Image Inpainting with Convolutional Neural Networks

Andrea Maggiordomo

May 28, 2021

1 Introduction

This document is my final project report for the PhD course "Deep Learning for Signal and Image Processing". For my project, I have chosen to implement, train and test two convolutional neural network models for image inpainting, which is the task of synthesizing missing image detail from an input image according to a given binary mask. This problem is a natural candidate for deep learning methods, and several solutions have been proposed in recent years. For the practical task, I have chosen to implement and test two models proposed in the literature:

1. "Context Encoders: Feature Learning by Inpainting" by Pathak et al. [6], a GAN-inspired model that employs adversarial training taken from the paper;
2. "Image Inpainting for Irregular Holes Using Partial Convolutions" by Liu et al. [4], an encoder-decoder architecture with custom "partial-convolution" layers that use the input mask to implement a re-weighting scheme and discard the masked pixel when computing the output values.

The goal of this document is to briefly outline the relevant topics presented during the course, and provide some details on the problem considered and the chosen architectures. In Section 2, background information on machine learning, artificial and deep neural networks, encoder-decoder architectures and generative adversarial networks is briefly discussed. The rest of the document focuses instead on CNN-based methods for the image inpainting problem: Section 3 introduces the problem some relevant literature; Sections 3.1 and 3.2 describe more in depth the models that have been implemented and trained; finally, Section 4 provides some information on the data and configurations used for training, as well as some inpainting results produced from both models.

2 Background

Machine learning is a computational framework where a particular function is learned in a data-driven fashion, progressively refining its accuracy as the number of data observations increase. The learned function is a typically non-convex differentiable parametric function (a parametric model), and the learning process consists in the numerical optimization of the model parameters to obtain an approximation of the desired ideal function.

At a high level, there are three main learning paradigms. In *supervised learning*, the parameters are learned from a sequence of (input, expected output) pairs by minimizing a loss function that measures the deviation of the transformed input from the expected output. The main drawback of this paradigm is that it requires pre-computed outputs that in many cases require manual interventions. *Reinforcement learning* techniques are typically used to train autonomous agents and do not rely on pre-computed outputs, instead using a feedback mechanism to evaluate the quality of the model behavior. In *unsupervised learning*, the goal of the model is to learn a classification of the input without any additional information, for example by clustering the input data after it is projected to a learned feature space.

The rest of this report only deals with supervised learning techniques.

2.1 Artificial Neural Networks

Artificial Neural Networks are parametric models in which the computation is described by a directed acyclic graph (this is not true for the case of recurrent neural networks, which are not discussed in this report). Each node of the graph is an artificial neuron, with a set of input and output connections from and towards other neurons in the network. The function computed by each neuron is an affine transformation of its inputs followed by a non-linear *activation* function, whose value is then forwarded to the output neurons. The weights and bias terms of the affine combinations computed at each neuron are the model parameter, randomly initialized and adjusted during the training process.

Let f be parametric model described by an artificial neural network, $\{x_i, y_i\}_i$ a sequence of (input, expected output) pairs, θ the model parameters and $\mathcal{L}_i$ an error function measuring the distance between two output values (the loss function). During the *forward pass*, the model computes a sequence of predictions $\{f(x_i, \theta)\}_i$. From these predictions, we can compute the gradient of the loss function w.r.t. the model parameters $\nabla_\theta \mathcal{L}$:

$$\nabla_\theta \mathcal{L} = \frac{1}{n} \sum_i \nabla_\theta \mathcal{L}_i \left(f(x_i, \theta), y_i \right) \tag{1}$$

(here a global loss is taken as the average of each sample loss); this step is referred to as the *backward pass* because the gradient of the model parameters is computed from a backward walk of the computation graph using the chain rule.

Then, we can use the computed gradient to update the weights of the model parameters in order to decrease the value of the loss function:

$$\theta' = \theta - \lambda \nabla_\theta \mathcal{L} \tag{2}$$

where λ is a small step size, as the negated gradient is only guaranteed to be a *local* descent direction. In machine learning the step size is usually referred to as the "learning rate". Since supervised learning generally requires large amounts of data to be effective, the model weights are generally updated from small subsets (*batches*) of the entire data sequence. This strategy is referred to as stochastic gradient descent.

Overfitting occurs when the learned parameters perform well on the data used for training but not on arbitrary data. This can be verified during training by monitoring the model performance on a separate (typically smaller) set of validation data: if the loss function computed on the validation data is increasing, the model is losing generalization ability and overfitting to the training data. This issue can be caused either by an inadequate model architecture that is difficult to train to achieve the desired output, or a distribution of the training data that is not representative of the actual data that the model will be applied to.

