Automatic Logo Generation with Conditional Generative Adversarial Networks

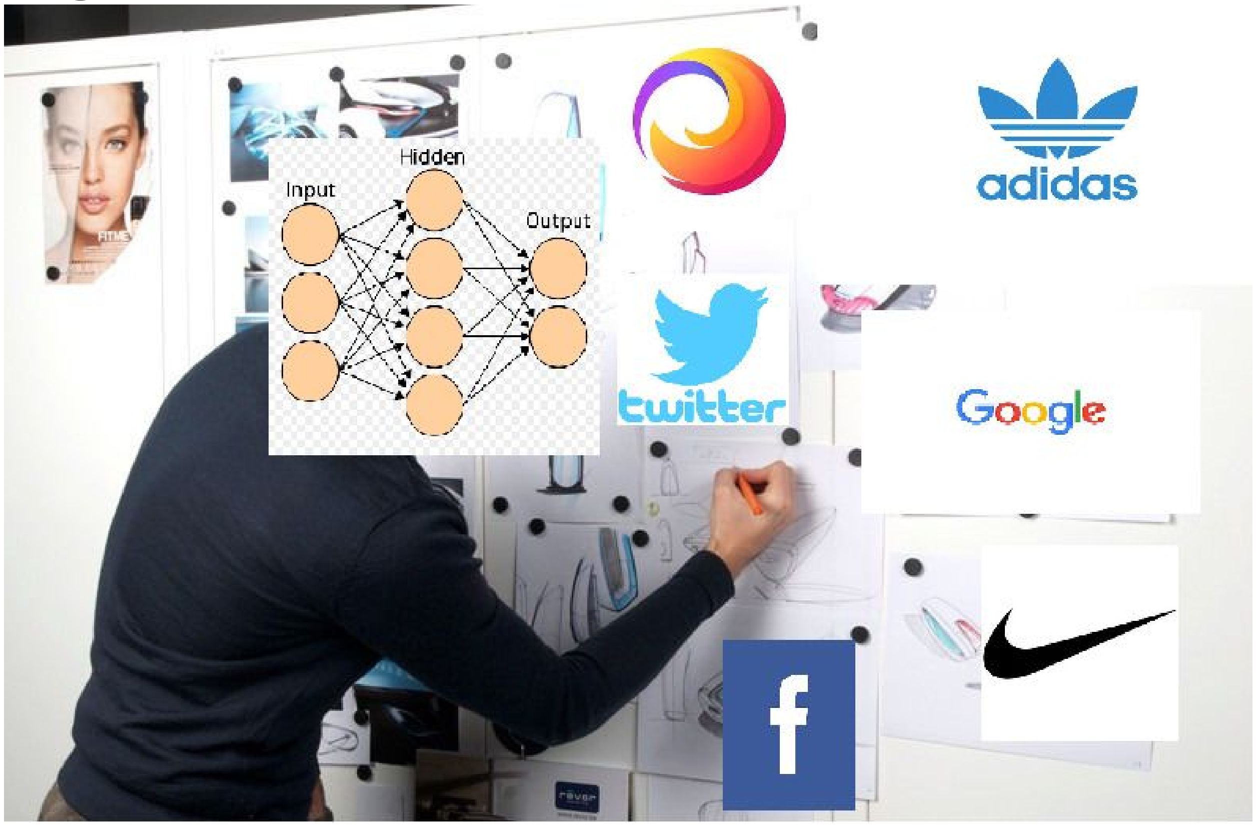
Abstract:

This project focuses on leveraging the strength of generative adversarial networks to generate a logo which can be used as an inspiration for a designer or an artist. In today's world, designers often face the issue of getting unique ideas and inspiration to create an aesthetic and good logo design. Additionally, the artists and designers are provided a set of constraints which pertain to the type of company, what it stands for, their color scheme. We can refer to these constraints as conditions and thus present a method that aids logo design based on conditioning from input words entered by the user.

We utilize an unsupervised deep learning approach i.e. Generative Models to synthesize logos. Specifically, we are building different variants of Generative Adversarial Networks a.ka. GAN's to generate meaningful logo designs with supervised conditioning.

We share the results from both an unsupervised random vector input to the model and compare and contrast it the results from a conditioned supervised approach. What can be observed is that the random vector generated results train faster than the supervised method and converge to human interpretable and aesthetically pleasing results faster than the supervised approach. Finally, we discuss possible extensions of this project using a reinforcement learning approach that could incorporate elements of both the supervised conditioned and unsupervised approaches.

Teaser figure:



Introduction:

Designing a logo is a repetitive task, it can take a designer several iterations to iterate over multiple designs and feedback of a company. Some of the most iconic logos that we take for granted today have a long history of multiple iterations and improvements based on back and forth between the designer and the company. [15] The generation process can be simplified as starting from random noise and slowly iterating and conditioning the logo design based on feedback and inputs. Which is a process that can be modelled by recent advancements in artificial intelligence, particularly Generative Adversarial Networks (GAN's).

The main aim of the project is to aid both designers and companies with generating logos. Starting with random noise and conditioning based on inputs allows a much more streamlined approach towards getting to the perfect logo. Our work will help them with a meaningful logo design based on given context which they can further build upon and generate an icon that we take for granted today.

To accomplish this task we leveraged deep learning approaches in the domain of computer vision, particularly in the domain of Creative AI and art generation and adapt them to the task of generating synthetic logos. We will leverage the Large Logo Dataset (LLD)[6] to build our model. This dataset has labeled 600k+ logos scraped from twitter. Using this dataset, we are building Generative Adversarial Networks to synthesize novel logo designs.

We also provide semantic information to synthesized logo by using logo labels information from LLD dataset. We use a conditional DCGAN approach for this problem where the logo label will be a condition to the GAN architecture.

For the generated system, the expected input in the system from the user is a dictionary word with an actual meaning in order to generate a logo design as an output which corresponds to the input words. This can easily be extended further with additional training so that the input is a bag of words (bow), that is multiple words based on which a logo design is generated.

