Department of Electronics and Communication Engineering

Rajiv Gandhi University of Knowledge Technologies

Nuzvid, Krishna Dt. - 521202

Tree Crown Extraction from Aerial Imagery using cGAN

Under the Guidance of

Dr. S N Omkar

Chief Research Scientist, Aerospace Department,

IISc-Bangalore.

Certificate

This is to certify that, this is a bonafide record of the project presented by me whose name is **V.V.S.S. Anil Kumar** with ID number **N140441**, during summer internship after Engineering 3rd year in partial fulfilment of the requirements of the degree of Bachelor of Technology in Electronics and Communication Engineering.

Prof. S N Omkar

(Project Guide)

Acknowledgement

I would like to express my sincere gratitude towards **Dr. S N Omkar** and **Mr. Shivlal** for providing me guidance, support and valuable suggestions throughout the course of our project.

Table of Contents

1 Abstract

Generative Adversarial Networks (GANs) are the special type of Neural Networks which are the state-of-the-art and showcasing tremendous performance in many areas especially in image-to-image translation based applications (Eg.: pix2pix). Usually, GAN works on the Adversarial (means that a fight between two persons in the pit) Logic. The architecture of GAN mainly contains two blocks which are the Generator and the Discriminator. These two are constructed using neural networks. And these two will fight like two players in the pit. The generator will be trying to produce similar result to our requirement while the discriminator will check this produced result and verifies whether to trust this or not.

And the Conditional Generative Adversarial Network (cGAN) is an extension of the Generative Adversarial Networks (GANs). For this cGAN, we additionally provide an extra input which is called as Condition or label. Now it's become closer to the supervised learning from semi-supervised learning. Using this cGAN**,** we built a program **'Tree-Crown Extraction from Aerial Imagery'** i.e., completely using Keras and Tensorflow. We've used the mango dataset which consists of 6360 aerial images of mango trees which are obtained from Drones from different angles and different places. In these, we've 5100 images for training and 1260 images for testing processes. In addition, we compared our cGAN model using two different generators, one is the most popular ResNet and the other is the U-Net. Results are neatly piled up and metrics are evaluated as to help studying about this model.

\

2 Introduction

2.1 Generative Adversarial Networks (GANs):

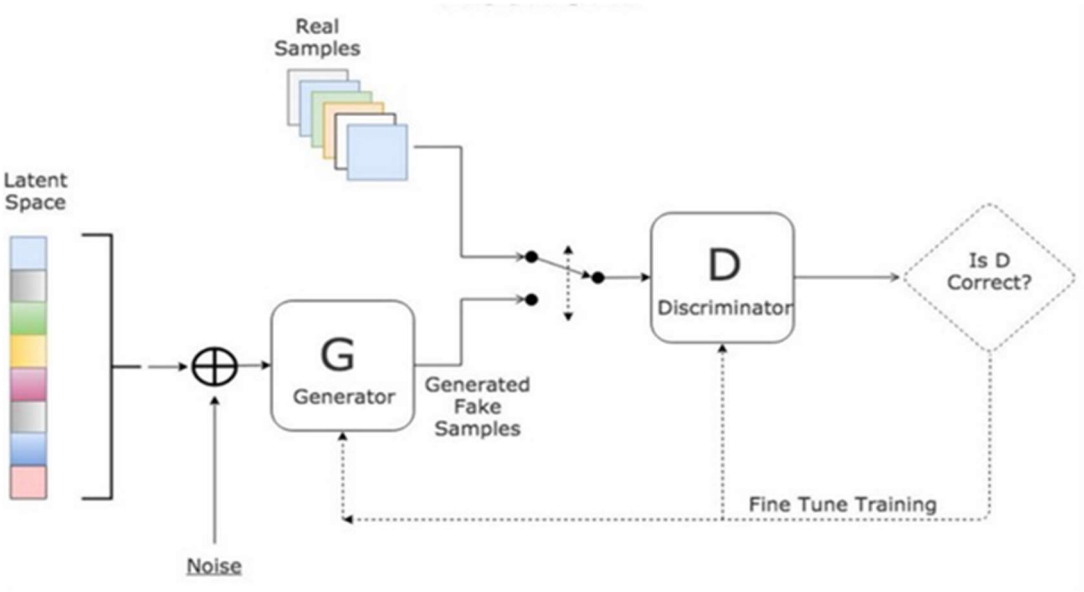
Generative Adversarial Networks (GANs) are evolved for unsupervised learning. So, for supervised learning we know the labels and those labels are embedded in the dataset themselves. So we can classify them easily, whereas in unsupervised learning the network doesn't know about labels. And GAN consists of a two adversarial blocks named the Generator and the Discriminator. Think it like a game where the Generator tries to produce some data from probability distribution and Discriminator acts like a judge. The Discriminator decides whether the given data is real (from the training distribution) or fake (Data generated from the random distribution). The Generator tries to optimize the loss to be minimum so that the generated data can match the true data from the training distribution. Or we can say that the Discriminator is guiding the Generator to produce more realistic data from the scratch. They both work like two players, fighting each other, in a pit. Hence the term **'Adversarial'** in the GAN. And GANs are to generate the desired data from the noise, these are the one type of **'Generative'** models which uses the Neural **'Networks'** in their architecture.