2.2 Deep and Convolutional Neural Networks

Deep artificial neural networks are organized in a hierarchy of *layers*, i.e. nodes that are at the same level in the computation graphs. The advantage of a deep network is that its architecture is able to learn an effective *representation* for the task at hand, i.e. the model learns the desired function by also learning a transformation of the input data to a suitable feature space.

With high-dimensional data, such as images, *convolutional neural networks* (CNNs) have proved to be highly effective. In a CNN, the feature extraction is performed by convolution layers, where the learnable parameters are the convolution filter weights; convolution layers are typically followed by non-linear activations, similarly to traditional artificial neural networks. A convolution layer performs a number of convolutions equal to the output channels of the layer, each computed over the entire set of input feature channels.

Convolutional networks can be used to project high-dimensional inputs to a low-dimensional feature space, or vice-versa. Autoencoders are encoder-decoder CNN architectures where the encoder module learns to project the input to a low-dimensional *latent* space, and the decoder module learns to

decode the latent feature vector to reproduce the input. Autoencoders can be used to learn compact representations and have a number of uses, e.g. data compression.

Generative models are trained to generate synthetic data by decoding latent space vectors. This is achieved by learning a decoding function that maps random latent vectors to output data that follows a desired distribution. Typically, generative models are trained using adversarial training with a generator network and a discriminator network jointly trained to compete with each other. The discriminator network is a binary classifier that learns to discern synthetic images from genuine ones, trying to maximize accuracy; the generator network is trained to trick the discriminator into classifying synthesized data as genuine. Generative models have shown remarkable results, especially in image synthesis, but in general it is difficult to condition the output to produce particular features. Training generative models can be challenging, as a number of "failure modes" can occur; as an example, if the discriminator becomes too effective at detecting synthesized images it will not provide sufficient information for the generator to improve, halting its progress.

3 CNN models for Image Inpainting

Image inpainting is an important computer vision problem with applications ranging from image restoration to object removal. Given an input image x and a binary mask M denoting with 1 the valid image pixels and with 0 the invalid or *missing* image pixels, the inpainting task consists in assigning colors to missing pixels in order to produce a plausible output image.

Deep learning models and CNNs in particular have been successfully applied in recent years to solve this and similar computer vision problems such as image super-resolution and image synthesis.

The Context Encoder [6] is a generative model designed to synthesize a missing patch from the surrounding context; although designed to work with input images of limited resolution, it is one of the first deep networks able to synthesize large missing regions. Iizuka et al. [2] extend the Context Encoder approach to arbitrarily sized inputs and employ both a local (patch-focused) and global discriminator to improve the inpainting quality. Liu et al. [4] propose to use partial convolution [5] layers to account for missing regions, and train their model with irregularly shaped masks. Recently, complex architectures with multiple modules have been proposed. Yu et al. [11] employ a coarse network to perform an initial rough reconstruction of the missing region followed by a refinement network whose output is then subject to both a local and global discriminator. They later evolve their architecture [12] adding gated convolutions and improving the refinement and discriminator networks. In a similar fashion, Zeng et al. [13] also propose a two-stage approach with a coarse inpainting and a refinement network; however, they exploit an iterative inpainting scheme where the refinement network also outputs a confidence map, which is used to fill only high-confidence pixels and iterate again over low-confidence regions. They also train their model on low-resolution images, exploiting an upsampling network to restore the full resolution of high-resolution inputs.

3.1 Context Encoders

The Context Encoder [6] is an encoder-decoder architecture that takes as input a *context*, i.e. an image missing a center square region, and synthesizes an image patch to replace the missing region. This model is an extension of DCGAN [8], from which it borrows the adversarial training scheme and the decoder and discriminator architecture. The model architecture (Figure 1) consists of a generator network F and a discriminator network D.

Input RGB images are 128×128 and normalized in the [-1,1] range, with missing pixels set to zero (medium intensity). The encoder projects the input image to a latent feature vector of dimension 4000, which is then decoded into a 64×64 image patch. The network is fully convolutional, with batch normalization [3] used both in the encoder and decoder module after each convolution or transposed convolution layer, followed by leaky ReLU activations in the encoder and ReLU activations in the decoder. The final output is ultimately mapped to the [-1, 1] range with a hyperbolic tangent activation layer.