Related Work:

Introduction to generative models:

Generative models are the state of the art of neural network advances. They provide the basis behind what makes this project possible. The two most common types of generative models that currently make up the state of the art of generative models that currently make this possible are known as Variational Autoencoders or (VAE) and Generative Adversarial Networks (GAN).[3] These networks can be understood as two players in a non zero sum game. They consist of a generative model which is trying to trick the decoding model into whether or not an output is an actual input or not. Both of these models are working with images in a high dimensional or 'latent space' and therefore with nudging by the user they are able to modify the output of the image. Through the targeted modification of images in the latent space, these generative networks produce new art in the form of logos based on our inputs.These models have been used frequently to generate fake art for example and could be conditioned based on a specific artists style or type of fake art that is wanted. We have extended this technique and presented it in a novel way as logo generation based on conditioning. This is discussed in much greater detail within our approach section. One of the major drawbacks of generative networks is that the generative part of the network which produces images from their latent form, ends up producing blurry images due to the tendency to generalize the input from the latent space. To produce crisp images GAN's need several modifications based on the task at hand are notoriously difficult in determining the right parameters.

GAN's have progressed slowly in very creative ways to produce results that make this project a possibility. Initially visually blending emoji's by Martin et. al [12] and then progressing to blending animals together [13] have several parallels to the goals of blending logos to create new logos. Similarly recent work regarding converting sketches to logos also provides us a basis from which to start building our GAN. Autodraw from Google which uses an RNN to create a sketch of an image from strokes has also gained lots of popular attention due to the generative ability of the RNN to help create interesting art from single strokes. [14]

Conditional Models:

Conditional models are an extension of GAN's in which class conditional outputs are required, thus adding a supervised element to an unsupervised model.[4] For example, when an encoder is provided a real image of a face but with class attributes of types of faces. It is thus possible to modify the output of the face with different inputs or classes. Other examples include creating images from text descriptions or high level descriptions of a scene. After we achieved reasonable outputs from our unsupervised GAN we began training it with conditioned words from a dictionary as inputs.

LLD Dataset:

Other notable related work that are key to this project is the Large Logo Dataset (LLD) which is a compilation of 600,000 logos scraped from twitter. Without this [11]

Reinforcement Learning with Generative Adversarial Networks:
As discussed above generative models have been successful in producing high quality images in the domain of natural images such as faces, paintings, bedrooms and other creative applications. However, for our task of generating logos, training a generative model to directly synthesize the logo is significantly more difficult since the logo needs to make sense as well as be conditioned based on the wants/desires of a company or designer. Despite the success of these models in generating the target entity, there are usually minor artifacts in the output. These artifacts are not noticeable in rich images but in the case of logos, a clear and distinct logo is preferred. They are usually comprised of easily composable shapes, colors and text. If the model had been constrained to just utilize these shapes / text objects to generate images, then the resulting logos would be less likely to contain artifacts.

SPIRAL [10] is a reinforcement learning based model that generates images in the 2D (or 3D) space using a blackbox rendering engine. It produces instructions that are passed onto the rendering engine which executes them and produces the output image. These images are naturally constrained to utilize the tools within the engine to produce the image. The SPIRAL architecture has been experimented on the MNIST (handwritten digits dataset), OMNIGLOT (handwritten characters dataset), CELEBA(celebrity faces dataset) and 3D images from MUJOCO.[16]

Contributions:
In this space our main contribution is to create a GAN that is able to successfully manipulate the latent space, additionally it can give designers a starting point by sending multiple random noise vectors to generate new logo designs which can be a basis for further refinement and creating the next best logo. We then improve upon the unsupervised approach with conditioning which helps create logos based on dictionary words. Our future work on this topic would be focused on extending the reinforcement learning approaches described above to create seamless, perfectly shaped logos.

Approach:
This project focuses on the unsupervised generation approach first since if no condition is provided, GANs produce random images based on random noise input vector and real training images used for the discriminator. Next the unsupervised approach is extended using conditioned approaches and reinforcement learning techniques.

Unsupervised Learning Approach
We used unsupervised generative models to create logos given a class of that logo as a condition. Our approach can be summarized in two layers. In the first layer we initialize word embedding to represent classes in the form of a vector. In the second layer, we build GANs focusing on one of its extensions popularly called as conditioned GAN’s. We are using word embedding as a class-condition information while building the network.

There has already been work done in space of unsupervised approach to generate logos using word and theme information [1]. But the previous work is based completely on Autoencoders. Though producing amazing results, the generated logos are restricted to the training sample space and lacks generation of novel logos. In our approach we first generated random completely new logos based on random vector inputs to GAN’s rather than autoencoders. Next we incorporated logos classes to build conditional GANs and produce class specific labels. We tested several different GAN architectures and found that for less pixelation we needed the GAN to be deep as well as use several stabilizing techniques such as using a wasserstein loss function[17] and incorporating gradient penalty [18].

Our GAN's were conditioned using the LLD dataset on over a hundred thousand labelled logos and used several GPUs to get meaningful speed. To increase loading speed we formatted png images into HDF5 format of the dataset, which made reading and working on the dataset relatively easier.

For logo class, we are using cluster labels generated through clustering the embeddings obtained from ResNet model to generate logos [6]. Different clustering results are available in the LLD dataset. We are using the results with 128 cluster labels to replicate the work that has been done by A. Sage [6].

Although we took inspiration of GAN architecture from the work by A.Sage we built our own architecture of GANs that would reduce pixelation and likely converge faster through gradual upconvolution and careful use of stride and padding.

We propose our own unique functions for the Generator and Discriminator which was finalized by making several tweaks and logging each tweak. Some of the notable aspects of our GAN architecture experiments are noted below:

1. Varying number of convolution layers and fully connected layers and their sizes
2. Batch normalization
3. Varying activation functions across different layers
4. Drop out
5. Data preprocessing techniques such as random horizontal flip, affine transformation, image resizing etc.
6. Testing various loss functions and understanding and implementing Wasserstein Loss.
7. Adding gradient penalty.

We have described each of these in detail below and in experiments and results section. The overall architecture of GAN can be summarized in the below plot.

