ALEXANDRIA UNIVERSITY

The Story Builder *Application*

by

Aliaa Othman
Mark Philip
Paula Boules
Peter Atef
Radwa ElMasry
Shereen Gamal

supervised by

Dr. Ahmed Elsayed

A Graduation Project for the fulfillment of the degree of Bachelor's of Engineering

in the
Faculty *of* Engineering
Computer & Systems Department

July 2018

"The true sign of intelligence is not knowledge, but imagination!"

Albert Einstein

Abstract

Who can tell a story with limitless imagination, better than kids. Our idea is just to give them the tool to bring it to life. A mobile application to assist kids - with a little help of their parents - in creating their own stories. To reach kids' imagination level, our application doesn't depend on a limited database, rather it draws the images itself. Our idea relies on generating images based on a textual description by the aid of a machine trained using Generative Adversarial Networks *GANs* which uses a large dataset to train a Model - basically a Neural Network - to draw images conditioned on description.

Acknowledgements

We have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. We would like to extend our sincere thanks to all of them.

We are highly indebted to *Dr.Ahmed Hassan Elsayed* for his guidance and constant supervision as well as for providing necessary information regarding the project and also for his support in completing the project.

Last, but not least, to our family, thank you for encouraging us in all of our pursuits and inspiring us to follow our dreams. We are grateful for all of your support emotionally and financially...

Contents

Abstract ii

Acknowledgements iii

List of Figures vii

List of Tables x

Abbreviations xi

1 Introduction 1
- 1.1 The Addressed Problem . . . 1
- 1.2 Our Solution in brief . . . 1
- 1.3 Impact of the Solution . . . 3
- 1.4 Structure of this document . . . 3

2 System Structure and Conceptual Design 5
- 2.1 System Modules . . . 5
  - 2.1.1 Android Application Module . . . 6
  - 2.1.2 GAN Module . . . 7
  - 2.1.3 Server Module . . . 7
- 2.2 Example scenario . . . 8

3 Background : Generative Adversarial Networks (GANs) 9
- 3.1 Introduction . . . 9
- 3.2 what is a GAN? . . . 10
  - 3.2.1 GAN Architecture . . . 10
  - 3.2.2 Generative vs. Discriminative . . . 10
- 3.3 How GANs Work . . . 11
- 3.4 Parts of training GAN . . . 13
- 3.5 Generative Adversarial Text to Image Synthesis . . . 13
  - 3.5.1 Method . . . 15

4 GAN Adaptation 16
- 4.1 Testing Pre-trained Models : Flowers . . . 16
- 4.2 Building Naive Model . . . 16

4.2.1 Hybrid model idea . . . 16
4.2.2 Hybrid model training set . . . 18
4.2.3 Training phase . . . 19
4.2.4 Results . . . 20
4.3 Data Preprocessing . . . 21
4.3.1 Policy of Images Collection . . . 21
4.3.2 Challenges . . . 21
4.3.2.1 Image various sizes and extensions . . . 22
Image editor program solution: . . . 22
Python script using Pillow: . . . 22
4.3.2.2 Image Description Policy and Guidelines . . . 25
Data Set description policy: . . . 25
DataSetManualMaker Beta program: . . . 25
Description process in action: . . . 26
Description Guidelines: . . . 27
4.3.2.3 Description Misspelling . . . 27
Python script using Enchant: . . . 27
Sample Runs: . . . 28
4.4 Training the New Model and Testing Results . . . 29
4.5 Decomposing the Model into smaller ones . . . 31

5 Story Builder Application 33
5.1 Front-End : Android Application . . . 33
5.1.1 Detailed software design . . . 34
5.1.1.1 Use Case Diagrams . . . 34
5.1.1.2 Sequence Diagrams . . . 38
Story Related Sequence Diagrams . . . 38
Scene Related Sequence Diagrams . . . 40
Entity Related Sequence Diagrams . . . 42
5.1.1.3 Class Diagram . . . 44
5.1.1.4 Database model . . . 49
Story Model . . . 49
Scene Model . . . 50
Entity Model . . . 50
5.1.2 User interface and main scenarios . . . 51
5.1.2.1 Main user interfaces . . . 51
Application entry UIs: . . . 51
Story creation UIs: . . . 51
Story Options . . . 51
Scenes creation UIs: . . . 52
Scenes Thumbnails: . . . 53
Create Entity UIs: . . . 54
Create Entity using Google Mic UIs: . . . 54
Entity can be dragged and dropped: . . . 55
Entity can be Resized and Rotated: . . . 56
Entity can be requested from GAN or Google: . . . 56
5.1.2.2 Complete Scenario . . . 58

5.1.3 Error Handling . . . 61
5.1.3.1 Story name faults . . . 61
5.1.3.2 Server Requests . . . 62
5.2 Back-End : Google Cloud Platform . . . 63
5.2.1 Instance Specification . . . 63
General Specification . . . 63
Boot disk and local disks . . . 63
Network Specification . . . 63
5.2.2 GAN Installation . . . 64
5.2.2.1 Requirement Libraries . . . 64
5.2.2.2 Training Process . . . 64
5.2.2.3 Testing Process . . . 64
5.2.2.4 Our Results . . . 64
5.2.3 Back-end Description . . . 65
5.2.4 Front-end Request Handling . . . 66
5.2.5 Generated Image enhancement . . . 66

6 Future Work **68**
6.1 Application side . . . 68
6.2 GAN model side . . . 68
6.3 Server Side . . . 69

A Appendix **70**
A.1 Steps to train a GAN . . . 70
A.2 Using Generator and Discriminator in image synthesis . . . 71
A.2.1 Deep symmetric structured joint embedding . . . 71
A.3 Method to generate images . . . 72
A.3.1 Network architecture . . . 72
A.3.2 Matching-aware discriminator (GAN-CLS) . . . 72
A.3.3 Learning with manifold interpolation (GAN-INT) . . . 73
A.3.4 Experimental Results of GAN . . . 74

References **76**

List of Figures

1.1 the interaction between back-end and front-end 2

2.1 Use case diagram 5
2.2 Example scenario 8

3.1 GAN I/OProcessing 11
3.2 GAN Structure 12
3.3 GAN Discriminator & Generator 12
3.4 GAN Discriminator & Generator in Training Pass 1 14
3.5 GAN Discriminator & Generator in Training Pass 2 14

4.1 the flower shown has orange petal 17
4.2 the flower shown has coloured petal 17
4.3 the flower shown has white petal giant yellow anther 18
4.4 the flower shown has white petal tiny yellow anther 18
4.5 Model training set elements 19
4.6 Training input pair 19
4.7 Results of Hybrid model 20
4.9 Alice the bird before resizing 24
4.10 Bob the bird before resizing 24
4.11 Alice the bird after resizing 24
4.12 Bob the bird after resizing 24
4.13 Alice the bird final result 24
4.14 Bob the bird final result 24
4.15 Checking descriptions until error is found 29
4.16 Description misspelling is corrected by user input from one trail 29
4.17 Description misspelling is corrected by user input with more than one trail 29
4.18 Testing GAN results 1 30
4.19 Testing GAN results 2 30
4.20 Testing GAN results 3 30
4.21 Testing GAN results 4 31
4.22 Testing GAN after decomposition 1 31
4.23 Testing GAN after decomposition results 2 32
4.24 Testing GAN after decomposition results 3 32

5.1 Use case diagram for story 34
5.2 Use case diagram for scene 35
5.3 Use case diagram for entity 36
5.4 Seq. of creating story object 38

5.5 Seq. of renaming story object . . . 39
5.6 Seq. of deleting story object . . . 39
5.7 Seq. of showing story object . . . 40
5.8 Seq. of opening a story object . . . 40
5.9 Seq. of creating scene object . . . 41
5.10 Seq. of opening scene object . . . 41
5.11 Seq. of deleting scene object . . . 41
5.12 Seq. of creating entity object . . . 42
5.13 Seq. of performing action on entity object . . . 43
5.14 Seq. of deleting entity object . . . 43
5.15 Class Diagram . . . 44
5.16 ERD diagram . . . 49
5.17 Screen shot of the external app icon . . . 51
5.18 Screen shot of the splash screen . . . 51
5.19 Screen shot of empty bookshelf . . . 52
5.20 Screen shot of created stories on the bookshelf . . . 52
5.21 Screen shot of story has no scenes yet . . . 52
5.22 Screen shot of story has 3 new scenes . . . 52
5.23 Screen shot of usual story scene to be thumbnail . . . 53
5.24 Screen shot of scene with thumbnail . . . 53
5.25 Screen shot of scene delete option . . . 53
5.26 Screen shot of entity description . . . 54
5.27 Screen shot of the entity after creation . . . 54
5.28 Screen shot of entity description by Mic . . . 55
5.29 Screen shot of output of entity description by Mic . . . 55
5.30 Screen shot of entity before drag and drop . . . 55
5.31 Screen shot of entity after drag and drop . . . 55
5.32 Screen shot of entity after resizing . . . 56
5.33 Screen shot of entity after rotation . . . 56
5.34 create a story . . . 58
5.35 list of books after creation . . . 58
5.36 list of scenes after creation . . . 59
5.37 create a scene . . . 59
5.38 create a new entity . . . 59
5.39 objects after request . . . 59
5.40 write a narration . . . 60
5.41 thumbnail . . . 60
5.42 screen options . . . 60
5.43 display mode . . . 60
5.44 Story name repeated error handling . . . 61
5.45 Empty story name error handling . . . 61
5.46 Bad internet connection during request handling . . . 62
5.47 Sequence Diagram to handle request . . . 67

A.1 GAN Architecture . . . 72

A.2 Zero-shot generated bird images using GAN, GAN-CLS, GAN-INT and GAN-INT-CLS. We found that interpolation regularizer was needed to reliably achieve visually-plausible results. 75
A.3 Zero-shot generated flower images using GAN, GAN-CLS, GAN-INT and GAN-INT-CLS. All variants generated plausible images. Although some shapes of test categories were not seen during training (e.g. columns 3 and 4), the color information is preserved 75

List of Tables

4.1 DataSetManualMaker program UI illustration 26

5.1 Table story model attributes 49
5.2 Table of scene model attributes 50
5.3 Table of entity model attributes 50
5.4 Illustration of scene components 57
5.5 instance specification in google cloud 63
5.6 Boot disk and local disks specification in google cloud 63
5.7 Network specification in google cloud 63
5.8 our results of training data 65

Abbreviations

AI	**A**rtificial **I**ntelligence
ML	**M**achine **L**earning
GAN	**G**enerative **A**dversarial **N**etwork
FAB	**F**loating **A**ction **B**utton

Dedicated to our families...

Chapter 1

Introduction

Story Builder Application using Generative Adversarial Networks (GANs).This application Uses Machine learning and Artificial Intelligence to build the generic model that generate images from given captions, These images are used to form the story for the children .

Our application falls under machine learning, deep learning and artificial intelligence .

1.1 The Addressed Problem

We thought about how to widen children's imagination, help them to learn new things and enjoy their time in a beneficial way. As we see now children use mobile phone may be better than children in the past so we thought how to exploit this for their sake.
Our main idea is to generate clip-arts/paintings from input text or speech using machine learning techniques such as GANs. The application which wraps our idea is a story builder for children.The child tells his caption or write it then this caption is send to the server that passes it to the model.The model generates the image and return it back to the child.The output is the generated images in the story.
The child can interact with these objects (e.g resizing , moving , rotating), make a sequence of scenes and finally save his story.

1.2 Our Solution in brief

Our solution in our story builder application is using GAN networks that generate images from captions, while the current solution is searching for the image in Google images for example, but Google is limited by its images and can't get images that don't exist

in Google images. But by using GAN networks we can create images that don't exist in any place or even reality .

High level functional description of the our solution:

1. input text or speech.
2. interact with the output objects.
3. resize, move, rotate.
4. add and delete objects.
5. add new scenes.
6. save the story.
7. show all my stories.

figure 1.1 show the interaction between the front-end and back-end when the child use the application.

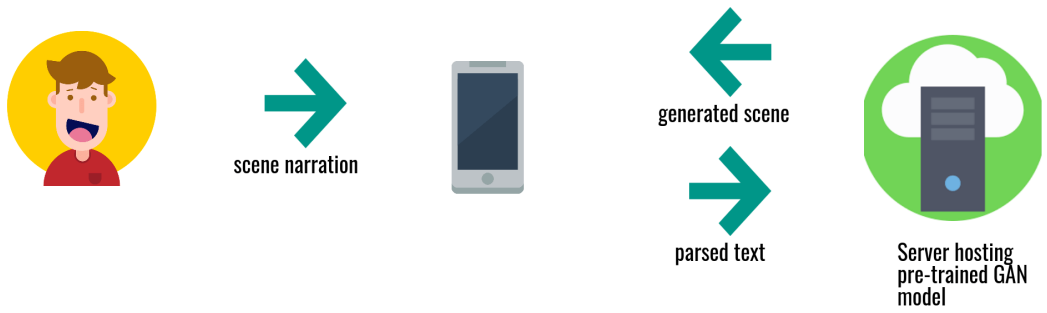


FIGURE 1.1: the interaction between back-end and front-end

1.3 Impact of the Solution

The impact of this solution can appear in :

- Helping the children imagine their own world by telling the story to the application and let the application create it .
- Improving different skills of children.
- Break their natural fear of speaking in public and describing their ideas and feelings.
- Allowing children to be proactive and socialize by sharing their stories with others.
- The proposed application can be extended to be a supplementary material In educational institutes by customizing it to achieve specific learning objectives, Also it can be customized for other commercial non-educational domains by extending the idea of scene creation based on narration .
- There is a ***social impact*** when sharing stories with others, ***commercial impact*** when the idea can be extended and generalized to be used in any other application and ***society impact*** because it helps parents create stories for their children, widen the gaze of helping them imagine and create their own stories.

1.4 Structure of this document

This section illustrate the upcoming chapters in this book :

Chapter 2 System Structure and Conceptual Design: This chapter illustrate the system modules.First it will alk about the application module then about GAN module and finally about server module.

Chapter 3 Generative Adversarial Networks (GANs): This chapter will take you deeper into the GAN networks, We will define first the GAN and then will illustrate how the GAN works and the steps that the GAN take to generate a good model that will be used to generate the image in the application.

Chapter 4 GAN Adaptation: After defining the GAN this chapter will show our work in GAN networks. First we will show the testing pre-trained models and evaluating this tests and then how to built the naive model and how we trained it to generate our

images *then* how we collected the data, did the preprocessing to it, train our model on Google cloud and generate images from captions and finally how we decomposed the model into smaller ones .

Chapter 5 Story Builder Application: In this chapter we will take the reader into our application and illustrate everything in it. *At the beginning* we will consider the application front-end and illustrating software design of our application including the class diagram and sequence diagram and use-case diagram *then* we will talk about database model including the story and scene and entity, Finally we will talk about application interfaces and error handling.
In the end of this chapter we will talk about the back-end part showing how the back end will contact with the front-end, how the request be handle from the back-end then generating the image and return it to the children.

Chapter 6 Future Work: Finally we will illustrate the future work of application and how it can be developed in the future .

Chapter 2

System Structure and Conceptual Design

2.1 System Modules

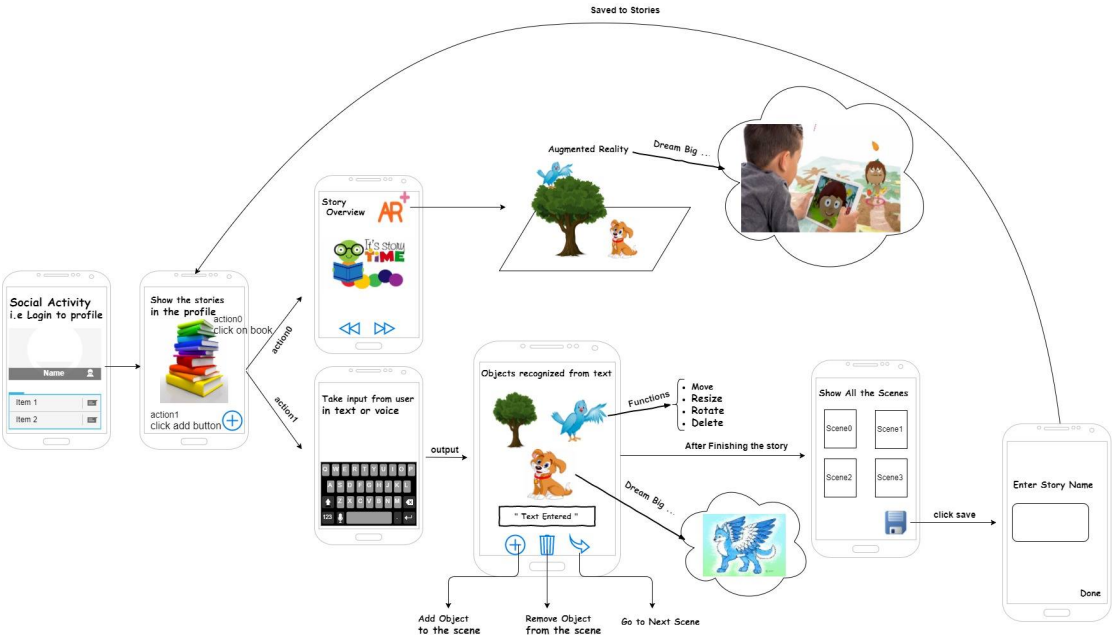


FIGURE 2.1: General Design of the Application

As we see in figure 2.1 the general design of the application , here the figure illustrate the actions that will be take by the children that they will use the application , at first show all books in the application then the children have two option to create a new book or see an existing one , if they choose to create new one the application ask them to enter the caption and generate the image for them then can control this image by

moving them or deleting or resizing after create a scenes the children save the book and return to the books screen.

In the next sections we will illustrate the modules that used in this application and the use of each of them.

2.1.1 Android Application Module

- The *front-end* of our software is an android application, as we see in figure 2.1 the application could be divided into three main screens as follows:

- **The Screen Showing all the Stories**
  This is the entrance screen of our application which shows a list of all the stories created by the user, the user would have two options here:

  1. **Click on a recent story:** Here the application would switch to another screen showing the scenes of the story.
  2. **Click on the Add button:** Here The application would create a new story with zero scenes and the user would have the option to add any scene to his story.

- **The Screen Showing all the Scenes**
  This the screen showing all the scenes of the story, here you see a grid view showing the thumbnails of each scene, the user would have three option here:

  1. **Click on a recent scene:** Here the application would switch to another screen showing the details of the scene i.e all the entities the user has created before on the same positions as he saved them.
  2. **Click on the Add button:** Here The application would create a new scene with zero entities and the user would have the option to add any entity to his story.
  3. **Click on the Show Story button:** Here the story is presented like a book where you can see your scenes one after another in a slide-like view.

- **The Scene Screen**
  This is the screen that opens when the user opens a recent scene or creates a new one, you can imagine this screen a beautiful colorful background and the entities are placed on it on the desired positions of the user, Here the user has many options to do:

1. **Click on the Add button:** Here the application would open a pop-up dialogue which have to options:
   (a) Enter the description of your entity as a text.
   (b) Use speech recognition to describe your entity by your voice.

   After sending the description to the server, the image is fetched from the module and appears on the screen of the scene.
2. **Click on an entity:** Here The user would have many options to do on the selected entity, the options are:
   (a) *Delete* the entity.
   (b) *Move* the entity to any desired place within the area of the screen.
   (c) *Rotate* the entity with any desired angle.
   (d) *Re-size* the entity to be smaller or bigger.

- The *database* of the application is sql-lite based and it consists of three tables. One to keep the information of the *story*, one to keep the *scenes* associated with each story, and the other for keeping the *entities* in each scene with their images and description. keeping all these information on the database allows the user to see all his created stories when he opens the application.

2.1.2 GAN Module

- This module is concerned with generating the desired image from the description of the user. The module repeat the process of generating the image every time i.e if you entered the same description many times, it would return a different image every time.

2.1.3 Server Module

- The *server (back-end)* of our application acts as the mediator between the *android application* module and the *GAN* module.
- The server is based on *Google Cloud* platform.
- The process is done as follows:
  1. The description entered by the user is sent from the application to the server.
  2. the server then deliver this description to the GAN module.
  3. the GAN module would generate the desired image on the server.
  4. The image is sent back to the application.

