**A Realistic Image Generation of Face From Text Description Using the Fully Trained Generative Adversarial Networks**

**1. INTRODUCTION:**

Generating images using the text description is one of most challenging and important task in machine learning. This task involves to handle the language modalities problems which include the control and management of the incomplete and ambiguous information using the natural language processing techniques and algorithms. After that, this information is used to learn by computer vision approaches and algorithms Currently, it is one of the latest research domain in computer vision. Generating images from text is the opposite process of image captioning and image classification, where we generate text and caption from images. Just like the image captions, text to image generation helps to find context and relationship between the image and the text along with exploring human visual semantics. Moreover, it has the large number of applications in art, designs, image retrieval and searching etc. Currently, most of the methods for generating images from text are based on the traditional method, in which the pretrained text encoder has been utilized in order to get the semantic vector from input descriptions. Based on the semantic vectors, conditional GAN are trained to generate the realistic images. Although, this method generates the high quality images, but they split the training method into two ways, based on the text encoder and image decoder. Most of generative adversarial network mostly focus on the generation of the synthesized images using the sentence level information. Generating the images using the sentence level information have most probably chances of loss of information at word level. As a result, the high quality images can’t be generated. Currently, most of the work which was done for the problem of “Text to Image” is based on the simple dataset problems such as birds and flowers. But, the work which mapped the objects along with scenes is very limited. To overcome this problem, et.al utilized the AttnGAN, but they were failed to achieve the good results as their output image is semantically not meaningful. They tried to explore the COCO dataset and mapped the object along with the scene with the sentence level information. However, the object and word level information is still missing. Text to face image generation is the sub domain of the text to image generation, where the ultimate goal is to generate the image using the user specified description about the face. So, basically there are two major tasks of generating face images from text. Fig. 1 shows the input and output models for a text to image synthesis system. Hence, text to face synthesis involves generating the high quality images and generating the appropriate images related to the given description. This task of generating the face images from the text description is more relevant to the public safety tasks. For example, we consider the scenario of the crime scene. In most of the countries, the witness of the crime scene has appeared before the law enforcement agencies in order to help in drawing the portrait of the suspicious criminal. The witness tells the description of the criminal to the portrait maker, then he/she draws the portrait of the criminal on the drawing board. The proposed work, will help to automate the whole task by negating the role of the portrait maker. The manual work is tedious and time consuming and requires professional knowledge and experience. Thus, this work will be helpful for the law-enforcement agencies. There are different datasets available for text to image synthesis, like CUB, Oxford102, and COCO. But there is no standard dataset, which is available for text to face generation. The work done in the domain of text to face generation is very limited. In this paper, self-generated dataset is also presented with the help of the google image search and two publically available datasets for face to text generation. The main motivation behind this research work is to generate the synthesized images of the face based on the text description. The proposed algorithms in this paper have ensured to generate the high quality images by preserving the face identity. Moreover, it is also capable of generating the exact images based on the given descriptions. This research work has also been utilized in many industrial applications like automatic sketch making of the suspected face in crime investigation departments. Currently, they utilize the human skills to draw the face sketch of suspected person based on the information of the eyewitness. That is tedious and requires a lot of time and technical skills to accomplish we have make the following contributions in our paper.

1. Generating the dataset related to the text to face images.

2. Proposed a Fully Trained Generative Adversarial Network, which have trainable text encoder as well as trainable image decoder.

3. Two discriminators are proposed to utilize the strength of joint learning.

4. Generating the photo-realistic images of the faces from the description by preserving details.

**1.1 Objective of the project:**

Text to face generation is a sub domain of text to image synthesis, and it has a huge impact along with the wide range of applications on public safety domain. Currently, due to the lack of dataset, the research work focused on the face to text generation is very limited. Most of the work for text to face generation till now based on the partially trained generative adversarial network, in which the pre-trained text encoder has been used to extract the sematic features of input sentence. Then these semantic features have been utilized to train the image decoder. But in this research work, we have proposed the fully trained generative adversarial network to generate the realistic and natural images. We have trained the text encoder as well as the image decoder at the same time to generate the more accurate and efficient results. In addition to proposed methodology, we have also generated the dataset by the amalgamation of LFW, CelebA and locally prepared dataset. We have also labeled the images according to our defined classes. Through performing different kind of experiments, we have proved that our proposed fully trained GAN outperformed by generating the good quality images with accordance to the input sentence. Moreover, the visual results have also strengthened our experiments by generating the face images according to the given query.

**2. LITERATURE SURVEY:**

**“Generative adversarial nets,”**

We propose a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game. In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to 1/2 everywhere. In the case where G and D are defined by multilayer perceptrons, the entire system can be trained with back propagation. There is no need for any Markov chains or unrolled approximate inference networks during either training or generation of samples. Experiments demonstrate the potential of the framework through qualitative and quantitative evaluation of the generated samples.