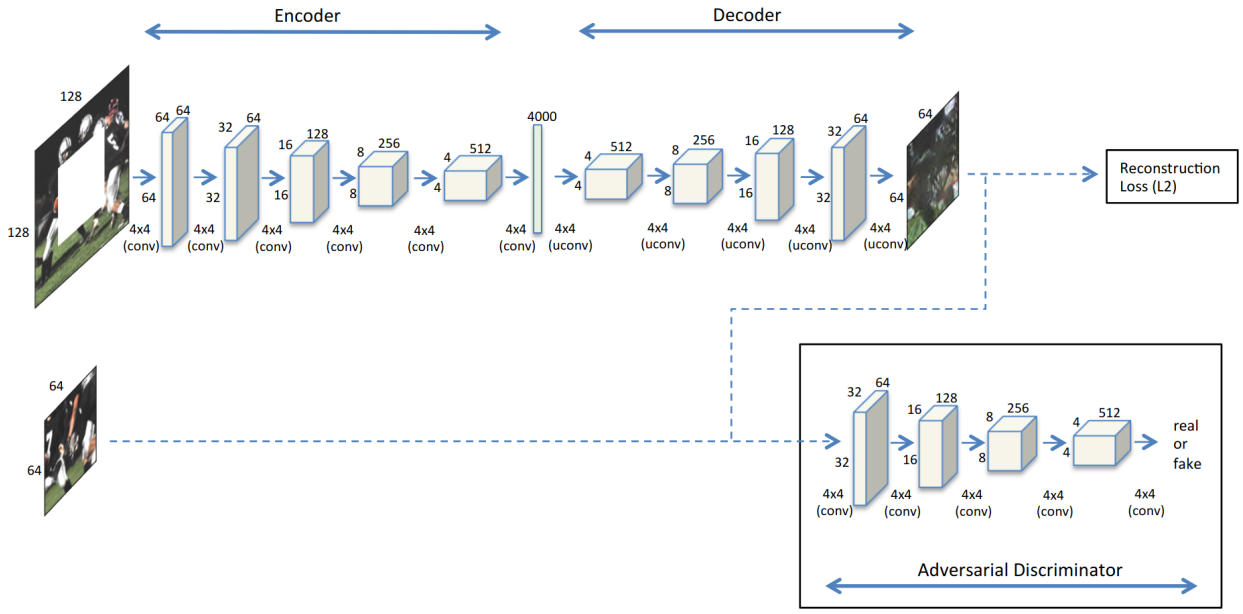


Figure 1: The Context Encoder training architecture. The generator network projects the masked input image to a 4000-dimensional latent vector, which is used to seed the generation of the missing image patch. The adversarial discriminator is jointly trained with the generator to classify original and synthesized image patches. Image taken from Pathak et al. [6].

The discriminator network is analogous to the encoder module, except the last convolution layer projects the image to a vector of dimension 1 and forwards it to a sigmoid activation layer to output a likelihood of the patch being real and *not* produced by the generator network.

3.1.1 Loss Function

The loss function used to train the context encoder combines a patch reconstruction loss $\mathcal{L}_{rec}$ and an adversarial loss term $\mathcal{L}_{adv}$.

The reconstruction loss is defined as the squared error of the reconstructed patch pixels relative to the ground truth image:

$$\mathcal{L}_{rec}(x) = \|(1 - M) \odot (x - F(M \odot x))\|_2^2. \tag{3}$$

According to the authors, training the context encoder with the reconstruction error alone does not allow the network to produce image patches with sharp detail, favoring rough outlines and blurry content. For this reason, they also employ an adversarial loss term

$$\mathcal{L}_{adv} = \max_D \mathbf{E}_{x \in \mathcal{X}}[\log(D(x)) + \log(1 - D(F(M \odot x)))]. \tag{4}$$

As is typical in this scenario, the generator network G and the discriminator network D are trained together using alternating optimization: at each iteration the discriminator is trained with a batch of real patches and a batch of synthesized patches. After updating the discriminator, new scores for the synthesized patches are computed and used to minimize the combined loss

$$\mathcal{L} = 0.999\mathcal{L}_{rec} + 0.001\mathcal{L}_{adv} \tag{5}$$

with respect to the generator parameters.

To ensure a smooth transition from the border of the predicted patch to the surrounding context, the authors suggest to use a larger weight for the reconstruction loss of pixels near the border of the reconstructed patch. This was achieved by applying a $10\times$ weight to the reconstruction loss of a 4-pixels wide border band.

