Semi Supervised Learning with AC-GAN

1 Introduction

Generative Adversarial Networks(GANs) are composed of two neural networks competing with each other, one that generates samples from noise and another which concludes whether the generated samples are a good approximation of some real data that it has been trained on[1]. GANs have been successfully implemented in semi supervised learning by providing a deep network with labelled and unlabelled data from both real and generated sources enabling it to perform classification much more efficiently than if it were simply presented with a conventional labelled dataset[2]. Such a classifier would theoretically be able to converge almost as quickly on dataset with fewer samples as it would converge on a dataset with many thousands of samples such as MNIST or CIFAR. In our project, we attempt to apply a Semi Supervised Learning technique using an optimized Auxiliary Classifier GAN for our own custom dataset obtained by web scraping. We also address the various issues mentioned in the referred research paper such as mode collapse, minibatch discrimination etc. as they relate to our own problem and experiment with various techniques to overcome them and converge towards an optimal solution.

2 Approach

2.1 Data Collection

We decided to assemble our own dataset comprised of images of car logos which were scraped from the internet. We decided to settle on car logo images because we could not find other projects which implemented a GAN to generate car logo images. We divided the data into 23 total classes, with each class corresponding to a different logo. The images scraped were in jpg format, having 4 channels(RGBA) and were resized during training to a standard 256x256 resolution.

Figure 1: Scraped and refined car logos dataset

We had to perform a lot of manual filtering and cleaning of individual images for each class because the logo images were mixed with images of entire cars featuring logos on them, graphically edited logos or even entirely unrelated images that were merely scraped due to the matching search term. Doing this would ensure that only relevant and structurally sound pictures would be fed to the GAN.

2.2 Previous Work and Model Creation

2.2.1 Traditional Semi Supervised Learning with GANs

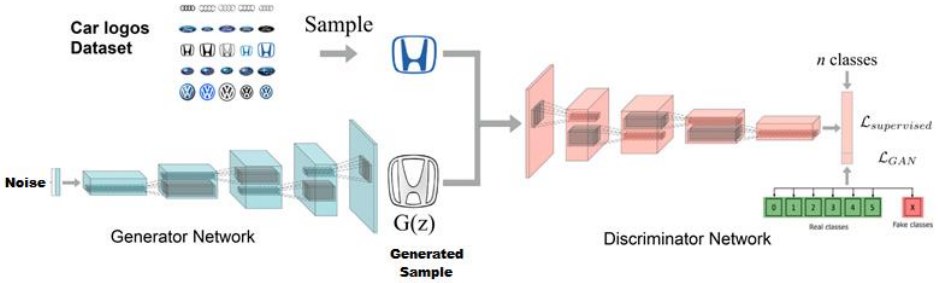
The original Semi Supervised GAN process as described in the referred paper[2] made use of the discriminator network as a multi-class classifier. This type of classifier would take a tiny portion of labeled data and a much larger amount of unlabeled data from the same data source. Pertaining specifically to our problem, the discriminator would be a classifier for 24 classes with 1 neuron (R/F neuron) representing the fake data output

(from the Generator) and the other 23 representing real data with classes. The discriminator needs to work with the following:

- Adjust Real/Fake neuron output label = 0, when receiving real unsupervised data
- Adjust Real/Fake neuron output label= 1, when receiving fake unsupervised data
- Adjust Real/Fake output label = 0 and corresponding label output = 1, when receiving real supervised data.

This combination of different sources of data help the discriminator classify more accurately than, if it had been only provided with a portion of labeled data[3].

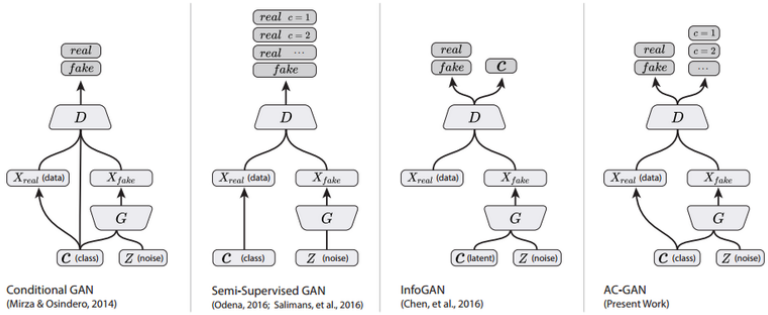
Figure 2: Traditional Semi Supervised GAN model for our problem



