CSCI 5922 - Neural Networks and Deep Learning
Final Project Report
Image Inpainting

Kodur Krishna Chaitanya
Deptartment of Computer Science
University of Colorado at Boulder
kodur.chaitanya@colorado.edu

Dhanendra Soni
Deptartment of Computer Science
University of Colorado at Boulder
dhanendra.soni@colorado.edu

Keval Shah
Deptartment of Computer Science
University of Colorado at Boulder
keval.shah@colorado.edu

Shreshtha Pankaj
Deptartment of Computer Science
University of Colorado at Boulder
shreshtha.pankaj@colorado.edu

Abstract

Inpainting is a process where missing or damaged parts of an artwork are filled in to present a complete image that is closely similar to the original [8]. Through this project, we wish to use the power of Deep Learning and particularly GANs (Generative Adversarial Networks), to replace missing portions of a digital image with meaningful content. We base our experiments on the CelebA-HQ dataset and present the results along with a Peak-Signal-to-Noise ratio.

1 Introduction

Image inpainting is a technique to generate or estimate suitable pixel-level information to fill a missing section of an image. It is useful in many applications like image restoration, image denoising or object removal etc. It may sound an easy task but generating missing pieces of an image and make it look realistic is a challenging problem. Generating a realistic image will require a higher level understanding of various features of the image and low level detailing to perfectly match it with the surroundings in terms of color and texture.

In this work we are trying to reproduce the work of Wang et al. [12] to perform image inpainting task on images from Celebrity-256 dataset. A typical example of image inpainting task is shown in Figure 1 where the first image is corrupted by applying a mask to it, the second image is the result of our model and the third one is the ground truth image. Typically, an inpainting task makes use of pixels for specific path-wise similarity measures to address the following problems:

- **Features for Inpainting:** To find the connections between the known and the missing parts of an image, a correct feature representation is very crucial. Recent developments in learning-based algorithms have provided a way to learn these features directly from data rather than handcraft all the features. These features can be categorized into two major categories, Global and Local: Global features to understand the high-level structural details like eyes and nose position on the face and local features for specific pixel-level details like skin color and texture. CNN-based methods utilize encoder-decoder networks and are suitable for extracting features from images.
- **Reliable Similar Patches:** Neatest-neighbor search is one of the key components in both learning-based and traditional methods. Although, this technique may affect the patch generation process if the missing area has a different structure than the surroundings. Another issue is that the nearest-neighbor is computationally time-consuming during the testing phase so in this work author has used the nearest-neighbor search only in the training phase. The testing phase is efficient without the need for any post-processing.

- **Spatial-variant Constraints:** The image inpainting task needs to consider the spatial constraints into account while generating the missing pixels. For example pixels near the boundary-area are more constrained than the pixel in the center of the area. There could be multiple candidates to fill in a missing pixel and adversarial networks help learn these spatially constraints features.

The framework defined by authors is a *Generative Multi-column Convolutional Neural Network* (GMCNN). The previous networks have used multi-scale or coarse-to-fine strategies that need resized images but this network can decompose input images into components with different feature resolutions. GMCNN can directly use full-resolution images to define multi-scale feature representations to capture global and local information due to its multi-column architecture. This network also has an *implicit diversified Markov Random Field* (ID-MRF) term in the training phase. ID-MRF is used as a regularization term which helps reduce the visual artifacts.

In GAN based learning, a CNN based context encoder-decoder is used to fill out the patches in the images. The output of the encoder-decoder is used as input for GAN to generate a realistic image. Finally, the generated image is fine-tuned by VGG16 by taking into account the ID-MRF loss. This method of fine-tuning is usually used in super-resolution and this trick is borrowed from there. In our case, the conv3_2 and conv4_2 are used to find the ID-MRF Loss.