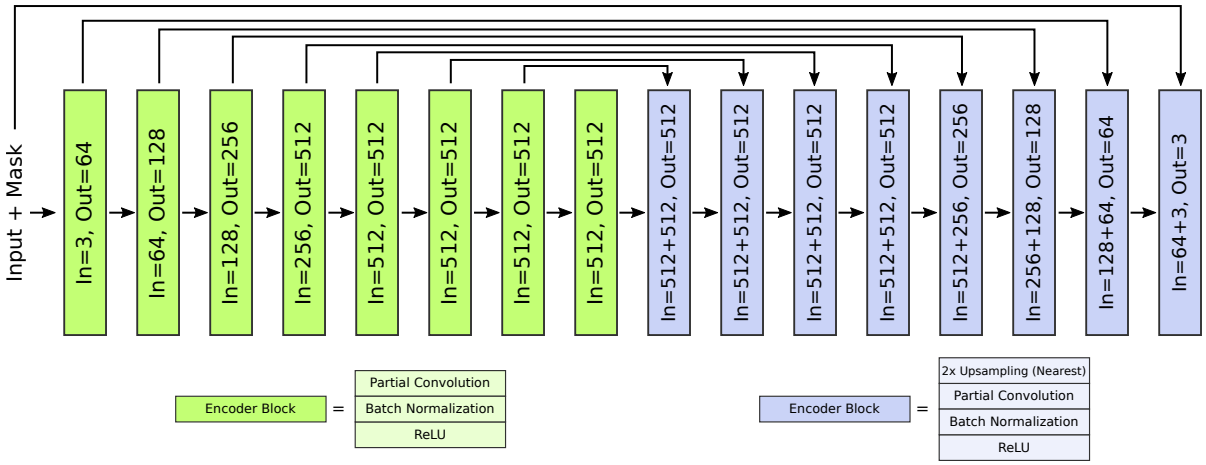


Figure 2: The U-Net architecture with the number of input and output features at each block.

3.2 U-Net with Partial Convolutions

A potential issue when using CNNs for image inpainting is that convolution layers in general are unable to discern between valid and masked pixels. Filling the image holes is a process that is learned during training and requires explicitly zeroing the missing pixels, but no additional semantic information is used by the network as a whole and the convolution layers in particular. Liu et al. [4] avoid this problem by adopting *partial convolution* layers [5].

Their model is based on U-Net [9], an encoder-decoder fully convolutional neural network with *skip connections*. A skip connection is used to bypass the hierarchical architecture of a convolutional network and connect the output of a layer not only to the input of the following layer, but also to the input of one or more other layers at deeper levels in the hierarchy. In this particular case, the network design is symmetrical: the model reconstructs a full resolution images, and skip links are used to connect the output of each encoder block to the input of the corresponding decoder block.

Each encoder block is composed of a (partial) convolution layer, a batch normalization layer and a ReLU layer. Decoder blocks perform a 2-factor nearest neighbor upsampling, concatenate the upsampled feature map with the output feature map of the corresponding encoder block, and perform a (partial) convolution followed by batch normalization and leaky ReLU activation. Input images are assumed to be normalized in the [0, 1] range, therefore the model uses a sigmoid function as final activation layer. The network architecture is outlined in Figure 2.

3.2.1 Partial Convolutions

The partial convolution layer performs two operations: it computes a reweighted convolution where each output value is scaled according to the number of valid pixels inside the corresponding convolution window, and produces an updated mask for the generated output feature map.

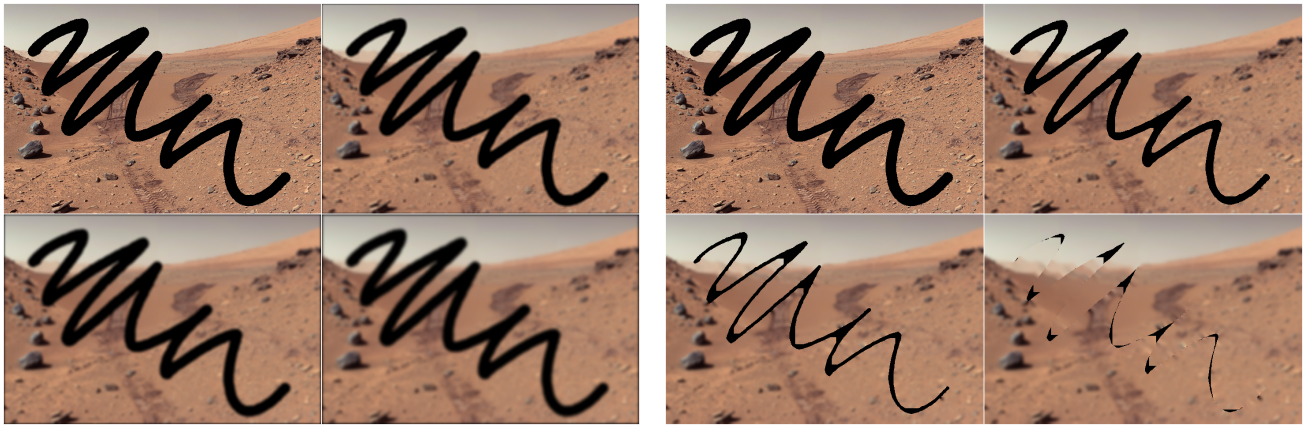
Let w_i be the convolution weights of a kernel window of size N, x_i the input feature values (i.e. the input window) at location x, and m_i the corresponding binary masks. The output x' of the partial convolution at x is defined as

$$x' = \begin{cases} \sum_i w_i m_i x_i \frac{N}{\sum_i m_i} + b & \text{if } \sum_i m_i > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

