Generative Adversarial Networks under Spatial Constraints

Cyprien Ruffino

March 20, 2018

1 Introduction

In this document, we will present our approach to the problem of training generative adversarial networks to generate data with respect to spatial constraints. This approach will be introduced both as a theoretical framework and with a dataset of ours as an example, which will facilitate both the formulation of the constraints and be a comprehensive visual support.

1.1 Data

In this document, we will use a proprietary geostatistical dataset which consists of black and white pictures describing underground channels (see figure 1). Those pictures will be described as a matrix $c \in M_{m \times n}$ with $c_{i,j} \in [-1, 1], i \leq n, j \leq m$ where -1 is black and 1 is white.

1.2 Spatial constraints

We introduce a new kind of constraints, spatial constraints, which consists of a subset of the data we want to generate (usually around $\sim 1\%$ of the data). This name comes from the fact that in the 2D case, it can be viewed as an image where the majority of its pixels are 0 (see figure 1).

As the dataset from which we want to learn does not contains such constraints, we generate them by sampling a data X from our distribution and by randomly selecting features in this data. In practice for the 2D case, we generate a mask $m \in M_{m \times n}$ with $M_{i,j} \in \{0, 1\}, i \leq n, j \leq m$ and take the Hadamard product $M \odot X$ between the sampled data and the mask.

1.3 Naming conventions

From now on, we will use the following naming conventions :

- $\mathbb{P}_r$ the real-data distribution : $x \sim \mathbb{P}_r, x \in M_{m,n}$
- $\mathbb{P}_z$ a "classical" distribution (in most cases, an uniform distribution $U[0, 1]^n$)

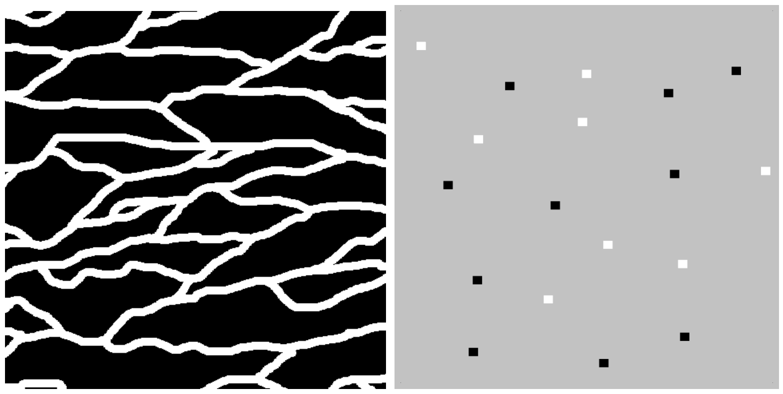


Figure 1: a) A Data sampled from our dataset, b) A constraint obtained from a sampled data and a mask

- $\mathbb{P}_m$ the distribution in which the constraint masks will be sampled (1.2) $m \sim \mathbb{P}_m, m \in \{0,1\}^{n \times n}$
- c a shortcut for $x \sim \mathbb{P}_r, m \sim \mathbb{P}_m \, m \odot x$. We will note $\mathbb{P}_c$ the distribution of c with respect to $\mathbb{P}_r$ and $\mathbb{P}_m$.

2 Background

2.1 Generative Adversarial Networks

Generative Adversarial Networks[**?**] are a method for training generative neural networks in an unsupervised way. It consists in a game between a generator G and a discriminator network D, in which the generator learns to produce new data and discriminator learns to distinguish real examples from generated ones.

More formally, training GANs is equivalent to solving the following minimax game :

$$\min_G \max_D L(D,G) = \mathop{\mathbb{E}}_{x \sim P_r}[\log(D(x))] + \mathop{\mathbb{E}}_{z \sim P_z}[\log(1 - D(G(z)))] \quad (1)$$

From this problem, one can derive a loss function for both the generator and the discriminator and find that minimizing those losses is equivalent to solving the minimax game, as well as minimizing the Kullback-Leibler divergence between the real-data distribution and the distribution of samples which can be generated by G.

In this document, we will focus on a more recent approach, the Wasserstein GAN ([**?**]), in which the problem is reformulated to minimize an Earth-Mover distance instead of the KL divergence. This new problem can be formulated as :

$$\min_G \max_D L(D,G) = \mathop{\mathbb{E}}_{x\sim P_r}[D(x)] + \mathop{\mathbb{E}}_{z\sim P_z}[1 - D(G(z))] \tag{2}$$

From this formulation, we can derive the the following loss fuctions, which we will use as our objectives :

$$L_D = \mathop{\mathbb{E}}_{x\sim P_r}[D(x)] - \mathop{\mathbb{E}}_{z\sim P_z}[D(G(z))] \tag{3}$$

$$L_G = -\mathop{\mathbb{E}}_{z\sim P_z}[D(G(z))] \tag{4}$$

2.2 Spatial Generative Adversarial Networks

Spacial Generative Adversarial Networks (SGANs)[?] are a category of GANs in which both the generator and the discriminator nets are fully-convolutional networks. This characteristic brings several specificities. First, the networks are unable to pick up global dependencies in the data, because the total size of the data is greater than the total receptive field of the networks. Then, the output is not a single scalar, but a matrix of values. so the new objective becomes the mean of the discriminator's output :

$$\min_G \max_D L(D,G) = \mathop{\mathbb{E}}_{x\sim P_r}[\frac{1}{m}\sum_{i=1}^{m}(D(x))] + \mathop{\mathbb{E}}_{z\sim P_z}[\frac{1}{m}\sum_{i=1}^{m}(1 - D(G(z)))] \tag{5}$$

Despite their inability to pick up global dependencies in the data, SGANs have several advantages over the regular GANs. First, because dense layers usually contains the majority of the network's weights, SGAN have far less parameters than more classical GANs. Then, as both of the networks are fully-convolutional, there is no restrictions over the size or shape of the input, as long as it keeps the same number of dimensions.

2.3 Conditional Generative Adversarial Networks

Conditional Generative Adversarial Networks[?] are a subclass of GANs in which some additional information y is given to both the generator and the discriminator as an input. This allows the conditioning of the generator without changing anything else in the learning algorithm.

$$\min_G \max_D L(D,G) = \mathop{\mathbb{E}}_{\substack{y\sim P_y \\ x\sim P_{r|k}}}[(D(x|y))] + \mathop{\mathbb{E}}_{\substack{\tilde{y}\sim P_y \\ z\sim P_z}}[(1 - D(G(z|\tilde{y})|\tilde{y}))] \tag{6}$$

3 Generative Adversarial Networks under spatial constraints

In this section, we will present the different three approaches we introduced to tackle this problem. We first introduce various objectives, derived from the original GAN formulation, which will be optimized in our approaches. We will then introduce an unsupervised approach, where the constraints are only additional inputs of both the generator and the discriminator and finally, we present a supervised approach in which a cost between the constraints and the generated data is explicitly added to the generator's objective.

