Project: Final Report

Image to Image translation using cGANs

Mentor: Dr. Pawan Kumar
Aman Krishna - 2018201070
Gyanshu Azad Singh - 2018201073

Introduction:

The paper talks about a general approach to image-to-image translation using Conditional Generative adversarial Networks. In this report we have discussed our understanding of the paper, its implementation, approach, the architecture used and the loss function. Following it, we have mentioned our results. The report includes:

- cGAN vs GAN
- Objective Function
- Architecture of the Network
- Building a Convolutional GAN for Pokemon Generation for getting comfortable with Convolutional GAN
- Results

cGANs (Conditional GANs) vs GAN

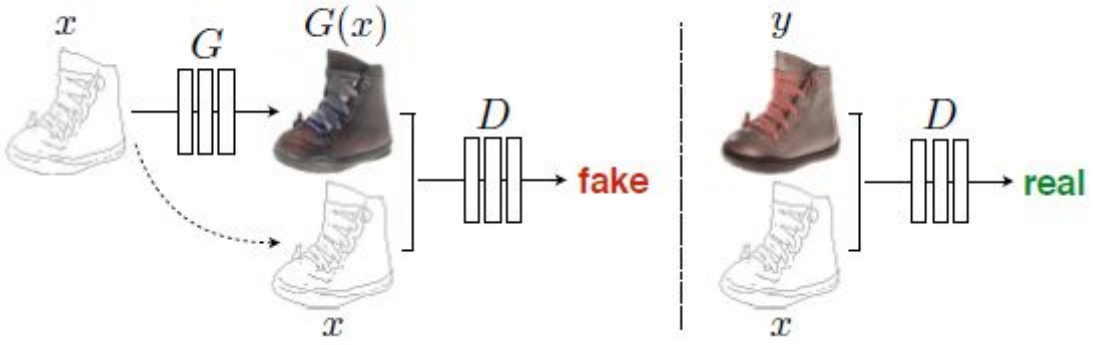
- In traditional GANs the generative model learns a mapping from random noise vector ***z*** to output image ***y***

G: *z* → *y*

- Conditional GANs learn a mapping from an observed input image ***x*** and random noise vector ***z*** to produce output image ***y***

G: { *x*, *z* } → *y*

- Unlike traditional GANs in cGANs the discriminator also observes the input image during Fake/Not Fake decision



Objective Function

- The objective function of a conditional GAN can be expressed as

$$\mathcal{L}_{cGAN}(G,D) = \mathbb{E}_{x,y}[\log D(x,y)] + \mathbb{E}_{x,z}[\log(1 - D(x, G(x,z))],$$

  Here, G tries to minimize the objective function while adversarial D tries to maximize it

$$\mathbf{G^* = \min_G \max_D L_{cGAN}(G,D)}$$

- Since the task of the Generator is not only to fool the Discriminator but also to be near the ground truth, a L1 loss is also added

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z}[\|y - G(x,z)\|_1].$$

- Also the L1 loss ensures correctness in low frequencies

- Hence the final objective function is

$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G,D) + \lambda \mathcal{L}_{L1}(G).$$

- In the final model the noise is provided in the form of dropouts applied on several layers of the Generator at both training and testing time

Network Architecture

- Like any GAN the Conditional GAN consists primarily of two parts
    - Generator
    - Discriminator

Generator

- Traditionally an Encoder-Decoder model is used for Generators. However, for many image translation problems there is a great deal of low level information shared between the input and the output
- Hence it is desirable to shuttle this information directly across the network
- Therefore, the paper uses a U-Net

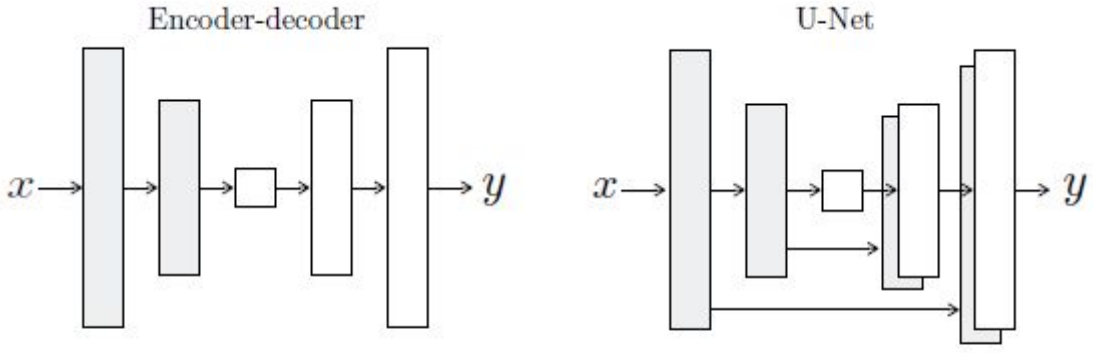
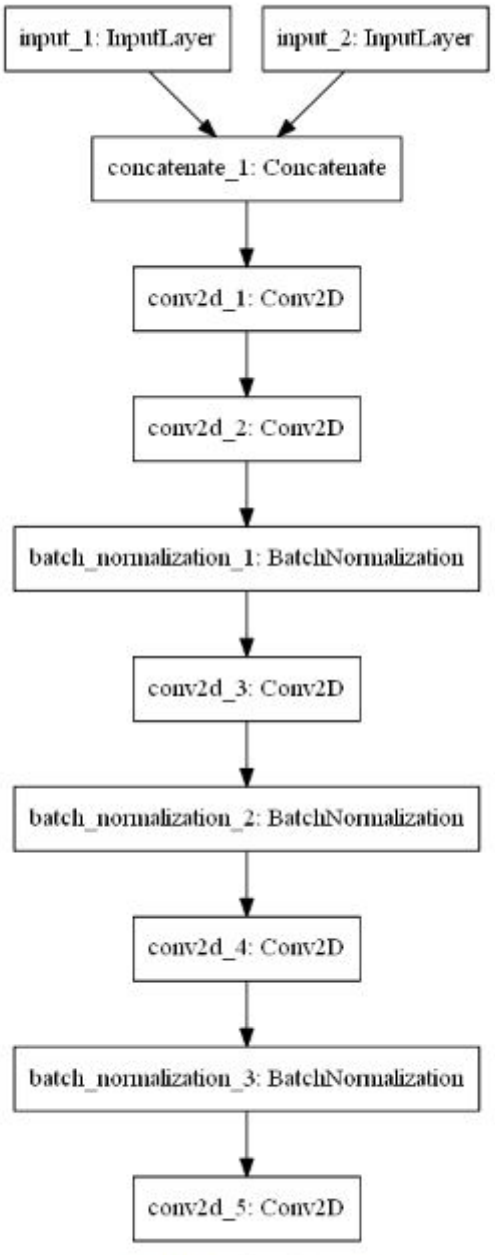


Figure: Two choices for the architecture of the generator.

