CNN BASED IMAGE INPAINTING

Aakansha Bhatt, Anika Tabassum, Sadia Mahtaban

University of California, Davis

ABSTRACT

Image inpainting is the process of filling in the missing areas of an incomplete image. In this project we implement a conventional neural network (CNN) based autoencoder-decoder image inpainting technique using structural similarity (SSIM) as metrics for training on CIFAR-10 dataset in Python using Keras and Tensorflow libraries. The region to be inpainted is created by the standard image processing idea of masking. We implemented random line and square masks and trained and tested our model for both. Our architecture performs better for the former. Apart from visual inspection, we used the peak signal-to-noise ratio (PSNR) and SSIM on test images to evaluate the effectiveness of our algorithm. Sample PSNR and SSIM values are in the 30dB range (typical for images) and close to +1 respectively, indicating the efficacy of our design. The codes are available at https://bit.ly/3goLH5j.

Index Terms— Image inpainting, deep learning, CNN, data generation

1. INTRODUCTION

Image inpainting is a task that aims to complete the missing regions of an image with visually realistic and semantically consistent content. Image inpainting can also be extended to videos. Although image inpainting has practical applications in unwanted object removal from an image, dis-occlusion, error concealment, and image editing, the task is still regarded as challenging. The result needs to have global semantic structure and fine detailed texture. CNNs have been suggested to predict the missing patches by propagating the surrounding convolutional features through a fully connected layer. Among the available challenges we had to choose from, the image inpainting challenge was our first choice because we were intrigued by the vast prospects of this technique - from taking the perfect Instagram photo to restoring historical images.

In this project we employ a deep CNN based autoencoder-decoder model on CIFAR-10 dataset for image inpainting. The model adopts a U-Net like architecture by adding multiple convolution layers and concatenating them [1]. We employ Adam optimizer, mean square error loss and SSIM metric for training. We implemented the code in Python using Keras and Tensorflow libraries in the Google Colab platform. Our masking was created by producing random thick lines or square blocks on the image to be used as training data. The model was tested using a testing data set that was not used in training. As a quantitative measurement, PSNR and SSIM metrics were used to verify the accuracy.

2. RELATED WORK

Classical inpainting methods, usually designed for single image inpainting are often based on local or non-local information available in the input image such as total variation (TV) [2] with prior knowledge, such as statistics of patch offsets, planarity or low rank (LR) to recover the image. Generally, traditional learning-free image inpainting methods can be roughly divided into two categories: diffusion-based [3] and patch-based methods [4]. These methods utilize only the internal information that achieves color consistency but fails in filling in semantic-aware contents. The recent development of deep learning has greatly improved the performance of image inpainting, especially in image categories like faces and complex natural scenes. The deep learning based inpainting approaches can learn semantic-aware models by training on large-scale datasets typically employing the generative adversarial networks. One of the earliest demonstrations of image inpainting with deep neural networks was by Xie *et al*. [5] in 2012. Pathak *et al*. [6] first proposed context encoders that utilized an encoder-decoder network to extract features and reconstruct the outputs. Yi *et al*. [7] proposed

the contextual residual aggregation module, and Zeng *et al*. [8] adopted a guided up-sampling network for high-resolution image inpainting. Based on deep CNNs, image inpainting methods typically utilize an encoder-decoder network to generate meaningful image content for hole filling. Most recent methods first predict coarse structure such as edges [9], structural information [10], segmentation maps [11] blurry images [7] or foreground contours [13], and then use these intermediate outputs as guidance for filling in details. Image inpainting methods have been applied in content removal [14, 4] and image restoration [11,12]. Researchers have been working to extend the techniques for image inpainting to fill spatio-temporal holes with plausible content in a video (video inpainting). In recent years, successful video inpainting employing deep neural networks have been reported in literature [15, 16]

3. DATA SET

We used the **CIFAR-10** dataset containing 50,000 train images and 10,000 test images. The images are from 10 classes shown in Figure 1 and all images have size 32x32x3. We chose a simple data set due to lack of computational power and time required for inpainting high resolution images.

Figure 1: CIFAR 10 dataset

Loading the dataset can be time and memory consuming. To combat this issue, a helper function GenerateMask() was used to generate the samples into multiple batches to feed them easily into our deep learning model. That is, data is generated in parallel by the CPU and then fed into the GPU. This means that data is generated at the start of each epoch rather than being generated all at once. This reduces memory loading and makes the process faster.

