Visual Program Synthesis

Undergraduate Project- CS498A

Hritvik Taneja 160300 **Divyat Mahajan** 14227

Mentor:, Prof. Vinay Namboodiri

Abstract

The task of automatic program generation is quite an important problem in AI and has received attention since the early days of AI Research. Traditional rule based methods for program synthesis tend to be ineffective due to a high amount of initial specifications required to generate program. Also, they are not robust to noisy input specifications data. With the recent surge in Deep Learning and its success at tasks like machine translation, image captioning, etc., researchers have started to experiment with Deep Learning based models for the task of program synthesis. Recently, some Deep Learning based models have achieved far better results than the rule based methods. In this project, we propose a Deep Generative framework for the same. Our model uses Conditional GAN to generate image described by the input caption and combines it with Attention based LSTM to generate the Logo program. Our model learns to generate both good synthesized images from Generator and syntactically and semantically correct code sequence for input captions describing geometrical figures like circles, polygons, etc.

Contents

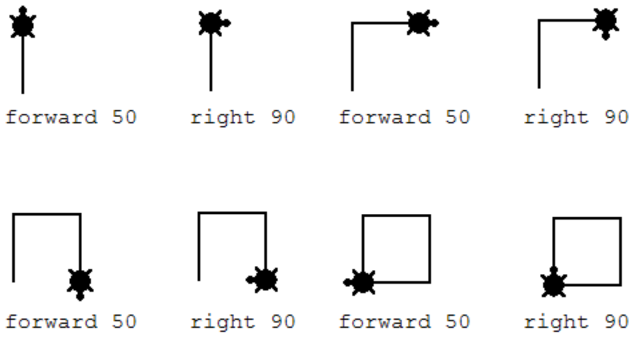
1. Problem Statement

Synthesis is the process of automatically translating thegiven input specification into a program. There are several ways to get around solving this problem of program generation:

- Traditional Rule based: Translate using pre defined rule. Such approaches mostly require a formal, precise specification that can be hard to obtain and also cannot perform well under noisy input
- Neural Program Synthesis: Model generates a program conditioned on the input specification, usually in the form of Input Output pairs
- Neural Program Induction: Models learns a latent representation of the program and generates output directly as per the input specifications

The purpose of our project is is to learn a model that generates logo program which achieves the task of creating output(images) as specified in the given input text caption. It involves learning a program representation under the Neural Program Synthesis approach, it can also be interpreted as converting pseduo code(input text caption) to logo code that can be compiled to generate the required output. Ex: Given the query ”This is an rectangle of side 10 and 20.”, the model should output the logo code `repeat 2 [fd 10 lt 90 fd 20 lt 90]`.

Logo is a general purpose programming language and it uses turtle graphics to produce graphics on screen using instructions that specify the direction and magnitude of movement. Imagine a robotic turtle starting at (0, 0) in the x-y plane. Give it the command `fd15`, and it moves (on-screen) 5 pixels in the direction it is facing, drawing a line as it moves. Give it the command `rt15`, and it rotates in-place 15 degrees clockwise. By combining together similar commands, intricate shapes and pictures can easily be drawn. A list of primitive Logo commands is provided here[1]. An example of a simple Logo program is as follows(step-by-step):



© 2000 Logo Foundation

We propose a generative model using Conditional GAN and attention based LSTM that learns to generate Logo code without explicitly depending on the input caption embedding. The architecture of our model is an extension of the model proposed by Reed Scott et al. [4]. We augment the conditional GAN architecture for text to image synthesis with an attention based Decoder LSTM to generate the Logo program token by token. The attention mechanism used in LSTM for program generation is similar to that described in the paper Show, Attend and Tell [5]. We present a model that only requires a simple specification to synthesize the complete program. It is also more robust to noisy input data since it does not completely depend upon the input information for generation of program. The hand engineered rules to translate input information to program in traditional models lead to poor performance in case of noisy data. But our model maps the initial textual specification to visual space and synthesizes the program from information in both the visual space and textual space, which would lead it to being more robust than rule based approaches.

2. Literature Review

In this section we review the important works that were central to designing our Generative Adversarial model for program synthesis.

2.1 Generative Adversarial Networks

Ian Goodfellow et al. [2] proposed a framework Generative Adversarial Networks(GAN) to learn generative models via adversarial training process. The framework consists of two models, Generator and Discriminator that are trained simultaneously in an adversarial game with the aim of training the Generator to learn a distribution over the training data and training the Discriminator to classify data correctly into the real data class(training data) and fake data class(synthesized data from Generator). The Generator learns a distribution over data $p_g(x|z, \theta_g)$ and Discriminator learns the distribution $p_d(x|\theta_d)$ to predict the probability of the data x coming the real data distribution. The framework is trained such that the Discriminator learns to differentiate between the actual data and the data samples from Generators distribution while simultaneously Generator learns to generate samples that would be be hard for the Discriminator to differentiate from real data. The adversarial setting corresponds to training procedure where both the components try to outperform each other. The Generator network updates its parameters in order to fool the Discriminator by synthesizing fake data samples which are similar to real data samples, with the hope that the Discriminator would not be able to distinguish between them and make a mistake by classifying them as real data samples. The Discriminator would update its parameters to minimize the mistakes and become better at classifying both the actual data and synthesized fake data from Generator.

The algorithm uses multi-layer perceptrons as the neural nets for both the Generator $\mathbb{G}$ and the Discriminator $\mathbb{D}$. The inputs to $\mathbb{G}$ are noise variables z, which are sampled from a prior $p_z(z)$. $\mathbb{G}$ models a function G such that $G(z, \theta_g)$ represents the generated image based on the noise input z. The discriminator model $\mathbb{D}$ learns a function D such that $D(x, \theta_d)$ represents the probability of the input x being from the original data distribution p_{data}. D is trained to maximize the probability of assigning the correct label to both actual data and counterfeits from G. We simultaneously train G to minimize the log probability of D “calling” it fake, given by $\log(1 - D(G(z)))$. Thus, both G and D play a min-max game that can be stated formally as a mix max two player game defined by the value function stated below:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(x)}[\log(1 - D(G(z)))]$$

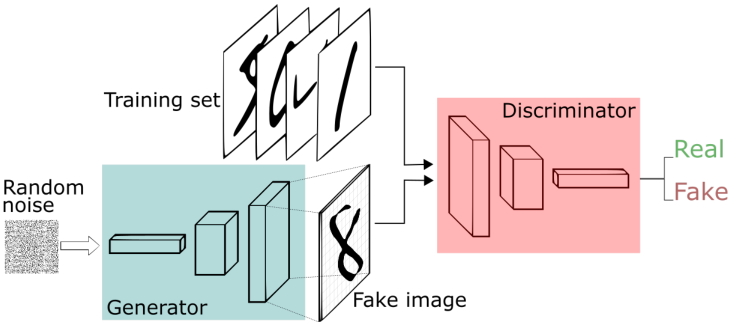
.

Algorithm 1 GAN

```
**Input:** minibatch images $x$, number of training batch steps $N$
for $n = 1$ to $N$ do
    $z \leftarrow \mathcal{N}(0,1)^Z$                    ▷ Draw sample of random noise
    $\hat{x} \leftarrow G(z, h)$                          ▷ Forward through generator
    $s_r \leftarrow D(x)$                                 ▷ real image
    $s_f \leftarrow D(\hat{x})$                           ▷ fake image
    $\mathcal{L}_D \leftarrow \log(s_r) + \log(1 - s_f)$
    $\mathcal{L}_G \leftarrow \log(s_f)$
    $D \leftarrow D - \frac{\partial \mathcal{L}_D}{\partial D}$
    $G \leftarrow G - \frac{\partial \mathcal{L}_G}{\partial G}$
end for
```

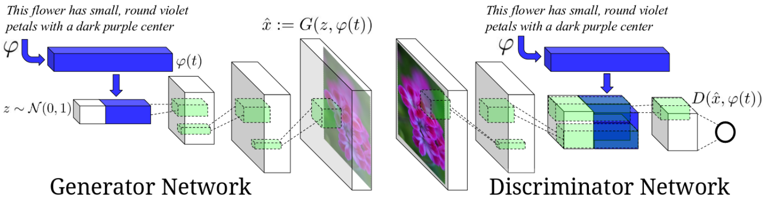
The authors claim that the game reaches an optima in theory, where the generator learns the data distribution p_{data} and the discriminator can no longer discriminate between a real(taken from the actual data distribution) or a fake(generated by the generator) image. Thus, we end up with a generative model which models the original data distribution very well.