3.1 Integrating constraints to the objectives

Our first approach is the spacial counterpart to the CGAN approach for constraining GANs. In this approach, the constraints are given to both of the networks as another input matrix and the standard GAN objective becomes :

$$L_D = \mathbb{E}_{\substack{x\sim\mathbb{P}_r\\ m\sim\mathbb{P}_m}}[D(x, m\odot x)] - \mathop{\mathbb{E}}_{\substack{\tilde{x}\sim\mathbb{P}_r\\ \tilde{m}\sim\mathbb{P}_m\\ z\sim\mathbb{P}_z}}[D(G(z,\tilde{m}\odot\tilde{x}),\tilde{m}\odot\tilde{x})] \tag{7}$$

$$L_G = \mathop{\mathbb{E}}_{\substack{z\sim\mathbb{P}_z\\ \tilde{x}\sim\mathbb{P}_r\\ \tilde{m}\sim\mathbb{P}_m}}[D(G(z,\tilde{m}\odot\tilde{x}))] \tag{8}$$

A second type of approach consists in training the generator to respect constraints by adding an explicit cost to the generator's objective, while still training it to generate data which are close to the real data distribution in an unsupervised way. This cost is a mean squared error between the values of the generated data at the constrained points and the constraint :

$$L_C = \mathbb{E}_{\substack{\hat{x}\sim\mathbb{P}_r\\ \hat{z}\sim\mathbb{P}_z\\ \hat{m}\sim\mathbb{P}_z}}||\hat{m}\odot\hat{x} - \hat{m}\odot G(\hat{z},\hat{m}\odot\hat{x})||_2 \tag{9}$$

We then can add this new objective as a penalty on the generator's cost :

$$L_{G'} = \mathop{\mathbb{E}}_{\substack{z\sim\mathbb{P}_z\\ \tilde{x}\sim\mathbb{P}_r\\ \tilde{m}\sim\mathbb{P}_m}}[D(G(z,\tilde{m}\odot\tilde{x}))] + \lambda L_C \tag{10}$$

Note that this objective is compatible with the unsupervised approach, as the constraints can be fed to the discriminator or not. In that case, the objectives become :

$$L_D = \mathop{\mathbb{E}}_{\substack{x\sim\mathbb{P}_r\\ m\sim\mathbb{P}_m}}[D(x, m\odot x)] - \mathop{\mathbb{E}}_{\substack{z\sim\mathbb{P}_z\\ \tilde{x}\sim\mathbb{P}_r\\ \tilde{m}\sim\mathbb{P}_m}}[D(G(z,\tilde{m}\odot\tilde{x}),\tilde{m}\odot\tilde{x})] \tag{11}$$

$$L_G = \mathop{\mathbb{E}}_{\substack{z\sim\mathbb{P}_z\\ \tilde{x}\sim\mathbb{P}_r\\ \tilde{m}\sim\mathbb{P}_m}} [D(G(z, \tilde{m} \odot \tilde{x}), \tilde{m} \odot \tilde{x})] + \lambda L_C \tag{12}$$

3.2 Learning with constraints

From these objectives, we derive several GAN formulations that integrates constraints. We first present the unsupervised approach, inherited from CGAN and optimizing the objective defined in (18) :

$$\min_G \max_D L(D,G) = \mathbb{E}_{\substack{x\sim\mathbb{P}_r\\ m\sim\mathbb{P}_m}} [D(x, m \odot x)] - \mathop{\mathbb{E}}_{\substack{\tilde{x}\sim\mathbb{P}_r\\ \tilde{m}\sim\mathbb{P}_m\\ z\sim\mathbb{P}_z}} [D(G(z, \tilde{m} \odot \tilde{x}), \tilde{m} \odot \tilde{x})] \tag{13}$$

Then, we propose the following approaches which introduce the explicit cost (20) added to the generator. This cost can be added as a penalty term with a weight λ :

$$\begin{aligned} \min_G \max_D L(D,G) = \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)] - \mathop{\mathbb{E}}_{\substack{\tilde{x}\sim\mathbb{P}_r\\ \tilde{m}\sim\mathbb{P}_m\\ z\sim\mathbb{P}_z}} [D(G(z, \tilde{m} \odot \tilde{x}))] \\ + \lambda\, \mathbb{E}_{\substack{\hat{x}\sim\mathbb{P}_r\\ \hat{z}\sim\mathbb{P}_z\\ \hat{m}\sim\mathbb{P}_z}} ||\hat{m} \odot \hat{x} - \hat{m} \odot G(\hat{z}, \hat{m} \odot \hat{x})||_2 \end{aligned} \tag{14}$$

Another solution is to optimize the explicit cost function separately, forming a multi-objective problem :

$$\begin{cases} \min_G \max_D L(D,G) = \mathbb{E}_{\substack{x\sim\mathbb{P}_r\\ m\sim\mathbb{P}_m}} [D(x)] - \mathop{\mathbb{E}}_{\substack{\tilde{x}\sim\mathbb{P}_r\\ \tilde{m}\sim\mathbb{P}_m\\ z\sim\mathbb{P}_z}} [D(G(z, \tilde{m} \odot \tilde{x}))] \\ \min_G L_C(G) = \mathbb{E}_{\substack{\hat{x}\sim\mathbb{P}_r\\ \hat{z}\sim\mathbb{P}_z\\ \hat{m}\sim\mathbb{P}_z}} ||\hat{m} \odot \hat{x} - \hat{m} \odot G(\hat{z}, \hat{m} \odot \hat{x})||_2 \end{cases} \tag{15}$$