2.2 Example scenario

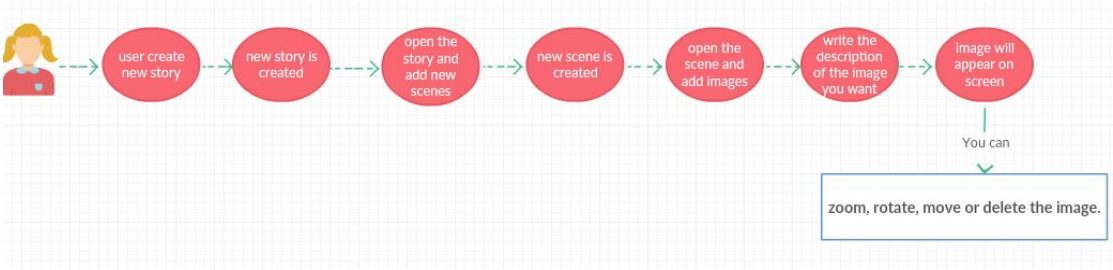


FIGURE 2.2: an example of a simple scenario.

Here is an explanation of a simple scenario shown in 2.2 :

1. User creates a new story, giving it a name.
2. The story is created and appears at the screen with the given name.
3. The user then, open the story to find it empty i.e without any scenes.
4. User adds a new scene.
5. The scene opens with zero entities.
6. To add a new entity, user enters the description of his entity either by text or voice.
7. The image of the entity appears on the screen.
8. User then can zoom, rotate, or delete the entity.

Chapter 3

Background : Generative Adversarial Networks (GANs)

"There are many interesting recent development in deep learning The most important one, in my opinion, is adversarial training (also called GAN for Generative Adversarial Networks). This, and the variations that are now being proposed is the most interesting idea in the last 10 years in ML, in my opinion."

Yann LeCun.

3.1 Introduction

Neural Networks have made great progress. They now recognize images and voice at levels comparable to humans. They are also able to understand natural language with a good accuracy But, even then, the talk of automating human tasks with machines looks a bit far fetched. After all, we do much more than just recognizing image/voice or understanding what people around us are saying.

There has been a large resurgence of interest in generative models recently . These are models that can learn to create data that is similar to data that we give them. The intuition behind this is that if we can get a model to write high-quality news articles for example, then it must have also learned a lot about news articles in general. Or in other words, the model should also have a good internal representation of news articles. We can then hopefully use this representation to help us with other related tasks, such as classifying news articles by topic.

Actually training models to create data like this is not easy, but in recent years a number of methods have started to work quite well. One such promising approach is using Generative Adversarial Networks (GANs). The prominent deep learning researcher and director of AI research at Facebook, Yann LeCun, recently cited GANs as being one of the most important new developments in deep learning.

3.2 what is a GAN?

Generative Adversarial Networks are a set of models that basically learn to create synthetic data that is similar to input data it's given. In more formal terms, a GAN is a generative model that learns the probability distribution (or data distribution) of the training examples it is given. From this distribution, we can then create sample outputs. GANs have seen their largest progress with image training examples, but this idea of modeling data distributions is one that can be applied with other forms of input.

3.2.1 GAN Architecture

GANs are an interesting idea that were first introduced in 2014 by a group of researchers at the University of Montreal lead by Ian Goodfellow (now at OpenAI). The main idea behind a GAN is to have two competing neural network models. One takes noise as input and generates samples (and so is called the generator). The other model (called the discriminator) receives samples from both the generator and the training data, and has to be able to distinguish between the two sources. These two networks play a continuous game, where the generator is learning to produce more and more realistic samples, and the discriminator is learning to get better and better at distinguishing generated data from real data. These two networks are trained simultaneously, and the hope is that the competition will drive the generated samples to be indistinguishable from real data.

figure 3.1 [1] show the input and output processing of GAN networks and how the generator and discriminator works with each other .

3.2.2 Generative vs. Discriminative

A discriminative model learns a function that maps the input data (x) to some desired output class label (y). In probabilistic terms, they directly learn the conditional distribution P(y|x).

[1] figure GAN I/O Processing link

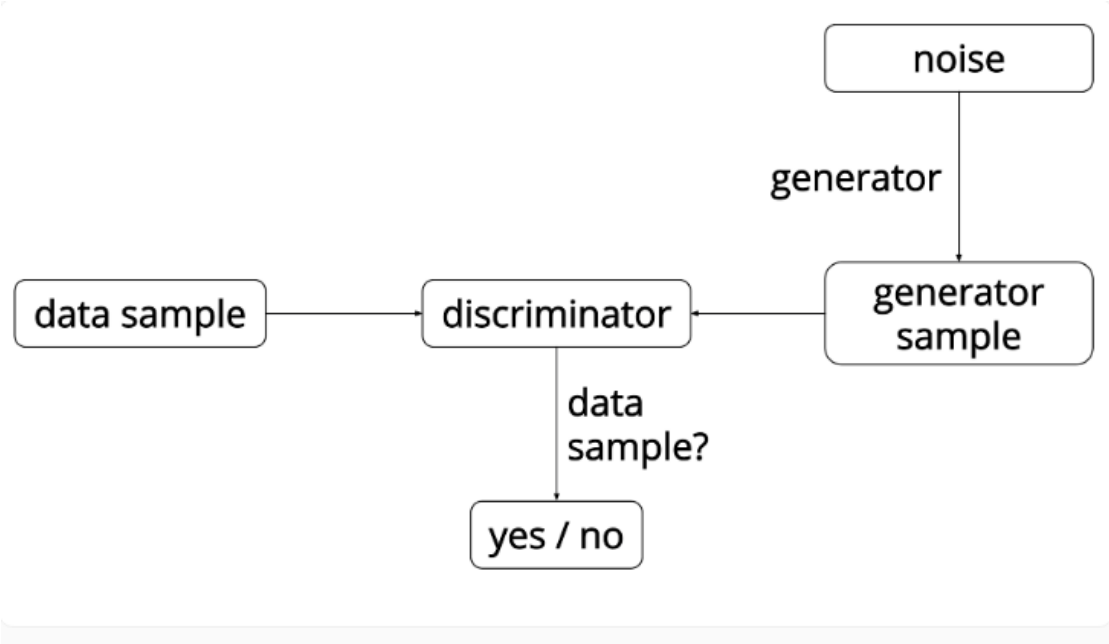


FIGURE 3.1: GAN I/O Processing

A generative model tries to learn the joint probability of the input data and labels simultaneously, i.e. P(x,y). This can be converted to P(y|x) for classification via Bayes rule, but the generative ability could be used for something else as well, such as creating likely new (x, y) samples.

The structure of two models shown in figure 3.2 [2] .

Both types of models are useful, but generative models have one interesting advantage over discriminative models they have the potential to understand and explain the underlying structure of the input data even when there are no labels. This is very desirable when working on data modelling problems in the real world, as unlabelled data is of course abundant, but getting labelled data is often expensive at best and impractical at worst.

3.3 How GANs Work

There are two main components of a GAN Generator Neural Network and Discriminator Neural Network .

As shown in figure 3.3 [3] the Generator Network takes an random input and tries to generate a sample of data.we can see that generator G(z) takes a input z from p(z),

[2] figure GAN Structure link
[3] figure GAN Discriminator and Generator link

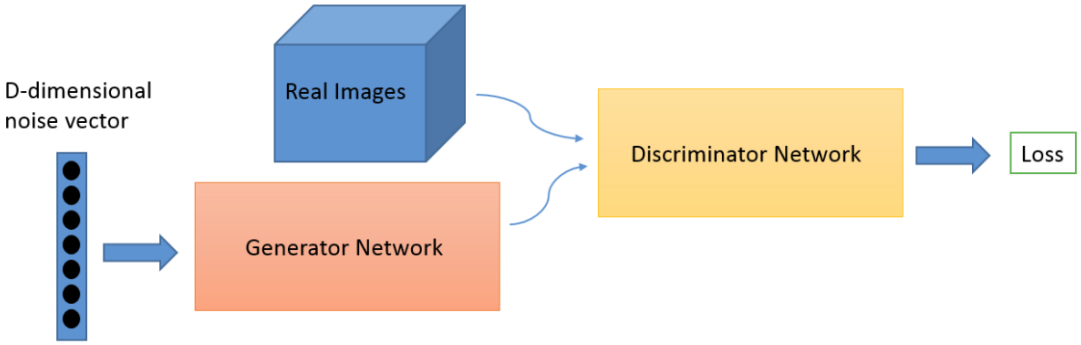


FIGURE 3.2: GAN Structure

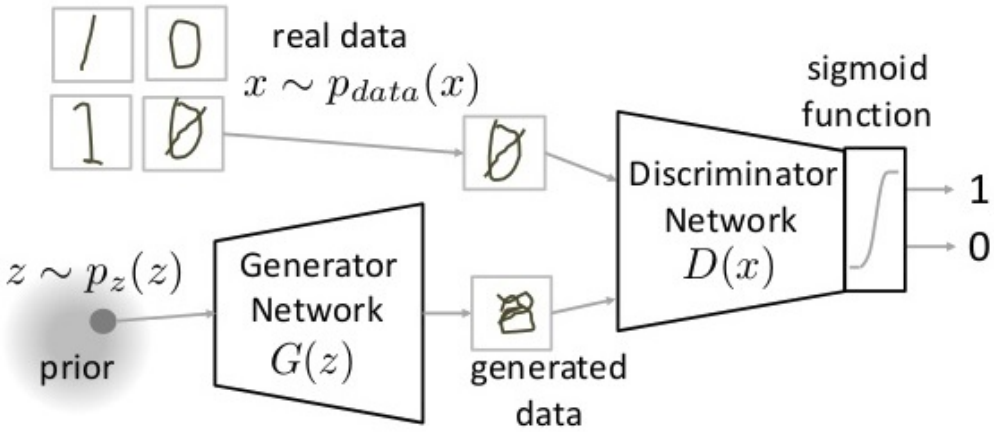


FIGURE 3.3: GAN Discriminator & Generator

where z is a sample from probability distribution p(z). It then generates a data which is then fed into a discriminator network D(x). The task of Discriminator Network is to take input either from the real data or from the generator and try to predict whether the input is real or generated. It takes an input x from pdata(x) where pdata(x) is our real data distribution. D(x) then solves a binary classification problem using sigmoid function giving output in the range 0 to 1.

Let us define the notations we will be using to formalize our GAN, Pdata(x) is the distribution of real data

- X $\rightarrow$ sample from pdata(x)
- P(z) $\rightarrow$ distribution of generator
- Z $\rightarrow$ sample from p(z)
- G(z) $\rightarrow$ Generator Network

- D(x) → Discriminator Network

Now the training of GAN is done (as we saw above) as a fight between generator and discriminator. This can be represented mathematically as

$$\min_{G} \max_{D} V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[logD(x)] + \mathbb{E}_{x \sim p_z(z)}[log(1 - D(G(z)))] \tag{3.1}$$

In our function V(D, G) the first term is entropy that the data from real distribution (pdata(x)) passes through the discriminator . The discriminator tries to maximize this to 1. The second term is entropy that the data from random input (p(z)) passes through the generator, which then generates a fake sample which is then passed through the discriminator to identify the fakeness . In this term, discriminator tries to maximize it to 0 (i.e. the log probability that the data from generated is fake is equal to 0). So overall, the discriminator is trying to maximize our function V.

On the other hand, the task of generator is exactly opposite, i.e. it tries to minimize the function V so that the differentiation between real and fake data is bare minimum. This, in other words is a cat and mouse game between generator and discriminator.

3.4 Parts of training GAN

So broadly a training phase has two main subparts and they are done sequentially

Pass 1: As shown in figure 3.4[4] train discriminator and freeze generator (freezing means setting training as false. The network does only forward pass and no backpropagation is applied)

Pass 2: As shown in figure 3.5 [5]train generator and freeze discriminator

Steps to train a GAN : see the appendix for more details about the steps .A.1

3.5 Generative Adversarial Text to Image Synthesis

Here we translate the text (single sentence) into image pixels.

Traditionally this type of detailed visual information about an object has been captured in attribute representations distinguishing characteristics the object category encoded into a vector ([1]Farhadi et al., 2009; [2]Kumar et al., 2009; [3]Parikh & Grauman, 2011;

[4] figure GAN Discriminator and Generator pass 1 link
[5] figure GAN Discriminator and Generator pass 2 link

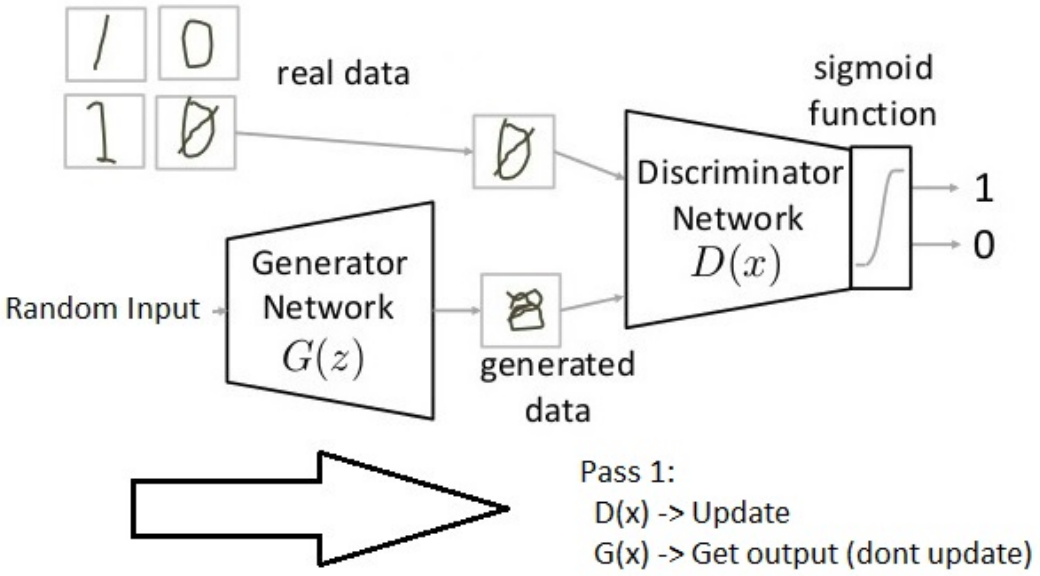


FIGURE 3.4: GAN Discriminator & Generator in Training Pass 1

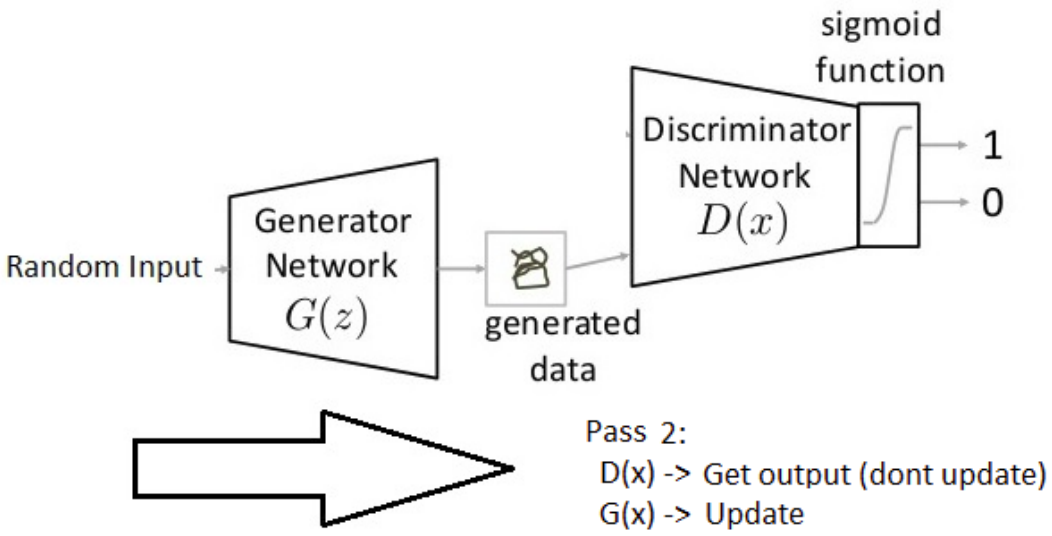


FIGURE 3.5: GAN Discriminator & Generator in Training Pass 2

[4]Lampert et al., 2014), in particular to enable zero-shot visual recognition ([5]Fu et al., 2014; [6]Akata et al., 2015), and recently for conditional image generation ([7]Yan et al., 2015).

This model consists of two big networks generator and discriminator .

Recently, deep convolutional and recurrent networks for text have yielded highly discriminative and generalizable (in the zero-shot learning sense) text representations learned

automatically from words and characters ([8]Reed et al., 2016). These approaches exceed the previous state- of-the-art using attributes for zero-shot visual recognition on the Caltech-UCSD birds database ([9]Wah et al., 2011), and also are capable of zero-shot caption-based retrieval. Motivated by these works, we aim to learn a mapping di- rectly from words and characters to image pixels.

To solve this challenging problem requires solving two sub problems: first, learn a text feature representation that captures the important visual details; and second, use these features to synthesize a compelling image that a human might mistake for real. Fortunately, deep learning has enabled enormous progress in both sub problems - natural language representation and image synthesis - in the previous several years, and we build on this for our current task.

This conditional multi-modality is thus a very natural application for generative adversarial networks ([10]Goodfellow et al., 2014), in which the generator network is optimized to fool the adversarially-trained discriminator into predicting that synthetic images are real. By conditioning both gen- erator and discriminator on side information (also studied by [11]Mirza & Osindero (2014) and [12]Denton et al. (2015)), we can naturally model this phenomenon since the discriminator network acts as a smart adaptive loss function

for more details about using generator and discriminator in image synthesis see .A.2 for structure loss function and classifier parameters see A.2.1

3.5.1 Method

Here we will train a deep convolutional generative adversarial network (DC-GAN) conditioned on text features encoded by a hybrid character-level convolutional-recurrent neural network. Both the generator network G and the discriminator network D perform feed-forward inference conditioned on the text feature .

for more details see A.3

Chapter 4

GAN Adaptation

4.1 Testing Pre-trained Models : Flowers

Here we used the idea of ([13]Scott Reed 2016), trained their idea by using *Google Cloud Platform*, installed requirement libraries, trained the code by using flowers from COCO database and finally test it by giving the query and produce 64 image for the given query. figures 4.1,4.2,4.3,4.4 show the results.

figure 4.1 show the flower has orange petal in most cases in the images .
figure 4.2 show that the keyword coloured doesn't understandable by the network (GAN) in this model.
figure 4.3 show the flower has white petal as described and in some cases we see that the anther is giant as described in the caption. figure 4.4 show the flower has white petal as described and in some cases we see that the anther is tiny as described in the caption.

4.2 Building Naive Model

4.2.1 Hybrid model idea

In this part we tried to test a simple idea which is *"Can GAN create output that it didn't see before ?"*. With more explanation could the GAN be creative enough to be able to generate new images with combinations of features that have never seen before together. For example if we train the GAN to draw **dogs and birds** and it was able to do that very well, can we generate a hybrid mix of these two animals, can we generate a dog with bird wings or a bird with dog head. So we will test our GAN structure if it can do something like that.

