Conditioning Generative Adversarial Networks on Feature Representations for Image to Image Translation

Ramaneek Gill
Department of Computer Science
University of Toronto
ramaneek.gill@mail.utoronto.ca

Charles Huang
Department of Computer Science
University of Toronto
cherls.huang@mail.utoronto.ca

Abstract

We explore various conditioning techniques for training conditional generative adversarial networks (cGAN) for the purpose of domain agnostic image to image translation. Generative adversarial networks (GAN) consist of two neural networks, a generator and a discriminator that are trained adversarial method. We attempt to show that we are able to generate image to image translations effectively by conditioning the generator on not only the raw output image but its cheaply computed low level features. We also compare how various computer vision processing techniques can be used for obtaining interesting outputs from a cGAN when conditioning either the generator or discriminator on a low level representation of the ground truth. In particular, we will be considering satellite images from Google Maps as the input and try to generate the non-satellite or 'maps' version of the satellite images. We train our models with data which we will collect using the Google Maps API and visually compare empirical results.

1 Introduction

1.1 Problem

Image to image translation is a problem that has many applications such as reversing popular Instagram or Snapchat filters, colorizing old black and white photographs, and automatically editing photos taken in dark settings to have more light exposure. In our paper we focus on unsupervised translation of satellite imagery to map imagery using GANs and cGANs. This application has widely available data and is constantly in demand of being up to date with the latest construction changes happening in cities.

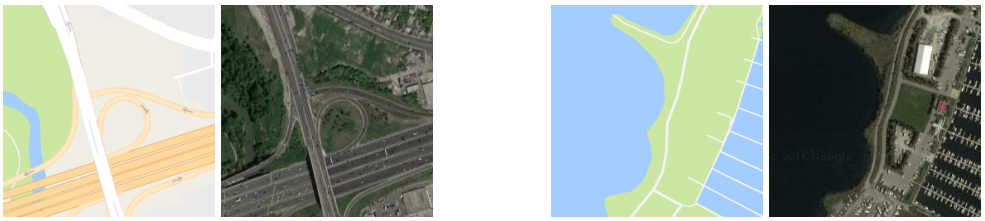

Figure 1: Two pair of example images from our dataset

1.2 Related Works

Domain agnostic image to image translation has recently made great improvements in comparison to the previous state of the art [1] with the use of conditional generative adversarial networks. The rise of GANs for vision related tasks can be accredited to the effective feature representations learned by convolutional neural networks (CNN) [2].

CNNs like most neural networks learn to minimize an objective function. Most use cases to date have been mainly focused on object detection and segmentation [1, 3] instead of image generation. Many problems exist today for generating sharp high quality images using CNNs that have just recently begun to be addressed with the use of GANs because of its highly adaptive nature of modelling the distribution of inputs [4].

Conditional generative adversarial networks so far have been the most successful at generating crisp images at a larger resolution [5, 6]. This is because the discriminator and generator are both conditioned on the input image and therefore both networks that make up the GAN are aware ground truth output. Figure 2 clarifies this structure further.

Object detection and segmentation has made great improvements with the introduction of CNNs, this is partly due to their ability to learn great low level features that are capable of representing high level and noisy images exceptionally well. Examples of this can be seen from using winning convolutional neural nets in the ImageNet competition as default feature extractors for a wide range of computer vision tasks [2]. These low level features learned by convolutional and deconvolutional filters work exceptionally well for generating images through the use of GANs as demonstrated by Xie et al.

GANs consist of two neural networks, a generative network $G(\mathbf{b})$ which maps to the data space and a discriminator network $D(\mathbf{x})$ which outputs the probability that $\mathbf{x}$ came from the data. At a high level, the goal of the generator is try to generate or mimic examples based off the training dataset using some random noise while the goal of the discriminator is try to distinguish whether some input is generated from the generator or a ground truth data sample.

Formally, G and D plays the following adversarial game with value function V

$$\min_G \max_B V(D,G) = \mathbb{E}_{q(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{p(\mathbf{z})}[\log(1 - D(\mathbf{z}))] \tag{1}$$

In this paper we explore the effectiveness of using standard computer vision processing techniques such as canny edge detectors and k-cluster because of their promising results in segmentation and emphasizing the relevant features in the satellite images. We use the results as conditioning inputs for the generator (Figure 2). We explore two more scenarios where interesting results can arise: flipping the conditioning and input image between the generator and discriminator every batch, and conditioning the discriminator on the feature extracted image and conditioning the generator on the input image.

