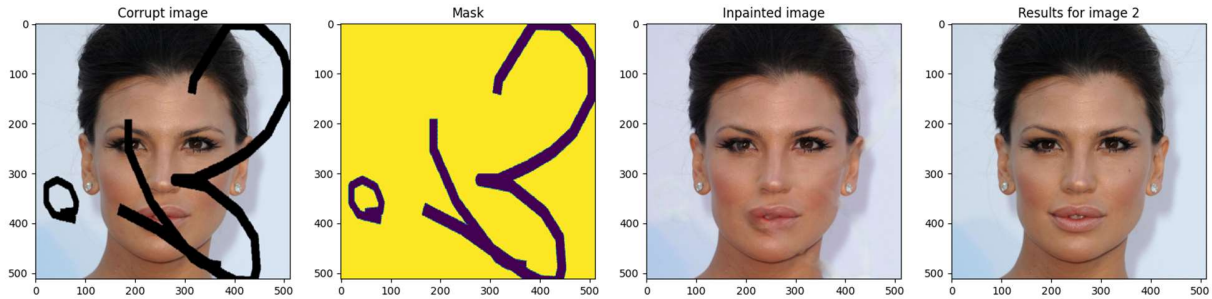
A. Baseline Model Results

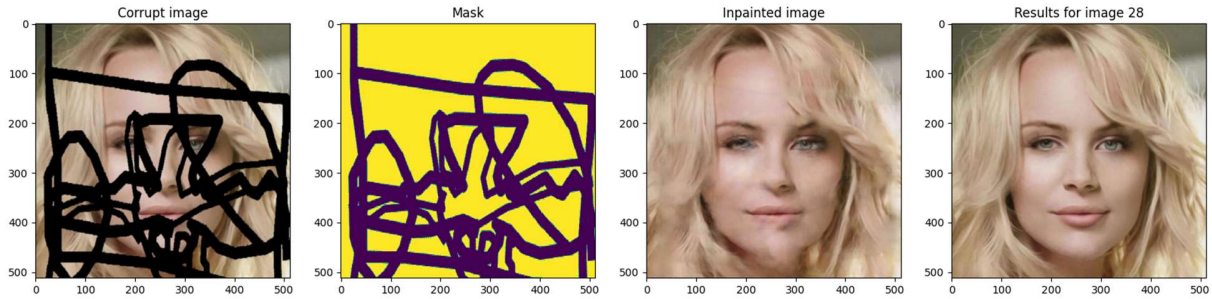
To gather our results, we trained on a training set and used a validation set to tune some parameters. We used a 60, 20, 20 split for training, validation, and test respectively. After tuning hyper parameters, we concatenated our training and validation sets to form one large test set, and trained the model on it. Based on our metrics, this method greatly increased the results of our testing. Overall, we believe our baseline model performed up to our standards and produces coherent and plausible outputs.

MODEL TRAINING

The above graphs display the loss, PSNR, SSIM, and L1 metrics per epoch. Across all graphs, we can see that the validation surpasses the training data at around 5-7 epochs across all metrics. However, continuing to train further does not result in overfitting, so we should continue to train the model.

MODEL PREDICTIONS





As we can see above, the baseline model produces a visually coherent image with generally correctly inpainted regions, although the inpainted facial features such as eyes and mouths, are quite blurry and in most cases inaccurate.

TEST METRICS

TEST SET EVALUATIONS

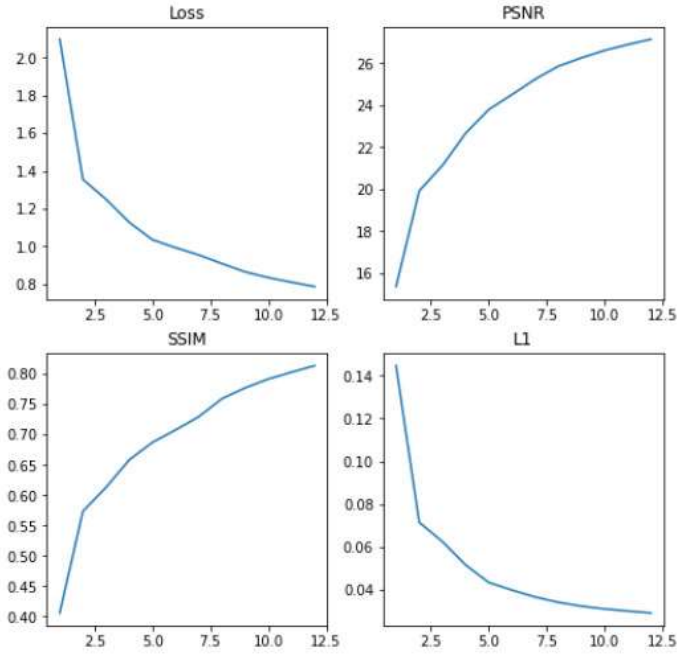
Metric	Value
Loss	0.8365
L1	0.0157
PSNR	31.7106
SSIM	0.9245