Finally, we can merge both of these approaches by giving the constraints as an input to the discriminator and adding the explicit objective :

$$\begin{aligned} \min_G \max_D L(D,G) = \mathbb{E}_{\substack{x\sim\mathbb{P}_r\\ m\sim\mathbb{P}_m}} [D(x, m \odot x)] - \mathop{\mathbb{E}}_{\substack{\tilde{x}\sim\mathbb{P}_r\\ \tilde{m}\sim\mathbb{P}_m\\ z\sim\mathbb{P}_z}} [D(G_\theta(z, \tilde{m} \odot \tilde{x}), \tilde{m} \odot \tilde{x})] \\ + \lambda\, \mathbb{E}_{\substack{\hat{x}\sim\mathbb{P}_r\\ \hat{z}\sim\mathbb{P}_z\\ \hat{m}\sim\mathbb{P}_z}} ||\hat{m} \odot \hat{x} - \hat{m} \odot G(\hat{z}, \hat{m} \odot \hat{x})||_2 \end{aligned} \tag{16}$$

$$\begin{cases} \min\limits_G \max\limits_D L(D,G) = \mathbb{E}_{\substack{x\sim\mathbb{P}_r \\ m\sim\mathbb{P}_m}} [D(x, m\odot x)] - \mathbb{E}_{\substack{\tilde{x}\sim\mathbb{P}_r \\ \tilde{m}\sim\mathbb{P}_m \\ z\sim\mathbb{P}_z}} [D(G(z,\tilde{m}\odot\tilde{x}),\tilde{m}\odot\tilde{x})] \\ \min\limits_G L_C(G) = \mathbb{E}_{\substack{\hat{x}\sim\mathbb{P}_r \\ \hat{z}\sim\mathbb{P}_z \\ \hat{m}\sim\mathbb{P}_z}} ||\hat{m}\odot\hat{x} - \hat{m}\odot G(\hat{z},\hat{m}\odot\hat{x})||_2 \end{cases} \tag{17}$$

4 Network architectures

In this section, we will present the different architectures we came with for both the supervised and the unsupervised approach. Those architectures are fully-convolutional and have a similar number of parameters (around 8 million). The constraint is added as additional channels for both the generator and the discriminator.

The different generators are made from transposed convolutional (also called deconvolution) layers, with ReLU activations. BatchNorm[?] is added between each layers, with the exception of the last layer. All the upscaling is done with striding (1/2 per layer).

The discriminator is the same as [?], which is itself a fully-convolutional adaptation of the DGCAN[?] discriminator. It consists in five convolutional layers with with LeakyReLU activations (except for the last one which has a sigmoid activation) and BatchNorm between all the layers except after the first layer and before the last one. Each layer has a striding of 2, allowing downscaling without pooling.

4.1 Dilatation only

This architecture uses dilated convolutions (or "atrous" convolutions) and keeps both the input noise and the constraint to the same constant size

4.2 Encoder-decoder

4.3 Encoder-decoder with upscaling

4.4 Upscaling and dilatation

4.5 Encoder-decoder with noise encoding

5 Learning constraints by gradient descent in the noise

This is done by backpropagating the gradients of the previous cost through the generator and optimizing it with gradient descent A venir ?

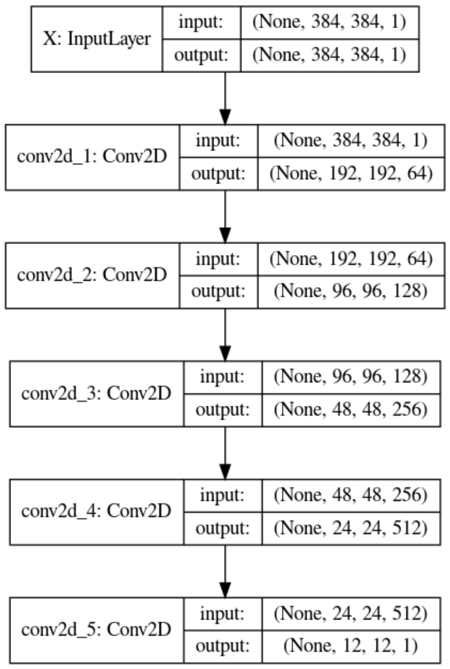

X: InputLayer
input: (None, 384, 384, 1)
output: (None, 384, 384, 1)
conv2d_1: Conv2D
input: (None, 384, 384, 1)
output: (None, 192, 192, 64)
conv2d_2: Conv2D
input: (None, 192, 192, 64)
output: (None, 96, 96, 128)
conv2d_3: Conv2D
input: (None, 96, 96, 128)
output: (None, 48, 48, 256)
conv2d_4: Conv2D
input: (None, 48, 48, 256)
output: (None, 24, 24, 512)
conv2d_5: Conv2D
input: (None, 24, 24, 512)
output: (None, 12, 12, 1)

Figure 2: Discriminator

6 Appendix

6.1 Network architectures diagrams

6.2 Reformulation with a constraint variable

6.3 Integrating constraints to the objectives

Our first approach is the spacial counterpart to the CGAN approach for constraining GANs. In this approach, the constraints are given to both of the networks as another input matrix and the standard GAN objective becomes :

$$L_D = \mathbb{E}_{\substack{x\sim\mathbb{P}_r \\ m\sim\mathbb{P}_m \\ c\sim\mathbb{P}_c(m,x)}} [D(x,c)] - \mathbb{E}_{\substack{\tilde{x}\sim\mathbb{P}_r \\ \tilde{m}\sim\mathbb{P}_m \\ z\sim\mathbb{P}_z \\ \tilde{c}\sim\mathbb{P}_c(\tilde{m},\tilde{x})}} [D(G(z,\tilde{c}),\tilde{c})] \tag{18}$$

$$L_G = \mathbb{E}_{\substack{z\sim\mathbb{P}_z \\ \tilde{x}\sim\mathbb{P}_r \\ \tilde{m}\sim\mathbb{P}_m \\ c\sim\mathbb{P}_c(m,x)}} [D(G(z,\tilde{c}))] \tag{19}$$

A second type of approach consists in training the generator to respect constraints by adding an explicit cost to the generator's objective, while still train-