The updated mask value at the corresponding location m' is defined as

$$m' = \begin{cases} 1 & \text{if } \sum_i m_i > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

As partial convolutions are repeatedly applied, the input binary mask progressively shrinks until no invalid pixels are left. As an example, Figure 3 shows the output of progressive applications of a 5×5



(a) 5×5 Mean Filter – Standard Convolution

(b) 5×5 Mean Filter – Partial Convolution

Figure 3: Comparison between successive applications of a mean filter to an image with missing pixels, using standard and partial convolutions. The partial convolution propagates valid information inside the missing region, progressively shrinking the masked area; it also prevents bleeding of invalid signal into the image due to padding, whereas with standard convolutions the padding value (zero in this case) causes the border area to become darker as the number of applied convolutions increases.

mean filter using the partial convolution scheme: border pixels are progressively smoothed inside the masked region, and the missing region area decreases in size until it completely disappears.

3.2.2 Loss function

The loss function the authors employ to train this model combines five different terms to capture per-pixel, style and content reconstruction accuracy, as well as smoothness to avoid sharp transitions from the reconstructed missing region to the surrounding areas. Let x denote the ground truth image, $x_{in} = M \odot x$ the input region with cleared missing pixels, x_{out} the network output and $x_{comp} = M \odot x_{in} + (1 - M) \odot x_{out}$ the reconstructed image with valid pixels recovered from the input image.

The first two terms $\mathcal{L}_{hole} = \frac{1}{N_x}\|(1 - M) \odot (x_{out} - x)\|_1$ and $\mathcal{L}_{valid} = \frac{1}{N_x}\|M \odot (x_{out} - x)\|_1$ are the mean absolute reconstruction errors restricted to the masked and unmasked pixels respectively (with N_x denoting the number of pixels in the ground truth image).

The third term is the perceptual loss $\mathcal{L}_{perceptual}$ introduced in Gatys et al. [1]:

$$\mathcal{L}_p = \sum_l \frac{\|\Phi_l(x_{out}) - \Phi_l(x)\|_1}{N_{\Phi_l}} + \sum_l \frac{\|\Phi_l(x_{comp}) - \Phi_l(x)\|_1}{N_{\Phi_l}}, \tag{8}$$

where $\Phi_l(\cdot)$ is the rectified feature map produced by the l-th layer of a VGG-16 [10] network pre-trained on the ImageNet dataset. The feature maps are extracted from the first three pooling layers. This loss is used to reproduce the content of the image independently from the particular style or texture. Intuitively, feature maps extracted from a neural network trained for image recognition become progressively more representative of the image *content* and independent from the image texture or style as the depth increases. Penalizing mismatches in the feature maps of the reconstructed and ground truth image guides the model towards restoring the correct image content in the missing regions.

The fourth and fifth terms model the style loss, again introduced in [1]:

$$\mathcal{L}_{style_{out}} = \sum_l \frac{1}{C_l C_l} \left\| K_l(\Phi_l(x_{out})^T \Phi_l(x_{out}) - \Phi_l(x)^T \Phi_l(x)) \right\|_1, \tag{9}$$

$$\mathcal{L}_{style_{comp}} = \sum_l \frac{1}{C_l C_l} \left\| K_l(\Phi_l(x_{comp})^T \Phi_l(x_{comp}) - \Phi_l(x)^T \Phi_l(x)) \right\|_1, \tag{10}$$

where $K_l = C_l H_l W_l$ is a normalization factor applied to the gram matrix of the rectified feature maps $\Phi_l^T \Phi_l$, which captures their correlation. This term is motivated by the idea that what we perceive as

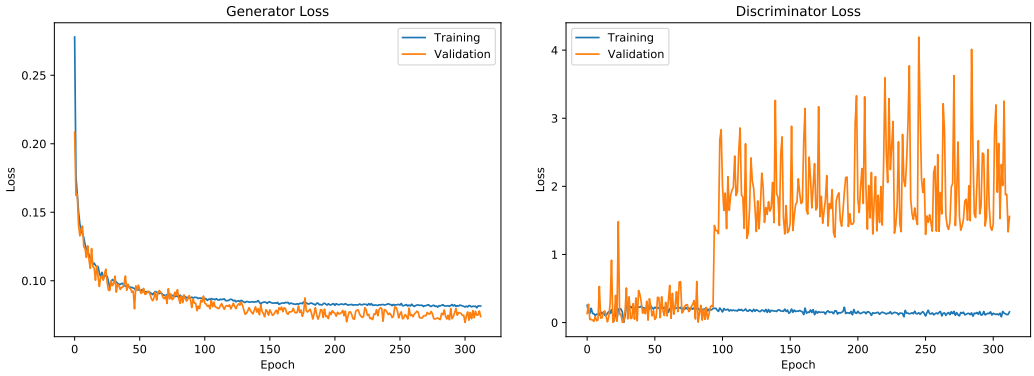


Figure 4: Generator and Discriminator loss evolution throughout the Context Encoder training process.