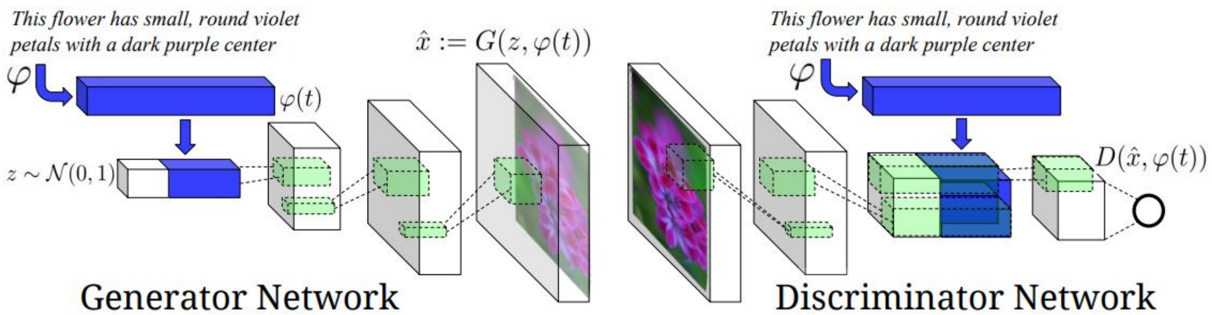

This flower has small, round violet petals with a dark purple center
φ
$\varphi(t)$
$z \sim \mathcal{N}(0,1)$
$\hat{x} := G(z, \varphi(t))$
Generator Network
This flower has small, round violet petals with a dark purple center
φ
$D(\hat{x}, \varphi(t))$
Discriminator Network

Figure 2. Our text-conditional convolutional GAN architecture. Text encoding $\varphi(t)$ is used by both generator and discriminator. It is projected to a lower-dimensions and depth concatenated with image feature maps for further stages of convolutional processing.

Conditioned GANs

We have resized all our input images to 32 x 32 size to be approximately the size of an icon and also lead to faster processing through this large generative network. As shown in the figure above, we have two inputs to the Generator model and 2 inputs to the discriminator model. For generator, we feed one dimensional random input vector of size 100 and a cluster label in the form of embedding of size 50. This cluster label serves as a condition to the GAN architecture. The generator network produces a fake image which is fed to the discriminator along with real images and their class labels to differentiate between real and fake images.

Generator Architecture
The cluster labels, obtained via clustering embeddings from ResNet, is first passed through an embedding layer of size 50. This means that each of the 128 cluster label maps to unique 50-element vector representation that will be learned by both generator and discriminator. The output of the embedding is then passed to a fully connected layer with a linear activation. The activations of the fully connected layer are resized into a vector of size 100 feature map. This is to match the feature map activations of the unconditional generator model which would have used only random input vector of size 100 as an input to generate the image.. The new feature map obtained is added as one more channel to the existing input feature map, resulting in feature maps that are then upsampled as in the prior model.

The updated feature map is then passed through convolutional layers. We are using 4 convolutional layers in the Generator network each followed by a batch normalization, except the last layer. We start with changing input feature map to 1024 dimensional vector and reduce the vector size after each convolution from size 1024 to size 3. The model uses best practices in literature for generator architecture such as the ReLU activation, a kernel size that is a factor of the stride size, and a hyperbolic tangent (tanh) activation function in the output layer. For our project, we have used kernel size of 4 and stride of either 1 or 2. In addition, we also use relu activation function after each convolutional output except the last layer where we use tanh activation function. We thought tanh would be a better candidate because a bounded activation could allow the model to learn more quickly to saturate and cover the color space of the training distribution. It could be that the symmetry of tanh is an advantage here, since the network should be treating darker colours and lighter colours in a symmetric way.

Discriminator Architecture
Similar to the generator defined above, a new second input is defined that takes an integer for the cluster label of the image. This has the effect of making the input image conditional on the provided cluster label. The cluster label is then passed through an Embedding layer with the size of 50 to get a 50 dimensional embedding. The output of the embedding is then passed to a fully connected layer with a linear activation. Importantly, the fully connected layer has enough activations that can be reshaped into one channel of a 32×32 image. The activations are reshaped into single 32×32 activation map and concatenated with the input image. This has the effect of looking like a two-channel input image to the next convolutional layer.

Therefore, the discriminator model has two inputs: : first the class label that passes through the embedding and the image, and their concatenation into a two-channel 32×32 image or feature map. This feature map is then passed through convolutional layers to be upsampled. For Discriminator network, we are also using 4 convolutional layers and each convolutional layer is followed by batch normalization. However, we are using leaky relu as the activation function after each convolutional layer instead of simple Relu as in Generator network. Leaky Relu allows small gradient signal for negative values. As a result, it makes the gradients from the discriminator flows stronger into the generator. Here, we go reverse and start with input vector with 3 input channels and gradually expand it to 1024 channels using convolutional layers. The kernel size and stride is 1 and 1/2 respectively, same as Generator network keeping in mind the best practices in literature for discriminator networks.

We have further elaborated our approach in experimental set up in the section below. While building the GAN model, the biggest challenge we faced was training of the model. We started by training model on our CPU’s. It was taking more than a day to complete a single epoch. We then trained the model using GPU’s. However, we had limited free GCP credits to train our model using GPU’s. Though, the speed of training greatly increased but we were limited in terms of the free credits we had available to run our experiments.

Experiments:

Train on clusters -

For this experiment we tried training the images in the clusters provided by the Large Logo Dataset. This was done because clusters tend to have similarity in structure. However, this did not produce adequate results as shown below.

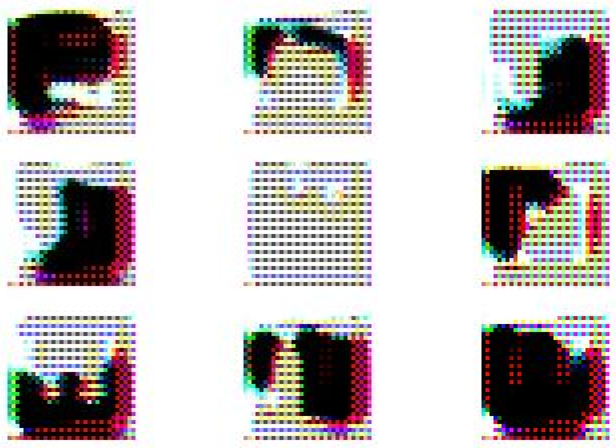


Figure 3: Final results for clustered logos as inputs. Even after extensive training do not provide meaningful results

Gradient Penalty -

Several experiments show that gradient penalty used in place weight clipping works better and produces better results. So, we decided to try this first. Gradient Penalty penalizes the model if the gradient norm moves away from its target norm value 1. However, from the results shown below if can be seen that even this produces results that don’t resemble logos. So, we decided to move forward with weight clipping.