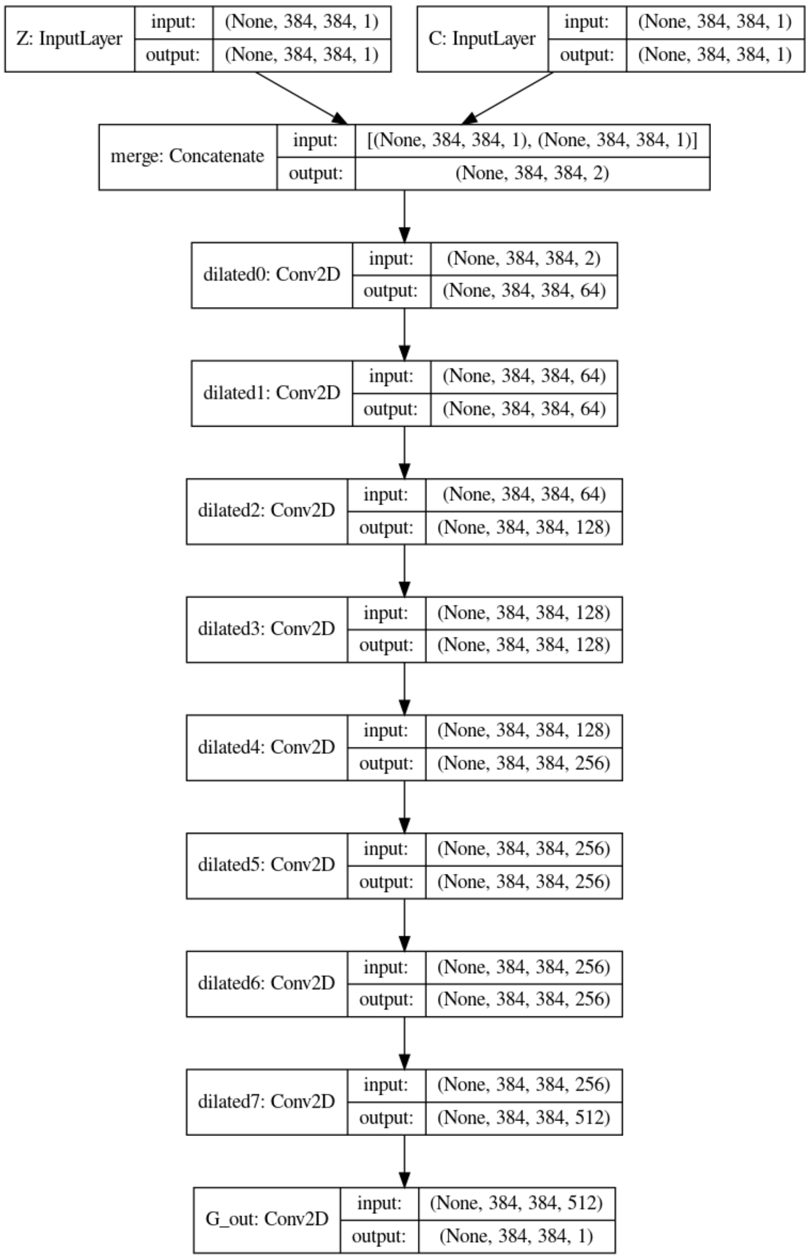
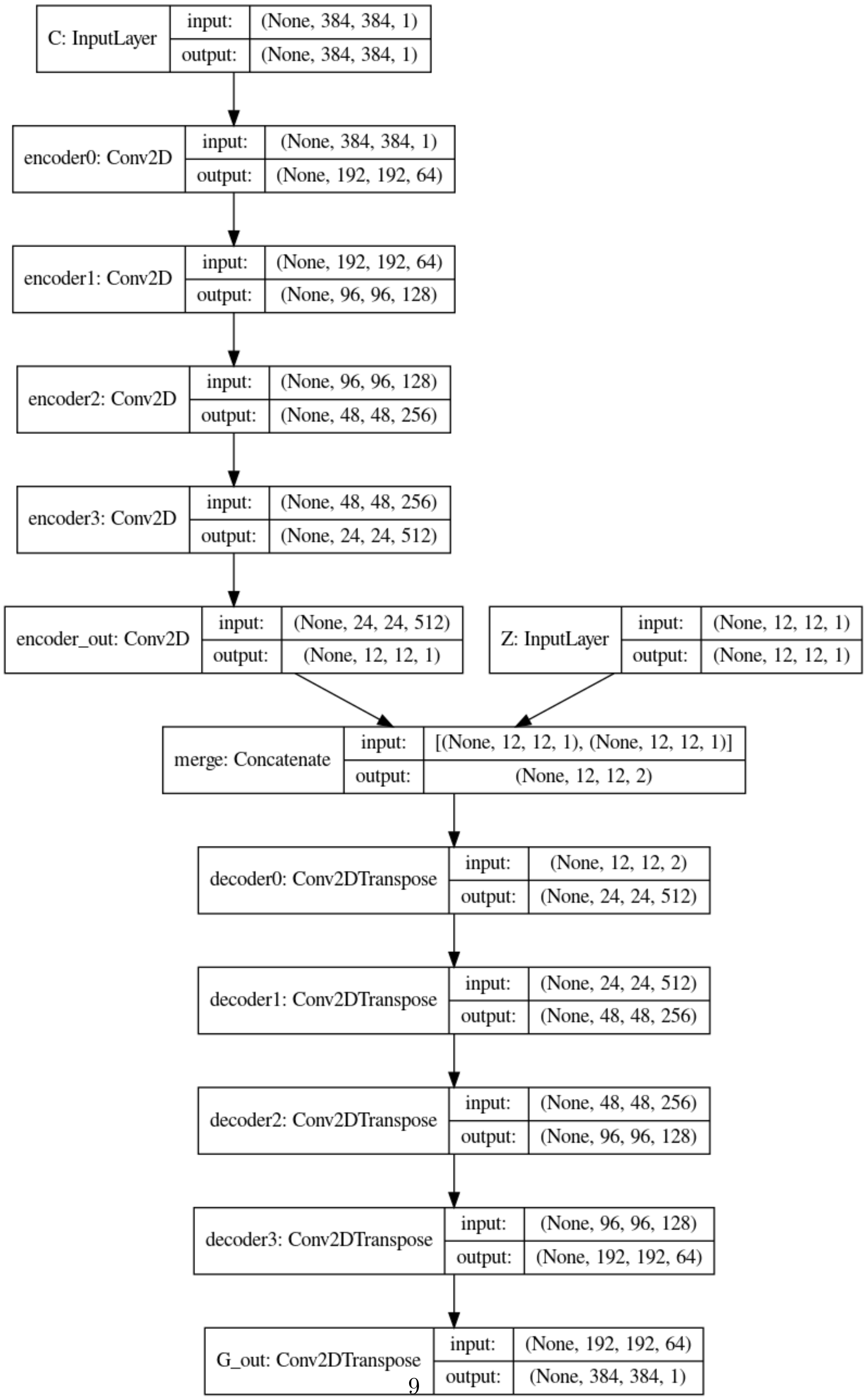