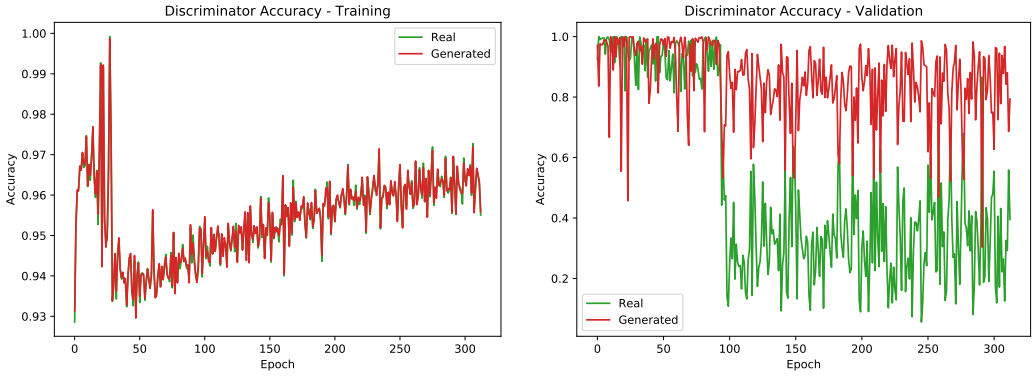


Figure 5: Discriminator classification accuracy evolution during training.

texture and style in an image can be captured with spatial statistics of the image features [7], and in particular is modeled accurately by just considering the correlation of deep feature maps.

The final loss term is the total variation loss $\mathcal{L}_{tv}$ restricted to the 1-pixel dilation of the missing region M_+, and is used to ensure smooth transition from the reconstructed missing region to the surrounding valid pixels:

$$\mathcal{L}_{tv} = \frac{\|\nabla((1 - M_+) \odot x_{comp})\|_1}{N_{x_{comp}}} \tag{11}$$

The composite loss the authors propose is then

$$\mathcal{L} = \mathcal{L}_{valid} + 6\mathcal{L}_{hole} + 0.05\mathcal{L}_{perceptual} + 120(\mathcal{L}_{style_{out}} + \mathcal{L}_{style_{comp}}) + 0.1\mathcal{L}_{tv}. \tag{12}$$

4 Training and Examples

Both models have been implemented, trained and tested with PyTorch. The models were trained using the validation split of the Places2 dataset, which contains 36500 images; of these, 1460 were held out for validation. Random cropping and mirror padding was used to augment the training samples.

4.1 Context Encoder

The Context Encoder is designed to be trained with images of size 128×128 and a deterministic 56×56 center mask. The generator and discriminator networks were trained with a batch size of 64 for 370 epochs using the Adam optimizer. Learning rates were 10^{-4} for the discriminator and 10^{-3} for the generator. Training losses and Discriminator accuracy are reported in Figures 4 and 5.

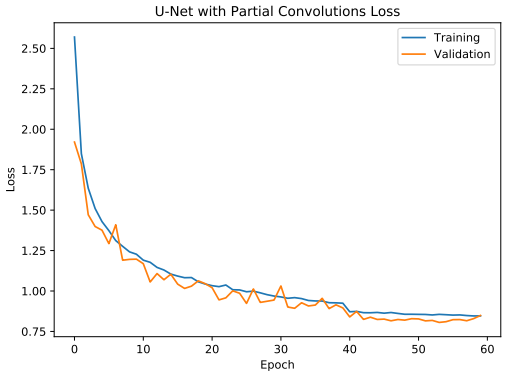


Figure 6: U-Net model training progress.

Inpainting results are shown in Figure 7. The examples show how the network progressively learns to hallucinate patches that are sharper and more coherent with the surrounding context. In particular, the generator significantly improves at synthesizing coherent sharp features such as edges and texture.

4.2 UNet with Partial Convolutions

The U-Net model was trained with inputs of size 512×512 and random binary masks, which were dynamically generated for each data sample during training. Since the missing pixels can interfere with batch normalization, the authors suggest to initially train the model with a learning rate of $2 \cdot 10^{-4}$, and then fine-tune the model freezing the batch normalization layers in the encoding blocks using a learning rate of $5 \cdot 10^{-5}$.

Training this model required significantly more time than the Context Encoder. The original paper suggests a training time of 10 days, followed by fine-tuning for 1 day on an nVidia V100 card. For the results shown in this report, the model was trained for 40 epochs and fine-tuned for further 20 epochs. The loss function progress is reported in Figure 6, while inpainting examples are reported in Figure 8.