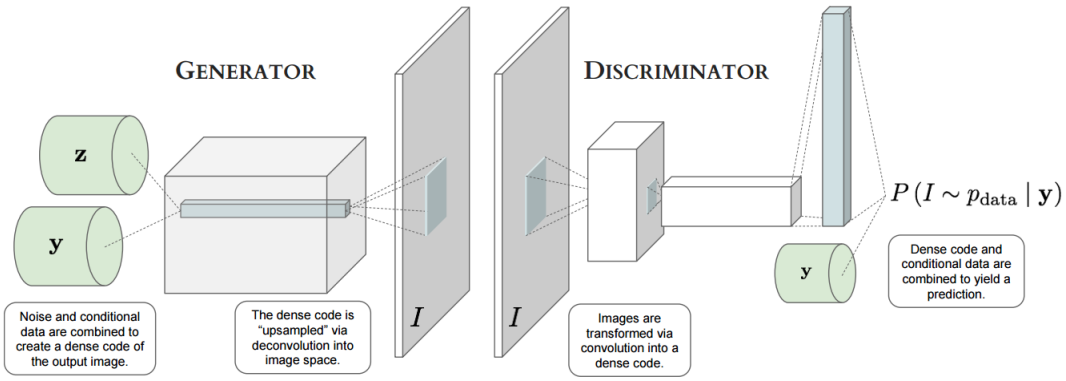

Figure 2: cGAN pipeline with the discriminator conditioned on y [6]

2 Methods

All code has been made public on GitHub at `https://github.com/RamaneekGill/MapGen`

2.1 Data Collection

For our dataset we leveraged Google Maps API to collect 1000 satellite images and their respective map images for a total of 2000 images. The images are sampled randomly from a rectangular region containing Toronto. Figure 3 shows the region from which the satellite and map imagery was sampled from. Images were sampled to be of size 256 by 256 pixels without the street labels shown. The script for downloading is available at `https://github.com/RamaneekGill/MapGen/blob/master/data/download_map_data.py`

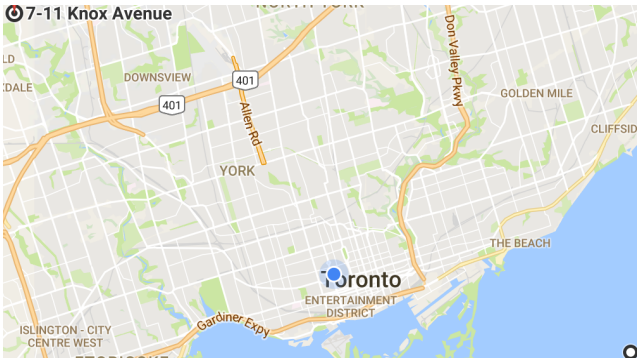

Figure 3: Region from where the dataset of satellite and map imagery was sampled from.

2.2 Model Architecture

We have implemented our code using Keras [7] as our framework of choice due to is popularity in open sourced AI development and research and because of its well documented and tested features.

We leverage the model architecture defined in Isola et al for our GAN and cGAN, their implementation is open sourced and was built upon the deep learning framework Torch [8]. Specifically we adopt idea to use an architecture akin to U-Net [9] as the generator in our GAN and cGAN, an example architecture can be seen in figure 4. U-net is an encoder-decoder model that is symmetrical in nature and has skip connections between each encoding and decoding pair layer.

Isola et al's modification to the U-Net is to make it deeper and use deconvolution layers instead of upsampling and to use batch normalization layers after every convolution or deconvolution layer to address internal covariate shifts [11] throughout the model when training. The exact implementation visualized with Keras is available at `https://github.com/RamaneekGill/MapGen/blob/master/generator_architecture.png`, it is too large to be visualized in a single page. We use the same generator architecture and also experiment with replacing the leaky ReLU activation functions with a parametric ReLU (PReLU) activation functions. PReLU activation functions have shown easy and cheap improvements in image classification because the slope of ReLU is learned through training process in neural networks [12]. We also initialize all weights in a similar manner as described in He et al, in Keras this is named `he_normal`.

For the discriminator we use the same architecture as defined in Isola et al, a visualization of this can be seen at `https://github.com/RamaneekGill/MapGen/blob/master/discriminator_architecture.png`.

In Isola et al's pix2pix framework they defined a custom loss function the empirically showed great results, due to time constraints their implementation could not be reverse engineered. Instead for our experiments we use a binary cross entropy as our objective function.