Z: InputLayer
input: (None, 384, 384, 1)
output: (None, 384, 384, 1)
C: InputLayer
input: (None, 384, 384, 1)
output: (None, 384, 384, 1)
merge: Concatenate
input: [(None, 384, 384, 1), (None, 384, 384, 1)]
output: (None, 384, 384, 2)
dilated0: Conv2D
input: (None, 384, 384, 2)
output: (None, 384, 384, 64)
dilated1: Conv2D
input: (None, 384, 384, 64)
output: (None, 384, 384, 64)
dilated2: Conv2D
input: (None, 384, 384, 64)
output: (None, 384, 384, 128)
dilated3: Conv2D
input: (None, 384, 384, 128)
output: (None, 384, 384, 128)
dilated4: Conv2D
input: (None, 384, 384, 128)
output: (None, 384, 384, 256)
dilated5: Conv2D
input: (None, 384, 384, 256)
output: (None, 384, 384, 256)
dilated6: Conv2D
input: (None, 384, 384, 256)
output: (None, 384, 384, 256)
dilated7: Conv2D
input: (None, 384, 384, 256)
output: (None, 384, 384, 512)
G_out: Conv2D
input: (None, 384, 384, 512)
output: (None, 384, 384, 1)

Figure 3: Dilatation only



C: InputLayer
input: (None, 384, 384, 1)
output: (None, 384, 384, 1)
encoder0: Conv2D
input: (None, 384, 384, 1)
output: (None, 192, 192, 64)
encoder1: Conv2D
input: (None, 192, 192, 64)
output: (None, 96, 96, 128)
encoder2: Conv2D
input: (None, 96, 96, 128)
output: (None, 48, 48, 256)
encoder3: Conv2D
input: (None, 48, 48, 256)
output: (None, 24, 24, 512)
encoder_out: Conv2D
input: (None, 24, 24, 512)
output: (None, 12, 12, 1)
Z: InputLayer
input: (None, 12, 12, 1)
output: (None, 12, 12, 1)
merge: Concatenate
input: [(None, 12, 12, 1), (None, 12, 12, 1)]
output: (None, 12, 12, 2)
decoder0: Conv2DTranspose
input: (None, 12, 12, 2)
output: (None, 24, 24, 512)
decoder1: Conv2DTranspose
input: (None, 24, 24, 512)
output: (None, 48, 48, 256)
decoder2: Conv2DTranspose
input: (None, 48, 48, 256)
output: (None, 96, 96, 128)
decoder3: Conv2DTranspose
input: (None, 96, 96, 128)
output: (None, 192, 192, 64)
G_out: Conv2DTranspose
input: (None, 192, 192, 64)
output: (None, 384, 384, 1)

