Conditional GAN for Image-to-Image Translation (Satellite ↔ google maps)

Yashshree Patil
Computer Science Department
Rutgers University
yap14@scarletmail.rutgers.edu

Abstract

Image-to-image translation aims to learn the mapping between two visual domains here satellite to the google map image. In this project cGAN (Conditional adversarial networks) will be investigated as a solution to the image-to-image translation problem of satellite- map translation and vice versa. We will be using Pix2Pix cGAN, an approach to train a deep CNN for image-to-image translation tasks/problems. The Pix2Pix model is a type of conditional GAN, or cGAN, where the generation of the output image is conditional on an input/source image. The GAN architecture is comprised of a generator model for outputting new plausible synthetic (fake) images, and a discriminator model that classifies images as real (from the dataset) or fake (generated) [6]. Dataset used is http://efrosgans.eecs.berkeley.edu/pix2pix/datasets/maps.tar.gz *dataset [4] which is a 255 MB zip file consisting of 2196 (Training and validation) image samples. If time permits we can expand the project to cGAN image to image translation on day-night, Winter - summer, black and white - color with relevant benchmark data sets.*

1. Introduction

In this project, I am going to inspect GAN in a conditional setting. For learning the generative model of data GAN is used Similarly conditional generative models are learned by cGANs. In image-to-image translation problems, the input image is conditioned to generate a corresponding output image. Hence, in this setting cGAN appears to be a suitable choice. To solve the problem of image-to-image translation of satellite images Pix2Pix approach of cGAN is used. We know that image-to-image translation requires specialized models and hand-crafted loss functions, which can be achieved by using Pix2Pix GAN as it provides a general-purpose model and loss function for image-to-image translation. [3]

2. Prior Work

Image-to-image translation problems are generally formulated on regression or per-pixel classification. It treats the output space as "unstructured" such that each output pixel is conditionally independent of all input images. Whereas Conditional GANs learn structured loss which penalizes the joint configuration of the output. [6]

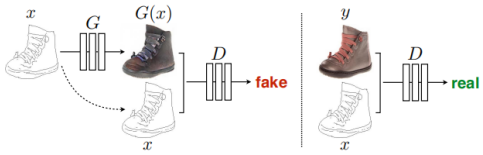

Figure 1. Generator G learns to fool the Discriminator D. Discriminator D learns to classify between fake and real. Unlike unconditional GAN. Both the G and D observe the input edge map.

Figure 1: In conditional GAN the loss is learned, and can, penalize any possible structure that changes between output and target.

Some of the researchers have used GANs for image-to-image translation but have used the unconditional GANs, UNIT [5] or multi-hop GAN [7], relying on L1, L2, or regression terms.

UNIT (Unsupervised image-to-image Translation): UNIT aims at learning a joint distribution of images in different domains by using images from the marginal distributions in individual domains. UNIT Framework is based on Coupled GANs. It translates an image from one domain to another without any corresponding images in two domains in the training dataset. But the framework has two limitations. First, the translation model is unimodal due to the Gaussian latent space assumption. Second, training could be unstable due to the saddle point searching problem. We plan to address these issues in future work [5].

GANHopper: Multi-Hop GAN for Unsupervised image-to-image Translation, transforms images gradually between two domains, through multiple hops. Instead of executing translation directly, it steers the translation by requiring the network to produce in-between images that resemble weighted hybrids between images from the input domains. GANHopper excels at image translations involving domain-specific image features and geometric variations while also preserving non-domain-specific features such as general color schemes [7].

I am not the first to apply GANs in the conditional setting. Prior and concurrent works have conditioned GANs on discrete labels, text, and images. Their work differs in architectural choices for the discriminator and generator. I have used "U-Net" Architecture for generator and convolutional "PatchGAN" classifier for discriminator, a similar architecture appeared in [6]. But unlike them, my focus is more on the image-to-image translation of Satellite to Google map and vice versa, we can say that it is a recreation of cGAN image-to-image translation [6] focusing more on satellite and google map images.

3. Technical Details

3.1. Architecture

The architecture used in this project consists of 2 models: The discriminator and The generator.

3.1.1 Discriminator

The discriminator model performs conditional image classification. The input to the discriminator is the source and the target image, in our case, it is Satellite and google maps image respectively. The discriminator is based on Deep CNN and the job of the discriminator is to predict the likelihood or the probability of whether the output image/target image is real or generated(Fake) translation of the input/source image.

