INF573
Sketch to Image Project

Alisa Kugusheva and Margarita Konnova

December 2021

1 Introduction

1.1 Problem

In this project we pose two problems: 1) creation a sketch of an object using an image of the object and the corresponding binary (background/object) segmentation mask, 2) automatic image-to-image translation. In general, image-to-image translation represents the task of translating one possible representation of a scene into another, given sufficient training data. We take this task in a form of creating a realistic image from the sketch obtained solving the first posed problem.

Sketch creation pipeline is described in the section 3.2. For the skecth creation we used one of the most popular edge detection algorithms – Canny edge detector [1] which uses a multi-stage algorithm to detect a wide range of edges in images. It is composed of 5 steps:

1. **Noise reduction.** Gaussian blur can be applied to smooth the image. This step is important because the algorithm is sensitive to the image noise.

2. **Gradient calculation.** Basically, it calculates the gradient of the image to detect the edge intensity and direction using edge detection

operators (Sobel filters, Prewitt operators, etc.).

3. **Non-maximum suppression.** The step is required to thin out the edges by going through all the points on the gradient intensity matrix and finding the pixels with the maximum value in the edge directions.

4. **Double threshold.** At this step the **strong** (ones that have an intensity so high that we can be sure they contribute to the final edge) and **weak** (ones that have an intensity value which is not small enough to be considered as non-relevant for the edge detection, but at the same time not enough to be considered as strong ones) pixels are identified.

5. **Edge Tracking by Hysteresis.** During this step the weak pixels can be converted into strong ones, if and only if at least one of the pixels around the one being processed is a strong one.

To create a realistic image form sketch we decided to use Generative Aversarial Networks (GANs). GANs are state-of-the-art neural network architecture in image generation tasks. The network consists of two different neural networks named generator and discriminator. In a classical approach the discriminator learns to classify between fake (synthesized by the generator) and the real image. The generator generate images and learns to fool the discriminator, so it can't say if the fake image is fake. The details are discusssed in the section 3.1.

Not only it is considered a really complicated and time-consuming task to experiment with GANs architecture to solve a specific problem and to train it from scratch, but it also requires a large computational power that we, unfortunately, don't have. Due to these reasons we decided to use an existing architecture that was created to solve similar tasks and a pre-trained model. We provided a research and found [3] where the authors implemented different tasks of image-to-image translation with Conditional Adversarial Networks and obtained a suitable architecture for

them. Then we searched for the implementation of the paper and found this GitHub with the PyTorch implementation of the needed architecture and pre-trained models of them. The task similar to ours is to make an image of a bag from sketch and to make an image of shoes. We used both of these pre-trained models to finetune them for our datasets and they seemed to give similar results.

1.2 Datasets

We used 2 different datasets to implement sketch to image creation. The first dataset consists of 5088 rectangular images of cars of size 1918×1280 and their binary segmentation masks (background/object). The second dataset consists of 7390 images of different sizes (between 333 and 500 for both vertical and horizontal directions) of cats and dogs of different breeds and their corresponding segmentation masks consisted of three labels: background, object and object boundary.

2 Related work

Generative Adversarial Networks were first introduced in 2014 in [2]. The concept is that we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake.

In contradiction to Convolutional Neural Networks (CNNs) where we have to specify what loss the network should minimise, GANs learn a loss that tries to classify if the output image is real or fake, while simultaneously training a generative model to minimize this loss. The motivation is simple: since we want to get sharp and realistic images, we can't just use Euclidean distance as a loss function, because Euclidean distance is minimized by averaging all plausible outputs, which causes blurry generated images, that look fake. But the problem of finding the exact loss

function that would make a model generate realistic images is really hard and requires an expert knowledge and taking to account specific characteristics of each dataset and task we are working on. Because GANs learn a loss that adapts to the data, they can be applied to a multitude of tasks that traditionally would require very different kinds of loss functions.

Conditional Generative Adversarial Nets (cGANs) were introduces in the same year in [4]. In the image-to-image translation tasks (such as sketch to image creation) we need to learn the mapping from input image to output image. Conditional GANs learn a mapping from observed image and random noise vector. This makes cGANs suitable for image-to-image translation tasks, where we condition on an input image and generate a corresponding output image.

In the paper [3] the authors used cGANs for different image-to-image translation tasks:

- semantic labels $\leftrightarrow$ photo,
- architectural labels $\rightarrow$ photo,
- map $\leftrightarrow$ aerial photo,
- BW $\rightarrow$ color photos,
- edges $\rightarrow$ photo,
- sketch $\rightarrow$ photo,
- day $\rightarrow$ night,
- thermal $\rightarrow$ color photos,
- photo with missing pixels $\rightarrow$ inpainted photo.