FIGURE 4.1: the flower shown has orange petal

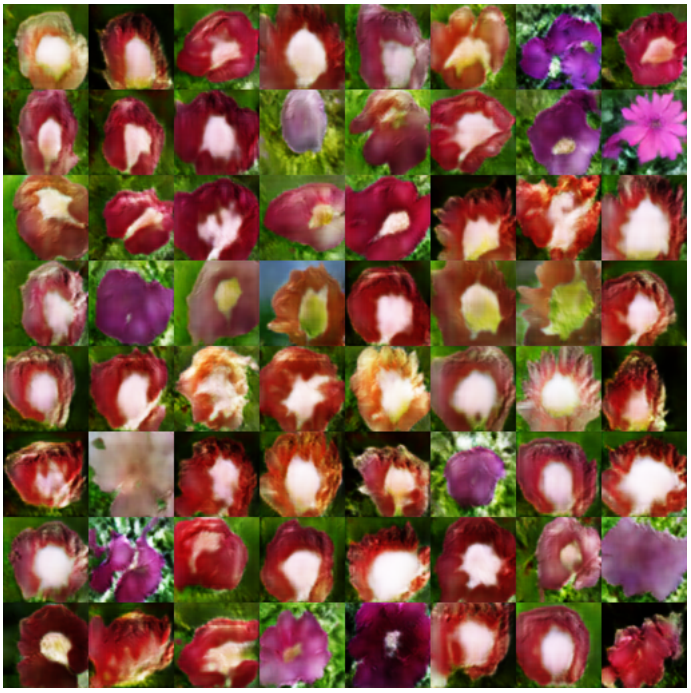


FIGURE 4.2: the flower shown has coloured petal

FIGURE 4.3: the flower shown has white petal giant yellow anther

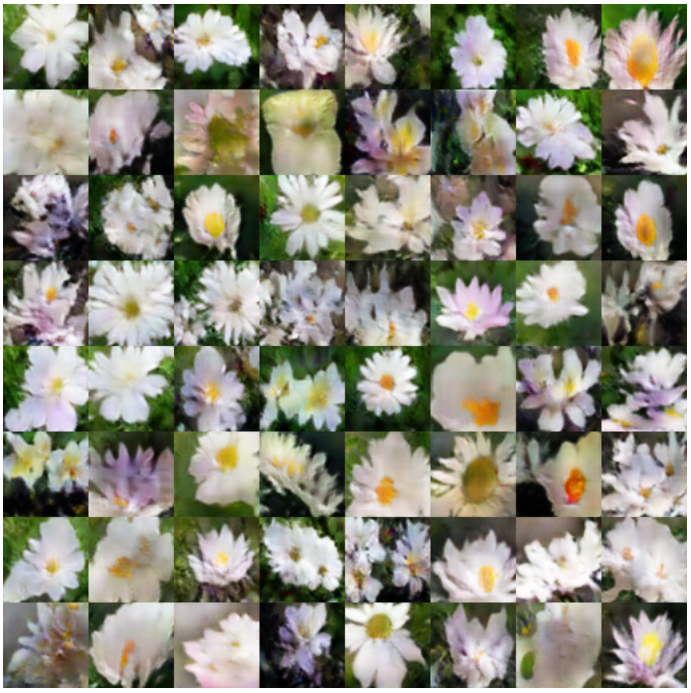


FIGURE 4.4: the flower shown has white petal tiny yellow anther

4.2.2 Hybrid model training set

We started with a very simple case to get results as fast as possible. We used only limited number of shapes and colors, created four shapes with four different colors. Every shape

has only two variations which is the colors.
We think that it is a very easy task to learn compared to the other task the GAN can do. Having that training set we replicate every shape many times until we reached the state where we have 128 shape image with 3 description for each one of them that create a training set of 369 sample.

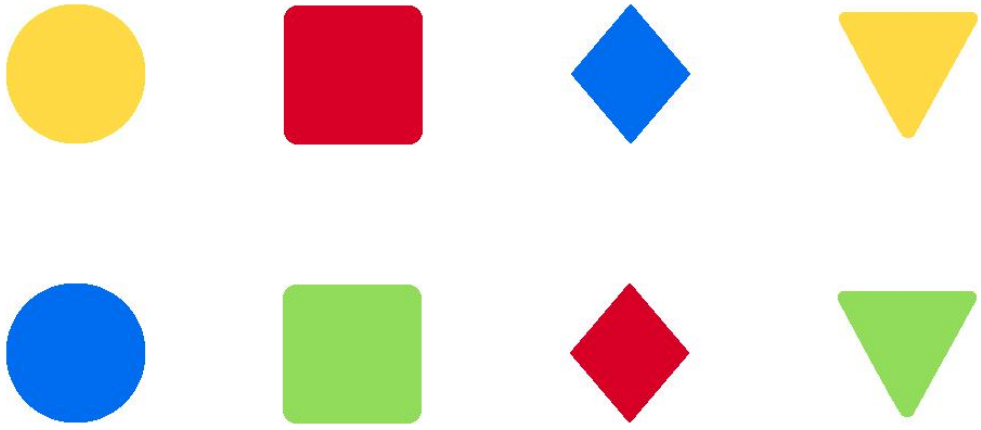


FIGURE 4.5: Model training set elements

The training input is pair of the the image and its description.

FIGURE 4.6: Training input pair

4.2.3 Training phase

We start the training on machine with core i7, NVIDIA GeForce 920MX 2GB GPU and 8GB Memory. The training took about 1 hour at that point the GAN was able to generate images as the ones it saw before. But after that point of training the GAN

start to overfit the data and start to spoil the good results and disfigure the generated images so we had to stop the training process and start to evaluate the results.

4.2.4 Results

After we stopped the training process, we start to analyze the results. Before overfitting the GAN was able to generate images with the same description and combinations of features that it saw before. So that prove the GAN was able to learn to generate images in that short amount of time. But it failed to generate images with descriptions and combinations of features that weren't in the training as "red circle", "blue square", "green rhombus" and "red triangle". The GAN couldn't make the mutation that we was trying to achieve. *It was unable to generate new images with combinations features it have never seen before.*

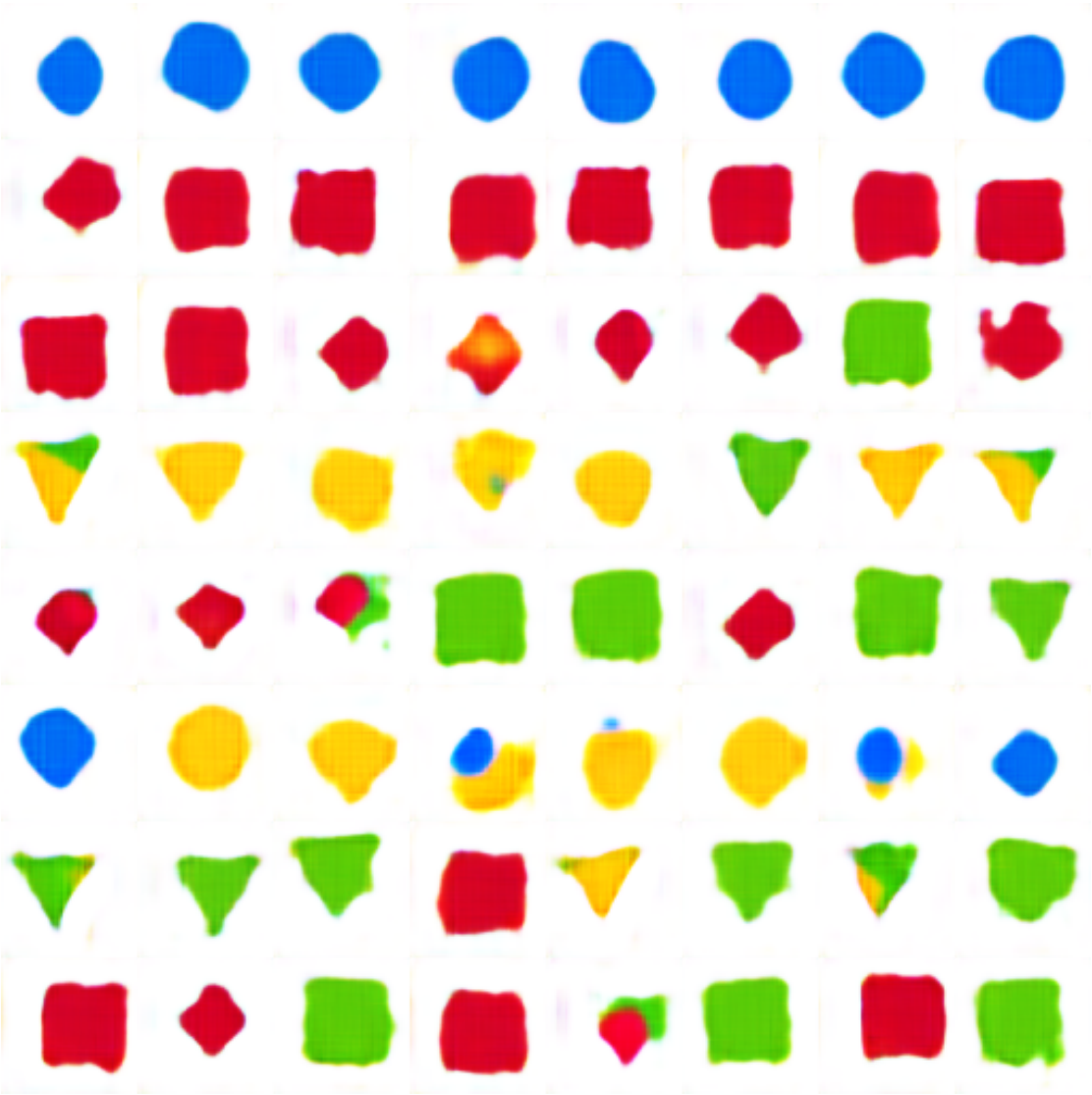


FIGURE 4.7: Results of Hybrid model

So the result we reached is that the GAN with the architecture we used was able to generate the same shape it saw before with the same combinations features but was unable to generate the mutated images we asked it to do.

A new paper was published this year 2018 addressed the same idea of generating a hybrid creature. The output of this technique was very good. They create that output by combining two exiting images for more information you can find it here [14].

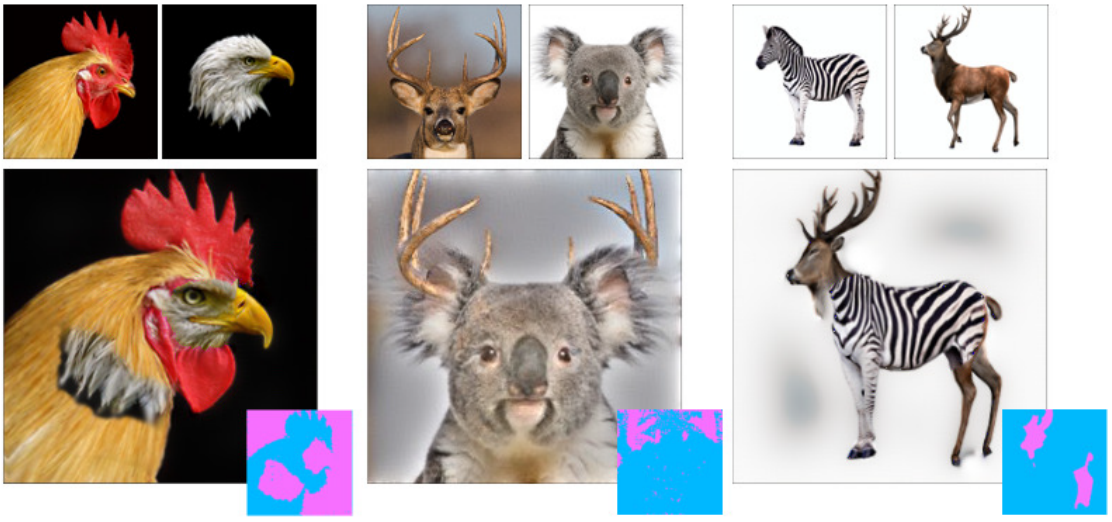


FIGURE 4.8: Output from [14] paper.

4.3 Data Preprocessing

4.3.1 Policy of Images Collection

We tried to have some guides and rules to follow so the training set be consist as much as possible and have enough variation to make the model learn something. The rules we followed while collecting the images for the data set :

- Image must contain only one bird.
- Image must have a white or no background.
- Images should have similar structure like how the bird stand.

4.3.2 Challenges

In this subsection we will highlight the main challenges we met while collecting the dataset which was needed to create our customized GAN model.

4.3.2.1 Image various sizes and extensions

The collected images from the plugin was about 1 thousand photos, these photos had the same domain -which is cartoon birds with white background-. These photos had high variety in terms of images sizes, images extensions.
This challenge arises due to the initial incompatibility between the GAN model which accepts JPG images with fixed size (256x256) and the collected various images. And to *solve* such a problem we had two options

- The first solution was to edit all the photos manually using any *image editor* program, this solution was totally waste of time as we will discuss later.
- The second solution was to write *python* script to edit the images programmatically.

Image editor program solution: This solution aimed to decrease the images' variations, and make them compatible with the GAN model, it based on using an image editor programs as Gimp or Photoshop and doing the following for each image.

1. resize all photos to be 256x256 while keeping the ratios of the image without cropping
2. change transparent pixels into white ones in PNG images
3. export all images as JPG

This solution seems great and doing its job, but not for large number of images, if we assumed that each image will take 2 minutes to be edited efficiently, so we need about 2K minutes approx 34 hours of work.
Some might say that this task can be done in parallel which will decrease the time needed dramatically, unfortunately there was another problem which was that some of these images editing programs affects the image size after editing. Images size will increase, which will increase the size of the data set, which will surly increase the time required for training of then model

Python script using Pillow: This solution emerged from the failure of the first solution, it made use of the superb python library *Pillow* which offers very impressive programming interface to manipulate images in different ways. The algorithm of the

Algorithm 1 Image Conversion Script

```
1: inputDir ← PATH_TO_INPUT_IMAGES
2: outputDir ← PATH_TO_OUTPUT_IMAGES
3: size ← (256, 256)
4: white ← (255, 255, 255){/*set white as array of RGB (255,255,255)*}
5: for each fileName in inputDir do
6:     image ← openImage(fileName)
7:     image ← thumbnail(image, size)
8:     image ← convert(image, RGBA)
9:     if image.getExtention()! = "jpeg" then
10:        image ← removeBackgroundBlackPixels(image)
11:    end if
12:    base ← newImage(size, white)
13:    base ← base.paste(image, FROM_CENTER)
14:    base.save(JPEG, outputDir)
15: end for
```

script is as follows, this script was developed by our team and code can be viewed on Github[1] :

- The first two lines in the algorithm used to locate the paths of the input raw images and the output of the resized images, the lines 3-4 set some constants need for upcoming usage.
- Each file in the input folder is open as image and then thumb-nailed to the desired size i.e. 256x256 pixels, the function of the *thumbnail(image,size)* -in line 7- function is to scale the image by height or width to reach the specified size.

To understand its effect let's see this example:
Consider the two images of these two birds: *Alice* the pink bird, and *Bob* the blue one

As one can notice that Alice in Fig 4.9 has a height larger than its width and Bob in Fig 4.10 has a width larger than its height
The effect of *thumbnail()* in line 7 can be realized in Fig 4.11. The image is matched by height while width is less than 256 pixels and in Fig 4.12 the width is matched to 256 but the height is less than 256 pixels.
Then to fill the gaps in width for Alice and height for Bob, in line 12 we created a new white square called *base*, then in line 13 we pasted in its center the resized image of Alice and Bob, the final results can be seen in Fig 4.13 and 4.14.
Reader must note that the blue dashed lines are for illustration only and not found in real data set

[1] The code for this algorithm can be found in this GitHub link

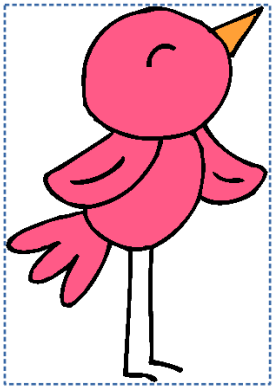


FIGURE 4.9: Alice the bird with dimensions 455x345

FIGURE 4.10: Bob the bird with dimensions 550x420

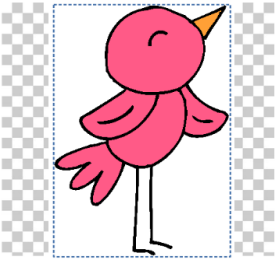


FIGURE 4.11: Alice image height is resized to be 256

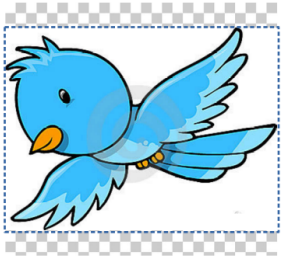


FIGURE 4.12: Bob image width is resized to be 256

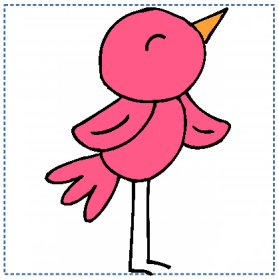


FIGURE 4.13: Alice image dimension is 256x256 with same scale

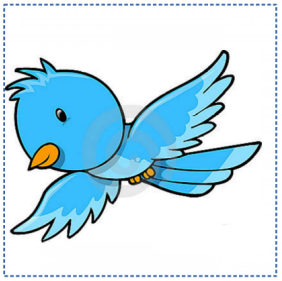


FIGURE 4.14: Bob image dimension is 256x256 with same scale

4.3.2.2 Image Description Policy and Guidelines

After solving the problem of the variety of the images, we faced two problems the first one was *how to define a policy for describing the whole data set ?*, the second problem was *what are the guidelines for describing a given image?*
In the following paragraphs, we will answer those two questions

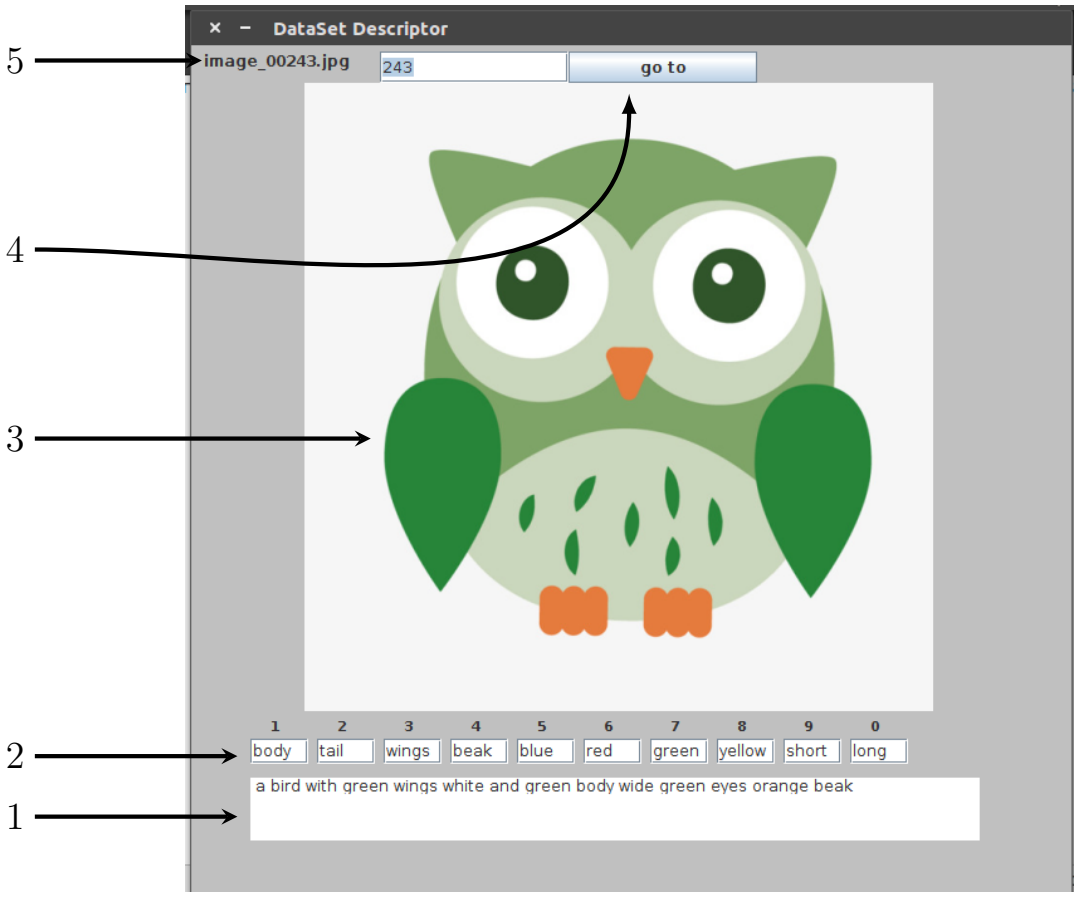
Data Set description policy: The team needed a policy to describe the whole data set efficiently and in reasonable amount of time, the following bullet points will illustrate for the reader the main criteria for such a policy :

- Each image in the data set must have 3 or more descriptions from 3 or more different persons to avoid over-fitting of description i.e. the descriptions would be almost the same if one person described an image with little details 3 times.
- The Description must me automated in terms of keeping track of each description with the corresponding image, such a correspondence will waste time if we handled it by our own hands, it will be prone for confusion of descriptions with images.
- Each description must be manually written by a team member, not automated, i.e. cant use image description AI models, because in this case we would test the accuracy for such the describing model not the GAN itself.
- Each team member may describe no more than 500 images i.e the half of the collected data set.
- Description must be done in parallel between the team member.