2.2 Conditional GANs:

Conditional GANs are the very interesting extension to the GAN framework. cGANs are allowed to generate images that have certain conditions or attributes. Let us take a GAN model which generates the digits from the random noise. And here in this model, we are unable to request a particular digit from the Generator. This problem can be addressed by a variation of GAN called Conditional GAN (cGAN). Through this, we can add an additional input which is literally called as a **'Condition'**. So using this feature of cGAN, we can modulate the .cGANs have one disadvantage that these are not strictly unsupervised and we need labeled data for them to work. If we want to generate particular data from the generator then we have to give that particular condition to both Generator and Discriminator. So now the discriminator has to classify the images and also it should check whether the condition is satisfied or not. From this we can generate what we want rather than generating all the data. By doing this we can reduce the training time. So many applications are there for cGANs, but now we will focus on Image-Label transformations.

3 Architecture & Training

3.1 Typical GAN Architecture:



3.2 Generator Network:

Generator network will take the inputs from noise data and produce samples nearer to true samples with some assumptions. And the generator network doesn't know what is the real data exactly is like. With some assumptions it will generate data and send it through the discriminator. Make these inputs go through the network and collect the generated outputs, Compare the "True distribution" and the generated one based on the available samples. Use backpropagation to make one step of gradient descent to lower the distance between true and generated distributions. As written above, when following these steps, we are applying a gradient descent over the network with a loss function that is the distance between the true and the generated distributions at the current iteration.

3.3 Discriminator Network:

Discriminator will take the inputs from both the generator and discriminator and it will try to differentiate the fake data from real data. And it will send the feedback to the generator, to improve its efficiency.

So, for every iteration it will send the feedback to the generator and it will try to produce the samples exactly like real data. However, when the iterations increase generator exactly produce the samples like real ones. So, then discriminator can't differentiate both, so it is in confusion state, then this min-max game will end.

3.4 Loss Function:

To learn the generator's distribution p_g over data x, we define a prior on input noise variables $p_z(z)$, then represent a mapping to data space as G (z; θ_g), where G is a differentiable function represented by a multilayer perceptron with parameters θ_g. We also define a second multilayer perceptron D (x; θ_d) that outputs a single scalar. D(x) represents the probability that x came from the data rather than pg. We train D to maximize the probability of assigning the correct label to both training examples and samples from G. We simultaneously train G to minimize log (1-D (G (z))). In other words, we can describe the loss function with value function V (G, D) as follows,

$$\min \max V(D,G) = E_{x\sim pdata(x)} [\log D(x)] + E_{z\sim pz(z)}[\log(1-D(G(z)))]$$

3.5 Training process:

We will train the network until our generator win the game which means that the probability of generating an image that can be identified as real by the discriminator. In this we will fit some hyper parameters which are very important to get efficient output. So, we have to tune these parameters in the training process.

