Image-to-Image Translation with a Conditional GAN
Creative AI
Final Project

Lex Janssens (s2989344)
Maksim Terentev (s2565137)

December 23, 2022

Project aims

In this project, we are implementing a drawing system to make facades. We use a conditional Generative Adversarial Network (cGAN) called pix2pix. This model trains in such a way that it creates mappings from the input image to the output image and improves this mapping method whilst training. In this case, the input images are facades consisting of rectangles. Each rectangle has a color that represents a different part of a facade. This could be a door, a wall, a balcony, etc. The output images are the real facades the drawings are based on. What we are doing in this program is making the drawing tool to let you use the trained model and create your own facades utilizing a variety of rectangles. This supports the availability of the creation of facades and the availability use of a trained AI model. People like architects or level designers in games could use such a system to help them design facades in an easy and accessible manner. The creative aspect of this system is thus that there are no limitations to whatever you can create with this system, just the arrangement of rectangles on your canvas. Therefore, it helps creativity through accessibility and ease of use.

Background research

Implementing a completely new system might be a very complicated and comprehensive task requiring significant time and effort. Hence, building a cGAN model on an existing system is wiser. Our project is based on the pix2pix project, where image-to-image translation is introduced using conditional Generative Adversarial Networks (cGANs). GANs allow us to generate novel data (e.g., images) using deep learning methods such as convolutional neural networks (CNNs). Generative adversarial models are trained through unsupervised learning, meaning they are expected to discover and learn patterns in the input data to generate new and novel examples, thus establishing a mapping from input to output. However, unsupervised learning is rather an inconvenient process as it is rather difficult to check the model performance. This problem has been solved in GANs, as the model is split into two sub-models trained through supervised learning: the

generator, which is trained to produce new examples, and the discriminator, which is trained to determine whether an example is real (from the dataset) or fake (generated by the generator). Those two models are trained together in a so-called zero-sum principle (and, thus, adversarial networks): the generator tries to produce an example to fool the discriminator, and the discriminator tries to catch the fake example generated by the generator (Brownlee, 2019). The conditional version of GAN is characterized by feeding the data Y we wish to condition on to both the generator and discriminator.
In the pix2pix project, the cGAN model can be trained on different datasets such as `facades`, `cityscapes`, `edges2shoes`, etc. For our project, we will only work with the `facades` dataset. Every element in the `facades` dataset consists of two images: the real facade (ground truth) and the input image, representing the corresponding real facade as can be seen in figure 1. Each such input image consists of several rectangles of different colours. Each color reference another object, e.g., blue means an entrance, and red means a door.

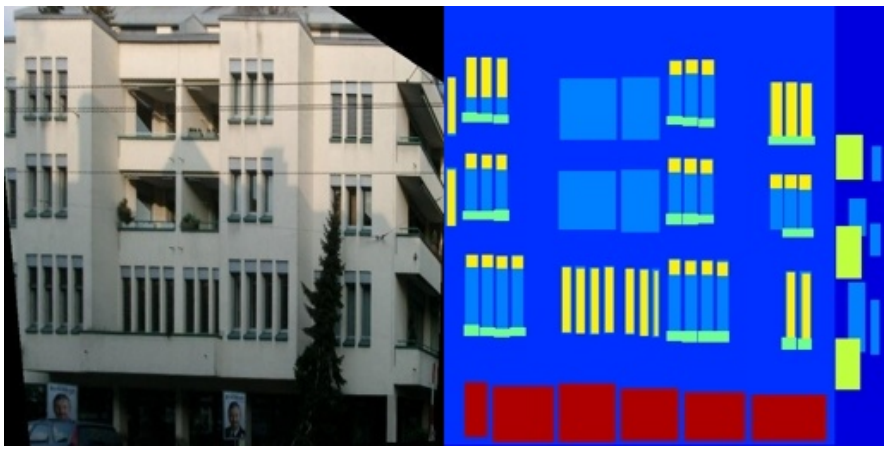


Figure 1: An element from the dataset

When the model is trained, the generator is able to generate (or predict) a new and novel image based on the input image from the dataset. Here comes the concern, as the cGAN model is being evaluated by feeding the input images from the data set that has been used to train this exact model, you cannot check whether this works for other samples. This is what we will be changing in our project. Namely, we will create a drawing tool where a facade can be assembled and fed into the generator to see how it performs. Our drawing tool is inspired by the work of Christopher Hesse called Image-to-Image Demo (see below). The demo consists of two windows. The first window is a scratch pad (input), where a sketch of, let's say, a cat can be drawn. The second window shows the output – a generated image from the input. This demo can be seen in figure 2.
There is a number of useful resources which can be used to explore the cGAN model further:

1. Image-to-Image Translation with Conditional Adversarial Nets contains all necessary information about the pix2pix project, links to the GitHub repositories, the code for both PyTorch and TensorFlow, the research paper, an interactive demo, and much more.