Figure 4: Encoder-decoder

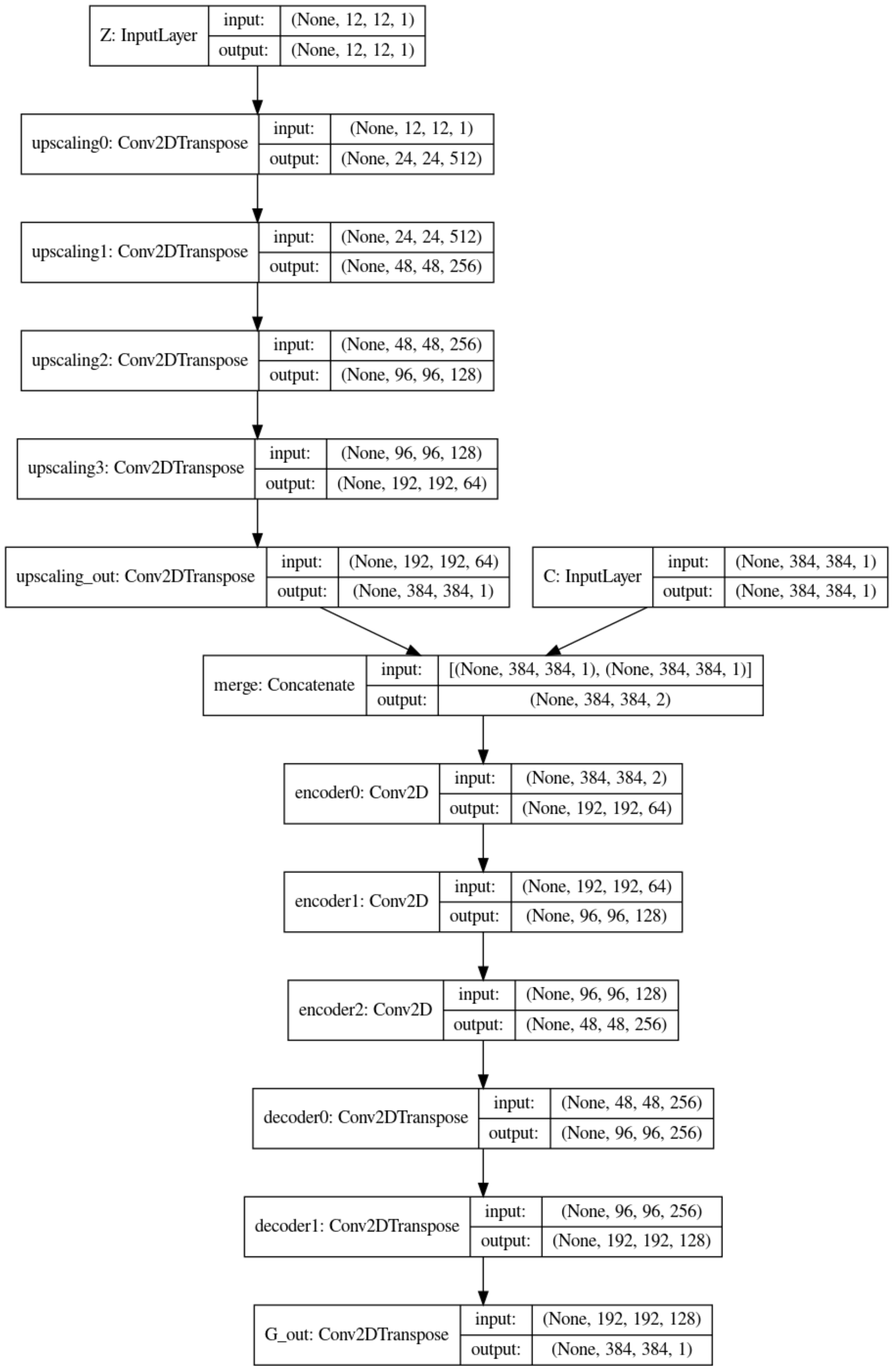

Z: InputLayer
input: (None, 12, 12, 1)
output: (None, 12, 12, 1)
upscaling0: Conv2DTranspose
input: (None, 12, 12, 1)
output: (None, 24, 24, 512)
upscaling1: Conv2DTranspose
input: (None, 24, 24, 512)
output: (None, 48, 48, 256)
upscaling2: Conv2DTranspose
input: (None, 48, 48, 256)
output: (None, 96, 96, 128)
upscaling3: Conv2DTranspose
input: (None, 96, 96, 128)
output: (None, 192, 192, 64)
upscaling_out: Conv2DTranspose
input: (None, 192, 192, 64)
output: (None, 384, 384, 1)
C: InputLayer
input: (None, 384, 384, 1)
output: (None, 384, 384, 1)
merge: Concatenate
input: [(None, 384, 384, 1), (None, 384, 384, 1)]
output: (None, 384, 384, 2)
encoder0: Conv2D
input: (None, 384, 384, 2)
output: (None, 192, 192, 64)
encoder1: Conv2D
input: (None, 192, 192, 64)
output: (None, 96, 96, 128)
encoder2: Conv2D
input: (None, 96, 96, 128)
output: (None, 48, 48, 256)
decoder0: Conv2DTranspose
input: (None, 48, 48, 256)
output: (None, 96, 96, 256)
decoder1: Conv2DTranspose
input: (None, 96, 96, 256)
output: (None, 192, 192, 128)
G_out: Conv2DTranspose
input: (None, 192, 192, 128)
output: (None, 384, 384, 1)

Figure 5: Encoder-decoder with upscaling

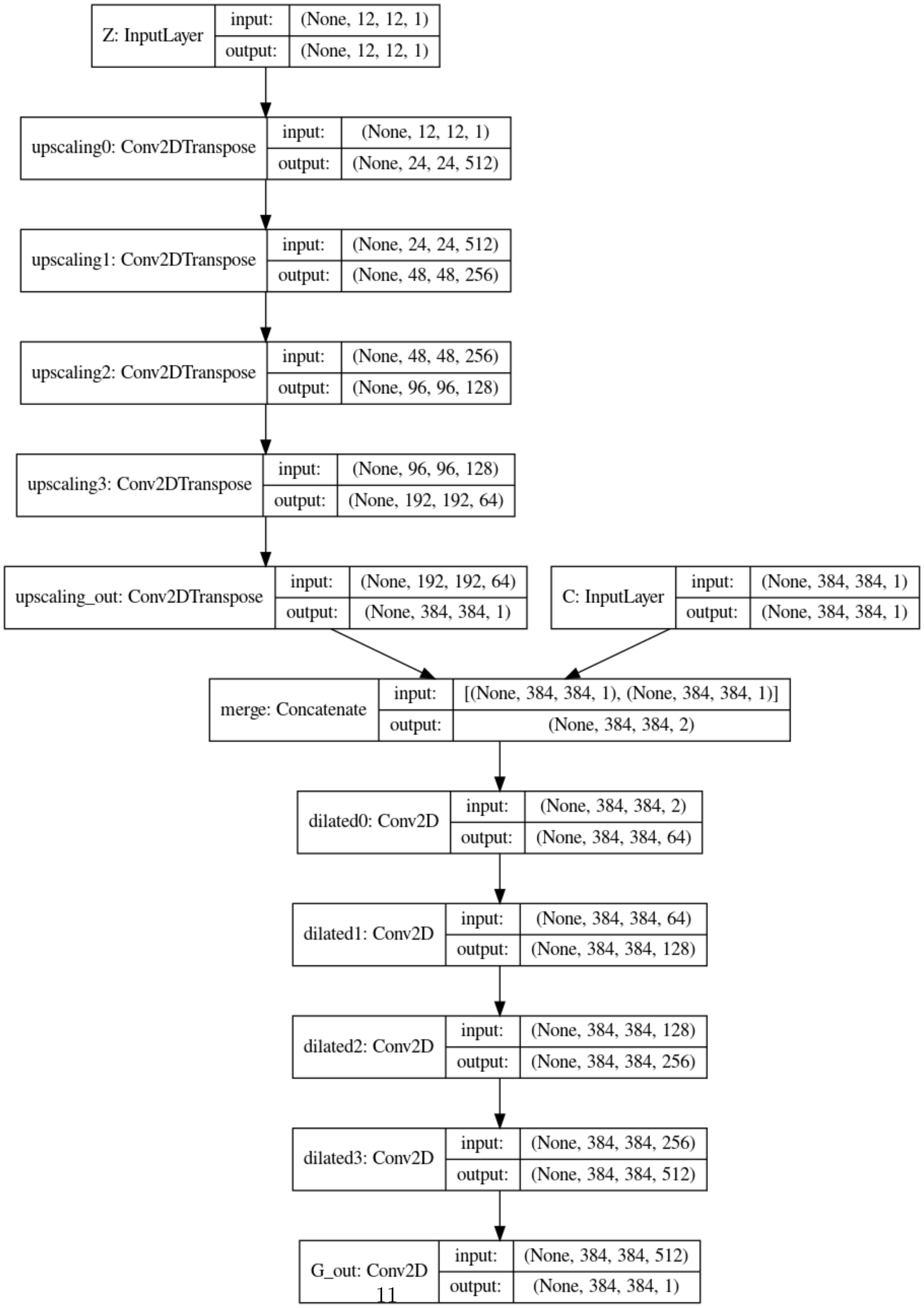

Z: InputLayer
input: (None, 12, 12, 1)
output: (None, 12, 12, 1)
upscaling0: Conv2DTranspose
input: (None, 12, 12, 1)
output: (None, 24, 24, 512)
upscaling1: Conv2DTranspose
input: (None, 24, 24, 512)
output: (None, 48, 48, 256)
upscaling2: Conv2DTranspose
input: (None, 48, 48, 256)
output: (None, 96, 96, 128)
upscaling3: Conv2DTranspose
input: (None, 96, 96, 128)
output: (None, 192, 192, 64)
upscaling_out: Conv2DTranspose
input: (None, 192, 192, 64)
output: (None, 384, 384, 1)
C: InputLayer
input: (None, 384, 384, 1)
output: (None, 384, 384, 1)
merge: Concatenate
input: [(None, 384, 384, 1), (None, 384, 384, 1)]
output: (None, 384, 384, 2)
dilated0: Conv2D
input: (None, 384, 384, 2)
output: (None, 384, 384, 64)
dilated1: Conv2D
input: (None, 384, 384, 64)
output: (None, 384, 384, 128)
dilated2: Conv2D
input: (None, 384, 384, 128)
output: (None, 384, 384, 256)
dilated3: Conv2D
input: (None, 384, 384, 256)
output: (None, 384, 384, 512)
G_out: Conv2D
input: (None, 384, 384, 512)
output: (None, 384, 384, 1)