3.6 Tuning of Hyper Parameters for Training:

1. **Batch normalization**:

- To increase the stability of a neural network, batch normalization normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation.

- We normalize the input layer by adjusting and scaling the activations. If we have more training examples then we normalize the layers to speed up the learning. And we can also add to hidden layers
- And it adds noise to the hidden layers like dropouts. So, when we use Batch normalization then we will use fewer dropouts because we don't want to lose more information.
- Batch normalization allows each layer of a network to learn by itself a little bit more independently of other layers.

2. Activation functions:

- **ReLU:** ReLU stands for rectified linear unit. ReLU is linear for all positive values, and zero for all negative values.

- **Leaky ReLU:** It overcomes the problem of 'Dying ReLU' that means for all negative values it will be zero. So that neurons won't learn anything. But in Leaky ReLU it has small slope for negative values. It speeds up training. Unlike ReLU, leaky ReLU is more "balanced," and may therefore learn faster.

3. Optimization algorithms:

- **Gradient Descent:** A simple optimization method in machine learning is gradient descent (GD). When you take gradient steps with respect to all 'm' examples on each step, it is also called Batch Gradient Descent. When we do the back propagation, the gradients may vanish in the middle of the layers before reach the input layer. So, then it's very difficult to get the exact features at the input layer.

- **Stochastic Gradient Descent:** In Stochastic Gradient Descent, we will use only 1 training example before updating the gradients. When the training set is large, SGD can be faster. But the parameters will "oscillate" toward the minimum rather than converge smoothly.

- **Mini Batch Gradient Descent:** It uses an intermediate number of examples for each step. With mini-batch gradient descent, we will loop over the mini-batches instead of looping over individual training examples. But here in mini-batch gradient descent it will update the parameters after seeing all the examples, so it may have some variance towards the convergence, and oscillations in its path. So, to this we have to add something which will take these gradients smoothly towards convergence.

- **Momentum (Beta):** Momentum takes into account the past gradients to smooth out the update. We will store the direction of the previous gradients in the variable v. formally; this will be the exponentially weighted average of the gradient on previous steps. If $\beta=0$, then this just becomes standard gradient descent without momentum. If we take the β value as large then the gradients become smoother but if the β value is too large then its fully smoothen the gradients which unnecessary. Common values for β range from 0.8 to 0.999.

- **Adam:** Adam is one of the most effective optimization algorithms for training neural networks. It combines ideas from RMSProp (described in lecture) and Momentum. It calculates an exponentially weighted average of past gradients, and store it in some variable and it will do biasing and it takes the correction value.

3.7 Regularization techniques:

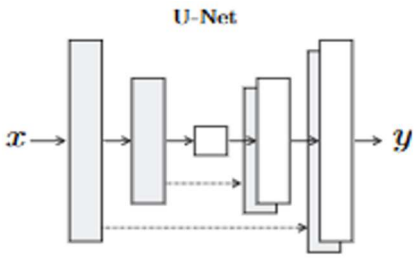
By using the non-regularized model the training set may over fitted. Sometimes it also includes the noisy points.

1. **L2 regularization:** The standard way to avoid over fitting is called L2 regularization. L2-regularization relies on the assumption that a model with small weights is simpler than a model with large weights. Thus, by penalizing the square values of the weights in the cost function you drive all the weights to smaller values. It becomes too costly for the cost to have large weights! This leads to a smoother model in which the output changes more slowly as the input changes.

2. **Dropout:** It randomly shuts down some neurons in each iteration. At each iteration, we will shut down each neuron of a layer with probability (1−keep_prob) or keep it with probability keep_prob.The neurons which were dropped that won't be used in either forward or back propagation in the iteration. So, at each iteration we will train the different model which contains some neurons. By doing this Dropout the neurons won't depend on some other neurons it will become less sensitive to other activation neurons.