In the helper function, we further prepare the data by the standard image processing idea of masking an image to add artificial deterioration. Two **masks** are defined: square and random lines. See Figure 2 below. To prevent overfitting to masks, we randomized the size and position of the square for the square mask and length and thickness of the lines for the irregular line mask.

(a) (b)

Figure 2: Mask with (a) random sized square, (b) random lines

Then to train the model in a self-supervised learning environment, the same number of X and y pairs are required. Here X will be batches of masked images, and y will be the original image. This is the essential part of the helper function acting as a Keras data generator for creating random batches of X and y pairs of desired batch size. We then apply a random mask to X in the same function. We define this method as __data_generation().

4. APPROACH

Our model for image inpainting is CNN based Autoencoder-Decoder illustrated in Figure 3. Autoencoders are a type of unsupervised neural network that accepts an input set of data, internally compresses the input data into a latent-space representation and reconstructs the input data from this latent representation. Mathematically, it is comprised of an encoder which describes the input, h = f(x), and a decoder that produces the reconstruction, r = g(h) or r = g(f(x)).

For the encoder there are four layers each performing two convolutions followed by max pooling of the size 2x2. A smaller kernel size (3x3) was chosen to differentiate between and extract the small and local features. The filter size starts from 32 in the first layer and ends with 256. Each convolution layer uses ‘ReLU’ activation.

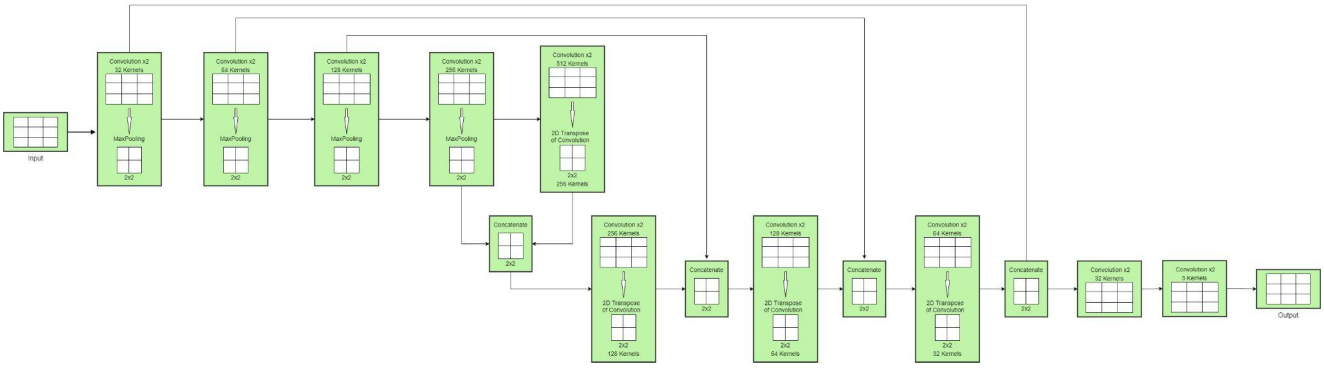

Figure 3: Model architecture

The decoder has four layers, each performing two convolutions followed by transposed convolution and concatenation. The (3x3) convolution filter size starts from 512 in the first layer and ends with 64. Each convolution layer uses 'ReLU' activation. The 2x2 filter is used for deconvolution. The filter sizes start from 256 in the first transposed convolution layer and end with 32. Here as well, the 'ReLU' activation is used. The concatenation is done for the output of the transposed convolution vectors and convolution layers from the encoder of the same sizes. Due to the influence of the concatenation operation, the size of the input and output remain the same. An additional layer of convolution is performed to get the output image with three kernels, similar to our input image.

The input and output images from our model have to be of the same size. This is because the inpainted image should be similar to the original image after removing the masking from the images. The number of layers is a hyperparameter determined by experimentation and optimized for our given task. There are 19 convolution layers in our model because it allows for extraction of as many features from the dataset as possible.