We were inspired by this work and made our project based on it.

3 Method

3.1 cGAN loss function

GANs are generative models that learn a mapping from random noise vector z to output image y, $G: z \rightarrow y$. In contrast, conditional GANs learn a mapping from observed image x and random noise vector z, to y, $G: x, z \rightarrow y$. The objective of a conditional GAN can be expressed as:

$$L_{cGAN}(G,D) = E_{x,y}[logD(x,y)] + E_{x,z}[log(1 - D(x,G(x,z))] \quad (1)$$

where G tries to minimize this objective against an adversarial D that tries to maximize it, i.e.

$$G^* = arg\ min_G\ max_D\ L_{cGAN}(G,D) \quad (2)$$

It was showed in some studies that it is beneficial to mix the GAN objective with a more traditional loss, such as $L2$ distance. The authors of the original image-to-image translation paper use $L1$ distance to avoid blurring that can be caused by $L2$ distance.

$$L_{L1}(G) = E_{x,y,z}[||yG(x,z)||_1] \quad (3)$$

The final objective is:

$$G^* = arg\ min_G\ max_D\ L_{cGAN}(G,D) + \lambda L_{L1}(G) \quad (4)$$

3.2 cGAN architecture

Both generator and discriminator use modules of the form convolution-BatchNorm-ReLu. A lot of works in image-to-image translation problem have used an encoder-decoder network. The concept is that an image is downsampled in the decoder that has n layers with decreasing number of parameters. The encoder has n correspondig layers with increasing number of parameters. The image in the decoder upsamples to the initial size. So, the encoder learns to make a small version of an input image, that will preserve all the important information of an image enough to be

reconstructed, and the decoder learns to reconstruct an image to the initial size. The problem with such a network is that important information can be lost in the encoder part. To overcome this problem it is better to use UNet-like architecture instead of encoder-decoder one. The U-Net was proposed in 2015 in [5] as a CNN for biomedical image segmentation but was used helpful in a lot of different tasks. The architecture consists of two paths: a contracting path and an expanding path which is symmetric to the contracting path, so it yields a U-shaped architecture, so it is quite similar to the autoencoder. The difference is that the U-Net architecture uses skip connections to combine low-level feature maps with higher-level ones to avoid the loss of higher-level information. Specifically, it uses skip-connection between each i-th and $n-i$-th layer. In this project we used UNet architecture as generator.

As it was already mentioned, general losses such as $L1$ or $L2$ produce blurry results on image generation problems. That means that using them we can not produce sharp, realistic results, but only low-level general information about image. So, it's a good way to use them for producing low frequencies. This motivates restricting the GAN discriminator to only model high-frequency structure. In order to model high-frequencies, it is sufficient to restrict our attention to the structure in local image patches. That's why we use $PatchGAN$ as a discriminator. It only penalizes structure at the scale of patches. This discriminator tries to classify if each $N \times N$ patch in an image is real or fake. The discriminator computes convolutions across the image, averaging all responses to provide the ultimate output of D. In the paper [3] it was shown that using 70×70 PatchGAN for image of size 286×286 is the optimal solution. The authors has experimented with sizes 1×1 (PixelGAN), 16×16 (small PatchGAN), 70×70 (medium PatchGAN) and 286×286 (ImageGAN). It was shown that the PixelGAN produces blurry results that don't look realistic, the 16×16 PatchGAN produces sharp outputs, but leads to tiling artifacts. The 70×70 PatchGAN alleviates these artifacts and achieves slightly better scores. The full 286×286 ImageGAN does not appear to improve the visual quality of the results, and gets a considerably lower metric that the authors evaluated. In our project we used the 70×70 PatchGAN, however

it has appeared to give tiling artifacts.

3.3 Edge detection

The edge detection part consisted of the following steps:

1. Load an image and its corresponding mask.

2. Transform them:

    - Resize both image and mask to a fixed size. For animals we used size of $(400, 350)$ which is close to the original size of the images. For cars we used size $(350, 270)$ since they are wide and short;
    - Make a center crop of the images and masks. As we mentioned above we used UNet architecture for discriminator. Such an architecture is constructed is a way that it is non-flexible to the input image size. We used an implementation with $(256, 256)$ input channels, so the input image should had size that is divided by 256 (256, 512, 1024, and so on). In our project we cropped all the images to the size of $(256, 256)$ to increase computational speed since we had only limited computational resources.

3. Delete background from the image.
   Since we only want to create an object of a particular class from a sketch, we don't need any background information of it. That's why we need segmentation masks and use them at this step. For animals dataset we first binarised the masks to have black background and white object. To delete the background we simply make the pixels white where the corresponding mask pixel label is black.