2.2.2 Auxiliary Classifier GANs

Conditional GANs were later developed to improve performance of Semi Supervised GAN based techniques by conditioning the generation process on something, say class. Wherein the CGAN approach uses the auxiliary class information alongside with partitioning the latent space, ACGAN improves the CGAN idea by introducing a classification loss which back-propagates through the discriminator and generator network. Much of our understanding of Auxiliary Classifier GANs(AC-GAN) has come from this paper[4]. The authors introduce a GAN architecture for generating high resolution images from the ImageNet dataset. This architecture makes it possible to split the generation process into many sub-models. GANs have trouble generating globally coherent images and that this architecture improves the coherence of generated samples. The approach of implementing an AC-GAN focuses on adding more structure to the GAN latent space[5] along with a specialized cost function that results in higher quality samples and that generating higher resolution images allow the model to encode more class-specific information, making them more visually discriminable than lower resolution images from a given data source.

Figure 3: Evolution of GAN models for Semi Supervised Learning



Source: https://becominghuman.ai/unsupervised-image-to-image-translation-with-generative-adversarial-networks

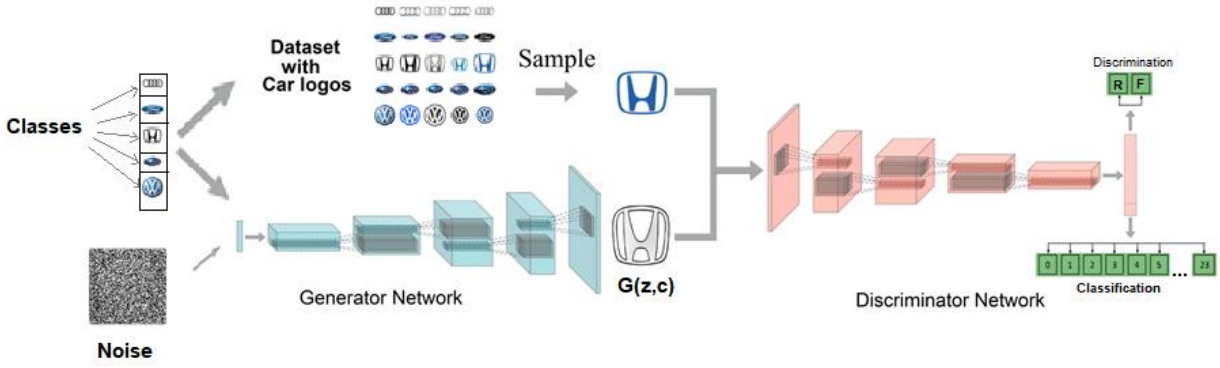
The AC-GAN model can perform semi-supervised learning by ignoring the component of the loss arising from class labels when a label is unavailable for a given training image. GANs often generate samples that are locally plausible but globally not realistic (e.g. a generated image of a dog has fur but the overall shape is not distinguishable). The basis of other GANs such as DC-GAN is to take a jumble of various images belonging to different classes as its input and to perform image generation considering the images of all classes rather than conditionally separating each class of images and performing generation for an image belonging to that class. For example, if we feed our data to a DC-GAN, it would generate new

images after taking into account all the different classes (logos/companies). The generated images would have all the features of each class, such as the little loops for Audi or the yellow colour in Chevrolet. By using an AC-GAN for learning we preserve the features of each class separately and use them for our image generation in a class-wise manner, so that say we generate images of the BMW logo, the only features to be considered would be "Circle", "Blue and White" and so on.

2.2.3 Forming a customised model and architecture

We implement the AC-GAN model shown above to our own data, making a few changes which are mentioned in detail under the "Experiments" section. Shown below is the AC-GAN model which will perform car logo classification for our data. The output of the generator would an image conditioned on class 'c' and the result of noise 'z'. So, let us define it as G(z,c). The image for the generated sample is a theoretical approximation of how a sample conditioned on the class of Honda logos should look like.