A major advantage of using Generative Adversarial Networks as generative models is that the whole framework can be trained using backpropagation. Hence, they do not involve any intractable probabilistic computations that would arise in other Deep Generative Models during Maximum Likelihood Estimation. However, GANs are implicit generative models, they learn a distribution over data that do not have any explicit form or have explicit likelihood expressions. We do not have a mathematical formulation of the distribution $p_g(x|z, \theta_g)$ learnt by GAN, we can only evaluate the distribution by sampling data from the distribution.



2.2 Generative Adversarial Text to Image Synthesis

Scott Reed et al. [4] proposed a text to image synthesis architecture based on conditional GANs. Conditional GAN is a modification over the GAN, the Generator is conditioned on random noise z concatenated with encoding representing additional information. The modified input to the Generator helps it to generate data samples capturing some particular information. For the text to image task, the text caption describing the image is the additional information that we would want to include in the Generator. A normal GAN does not have control over input noise and hence it can only learn to generate data samples similar to actual data samples but it would not be able to preserve a relationship between the specific description in text caption and the image generated, which is a very important criteria for text to image task. Hence, the paper proposes using Conditional GAN that encode the textual information in the caption and feed it to the Generator. The Discriminator architecture is also modified to include the textual information with the aim of it training to identify the mismatch between the text caption and image along with classiying real images from fake images.


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

The discriminator observes two kinds of inputs: real images with matching text, and synthetic images with arbitrary text. Hence it has to separate two sources of error: synthesized images and images from actual training data that mismatch the encoded caption textual information. This is done by modifying the GAN training algorithm to incorporate an additional loss term over the loss function in GAN. The additional loss term corresponds to the real image with incorrect caption that the Discriminator is trained to predict as fake class.
Modified Loss function:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x,h)] + \mathbb{E}_{x \sim p_{data}(x)}[\log(1 - D(x,\hat{h}))]$$

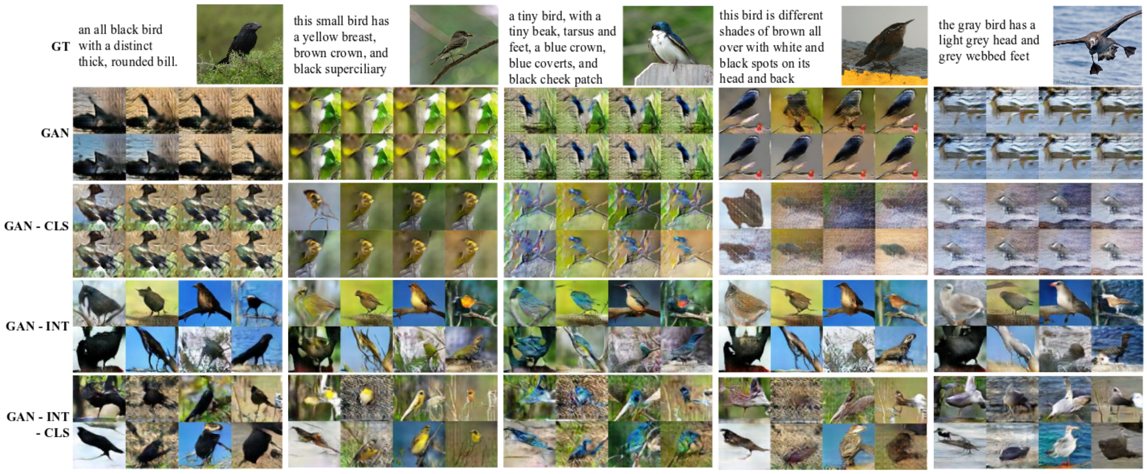
$$+\mathbb{E}_{z \sim p_z(x)}[\log(1 - D(G(z),h))]$$

where h corresponds to the correct caption embedding corresponding the real image x and the $\hat{h}$ corresponds to a wrong caption embedding

Algorithm 2 GAN-CLS training algorithm with step size α, using minibatch SGD for simplicity.

```
**Input:** minibatch images $x$, matching text $t$, mis-matching $\hat{t}$, number of training batch steps $S$
**for** $n = 1$ **to** $S$ **do**
    $h \leftarrow \phi(t)$                                  ▷ Encode matching text description
    $\hat{h} \leftarrow \phi(\hat{t})$                      ▷ Encode mis-matching text description
    $z \leftarrow \mathcal{N}(0,1)^Z$                       ▷ Draw sample of random noise
    $\hat{x} \leftarrow G(z,h)$                             ▷ Forward through generator
    $s_r \leftarrow D(x,h)$                                 ▷ real image, right text
    $s_w \leftarrow D(x,\hat{h})$                           ▷ real image, wrong text
    $s_f \leftarrow D(\hat{x},h)$                           ▷ fake image, wrong text
    $\mathcal{L}_D \leftarrow \log(s_r) + \frac{\log(1-s_w)+\log(1-s_f)}{2}$
    $D \leftarrow D - \frac{\partial \mathcal{L}_D}{\partial D}$        ▷ Update discriminator
    $\mathcal{L}_G \leftarrow \log(s_f)$
    $G \leftarrow G - \frac{\partial \mathcal{L}_G}{\partial G}$        ▷ Update generator
**end for**
```

Scott Reed's method is the first end-to-end differentiable architecture from the character level to pixel level. They obtain very good results on the CUB dataset. The following image from the paper highlights their results. The top row, GT, refers to ground truth and the other rows show the generated images by different versions of their algorithm.


GT
an all black bird with a distinct thick, rounded bill.
this small bird has a yellow breast, brown crown, and black superciliary
a tiny bird, with a tiny beak, tarsus and feet, a blue crown, blue coverts, and black cheek patch
this bird is different shades of brown all over with white and black spots on its head and back
the gray bird has a light grey head and grey webbed feet
GAN
GAN - CLS
GAN - INT
GAN - INT - CLS

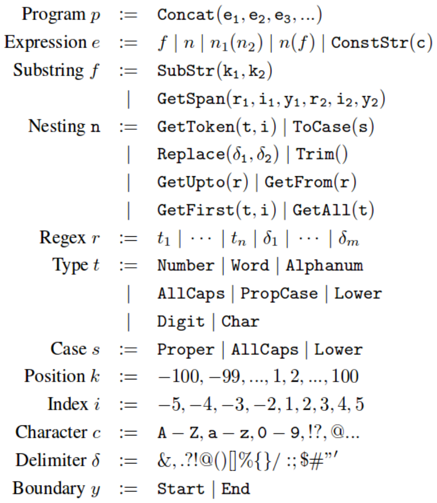
Generative Adversarial Text to Image Synthesis

2.3 RobustFill: Neural Program Learning under Noisy I/O

This paper [3] proposes neural models based on attention RNN for the task of program synthesis and compares them on real world learning tasks. The work is based upon the two recent approaches towards program synthesis: Neural Program Induction and Neural

Program Synthesis. In the Neural Program Induction approach, the model is trained to predict the Output for the given Input by learning a latent representation of the program. The Neural Synthesis approach conditions a model on the Input Output pairs and learns to generate the program.

For both the above approaches, the problem is formulated as given Input Output string examples and unlabelled test Input strings, predict the Output for the test Input strings. This corresponds to the task of learning a program P that can correctly map the Input strings to the Output strings. In Neural Program Synthesis approach, the model has to additionally generate the program token by token. The program is represented using Domain Specific Language(DSL), which models string transformations. The complete details of the DSL can be found in the paper, a brief summary of it is presented in the image below