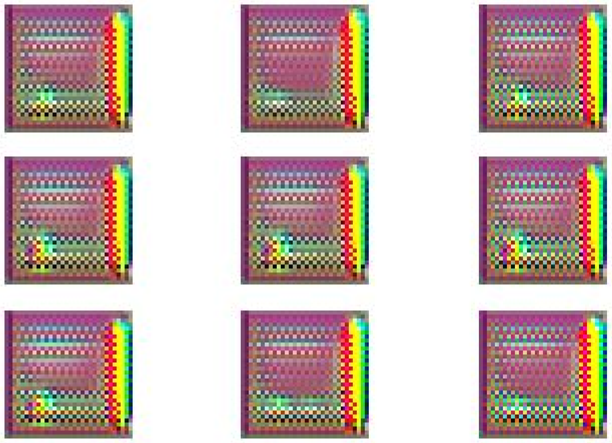


Figure 4: Even after incorporating a gradient penalty, meaningful results are not derived from the model.

Weight Clipping -

We decided to try weight clipping since Gradient Penalty didn't work very well. It is applied on the discriminator. After every weight update, the weights of the discriminator are clipped to be in range (-0.01, 0.01). It reduces the capacity of the discriminator. It behaves as a weight regulation. With weight clipping it can be seen that the results after training finally resemble logos. So, we decided to proceed with weight clipping instead of gradient clipping.

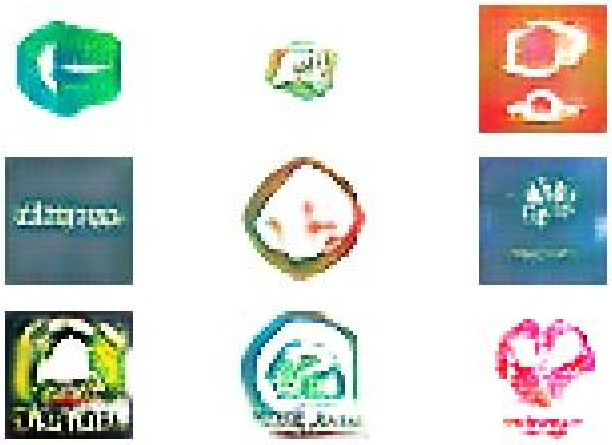


Figure 5: Weight Clipping provided much better results, starting to resemble logos and patterns with little pixelation.

Deeper Network -

We experimented with increasing the layers in the network with the intention that the deeper network will learn more of the structure of logos and produce better along with the gradient clipping added before. However, the network fails to produce better results than the less deeper network**.**

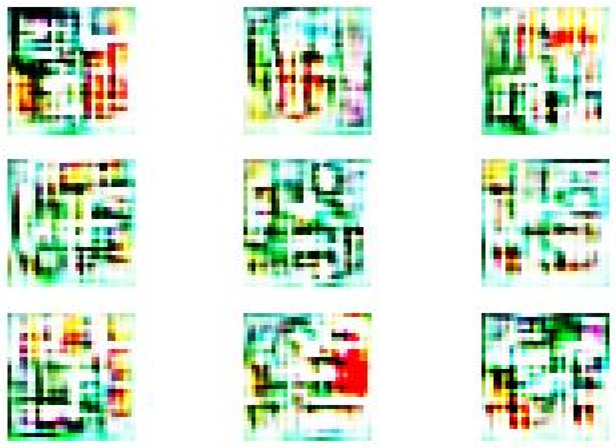


Figure 6: Final results from a deeper network, showing that a larger network does not affect the results

Interpolation -

The previous network used a Conc2DTranspose layer from pytorch for upsampling. This produced a checkerboard pattern. We found that using interpolation could reduce the checkerboard pattern. This does happen but after many epochs of training the results still don’t look like logos as can be seen from the figure below.

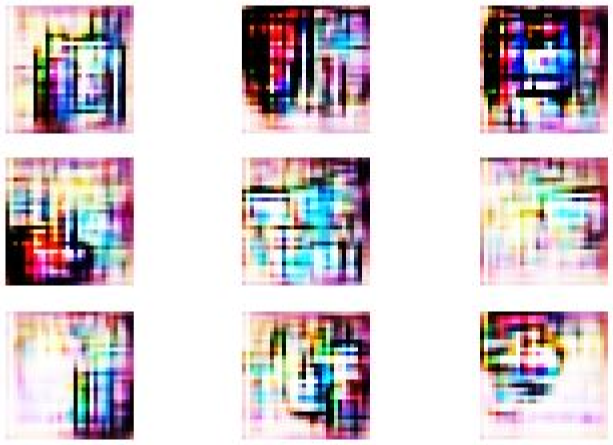


Figure 7: Interpolations, also does not seem to provide any improvement and similar to a larger network also leads to a degradation in results.

Results:

We setup a Wasserstein GAN using a standard Generator (Up Convolutional layers with BatchNorm) and Discriminatior (Convolutional layers with BathNorm) models inspired fromDCGAN style architecture. The architecture of both Generator & Discriminator has been described in approach section above. We train the Generator for 2 iterations for every iteration of Discriminator training. We use ADAM Optimizer for the setup using a learning rate of 1e-4.

The Dataset size is 120,000 images of logos. Since, we are using unsupervised approach, we have not split our dataset into test and train. We are using complete dataset to train the model. However, we realized that there were no cluster labels for few 1000 images. We removed these images from the training data. Hence, our training data comprises of all the images and cluster labels obtained from clustering the embeddings obtained from ResNet. After training, we use a random normal input vector and a cluster label to generate novel images from the given cluster.

For Discriminator, the loss is of two types i.e. real loss and fake loss. The real loss is simple obtained by passing real images through Discriminator network and then taking the mean of the outcome. Similarly, the fake loss is obtained by passing fake images i.e. the ones generated by Generator network through the Discriminator architecture. We finally take the mean of the output to get the loss. We sum up the two losses i.e. real loss + fake loss to get total loss and then use Adam Optimizer to optimize our loss function.