3.8 Architecture of Conditional gan:

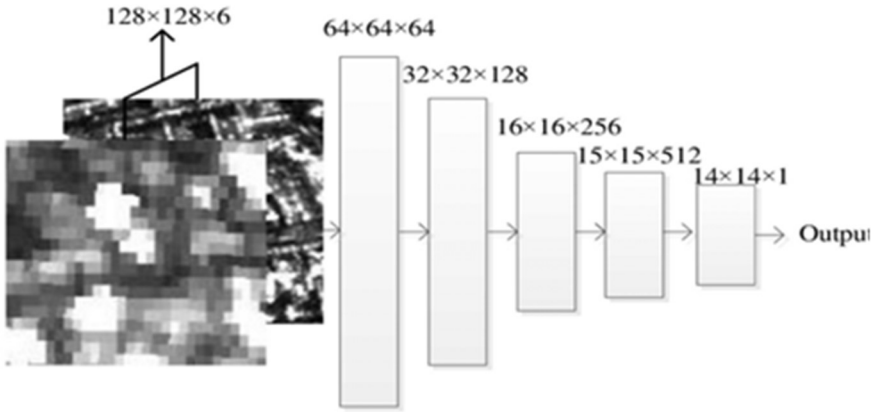
3.9 Generator architecture for Tree crown extraction:



For this application we will use U-net generator as well as ResNet generator. In image-image translation the problem is like mapping of high resolution input grid to a low resolution input grid. And the difference mainly appear in spatially. By remembering all those things we have to model the Generator. In this we used encoder-decoder network. So we will pass the input through all the layers up to where we will start decoding. U-net's network has skip connections between Encoder layers and Decoder layers. The first layer of Encoder is directly passed to the last layer of the Decoder and output of the second layer of Encoder is pass to the second last layer of the Decoder and so on. The Encoder network of the Generator network has seven convolutional blocks. Each convolutional block has a convolutional layer, followed a LeakyRelu activation function. Each convolutional block also has a batch normalization layer except the first convolutional layer. The Generator's Decoder Architecture. The Decoder network of the Generator network has seven upsampling convolutional blocks. Each upsampling convolutional block has an upsampling layer, followed by a convolutional layer, a batch normalization layer and a ReLU activation function.

3.10 Discriminator:

When we calculate the losses for an image, in that it can capture the low frequencies, but it fails to capture the high frequencies. So now we will restrict GAN discriminator to only model high-frequency structure. And L1 loss will take care of low-frequency structure. So we can restrict our model to the structure in local image patches. There is no need to go through the entire image. So in this CGANS we will design discriminator as to classify every N*N patches in the image and we run this discriminator convolutationally across the image, averaging all responses to provide the ultimate output of Discriminator.



3.11 Dataset:

Used Mango dataset which is obtained from Drone's Aerial imagery. It consists of 6000 training images and 2000 test images. And we used COCO data set 2017 for training using this same program/algorithm. It is around 18GB having images of different dimensions. Used 800 images for training (Very less, you can take more (approx. 350 thousand according to original paper) if you can collect and have a good GPU). Preprocessing includes cropping images so that we can have the same dimension images. Images with the same width and height are preferred. I used images of size 384 for high resolution.

3.12 Training procedure:

Generator will be trained on both the images and labels and that labels corresponding to that particular image. Discriminator take the both generated and original label and depending upon the condition it will create label. When gradient converges to local minimum then we will stop the training. We will adjust the learning rate to get better performance. Discriminator will send the feedback to the generator, then it will update its parameters respectively.

3.13 Tuning of Hyper Parameters for Training:

As we mentioned in the GAN's section, here also same procedure will be followed to tune hyper parameters. **Batch normalization**, **Activation functions (**ReLU, Leaky ReLU), **Optimization algorithms (**Gradient Descent, Stochastic Gradient Descent, Mini Batch Gradient Descent, Momentum, Adam).