Such criteria was the building block to our policy for describing the data set, this policy was established by developing a *JAVA* desktop application *DataSetManualMaker*.

***DataSetManualMaker* Beta program:** A simple Java desktop application was totally developed by the team, and code can be viewed on github[2], to realize the image description policy stated previously. The program main goal was to organize the description process among the team members
The user interface was very easy to deal with and here is the explanation next figure and table 4.1 as shown.

[2]This program can be found in this GitHub link , the README is included



Number	Refers to
1	Description bar where one can write the description there
2	Number-indexed shortcuts for the most frequent words in the describing birds, as (beak , wings , . . . etc)
3	The image to be described
4	Search bar to find a specific image using its number
5	Current image name

TABLE 4.1: DataSetManualMaker program UI illustration

The usage for this program was very trivial, the descriptor write the description in the description bar, according to the image shown, after that the user press **shift** button to save the text in the description bar in a file text with the same name as the image in descriptions folder and then show the next image and so on . . .
If the user wants to access a specific image, he/she can write its name in the search bar, where the arrow number 4 points in the previous figure.

Description process in action: Using the *DataSetManualMaker* program and based on the *policy*, the process of desription was as follows :

Team members divided into two groups each group consists of 3 persons, each group describe one half of the data set, using the *DataSetManualMaker* program, each person write one description on each image. After combining the description folder for each group, each image had got 3 descriptions from 3 different persons in parallel, so as reader can see that all points of the policy criteria was achieved.

Description Guidelines: But to achieve high level of consistency in description , since not one person is describing, and make maximum use of the powerful - GAN - to generate high quality image. The team had to agree on general guidelines for describing each image. The main followed guidelines were :

- Give a name for main kinds of birds which were*a (parrot, hawk, duck flamingo, owls)* else just write *a bird.*
- Describe the color to the main parts of the bird -if possible to see in the image - as *(head , eye pupil, beak, head ,neck, body, wings , tail, legs)* and if there is visible trends in the body as *spots*, you can mention them.
- Give description for the size of parts of the birds if apparent as *(very small, small, large, very large)* or mention nothing if normal or average size.
- Use main colors in description as *(red, blue white, . . . etc)* and for the secondary colors, write the nearest prime color, and if many colors in same part in the body of the bird, write *colorful.*

4.3.2.3 Description Misspelling

After description of the data set, the team wrote more than 30,000 words in total. nother problem aroused which is the *spelling* of the words of the description.
Reader may see it is not big problem, unfortunately, it was a problem for us because the the GAN gives weight for each word in each description, this weight is incremented for each word in a file called *vocab.pickle*, the process of assigning of weight to words will be accurate if misspelled words is introduced in the description of the images.
The solution of manual words checking is - obviously - not feasible, so we chose an automated one, which is writing a script that will handle such functionality with minimal interference from the team.

Python script using Enchant: Using *enchant* python library, it was trivial step to eliminate the misspelled words from our description, we wrote a small script which

-with minimal manual interference- removed all the misspelled words.

The algorithm of the script is developed wholly by our team and the code can be viewed on GitHub [3]. The idea of the algorithm is to traverse over all the words of the description, each word is checked to be whether well-spelled or not using English dictionary in *enchant*, if it is misspelled, the algorithm will pause execution and ask the user to type the word correctly, the pseudo code of the algorithm is as follows :

Algorithm 2 Description Spelling Checker

```
dictionary ← enchant.getDictionary("ENGLISH")
inputDir ← PATH_TO_INPUT_IMAGES
outputDir ← PATH_TO_OUTPUT_IMAGES
for each fileName in inputDir do
    description ← openFile(fileName)
    words ← description.split()
    newWords ← newarray
    for index in range (0, words.length) do
        word ← word.trim()
        while word is not in the dictionary do
            print(fileName + "" + word)
            word ← user_input() {loop until user enter correct word}
        end while
        newWords[index] ← word
    end for
    newDescription ← newWords.string
    save(outputDir/fileName, newDescription)
end for
```

Sample Runs: This paragraph shows the results of running the script over our description, if the script finds no error in a description it prints out *image name => done* and so on until misspelled word is found as in Fig 4.15, after finding the misspelled word, the script will ask the user to type the correct corresponding word as in Fig 4.16, if the the user misspelled the correction, the script will keep asking for the correct word as in Fig 4.17.

[3] The code for this algorithm can be found in this GitHub link

```
image_00132.txt is done
image_00301.txt is done
image_00083.txt is done
image_00370.txt is done
image_00308.txt is done
image_00018.txt is done
image_00052.txt is done
image_00068.txt is done
image_00237.txt is done
image_00282.txt is done
image_00221.txt  ==>  voilet
```

FIGURE 4.15: Checking descriptions until error is found

```
image_00343.txt  ==>  rounf
write it correct : round
image_00343.txt is done
```

FIGURE 4.16: Description misspelling is corrected by user input from one trail

```
image_00132.txt  ==>  whte
write it correct : whize
write it correct : wiiiize
write it correct : abiad
write it correct : white
image_00132.txt is done
```

FIGURE 4.17: Description misspelling is corrected by user input with more than one trail

4.4 Training the New Model and Testing Results

After collecting the date set images and describe each image, we start *the training phase*. We use *Google cloud computing engine* instance with 8 cores and 16G of memory. The training took a long to train cause we couldn't get a GPU instance form Google cloud. We tried the same training on a local machine with core i7, NVIDIA GeForce 920MX 2GB GPU and 8GB Memory. The training time was much less than other one, in fact the training with GPU was 4 times faster than training on a machine without GPU. But that wasn't the maximum speed up as our GPU has only 2G of memory the GAN model

and the training data couldn't all fit in there. So the GPU had to copy the data in and out many times so the bottle neck become the copying of the data not the computation itself. After the training with all the data set we get that results as figures 4.18, 4.19, 4.20 and 4.21 show. We have noticed that the Owl images was performing much better than the other ones so we made more investigation to that situation. We found that most of Owl picture has the same body structure for example the eyes location and the way they stand they had less variation form the other types of birds.

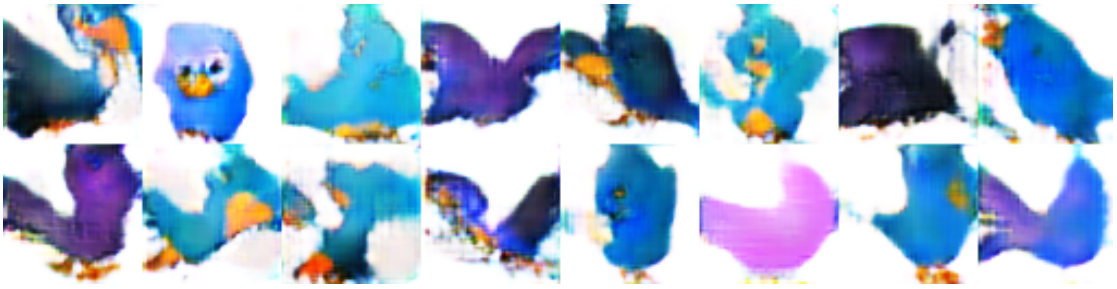


FIGURE 4.18: a bird with blue body orange beak.

FIGURE 4.19: a parrot with green body yellow beak yellow legs brown eyes.

FIGURE 4.20: owl has green body brown legs blue eyes orange beak.

FIGURE 4.21: duck with yellow body orange beak and black eye.

So we thought it may be easier to the GAN to learn to generate only owls and that what we tried next and will talk about in the next section of this chapter. The main factor we think was affecting the result is the variation in the images knowing that we don't have much data to train on we know we lack the huge data the deep learning and GANs usually need. all of that was an alarm to us to try something new so we decided to split and decompose the training and the Birds model.

4.5 Decomposing the Model into smaller ones

On our way to try to improve the quality of the images, we tried to decompose the training and the data set. We split the data set to many sections a section for every kind of birds one for Owls one for Ducks etc. We started with Owls the training here was much faster as we cut down the size of the training set. The result start to appear fast and owls eyes come to life. The model was doing good as figures 4.22, 4.23 and 4.24 show was able to generate much better output than the older model also the data set was much smaller but the variation was way much less the variation in the original data set.

FIGURE 4.22: a purple owl with big black eyes yellow legs orange nose.

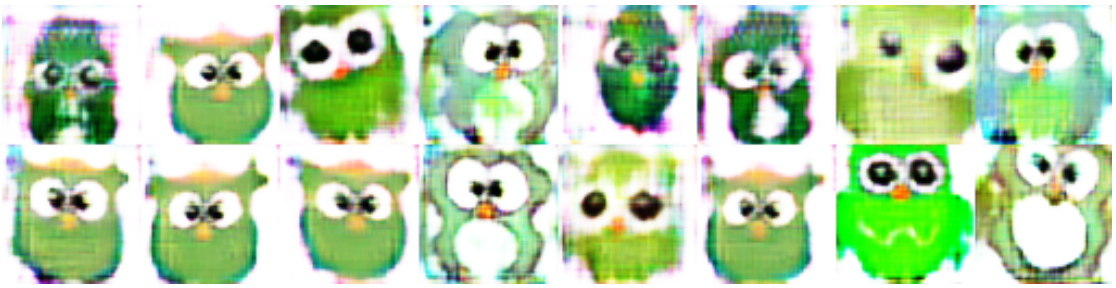


FIGURE 4.23: owl has green body brown legs blue eyes orange beaks.

FIGURE 4.24: bird has green body yellow legs yellow beak black eyes.

which we think made the huge difference in the shapes of the output images so we decided to continue in that path until we reach the state where we have enough data to train model for all types of birds as we originally hoped.

Chapter 5

Story Builder Application

In this chapter reader will find the detailed structure of our application *Story Builder*, it is divided into two main parts, the first part is concerned with the front-end, while the second part is the back-end.
This chapter is important for the sake of future maintaining and continuity of work, as core points of design and implementation is documented .
Reader can find the link to the repository of the application on GitHub by clicking here

5.1 Front-End : Android Application

This section is dedicated to highlight the main hierarchy of the front-end i.e. *Android app*, in the first section reader can find the detailed software structure of the application, in terms of main use-cases which represents the user main specifications and scenarios, sequence diagrams to illustrate individual use-cases, class diagrams to show how the application hierarchy is designed and built, and finally user can find the database model to figure out how the application local storage is working .

5.1.1 Detailed software design

5.1.1.1 Use Case Diagrams

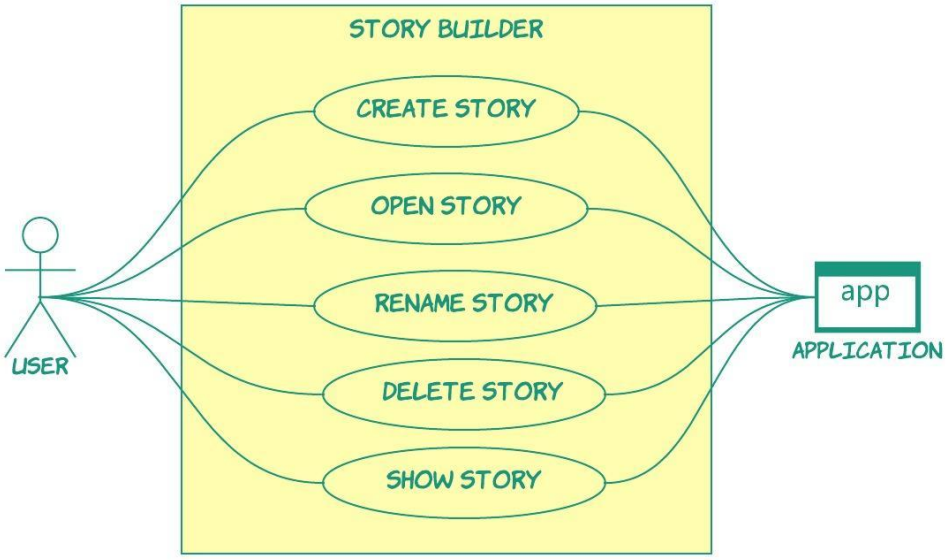


FIGURE 5.1: Use case diagram for story

- **Create Story Use Case:**
  Actors: The user (i.e the child or his parents) and the application.
  Description: The user creates a new story by the specified name.
  Data: The story name.
  Stimulus: User click the new story button.
  Comment: User is allowed to add any number of stories as he wants.

- **Open Story Use Case:**
  Actors: The user (i.e the child or his parents) and the application.
  Description: The user sees a list of his stories and clicks at any story he wants so as to open it.
  Data: The story clicked by the user.
  Stimulus: User click on the specified story he wants.
  Comment: User is allowed to open one story at a time.

- **Rename Story Use Case:**
  Actors: The user (i.e the child or his parents) and the application.
  Description: The user have the right to rename any of his stories.
  Data: The story new name.
  Stimulus: User click the rename story button and then enters a new name for

the story.

Comment: Any name is accepted.

- **Delete Story Use Case:**

  Actors: The user (i.e the child or his parents) and the application.

  Description: The user deletes any story of his stories.

  Data: The story to be deleted.

  Stimulus: User click the delete story button.

  Comment: User is allowed to remove any story as he wants.

- **Show Story Use Case:**

  Actors: The user (i.e the child or his parents) and the application.

  Description: The user can preview his story as a whole i.e see all the scenes created in order as a slide show.

  Data: The story to be shown.

  Stimulus: User click the show story button.

  Comment: User is allowed to show any of stories.

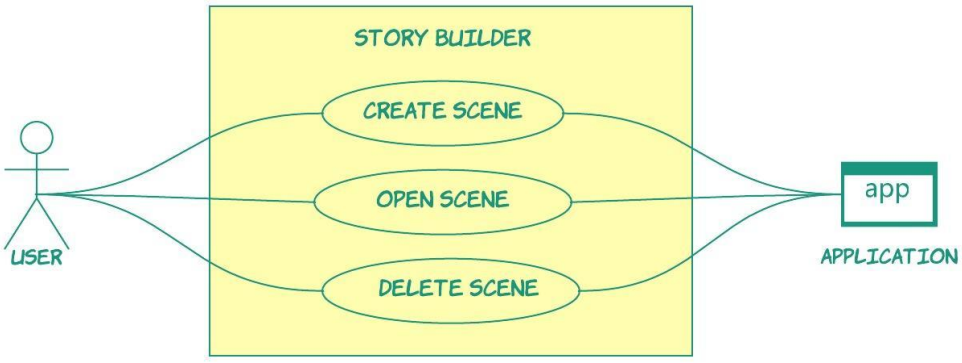


FIGURE 5.2: Use case diagram for scene

- **Create Scene Use Case:**

  Actors: The user (i.e the child or his parents) and the application.

  Description: After the user enters any of his stories, he then can add any number of scenes to the story.

  Data: The new scene to be added to the specified story.

  Stimulus: User click the new scene button.

  Comment: User is allowed to add any number of scenes as he wants.

- **Open Scene Use Case:**

  Actors: The user (i.e the child or his parents) and the application.

  Description: The user sees a list of his scenes and clicks at any story he wants so as to open it.

Data: The scene clicked by the user.

Stimulus: User click on the specified scene he wants.

Comment: User is allowed to open one scene at a time.

- **Delete Story Use Case:**

  Actors: The user (i.e the child or his parents) and the application.

  Description: The user deletes any scene of his scenes.

  Data: The scene to be deleted.

  Stimulus: User click the delete scene button.

  Comment: User is allowed to remove any scene as he wants.

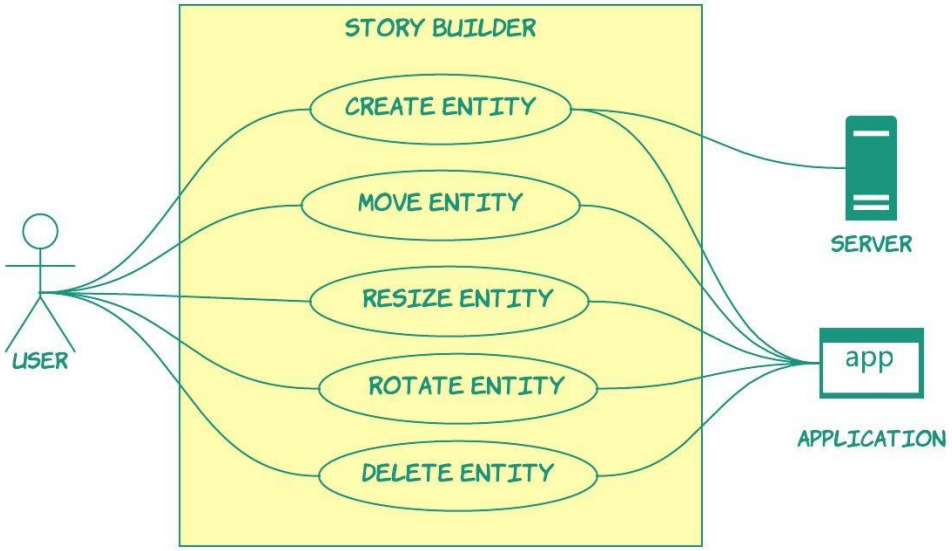


FIGURE 5.3: Use case diagram for entity

- **Create Entity Use Case:**

  Actors: The user (i.e the child or his parents) ,the application, the server (to generate the image of the entity).

  Description: After the user enters any of his scenes, he then can add any number of entities to the scene.

  Data: The description of the new entity to be added *for example: A blue bird with orange beak and black eyes.*

  Stimulus: User click the new entity button then writes or records (by his voice) the description of the new entity.

  Comment: User is allowed to add any number of entities as he wants, same description of an entity generate a new image every time.

- **Move Entity Use Case:**

  Actors: The user (i.e the child or his parents) and the application.

Description: The user is allowed to move any of his entity by dragging and dropping it.

Data: The entity to be moved.

Stimulus: User drags any entity to move it to a new place then drops it .

Comment: User is allowed to move any of his entities on the specified area of the scene.

- **Re-size Entity Use Case:**

  Actors: The user (i.e the child or his parents) and the application.

  Description: The user is allowed to re-size any of his entities i.e zoom it in or out.

  Data: The entity to be re-sized.

  Stimulus: User use his hand gestures by zooming in or out so as to re-size his entity.

  Comment: User is allowed to re-size any of his entities on the specified area of the scene.

- **Rotate Entity Use Case:**

  Actors: The user (i.e the child or his parents) and the application.

  Description: The user is allowed to rotate any of his entities.

  Data: The entity to be rotated.

  Stimulus: User click on rotate button on the entity to rotate it by any angle he wants.

  Comment: User is allowed to rotate any of his entities.

- **Delete Entity Use Case:**

  Actors: The user (i.e the child or his parents) and the application.

  Description: The user deletes any entity of his entities.

  Data: The entity to be deleted.

  Stimulus: User click the delete entity button.

  Comment: User is allowed to remove any entity as he wants.

5.1.1.2 Sequence Diagrams

This subsection is dedicated to the detailed sequence of each use case of in the application, where the interaction between the user, application modules and back-end server. The user UI transfers the user actions to the application and receive various responses from the application modules and the back-end servers .