For Generator network, we simply pass the generated fake images through discriminator model and take negative of the mean of the output as generator loss.

Below are the loss curve for the conditional GAN training.

Generator Loss :

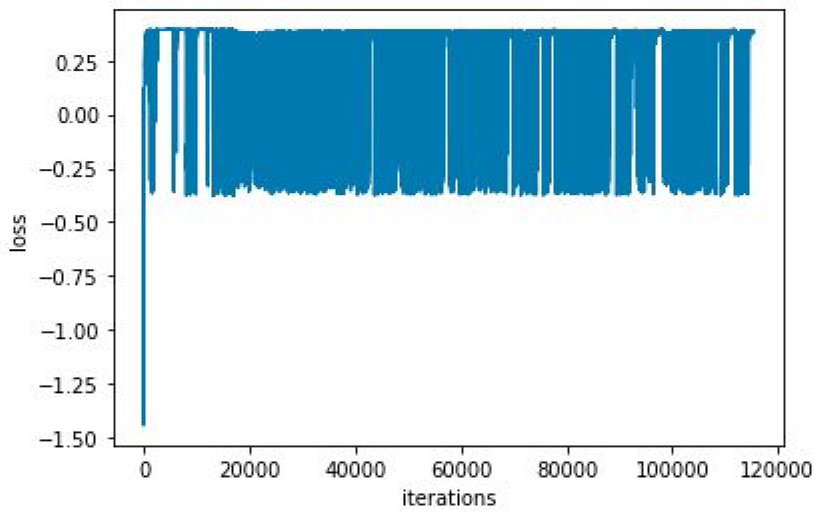
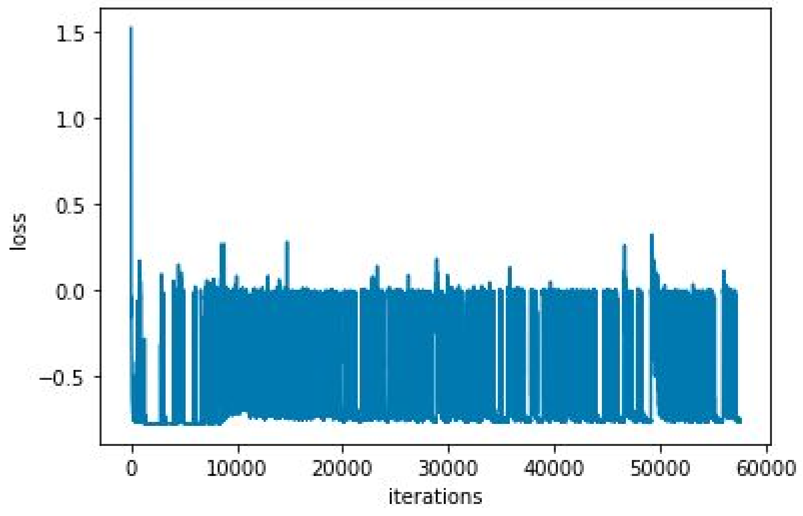


Figure 8: Generator Loss curve is unpredictable but the network does learn

Discriminator Loss :



Our evaluation metric for now is to print a few generated logos every 10 iterations of training in order to inspect what the GAN is learning and monitor the Inception score.

The obvious baseline in our case would be to simply generate image from random normal vector without passing it through any Generative models. We have displayed an output of random input vector below.

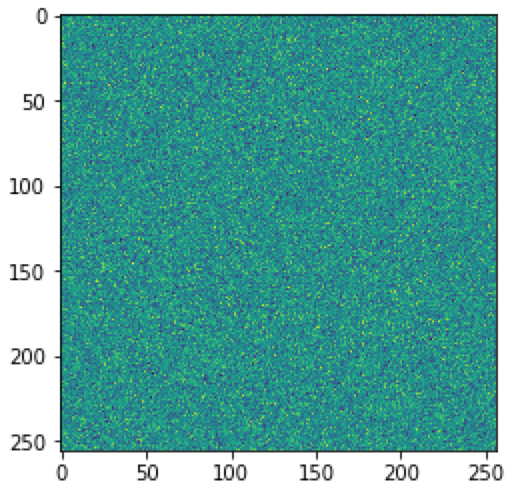


Fig 9: Noisy Image from Random Input (Obvious Baseline)

As you can see the most obvious baseline is all noisy random image. We are building a generative model which would improve on the noisy input.

Inception score :

The Inception Score is an objective metric for evaluating the quality of generated images, specifically synthetic images output by generative adversarial network models. The higher the inception the better the results and more aesthetically pleasing. From the plot it can clearly be seen that the inception score is increasing as the number of iterations increase. This shows that the GAN produces better results over time during training.

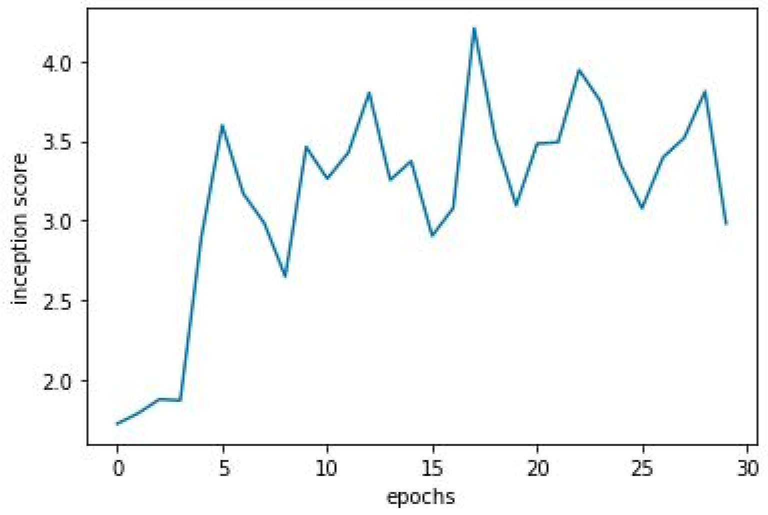


Figure 10: Inception scores increasing over time with more epochs.