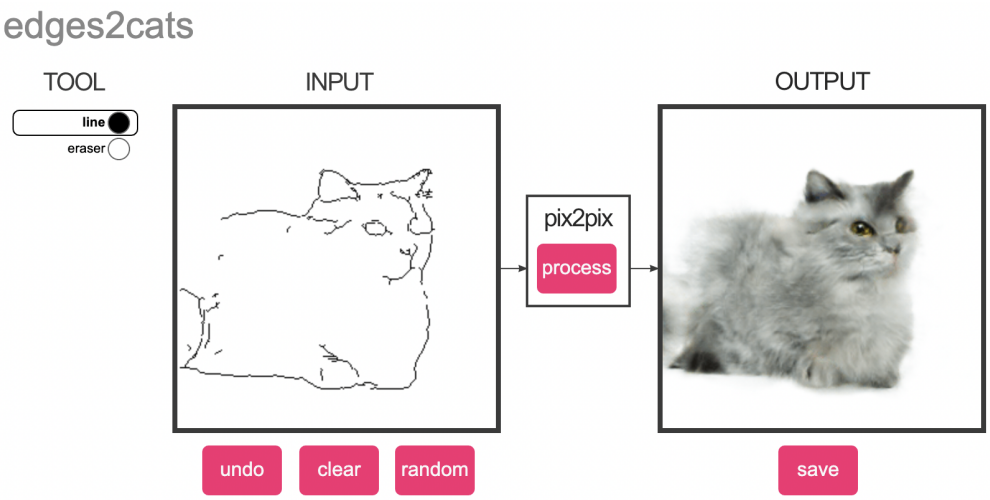


Figure 2: Image-to-Image Demo

2. The TensorFlow GitHub repository of the pix2pix project.
3. Different datasets for the cGAN model from pix2pix developers.
4. The interactive Image-to-Image Demo by Christopher Hesse to explore the cGAN model.

The pix2pix project is based on the research paper "Image-to-Image Translation with Conditional Adversarial Networks" by Isola et al., 2018, where cGANs are being investigated as a general-purpose solution to the image-to-image translation problem.

Implementation

Our project is an example of a proof-of-concept research, where we evaluate the cGAN model on the custom images created in the drawing tool. We can prove that the model performs as intended by feeding the trained model with the novel data and checking whether the provided input is mapped (or transformed) into the output (generated image) correctly.
As mentioned in the previous section, our cGAN model is based on the pix2pix TensorFlow project. For the purposes of our project, we had to adjust or add some code. However, this did not change the default model but added some custom functionality, such as generating an image from a custom input. The drawing tool is created ourselves and only inspired by the demo.
In order to make an interactive system for the model, we In order to make an interactive system for the model, we created a GUI using the Pygame library. In this GUI, you can draw samples, which will automatically be transformed utilizing the model into an output image. In this system, you are able to use up to twelve colors, each indicating another part of the image. These can be, for instance, doors, walls, balconies, pillars, etc. While drawing on the left board, you will notice that your drawing automatically converts to the right board.

For ease of use, we also implemented undo and redo functions. Furthermore, we implemented save and import functions, so you can continue with the drawing you are working on. If you save a drawing, a cache file will be created, which contains the history of your drawing, and allows you to undo parts later on if you decide to import and work on it. Along with this cache file, it also saves an image of your drawing and an image of the model's output. This file should remain untouched. Speaking of untouched files, the files generated by the program once you start it should also remain untouched and will be deleted automatically once closing the program. Additionally, we implemented a clear button that clears the canvas and history. Furthermore, a random button allows you to start on a random sample input from the data set as a basis to continue. Lastly, by clicking on the button in the bottom left corner of the output (or the output image itself), the canvas shows your drawing over the picture such that you can see what part of your drawing is responsible for the given output by the model. It also tells you what object your mouse is hovering over. The toolbar can be seen in figure 3. Our GitHub contains the code and instructions on how to use it. This does not include the checkpoint of the pre-trained model. This is, however, included in the submission.