$$
\begin{array}{rcl}
\text{Program } p & := & \texttt{Concat}(\texttt{e}_1, \texttt{e}_2, \texttt{e}_3, ...) \\
\text{Expression } e & := & f \mid n \mid n_1(n_2) \mid n(f) \mid \texttt{ConstStr}(\texttt{c}) \\
\text{Substring } f & := & \texttt{SubStr}(\texttt{k}_1, \texttt{k}_2) \\
 & \mid & \texttt{GetSpan}(\texttt{r}_1, \texttt{i}_1, \texttt{y}_1, \texttt{r}_2, \texttt{i}_2, \texttt{y}_2) \\
\text{Nesting n} & := & \texttt{GetToken}(\texttt{t}, \texttt{i}) \mid \texttt{ToCase}(\texttt{s}) \\
 & \mid & \texttt{Replace}(\delta_1, \delta_2) \mid \texttt{Trim}() \\
 & \mid & \texttt{GetUpto}(\texttt{r}) \mid \texttt{GetFrom}(\texttt{r}) \\
 & \mid & \texttt{GetFirst}(\texttt{t}, \texttt{i}) \mid \texttt{GetAll}(\texttt{t}) \\
\text{Regex } r & := & t_1 \mid \cdots \mid t_n \mid \delta_1 \mid \cdots \mid \delta_m \\
\text{Type } t & := & \texttt{Number} \mid \texttt{Word} \mid \texttt{Alphanum} \\
 & \mid & \texttt{AllCaps} \mid \texttt{PropCase} \mid \texttt{Lower} \\
 & \mid & \texttt{Digit} \mid \texttt{Char} \\
\text{Case } s & := & \texttt{Proper} \mid \texttt{AllCaps} \mid \texttt{Lower} \\
\text{Position } k & := & -100, -99, ..., 1, 2, ..., 100 \\
\text{Index } i & := & -5, -4, -3, -2, 1, 2, 3, 4, 5 \\
\text{Character } c & := & \texttt{A} - \texttt{Z}, \texttt{a} - \texttt{z}, \texttt{0} - \texttt{9}, !?, @... \\
\text{Delimiter } \delta & := & \&, .?!@()[]\%\{\}/ :; \$\#"' \\
\text{Boundary } y & := & \texttt{Start} \mid \texttt{End}
\end{array}
$$

String Transformation DSL.

The Neural Program Synthesis task is modelled as a sequence-to-sequence generation task. The observed I/O is encoded using recurrent neural networks(RNN), and generates program sequence P using another RNN, one token at a time. They also use Multi Example Pooling to encode variable length I/O sequences. They describe and evaluate the following multi-attentional variants of the attention RNN architecture.

- **Basic Seq-to-Seq**: Normal Encoder Decoder LSTM architecture without any attention mechanism. The encoded I/O is used to initialize the hidden states of the Decoder LSTM and token output of Decoder LSTM across time step would lead to the synthesized program.

- **Attention-A**: The LSTM associated for processing Output and Program token have attention mechanisms. The attention mechanism for Output associated LSTM cell uses the hidden states of Input LSTM for attention while the Program associated LSTM cell uses the hidden states of Output LSTM for attention.

- **Attention-B**: The LSTM associated with Output has same attention mechanism as in case A but the Program LSTM has double attention mechanism, it uses both the hidden states of Input and Output LSTM for attention

- **Attention-C**: Similar structure to as in case Attention-B, but the LSTM associated with Input and Output are both Bi Directional

Their Program Synthesis results using the attention RNN variants show huge improvement over the previous approaches on FlashFillTest dataset. They also demonstrate that their model is robust to moderate noise in Input Output examples.

2.4 Neural Image Caption Generation with Visual Attention

This paper [5] proposes an attention based model for the task of image captioning. The advantage of using attention based models is that rather than learning a static encoding of the information in image, we can learn the features dynamically and hence focus on relevant part of image during generation of a particular token. This work introduces two attention based caption generator frameworks: Soft Attention and Hard Attention. The Soft Attention mechanism can be trained using backpropagation like in normal neural networks while the Hard Attention mechanism can be trained by minimizing the Variational Lower Bound.

In a normal sequence to sequence model, attention mechanism in Decoder LSTM can be used by using the hidden states of the Encoder LSTM. However, this standard model of soft attention cannot be directly implemented here since we do not have an Encoder LSTM in the task of Image Captioning. The attention mechanism proposed in the paper overcomes this problem by developing a set of annotation feature vectors $\{a_i\}$ constructed from the image using CNN and an attention framework f_{att} that predicts the weight α_i for each annotation feature a_i. This is used compute the context vector $\hat{z_t}$. Hence, the CNN architecture for constructing the annotation vector $\{a_i\}$ serves as the Encoder and the Decoder LSTM uses attention framework to determine which annotation vectors to focus more for computing the hidden states at a particular time step.

$$e_{ti} = f_{att}(a_i, h_{t-1})$$

$$\alpha_{ti} = \frac{exp(e_{ti})}{\sum_{k=1}^{L} exp(e_{tk})}$$

$$\hat{z_t} = \psi(\{a_i\}, \{\alpha_i\})$$

The hidden state and memory state of the LSTM are initialised by feeding the average of annotation vectors into Multi Layer Perceptron layers $f_{init,c}$ and $f_{init,h}$. The updates for the hidden states and memory states of LSTM for the further time step depend on the previous state and the context vector, defined below:

$$c_o = f_{init,c}(\sum_{i}^{L} a_i/L)$$

$$h_o = f_{init,h}(\sum_i^L a_i/L)$$

$$\begin{pmatrix} \boldsymbol{i}_t \\ \boldsymbol{f}_t \\ \boldsymbol{o}_t \\ \boldsymbol{g}_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} T_{D+m+n,n} \begin{pmatrix} \boldsymbol{E}\boldsymbol{y}_{t-1} \\ \boldsymbol{h}_{t-1} \\ \hat{\boldsymbol{z}}_t \end{pmatrix}$$

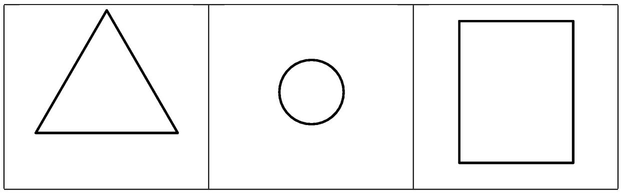
$$c_t = f_t * c_{t-1} + i_t * g_t$$

$$h_t = o_t * tanh(c_t)$$

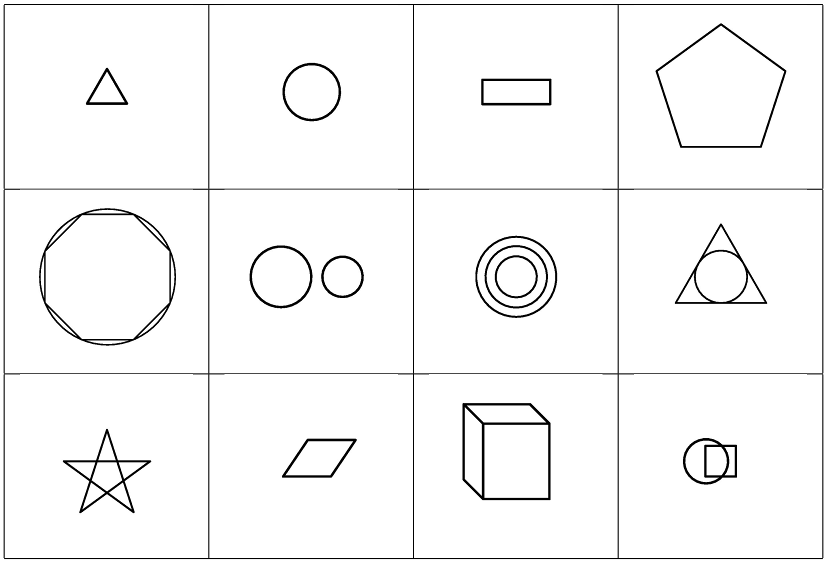
3. Method

3.1 Dataset

Due to no standard datasets available for testing, we created two datasets Basic and Complex to test the task of generating Logo Code that create geometrical shapes. Basic dataset has 10,000 images with 3 different categories. Complex dataset has 20,000 images with 12 different categories. An example of the images is shown below.