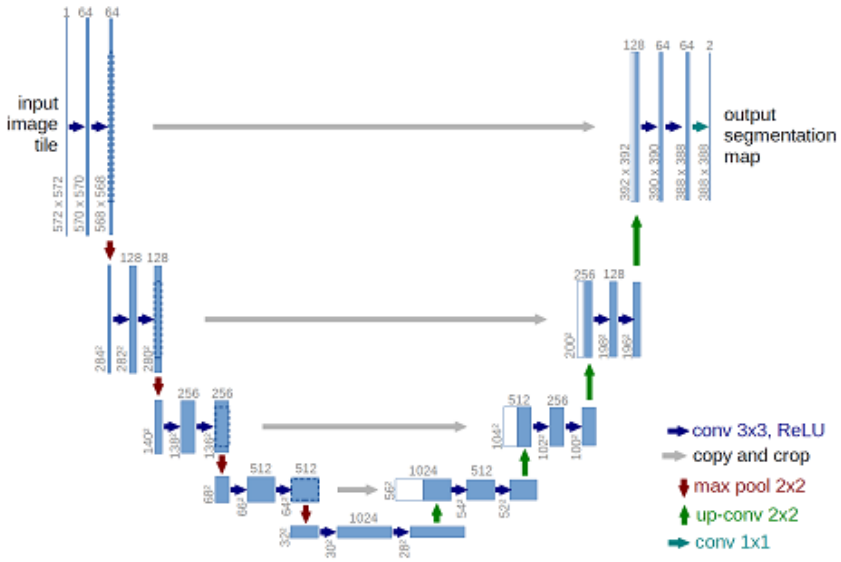

Figure 4: A sample U-Net architecture [10]

2.3 Conditioning the GAN

When conditioning the GAN we modify the `num_input_channels` value in `settings.py` to be 6, and concatenate the conditioning image to the `generated_images` in the `train` method. This allows images to be accessed by the generator model for generating output images. The image type for using for conditioning must be an RGB image, hence the change of the number of input channels from 3 to 6 because of the concatenation. Images used for the purpose of conditioning the generator or discriminator can be an raw image of size `image_width` by `image_height` as defined in `settings.py`.

2.4 K-Means Clustering for Conditioning

For processing images with k-means clustering we leverage the implementation in `OpenCV` [13]. We set $k = 4$ when clustering. Figure 5 shows example outputs of k-means clustering processing.

K-means clusters images on color or intensity. Given k clusters and n features, by randomly sampling k points the clusters can be initialized. The algorithm then is trained in an unsupervised fashion such that feature n_i belong to the closest cluster k_i. After each iteration of assigning features to clusters the means for each cluster and its members are computed, if the mean doesn't change very much it is assumed to have learned a converged model.

Figure 5: Applying 4-means clustering on map and satellite imagery. The left set is a good transformation, the right set is considered a poor representation from our implementation.

2.5 Canny Edge Detection for Conditioning

For processing images with a canny edge detector we leverage the implementation in `OpenCV`. We set threshold for canny edge detection for a minimum edge to 50 and for a maximum edge 300 for the intensity of a gradient. Figure 6 shows example outputs of the canny edge detector.

Canny edge detection works by first smoothing an image by convolving the image with a Gaussian kernel. Gradients are then computed among the pixels, ridges in gradients between the thresholds imply a detected edge which are then set to `True` through non-maximal suppression. This algorithm mathematically computes pixel intensities to determine changes "quick" changes in an image, often these are edges. Low minimum thresholds and high maximum thresholds make the detector too sensitive, this can be seen in the right two images within figure 6.

Figure 6: Applying a canny edge detector with a minimum gradient intensity of 50 and maximum gradient intensity of 300 to map and satellite imagery. The left set is a good transformation, the right set is considered a poor representation from our implementation.

2.6 Gradient Magnitude for Conditioning

For processing images using gradient magnitude, we apply a Gaussian filter with $\sigma = 0.5$ and calculate the gradient with respect to the horizontal component and vertical component. We take the euclidean norm of these components to find the magnitude of the gradient. Figure 7 shows two pairs of examples.

Gradient magnitudes of an image showcase how image's pixels change in relation to its nearby pixels. It is able to detect rough edges in an image. For edge detection canny edge detectors are a more sophisticated method and are generally shown to have performed more robustly in image segmentation tasks. The gradients are computed in a horizontal and vertical direction after smoothing the image with a Gaussian kernel. These two perpendicular vectors are the consumed to compute the Euclidean norm to find the magnitude of a gradient. The higher the magnitude of a given pixel, the higher the change in images' pixel in relation to its neighbors.

Figure 7: Calculating the gradient magnitude with $\sigma = 0.5$ on map and satellite imagery. The left set is a good transformation, the right set is considered a poor representation from our implementation.

2.7 Training Process

To train the GAN we first call `train.load_data()` to load the images in to memory. Then we call `model.create_model()` to define the graph to compute the tensors that represent the GAN.