We include the above graph to show that after merging the data, our training appears much more stable and results in more accurate data. We confirm our model to be a suitable baseline for comparison with

B. Proposed NDMHA Model

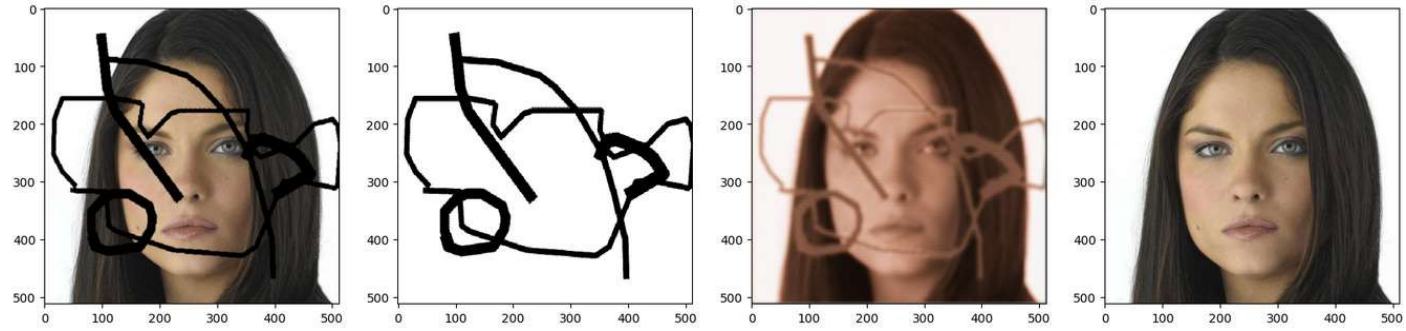
Below are our findings when training our implementation of the proposed model. We were only abel to train on 750 images for 12 epochs.

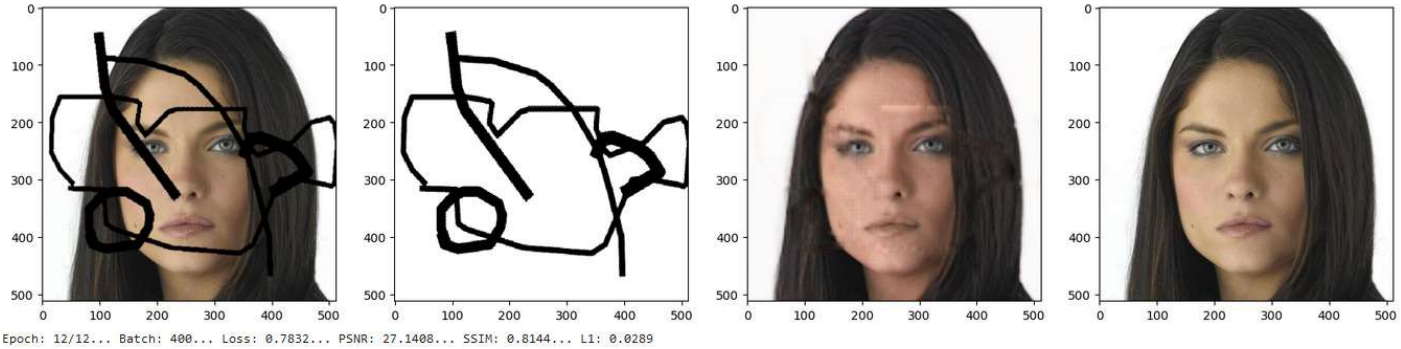
MODEL TRAINING



The above graph displays the metrics over the course of the training, displaying a strong and stable growth pattern. More time and data would prove the model to be even stronger.