There are multiple hyper parameters for our learning algorithm. To mention a few, here are a few of them:

Learning Rate - The learning rate is one of the most important hyperparameter. Small changes to learning rate can cause large changes in the training network. It's recommended to use higher learning rates when we have large batch sizes. However, currently we are using a fixed learning rate of 1e-3. In future, we plan to schedule our learning rate and see if we can get some improvement using something like exponential learning rate decay.

Batch Size - Batch size is the number of images that the generator or discriminator network will try to learn and differentiate in one iteration. The images in the batch are randomly chosen from the pool of full set of images. The choice of batch size is crucial since it directly affects the training. Large Batch sizes are recommended for stabilized GAN training. Currently, we are using batch size of 32 which is theoretically recommended.

Stride - The stride defines the step size of the kernel when traversing the image while making convolutions. We use stride in convolutional layers to perform downsampling in the discriminator model. Similarly, fractional stride (deconvolutional layers) is being used in the generator for upsampling. For our network, we are using stride of 1 or 2 for both generator and discriminator.

Kernel Size - The kernel size refers to the (width, height) of the filter mask while performing convolutions. The kernel size defines the output of each convolutional layer and it is helpful in spatial upsampling and downsampling with Discriminator and Generator Networks. We have used kernel size of 4 for both frameworks.

Loss Function- The loss function of a GAN determines how much the input image differentiates. For our GAN networks there are two types of loss functions, real and fake losses that are deviations from real and fake images. We found that a specific type of loss that stabilizes GAN's known as Wasserstein Loss.

Wasserstein-GAN- This is a change to a classical GAN discriminator to classify or predict the probability of generated images as being real or fake, the WGAN changes or replaces the discriminator model with a critic that scores the realness or fakeness of a given image.
This change is inspired by a mathematical explanation that training the generator should seek a minimization of the distance between the distribution of the data observed in the training dataset and the distribution observed in generated examples. The argument contrasts different distribution distance measures, such as Kullback-Leibler (KL) divergence, Jensen-Shannon (JS) divergence, and the Earth-Mover (EM) distance, referred to as the Wasserstein distance. [17]

Gradient Penalty - Instead of applying clipping, WGAN-GP penalizes the model if the gradient norm moves away from its target norm value 1. Batch Normalization is avoided for the discriminator also known as critic but a lot of additional computational complexity is added to the model. Gradient penalty allowed us to definitely produce clearer and crisper logos however, it slowed down training further and caused us to reduce batch size to 8.

There are several other parameters such as size of convolutional and fully connected layers, different optimizers such as RMSProp vs Adam etc. which we have tried. We have mentioned our final parameters in the approach section.

There were multiple things that we have tried as discussed above. However, visually we found a difference with use of Batch Normalization while training Generator and Discriminator networks. Batch Normalization helps by standardizing the activations from previous layer to have zero mean and unit variance. This helps in stabilizing the training process. Before using batch normalization in any of the networks, following was the output that we got after running our model for 100 iterations.

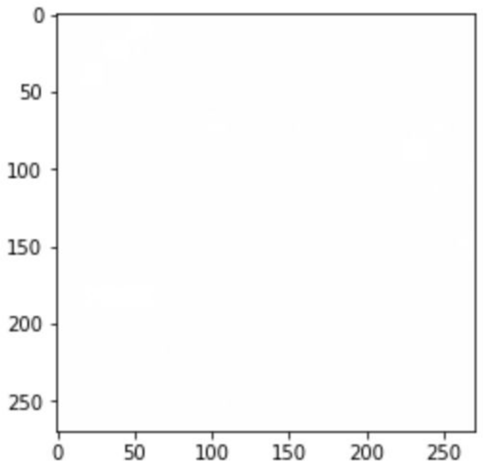


Fig 11: Output Image from Random Input after training GAN without Batch Normalization for 100 iterations

As we can see the generated logo from random normal vector without using Batch norm gives completely blank image after 100 iterations. We looked into results of first 9 iterations, which is displayed below.

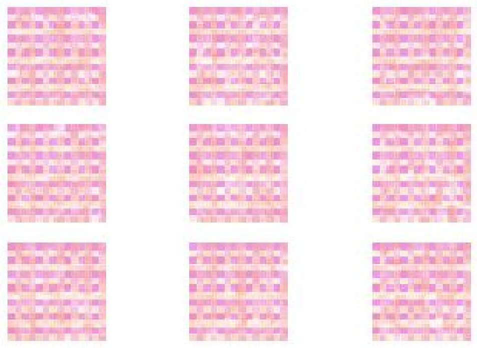


Fig 12: Output Image from Random Input after training GAN without Batch Normalization for 1-9 iterations

We got some light patches for first 9 iterations but as we further train we only got blank output until next 3000 iterations.

We then used Batch norm layers in both the discriminator and generator models, except the output of the generator and input to the discriminator. Below is the result that we obtain from running the model for 100 iterations.

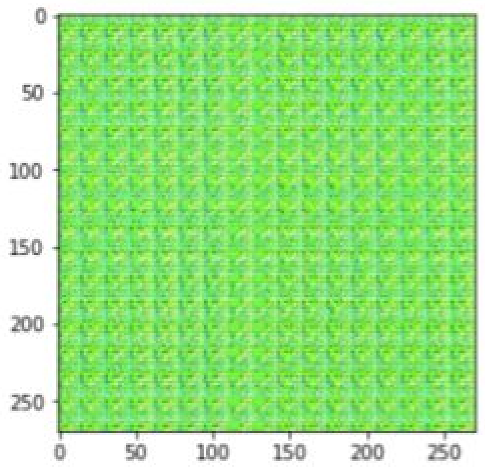
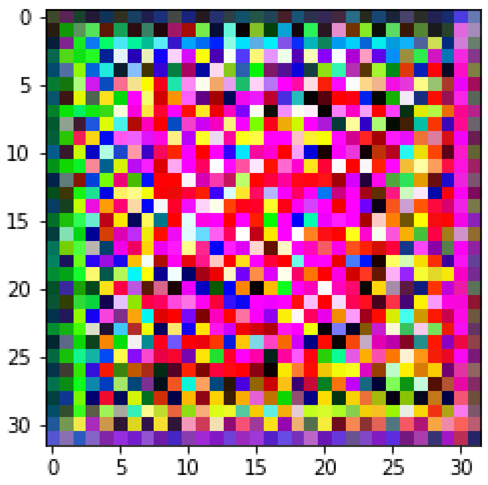


Fig 13: Output Image from Random Input after training GAN with Batch Normalization for 100 iterations

As can be seen, batch normalization gives us colorful patches much before the architecture without batch normalization. This is because it helps in stabilizing GAN training.