- The “U-Net” is an encoder-decoder with skip connections between mirrored layers in the encoder and decoder stacks.

Discriminator

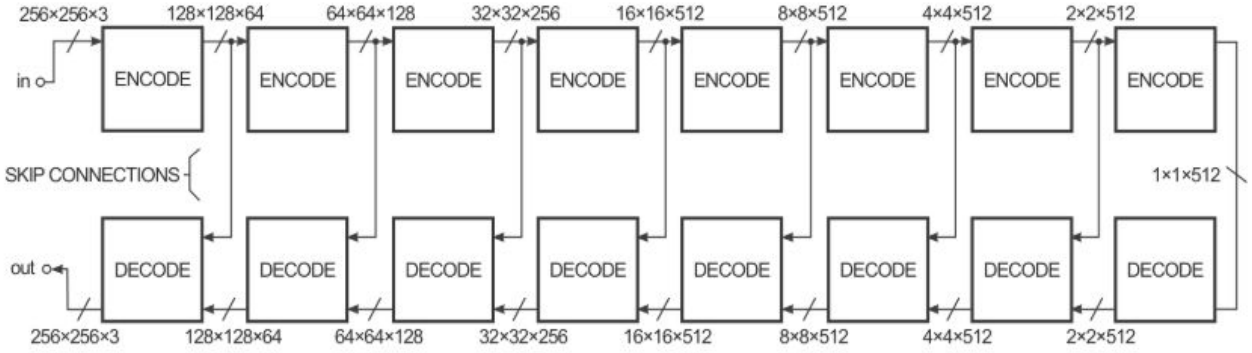
- The L1 loss introduced in the objective function accurately captures the low frequencies (low pixel intensity variation) in the output image
- Therefore, the work of the Discriminator is limited to model the high-frequency structure



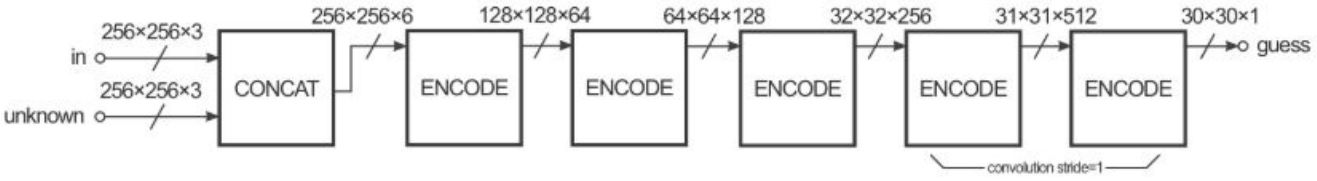
Discriminator's Architecture

Block Diagram of Generator and Discriminator

- Following is the block diagram of the generator



- Following is the block diagram of the discriminator



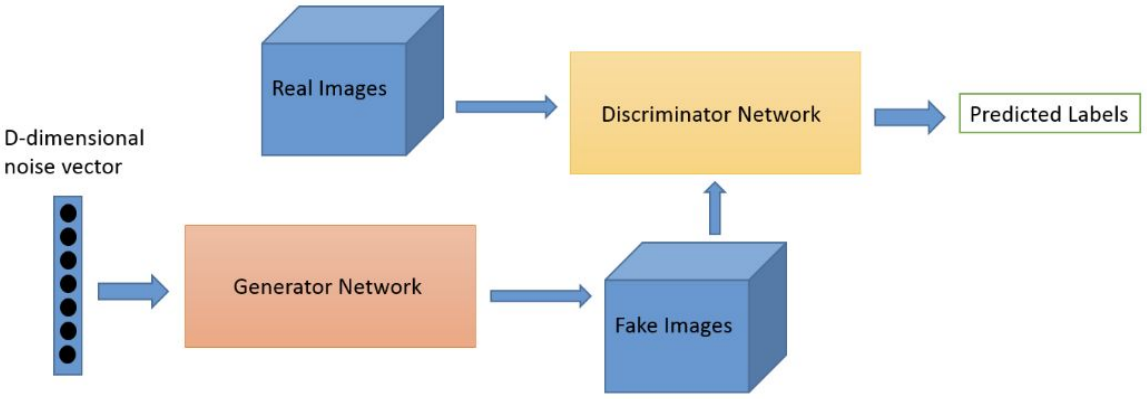
Understanding GANs:

A GAN has 4 components:

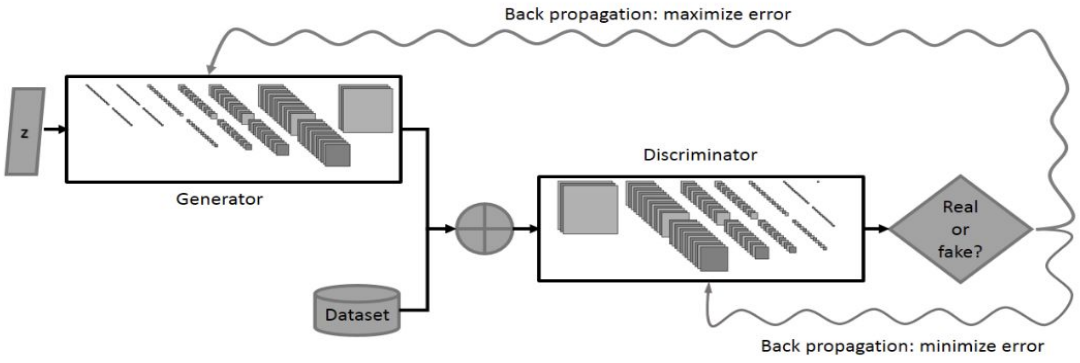
- Generative Network: In traditional GANs the generative model learns a mapping from random noise vector ***z*** to output image ***y***

$$G: z \rightarrow y$$

- Discriminator Network: The goal of the discriminator network is to distinguish real images from the fake images (provided by the generator)