4. Create a skecth from image.
   For skecthes creation we used Canny edge detector from OpenCV library. Canny edge detection algorithm has 2 thresholds as parameters. We tried to find the optimal thresholds for our datasets: for the

animals dataset they were 110 and 150, and for the cars dataset we used 80 and 120, for threshold 1 and threshold 2 correspondingly.

5. Stack the sketch 3 times among the last axis to have 3 color channels (as the image) and be digestible for the GAN.

6. Stack the image and skecth together as one image file of the size $(512, 256)$.
   This step is necessary for the chosen GAN implementation. It uses the left part of an input image as an example and the right part of an input image as a sketch to generate an image from.

4 Results

As a result of our project we present a macOS application which allows to draw the sketch and see the generated image, a pipeline of Python 3 code in a form of Jupyter Notebook and two models trained for two different tasks. If you have a dataset with segmentation masks you can use it and: 1) get sketches from images, 2) train a sketch to image generating model, or 3) use the pre-trained models to generate images from your sketches (if they are similar to cars or animals).

The idea behind the macOS app is to make models more interactive and easier to test. No steps such as uploading the sketch image or manipulating with the images folders are required. The application consists of the canvas where the sketch can be draw, three buttons and the place where the result generated image is shown. The first button allows to clear all the drawings, the second applies the model and displays the generated image and the third allows to switch between the edges2cars and edges2pets models.

For both datasets we used the model pre-trained to generate handbags to images. Since GANs losses can be unstable from epoch to epoch and are not really representative, we checked the performance by generating an image from a random sketch every epoch.

4.1 Cars dataset

Cars generation appeared to be not a very complicated task. The results were more or less the same after the 10-th epoch of training, so we stopped training at the 16-th epoch and saved the latest model. The results on different images are presented on Fig. 1, 2 in Appendix. The model is quite good at creating black and white cars, but it messes up with the colorful cars (Fig. 3).

4.2 Animals dataset

The animals creation is a much more complicated task due to several reasons. First of all, the dataset consists both of cats and dogs that are quite different from each other, so it's harder for a network to learn to generate them both at the same time. Plus, the breeds of both cats and dogs are different, so the animals are very different even in one class. The second reason in our opinion is that the segmentation masks of the dataset and sometimes images themselves are quite bad and cause troubles in a lot of cases (see Fig. 4, 5). And the third reason is that the obtained edges sometimes are bad, especially in case of dogs with big fur (see Fig. 6).

References

[1] John Canny. “A Computational Approach to Edge Detection”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6 (1986), pp. 679–698. DOI: 10.1109/TPAMI.1986.4767851.

[2] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[3] Phillip Isola et al. “Image-to-Image Translation with Conditional Adversarial Networks”. In: *CoRR* abs/1611.07004 (2016). arXiv: 1611.07004. URL: http://arxiv.org/abs/1611.07004.

[4] Mehdi Mirza and Simon Osindero. “Conditional Generative Adversarial Nets”. In: *CoRR* abs/1411.1784 (2014). arXiv: 1411.1784. URL: http://arxiv.org/abs/1411.1784.

[5] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: *CoRR* abs/1505.04597 (2015). arXiv: 1505.04597. URL: http://arxiv.org/abs/1505.04597.