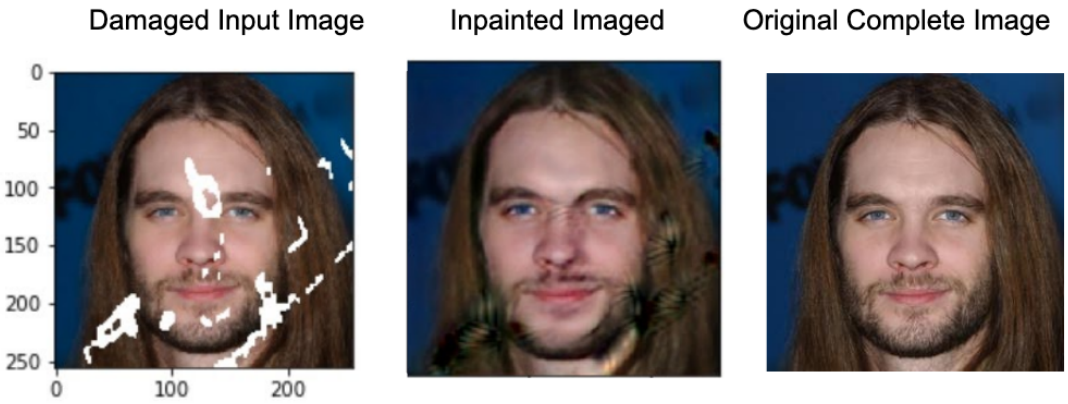

Figure 1: An example of a corrupted image with a mask, generated image from our model and the ground truth image in order from left to right

2 Related Work

Image inpainting has been an advanced and one of the most interesting problems of Computer Vision applications. On top of the tremendous progress on defining, classifying, and detecting missing objects, the most challenging part remains to generate natural images. Moreover, it is important to note that the tasks of hole-filling cannot be handled by classical texture synthesis and scene completion - which has to do with the typical cut-paste modifications based on the nearest-neighbors from a huge collection of images [3].

This blocker in Computer Vision has been cleared through the introduction of Convolutional Neural Networks (CNNs) in this sub-domain [5]. The success in the initial application of deep learning models has led to a way to tackle more challenging inputs with increasing accuracy results with minimal noise. The next player to set a landmark in fixing corrupted natural images was the integration of Generative Adversarial Networks (GAN) for the sole purpose of realistic image generation for hole-filling. Since the ultimate goal was to make the image fill undetected by the human eye, this integration into the convolutional architecture was foreseen by many researchers. In a more recent article, Radford *et al.* [11] implemented a novel convolutional architecture supported by the optimization of hyper-parameters for GAN.

In the current scenario, research groups are oriented towards finding the perfect architecture and tuning parameters to create the ultimate deep-learning powered inpainter. The initial work of Pathak *et al.* [10] introduced a CNN based upon an encoder-decoder CNN tagged with minimizing reconstruction loss and adversarial loss on each pixel. This was set as the core of the architecture that was going to be used by the upcoming papers that improved performance significantly. A U-net like neural structure that was presented [14] became a popular network layer layout as well.

In the recent work by Nazeri Kamyar *et al.* [7], the authors propose a named model, EdgeConnect that builds upon an edge generator followed by an image generator. This is one of the later implementations of explicitly incorporating image structure knowledge to produce better results. Following this approach, researchers attempted to introduce contour generators to work better with images with salient objects [13].

Our implementation is in-line with continuation from the work of Wang *et al.* [12]. The authors have developed an end-to-end trainable system that takes an input image along with a binary region, known as a mask. This mask marks each pixel with a 0 or a 1 based on the region that is to be inpainted. Their proposed Generative Multi-column Convolutional Neural Network (GMCNN), is tailored to build the missing portion of the image in two steps. The first step involves defining and setting the structure of the generated image. The following step deals with making the incomplete filled portion more realistic and tuned with the existing image. We discuss the architecture and working in detail in section 3.1

3 Approach and Methods

3.1 Network Architecture

The proposed network *Generative Multi-column Convolutional Neural Network (GMCNN)* in the original paper shown in Figure 2, primarily consists of three sub-networks: a generator network to produce the results, local and global discriminators for adversarial training and a pre-trained VGG network for ID-MRF loss calculation. Only the generator network is used while testing the network.

The generator network has three parallel encoder-decoder branches to extract the various level of feature information from input image X and applied mask M. Then a decoder module common to all three branches is used to transform these features into the original image space $\hat{Y}$. The multiple branches with different levels of configurations are useful to capture a different level of information from an image. The generator network is trained to generate optimal feature representations using the provided data.

The output of each branch is up-sampled and concatenated into a feature map F. The feature map F is further transformed into image space using the shared decoding module having two convolutional layers. This decoding module is denoted by $d(\cdot)$ provides the output $\hat{Y}$ of the generator network as $\hat{Y} = d(F)$. Minimizing the loss between this output $\hat{Y}$ and original image Y make these branches capture the required information from the input. The decoding module $d(\cdot)$ influences the generate features in each of the branches in the training phase.