References

[1] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[2] S. Iizuka, E. Simo-Serra, and H. Ishikawa. Globally and locally consistent image completion. *ACM Trans. Graph.*, 36(4), July 2017. ISSN 0730-0301. doi: 10.1145/3072959.3073659. URL `https://doi.org/10.1145/3072959.3073659`.

[3] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[4] G. Liu, F. A. Reda, K. J. Shih, T.-C. Wang, A. Tao, and B. Catanzaro. Image inpainting for irregular holes using partial convolutions. In *The European Conference on Computer Vision (ECCV)*, 2018.

[5] G. Liu, K. J. Shih, T.-C. Wang, F. A. Reda, K. Sapra, Z. Yu, A. Tao, and B. Catanzaro. Partial convolution based padding. In *arXiv preprint arXiv:1811.11718*, 2018.

[6] D. Pathak, P. Krähenbühl, J. Donahue, T. Darrell, and A. Efros. Context encoders: Feature learning by inpainting. 2016.

[7] J. Portilla and E. P. Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *International Journal of Computer Vision*, 40:49–71, 2000.

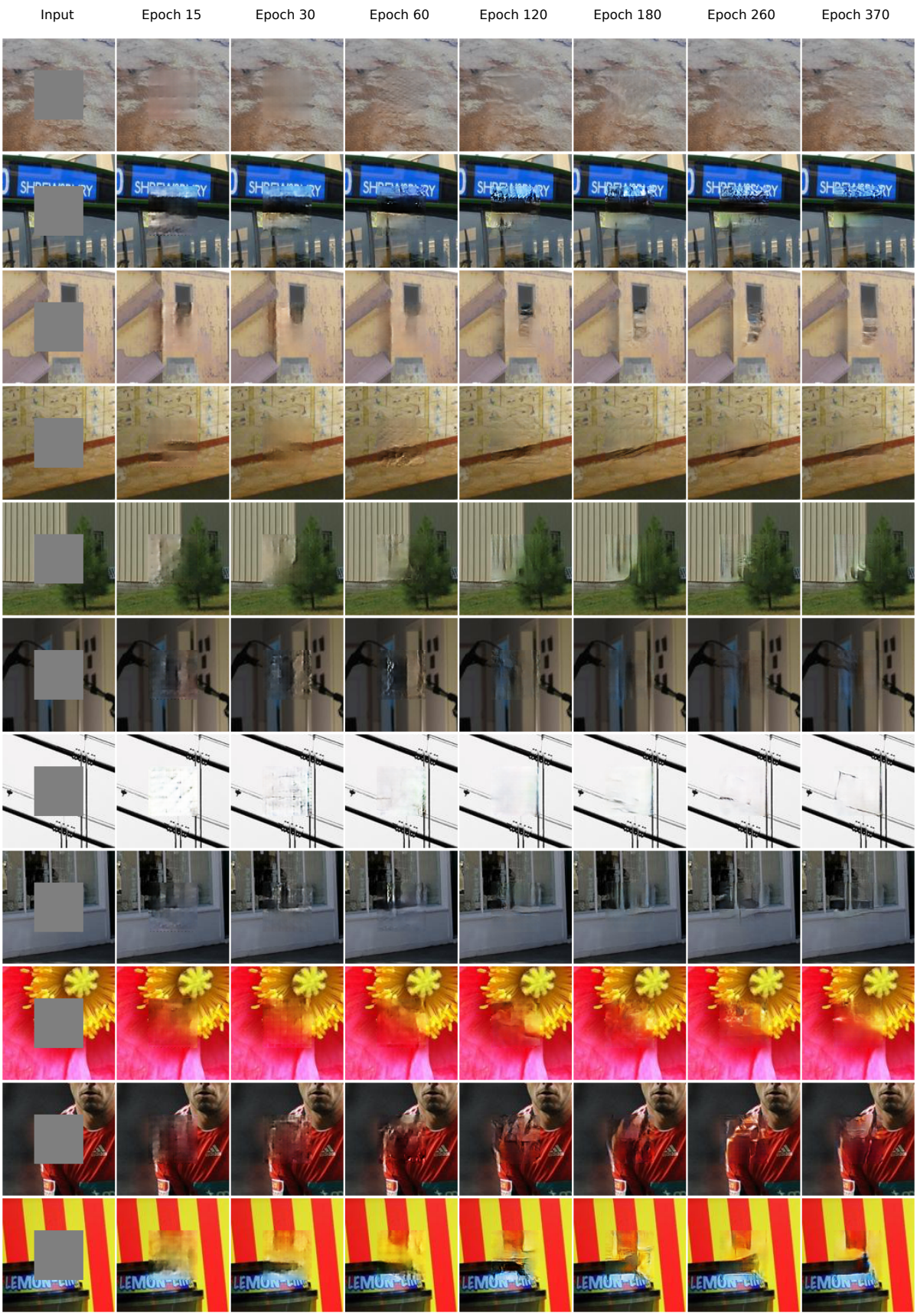


Figure 7: Context Encoder inpainting progress during training. Samples taken from the 1460 validation images.