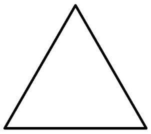


Basic Dataset images



Complex Dataset images

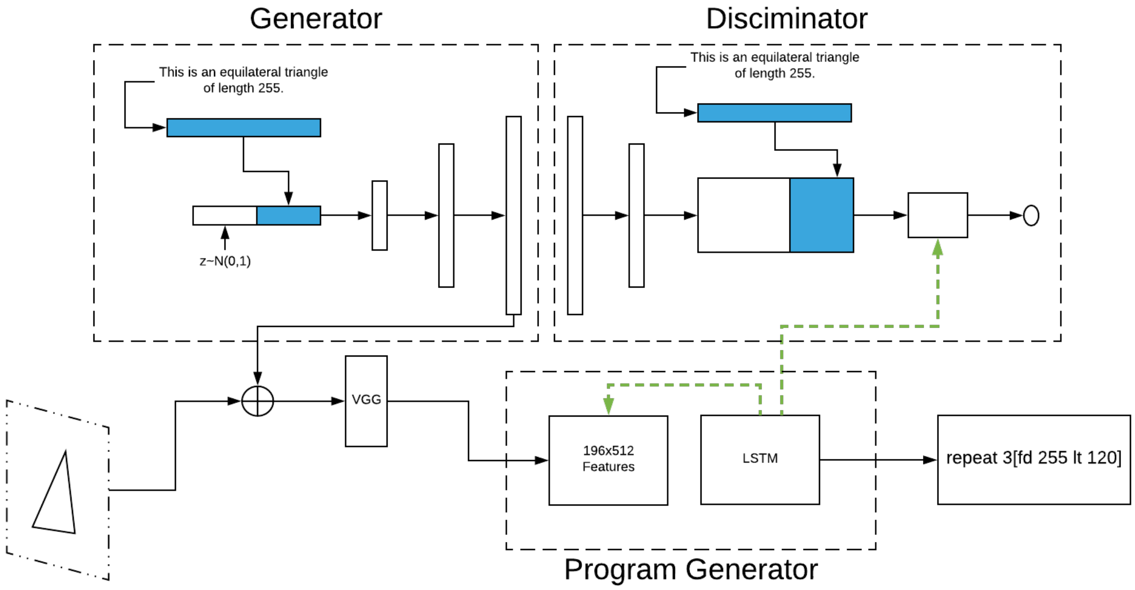
Apart from the images both the datasets comprise of description as well as logo code for each image. An example of the logo code and description is shown below.

Image	
Description	This is an equilateral triangle of side length: 656.
Logo Code	`repeat 3 [fd 656 lt 120]`

Example from Basic Dataset

3.2 Network Architecture and Core Algorithm

We use a conditional DC-GAN architecture as proposed by [4] to generate images (conditioned on text caption). The text caption is encoded using skipthoughts and then appended with the noise vector before feeding into the Generator We augment the text to image architecture by adding $\mathcal{L}1$ loss term between the encoding of the real and fake image. This encoding is extracted out from the penultimate layer of discriminator. We use a Decoder LSTM network to synthesize Logo code with attention framework similar to the paper Show Attend Tell. The annotation vectors comprises of features constructed using VGG network on image along with the encoding extracted out of the penultimate layer of the discriminator. We can also incorporate additional annotation vector such as the skipthoughts encoding of the caption. And we empirically see that the code generated after adding this skipthoughts embedding predict numbers that are close to the original.



Overall Architecture without caption attention

Our algorithm as described by the following pseudo-code:

Algorithm 3 Model

```
Input: minibatch images $x$, matching code $c$, matching description $t$, different image $x_1$,
number of training batch steps $N$
for $n = 1$ to $N$ do
    $h \leftarrow \phi(t)$                                   ▷ Encode matching text description
    $z \leftarrow \mathcal{N}(0,1)^Z$                        ▷ Draw sample of random noise
    $\hat{x} \leftarrow G(z,h)$                              ▷ Forward through generator
    $s_r, pl_r \leftarrow D(x,h)$                            ▷ real image, right description
    $s_w, pl_w \leftarrow D(x_1,h)$                          ▷ mis-matching image and description
    $s_f, pl_f \leftarrow D(\hat{x},h)$                      ▷ fake image, right description
    $c_r \leftarrow PG(x, pl_r)$                             ▷ Forward through program generator, real image
    $c_f \leftarrow PG(\hat{x}, pl_f)$                       ▷ Forward through program generator, fake image
    $\mathcal{L}_E \leftarrow |pl_r - pl_f|$                 ▷ L1 loss
    $\mathcal{L}_D \leftarrow \log(s_r) + \frac{\log(1-s_w)+\log(1-s_f)}{2} + \mathcal{L}_E$
    $\mathcal{L}_G \leftarrow \log(s_f)$
    $\mathcal{L}_R \leftarrow seq2seq\_loss(c, c_r) + seq2seq\_loss(c, c_f)$   ▷ LSTM Loss
    $D \leftarrow D - \frac{\partial \mathcal{L}_D}{\partial D}$
    $G \leftarrow G - \frac{\partial \mathcal{L}_G}{\partial G}$
    $R \leftarrow R - \frac{\partial \mathcal{L}_R}{\partial R}$
end for
```

4. Experiments

The end to end training of the model is tricky and performing well on both the tasks of generating good synthesized images and good synthesized logo program is challenging. After a lot of tuning and training methodologies for LSTM we arrive at the following method to generate good results. We add teacher forcing for program generation from both fake and real image during the training phase of the model to facilitate LSTM in tuning its parameters. Hence, the LSTM is updated on the loss contribution from Fake Image, Real Image, Fake Image Teacher Forcing and Real Image Teacher Forcing. We learn the parameters by first updating the parameters of LSTM and Discriminator together and then updating the parameters of Generator. The test dataset comprises of 1000 data points with the same structure and specifications as the train dataset: (input caption, image, logo code) triplet

4.1 Basic Dataset with caption attention

4.1.1 Basic Dataset with caption attention, Real Image while Training

This is the code generated when we pass a real image from the basic dataset to the program generator. We can see that the model is able to capture the structure of the logo code and the numbers predicted are pretty close. The code is also syntactically correct.

Actual Code	repeat 3 [fd 375 lt 120]	repeat 2 [fd 586 lt 90 fd 524 lt 90]
Epoch 0	repeat 3 [fd 4 100	repeat [fd 5 9
Epoch 15	repeat 3 [fd 382 lt 120]	repeat 2 [fd 580 lt 90 fd 492 lt 90]
Epoch 30	repeat 3 [fd 362 lt 120]	repeat 2 [fd 585 lt 90 fd 522 lt 90]
Epoch 45	repeat 3 [fd 303 lt 120]	repeat 2 [fd 583 lt 90 fd 520 lt 90]
Epoch 60	repeat 3 [fd 301 lt 120]	repeat 2 [fd 500 lt 90 fd 500 lt 90]
Epoch 75	repeat 3 [fd 370 lt 120]	repeat 2 [fd 590 lt 90 fd 500 lt 90]
Epoch 90	repeat 3 [fd 372 lt 120]	repeat 2 [fd 583 lt 90 fd 520 lt 90]
Epoch 105	repeat 3 [fd 370 lt 120]	repeat 2 [fd 553 lt 90 fd 582 lt 90]
Epoch 120	repeat 3 [fd 370 lt 120]	repeat 2 [fd 580 lt 90 fd 500 lt 90]
Epoch 135	repeat 3 [fd 370 lt 120]	repeat 2 [fd 580 lt 90 fd 525 lt 90]
Epoch 150	repeat 3 [fd 370 lt 120]	repeat 2 [fd 581 lt 90 fd 511 lt 90]
Epoch 165	repeat 3 [fd 375 lt 120]	repeat 2 [fd 580 lt 90 fd 523 lt 90]
Epoch 180	repeat 3 [fd 375 lt 120]	repeat 2 [fd 580 lt 90 fd 500 lt 90]
Epoch 195	repeat 3 [fd 375 lt 120]	repeat 2 [fd 580 lt 90 fd 520 lt 90]
Epoch 210	repeat 3 [fd 375 lt 120]	repeat 2 [fd 585 lt 90 fd 526 lt 90]
Epoch 225	repeat 3 [fd 375 lt 120]	repeat 2 [fd 584 lt 90 fd 522 lt 90]
Epoch 240	repeat 3 [fd 375 lt 120]	repeat 2 [fd 581 lt 90 fd 523 lt 90]
Epoch 255	repeat 3 [fd 375 lt 120]	repeat 2 [fd 583 lt 90 fd 523 lt 90]
Epoch 270	repeat 3 [fd 374 lt 120]	repeat 2 [fd 599 lt 90 fd 512 lt 90]
Epoch 285	repeat 3 [fd 375 lt 120]	repeat 2 [fd 583 lt 90 fd 523 lt 90]

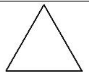
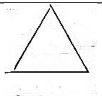
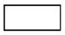
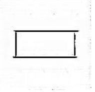
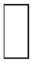
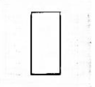
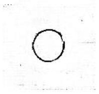
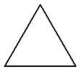
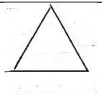
4.1.2 Basic Dataset with caption attention, Fake Image while Training

This is the code generated when we pass a fake image generated from the generator trained on the basic dataset, to the program generator. We can see that the model is able to capture the structure of the logo code and the numbers predicted are pretty close. The code is also syntactically correct. We also see that there is no difference between the program generated from the real and fake image, this shows that the generator is learning to produce good enough images.