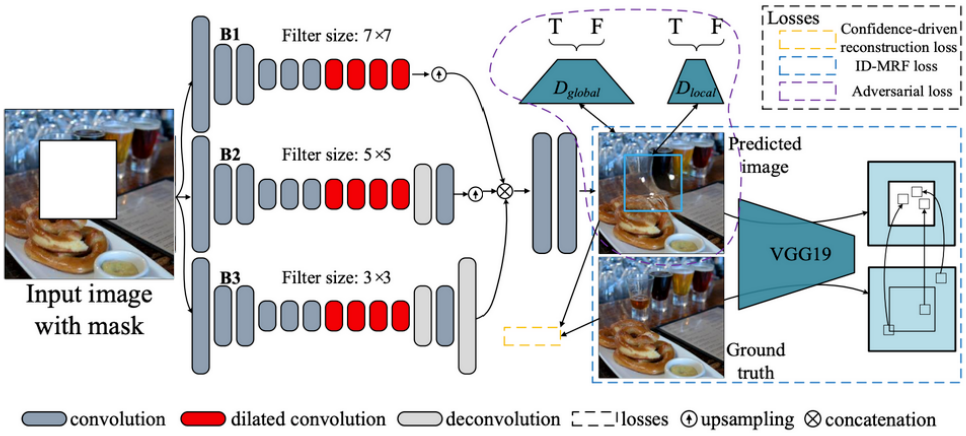

Figure 2: Overview of the Model and Network Layers. [12]

3.2 Loss Functions

We wanted to exploit the performance gains from the hybrid loss function used by [12]. Most of the loss functions are borrowed from there.

Spatial Variant Reconstruction Loss L_c: One of the major improvements in inpainting results was the usage of pixel-wise reconstruction loss. Initially, we set the confidence of known pixels as 1. Next, to transfer the confidence

of the known pixels a Gaussian filer g is used. Starting with $\mathbf{M}_w$ as the mask of the input image, we obtain a loss weight mask $\mathbf{M}_w$ as follows:

$$\mathbf{M}_w^i = (g * \bar{\mathbf{M}}^i) \odot \mathbf{M} \tag{1}$$

Here, $\odot$ is the Hadamard product. Let G be our generative model. Then, $G(|\mathbf{X}, \mathbf{M}|; \theta)$ is the output of our model G with θ being the learn-able parameters, $\mathbf{Y}$ is the ground truth image.

$$\mathbf{L}_c = ||(\mathbf{Y} - G(|\mathbf{X}, \mathbf{M}|; \theta)) \odot \mathbf{M}_w||_1 \tag{2}$$

Adversarial Loss L_{adv}: We define the adversarial loss for the generator as:-

$$\mathbf{L}_{adv} = \mathbf{E}_{\mathbf{X} \sim P_{\mathbf{X}}}[D(G(\mathbf{X}; \theta))] + \lambda_{gp} E_{\hat{\mathbf{X}} \sim P_{\hat{\mathbf{X}}}}[(||\nabla_{\hat{\mathbf{X}}} D(\hat{\mathbf{X}})) \odot \mathbf{M}_w||_2 - 1)^2] \tag{3}$$

where, $\hat{\mathbf{X}} = tG([\mathbf{X}, \mathbf{M}]); \theta) + (1 - t)\mathbf{Y}$ and $t \in [0, 1]$.

ID-MRF Loss L_{mrf}: This loss is computed at many feature layers from VGG19 or VGG16. Based on common practices [6] [2], $conv4\_2$ is used to in describing the semantic structure of the image. Next, $conv3\_2$ and $conv4\_2$ are used to describe the texture of the image as highlighted below:-

$$\mathbf{L}_{mrf} = \mathbf{L}_{\mathbf{M}}(conv4\_2) + \sum_{\mathbf{t}=3}^{4} \mathbf{L}_{\mathbf{M}}(conv\mathbf{t}\_2) \tag{4}$$

Model Objective: From the above equations 2, 3 and 4: the final model objective of the net can be defined as the:-

$$\mathbf{L} = \mathbf{L}_c + \lambda_{mrf}\mathbf{L}_{mrf} + \lambda_{adv}\mathbf{L}_{adv} \tag{5}$$

where, λ_{adv} and λ_{mrf} are constants used to balance adversarial training and local structure regularization.