3.14 Regularization techniques:

By using the non-regularized model the training set may over fitted. Sometimes it also includes the noisy points. So we use these techniques: L2 regularization, Dropout. Detailed explanations are already provided in the above GAN's section.

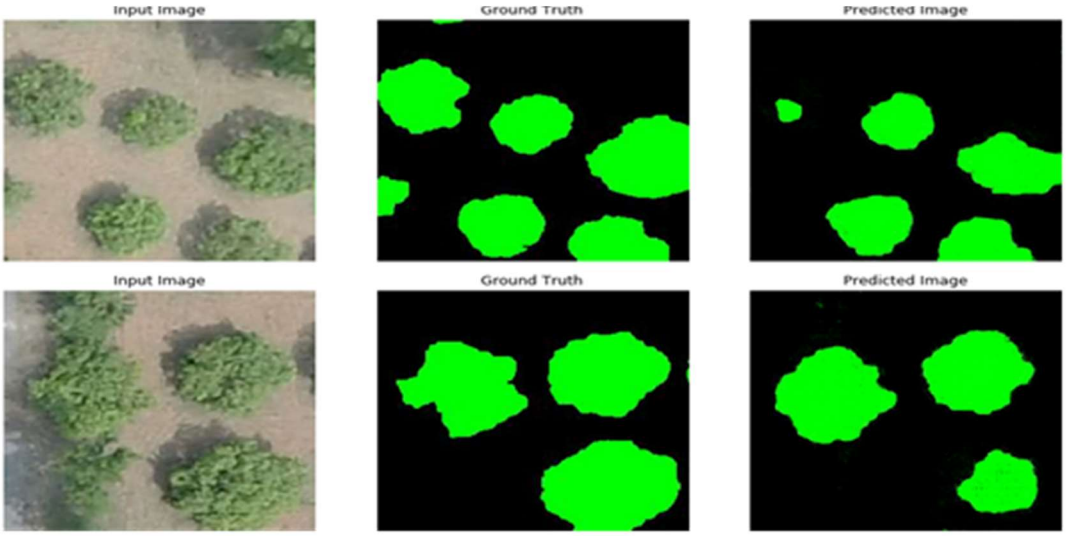
4 Applications of cGAN

There is a wide-ranging suite of possible applications for cGANs.

- **Image-to-image translation** - cGANs were demonstrated to be effective at synthesizing photos from label maps, reconstructing objects from edge maps, and colorizing images, among other tasks. This led to the development of cGAN-based software, pix2pixHD34.
- **Text-to-image synthesis** - an experimental TensorFlow implementation of synthesized images on the top of implementation of TensorFlow.
- **Video generation** - deep neural network that can predict the future frames in a natural video sequence.
- **Convolutional face generation** - cGANs use to generate faces with specific attributes from nothing but random noise.8
- **Generating shadow maps** - introduced an additional sensitivity parameter to the generator that effectively parameterized the loss of the trained detector, proving more efficient than previous state-of-the-art methods.
- **Diversity-sensitive computer vision tasks** - explicitly regularizes the generator to produce diverse outputs depending on latent codes, with demonstrated effectiveness on three cGAN tasks: image-to-image translation, image in-painting, and video prediction / generation.