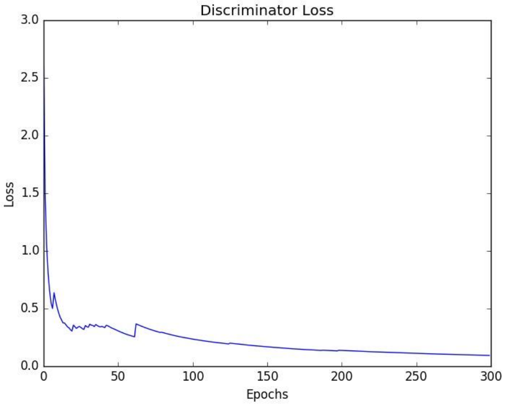
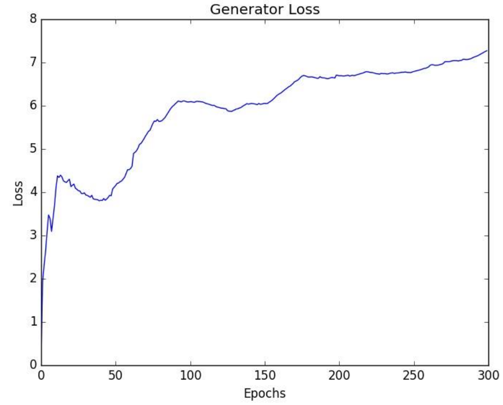
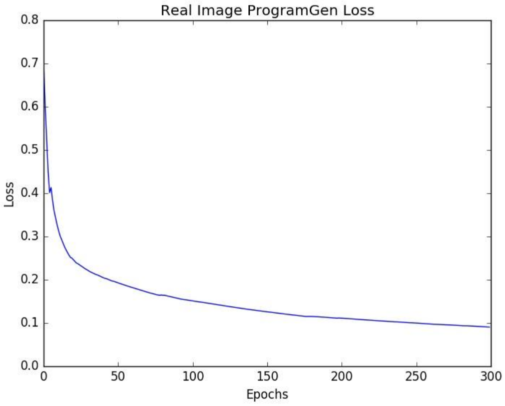
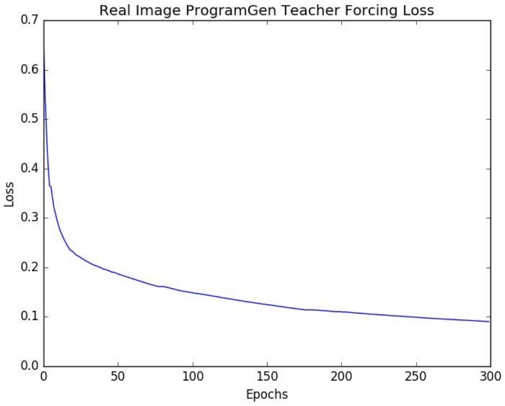
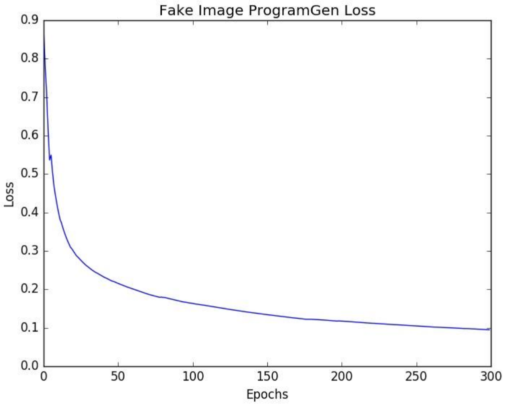
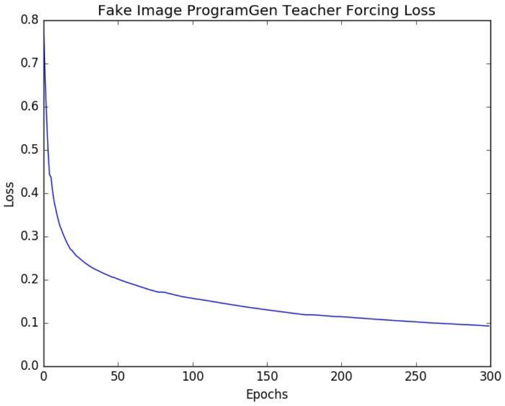
Actual Code	repeat 3 [fd 375 lt 120]	repeat 2 [fd 586 lt 90 fd 524 lt 90]
Epoch 0	repeat 3 [fd 4 200	repeat 2 [fd 5
Epoch 15	repeat 3 [fd 319 lt 120]	repeat 2 [fd 580 lt 90 fd 492 lt 90]
Epoch 30	repeat 3 [fd 362 lt 120]	repeat 2 [fd 585 lt 90 fd 522 lt 90]
Epoch 45	repeat 3 [fd 303 lt 120]	repeat 2 [fd 581 lt 90 fd 520 lt 90]
Epoch 60	repeat 3 [fd 301 lt 120]	repeat 2 [fd 500 lt 90 fd 500 lt 90]
Epoch 75	repeat 3 [fd 370 lt 120]	repeat 2 [fd 590 lt 90 fd 500 lt 90]
Epoch 90	repeat 3 [fd 372 lt 120]	repeat 2 [fd 583 lt 90 fd 520 lt 90]
Epoch 105	repeat 3 [fd 370 lt 120]	repeat 2 [fd 553 lt 90 fd 582 lt 90]
Epoch 120	repeat 3 [fd 370 lt 120]	repeat 2 [fd 580 lt 90 fd 500 lt 90]
Epoch 135	repeat 3 [fd 370 lt 120]	repeat 2 [fd 580 lt 90 fd 525 lt 90]
Epoch 150	repeat 3 [fd 370 lt 120]	repeat 2 [fd 581 lt 90 fd 511 lt 90]
Epoch 165	repeat 3 [fd 375 lt 120]	repeat 2 [fd 580 lt 90 fd 523 lt 90]
Epoch 180	repeat 3 [fd 375 lt 120]	repeat 2 [fd 580 lt 90 fd 500 lt 90]
Epoch 195	repeat 3 [fd 375 lt 120]	repeat 2 [fd 580 lt 90 fd 520 lt 90]
Epoch 210	repeat 3 [fd 375 lt 120]	repeat 2 [fd 585 lt 90 fd 526 lt 90]
Epoch 225	repeat 3 [fd 375 lt 120]	repeat 2 [fd 584 lt 90 fd 522 lt 90]
Epoch 240	repeat 3 [fd 375 lt 120]	repeat 2 [fd 583 lt 90 fd 523 lt 90]
Epoch 255	repeat 3 [fd 375 lt 120]	repeat 2 [fd 583 lt 90 fd 523 lt 90]
Epoch 270	repeat 3 [fd 374 lt 120]	repeat 2 [fd 599 lt 90 fd 512 lt 90]
Epoch 285	repeat 3 [fd 375 lt 120]	repeat 2 [fd 583 lt 90 fd 523 lt 90]

4.1.3 Basic Dataset with caption attention, while Testing

This table shows the final results of the model. We can see that the fake image produced by the generator is very similar to the actual image. And the fake code produced by the program generator is also very close to the actual code.

Real Image	Fake Image	Real Code	Fake Code
		repeat 3 [fd 653 lt 120]	repeat 3 [fd 653 lt 120]
		repeat 2 [fd 487 lt 90 fd 213 lt 90]	repeat 2 [fd 485 lt 90 fd 222 lt 90]
		repeat 2 [fd 210 lt 90 fd 461 lt 90]	repeat 2 [fd 210 lt 90 fd 461 lt 90]
		circle 161	circle 161
		repeat 3 [fd 616 lt 120]	repeat 3 [fd 612 lt 120]

4.1.4 Basic Dataset with caption attention - Plots

Discriminator Loss	Generator Loss
	
Real Image ProgramGen Loss	Real Image ProgramGen Teacher Forcing Loss
	
Fake Image ProgramGen Loss	Fake Image ProgramGen Teacher Forcing Loss
	

4.2 Basic Dataset without caption attention

4.2.1 Basic Dataset without caption attention, Real Image while Training

This is the code generated when we pass a real image from the basic dataset to the program generator. We can see that the model is able to capture the structure of the logo code. The code is also syntactically correct. But unlike the case in which we had caption attention the numbers are not that accurate.