Figure 4: Architecture of an AC-GAN for car logos classification



2.2.4 Metrics for measuring performance

All generative models are probabilistic in nature. When we ask a generative model to generate a sample say an image, we are simply sampling from a probability distribution. This means that if we can compare the probability distributions of our *Generator* and that of our original data then we will have an evaluation metric. A kernel function can be used to define these probability distributions more accurately. We can take our generated samples from the *Generator* and see the probability of that sample being taken from the original distribution. Researchers have also utilized *KL-divergence* for comparing samples generated from two distributions[6]. We use the following metrics:

- Inception Score: uses two criteria in measuring the performance of GAN, the quality of the generated images, and their diversity. The formula is given as follows[7]:

$$\mathrm{IS}(G) = \exp\left(\mathbb{E}_{\mathbf{x}\sim p_a} D_{KL}(p(y|\mathbf{x}) \parallel p(y)) \right)$$

- Fréchet Inception Distance: use the Inception network to extract features from an intermediate layer and then model the data distribution for these features using a multivariate Gaussian distribution with mean μ and covariance Σ. The FID between the real images x and generated images g is computed as[7]:

$$\mathrm{FID}(x, g) = ||\mu_x - \mu_g||_2^2 + \mathrm{Tr}(\Sigma_x + \Sigma_g - 2(\Sigma_x\Sigma_g)^{\frac{1}{2}})$$

3 Experiments and Methodology

3.1 Resources and Technologies

We used a cloud GPU provided by SFU to train our model. The GPU was **Nvidia GTX Titan X** with the following specs:

- 3072CUDA Cores.
- 1000Base Clock (MHz)
- 7.0 Gbps Memory Clock.

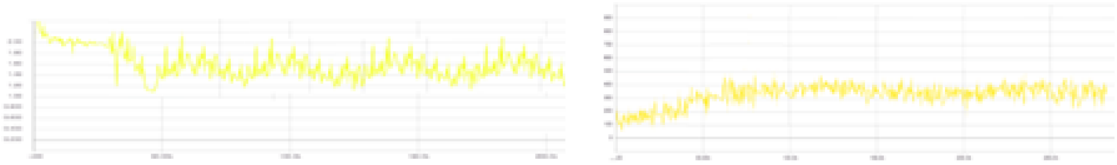
- 12 GB Standard Memory
- GDDR5Memory Interface.
- 384-bitMemory Interface Width.

3.2 Experimentation

3.2.1 Traditional SSL-GANs

The Tensorflow implementation[8] of GANs for Semi Supervised Learning is given as an open source project based on Goodfellow's paper[2]. We trained our dataset initially with this basic model twice and report the average accuracy.

Figure 5: The supervised loss (left) and classification accuracy(right) for SSGAN on our dataset



As we can see, the SSGAN does not perform very well, yielding a lower accuracy than if it were trained on datasets containing thousands of samples such as MNIST or SVHN. We next attempt to address the various implications arising out of GAN use and follow the best practices[8] that have been suggested to improve GAN performance.

3.2.2 AC-GAN model construction and implementation

We used Keras on Tensorflow to implement an AC-GAN(unoptimized). We acquired the base code from an open source project[9] which implements the referenced paper[4] by Augustus Odena from Google Brain. Our dataset consists of 23 classes, corresponding to logos of different companies where each class has exactly 100 images, so our dataset has a meagre 2300 images and hence train our network conditioned on class with **Adam optimizer**[10] for 2000 epochs. The high number of epochs compensates for the lesser amount of samples and we have a batch-size of 64(A matrix of 8x8 image samples). We changed the base code in a number of ways by adding new layers and altering several others. The detailed information is as follows :

Generator:

- Upscaled noise vector to 128*128*3
- Formed a weight vector from gaussian distribution having a standard deviation of 0.02

- **Layer1**:
  - Transpose convolution with kernel size 5x5 and strides 2,2 generating a matrix of 8*8*256
  - Batch normalisation with epsilon=1e-5, decay = 0.9
  - ReLU activation function to stabilise the output of each layer.
- **Layer2**: Transpose convolution with kernel size 5x5 and strides 2,2 generating a matrix of 16*16*128
- **Layer3**: Transpose convolution with kernel size 5x5 and strides 2,2 generating a matrix of 32*32*64
- **Layer4**: Transpose convolution with kernel size 5x5 and strides 2,2 generating a matrix of 64*64*32
- **Layer5**:
  - Transpose convolution with kernel size 5x5 and strides 2,2 generating a matrix of 128*128*3
  - Tan(h) activation function.