MODEL PREDICTIONS

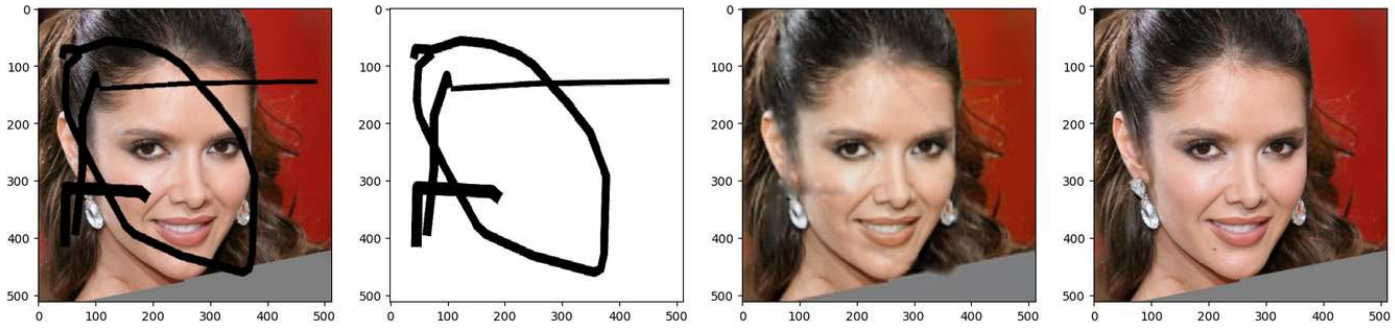




The images above are examples from the model's training period, specifically form epoch 2 and epoch 12. The above predictions seem to be of much worse quality compared to the ground truth images, but considering how little training time and data have been allowed, the growth of the model is very hard to ignore. We can also see a stark increase in the metrics, specifically in PSNR, which grows to **27.14**. For comparison, the paper concludes with a **PSNR of 28.19.**

TEST PREDICTION

Average inference time: 0.0109



The above image is a prediction made using the fully trained model. We notice comparable level of resolution in the prediction and ground truth images, but suffer colorization losses, and still suffer from blurry regions. This image comes from the test set and results in a test **PSNR of 26.4289.**

In order for our implementation to reach similar standing with regard to the proposed model, we would need much more data and to train on way more epochs. For comparison's sake, we trained our implementation on 750 images, masks, and targets, for 12 epochs due to hardware constraints and incredible training resistance with regards to divergence and the lack of a full loss term. The paper trained on 27,00 images, with two variations of masks, trained for 200 epochs, and trained for 3 days. We believe the proposed model to be as strong as it has claimed to be, but we are unable to fully validate their findings due to the aforementioned issues.

C. Research Findings

The authors of the paper did, however provide samples of their benchmark comparisons with existing models, depicted below. As we can visually confirm, the proposed model generates realistic images that are strikingly similar to the ground truth image, despite the gaps of information brought in by the corrupted images. The related table of metric comparisons also shows their model to be smaller, faster, and overall, more efficient with strikingly greater results than the existing models. We are unable to verify their findings because as previously stated, neither their code nor their models are publicly available.

Fig 6. Benchmark comparison of proposed and existing models (outputs comparison)

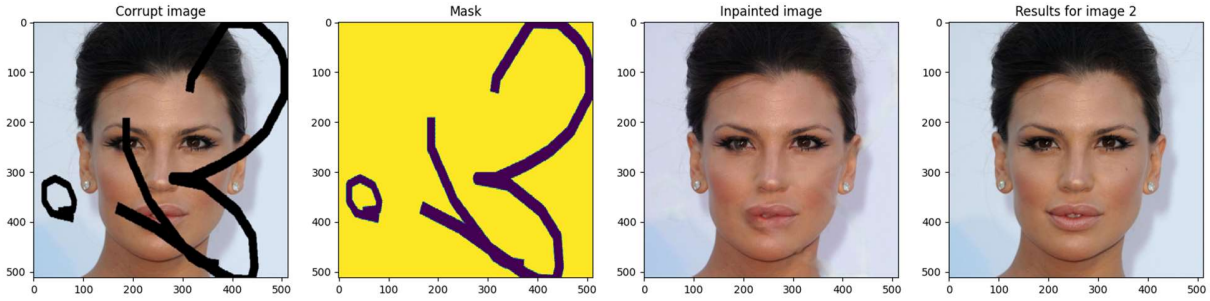
Method	PSNR	Parameters (Millions)	Run Time	GMAC
GMCNN [3]	24.59	3.115	0.85	-
SN [4]	25.09	54.94	0.12	140.20
EC [5]	26.55	53	0.25	257.96
GConv [6]	25.74	4.05	0.91	111.14
PIC [7]	25.46	3.636	0.98	-
RFR [8]	27.04	31	0.34	412.22
HR [9]	27.36	30	0.22	47.70
CTSDG [10]	27.61	52.14	0.28	61.08
MAT [11]	27.75	60	0.78	435.30
Ours	**28.19**	4.1	**0.08**	15.01