Appendices

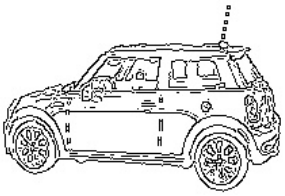
(a) Result of car generation

(b) Sketch

(c) Ground truth

Figure (1) White car

(a) Result of car generation



(b) Sketch

(c) Ground truth

Figure (2) Colorful car

(a) Result of car generation **(b)** Ground truth

Figure (3) Problem with color

Figure (4) Problem with image

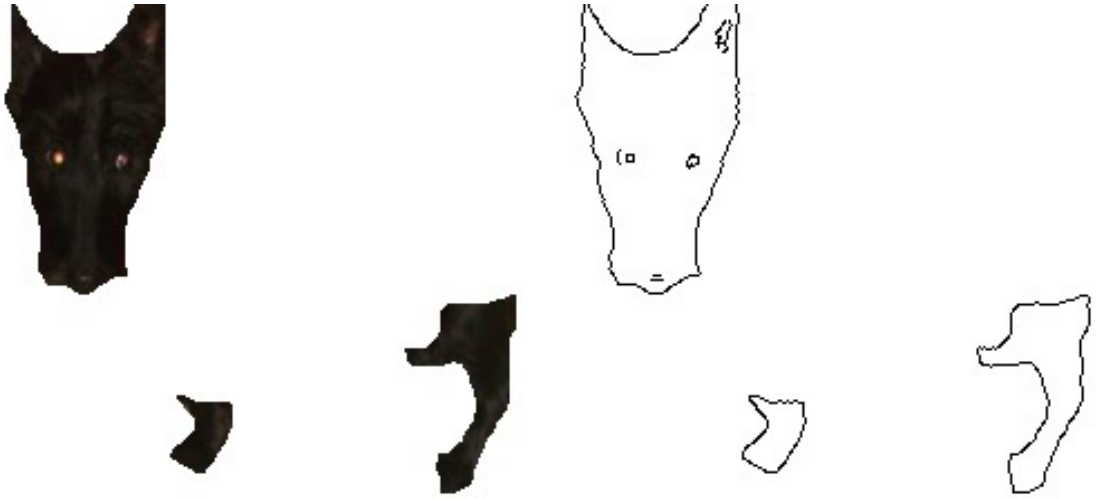


Figure (5) Problem with mask

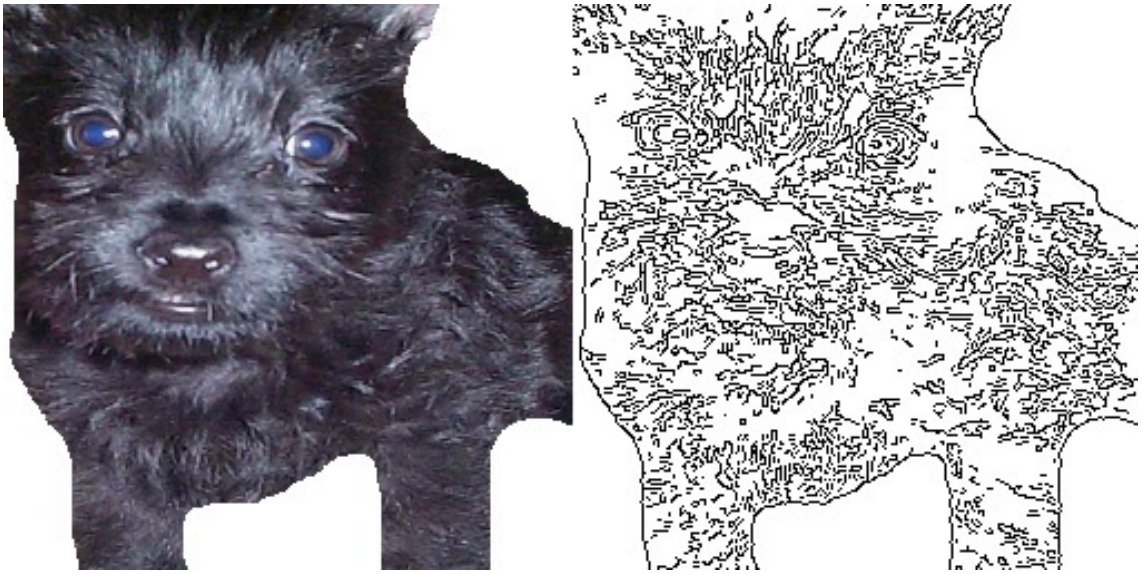


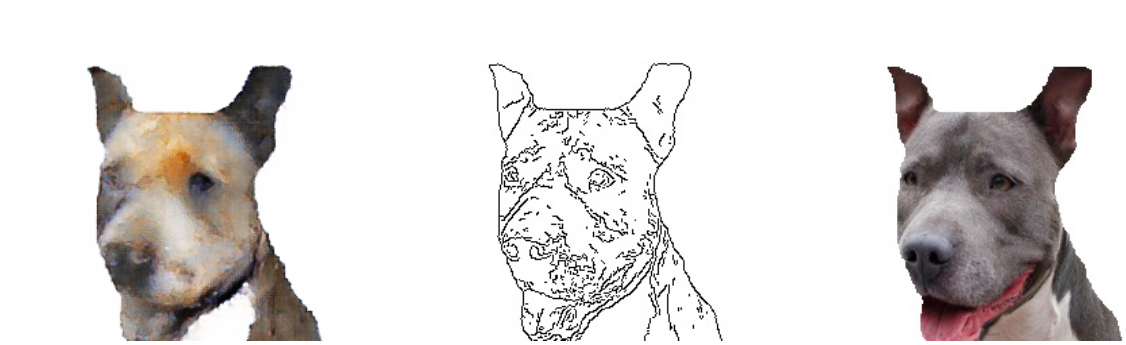
Figure (6) Problem with edges for furry animals

(a) Result of cat generation **(b)** Sketch **(c)** Ground truth

Figure (7) Result of cat generation

(a) Result of cat generation **(b)** Sketch **(c)** Ground truth

Figure (8) Result of dog generation



(a) Result of cat generation **(b)** Sketch **(c)** Ground truth

Figure (9) Another dog generation