4 Our Contribution

4.1 Modified Network Architecture

The original reference architecture from the paper has been adopted with some changes. Two types of attention mechanisms have been tested. One is using Squeeze-and-Excitation [4] mechanism in 4 and the other is with A^2 double attention networks [1] in 3. Both of the methods yield relatively same results.

4.2 Image Quality Improvements - Attention Layer

When both of the architectures in 3 and 4 are compared, just using Squeeze-and-Excitation block as in Fig 4 slightly outperforms the A^2 double attention networks. The reason for including the attention mechanism originates from [15] where the authors have noted that attention mechanism helps in generating better inpainting results. Finally the table 1 shows improved Peak Signal to Noise Ratio (PSNR) numbers in architectures where the attention mechanism has been added.

4.2.1 Squeeze and Excitation Attention

Squeeze and Excitation Attention is a creative attention mechanism where first for each layer(original tensor) a global pooling is taken. This outputs a tensor whose shape is equal to number of layers. This tensor is then excited by an activation function. Activation function can be chosen from a range of activation functions like sigmoid, tanh etc. Lets call this intermediate tensor. This output is then multiplied to original layer on which pooling was applied. The result is a hybrid output with same tensor shape as original tensor. As global pooling is done here the output tensor has a context of whole image. As the intermediate tensor is learned by the model, and the intermediate tensor is basically a weight multiplied to original tensor, less important layers have low weight and more important layers have more weight.

4.2.2 A^2 Double Attention

In this attention mechanism there are two blocks. One feature gathering and the other feature distribution. Feature gathering block is usually placed at the start of the context encoder. Feature Distribution is usually placed at the end of context decoder. In this way this mechanism ensures that important features are identified and disseminated.

4.3 Tuning and Comparison between architectures

Usually, in a context encoder, the middle layers (feature Space) are dense layers as mentioned in [10]. But this causes a lot of features to be lost in the feature space. One solution to this is to add multiple dilated convolution layers in place of dense layers to keep the feature space rich [16]. Hence there are many dilated convolution layers in the middle of all three parallel context encoder-decoders in our architectures.

The reason we think the architecture in 3 performed a bit poorly was because it started developing the checkerboard artifacts problem elucidated in [9]. The more the artifacts the lesser the PSNR would be. We ran both models in Figs 4 and 3 for 7k steps and architecture in 3 developed artifacts earlier than architecture in 4 which is responsible for decreasing the PSNR. The results of the PSNR are shown in the table 1.

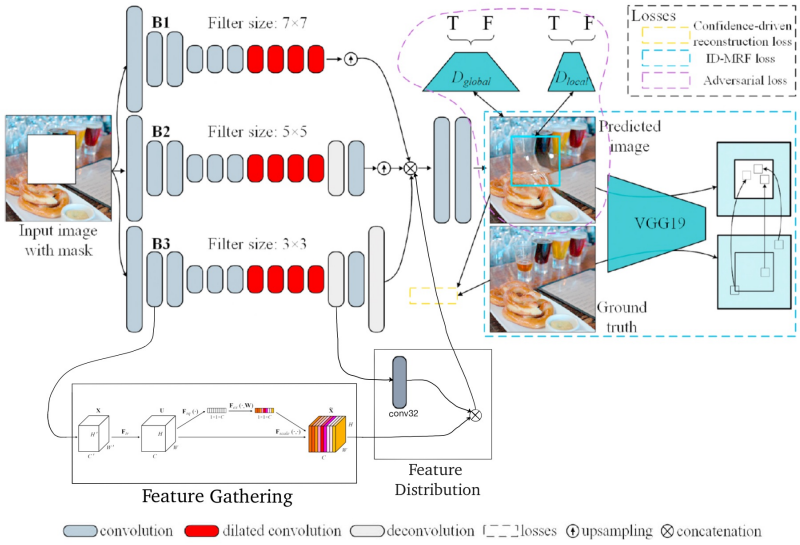

Figure 3: Overview of the Model and Network Layers with A2 double attention with squeeze and excite inside feature gathering block and feature distribution block. Main architecture taken from [12]

4.4 Conversion from TensorFlow v1 to TensorFlow v2

We have used the GitHub repository mentioned in section 6.4 as our base code. keeping the Keras implementation, we have replaced the TensorFlow v1 back end with the TensorFlow v2 back end. TensorFlow v1 code works in two steps: building the computational graph and create a session to execute it. It was difficult to create big models and debug them for errors. TensorFlow v2 provides the implementation of Eager Execution by default, so we don't need to create the sessions to run the computational graph. It allows us to run the code and view the results without the need for creating a session.