Initially, we got very patchy / grid-like results as shown below.



We believe this was due to the insufficient training of generator network. Initially, we were training the discriminator network more than the generator network. However, when we switched i.e. trained generator twice for every iteration of discriminator, we got smooth images after 4 epochs.

Qualitative results

Show several visual examples of inputs/outputs of your system (success cases and failures) that help us better understand your approach.

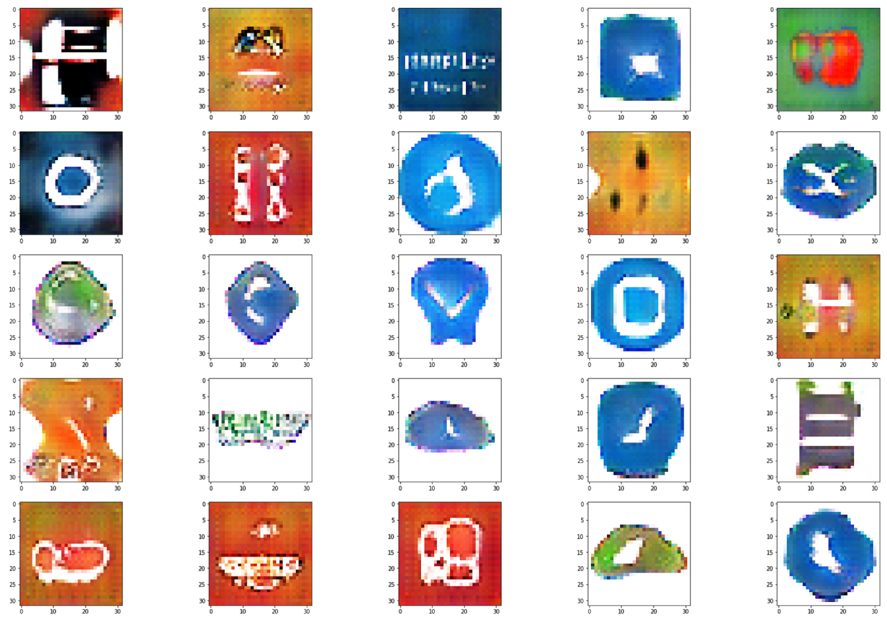


Fig 14: Output Image from Random Input after training GAN for 50 Epochs. Each logo generated is from same random noise but with different cluster labels labelled 1 to 25

The logos displayed above were generated from the same random normal input vector after training it for 50 epochs. Each label starting from top right to bottom left corresponds to cluster label from 1 to 25.

We also have tracked how our generated images change when we keep on training for a longer time. Below are the nine images that we generated when trained for first nine iterations of 1st epoch. Since the model has just started learning. We don't see a reasonable output yet.

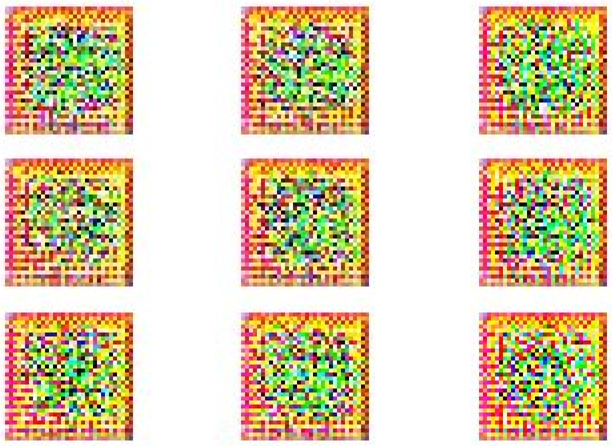


Fig 15: Output Image from Random Input after training GAN for 1-9 iterations (Epoch 1)

However, when we continue our training, we start getting some reasonable output after 6th Epoch.