Story Related Sequence Diagrams

- **Creating** story which takes user input parameters as story name, story cover color and etc, then the *BookShelf* activity will request a customized story object from the *StroyFactory* according to the user parameters, the latter will return story object, the *BookShelf* activity will request from the app *Engine* to save the story which will delegate the request to the *SQLiteDBHandler* module, the diagram is found in Fig 5.4

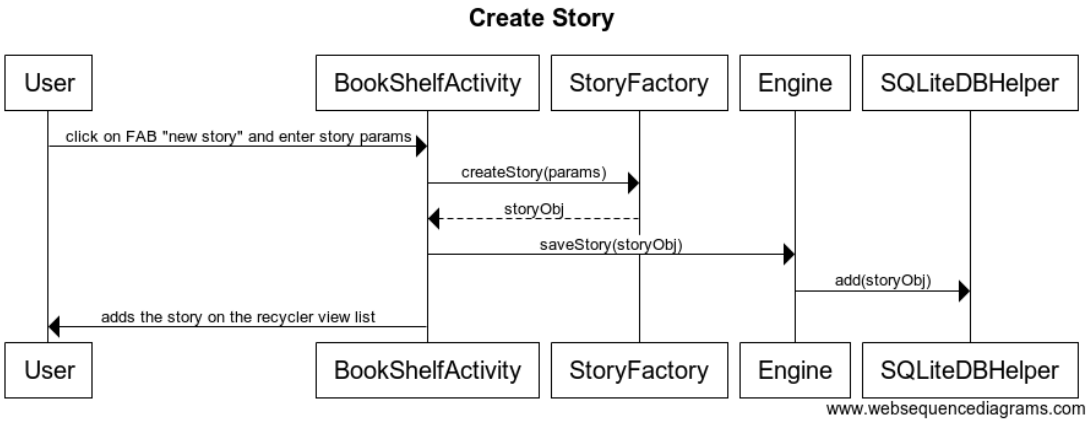


FIGURE 5.4: Sequence diagram of creating story object

- **Renaming** story sequence takes place when user press on rename option of specific story, the *BookShelf* activity will request from the app *Engine* to update the story which will delegate the request to the *SQLiteDBHandler* module, the diagram is found in Fig 5.5

- **Delete** story sequence takes place when user press on delete option of specific story, the *BookShelf* activity will request from the app *Engine* to delete the story which will delegate the request to the *SQLiteDBHandler* module, the diagram is found in Fig 5.6

- **Show** story sequence takes place when user press on play option of specific story, the *BookShelf* activity will start the activity of *PagerMainFragment* which will

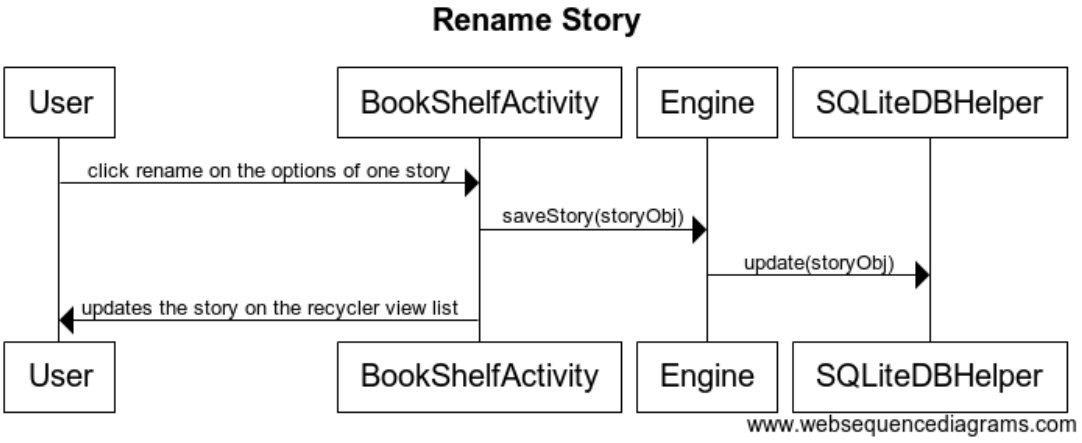


FIGURE 5.5: Sequence diagram of renaming story object

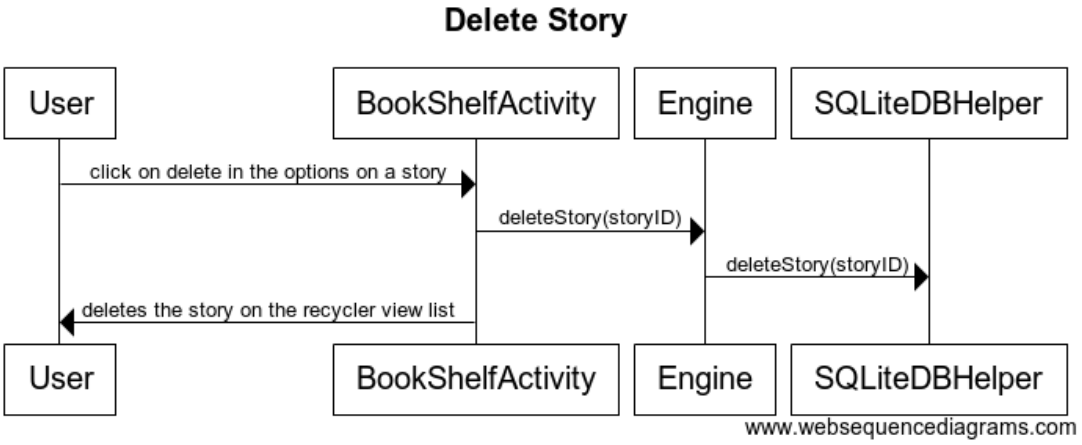


FIGURE 5.6: Sequence diagram of deleting story object

create new instance from *SwipeAdaptor* with the chosen story, *PagerMainFragment* its other job to listen to the user swipe to indicate the scene number of the current story, and pass this index to the adaptor, which in turn will return a *PageFragment* to the *PagerMainActivity* which contains the scene narration and entities as well,the diagram is found in Fig 5.7

- **Opening** story sequence takes place when user press on a specific story, the *BookShelf* activity will start the activity of *SceneEngine* which will load the old scenes of the chosen story and show them to the user by updating the recycler view list the diagram can be found in Fig 5.8

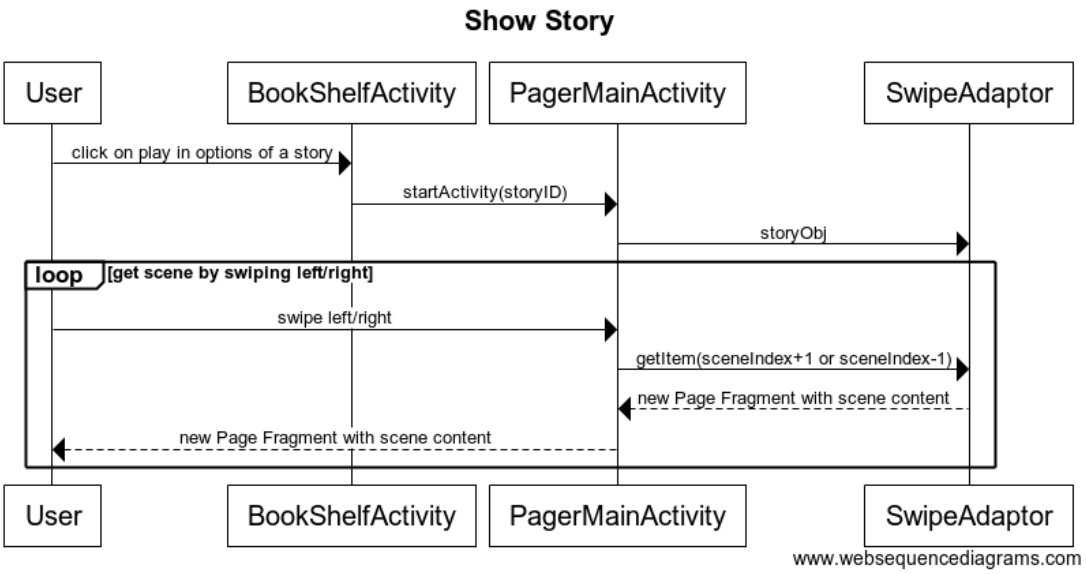


FIGURE 5.7: Sequence diagram of showing story object

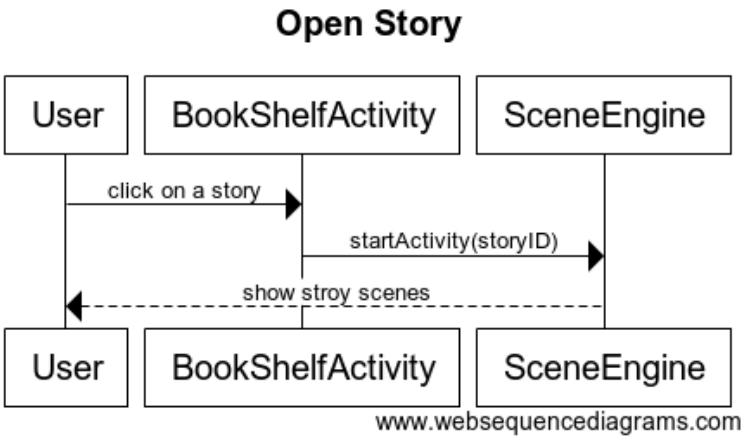


FIGURE 5.8: Sequence diagram of opening a story object

Scene Related Sequence Diagrams

- **Creating** new scene when the user press on the floating action button of creating new scene in *SceneEngine* activity of the current story, which will request a default scene object from the *SceneFactory*, the latter will return scene object, the diagram can be found in Fig 5.9

- **Opening** a scene takes place when user press on a specific scene in the current story, the *SceneEngine* activity will start the activity of *SceneCreator* which will load the old entities of the chosen scene -if any- and show them to the user, the diagram can be found in Fig 5.10

- **Delete** a scene takes place when user press on delete option of specific scene, the *SceneEngine* activity will request from the app *Engine* to delete the scene which

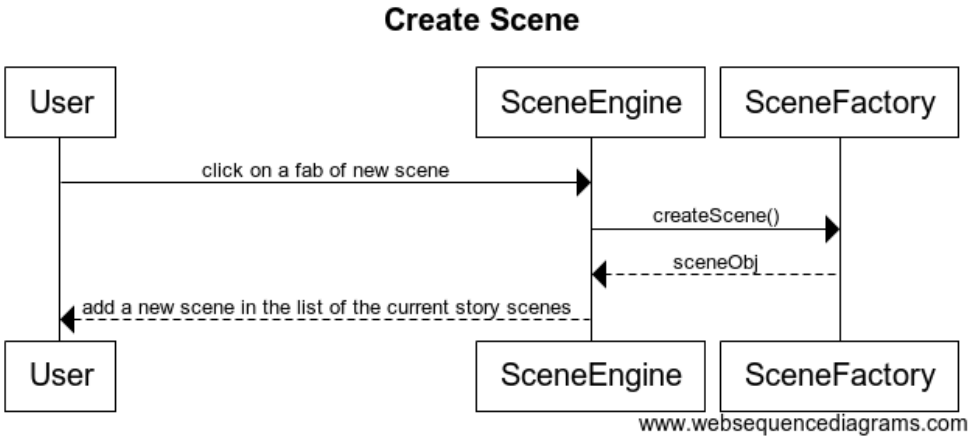


FIGURE 5.9: Sequence diagram of creating scene object

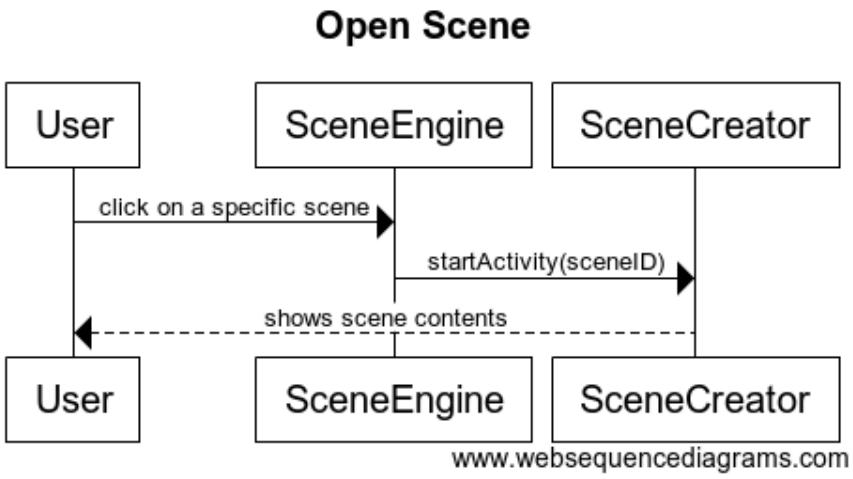


FIGURE 5.10: Sequence diagram of opening scene object

will delegate the request to the *SQLiteDBHandler* module, the diagram is found in Fig 5.11

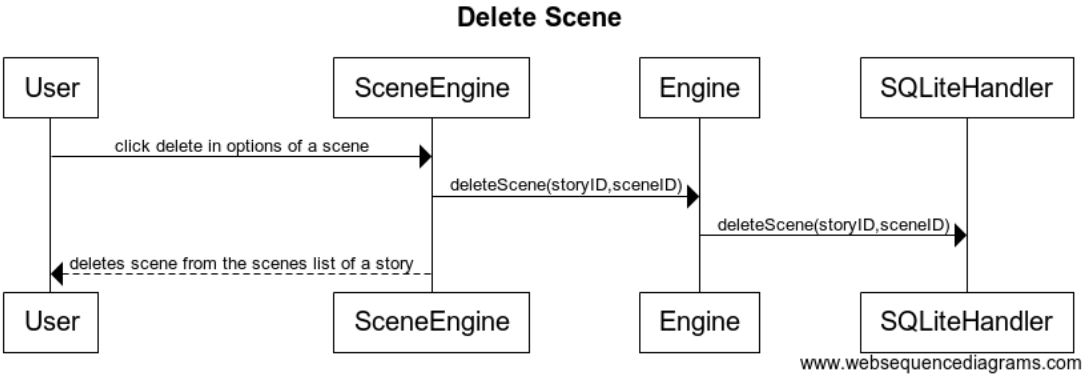


FIGURE 5.11: Sequence diagram of deleting scene object

Entity Related Sequence Diagrams

- **Creating** new entity when the user press on the floating action button of creating new entity, the user can enter the description of the entity using either one of two alternatives, text input or using voice recognition of *Google Mic*, the floating button is in *SceneCreator* activity of the current story, which will issue GET request from the *GoogleCloud* which the back-end of the application, the latter will start the *GAN Model* and make it to create the image which corresponds to the requested description, the *Model* will create the image, *GoogleCloud* instance will return bits of the created image to *SceenCreator*, which will request from *EntityFactory* to create entity object based on the bitmap returned,the diagram can be found in Fig 5.12

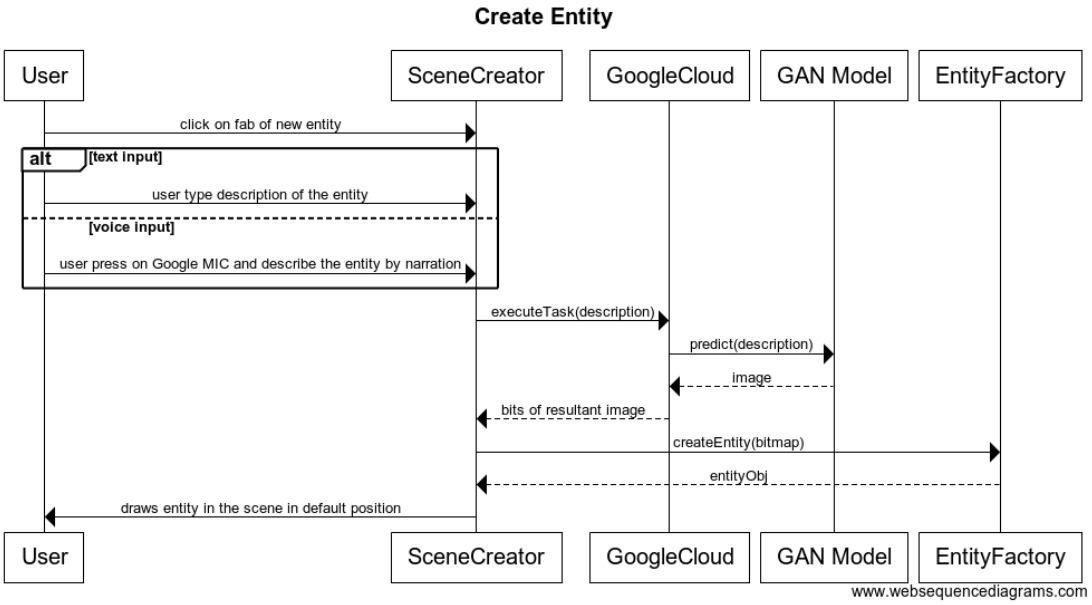


FIGURE 5.12: Sequence diagram of creating entity object

- **Moving/Resizing/Rotating** an entity takes place when user preform the appropriate gesture of a specific action on the chosen entity, the *SceneCreator* will caliprate the affected parameters in the chosen entity object, and redraw it for the user, the diagram can be found in Fig 5.13

- **Delete** an entity takes place when user press on delete option of specific entity, the *SceneCreator* activity will request from the app *Engine* to delete the entity which will delegate the request to the *SQLiteDBHandler* module, the diagram is found in Fig 5.14

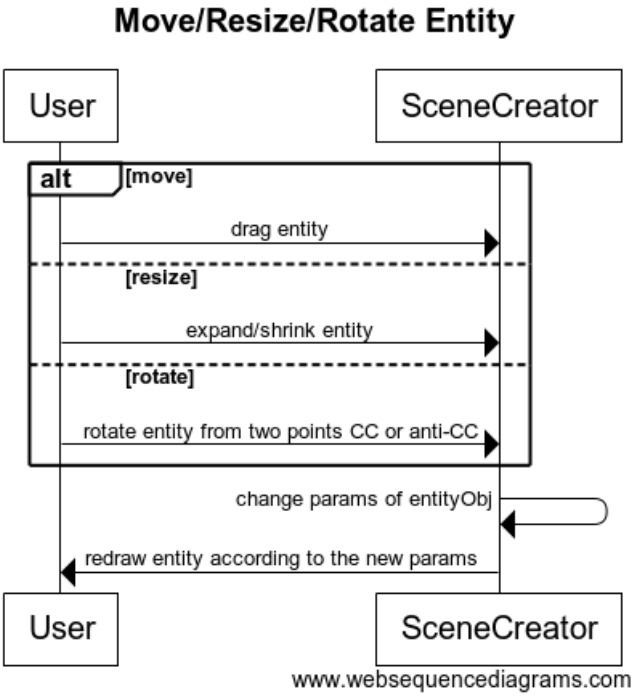


FIGURE 5.13: Sequence diagram of performing action on entity object

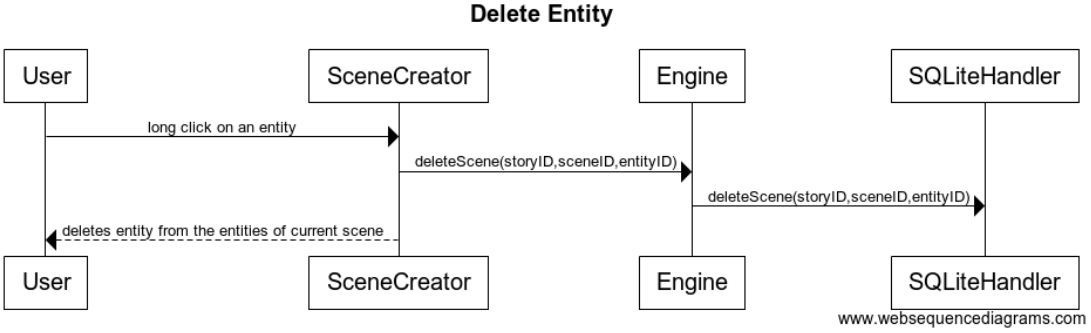


FIGURE 5.14: Sequence diagram of deleting entity object

5.1.1.3 Class Diagram

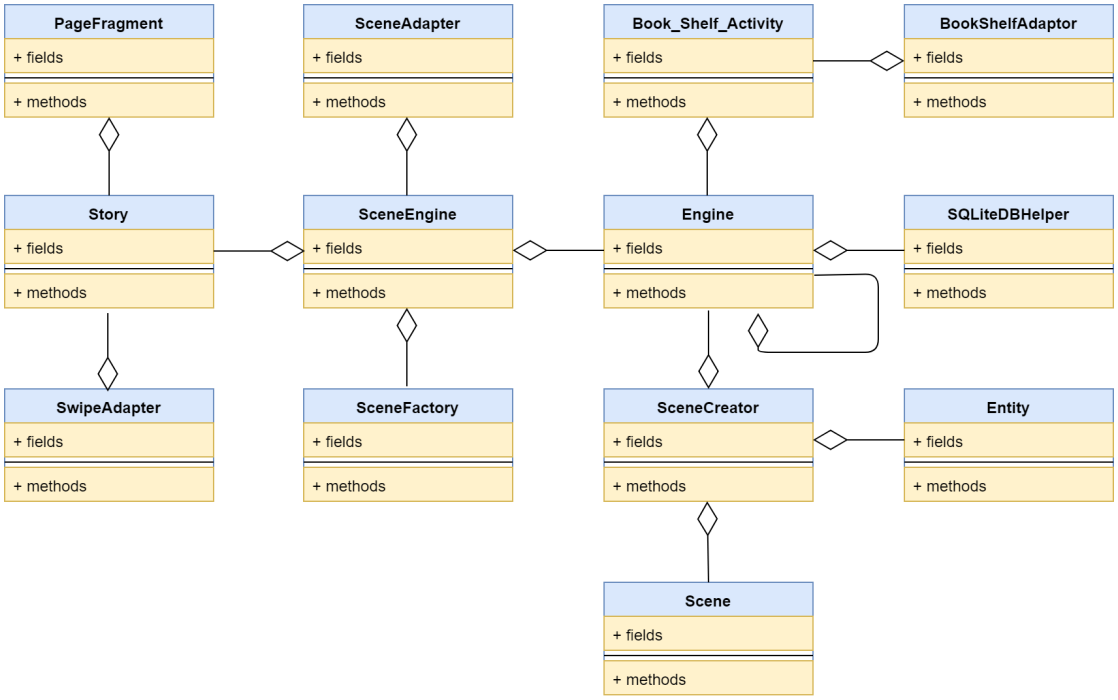


FIGURE 5.15: Class diagram for Story Builder

- **Story Class:**

  Fields:

  -storyId:Integer.