Actual Code	repeat 2 [fd 595 lt 90 fd 371 lt 90]	repeat 2 [fd 586 lt 90 fd 524 lt 90]
Epoch 0	repeat 2 [fd 9	repeat 3 [fd 5 100
Epoch 15	repeat 2 [fd 688 lt 90 fd 349 lt 90]	repeat 3 [fd 669 lt 120]
Epoch 30	repeat 2 [fd 680 lt 90 fd 399 lt 90]	repeat 3 [fd 667 lt 120]
Epoch 45	repeat 2 [fd 590 lt 90 fd 369 lt 90]	repeat 3 [fd 666 lt 120]
Epoch 60	repeat 2 [fd 580 lt 90 fd 328 lt 90]	repeat 3 [fd 696 lt 120]
Epoch 75	repeat 2 [fd 620 lt 90 fd 311 lt 90]	repeat 3 [fd 679 lt 120]
Epoch 90	repeat 2 [fd 690 lt 90 fd 311 lt 90]	repeat 3 [fd 699 lt 120]
Epoch 105	repeat 2 [fd 500 lt 90 fd 361 lt 90]	repeat 3 [fd 689 lt 120]
Epoch 120	repeat 2 [fd 590 lt 90 fd 361 lt 90]	repeat 3 [fd 689 lt 120]
Epoch 135	repeat 2 [fd 591 lt 90 fd 371 lt 90]	repeat 3 [fd 699 lt 120]
Epoch 150	repeat 2 [fd 581 lt 90 fd 374 lt 90]	repeat 3 [fd 691 lt 120]
Epoch 165	repeat 2 [fd 590 lt 90 fd 370 lt 90]	repeat 3 [fd 690 lt 120]
Epoch 180	repeat 2 [fd 599 lt 90 fd 370 lt 90]	repeat 3 [fd 690 lt 120]
Epoch 195	repeat 2 [fd 591 lt 90 fd 370 lt 90]	repeat 3 [fd 690 lt 120]
Epoch 210	repeat 2 [fd 599 lt 90 fd 371 lt 90]	repeat 3 [fd 690 lt 120]
Epoch 225	repeat 2 [fd 599 lt 90 fd 371 lt 90]	repeat 3 [fd 690 lt 120]
Epoch 240	repeat 2 [fd 599 lt 90 fd 371 lt 90]	repeat 3 [fd 690 lt 120]
Epoch 255	repeat 2 [fd 590 lt 90 fd 371 lt 90]	repeat 3 [fd 690 lt 120]
Epoch 270	repeat 2 [fd 596 lt 90 fd 371 lt 90]	repeat 3 [fd 690 lt 120]
Epoch 285	repeat 2 [fd 590 lt 90 fd 371 lt 90]	repeat 3 [fd 690 lt 120]

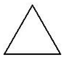
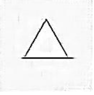
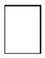
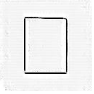
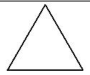
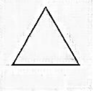
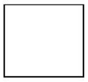
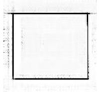
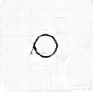
4.2.2 Basic Dataset without caption attention, Fake Image while Training

This is the code generated when we pass a fake image generated from the generator trained on the basic dataset, to the program generator. We can see that the model is able to capture the structure of the logo code. The code is also syntactically correct. We also see that there is a bit difference between the program generated from the real and fake image, the main reason for this might be due to the high variance which we didn't had while using caption attention.

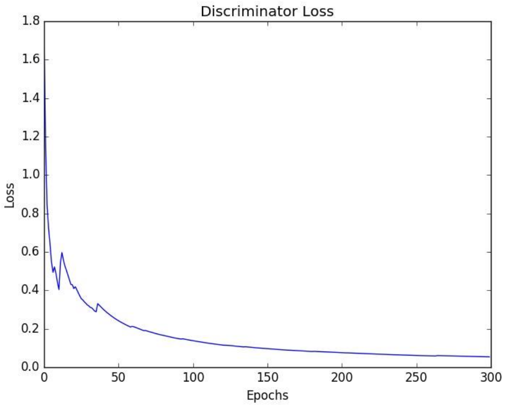
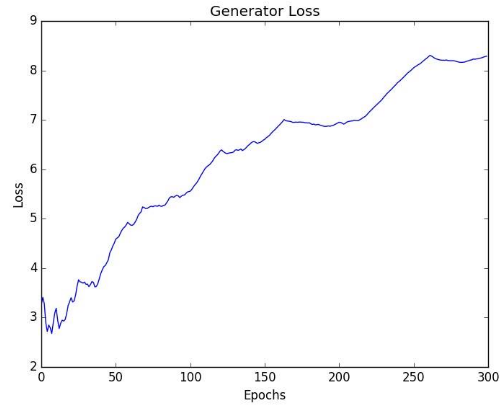
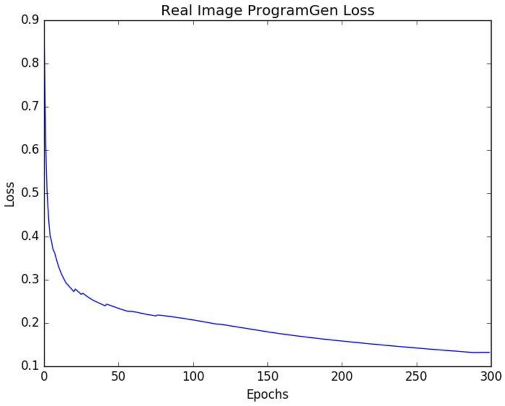
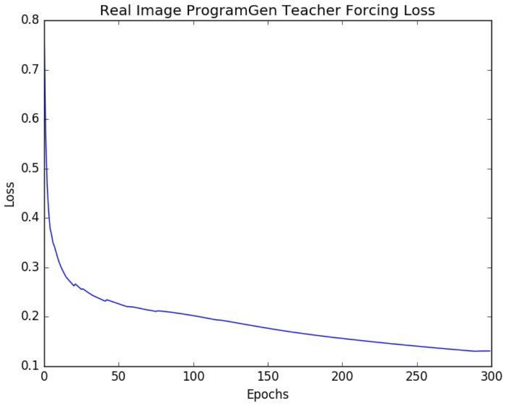
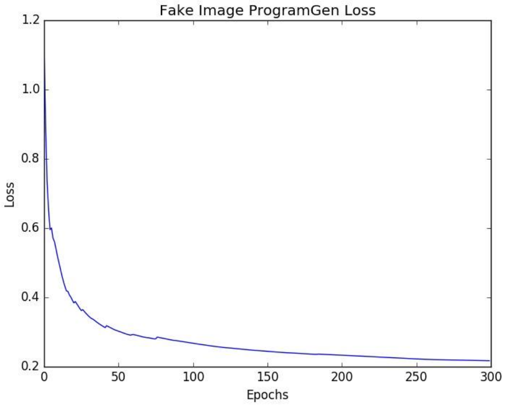
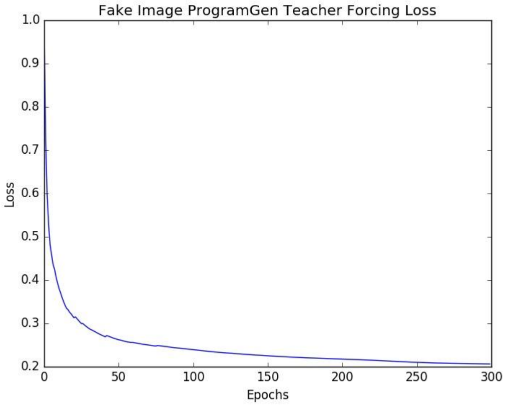
Actual Code	repeat 2 [fd 595 lt 90 fd 371 lt 90]	repeat 2 [fd 586 lt 90 fd 524 lt 90]
Epoch 0	repeat 2 [fd 4 9	repeat 3 [fd 5 100
Epoch 15	repeat 2 [fd 581 lt 90 fd 211 lt 90]	repeat 3 [fd 669 lt 120]
Epoch 30	repeat 2 [fd 681 lt 90 fd 311 lt 90]	repeat 3 [fd 677 lt 120]
Epoch 45	repeat 2 [fd 690 lt 90 fd 301 lt 90]	repeat 3 [fd 666 lt 120]
Epoch 60	repeat 2 [fd 680 lt 90 fd 428 lt 90]	repeat 3 [fd 596 lt 120]
Epoch 75	repeat 2 [fd 610 lt 90 fd 321 lt 90]	repeat 3 [fd 691 lt 120]
Epoch 90	repeat 2 [fd 520 lt 90 fd 211 lt 90]	repeat 3 [fd 688 lt 120]
Epoch 105	repeat 2 [fd 680 lt 90 fd 421 lt 90]	repeat 3 [fd 696 lt 120]
Epoch 120	repeat 2 [fd 681 lt 90 fd 321 lt 90]	repeat 3 [fd 676 lt 120]
Epoch 135	repeat 2 [fd 611 lt 90 fd 211 lt 90]	repeat 3 [fd 690 lt 120]
Epoch 150	repeat 2 [fd 580 lt 90 fd 299 lt 90]	repeat 3 [fd 686 lt 120]
Epoch 165	repeat 2 [fd 680 lt 90 fd 300 lt 90]	repeat 3 [fd 696 lt 120]
Epoch 180	repeat 2 [fd 510 lt 90 fd 391 lt 90]	repeat 3 [fd 696 lt 120]
Epoch 195	repeat 2 [fd 689 lt 90 fd 321 lt 90]	repeat 3 [fd 690 lt 120]
Epoch 210	repeat 2 [fd 500 lt 90 fd 211 lt 90]	repeat 3 [fd 696 lt 120]
Epoch 225	repeat 2 [fd 510 lt 90 fd 399 lt 90]	repeat 3 [fd 699 lt 120]
Epoch 240	repeat 2 [fd 610 lt 90 fd 391 lt 90]	repeat 3 [fd 699 lt 120]
Epoch 255	repeat 2 [fd 610 lt 90 fd 321 lt 90]	repeat 3 [fd 699 lt 120]
Epoch 270	repeat 2 [fd 500 lt 90 fd 261 lt 90]	repeat 3 [fd 699 lt 120]
Epoch 285	repeat 2 [fd 510 lt 90 fd 490 lt 90]	repeat 3 [fd 699 lt 120]