For building the discriminator function I have used PatchGAN model, it is based on an effective receptive field of the model, which gives a correlation between an output patch to the number of pixel in the input picture, it is designed in such a way that the output prediction model is patched to 70x70 square. This approach can be widely used because this model can take up images of different sizes. i.e. it can input images of the size which is not equal to 256x256 pixels.
It is not necessary to take 70x70 PatchGAN we can also take 16X16 or 1x1 but the highest accuracy obtained is in 70x70 PatchGAN (see Evaluation).

The discriminator model build for this project takes up 2 input concatenated images, which are runs through the discriminator model to give a predicted patch output. The output of the discriminator gives the probability or likelihood that the patch of the input image is real. These output values can later be used to calculate the overall Probability (by taking the average) or likelihood or the classification score. [1]

Binary cross-entropy is used to optimize the discriminator model and weighting is used so that updates to the model have half (0.5) the usual effect. This weighting of model updates is used to slow down the changes to the discriminator, relative to the generator model during training.

The discriminator function implements the 70×70 PatchGAN discriminator model as per the design of the model in the paper. The model takes two input images that are concatenated together and predicts a patch output of predictions. The model is optimized using binary cross-entropy, and weighting is used so that updates to the model have half (0.5) the usual effect. This weighting of model updates to slow down changes to the discriminator, relative to the generator model during training. [2]

3.1.2 Generator

The discriminator model is simpler than the generator model, The generator is a complex encoder-decoder model with a U-Net architecture.
[1] The input of the generator model is the source image (satellite image)
The output of the generator model is the target image (google maps image)
This is achieved by the following process:

1. The image is first down-sampled through the U-Net Architecture to the bottleneck layer this process is also known as encoding.

2. Then it is upsampled through the bottleneck representation to the output image size this process is also known as decoding.

The overall shape of this architecture looks like an alphabet "U" hence, known as U-net architecture. How is U-net different from the simple encoder-decoder architecture of the generator? The U-net architecture has skip connections between the encoding and the decoding layers.

Figure 2 makes the skip-connections clear, showing how the first layer of the encoder is connected to the last layer of the decoder (dotted arrow), and so on.

The encoder and decoder consist of convolutional, dropout, activation layer, and batch normalization blocks. These all blocks are standardized blocks this means that a function can be build to construct each block of layer and call it recursively to build multiple encoder and decoder layers. This function can be called to build layers of encoder and decoder separately. In the output layer, tann activation function is used so that the pixel value in the generated image will be between [-1,1]. [2]

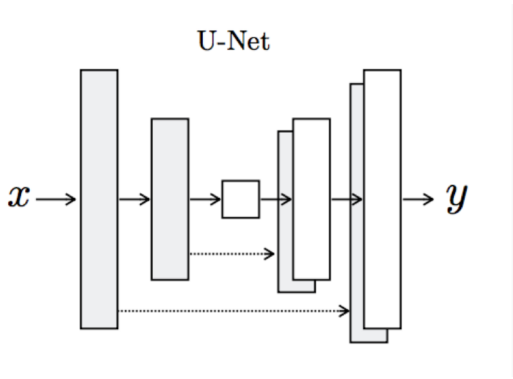

Figure 2. Generator Model : U-Net Architecture

4. Working

4.1. Dataset and Pre-processing

Dataset that I am using is : http://efrosgans.eecs.berkeley.edu/pix2pix/datasets/maps.tar.gz [4].

This dataset comprises the satellite images of New York and their corresponding Google maps pages.

There are a total of 2195 satellite-google map image pairs. Such that 1096 image pairs are in the Train folder and 1099 image pairs are in the validation folder. Each image contains both a google maps image and its corresponding satellite image. (Satellite image on the left and google map image on the right)
[2] Dimensions of the images are 1,200 pixels in width and 600 pixels in height.

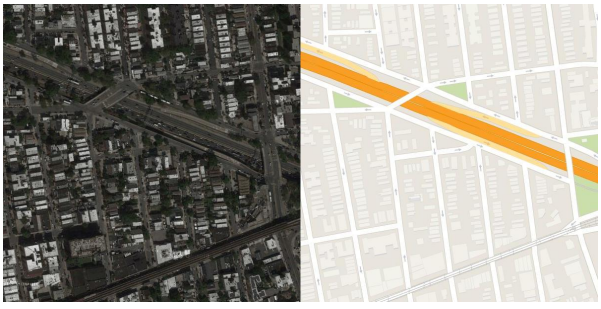

Figure 3. Dataset Images

Each image consists of both google maps and satellite images hence we need to prepare the dataset for proper training (by separating the satellite and google maps images). This can be done by using Pix2Pix GAN model in Keras. The images in the training dataset will be loaded, rescaled, and split into the satellite and Google map elements and the resulting image will be 1096 separate image pairs with the dimension of 256x256 pixels. [2]