- Dataset: This contains the real image dataset used to train the Discriminator network
- Random Noise: The random noise that goes into the generator as a source of entropy



Objective Function:

- The following is the objective function for the network

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data x}} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data G(z)}}) \right]$$

Objective Function

- Since a game-theoretic approach is taken, our objective function is represented as a minimax function
- The discriminator tries to maximize the objective function, therefore we can perform gradient ascent on the objective function

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Gradient Ascent on Discriminator

- The generator tries to minimize the objective function, therefore we can perform gradient descent on the objective function

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

Gradient Descent on Generator

- By alternating between gradient ascent and descent, the network can be trained

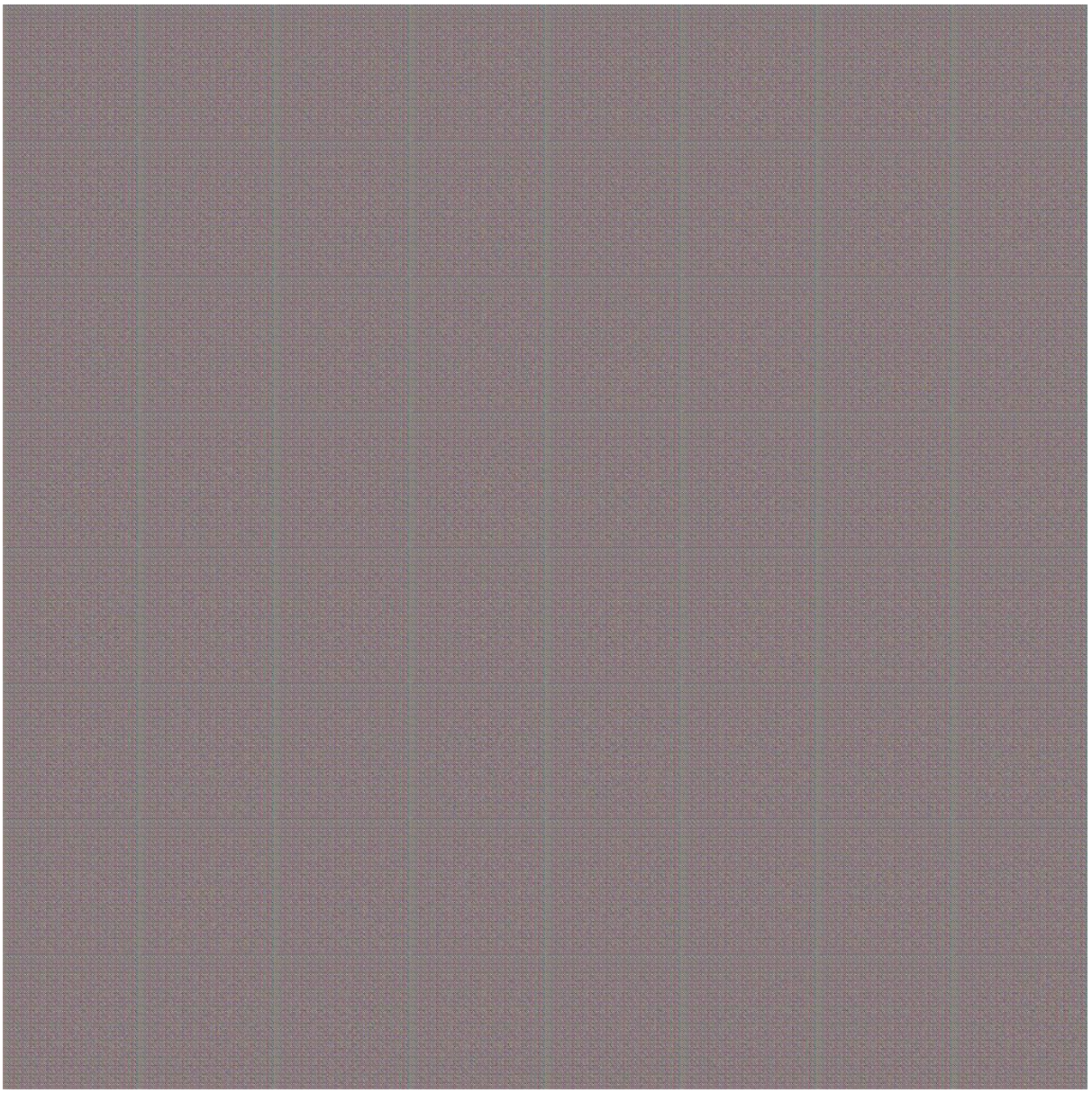
Pokemon Generation using GAN:

- We used Tensorflow to create a GAN to create new pokemons
- The real pokemon dataset contains around 100 pokemons