5 Implementation Details

Since the problem domain is around digital images, the underlying data processing and model execution becomes a computationally demanding task. We have used the best available GPUs along-with the most optimized libraries to get the training done in a reasonable time. The following sub-sections elaborate on the specifications in detail:

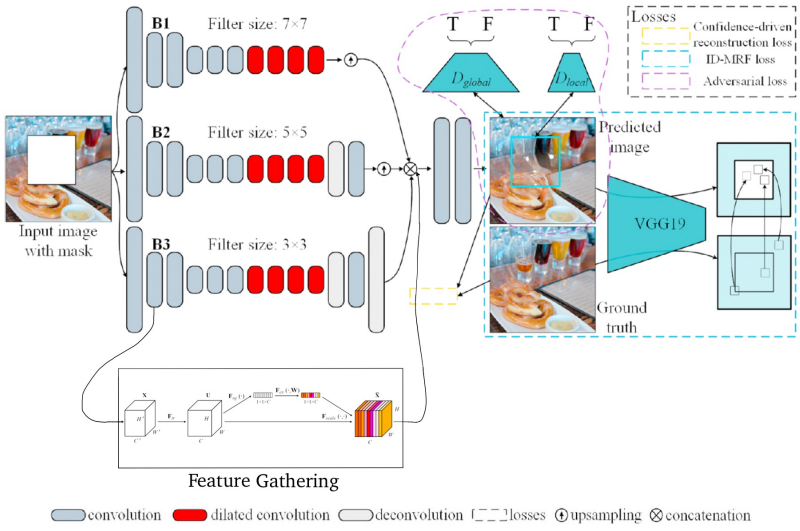

Figure 4: Overview of the Model and Network Layers with just A2 double attention with squeeze and excite inside feature gathering block. Main architecture taken from [12]

5.1 Dataset:

We chose the CelebHQ data set to train our model. The dataset comprises of 30K images of celebrates. Each image in the dataset was a 256*256 jpg images. The dataset can be downloaded from here.

For masks, Nvidia irregular holes dataset has been used which can be downloaded from here.

When an original image is combined with a mask that's when it generates a masked image that acts as input to the context encoder and decoder in the architecture. As we have both the masked image and ground truth image both of them are used to train the GAN.

5.2 Training and Testing

For every input image Y, we sampled a binary mask (with value 0 for known and 1 for unknown pixels) at random locations in the image. Next, the input image is produced by applying the binary mask to the original input image as, $X = Y \odot (1 - M)$. The concatenation of X and M is fed to the model G as input. All inputs and outputs are linearly scaled within the range [-1, 1]. The final prediction $\hat{Y}$ [12] is computed as follows:

$$\hat{Y} = Y \odot (1 - M) + G([X, M]) \odot M$$

To train the model we randomly chose 95% of the dataset. The total trainable data set comprised of 28.5k images. The rest 5% data which were the 1.5K samples were left for testing. The below section explains the system requirements.

5.2.1 Software and Libraries:

- Python 3 and standard libraries for image operations - openCV
- Author's code - TensorFlow 1
- Our Implementation - Keras 2.3 on TensorFlow 2.1 Backend
- Scikit Image for generating metrics like PSNR

5.2.2 GPU configuration:

- Nvidia Tesla K80 or Tesla V100 GPU based Virtual Machine on the Google Cloud Platform

6 Experiments and Results

6.1 Generated Images

Figure 5 shows the samples of images generated by the model. On the left is the input image to the model, image in the middle is the output image generated by the model, and the image on the right is the ground truth image.

Figure 5: Results of Architecture in Fig 3

6.2 Performance

- **Qualitative Evaluation:**

  We also tried to change the filter used in Spatial Variant Reconstruction Loss mentioned in equation 1 from gaussian to laplacian, by equating the location of laplacian to Gaussian's mean and scale of the laplacian to the gaussian's standard deviation so that both the probability density function's graphs are nearly same. The reason to make such change is that Laplacian provides a much sharper weighted mask compared to gaussian. This didn't make a substantial effect on the average PSNR.