Figure 6: Upscaling and dilatation

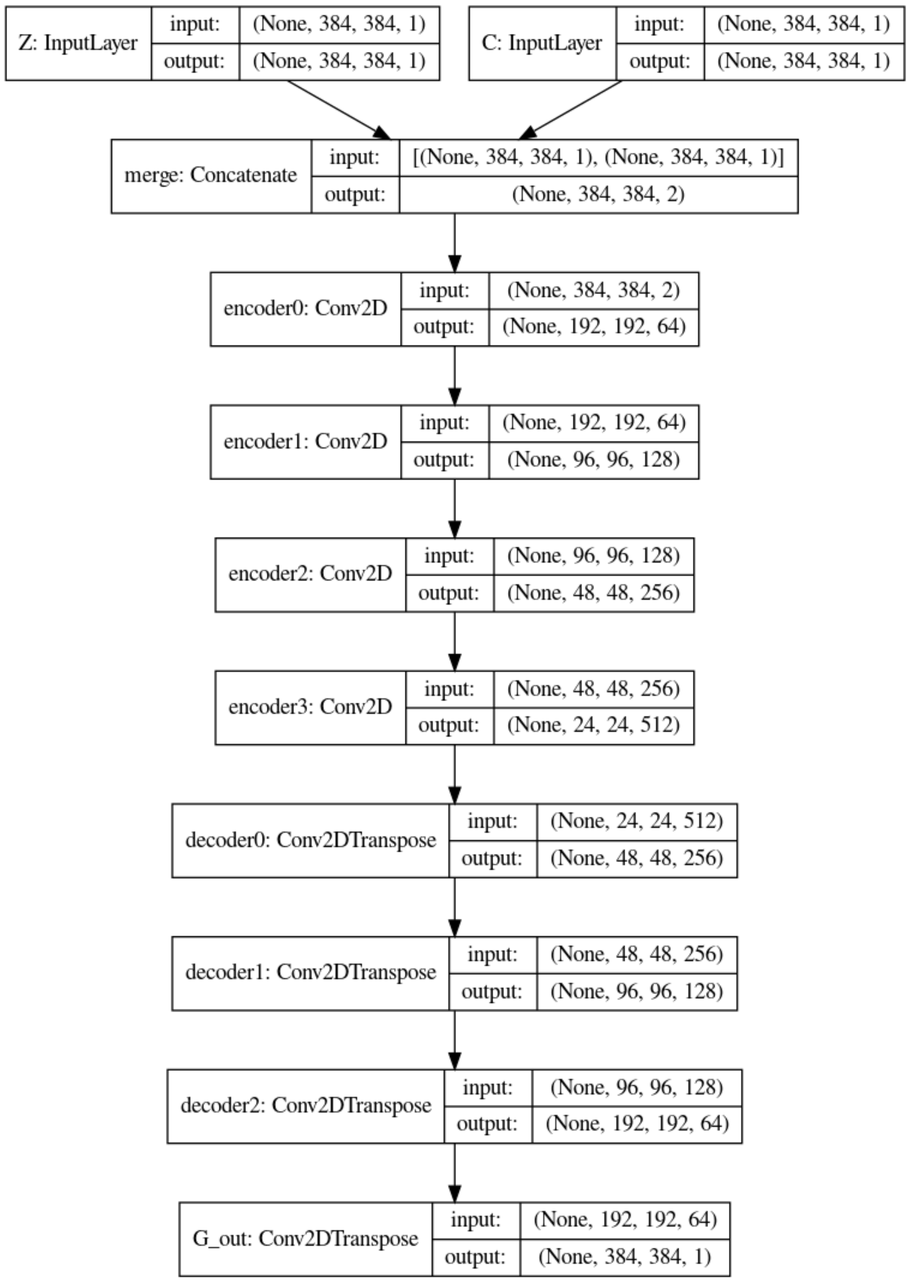

Z: InputLayer
input: (None, 384, 384, 1)
output: (None, 384, 384, 1)
C: InputLayer
input: (None, 384, 384, 1)
output: (None, 384, 384, 1)
merge: Concatenate
input: [(None, 384, 384, 1), (None, 384, 384, 1)]
output: (None, 384, 384, 2)
encoder0: Conv2D
input: (None, 384, 384, 2)
output: (None, 192, 192, 64)
encoder1: Conv2D
input: (None, 192, 192, 64)
output: (None, 96, 96, 128)
encoder2: Conv2D
input: (None, 96, 96, 128)
output: (None, 48, 48, 256)
encoder3: Conv2D
input: (None, 48, 48, 256)
output: (None, 24, 24, 512)
decoder0: Conv2DTranspose
input: (None, 24, 24, 512)
output: (None, 48, 48, 256)
decoder1: Conv2DTranspose
input: (None, 48, 48, 256)
output: (None, 96, 96, 128)
decoder2: Conv2DTranspose
input: (None, 96, 96, 128)
output: (None, 192, 192, 64)
G_out: Conv2DTranspose
input: (None, 192, 192, 64)
output: (None, 384, 384, 1)

Figure 7: Encoder decoder with large noise

ing it to generate data which are close to the real data distribution in an unsupervised way. This cost is a mean squared error between the values of the generated data at the constrained points and the constraint :

$$L_C = \mathbb{E}_{\substack{\hat{x}\sim\mathbb{P}_r \\ \hat{z}\sim\mathbb{P}_z \\ \hat{m}\sim\mathbb{P}_z \\ \hat{c}\sim\mathbb{P}_c(\tilde{m},\tilde{x})}} ||\hat{c} - \hat{m} \odot G(\hat{z}, \hat{c})||_2 \tag{20}$$

We then can add this new objective as a penalty on the generator's cost :

$$L_{G'} = \mathop{\mathbb{E}}_{\substack{z\sim\mathbb{P}_z \\ \tilde{x}\sim\mathbb{P}_r \\ \tilde{m}\sim\mathbb{P}_m \\ \tilde{c}\sim\mathbb{P}_c(\tilde{m},\tilde{x})}} [D(G(z, \tilde{c}))] + \lambda L_C \tag{21}$$