Discriminator:

- **Layer1**:
  - Convolution of input matrix to generate 64*64*3 matrix

- Each convolution layer is followed by a batch normalisation layer having epsilon=1e-5, decay = 0.9 except the output layer.
- Similarly, each convolution layer is followed by a Leaky ReLU activation function

- **Layer2**: Convolution of input matrix to generate 128* 128*3 matrix

- **Layer3**: Convolution of input matrix to generate 256*256*3 matrix

- **Layer4**:
  - Convolution of input matrix to generate 512*512*3 matrix
  - After this the matrix is multiplied with a weight matrix taken from the normal distribution with standard deviation = 0.02
  - The logits generated are passed through a sigmoid function.

3.2.3 Improvements to enhance performance

We address some of the ways in which our model could possibly fail and introduce possible solutions to achieve convergence:

- *Feature Matching*:

The logical thing to do while designing a GAN is to aim for a cost function such that the generator finds the best image which can be classified by the discriminator as real. However, while both generator and discriminator keep optimizing by competing against one another, the model might never converge due to the adversarial nature of the model. Feature matching changes the cost function for the generator to minimizing the statistical difference between the features of the real images and the generated images. For this model, we shall measure the L2-distance between the means of their feature vectors[11]. The new cost function is given by:

$$||\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\mathbf{f}(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\mathbf{f}(G(\boldsymbol{z}))||_2^2$$

- *Minibatch Discrimination*: This problem is solved by default in AC-GANs because the generator and discriminator are both conditioned on class from the beginning.
- *One sided label smoothing*: In general, deep networks use very few features to classify an object. If the discriminator depends on a small set of features to detect real images, the generator may just produce these features only to exploit the discriminator. To avoid this, we penalize the discriminator when the prediction for any real images goes beyond 0.9 i.e. (*D(real image)>0.9*). In other words, we change our target label value to be 0.9 instead of 1.0.
- *Balancing discriminator/generator loss*: We experiment by maintaining a 3:1 ratio between the number of the gradient descent iterations on the discriminator and the generator. The problem could be solved by a cost function that does not have a close-to-zero gradient when the generator is not performing well. For example:

$$\nabla_{\theta_g} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \rightarrow 0 \quad \textit{change to} \quad \nabla_{\theta_g} - \log\left(D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$$

3.3 Results

Table 1: Accuracy and Loss values for different models for training set

Description	Generator Loss	Discriminator Loss	Training Accuracy
SSL-GAN	-366.45	174.1	--
SSL-GAN (Minibatch Discrimination)	-135.22	171.82	--
SSL AC-GAN (with changed layers)	3.10	5.17	45.166
SSL AC-GAN (optimized)	3.14	5.28	52.000

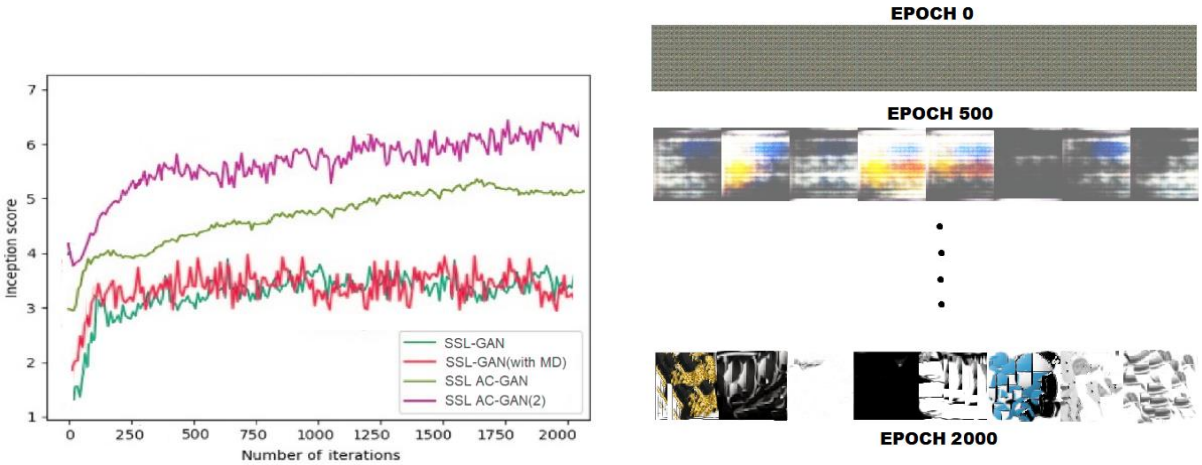


Figure 6: The inception score for all models tested(left) and the progression of images[8 samples] generated(right)