The actual training process is defined in `train.train`. Training a GAN is generally a difficult task that requires a variety of fine tuning to get decent generations [14].

First the generator is asked to generate images based on the inputs, the inputs can be conditioned with any extra information either from the input or output images. This includes the raw image or either the images' representation when applied to canny edge detector, k-means clustering, or Gaussian gradient magnitude computer vision processing. The generated images and ground truth outputs are then fed to just the discriminator to train it to distinguish between fake and real images. After the discriminator has been training for a mini-batch its weights are frozen so that they cannot be updated until the next batch. Then the entire GAN is trained to generate images with the goal of convincing the discriminator to output `True`. This step is crucial in that the generator portion of the GAN is optimizing to make the discriminator unable to be able to distinguish between real and fake image. The backpropogation in this step uses an already "trained" discriminator's weights to optimize the weights in the generator portion of the GAN to create legible outputs.

2.8 Contribution

I (Ramaneek Gill) contributed to the literature review and idea formulation greatly. Implementation wise the reverse engineering of Isole et al's pix2pix framework was my task. I was able to successfully recreate the model architecture for the GAN and cGAN using a U-Net as a generator. I also wrote 80% of the script for mining the data off Google Maps. Charles Huang implemented the computer vision algorithms to use for training and the batch loading code for feeding the training process.

Both of us worked on debugging a Keras limitation equally. This is shown in more detail in the Challenges section.

For a more detailed view of exact code the commit history and logs can be viewed on GitHub.

Figures 1, 5, 6, and 7 were created by Charles. Figure 3 was created by myself.

3 Challenges

Our goal for this project was to aim for a publication. We chose to focus on generative adversarial networks since they have been gaining popularity among the academic community. With the recent release of code and astonishing results of Isola et al we chose to focus on image to image translation with fundamental computer vision approach to conditional GANs.

Generative adversarial networks are fairly new and there isn't a wide variety of readable and reproducible open sourced implementations available online when compared to convolutional neural networks. This greatly limited us and caused many failed experimentations with other code. Coupled with the fact that even though pix2pix is open source it isn't very readable because of the unfamiliar Lua scripting language and use of closures this project was soon to be realized as too ambitious to complete entirely.

We ran into a critical limitation of the Keras framework when compiling a GAN by combining two different generator and discriminator models with skip connections that prevented backpropagation. The details of the error were that the discriminator after having been compiled into a GAN could not be accessed independently for weight updates and loss calculations, this prevents us from training in an adversarial fashion. After further investigation we were able to create vanilla GANs without skip connections to successfully train on the same training code which concluded the error being an issue with the Keras framework.

4 Results

Describe and demonstrate your results. Compare various steps of your process, e.g. how much additional accuracy is gained by each step. Show example images and/or videos. Explain why your method works when it does, and why it fails. Show examples and justify (i.e. if it a is problem with the algorithm or a problem with your implementation).

Unfortunately we were unable to achieve any training results because of a limitation in Keras. The issue we have run into specifically is explained in more in the latter half of the challenges section.

Training our generator or discriminator alone works, however when combining the two to create a GAN backpropagation fails. We are able to create a very basic GAN and are able to train it so we have concluded that the issue lies within Keras.

5 Conclusion

We were able to successfully build a pipeline for end to end deep learning. This consisted of scripts that could scrape satellite and map imagery from Google Maps, apply k-means clustering, canny edge detection or calculate gradient magnitudes for the images,and build complex GANs and cGANs all in a modular fashion. What did not work was our training method for GANs. In the future before investing so heavily into a popular deep learning framework it should be tested in a quick environment to ensure it is the correct tool of choice for facilitating research.

References

[1] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *arXiv preprint arXiv:1611.07004*, 2016.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[3] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks*, 8(1):98–113, 1997.

[4] Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A Efros. Generative visual manipulation on the natural image manifold. In *European Conference on Computer Vision*, pages 597–613. Springer, 2016.

[5] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[6] Jon Gauthier. Conditional generative adversarial nets for convolutional face generation. *Class Project for Stanford CS231N: Convolutional Neural Networks for Visual Recognition, Winter semester*, 2014, 2014.

[7] François Chollet. keras. `https://github.com/fchollet/keras`, 2015.

[8] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.

[9] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer, 2015.

[10] Marko Jocic. ultrasound-nerve-segmentation. `https://github.com/jocicmarko/ultrasound-nerve-segmentation`, 2015.

[11] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

[13] Itseez. Open source computer vision library. `https://github.com/itseez/opencv`, 2015.

[14] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. *arXiv preprint arXiv:1609.04802*, 2016.