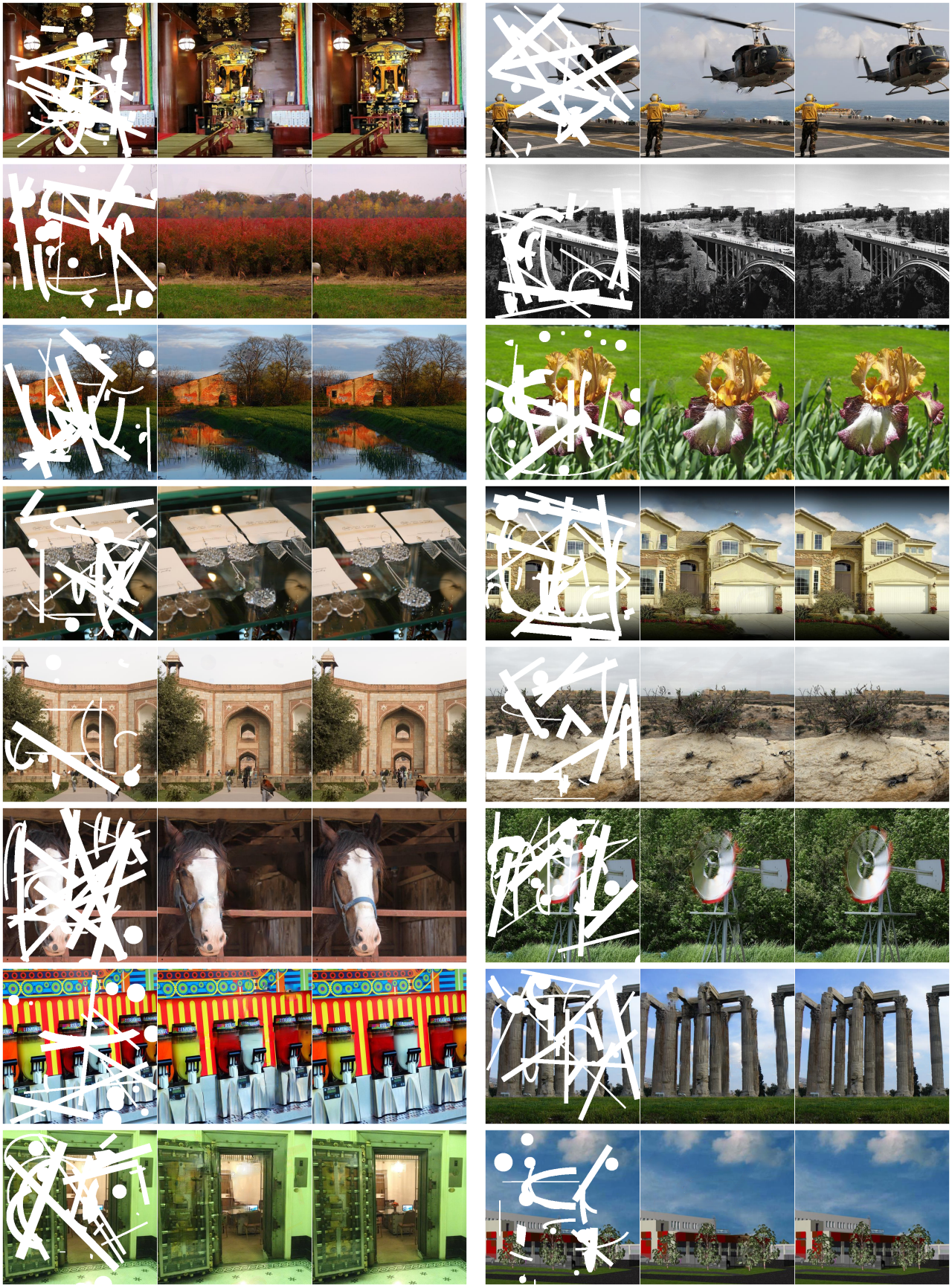


Figure 8: U-Net with partial convolutions validation samples (input, input with missing region inpainted, ground truth).

[8] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.

[9] O. Ronneberger, P.Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351 of *LNCS*, pages 234–241. Springer, 2015. URL `http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a`. (available on arXiv:1505.04597 [cs.CV]).

[10] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[11] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang. Generative image inpainting with contextual attention, 2018.

[12] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. Huang. Free-form image inpainting with gated convolution, 2019.

[13] Y. Zeng, Z. Lin, J. Yang, J. Zhang, E. Shechtman, and H. Lu. High-resolution image inpainting with iterative confidence feedback and guided upsampling, 2020.