  -storyName:String.

  -cover:String.

  -coverColor:String.

  Methods:

  +getStoryName():String.

  +setStoryName(String storyName):void.

  +getCover():String.

  +setCover(String cover):void.

  +getCoverColor():String.

  +setCoverColor(String coverColor):void.

  +getScenes():List $\langle Scene \rangle$.

  +setScenes(List $\langle Scene \rangle$ scenes):void.

- **Scene Class:**

  Fields:

  -narration:String.

  -id:Integer.

-storyId:Integer.

-cover:String.

-entities:List $\langle Entity \rangle$.

Methods:

+getNarration():String.

+setNarration(String narration):void.

+getId():Integer.

+setId(Integer id):void.

+getStoryId():Integer.

+setStoryId(Integer storyId):void.

+getCover():Bitmap.

+setCover(Bitmap cover):void

+getEntities():List $\langle Entity \rangle$.

+setEntities(List $\langle Entity \rangle$ entities):void.

- **Entity Class:**

Fields:

-id:Integer.

-classification:String.

-name:String.

-image:Bitmap.

-positionX:float.

-positionY:float.

-rotationAngle:float.

-scaleX:float

-scaleY:float.

Methods:

+getId():Integer.

+setId(Integer id):void.

+getName():String.

+setName(String name):void.

+getImage():Bitmap.

+setimage(Bitmap image):void.

+getPositionX():float.

+setPositionX(float positionX):void.

+getPositionY():float.

+setPositionY(float positionY):void.

+getRotationAngle():float.

+setRotationAngle(float rotationAngle):void.

+getScaleX():float.

+setScaleX(float scaleX):void.

+getScaleY():float.

+setScaleY(float scaleY):void.

- **BookShelfAdaptor Class:**

  Fields:

  -storyList:List $\langle Story \rangle$.

  -shelfInstance:BookShelfActivity.

  Methods:

  +onCreateViewHolder(ViewGroup parent, int i):StroyCardViewHolder.

  +onBindViewHolder(final StroyCardViewHolder holder, final int index):void.

  + getItemCount():int.

  +onAttachedToRecyclerView(RecyclerView recyclerView):void.

- **SceneAdapter Class:**

  Fields:

  -scenesList:List $\langle Scene \rangle$.

  -sceneEngineInstance:SceneEngine.

  Methods:

  +onCreateViewHolder(ViewGroup parent, int viewType):MySceneHolder.

  +onBindViewHolder(final MySceneHolder holder, final int position):void.

  + getItemCount():int.

- **SwipeAdapter Class:**

  Fields:

  -story:Story.

  Methods:

  +getItem(int position):Fragment.

  +getCount():int.

- **Engine Class:**

  Fields:

  -instance:Engine.

  -db:SQLiteDBHelper.

  Methods:

  +loadStroies():List $\langle Story \rangle$

  +saveStroies(Story story):void.

- **SQLiteDBHelper Class:**

  Fields:

  -String SQL_CREATE_STORY.

  -String SQL_CREATE_SCENE.

  -SQL_CREATE_ENTITY.

  Methods:

  +addStory(Story story):boolean.

  +getLastStoryId():int.

  +getLastSceneId():int.

  +getLastEntityId():int.

  +getAllStories():ArrayList(Story).

  +getAllScenes(int storyId)ArrayList(Scene)

  +getAllEntities(int sceneId):ArrayList(Entity)

  +getStory(Integer storyId):Story.

  +getStoryByName(String name):boolean.

  +updateStory(Story story).

  +deleteStory(int storyId):void.

  +deleteScene(int sceneId):void.

  +deleteEntity(int entityId):void.

- **Book_Shelf_Activity Class:**

  Fields:

  -RecyclerView bookShelfRV.

  +List(Story) stories.

  -Engine engine.

  Methods:

  -loadStories():void.

  -setupEngine():void.

  -setBookShelfContent(List(Story)):void.

  -setupBookShelf():void.

- **PageFragment Class:**

  Fields:

  -TextView textView.

  -ViewGroup displayView.

  -Story story.

  -int sceneIndex.

Methods:

-loadScene():void.

-showSceneEntities():void.

-showNarrationText():void.

-showEntity(Entity entity):void.

- **SceneCreator Class:**

Fields:

-List(Entity) entities.

-Scene scene.

-Engine engine.

-Map(ImageView,Entity) imageEntityMap.

Methods:

-createEntity(String descreption):void.

-searchGoogle(String descreption):Bitmap.

-parseResponse(String JsonResponse):Bitmap.

-loadEntity():void.

-showEntity(Entity entity):void.

- **SceneEngine Class:**

Fields:

-RecyclerView recyclerView.

-int story_index.

-Engine engine.

-SceneFactory sceneFactory.

-Story story.

Methods:

-getNewScene():Scene.

-setRecyclerView():void.

-updateScenesListView():void.

- **MainActivity Class:**

Fields:

+Context appContext.

+AppCompatActivity appCompatActivity.

Methods:

-onCreate(Bundle savedInstanceState):void.

-gotoOtherActivity(Class someClass):void.

5.1.1.4 Database model

This subsection is dedicated to the database schema.The database consists of three tables. One to keep the information of the *story* , one to keep the *scenes* associated with each story , and the other for keeping the *entities* in each scene with their images and description.The following 5.16 shows the database schema in details.

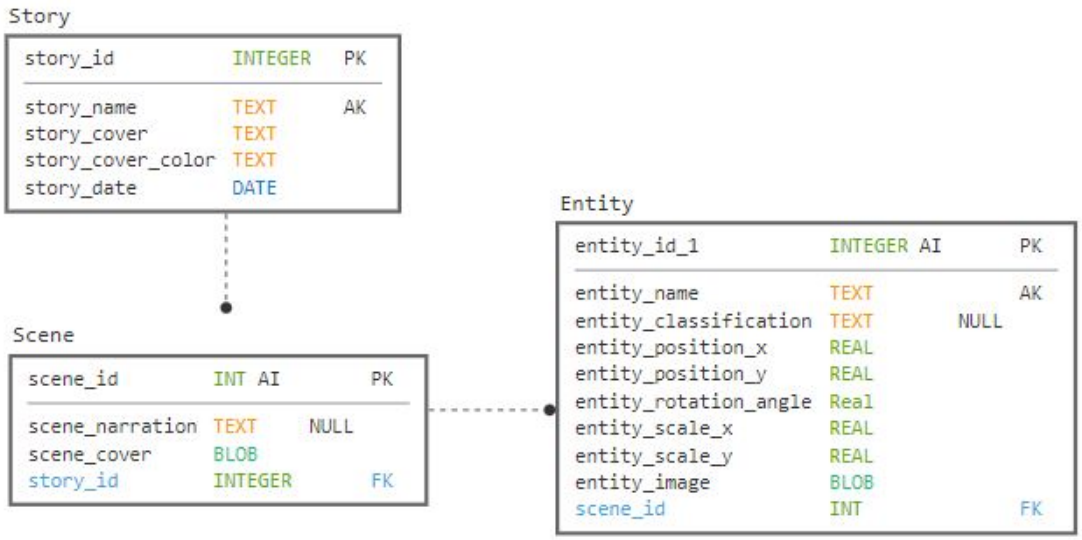


FIGURE 5.16: Entity relationship diagram

Here comes an explanation for each attribute in the tables :

Story Model

Attribute	Description
story_id	represents the primary key for the story table.
story_name	explain the name of the created story which specified by the user and it is unique attribute in the database.
story_cover	relates to the story cover image.
story_cover_color	relates to the different colors of the story cover.
story_date	indicates the time which the story created at.

TABLE 5.1: Table story model attributes

Scene Model

Attribute	Description
scene_id	represents the primary key for the scene table.
scene_cover	the thumbnail of the scene cover.
scene_narration	description of the scene given by the user.
story_id	a foreign key to the story table.

TABLE 5.2: Table of scene model attributes

Entity Model

Attribute	Description
entity_name	represents the name of each entity created in the scene.
entity_classification	represents the type of the entity _for future work_ we can have different types of animals or even different things.
entity_position_x	represents the X position of the entity image on the screen.
entity_position_y	represents the Y position of the entity image on the screen.
entity_rotation_angle	represents the rotation angle of the entity image on the screen.
entity_scale_x	represents the X scale of the entity image on the screen.
entity_scale_y	represents the Y scale of the entity image on the screen.
entity_image	the image associated with this entity.
scene_id	a foreign key to the scene table.

TABLE 5.3: Table of entity model attributes

5.1.2 User interface and main scenarios

This subsection is dedicated to provide a set of images which represents the manual for the application usage, the first part of this section contains general screen shots to show the realization of the functional features of this application, while the second part is providing one complete scenario

5.1.2.1 Main user interfaces

Application entry UIs: When the user clicks on the app icon as in Fig 5.17, the first interface will be the *Splash* screen as in Fig 5.18 which can highlight some information as the sponsor name, version of app, ...etc

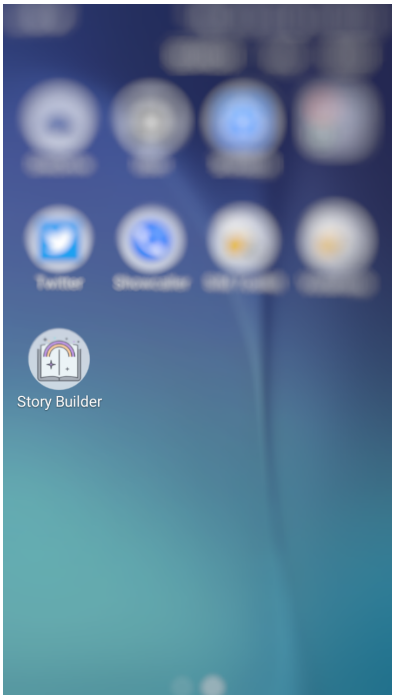


FIGURE 5.17: Screen shot of the external app icon

FIGURE 5.18: Screen shot of the splash screen

Story creation UIs: If the user opens the app for the first time, he/she will find an empty bookshelf as in Fig 5.19, using *fab* button the user can create their own stories as in Fig. 5.20

Story Options each story has *options* button on the bottom right corner on the story view, this option button provide the functionality of renaming, displaying the story or even deleting it as in Fig 5.42

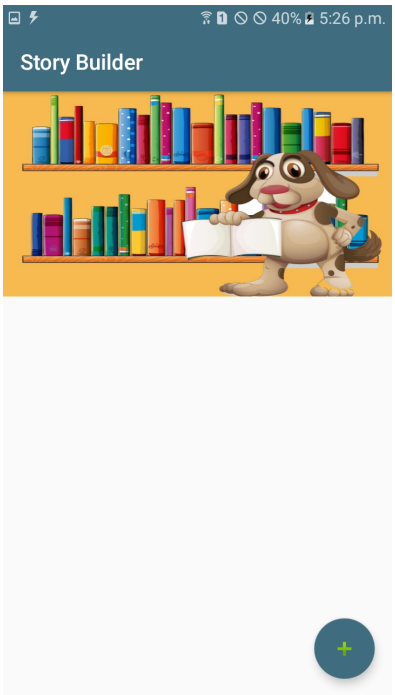


FIGURE 5.19: Screen shot of empty bookshelf

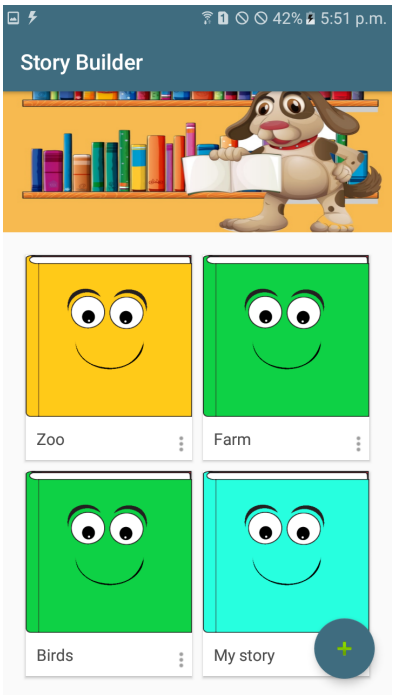


FIGURE 5.20: Screen shot of created stories on the bookshelf

Scenes creation UIs: When the user opens a newly created story, the scenes list will be empty as in Fig 5.21, only by the means of the *FAB* button, user can create scenes in the story as he/she wishes, for example 3 scenes as in Fig 5.22.

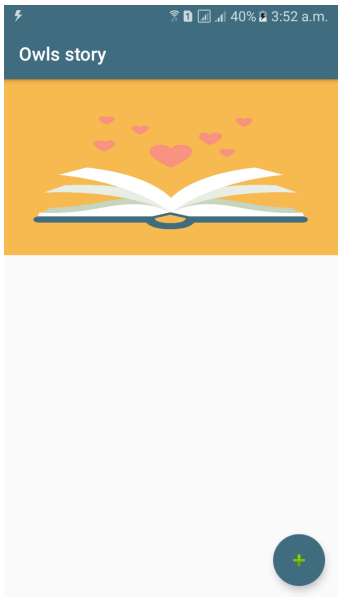


FIGURE 5.21: Screen shot of story has no scenes yet

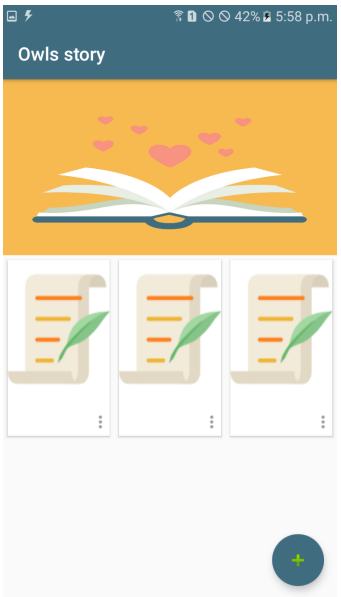


FIGURE 5.22: Screen shot of story has 3 new scenes

Scenes Thumbnails: When a user opens a scene and creates entities in it as in Fig 5.23, when the user press back button, he/she will find that the default image of the scene cover has changed to be descriptive thumbnail to ease the scenes identification by user without opening them as in Fig 5.24.

FIGURE 5.23: Screen shot of usual story scene to be thumbnail

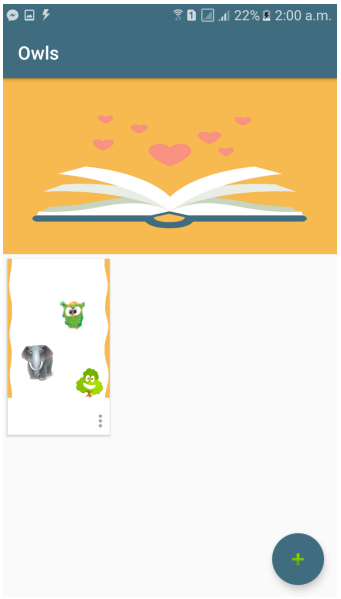


FIGURE 5.24: Screen shot of scene with thumbnail

User also can delete scene by pressing scene options as in Fig 5.25

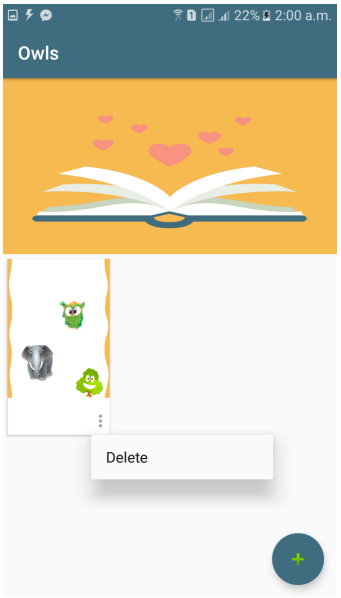


FIGURE 5.25: Screen shot of scene delete option

Create Entity UIs: When the user press on the *add* button inside any scene, dialogue pop-up appears to the user, the user can type the description of the entity as if Fig 5.26, the server will respond with the desired entity 5.27

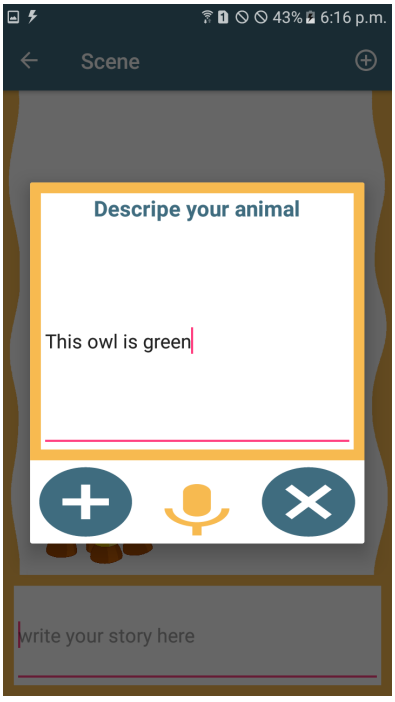


FIGURE 5.26: Screen shot of entity description

FIGURE 5.27: Screen shot of the entity after creation

Create Entity using Google Mic UIs: User can describe entity using Google Mic which is embedded in dialogue pop-up instead of typing, as in figures 5.28 and 5.29, where the user has said *Owl* and it has been correctly recognized by the mic

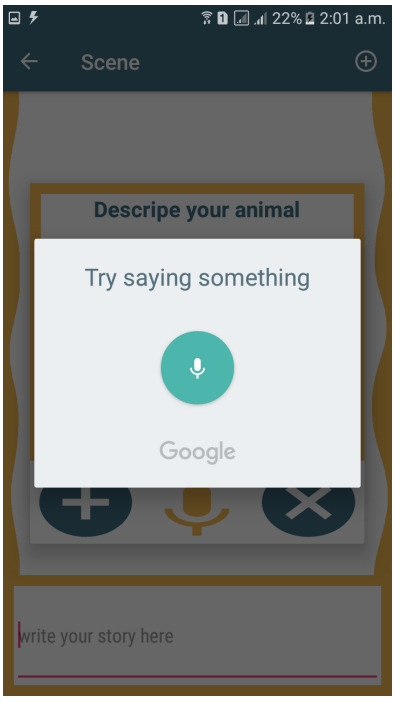


FIGURE 5.28: Screen shot of entity description by Mic

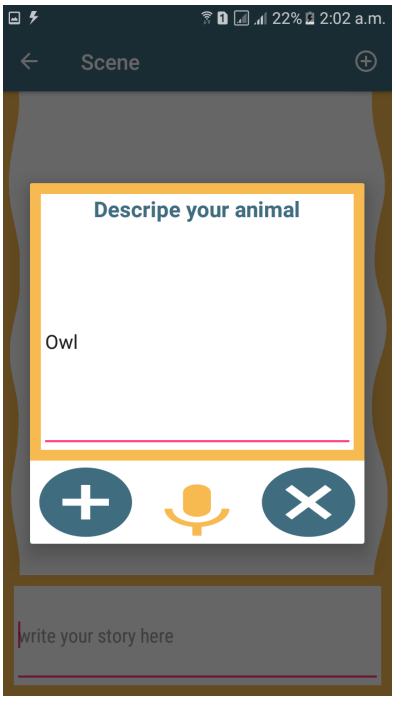


FIGURE 5.29: Screen shot of output of entity description by Mic

Entity can be dragged and dropped: Entities can me dragged and dropped freely by the user to provide flexibility in the scene creation as in figure 5.30 and 5.31