3.4 Problems faced

1) The biggest problem we faced was lack of proper resources for training our dataset. We did not have dedicated GPUs on our systems but were provided with a cloud GPU by the school. However, it was limited in its capabilities and very slow due to several people training their models on it at once and each user was allocated a mere 8% of the total 12GB memory.
2) The implementation of code was in TensorFlow and it was difficult and very time consuming to understand the usage and also to understand and reuse the source code we found for the basic models.
3) A lot of images in our dataset were unusable, so we had to do some manual filtering and refining, discarding outliers and hence this step was also very time consuming.
4) We faced mode collapse situation with the SSL-GAN where all the images being generated looked the same and only a few classes were being generated.
5) The generator and discriminator networks got stuck at a checkpoint after which there were no further changes in the images being generated.
6) Feature matching: As described by the research papers we read, we implemented features at the intermediate layers in the discriminator to match for real and fake images and make a supervisory signal to train the generator. We found training only based on this feature matching metric ineffective by itself contrary to what is mentioned in the paper - the discriminator attains almost zero loss right at the beginning.

4 Contributions

Our group consisted of only two people and furthermore there were no modular components. Hence there was equal contribution and effort among both members of our group, with each component being implemented jointly. The various efforts involved in our project were :

- Data Scraping and refinement
- Research
- Expanding and improving existing open source codes
- Layer adjusting and optimization
- Testing and Debugging
- Report and Poster

5 Conclusion

We successfully trained four different models with our custom generated dataset. We managed to generate fake samples from our dataset using an AC-GAN with optimization and modified layers. However as shown in the figure above, the quality of samples generated is far from satisfactory and easily distinguishable as computer generated. We observed the following :

1)Initialization - We were not able to get any result with xavier initialization or with random normal intialization of deviation 0.2. Looking at the scale of gradients it just made sense to start with much smaller weights so that

generator can start moving in the right direction from the beginning. We ended up using weights with random normal initialization with deviation of 0.02.
2)Batch normalization - These are supposed to help gradient flow but an improper decay or slow decay on the parameters used in batch norm is again going to hinder gradient flow. 0.9 seemed like the sweet spot.
3)The dataset having inputs that similar traits to some extent seemed important – For example, trying to create varied outputs like a Honda logo and a Tesla logo without doing a conditional generation is more complicated in terms of training. For experimental purposes start with a small dataset and see if the gradients and initializations are in the right place for the model to train.

We learned the following principles from our project:

- GANs allow the model to handle multimodal data well. For semi-supervised learning, features from the discriminator or inference net could improve performance of classifiers when limited labeled data is available;
- One can use adversarial nets to implement a stochastic extension of the deterministic Multi-Prediction Deep Boltzmann Machines;
- Conditional GAN models offer much better generated samples(image quality as described by the inception score) than traditional GAN classifiers, although the accuracy may be lower. This is because conventional loss metrics such as log likelihood only take into account the optimization of the network with regard to the cost function and hence ignore more subjective measurements such as generated sample quality.

6 References

[1] Goodfellow I. (2016), *NIPS 2016 Tutorial : Generative Adversarial Networks*[link]

[2] Goodfellow I. , Salimans T. , Zaremba W. , Cheung V., Radford A. and Chen X.(2016, June) *Improved Techniques for Training GANs*[link]

[3] Semi-Supervised Learning and GANs [link]

[4] Odena A. et al.(2017, June) *Conditional Image Synthesis with Auxiliary Classifier GANs* [link]

[5] AC-GAN and image synthesis [link]

[6] Ali Borji (2018, October) *Pros and Cons of GAN Evaluation Measures.* [link]

[7] GAN—How to measure GAN performance?[link]

[8] A Tensorflow implementation of Semi-supervised Learning Generative Adversarial Networks (NIPS 2016: Improved Techniques for Training GANs)[link]

[9] A tensorflow implementation of google's AC-GAN (Auxiliary Classifier GAN)[link]

[10] Andrew NG : Adam optimizer Algorithm[link]

[11] Mroueh et al.(2017, June), *McGan: Mean and Covariance Feature Matching GAN*[link]