  The results generated by our model are reasonable and realistically structured even if we mask some of the main features in the image. for example, if we mask out eyes, nose, and mouth from an image (since we are using only face images), our network is able to generate the eyes, nose, and mouth at the right spots with reasonably good quality. The main contributors to this reasonable structures generated by the

Method Name	*PSNR*
Original Architecture[12]	25.70
Original Architecture with only A^2 double attention(Fig 3)	27.3
Original Architecture with only Squeeze and Excite attention (Fig 4)	27.78

Table 1: PSNR Results of different architecture

network are multi-column architecture. The realistic textures are generated by ID-MRF regularization and adversarial training. Since we have used random sized and shaped masks, our network can generalize to a variety of inputs (face images).

- **Quantitative Evaluation:** We have taken PSNR (Peak Signal to Noise Ratio) as the main metric for calculating the quality of image generation. For calculating the PSNR we have taken 200 original images from CelebHQ dataset and 200 masks from the mask dataset randomly and kept it constant throughout our experimentation process to find if there is any improvement due to the changes we make to the architecture. Those datasets can be found here. This is the process suggested by main research paper [12] we are following. For PSNR calculations please refer to table 1. As you can see the model architectures with attention in them outperform the original model.

6.3 Challenges

We encountered blockers and limitations on almost all the levels of working on this project. We would like to mention that it was challenging in the right amount. With most of the time spent on figuring out technical nits and bits, the major challenges are as follows:-

- **Upgrading to TensorFlow v2:** We had significant issues with using the output of intermediate VGG layers to compute the local and global losses. To be specific, We spent almost a week on fixing the ID-MRF loss in VGG layer. This was caused by the fact the TensorFlow v1 implementation took intermediate VGG layer outputs and stacks them to the final output. This was something that was not possible in TensorFlow v2 as the VGG model was directly coming from the Keras library.
- **Running the model on GPU:** We were using our virtual machines on google cloud to train the model. It is very important to set the correct CUDA and CUDNN library version for the model to run on the GPU. The older version of CUDA and CUDNN libraries are not compatible with the TensorFlow v2.
- **Limited Computational Resources:** The initial GCP credits were of great help and enabled us to complete the initial implementation in time. We had to use our GPU to extend the training sessions towards the end of the term. If we had access to tremendous amounts of resources, we would have loved to execute the model on a bigger dataset with a wider range of images.

6.4 Code Repository

Our model code is built on top of the Keras implementation (TensorFlow v1) of [12] that can be found at

https://github.com/tlatkowski/inpainting-gmcnn-keras

Our project code repository in TensorFlow v2 forked from the above URL can be found at

https://github.com/shreshtha-pankaj/inpainting-gmcnn-keras/tree/gpu_working_code.

The weights required to run the model can be found in the text file here

https://github.com/shreshtha-pankaj/inpainting-gmcnn-keras/blob/gpu_working_code/outputs/weights.info

We have generated an ipynb file for quick testing and calculating the Average PSNR which can be found at

https://github.com/shreshtha-pankaj/inpainting-gmcnn-keras/blob/gpu_working_code/Image-Inpainting-Demo.ipynb

NOTE: Training and Testing steps are included as appendix at the end of the paper.

7 Future Work

We would be highly interested in extending this project and the neural architecture to videos. The amount of complexity that comes with changing the subject to videos is several times more when compared to images. This move exponentially increases the requirement of data, computation and validation challenges. We also faced a

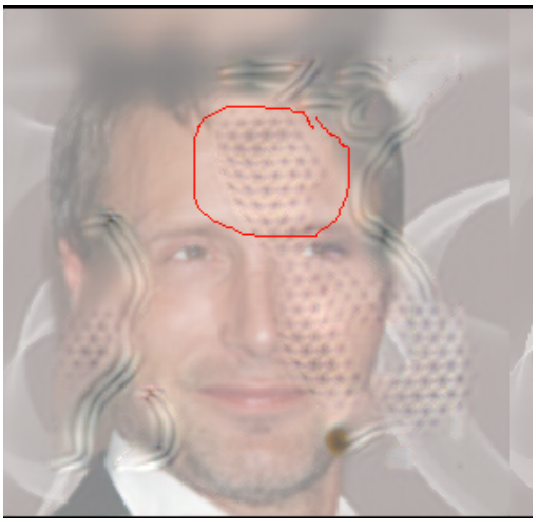

Figure 6: The part circled in red are the checkerboard artifacts

problem of checkerboard artifacts [9] when training a model for more than a day. We wanted to get a more realistic image but in turn, the image developed these artifacts as shown in Fig 6 We would be improving based on the suggestions in [9].