4.2.3 Basic Dataset without caption attention, while Testing

This table shows the final results of the model. We can see that the fake image produced by the generator is very similar to the actual image. So, the main reason that the program generated is not that good because we aren't using caption attention.

Real Image	Fake Image	Real Code	Fake Code
		repeat 3 [fd 483 lt 120]	repeat 3 [fd 471 lt 120]
		repeat 2 [fd 311 lt 90 fd 409 lt 90]	repeat 2 [fd 301 lt 90 fd 424 lt 90]
		repeat 3 [fd 653 lt 120]	repeat 3 [fd 619 lt 120]
		repeat 2 [fd 683 lt 90 fd 617 lt 90]	repeat 2 [fd 689 lt 90 fd 508 lt 90]
		circle 147	circle 144

4.2.4 Basic Dataset without caption attention - Plots

Discriminator Loss	Generator Loss
	
Real Image ProgramGen Loss	**Real Image ProgramGen Teacher Forcing Loss**
	
Fake Image ProgramGen Loss	**Fake Image ProgramGen Teacher Forcing Loss**
	

4.3 Complex Dataset with caption attention

4.3.1 Complex Dataset with caption attention, Real Image while Training

This is the code generated when we pass a real image from the complex dataset to the program generator. We can see that the model is able to capture the structure of the logo code. The code is also syntactically correct.

Actual Code	circle 84 pu rt 90 fd 43 rt 270 pd circle 127	repeat 6 [fd 302 rt 144]
Epoch 0	circle 1	repeat [4
Epoch 15	circle 92 pu rt 90 fd 33 rt 270 pd circle 116	repeat 6 [fd 236 rt 144]
Epoch 30	circle 80 pu rt 90 fd 44 rt 270 pd circle 124	repeat 6 [fd 390 rt 144]
Epoch 45	circle 85 pu rt 90 fd 47 rt 270 pd circle 127	repeat 6 [fd 303 rt 144]
Epoch 60	circle 88 pu rt 90 fd 48 rt 270 pd circle 127	repeat 6 [fd 304 rt 144]
Epoch 75	circle 88 pu rt 90 fd 49 rt 270 pd circle 127	repeat 6 [fd 303 rt 144]
Epoch 90	circle 84 pu rt 90 fd 49 rt 270 pd circle 122	repeat 6 [fd 302 rt 144]
Epoch 105	circle 88 pu rt 90 fd 49 rt 270 pd circle 127	repeat 6 [fd 302 rt 144]
Epoch 120	circle 88 pu rt 90 fd 47 rt 270 pd circle 127	repeat 6 [fd 302 rt 144]
Epoch 135	circle 88 pu rt 90 fd 49 rt 270 pd circle 127	repeat 6 [fd 302 rt 144]
Epoch 150	circle 88 pu rt 90 fd 49 rt 270 pd circle 127	repeat 6 [fd 302 rt 144]
Epoch 165	circle 88 pu rt 90 fd 49 rt 270 pd circle 127	repeat 6 [fd 302 rt 144]
Epoch 180	circle 88 pu rt 90 fd 49 rt 270 pd circle 127	repeat 6 [fd 302 rt 144]
Epoch 195	circle 82 pu rt 90 fd 45 rt 270 pd circle 124	repeat 6 [fd 302 rt 144]
Epoch 210	circle 88 pu rt 90 fd 49 rt 270 pd circle 127	repeat 6 [fd 302 rt 144]
Epoch 225	circle 88 pu rt 90 fd 49 rt 270 pd circle 127	repeat 6 [fd 302 rt 144]
Epoch 240	circle 88 pu rt 90 fd 49 rt 270 pd circle 127	repeat 6 [fd 302 rt 144]

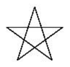
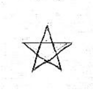
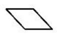
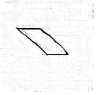
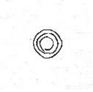
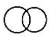
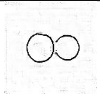
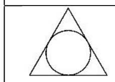
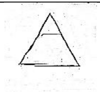
4.3.2 Complex Dataset with caption attention, Fake Image while Training

This is the code generated when we pass a fake image generated from the generator trained on the complex dataset, to the program generator. We can see that the model is able to capture the structure of the logo code. The code is also syntactically correct.