The Adam optimizer and mean squared error loss function were used to train our model. Each epoch has 1562 training images from the data generation function. After optimizing the number of epochs by noting the decrease in loss value for each epoch, and a plateauing of our metric data, it was noted that 10 epochs produce best outputs while keeping the runtime short. This has been illustrated in Figure 4.

```
Epoch 1/10
1562/1562 [==============================] - 44s 27ms/step - loss: 0.0039 - ssim: 0.8886
Epoch 2/10
1562/1562 [==============================] - 41s 26ms/step - loss: 0.0017 - ssim: 0.9358
Epoch 3/10
1562/1562 [==============================] - 42s 27ms/step - loss: 0.0015 - ssim: 0.9413
Epoch 4/10
1562/1562 [==============================] - 43s 27ms/step - loss: 0.0014 - ssim: 0.9447
Epoch 5/10
1562/1562 [==============================] - 44s 28ms/step - loss: 0.0013 - ssim: 0.9472
Epoch 6/10
1562/1562 [==============================] - 44s 28ms/step - loss: 0.0013 - ssim: 0.9486
Epoch 7/10
1562/1562 [==============================] - 43s 28ms/step - loss: 0.0012 - ssim: 0.9499
Epoch 8/10
1562/1562 [==============================] - 43s 28ms/step - loss: 0.0012 - ssim: 0.9511
Epoch 9/10
1562/1562 [==============================] - 43s 28ms/step - loss: 0.0011 - ssim: 0.9519
Epoch 10/10
1562/1562 [==============================] - 44s 28ms/step - loss: 0.0011 - ssim: 0.9526
```

Figure 4: Training: loss decreases with number of epochs while SSIM increases indicating model is learning

We use the SSIM index for training. The SSIM defines a structural similarity measure between two images, calculating the similarities between luminance, contrast, and structure. This metric is derived from the human visual system model. The SSIM value ranges from -1 to +1, with -1 indicating very different images and +1 indicating very similar or the same image. The SSIM index is calculated on various windows of an image. The measure between two windows x and y of common size $N \times N$ is

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

where μ_i denotes average of i, σ_i^2 denotes average of i where $i = x, y$, σ_{xy} is the covariance of x and y, $c_1 = (k_1 L)^2$, $c_2 = (k_2 L)^2$ two variables to stabilize the division with weak denominator. L is the dynamic range, $k_1 = .01$ and $k_2 = .03$ by default. We use the SSIM function built-in Tensorflow for our task.

Another metric used for quantitative measurement for testing is PSNR with the SSIM. PSNR, which is calculated from the mean squared error, is a metric that quantifies the difference in values of each of the corresponding pixels between the two images. A larger PSNR value denotes a higher similarity of the two images. PSNR is defined as follows

$$PSNR = 10log_{10}(\frac{(255)^2}{MSE})$$

where 255 is the max value and MSE is the mean squared error.

5. RESULTS

Figure 5 shows the input images, masked images and inpainted results of our model for **random line masks**. Visually, we see the inpainting algorithm performs well enough for the toy datasets but there is still some blurriness with some lack of texture. For smaller masks, the inpainted image is a good representation of the ground truth.

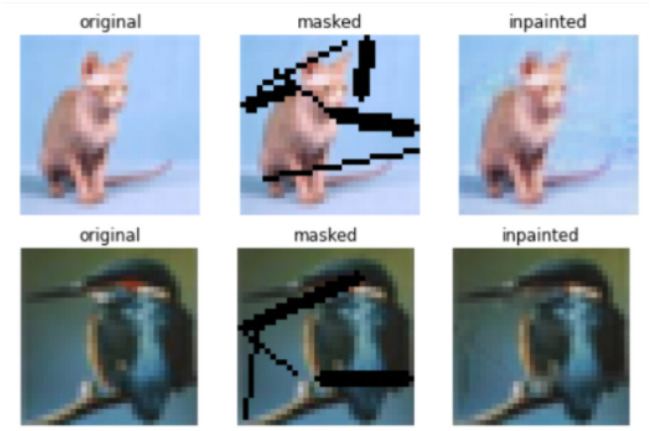

Figure 5: Inpainted image results for random lines mask

For objective discussion and analysis of the inpainting effect, we used the PSNR and SSIM to evaluate the effectiveness of our algorithm. We present sample PSNR and SSIM values for test images in Table 1. As we can see PSNR values are in the 30dB range which is typical for images reported in literature. Moreover, SSIM values are very close to +1, which indicates that the inpainted images are very similar to the ground truth. This in turn indicates that our model works fairly well.

Table 1. Sample PSNR and SSIM for test images

Sample PSNR values for test images						
38.27	40.03	29.01	33.14	32.69	32.22	34.44
Sample SSIM values for test images						
0.917	0.946	0.976	0.970	0.977	0.985	0.971

We also trained and tested our model for square mask images and the results are shown in Figure 6. We see that our model shows better performance for random line masks than for square masks. As we implemented a simpler model, the large square artifacts are harder to ameliorate. This can be improved by employing more complex neural networks such as GANs and partial convolution methods.