8 Conclusion

Our implementation of a complex deep learning model, coupled with additional optimizations has been able to tackle the challenges of image inpainting. We have managed to achieve significant improvements over each iteration of change, that was made to the initial model that we started off with. Our model that was also migrated to Tensorflow v2, delivers good quality results that are comparable to the results presented in high-quality research articles. We were able to obtain a PSNR value of 27.78 in our generated images, which is marginally better than the value obtained by [12].

Acknowledgments

We would like to thank the instructors of the course, **Adam Bloniarz** and **Shumin Wu** for the great learning experience, and for pointing us in the right direction towards successful the project completion. We highly appreciate the Google Cloud Platform's help for providing us credits worth of USD 200 that were used towards training the model.

References

[1] Yunpeng Chen, Yannis Kalantidis, Jianshu Li, Shuicheng Yan, and Jiashi Feng. Aˆ2-nets: Double attention networks, 2018.

[2] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[3] James Hays and Alexei A Efros. Scene completion using millions of photographs. *ACM Transactions on Graphics (SIGGRAPH 2007)*, 26(3), 2007.

[4] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. Squeeze-and-excitation networks, 2017.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[6] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild, 2014.

[7] Kamyar Nazeri, Eric Ng, Tony Joseph, Faisal Z. Qureshi, and Mehran Ebrahimi. Edgeconnect: Generative image inpainting with adversarial edge learning, 2019.

[8] Richard Newman. Conservation and care of museum collections, 2011.

[9] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts, Oct 2016.

[10] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A. Efros. Context encoders: Feature learning by inpainting, 2016.

[11] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015.

[12] Yi Wang, Xin Tao, Xiaojuan Qi, Xiaoyong Shen, and Jiaya Jia. Image inpainting via generative multi-column convolutional neural networks. *CoRR*, abs/1810.08771, 2018.

[13] Wei Xiong, Jiahui Yu, Zhe Lin, Jimei Yang, Xin Lu, Connelly Barnes, and Jiebo Luo. Foreground-aware image inpainting, 2019.

[14] Zhaoyi Yan, Xiaoming Li, Mu Li, Wangmeng Zuo, and Shiguang Shan. Shift-net: Image inpainting via deep feature rearrangement, 2018.

[15] Jie Yang, Zhiquan Qi, and Yong Shi. Learning to incorporate structure knowledge for image inpainting, 2020.

[16] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions, 2015.

Appendix

Training steps

- Clone the repository using
  `git clone -b gpu_working_code https://github.com/shreshtha-pankaj/inpainting-gmcnn-keras.git`
- Download the training data from here and extract the zip in your preferred directory.
- Create an environment variable `$inpainting` and set its value to the path of the downloaded dataset.
- Install the dependencies from the `requirements.txt` using `pip` or `conda`.
- Run the warmup training using this command
  ```
  python runner.py -train_path "$inpainting/train" -mask_path "$inpainting/mask"
  -experiment_name dl_project -warm_up_generator
  ```
- Run the final train with this command
  ```
  python runner.py -train_path "$inpainting/train" -mask_path "$inpainting/mask"
  -experiment_name dl_project -from_weights
  ```
- Since the source is forked from the original author, detailed instructions of running the code can be found here [12]

Testing steps

- Clone the repository using
  `git clone -b gpu_working_code https://github.com/shreshtha-pankaj/inpainting-gmcnn-keras.git`
- Download the weights from the GDrive link and paste the whole folder named **dl_project** into the **outputs** folder inside the cloned project directory
- We have generated an ipynb file for quick testing and calculating the Average PSNR which can be found at **https://github.com/shreshtha-pankaj/inpainting-gmcnn-keras/blob/gpu_working_code/Image-Inpainting-Demo.ipynb**