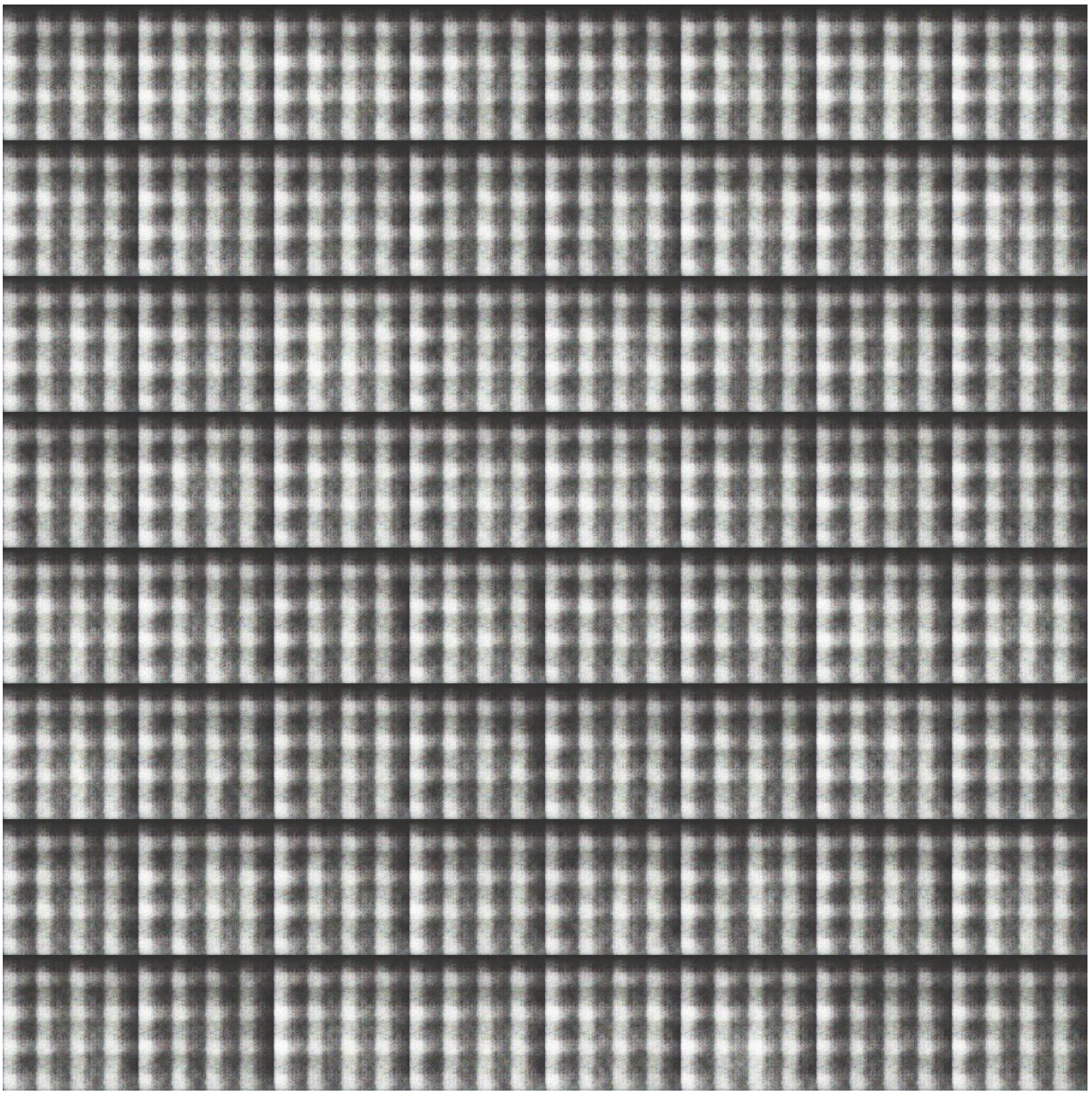
- The dataset was preprocessed to resize the images to 128x128x3 size
- The Discriminator is trained more than the Generator to obtain better results

Results for Pokemon Generation:

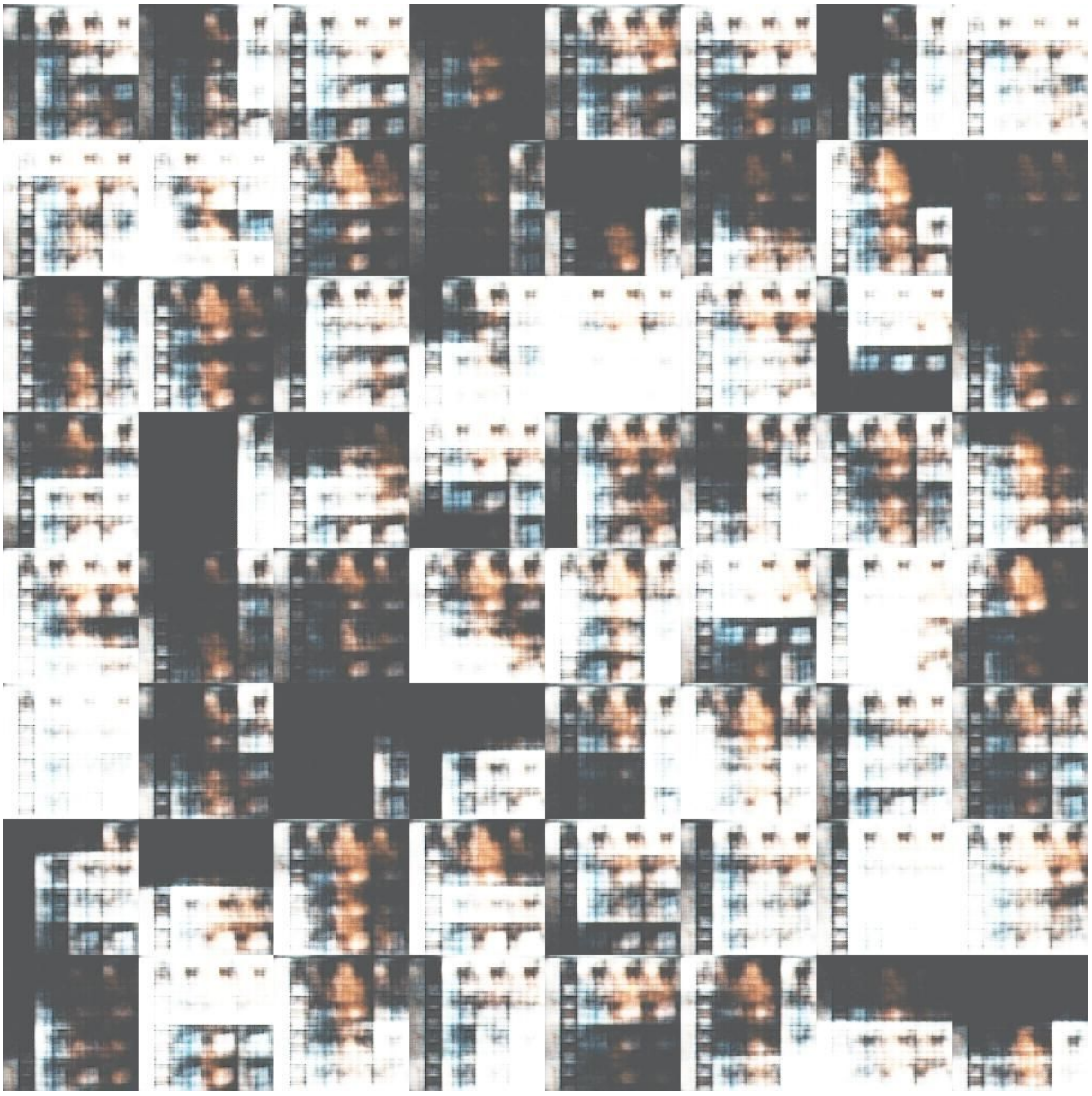
- **After 1 epoc**



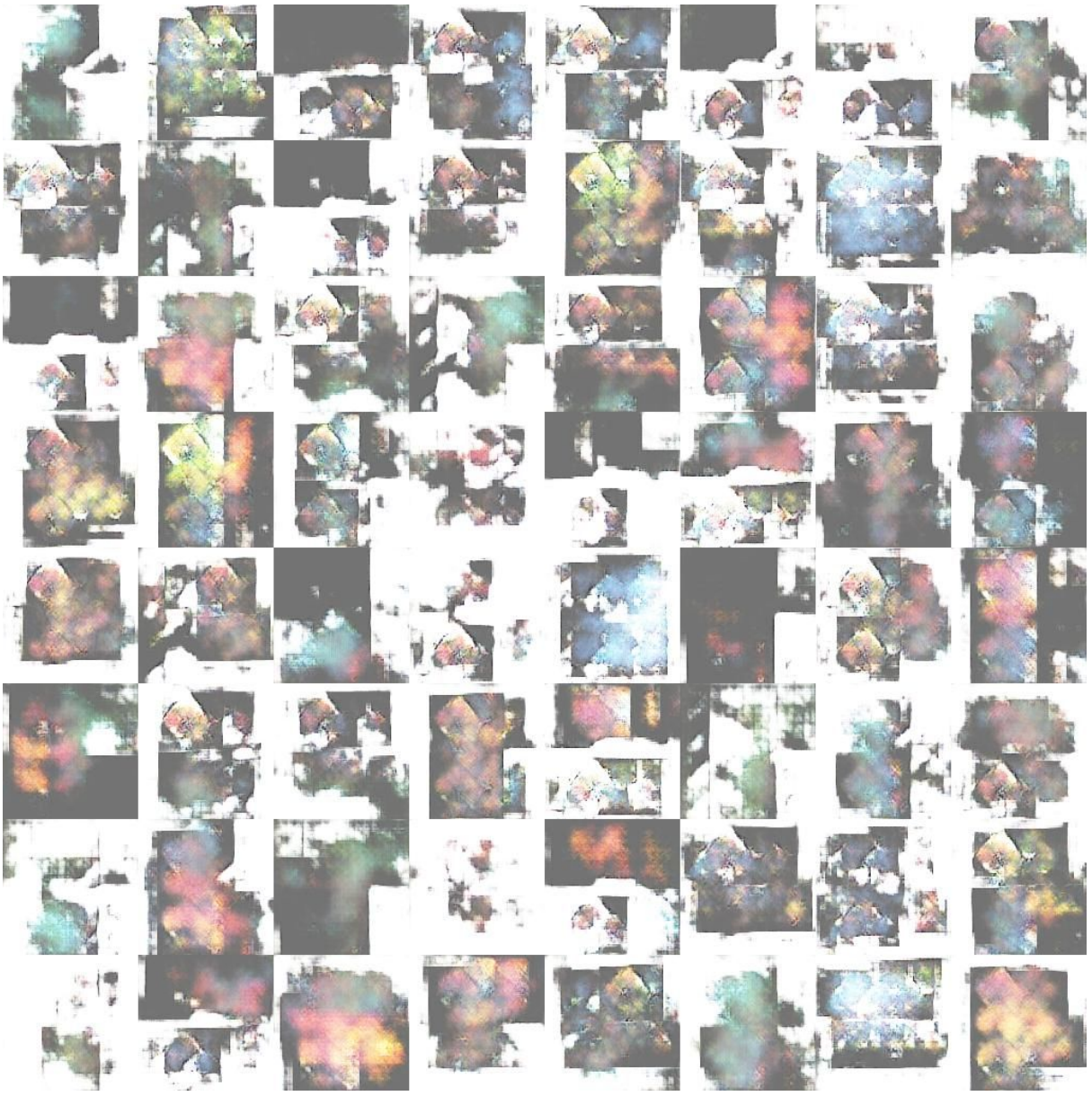
- **After 100**



- **After 500**



- **After 2500**