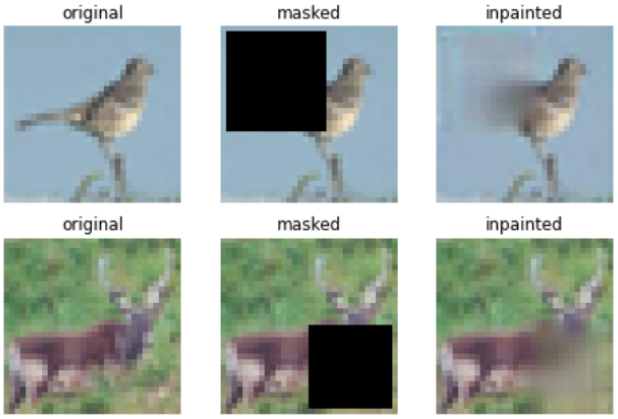

Figure 6: Inpainted image results for square mask

10. CONCLUSION

We built a relatively efficient CNN-based image inpainter using encoder and decoder. With the data generator, we decreased our algorithm training and running time. We applied different masks to study the robustness of our model. The inpainted results were visually pleasing for the low resolution images in our dataset. Also, our PSNR and SSIM values prove that our model works with a high degree of accuracy. Overall, we are satisfied with the results, but there is room for improvement.

Our limiting factor was that we could only calculate and present the PSNR and SSIM values for some of the images. Hence, we were unable to calculate the averages of the two for our dataset. In the future, we plan on automating the process to calculate the average for the entire dataset. This would provide us with an accurate approximation of our model functionality. Another challenge we faced is that our model could not precisely inpaint the images with large square masks. We plan on trying to better the results by using generative adversarial networks (GANs) with our implemented model.

In the future, we would also like to implement a differentiating function that can detect the objects to be removed. This is where an attention map would be obtained for a given input with back-propagation on a CNN. This particular method has been used to de-rain and de-haze images and it can be beneficial in making our inpainting model more robust.

11. CONTRIBUTIONS

Aakansha contributed to performing data generation, creating the CNN model and masking. Sadia and Anika performed all the operations required for generating PSNR and SSIM coefficients. Testing, data collection and analysis was conducted collectively by all authors. All authors wrote, read and approved the final report collectively.

12. REFERENCES

[1] Thakur, Ayush and Paul, Sayak. [https://github.com/ayulockin/deepimageinpainting]

[2] Afonso, Manya V., José M. Bioucas-Dias, and Mário AT Figueiredo. "An augmented Lagrangian approach to the constrained optimization formulation of imaging inverse problems." *IEEE Transactions on Image Processing* 20.3 (2010): 681-695.

[3] Ashikhmin, Michael. "Synthesizing natural textures." *Proceedings of the 2001 symposium on Interactive 3D graphics*. 2001.

[4] Barnes, Connelly, et al. "PatchMatch: A randomized correspondence algorithm for structural image editing." *ACM Trans. Graph.* 28.3 (2009): 24.

[5] Xie, Junyuan, Linli Xu, and Enhong Chen. "Image denoising and inpainting with deep neural networks." *Advances in neural information processing systems* 25 (2012): 341-349.

[6] Pathak, Deepak, et al. "Context encoders: Feature learning by inpainting." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

[7] Yi, Zili, et al. "Contextual residual aggregation for ultra high-resolution image inpainting." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020.

[8] Zeng, Yu, et al. "High-resolution image inpainting with iterative confidence feedback and guided upsampling." *European Conference on Computer Vision*. Springer, Cham, 2020.

[9] Nazeri, Kamyar, et al. "Edgeconnect: Generative image inpainting with adversarial edge learning." *arXiv preprint arXiv:1901.00212* (2019).

[10] Ren, Yurui, et al. "Structureflow: Image inpainting via structure-aware appearance flow." *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019.

[11] Song, Yuhang, et al. "Spg-net: Segmentation prediction and guidance network for image inpainting." *arXiv preprint arXiv:1805.03356* (2018).

[12] Wan, Ziyu, et al. "Bringing old photos back to life." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020.

[13] Wei Xiong, Jiahui Yu, Zhe Lin, Jimei Yang, Xin Lu, Connelly Barnes, and Jiebo Luo. Foreground-aware image inpainting. In Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR), 2019

[14] Shetty, Rakshith, Mario Fritz, and Bernt Schiele. "Adversarial scene editing: Automatic object removal from weak supervision." *arXiv preprint arXiv:1806.01911* (2018).

[15] Kim, Dahun, et al. "Deep video inpainting." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019.

[16] Chang, Ya-Liang, et al. "Learnable gated temporal shift module for deep video inpainting." *arXiv preprint arXiv:1907.01131* (2019).