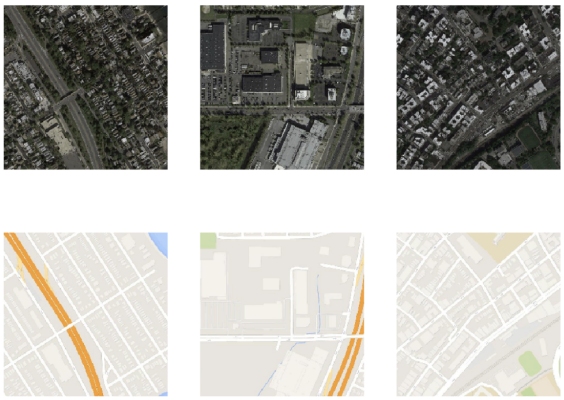

Figure 4. Data after prepossessing

A plot of three image pairs after pre-processing of train data is created showing the satellite images on the top and Google map images on the bottom, Figure 4.

It can be observed that the satellite images are complex in appearance and even though the google maps images are simpler, they are color-coding as for parks the color is green, while major roads are colored in yellow and water is color in blue, etc.

4.2. Training

4 input parameters required by the training function are generator, discriminator, composite model, and loaded dataset (pre-processed dataset). For this project, I have used 100 training epochs (Due to hardware constraints) and batch size = 1. But training epochs of 200 is advisable [6]

Composite model is the combination of Generator and discriminator (such that the generator is stacked on the discriminator)

For training as mentioned above, we are going to use a fixed iteration of 100 epochs. The totals images available in the train data are 1096. 1 epoch means running over all the images once (i.e. 1 iteration through all the images in train dataset) batch size of 1 means 1096 training steps. The generator model is saved and evaluated at every 10 epochs i.e. 10 epochs covers 10960 training steps. therefore, to complete 100 epochs there will be a total of 109600 training steps to be completed.

Training steps:

1. In each training step a batch of real examples are selected at random, then a generator is used to generate a batch of matching generated (Fake) samples using the real image source from the dataset.

2. The discriminator is then updated with a batch of fake and real images.

3. the generator model is updated providing the real source images as input and providing class labels of 1 (real) and the real target images as the expected outputs of the

model required for calculating loss. The generator has two loss scores as well as the weighted sum score. [1]

4. The weighted sum score is more important to us as it is used to update the model weights.

5. The loss of each iteration is stored and the performance of the model is evaluated after 10 training epochs i.e 10960 training steps.

From figure 5, This is the output obtained after completing 100 epochs and we can see that The loss is reported at each-training iteration.

We have 3 types of Loss (d1, d2, and g)

d1 - discriminator loss (real image example)

d2 - discriminator loss (Generated/fake image example)

g - weighted average of cGAN and L1 loss

NOTE: If the loss for the discriminator goes to zero and stays there for a long time, it is an example of a training failure (consider re-starting the training run). [2]

```
>1, d1[0.566] d2[0.520] g[82.266]
>2, d1[0.469] d2[0.484] g[66.813]
>3, d1[0.428] d2[0.477] g[79.520]
>4, d1[0.362] d2[0.405] g[78.143]
>5, d1[0.416] d2[0.406] g[72.452]
...
>109596, d1[0.303] d2[0.006] g[5.792]
>109597, d1[0.001] d2[1.127] g[14.343]
>109598, d1[0.000] d2[0.381] g[11.851]
>109599, d1[1.289] d2[0.547] g[6.901]
>109600, d1[0.437] d2[0.005] g[10.460]
>Saved: plot_109600.png and model_109600.h5
```

Figure 5. Complete Training

After every 10 epochs, a model is saved in the same directory with .h5 extension the file name is the last training iteration number before the creation of the model. Figure 6, shows the images generated after 10 epochs (First model) map images are generated that look plausible (a basic skeleton of the satellite image is created in google maps, larger structures are in place with the right colors), but the lines for streets are not entirely straight, it contains some blurring.