Note that this objective is compatible with the unsupervised approach, as the constraints can be fed to the discriminator or not. In that case, the objectives become :

$$L_D = \mathop{\mathbb{E}}_{\substack{x\sim\mathbb{P}_r \\ m\sim\mathbb{P}_m \\ c\sim\mathbb{P}_c(m,x)}} [D(x, c)] - \mathop{\mathbb{E}}_{\substack{z\sim\mathbb{P}_z \\ \tilde{x}\sim\mathbb{P}_r \\ \tilde{m}\sim\mathbb{P}_m \\ \tilde{c}\sim\mathbb{P}_c(\tilde{m},\tilde{x})}} [D(G(z, \tilde{c}), \tilde{c})] \tag{22}$$

$$L_G = \mathop{\mathbb{E}}_{\substack{z\sim\mathbb{P}_z \\ \tilde{x}\sim\mathbb{P}_r \\ \tilde{m}\sim\mathbb{P}_m \\ \tilde{c}\sim\mathbb{P}_c(\tilde{m},\tilde{x})}} [D(G(z, \tilde{c}), \tilde{c})] + \lambda L_C \tag{23}$$

6.4 Learning with constraints

From these objectives, we derive several GAN formulations that integrates constraints. We first present the unsupervised approach, inherited from CGAN and optimizing the objective defined in (18) :

$$\min_G \max_D L(D, G) = \mathbb{E}_{\substack{x\sim\mathbb{P}_r \\ m\sim\mathbb{P}_m \\ c\sim\mathbb{P}_c(m,x)}} [D(x, c)] - \mathop{\mathbb{E}}_{\substack{\tilde{x}\sim\mathbb{P}_r \\ \tilde{m}\sim\mathbb{P}_m \\ z\sim\mathbb{P}_z \\ \tilde{c}\sim\mathbb{P}_c(\tilde{m},\tilde{x})}} [D(G(z, \tilde{c}), \tilde{c})] \tag{24}$$

Then, we propose the following approaches which introduce the explicit cost (20) added to the generator. This cost can be added as a penalty term with a weight λ :

$$\begin{aligned} \min_G \max_D L(D, G) = \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)] - &\mathop{\mathbb{E}}_{\substack{\tilde{x}\sim\mathbb{P}_r \\ \tilde{m}\sim\mathbb{P}_m \\ z\sim\mathbb{P}_z \\ \tilde{c}\sim\mathbb{P}_c(\tilde{m},\tilde{x})}} [D(G(z, \tilde{c}))] \\ &+ \lambda \mathbb{E}_{\substack{\hat{x}\sim\mathbb{P}_r \\ \hat{z}\sim\mathbb{P}_z \\ \hat{m}\sim\mathbb{P}_z \\ \hat{c}\sim\mathbb{P}_c(\hat{m},\hat{x})}} ||\hat{c} - \hat{m} \odot G(\hat{z}, \hat{c})||_2 \end{aligned} \tag{25}$$

Another solution is to optimize the explicit cost function separately, forming a multi-objective problem :

$$\begin{cases} \min\limits_G \max\limits_D L(D,G) = \mathbb{E}_{\substack{x\sim\mathbb{P}_r \\ m\sim\mathbb{P}_m \\ c\sim\mathbb{P}_c(m,x)}} [D(x)] - \mathbb{E}_{\substack{\tilde{x}\sim\mathbb{P}_r \\ \tilde{m}\sim\mathbb{P}_m \\ \tilde{c}\sim\mathbb{P}_c(\tilde{m},\tilde{x}) \\ z\sim\mathbb{P}_z}} [D(G(z,\tilde{c}))] \\ \min\limits_G L_C(G) = \mathbb{E}_{\substack{\hat{x}\sim\mathbb{P}_r \\ \hat{z}\sim\mathbb{P}_z \\ \hat{m}\sim\mathbb{P}_z \\ \hat{c}\sim\mathbb{P}_c(\hat{m},\hat{x})}} ||\hat{c} - \hat{m}\odot G(\hat{z},\hat{c})||_2 \end{cases} \tag{26}$$

Finally, we can merge both of these approaches by giving the constraints as an input to the discriminator and adding the explicit objective :

$$\begin{aligned} \min_G \max_D L(D,G) = \mathbb{E}_{\substack{x\sim\mathbb{P}_r \\ m\sim\mathbb{P}_m \\ c\sim\mathbb{P}_c(m,x)}} [D(x,c)] - \mathbb{E}_{\substack{\tilde{x}\sim\mathbb{P}_r \\ \tilde{m}\sim\mathbb{P}_m \\ \tilde{c}\sim\mathbb{P}_c(\tilde{m},\tilde{x}) \\ z\sim\mathbb{P}_z}} [D(G_\theta(z,\tilde{c}),\tilde{c})] \\ + \lambda\,\mathbb{E}_{\substack{\hat{x}\sim\mathbb{P}_r \\ \hat{z}\sim\mathbb{P}_z \\ \hat{m}\sim\mathbb{P}_z \\ \hat{c}\sim\mathbb{P}_c(\hat{m},\hat{x})}} ||\hat{c} - \hat{m}\odot G(\hat{z},\hat{c})||_2 \end{aligned} \tag{27}$$

$$\begin{cases} \min\limits_G \max\limits_D L(D,G) = \mathbb{E}_{\substack{x\sim\mathbb{P}_r \\ m\sim\mathbb{P}_m \\ c\sim\mathbb{P}_c(m,x)}} [D(x,c)] - \mathbb{E}_{\substack{\tilde{x}\sim\mathbb{P}_r \\ \tilde{m}\sim\mathbb{P}_m \\ \tilde{c}\sim\mathbb{P}_c(\tilde{m},\tilde{x}) \\ z\sim\mathbb{P}_z}} [D(G(z,\tilde{c}),\tilde{c})] \\ \min\limits_G L_C(G) = \mathbb{E}_{\substack{\hat{x}\sim\mathbb{P}_r \\ \hat{z}\sim\mathbb{P}_z \\ \hat{m}\sim\mathbb{P}_z \\ \hat{c}\sim\mathbb{P}_c(\hat{m},\hat{x})}} ||\hat{c} - \hat{m}\odot G(\hat{z},\hat{c})||_2 \end{cases} \tag{28}$$