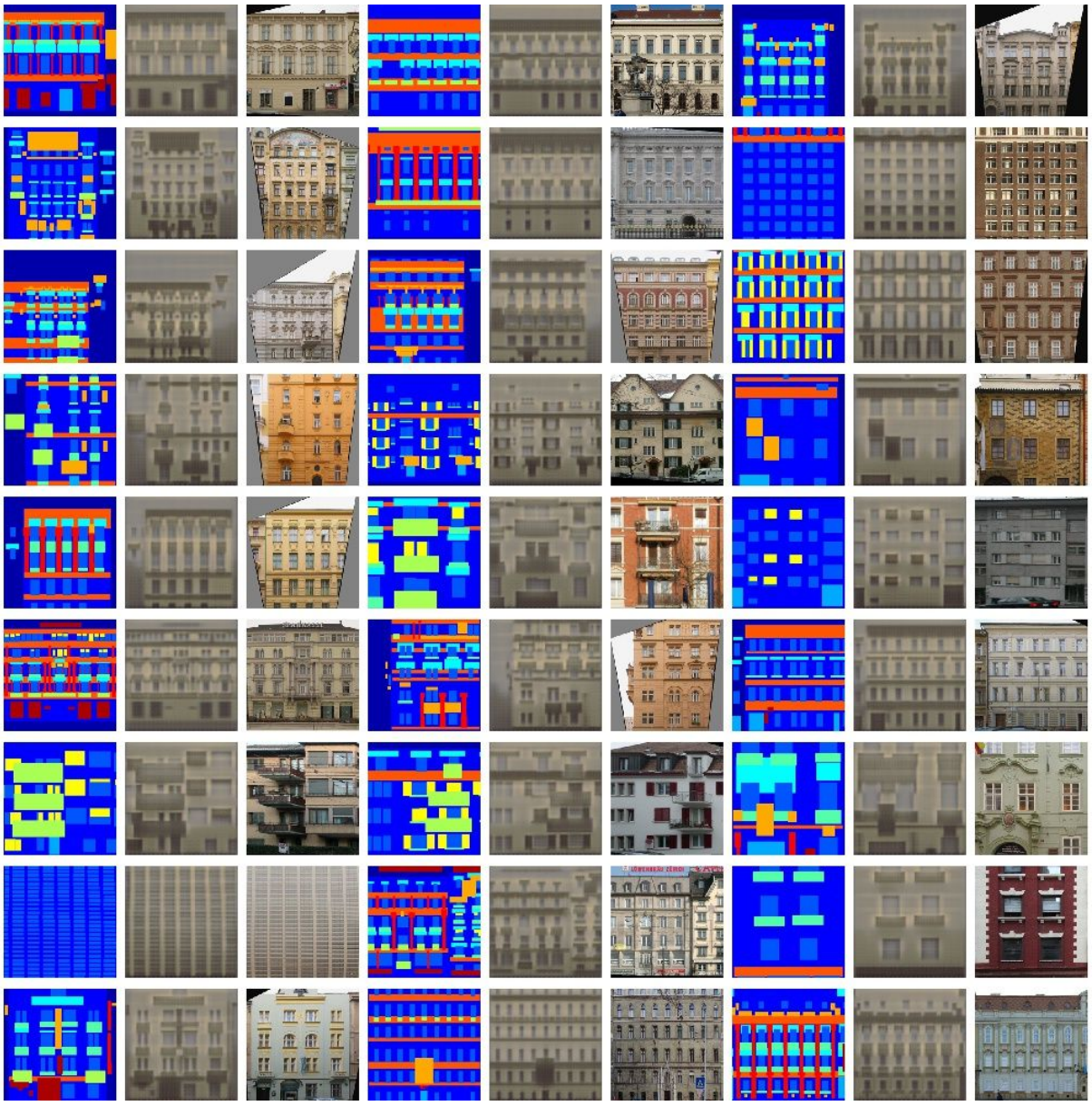


- **Final results**

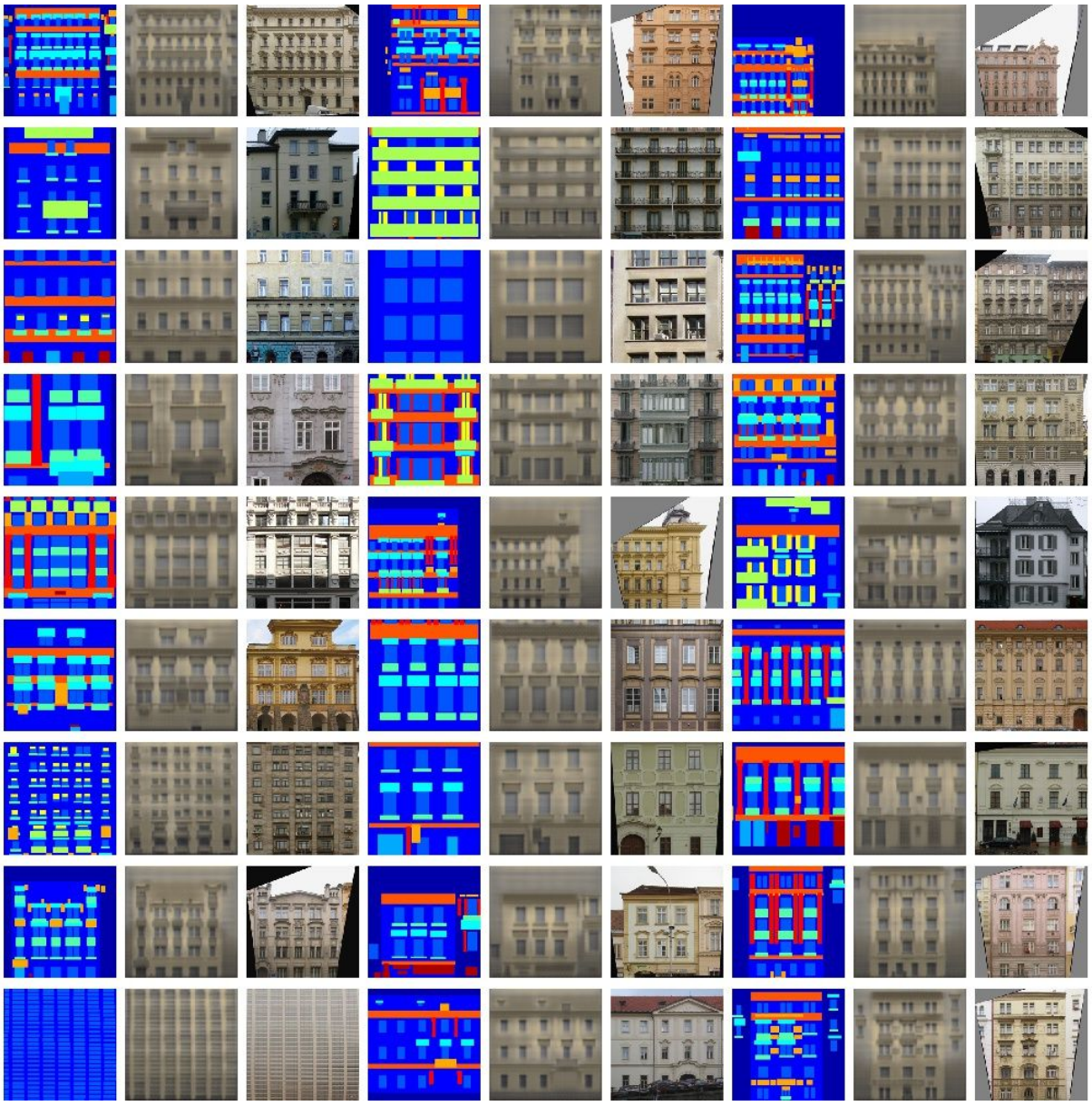
Results for Image-to-image translation:

- **Iteration 0**

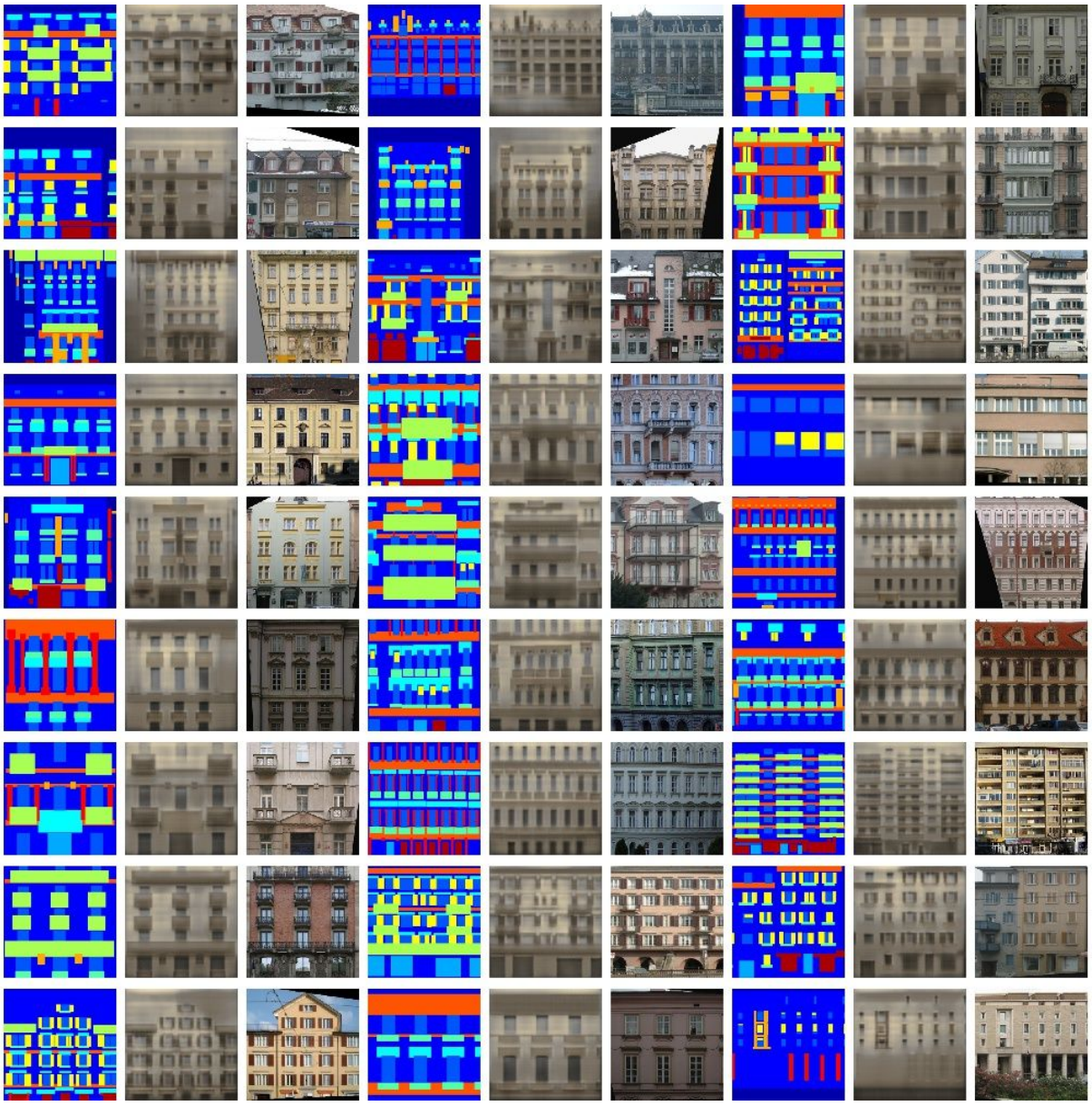
- **Iteration 500**



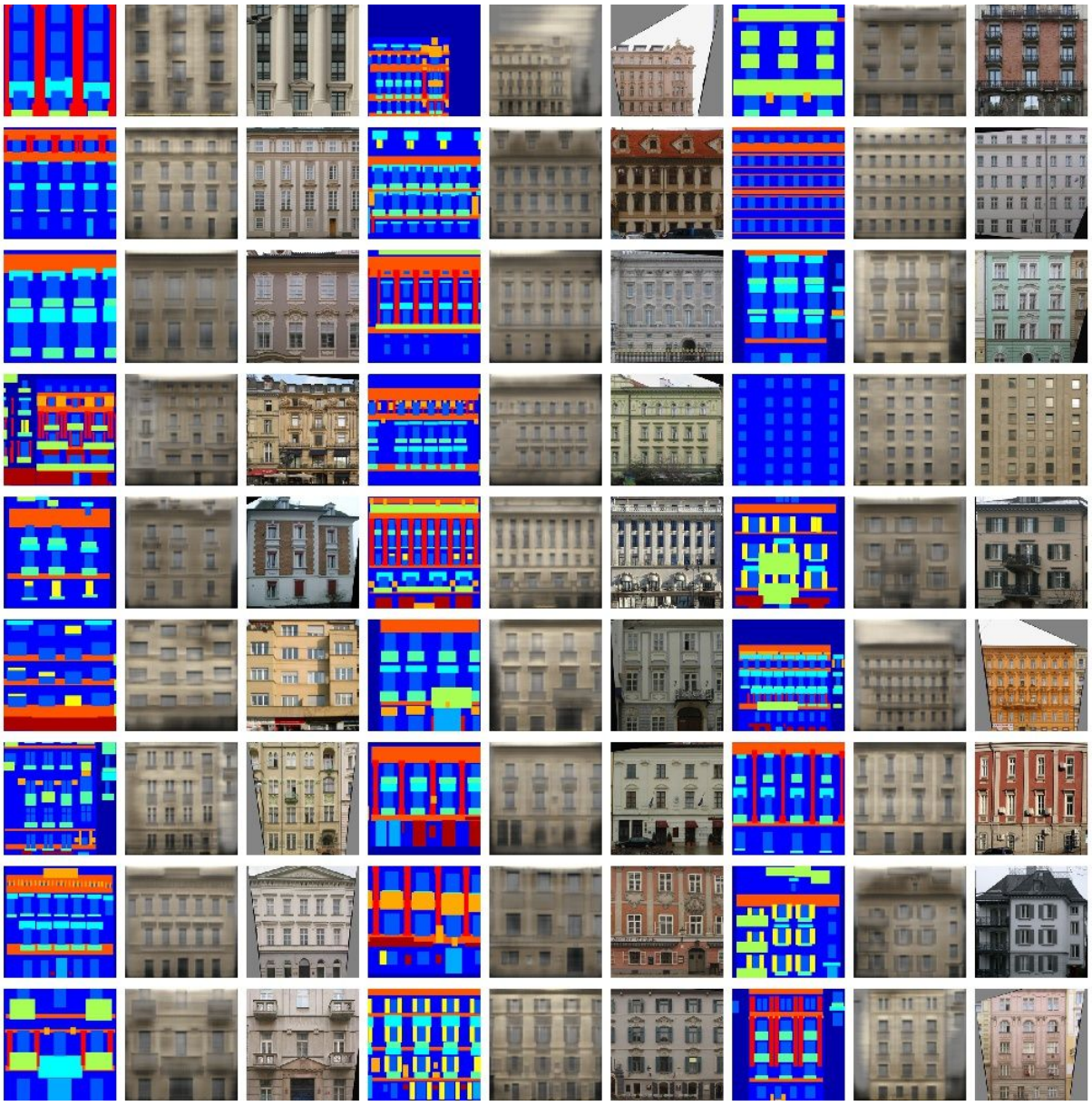
- **Iteration 1000**:



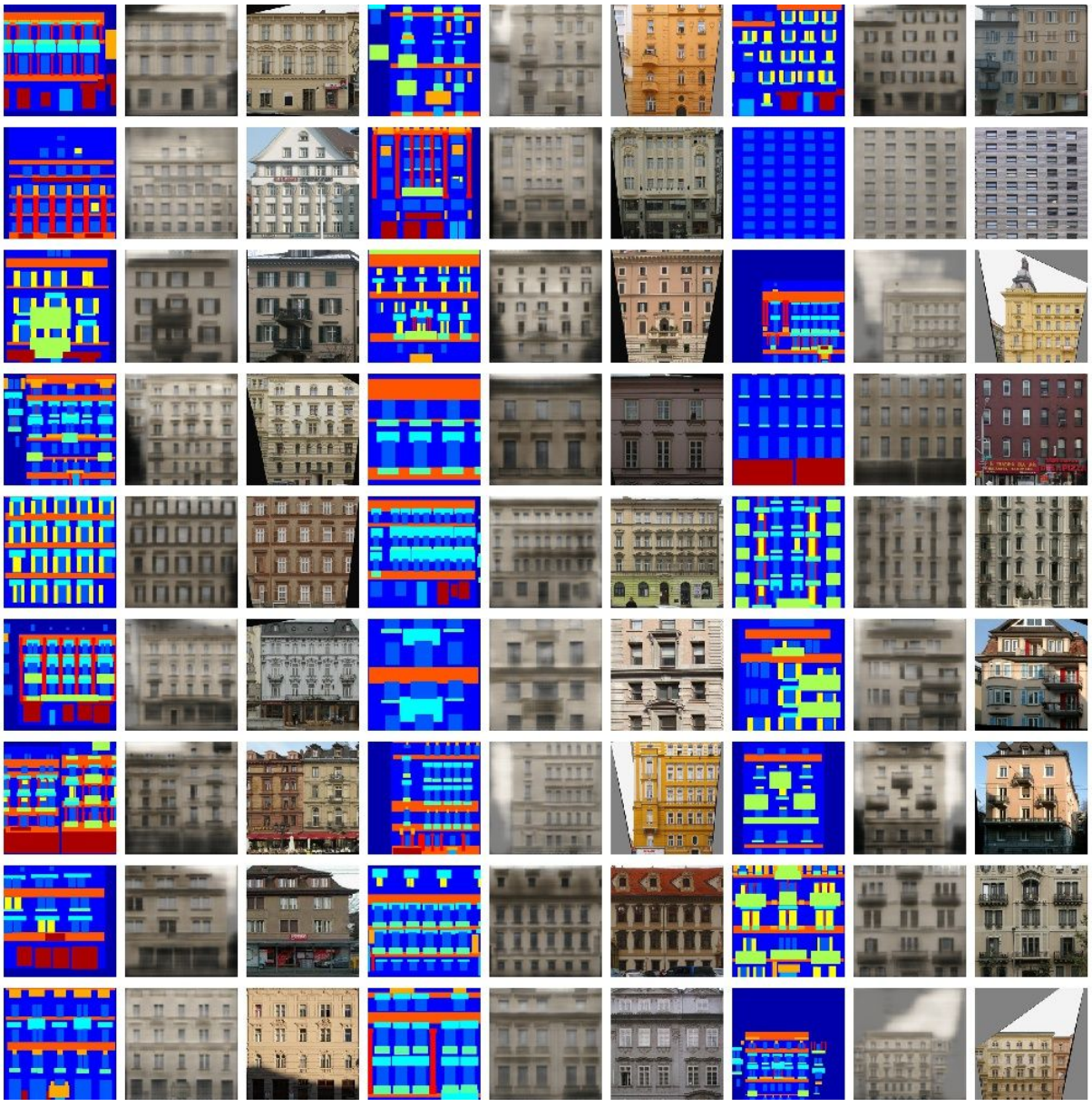
- **Iteration 2000:**



- **Iteration 4000:**



- **Iteration 6000:**



- **Iteration 8000:**

- **Iteration 10000:**