FIGURE 5.30: Screen shot of entity before drag and drop

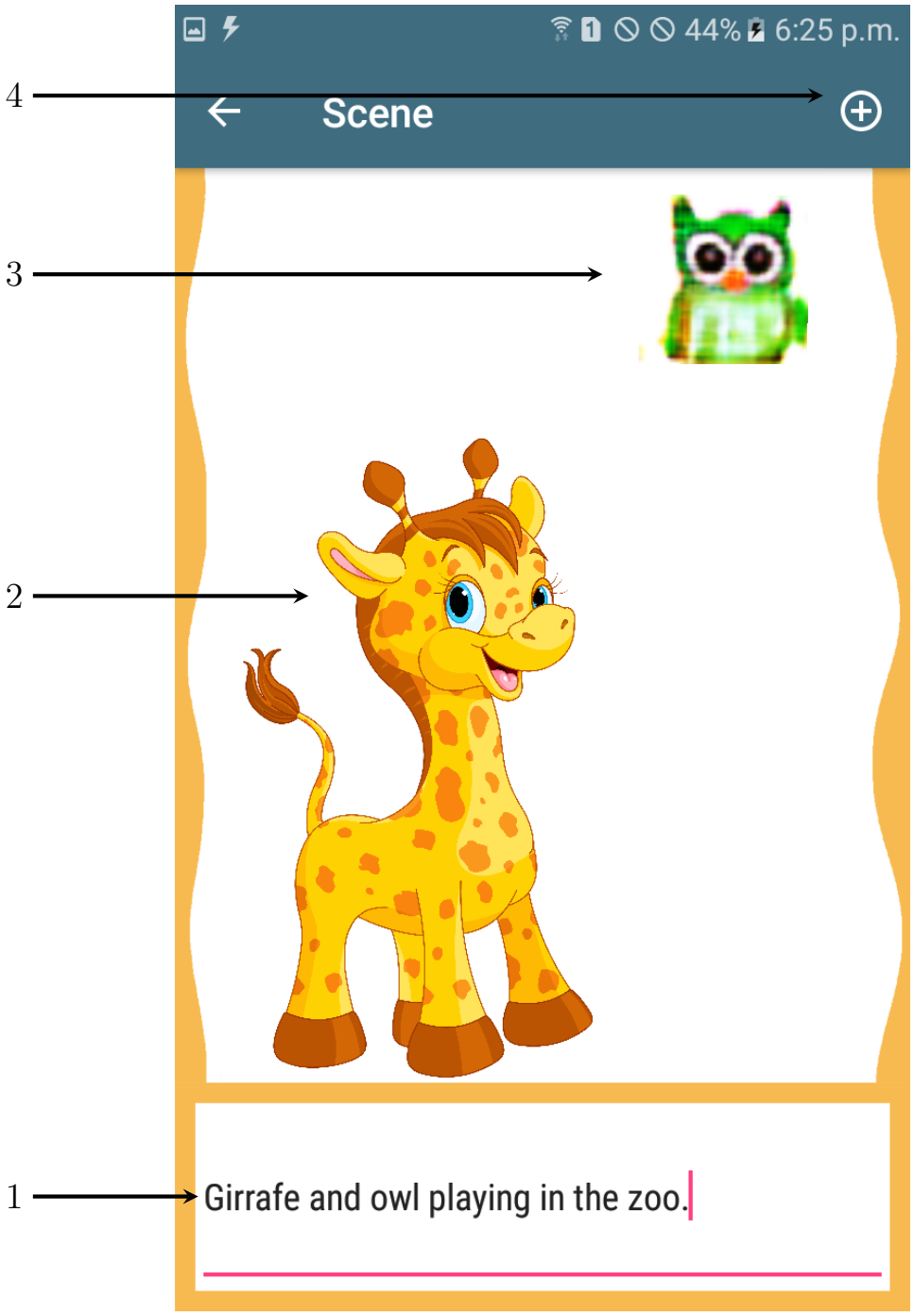
FIGURE 5.31: Screen shot of entity after drag and drop

Entity can be Resized and Rotated: to increase flexibility in story creation more and more, user can resize entity as in Fig 5.32 and rotate it in any direction as in Fig 5.33

FIGURE 5.32: Screen shot of entity after resizing

FIGURE 5.33: Screen shot of entity after rotation

Entity can be requested from GAN or Google: since our predicting GAN model -in this development version of app- is not covering all animals the child might think of, we used a temporary tool which is *Google Images API* to get images for animals other than owls, since our GAN model can predict owls, here is example in the next figure where the user wanted to create a scene in which a tall *Giraffe* talks to a green *Owl* while playing in the zoo, the owl will be requested from our server, while the *Giraffe* will be requested from *Google Images API*, the descriptions of each animal is mentioned in table 5.4



Number	Refers to
1	Narration field where user can write the narration of the story there
2	Giraffe image requested from Google Images API as our model does not generate animals other than owls in this version of the application
3	Owl image generated by our GAN model
4	Add new entity button

TABLE 5.4: Illustration of scene components and the source of each entity

5.1.2.2 Complete Scenario

1. The user open the application the list of stories that created before appear to him.
2. The user will click on plus icon to create a new story, pop-up appear, the child enters the story name *Owls story* then after this the new book added to the list of books in the opening screen figures 5.34, 5.35 show this actions.
3. The child opens the book and now no scenes in it. By click on plus icon the child can create a new scene. The list of scenes appears now to the child figures 5.37 , 5.36 show this actions.
4. the child now can add an object by click on plus icon and write *blue owl* the blue owl is created. Here there are two items sun and tree, we create them using *Google Custom Serach API* as shown in paragraph 5.1.2.1 figures 5.38 , 5.39 show this actions.
5. After this the child put the owl on tree under sun and write a narration below the image *a blue owl over a tree* figure 5.40 show this.
6. The child then can press back to see it as thumbnail figure 5.41 shows this.
7. Then press options and choose play figure 5.42 shows this.
8. The display mode is play now figure 5.43 shows this.

FIGURE 5.34: create a story

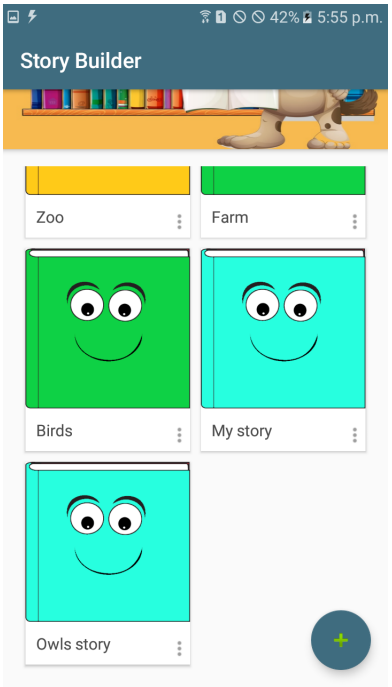


FIGURE 5.35: list of books

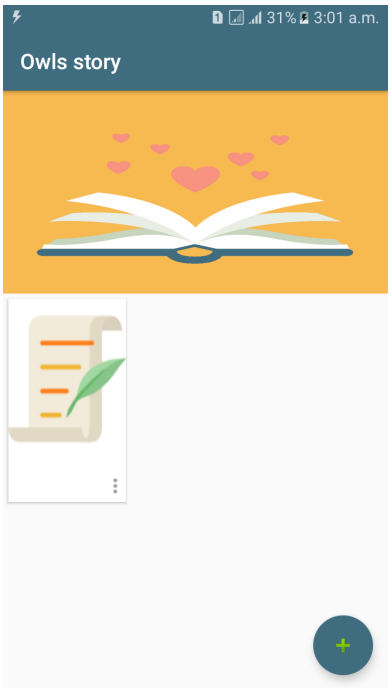


FIGURE 5.36: list of scenes

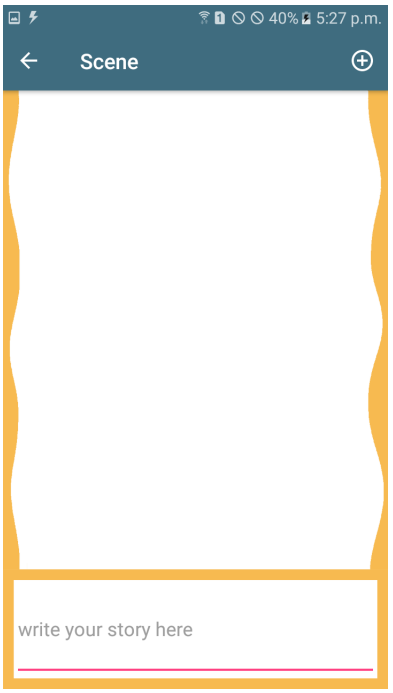


FIGURE 5.37: create a scene

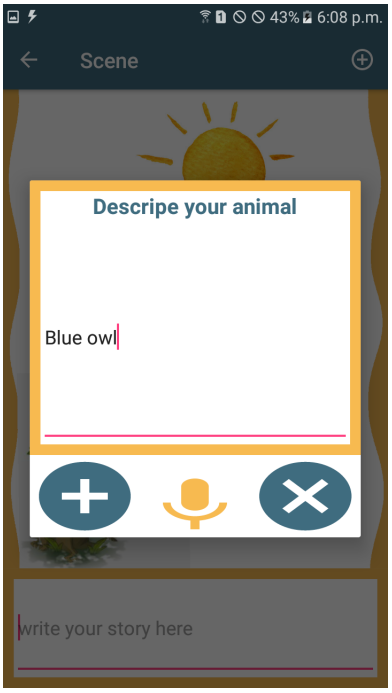


FIGURE 5.38: create a new entity

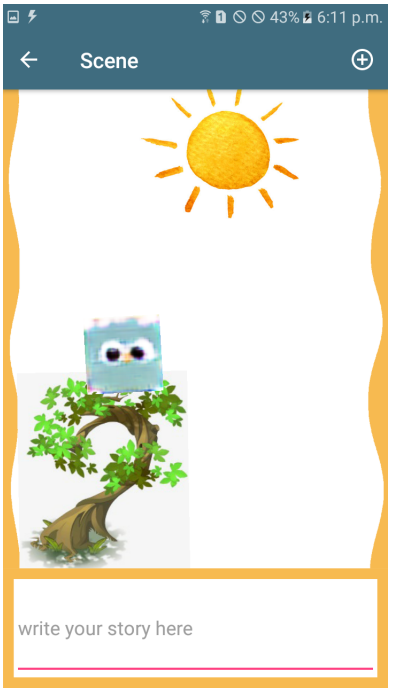


FIGURE 5.39: objects after request

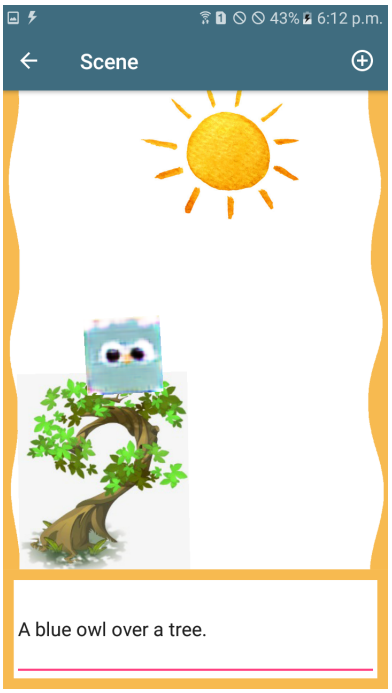


FIGURE 5.40: write a narration

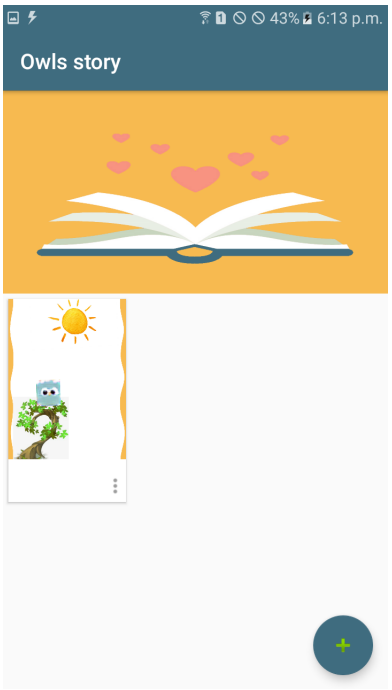


FIGURE 5.41: thumbnail

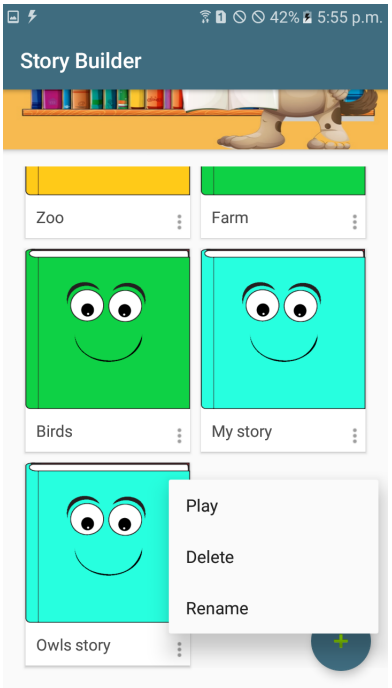


FIGURE 5.42: screen options

FIGURE 5.43: display mode

5.1.3 Error Handling

Main feature of this application is it based mainly on the input from the user, as story name, number of scenes in this story, entities number and entities description which is passed to the GAN model latter. So that was a priority to protect the application and filter out the input as possible, accordingly, this section is dedicated for the error handling all over the application.

5.1.3.1 Story name faults

Story name repeat error

According to our database schema design, the *story name* is a *unique* attribute, so the application will restrict the input of the story name from the user side to ensure the safety of the database, in other words, if user type a story name which was given to other story previously, the application will reject it as shown in Fig 5.44.

Empty story name error

According to our database schema design, the *story name* is a *not null* attribute with no default value, so the application will restrict the input of the story name from the user side to ensure the safety of the database, in other words, if user type a story name as empty string, the application will reject it as shown in Fig 5.45.

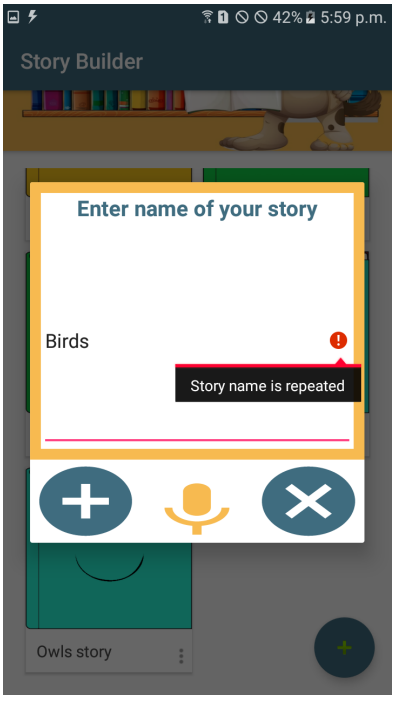


FIGURE 5.44: Story name repeated error handling

FIGURE 5.45: Empty story name error handling

5.1.3.2 Server Requests

This section illustrates the precautions taken to ensure the safety of the application and protect it from crashes when the user request images from server, after the user has just written the description of the desired entity, and has committed the request. The application check whether if the user's device has is good internet connection or not, in this context *good internet connection* means stable cellular data or stable WIFI connection.

If the connection is good the application will permit the request and wait for the response from the server, else it will block the request and shows the user a toast message to make the error understood by the user, as shown in Fig 5.46, after the user requested the elephant image, the connection become bad, so user cant request any more images until stable internet connection is provided to the application.

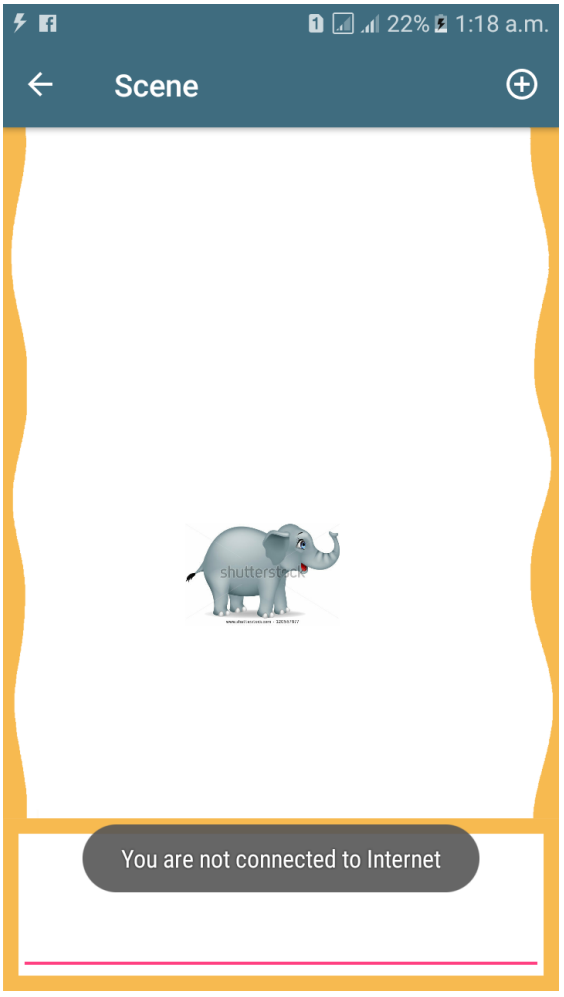


FIGURE 5.46: Bad internet connection during request handling

5.2 Back-End : Google Cloud Platform

5.2.1 Instance Specification

General Specification

Attribute	Specification
OS	ubuntu
Machine type	custom (8 vCPUs, 14 GB memory)
Zone	us-east1-b
Creation time	Feb 5, 2018, 12:41:01 AM

Table 5.5: instance specification in google cloud

Boot disk and local disks

Attribute	Specification
Size (GB)	80
Type	Standard persistent disk
Encryption	Google managed
Mode	Boot, read/write

Table 5.6: Boot disk and local disks specification in google cloud

Network Specification

Attribute	Specification
Network tags	http-server, https-server
Service account	823869517552-compute@developer.gserviceaccount.com
Primary internal IP	10.142.0.2
External IP	Ephemeral

Table 5.7: Network specification in google cloud

5.2.2 GAN Installation

This is an experimental tensorflow implementation of synthesizing images[13]. The images are synthesized using the GAN-CLS Algorithm from the paper Generative Adversarial Text-to-Image Synthesis. This implementation is built on top of the excellent DCGAN in Tensorflow.

5.2.2.1 Requirement Libraries

- TensorFlow 1.0+
- TensorLayer 1.4+
- NLTK : for tokenizer

5.2.2.2 Training Process

by using the code introduced by ([13]scott 2016) and put it in the instance we used our data as described in 4.3 then we trained the Generator and Discriminator by our data .

5.2.2.3 Testing Process

After run the code for a long time to produce a good accuracy we create a prediction script to predict an image from caption .

5.2.2.4 Our Results

We entered 6 captions :

1. this owl is green with black eyes and yellow beak
2. this owl is yellow with black eyes
3. this owl is purple with yellow eyes
4. this owl is purple with yellow beak and black eyes
5. this owl is green with black eyes
6. this owl is orange and green with yellow beak and black eyes

table 5.8 show our results , by entering this captions .

Generated Image	Description
	this owl is green with black eyes and yellow beak
	this owl is yellow with black eyes
	this owl is purple with yellow eyes
	this owl is purple with yellow beak and black eyes
	this owl is green with black eyes
	this owl is orange and green with yellow beak and black eyes

TABLE 5.8: our results of training data

As we saw in the results the model understand some color and not understand the others for example the yellow eyes not appear in the query number 3 because non of training owls have a yellow birds but the black eyes appear a lot because the training owls have all a black eyes.

5.2.3 Back-end Description

By using Flask framework we create a server side to handle the requests from the application.