5 Results and Metrics

Test results for Tree crown detection when trained on Mango Dataset



Test result for an image from COCO dataset when trained on COCO dataset

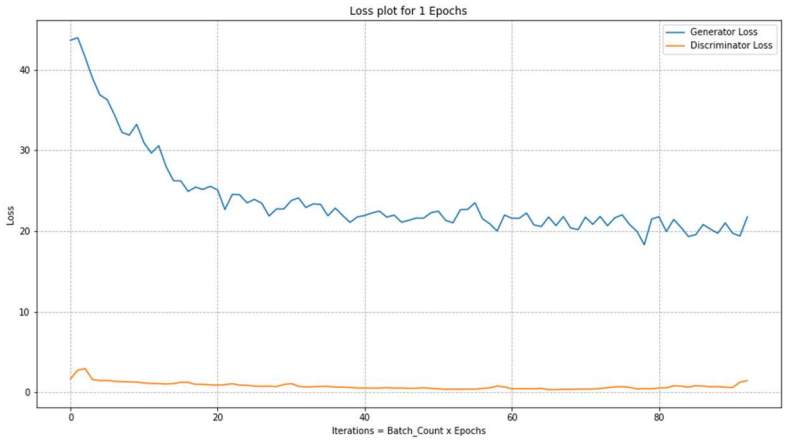
Metrics

```
In [102]: get_metrics(normalize=False)
```

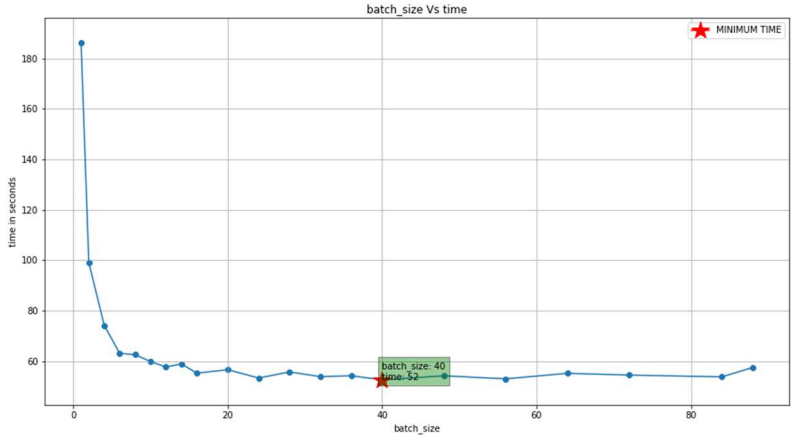
Out[102]:

	TP	FN	FP	TN	Accuracy	Precision	Recall	F1_score
test_img_1	48973	2130	545	13888	0.959183	0.988994	0.958319	0.973415
test_img_2	59325	288	792	5131	0.983521	0.986826	0.995169	0.990980
test_img_3	40168	5954	149	19265	0.906876	0.996304	0.870908	0.929395
test_img_4	41528	5582	871	17555	0.901535	0.979457	0.881511	0.927907
test_img_5	38468	7305	10	19753	0.888382	0.999740	0.840408	0.913176
test_img_6	53911	1581	438	9606	0.969193	0.991941	0.971509	0.981619
test_img_7	56081	1505	307	7643	0.972351	0.994556	0.973865	0.984102
test_img_8	59797	346	378	5015	0.988953	0.993718	0.994247	0.993983
test_img_9	50255	1198	271	13812	0.977585	0.994636	0.976717	0.985595
test_img_10	38226	9173	108	18029	0.858383	0.997183	0.806473	0.891745
test_img_11	49033	2033	485	13985	0.961578	0.990206	0.960189	0.974966

Generator and Discriminator Losses



Batch size vs Time Plot



6 Future Works

In this project we only focused on the extraction of the crown of the tree. In future we may try for detection of tree type. And we can also try different new datasets on our program and can obtain better results like we did for the COCO dataset in this project.

7 References

- Mehdi Mirza, Simon Osindero. Conditional Generative Adversarial Nets. 2014.
- Augustus Odena, Christopher Olah, Jonathon Shlens. Conditional Image Synthesis with Auxiliary Classifier GANs. 2016.
- http://cs231n.stanford.edu/reports/2017/pdfs/17.pdf
- https://medium.com/@jonathan_hui/gan-super-resolution-gan-srgan-b471da7270ec
- https://arxiv.org/pdf/1609.04802.pdf
- http://zpascal.net/cvpr2017/Isola_Image-To-Image_Translation_With_CVPR_2017_paper.pdf
- https://becominghuman.ai/unsupervised-image-to-image-translation-with-generative-adversarial-networks-adb3259b11b9