Fig 7. Benchmark comparison of proposed and existing (performance comparison)

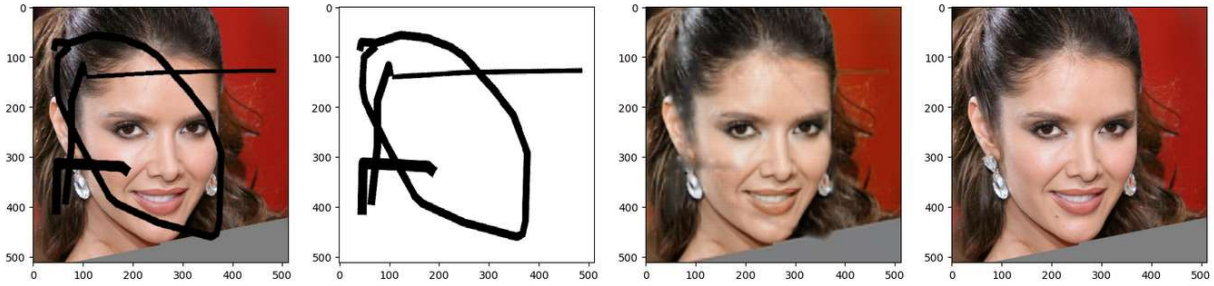
D. CONCLUTION

Comparing our baseline model with our implementation of the proposed model, we can confidently conclude that our baseline model slightly exceeds the performance of the proposed implementation. HOWEVER, this is only due to our metric and loss value failures, our inability to train the full complexity of the model, and the hardware restrictions. Without these massive roadblocks, we wholeheartedly support the findings presented in the paper. We believe that with more suitable training conditions, we can certainly reach the same findings discussed in the paper.

BASELINE MODEL



PROPOSED MODEL



However, we can still discuss what we had learn from this project:

Indeed, even if we don't have any results and we do not really know if we succeed in accurately implementing the proposed model in the paper, we gained valuable knowledge about various new kinds of layers with their advantages and disadvantages (for example Deformable Convolution, Nested Deformable Multi-Head Attention, Gated Convolution, etc.). We also discovered a few new metrics and their associated interpretation, and implementation etc. Most importantly, we feel, is that we acquired a first-hand, strong experience with Pytorch, one of the most important frameworks in deep learning.

So overall, this project set a challenging and ambitious goal and in turn provided a rewarding learning experience, despite and because of the technical difficulties we faced during the training process.

E. REFERENCES

[1] Shruti S Phutke and Subrahmanyam Murala. Nested Deformable Multi-head Attention for Facial Image Inpainting. CVPR Lab, Indian Institute of Technology Ropar, https://openaccess.thecvf.com/content/WACV2023/papers/Phutke_Nested_Deformable_MultiHead_Attention_for_Facial_Image_Inpainting_WACV_2023_paper.pdf, 2023.

[2] Ziwei Liu, Ping Luo, Xiaogang Wang and Xiaoou Tang. Large-scale CelebFaces Attributes (CelebA) Dataset. Multimedia Laboratory, The Chinese University of Hong Kong, https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html, 2020.

[3] Tero Karras, Timo Aila, Samuli Laine, Jaakko Lehtinen, Progressive Growing of GANs for Improved Quality, Stability, and Variation. ICLR, https://github.com/tkarras/progressive_growing_of_gans, 2018.