Figure 3: Toolbar

Example of output

As mentioned, the program has a save function. The save function saves three files: your drawing as 256x256 jpg file, the output of the model as 256x256 jpg and a cache file, which contains the history of your drawing.
We trained the model first for 40k steps. One of the output examples can be seen in figures 4, 5 and 6.

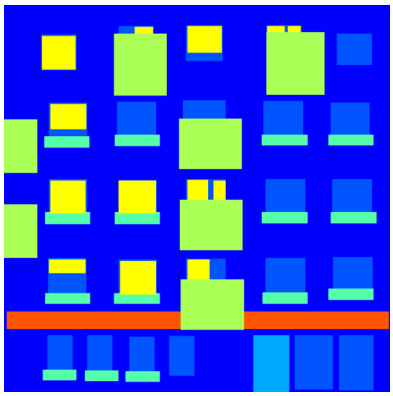


Figure 4: Input canvas drawing

Figure 5: Output from the model trained on 40K epochs

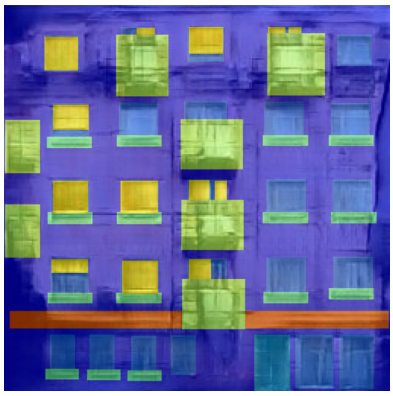


Figure 6: Output with drawing overlay

We let the model train for another 40k steps, thus in total 80k steps, to improve the model as well as slightly increasing the resolution. One of the output examples can be seen in figures 7, 8 and 9.

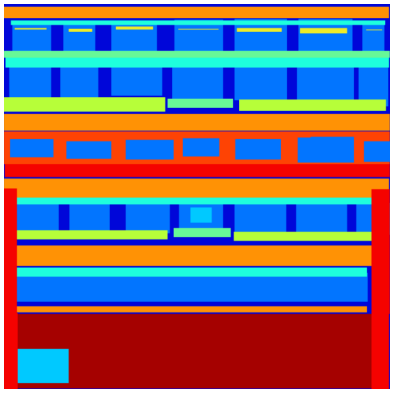


Figure 7: Input canvas drawing

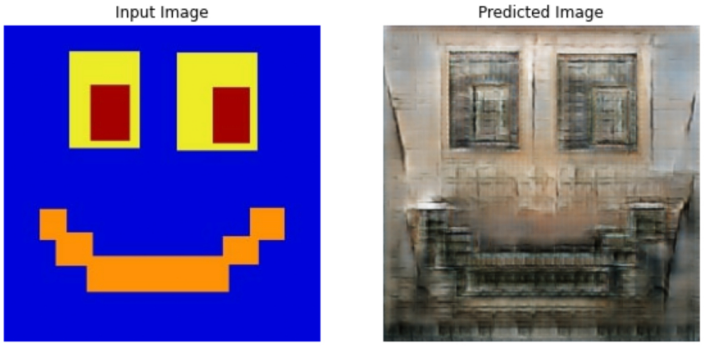
Figure 8: Output from the model trained on 80K epochs

Figure 9: Output with drawing overlay

You can see a good distinction between the model trained with 40K steps and the model trained with 80K steps. The output image of the 40K model, seen in figure 5, has various vague spots where you cannot distinguish any particular feature at some spot. The 80K model distinguishes these features better, and fewer vague spots can be found in that output image, seen in figure 8.

Reflection and future work

The work on this project went very well! Everything that had been planned to be implemented has been implemented on time. There was confusion at the beginning of the project, as it was unclear where to start and what to implement. However, the direction was chosen after some discussion, and the work began. All in all, we are happy with the result.
To conclude, this is a very successful project as it may play an important role in Creative AI research. The created drawing tool may become an excellent opportunity for the cGAN researchers to evaluate their model as well as to create custom datasets using our drawing tool. In the future, the drawing tool can be extended to work with different datasets.



References

Brownlee, J. (2019). *A Gentle Introduction to Generative Adversarial Networks (GANs).* Retrieved December 23, 2022, from https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/

Isola, P., Zhu, J.-Y., Zhou, T., & Efros, A. A. (2018). Image-to-image translation with conditional adversarial networks.