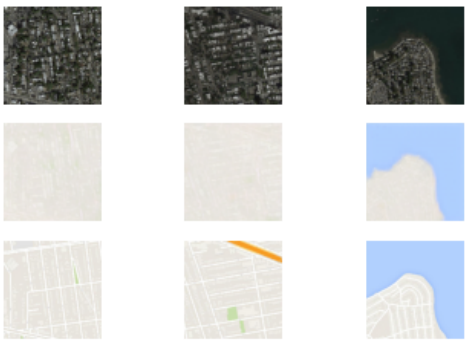

Figure 6. 10 epochs

Figure 7, shows images generated after 50 epochs, begin to look very realistic, much better than the plot generated after 10 epochs. Quality appears to remain good for the remainder of the training process.

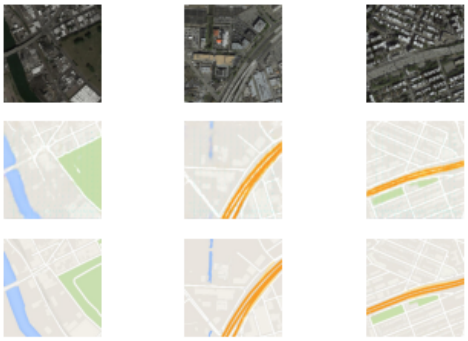

Figure 7. 50 epochs

Figure 8, shows images generated after 100 epochs. If we look carefully there is a notable difference between the first generated and the first target image (on the left-hand side) is that the generated image includes more useful detail than the real Google map image.

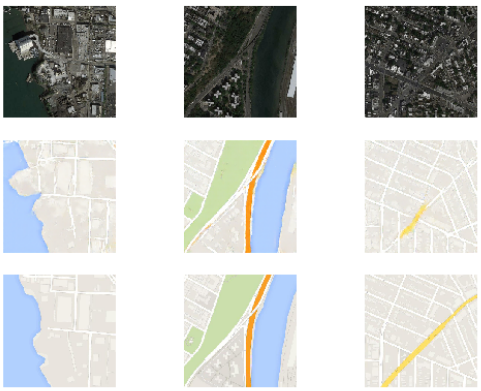

Figure 8. 100 epochs

4.3. Results

As said earlier after running 100 epochs/109600 training iterations the Pix2Pix model results in 10 generated image plots and 10 saved models. To use these models in image-to-image translation we need to load the most accurate model, training dataset, and choose a random image pair to test on the model. This image chosen is from the training dataset.

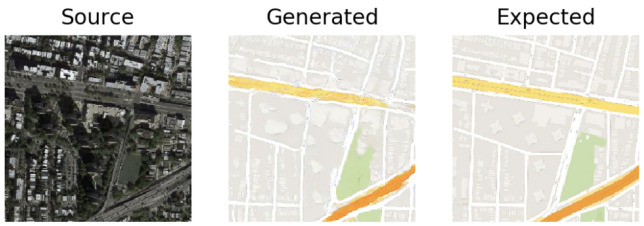

Figure 9. Image Translation

We can see that the generated image shows the large roads with yellow and orange color the parks are highlighted in green. The generated image is not of very high quality but is close to the expected image.

After testing it on an image pair from the training dataset, We can now use the model to translate a given standalone image, figure 10

Figure 10. Standalone image, from validation set

For standalone an image was selected from the validation dataset and the satellite element of the image was cropped and pasted/saved in the working directory and used as input to the model.

We must load the image as a NumPy array of pixels with the size of 256×256, rescale the pixel values to the range [-1,1], and then expand the single image dimensions to represent one input sample. [1]

Running the example will load the image from the file, creates a translation google maps image for it, and display the output on the screen.

Figure 11, The generated image appears close to the accurate translation of the input image. The streets are still not in straight lines and not all buildings are visible clearly. With more training (increasing epochs) or choice of a different model (for generator/discriminator), higher-quality images can hopefully be generated. [1]

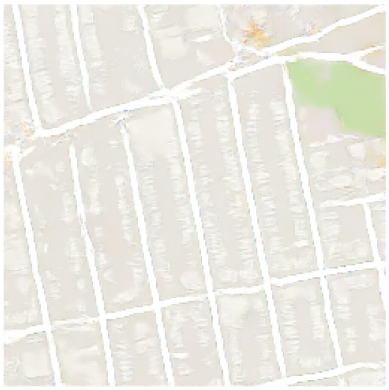

Figure 11. Predicted standalone output

5. Google maps to satellite

After developing a good translation model for Satellite to google maps translation, it is quite easy to build a model that transforms Google maps to the satellite. To achieve this I had used the same code to train the model with only one difference while loading the dataset/the real samples switch the satellite images with the google maps images and the model will be trained such that when given a google map image it would develop its corresponding satellite image.

```
>1, d1[0.442] d2[0.650] g[49.790]
>2, d1[0.317] d2[0.478] g[56.476]
>3, d1[0.376] d2[0.450] g[48.114]
>4, d1[0.396] d2[0.406] g[62.903]
>5, d1[0.496] d2[0.460] g[40.650]
...
>109596, d1[0.311] d2[0.057] g[25.376]
>109597, d1[0.028] d2[0.070] g[16.618]
>109598, d1[0.007] d2[0.208] g[18.139]
>109599, d1[0.358] d2[0.076] g[22.494]
>109600, d1[0.279] d2[0.049] g[9.941]
>Saved: plot_109600.png and model_109600.h5
```