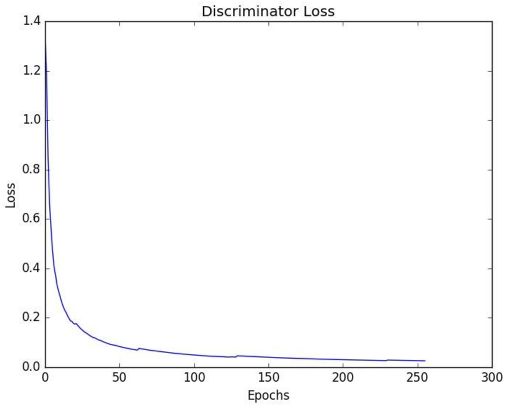
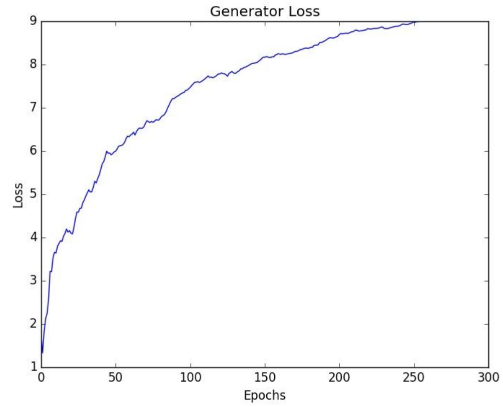
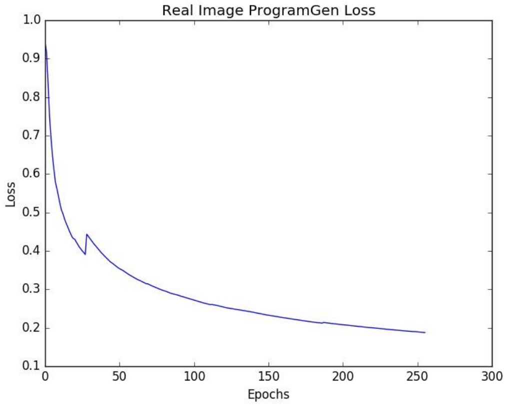
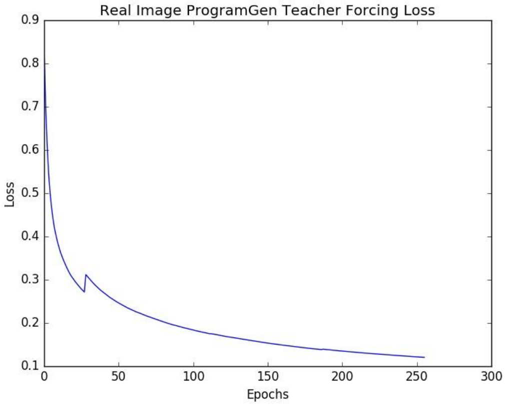
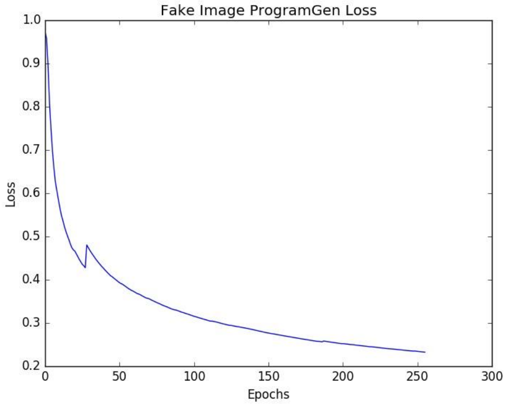
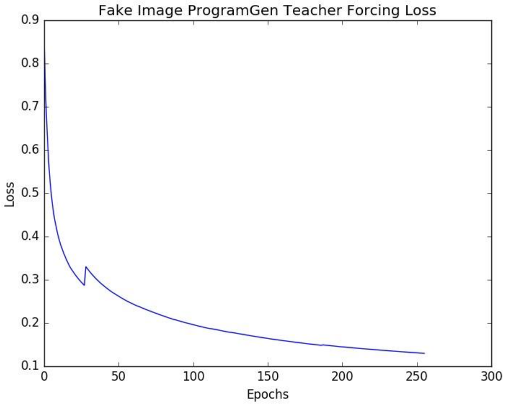
Actual Code	circle 84 pu rt 90 fd 43 rt 270 pd circle 127	repeat 6 [fd 302 rt 144]
Epoch 0	circle 1	repeat [[
Epoch 15	circle 92 pu rt 90 fd 33 rt 270 pd circle 116	repeat 6 [fd 330 rt 144]
Epoch 30	circle 88 pu rt 90 fd 44 rt 270 pd circle 124	repeat 6 [fd 394 rt 144]
Epoch 45	circle 82 pu rt 90 fd 47 rt 270 pd circle 127	repeat 6 [fd 363 rt 144]
Epoch 60	circle 88 pu rt 90 fd 48 rt 270 pd circle 127	repeat 6 [fd 304 rt 144]
Epoch 75	circle 82 pu rt 90 fd 49 rt 270 pd circle 127	repeat 6 [fd 303 rt 144]
Epoch 90	circle 82 pu rt 90 fd 49 rt 270 pd circle 122	repeat 6 [fd 302 rt 144]
Epoch 105	circle 88 pu rt 90 fd 49 rt 270 pd circle 127	repeat 6 [fd 302 rt 144]
Epoch 120	circle 88 pu rt 90 fd 47 rt 270 pd circle 127	repeat 6 [fd 302 rt 144]
Epoch 135	circle 88 pu rt 90 fd 49 rt 270 pd circle 127	repeat 6 [fd 302 rt 144]
Epoch 150	circle 88 pu rt 90 fd 49 rt 270 pd circle 127	repeat 6 [fd 302 rt 144]
Epoch 165	circle 88 pu rt 90 fd 49 rt 270 pd circle 127	repeat 6 [fd 302 rt 144]
Epoch 180	circle 88 pu rt 90 fd 49 rt 270 pd circle 127	repeat 6 [fd 302 rt 144]
Epoch 195	circle 82 pu rt 90 fd 45 rt 270 pd circle 124	repeat 6 [fd 302 rt 144]
Epoch 210	circle 88 pu rt 90 fd 49 rt 270 pd circle 127	repeat 6 [fd 302 rt 144]
Epoch 225	circle 88 pu rt 90 fd 49 rt 270 pd circle 127	repeat 6 [fd 302 rt 144]
Epoch 240	circle 88 pu rt 90 fd 49 rt 270 pd circle 127	repeat 6 [fd 302 rt 144]

4.3.3 Complex Dataset with caption attention, while Testing

This table shows the final results of the model. We can see that the fake image produced by the generator is not very similar to the actual image because of the complexity. So, the main reason that the program generated is not that good because the fake image produced by the generator is not that good.

Real Image	Fake Image	Real Code	Fake Code
		repeat 6 [fd 481 rt 144]	repeat 6 [fd 491 rt 144]
		fd 237 lt 47 fd 241 lt 133 fd 237 lt 47 fd 241	fd 231 lt 47 fd 231 lt 133 fd 239 lt 47 fd 23
		circle 62 pu rt 90 fd 33 rt 270 pd circle 95 pu rt 90 fd 47 rt 270 pd circle 142	circle 62 pu rt 90 fd 36 rt 270 pd circle 93 pu rt 90 fd 49 rt 270 pd circle 140
		circle 111 pu lt 90 fd 111 rt 90 fd 147 rt 90 fd 104 rt 270 pd circle 104	circle 110 pu lt 90 fd 111 rt 90 fd 143 rt 90 fd 101 rt 270 pd circle 104
		repeat 3 [fd 667 lt 120] pu fd 333.5 pd circle 385.09	repeat 3 [fd 675 lt 120] pu fd 334.5 pd circle 302.5

4.3.4 Complex Dataset with caption attention - Plots

Discriminator Loss	Generator Loss
	
Real Image ProgramGen Loss	Real Image ProgramGen Teacher Forcing Loss
	
Fake Image ProgramGen Loss	Fake Image ProgramGen Teacher Forcing Loss
	

5. Conclusion

5.1 Extensions/ Further work

Following are some further experiments that can be performed

- To determine the true extent to which the annotation vectors from images help in predicting accurate dimension sizes in program synthesis, we should experiment and compare the results without using the VGG network generated annotation vectors.

- Experiment and determine how the results compare with a Baseline model for the task, like a sequence to sequence to model that takes text caption as input and generates logo program as output

- Quantify the results using some metric like L2 loss between the images created by synthesized code and actual logo code.

Our prime vision with the project is to develop our model into an interactive educational platform for language Logo. The key aspect of our model is that information is mapped from textual space to visual space, which is also the output space of the logo program. Hence, our work can be extended towards tasks like program reconstruction from incomplete output(images). Consider the case of a student who wrote a buggy Logo code with the aim to generate a certain pattern and now cannot debug the code. Our model can extended to create correct Logo code from the wrong/incomplete image provided by the student along with the specifications from the student about what it intended to draw. If the student provides the intended specifications, our model can generate the correct Logo code and hence prove to be a good debugging tool. Recently, there is work done by Evan et al. [9] that attempts to solve the above problem of creating a complete program from the incomplete, buggy programs. Our work can also be extended along the similar lines to be converted into an educational platform.

5.2 Acknowledgements

Besides the invaluable comments and suggestions from our mentor throughout the course of this project, we are also thankful to B.V. Raghav and Vinod Kumar Kurmi for their help regarding the training issues of the model. We also acknowledge the contributions of Apurv Gupta, who worked with us on the project as part of his Undergraduate Project in Mathematics and Statistics Department. Our code is largely built over Parth Neekhara's famous github repository on conditional DCGAN. Several issues raised over his github repository helped us tune our model to our needs.

References

[1] http://www.steveharrell.com/computer/mslogocommands.html

[2] Goodfellow, Ian, et al. "Generative adversarial nets." Advances in neural information processing systems. 2014.

[3] Devlin, Jacob, et al. ”Robustfill: Neural program learning under noisy I/O.” arXiv preprint arXiv:1703.07469 (2017).

[4] Reed, Scott, et al. ”Generative adversarial text to image synthesis.” arXiv preprint arXiv:1605.05396 (2016).

[5] Xu, Kelvin, et al. ”Show, attend and tell: Neural image caption generation with visual attention.” International conference on machine learning. 2015.

[6] Donahue, Jeff, Philipp Krhenbhl, and Trevor Darrell. ”Adversarial feature learning.” arXiv preprint arXiv:1605.09782 (2016).

[7] Salimans, Tim, et al. ”Improved techniques for training gans.” Advances in Neural Information Processing Systems. 2016.

[8] Kiros, Ryan, et al. ”Skip-thought vectors.” Advances in neural information processing systems. 2015.

[9] Hernandez, Evan, Ara Vartanian, and Xiaojin Zhu. ”Program Synthesis from Visual Specification.” arXiv preprint arXiv:1806.00938 (2018).

[10] https://github.com/paarthneekhara/text-to-image

[11] https://github.com/pytorch/examples/tree/master/dcgan