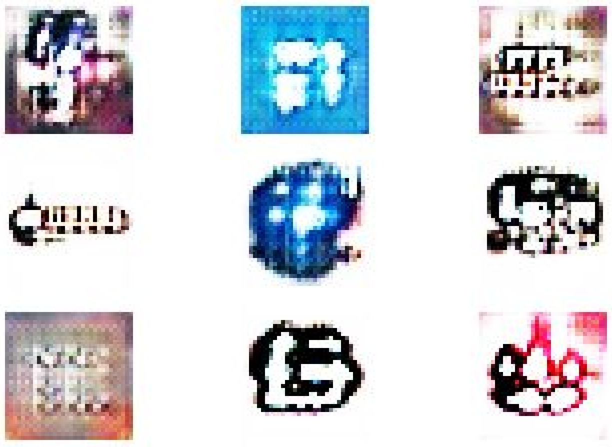


Fig 16: Output Image from Random Input after training GAN for 6 Epochs

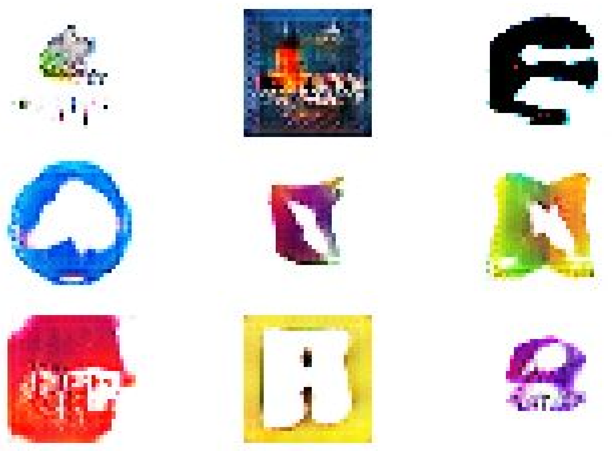


Fig 17: Output Image from Random Input after training GAN for 32 epochs

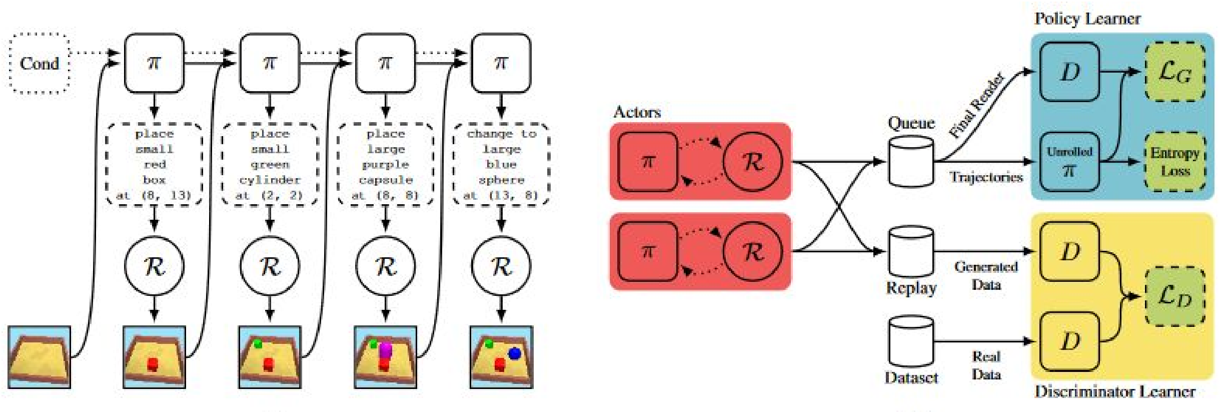
As we can see, training our model on the 32nd epoch for more number of iterations, gives us more colorful logos. However, the shape is a little distorted and more random than we would like. On further training until 50th Epochs, we got good labels for each cluster. Since, there are 128 clusters it's difficult to show examples from all of them. Also, the clusters do not have any semantic meaning. Therefore, it's difficult to deduce meaning of logos produced for each cluster label. To improve further dictionary word conditioning would be helpful which is discussed further in the next section.

Conclusion and future work:

In conclusions are results show that with sufficient training and resources, unsupervised and conditioned approaches can lead to exciting results, generating completely novel logos based on user inputs. There is still lots of improvement that's possible by conditioning based on user defined inputs, however, we found from the experiments that conditioning such as conditioning with dictionary words increases the complexity significantly and risks needing exponentially greater examples as well as much larger training resources. A possible way to reduce this challenge could be using the reinforcement learning training of GAN's described in the next section:

Reinforcement Learning as further improvement

This architecture consists of reinforcement learning agent that aims to learn to produce instructions to the rendering engine. These instructions are executed to produce the image. This image is passed to a discriminator network to determine if the image was generated by the agent or from the ground truth image (fake or real). The pixel wise reconstruction loss of the image along with the discriminator output together act as the reward signal for the reinforcement learning agent. This allows the reinforcement learning agent to learn in an unsupervised fashion instead of requiring pairwise labels between the instructions and images.



Cond
place small red box at (8, 13)
place small green cylinder at (2, 2)
place large purple capsule at (8, 8)
change to large blue sphere at (13, 8)
Policy Learner
Actors
Queue
Final Render
Trajectories
Unrolled
Entropy Loss
Replay
Generated Data
Dataset
Real Data
Discriminator Learner

(a) (b)

- <u>References</u>: List out all the references you have used for your work.

[1] Tendulkar P.; Krishna K.; Selvaraju R.' and Devi Parikh, L. 2019. T*hematic Reinforcement for Artistic Typography*
[2] Sage, A.; Agustsson, E.; Timofte, R.; and Van Gool, L. 2018. Logo synthesis and manipulation with clustered generative adversarial networks. *In CVPR*
[3]Zhang, Z., Zhang, R., Li, Z., Bengio, Y., & Paull, L. (2019). Perceptual Generative Autoencoders. *arXiv preprint arXiv:1906.10335*
[4]Mirza, Mehdi, and Simon Osindero. "Conditional generative adversarial nets." *arXiv preprint arXiv:1411.1784 (2014).*
[5]Wikipedia contributors. "Structural similarity." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 10 Aug. 2019. Web. 5 Oct. 2019.
[6]A. Sage, E. Agustsson, R. Timofte, and L. Van Gool, "Logo synthesis and manipulation with clustered generative adversarial networks," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018.
[7]
https://medium.com/@jonathan_hui/gan-how-to-measure-gan-performance-64b988c47732
[8] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. Generative adversarial text to image synthesis. In M. F. Balcan and K. Q. Weinberger, editors, Proceedings of The 33rd International Conference on Machine Learning, volume 48 of Proceedings of Machine Learning Research, pages 1060–1069, New York, New York, USA, 20–22 Jun 2016. PMLR.
[9]H. Zhao, O. Gallo, I. Frosio and J. Kautz, "Loss Functions for Image Restoration With Neural Networks," in *IEEE Transactions on Computational Imaging*, vol. 3, no. 1, pp. 47-57, March 2017.
[10] Ganin, Yaroslav, et al. "Synthesizing programs for images using reinforced adversarial learning." *arXiv preprint arXiv:1804.01118* (2018).
[11] Sage. "Large Logo Dataset." *LLD - Large Logo Dataset,* https://data.vision.ee.ethz.ch/sagea/lld/.
[12]Martins, P.; Urbancic, T.; Pollak, S.; Lavrac, N.; and Cardoso, A. 2015. *The good, the bad, and the aha! blends.* In ICCC
[13] Martins, P.; Cunha, J. M.; and Machado, P. 2018. *How shell and horn make a unicorn: Experimenting with visual blending in emoji.* In ICCC.

[14] Ha, D., and Eck, D. 2018. A neural representation of sketch drawings. In ICLR.
[15]Think Marketing, Story of the Evolution of the Apple Logo. *Accessed October 23rd 2019, https://thinkmarketingmagazine.com/apple-logo-evolution-story/*
[16] Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." *arXiv preprint arXiv:1509.02971*(2015).
[17] Arjovsky, Martin, Soumith Chintala, and Léon Bottou. "Wasserstein gan." *arXiv preprint arXiv:1701.07875* (2017).
[18] Gulrajani, Ishaan, et al. "Improved training of wasserstein gans." *Advances in neural information processing systems*. 2017.