We have 3 functions in the back-end server side :

- first function to load the model and the checkpoints that will be used to generate the image from the caption.
- second function to generate the image from the given caption

- the last function to handle the request and take the query (caption) from the front-end (application)

Here the pseudo code of the server side algorithm 3 describe the operations that take place :

Algorithm 3 Server Side pseudo code

```
inputCaption ← caption_from_request
if first_request then
    model ← model.load
end if
generated_image ← model.generate(inputCaption)
output_image ← generated_image
```

5.2.4 Front-end Request Handling

When receiving the request from the application (front-end) **the sequence of operation that take place :**

1. Before the first request to the server the server load the model and checkpoints that will be used to generate the image
2. then the server take the caption that included in the request and run the model to generate image from that caption
3. after generate the image the server return the image to the front-end (application)

sequence diagram for the request handler see figure 5.47 .this figure show that the when the user enter the caption the front-end take this caption and pass it to the back-end that will run the model and produce the image then return the image back to the front-end .

5.2.5 Generated Image enhancement

After creating the image from the GAN model the images is created with white background so our enhancement to this image to remove this background by writing python script to remove this background and return the image after this enhancement to the front-end.

Here the pseudo code of this script 4 describe the operations that take place :

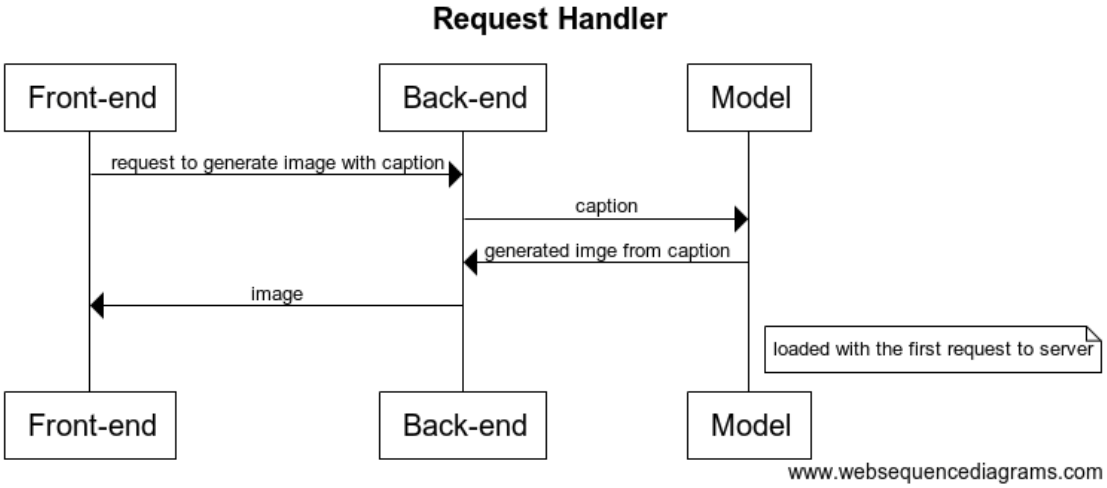


FIGURE 5.47: Sequence Diagram to handle request

Algorithm 4 Python script to remove background

```
inputImage ← get_generated_image
for index in range (0, inputImage.length) do
    if pixel_is_white then
        enhanced_image ← set_pixel_to_transparent
    else
        enhanced_image ← inputImage[index]
    end if
end for
output_image ← enhanced_image
```

Chapter 6

Future Work

Having this idea as a graduation project makes time a limit to introduce all the features we want the application to have, For this it is a must to have a plan for what might be our future work.
The upcoming sections will show for the reader our planed future work for each side of the project.

6.1 Application side

- Create an account for each user, give him the ability to sign in and sign out and keep his stories.
- User can be able to identify his favourite characters from the generated ones, name them, use them in different stories and share his favourite characters with other users.
- Users can share their stories with each other and ask their friends to share editing stories with them by this we form a community for children to share their work in it.
- Users can show images with Augmented Reality feature this make the story more realistic and add a sense of fun to the application.
- Users can record their narration for the story and hear it while browsing it .

6.2 GAN model side

- Increase the accuracy for the generated images.

- Introduce more images that GAN can generate.
- Handle all images to be with a transparent background.
- Introduce more imaginative images to user using his description for example a dog with wings.

6.3 Server Side

- Use framework that is ready for production as Flask is only concerned with Development .
- Increase the scalability of the application and handling many requests at the same time.

Appendix A

Appendix

A.1 Steps to train a GAN

1. Define the problem. Do you want to generate fake images or fake text. Here you should completely define the problem and collect data for it.
2. Define architecture of GAN. Define how your GAN should look like. Should both your generator and discriminator be multi layer perceptrons, or convolutional neural networks? This step will depend on what problem you are trying to solve.
3. Train Discriminator on real data for n epochs. Get the data you want to generate fake on and train the discriminator to correctly predict them as real. Here value n can be any natural number between 1 and infinity.
4. Generate fake inputs for generator and train discriminator on fake data. Get generated data and let the discriminator correctly predict them as fake.
5. Train generator with the output of discriminator. Now when the discriminator is trained, you can get its predictions and use it as an objective for training the generator. Train the generator to fool the discriminator.
6. Repeat step 3 to step 5 for a few epochs.
7. Check if the fake data manually if it seems legit. If it seems appropriate, stop training, else go to step 3. This is a bit of a manual task, as hand evaluating the data is the best way to check the fakeness. When this step is over, you can evaluate whether the GAN is performing well enough.

Now just take a breath and look at what kind of implications this technique could have. If hypothetically you had a fully functional generator, you can duplicate almost

anything. To give you examples, you can generate fake news; create books and novels with unimaginable stories; on call support and much more. You can have artificial intelligence as close to reality; a true artificial intelligence! Thats the dream!!

A.2 Using Generator and Discriminator in image synthesis

Generative adversarial networks (GANs) consist of a generator G and a discriminator D like minimax game The discriminator tries to distinguish real training data from synthetic images, and the generator tries to fool the discriminator.

$$\min_{G}\max_{D} V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[logD(x)] + \mathbb{E}_{x\sim p_z(z)}[log(1-D(G(z)))] \tag{A.1}$$

[10]Goodfellow et al. (2014) prove that this minimax game has a global optimium precisely when pg = pdata, and that under mild conditions (e.g. G and D have enough capacity) pg converges to pdata. In practice, in the start of training samples from D are extremely poor and rejected by D with high confidence. It has been found to work better in practice for the generator to maximize log(D(G(z))) instead of minimizing $log(1-D(G(z)))$

A.2.1 Deep symmetric structured joint embedding

Here we follow the approach of [8]Reed et al to obtain a discriminative victor by using a deep convolutional and recurrent text encoder . the text classifier induced by the learned correspondence function f_t is trained by optimizing the following structured loss:

$$\frac{1}{N}\sum_{n=1}^{N}\Delta(y_n, f_v(v_n)) + \Delta(y_n, f_t(t_n)) \tag{A.2}$$

where $\{(v_n, t_n, y_n) : n = 1, ..., N\}$ is the training data set , Δ is the 0-1 loss v_n are the images, t_n are the corresponding text descriptions, and y_n are the class labels Classifiers f_v and f_t are parametrized in classifier parameters figure:

$$f_v(v) = \arg\max_{y\in\gamma} \mathbb{E}_{t\sim\tau(y)}[\phi(v)^T\varphi(t)] \tag{A.3}$$

$$f_t(t) = \arg\max_{y\in\gamma} \mathbb{E}_{v\sim\nu(y)}[\phi(v)^T\varphi(t)] \tag{A.4}$$

A.3 Method to generate images

A.3.1 Network architecture

In the generator G , first we sample from the noise prior and we encode the text query t using text encoder . the description embedding is first compressed using a fully connected layer to small dimension followed by leaky-ReLU and then concatenate to the noise vector, following this inference proceeds as in a normal deconvolutional network , we feed forward it through the generator G , a synthetic image is generated Image generation corresponds to feed-forward inference in the generator G conditioned on query text and a noise sample.

In the discriminator D, we perform several layers of stride-2 convolution with spatial batch normalization ([15]Ioffe & Szegedy, 2015) followed by leaky ReLU. We again reduce the dimensionality of the description embedding in a (separate) fully-connected layer followed by rectification. Batch normalization is performed on all convolutional layers.

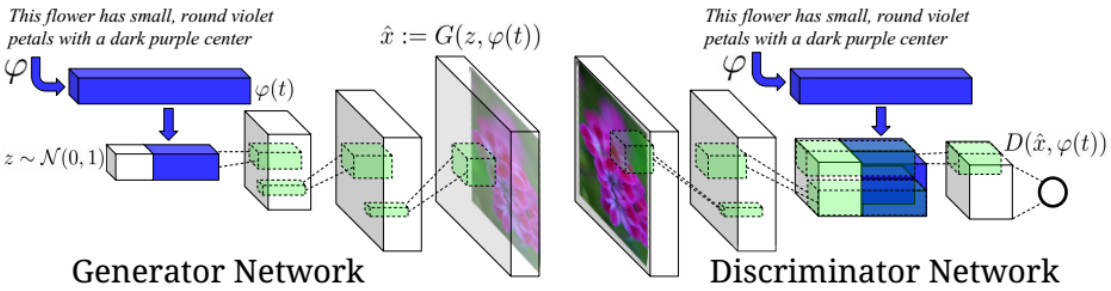


FIGURE A.1: GAN Architecture

A.3.2 Matching-aware discriminator (GAN-CLS)

The most straightforward way to train a conditional GAN is to view (text, image) pairs as joint observations and train the discriminator to judge pairs as real or fake. This type of conditioning is naive in the sense that the discriminator has no explicit notion of whether real training images match the text embedding context

However, as discussed also by ([16]Gauthier, 2015), the dynamics of learning may be different from the non conditional case. In the beginning of training, the discriminator ignores the conditioning information and easily rejects samples from G because they do not look plausible.Once G has learned to generate plausible images, it must also learn to align them with the conditioning information, and likewise D must learn to evaluate whether samples from G meet this conditioning constraint.

In naive GAN, the discriminator observes two kinds of inputs: real images with matching text, and synthetic images with arbitrary text. Therefore, it must implicitly separate two sources of error: unrealistic images (for any text), and realistic images of the wrong class that mismatch the conditioning information. Based on the intuition that this may complicate learning dynamics, we modified the GAN training algorithm to separate these error sources. In addition to the real / fake inputs to the discriminator during training, we add a third type of input consisting of real images with mismatched text, which the discriminator must learn to score as fake. By learning to optimize image / text matching in addition to the image realism, the discriminator can provide an additional signal to the generator.

Algorithm 5 summarizes the training procedure .

Algorithm 5 GAN-CLS training algorithm with step size α, using minibatch SGD for simplicity.

1: Input: minibatch images x, matching text t, mismatching $\hat{t}$, number of training batch steps S.
2: **for** $n = 1$ **to** S **do**
3: $h \leftarrow \varphi(t)$ {Encode matching text description}
4: $\hat{h} \leftarrow \varphi(\hat{t})$ {Encode mis-matching text description}
5: $z \sim \mathcal{N}(0, I)^z$ {Draw sample of random noise}
6: $\hat{x} \leftarrow G(z, h)$ {Forward through generator}
7: $s_r \leftarrow D(x, h)$ {real image, right text}
8: $s_w \leftarrow D(x, \hat{h})$ {real image, wrong text}
9: $s_f \leftarrow D(\hat{x}, h)$ {fake image, right text}
10: $\mathcal{L}_D \leftarrow log(s_r) + (log(1 - s_w) + log(1 - s_f))/2$
11: $D \leftarrow D - \alpha \partial \mathcal{L}_D / \partial D$ {Update discriminator}
12: $\mathcal{L}_G \leftarrow log(s_f)$
13: $G \leftarrow G - \alpha \partial \mathcal{L}_G / \partial G$ {Update generator}
14: **end for**

A.3.3 Learning with manifold interpolation (GAN-INT)

Deep networks have been shown to learn representations in which interpolations between embedding pairs tend to be near the data manifold ([17]Bengio et al., 2013; [8]Reed et al.,2014). Motivated by this property, we can generate a large amount of additional text embeddings by simply interpolating between embeddings of training set captions. Critically these interpolated text embeddings need not correspond to any actual human-written text, so there is no additional labeling cost. This can be viewed as adding an additional term to the generator objective to minimize:

$$\mathbb{E}_{t_1,t_2\sim p_{data}}[log(1 - D(G(z, \beta t_1 + (1-\beta)t_2)))] \tag{A.5}$$

Because the interpolated embeddings are synthetic, the discriminator D does not have real corresponding image and text pairs to train on. However, D learns to predict whether image and text pairs match or not. Thus, if D does a good job at this, then by satisfying D on interpolated text embeddings G can learn to fill in gaps on the data manifold in between training points.

A.3.4 Experimental Results of GAN

In this section we first present results on the CUB dataset of bird images and the Oxford-102 dataset of flower images. CUB has 11,788 images of birds belonging to one of 200 different categories. The Oxford-102 contains 8,189 images of flowers from 102 different categories. As in Akata et al. (2015) and Reed et al. (2016), we split these into class-disjoint training and test sets. CUB has 150 train+val classes and 50 test classes, while Oxford-102 has 82 train+val and 20 test classes. For both datasets, we used 5 captions per image. During mini-batch selection for training we randomly pick an image view (e.g. crop, flip) of the image and one of the captions.

For text features, we first pre-train a deep convolutional-recurrent text encoder on structured joint embedding of text captions with 1,024 dimensional GoogLeNet image embedings (Szegedy et al., 2015) as described in subsection 3.2. For both Oxford-102 and CUB we used a hybrid of character-level ConvNet with a recurrent neural network (char-CNN-RNN) as described in (Reed et al., 2016). Note, however that pre-training the text encoder is not a requirement of our method and we include some end-to-end results in the supplement. The reason for pre-training the text encoder was to increase the speed of training the other components for faster experimentation. We also provide some qualitative results obtained with MS COCO images of the validation set to show the generalizability of our approach.

We used the same GAN architecture for all datasets. The training image size was set to 64 64 3. The text encoder produced 1024-dimensional embeddings that were projected to 128 dimensions in both the generator and discriminator before depth concatenation into convolutional feature maps. As indicated in Algorithm 1, we take alternating steps of updating the generator and the discriminator network. We used the same base learning rate of 0.0002, and used the ADAM solver (Ba & Kingma, 2015) with momentum 0.5. The generator noise was sampled from a 100-dimensional unit normal distribution. We used a minibatch size of 64 and trained for 600 epochs. Our implementation was built on top of dcgan.torch.

This results from ([13]here)

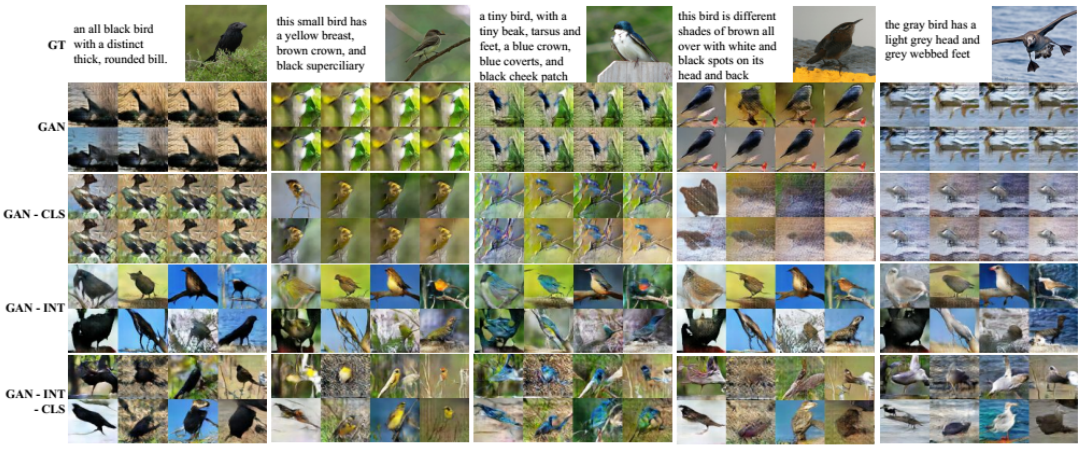


FIGURE A.2: Zero-shot generated bird images using GAN, GAN-CLS, GAN-INT and GAN-INT-CLS. We found that interpolation regularizer was needed to reliably achieve visually-plausible results.

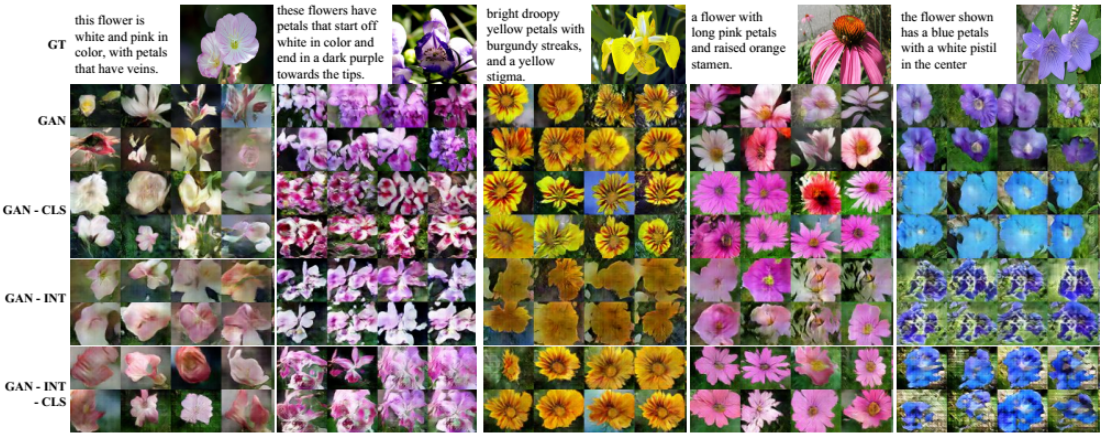


FIGURE A.3: Zero-shot generated flower images using GAN, GAN-CLS, GAN-INT and GAN-INT-CLS. All variants generated plausible images. Although some shapes of test categories were not seen during training (e.g. columns 3 and 4), the color information is preserved

References

[1] Farhadi, A., Endres, I., Hoiem, D., Forsyth, and D. Describing objects by their attributes. *In CVRP*, 2009.

[2] Kumar, N., Berg, A. C., Belhumeur, P. N., Nayar, and S. K. Attribute and simile classifiers for face verification. *In ICCV*, 2009.

[3] Parikh, D., Grauman, and K. Relative attributes. *In ICCV*, 2011.

[4] Lampert, C. H., Nickisch, H., , Harmeling, and S. Attribute-based classification for zero-shot visual object categorization. *In TPAMI*, 2014.

[5] Fu, Y., Hospedales, T. M., Xiang, T., Fu, Z., , Gong, and S. Transductive multi-view embedding for zero-shot recog-nition and annotation. *In ECCV*, 2014.

[6] Akata, Z., Reed, S., Walter, D., Lee, H., Schiele, and B. Evaluation of output embeddings for fine-grained im-age classification. *In CVPR*, 2015.

[7] Yan, X., Yang, J., Sohn, K., Lee, and H. Attribute2image:conditional image generation from visual attributes. *arXiv preprint arXiv:1512.00570*, 2015.

[8] Reed, S., Lee Akata, Z., H., , Schiele, and B. Learning deeprepresentations for fine-grained visual descriptions. *In CVPR*, 2016.

[9] Wah, C., Branson, S., Welinder, P., Perona, P., , Be-longie, and S. The caltech-ucsd birds-200-2011 dataset. 2011.

[10] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B. andWarde Farley, D., Ozair, S., Courville, A., , Bengio, and Y. Generative adversarial nets. *In NIPS*, 2014.

[11] Mirza, M., Osindero, and S. Conditional generative adversarial nets.s. 2014.

[12] Denton, E. L., Chintala, S., Fergus, R., and et al. Deep generative image models using a laplacian pyramid of adversarial networks. *In NIPS*, 2015.

[13] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. *in ICML*, 2016.

[14] Kfir Aberman, Jing Liao, Mingyi Shi, Dani Lischinski, Baoquan Chen, and Daniel Cohen-Or. Neural best-buddies: Sparse cross-domain correspondences. *arXiv:1805.04140*, 2018.

[15] Ioffe, S., Szegedy, and C. Batch normalization: Accelerat-ing deep network training by reducing internal covariateshift. *In ICML*, 2015.

[16] Gauthier and J. Conditional generative adversarial nets for convolutional face generation. *Technical report*, 2015.

[17] Bengio, Y., Mesnil, G., Dauphin, Y., and Rifai S. Better mixing via deep representations. *In ICML*, 2013.