Figure 12. Training for translation of google maps to satellite images

6. Evaluation

The model is evaluated on FCN scores in different working conditions.

Figure 14 Best FCN score is obtained by using receptive field size of the discriminator as 70x70 followed by 16x16, 286x286 , 1x1.

Note: Input images are 256 × 256 pixels and other/larger

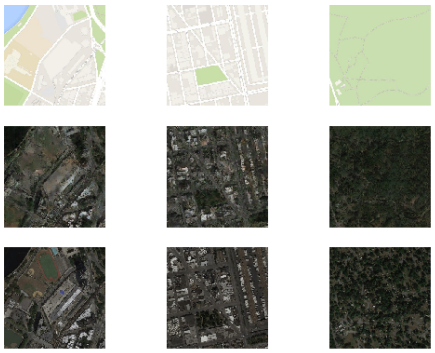

Figure 13. Predicted output for Google maps to satellite

Discriminator receptive field	Per-pixel acc.	Per-class acc.	Class IOU
1×1	0.39	0.15	0.10
16×16	0.65	0.21	**0.17**
70×70	**0.66**	**0.23**	**0.17**
286×286	0.42	0.16	0.11

Figure 14. FCN scores for different receptive field sizes of the discriminator.

receptive fields are padded with zeros. [6]
Figure 15 data is generated by running 10 epochs, the best FCN score is obtained by using U-net Architecture (L1 + cGAN) followed by U-net (L1), Encoder-decoder(L1), Encoder-decoder(L1+cGAN)

Loss	Per-pixel acc.	Per-class acc.	Class IOU
Encoder-decoder (L1)	0.35	0.12	0.08
Encoder-decoder (L1+cGAN)	0.29	0.09	0.05
U-net (L1)	0.48	0.18	0.13
U-net (L1+cGAN)	**0.55**	**0.20**	**0.14**

Figure 15. FCN scores for different generator architectures (and objectives)

7. Extension and Conclusion

1. Standalone Satellite.

Just as we tried to convert a standalone image from satellite to google maps. Similarily we can try for google maps t satellite.
Choose a random image from the validation dataset, crop its google map picture, and check the translation of a standalone image for a google map to the satellite. Is the generated image similar to the target image?

2. More Training/more epochs.

Instead of training the data on 100 epochs train it on 200 epochs and check whether the additional training results in improvement of the output image sharpness or quality.

3. Image Augmentation.

Use some minor image augmentation during training as described in the Pix2Pix paper [6] and evaluate whether it results in better quality generated images.

cGANs appear to be a promising approach for Satellite to google maps and vice versa, we can expand this to other image translation tasks such as conversion from day to night/ winter to summer, etc.
Because of its feature to learn loss adapted to the task and data at hand. It can be applied to a wide variety of settings/applications.

References

[1] I. Ahmed. cgan for image-to-image translation,https://www.kaggle.com/ibtesama/image-to-image-translation-with-cgan. 2, 4, 5

[2] J. Brownlee. How to develop a pix2pix gan for image-to-image translation,https://machinelearningmastery.com/how-to-develop-a-pix2pix-gan-for-image-to-image-translations/. 2, 3, 4

[3] J. Brownlee. Pix2pix, https://machinelearningmastery.com/a-gentle-introduction-to-pix2pix-generative-adversarial-network/. 1

[4] Dataset. http://efrosgans.eecs.berkeley.edu/pix2pix/datasets/maps.tar.gz. 1, 3

[5] J. K. Ming-Yu Liu, Thomas Breuel. Unsupervised image-to-image translation networks. 1

[6] T. Z. A. A. E. Phillip Isola, Jun-Yan Zhu. Image-to-image translation with conditional adversarial networks. 1, 2, 3, 6

[7] D. R. D. C.-O. Wallace Lira1, Johannes Merz1 and H. Zhang. Multi-hop gan for unsupervised image-to-image translation. 1, 2