**“Inferring semantic layout for hierarchical text-to-image synthesis,”**

We propose a novel hierarchical approach for text-to-image synthesis by inferring semantic layout. Instead of learning a direct mapping from text to image, our algorithm decomposes the generation process into multiple steps, in which it first constructs a semantic layout from the text by the layout generator and converts the layout to an image by the image generator. The proposed layout generator progressively constructs a semantic layout in a coarse-to-fine manner by generating object bounding boxes and refining each box by estimating object shapes inside the box. The image generator synthesizes an image conditioned on the inferred semantic layout, which provides a useful semantic structure of an image matching with the text description. Our model not only generates semantically more meaningful images, but also allows automatic annotation of generated images and user-controlled generation process by modifying the generated scene layout. We demonstrate the capability of the proposed model on challenging MS-COCO dataset and show that the model can substantially improve the image quality, interpretability of output and semantic alignment to input text over existing approaches.

**“Conditional image generation with pixelcnn decoders,”**

This work explores conditional image generation with a new image density model based on the Pixel CNN architecture. The model can be conditioned on any vector, including descriptive labels or tags, or latent embeddings created by other networks. When conditioned on class labels from the Image Net database, the model is able to generate diverse, realistic scenes representing distinct animals, objects, landscapes and structures. When conditioned on an embedding produced by a convolution network given a single image of an unseen face, it generates a variety of new portraits of the same person with different facial expressions, poses and lighting conditions. We also show that conditional Pixel CNN can serve as a powerful decoder in an image auto encoder. Additionally, the gated convolution layers in the proposed model improve the log-likelihood of PixelCNN to match the state-of-the-art performance of PixelRNN on Image Net, with greatly reduced computational cost.

**“Stackgan++: Realistic image synthesis with stacked generative adversarial networks,”**

Although Generative Adversarial Networks (GANs) have shown remarkable success in various tasks, they still face challenges in generating high quality images. In this paper, we propose Stacked Generative Adversarial Networks (Stack GAN) aimed at generating high-resolution photorealistic images. First, we propose a two-stage generative adversarial network architecture, StackGAN-v1, for text-to-image synthesis. The Stage-I GAN sketches primitive shape and colors of the object based on given text description, yielding low-resolution images. The Stage-II GAN takes Stage-I results and text descriptions as inputs, and generates high-resolution images with photo-realistic details. Second, advanced multi-stage generative adversarial network architecture, StackGAN-v2, is proposed for both conditional and unconditional generative tasks. Our StackGAN-v2 consists of multiple generators and discriminators in a tree-like structure; images at multiple scales corresponding to the same scene are generated from different branches of the tree. StackGAN-v2 shows more stable training behavior than StackGAN-v1 by jointly approximating multiple distributions. Extensive experiments demonstrate that the proposed stacked generative adversarial networks significantly outperform other state-of-the-art methods in generating photo-realistic images

**“The caltech-ucsd birds-200-2011 dataset,”**

CUB-200-2011 is an extended version of CUB-200 [7], a challenging dataset of 200 bird species. The extended version roughly doubles the number of images per category and adds new part localization annotations. All images are annotated with bounding boxes, part locations, and at- tribute labels. Images and annotations were filtered by mul- tiple users of Mechanical Turk. We introduce benchmarks and baseline experiments for multi-class categorization and part localization.

**“Automated flower classification over a large number of classes,”**

We investigate to what extent combinations of features can improve classification performance on a large dataset of similar classes. To this end we introduce a 103 class flower dataset. We compute four different features for the flowers, each describing different aspects, namely the local shape/texture, the shape of the boundary, the overall spatial distribution of petals, and the colour. We combine the features using a multiple kernel framework with a SVM classifier. The weights for each class are learnt using the method of Varma and Ray, which has achieved state of the art performance on other large dataset, such as Caltech 101/256. Our dataset has a similar challenge in the number of classes, but with the added difficulty of large between class similarity and small within class similarity. Results show that learning the optimum kernel combination of multiple features vastly improves the performance, from 55.1% for the best single feature to 72.8% for the combination of all features

**“Microsoft coco: Common objects in context,”**

We present a new dataset with the goal of advancing the state-of-the-art in object recognition by placing the question of object recognition in the context of the broader question of scene understanding. This is achieved by gathering images of complex everyday scenes containing common objects in their natural context. Objects are labeled using per-instance segmentations to aid in precise object localization. Our dataset contains photos of 91 objects types that would be easily recognizable by a 4 year old. With a total of 2.5 million labeled instances in 328k images, the creation of our dataset drew upon extensive crowd worker involvement via novel user interfaces for category detection, instance spotting and instance segmentation. We present a detailed statistical analysis of the dataset in comparison to PASCAL, ImageNet, and SUN. Finally, we provide baseline performance analysis for bounding box and segmentation detection results using a Deformable Parts Model.

**“Auto-encoding variation bayes,”**

How can we perform efficient inference and learning in directed probabilistic models, in the presence of continuous latent variables with intractable posterior distributions, and large datasets? We introduce a stochastic variation inference and learning algorithm that scales to large datasets and, under some mild differentiability conditions, even works in the intractable case. Our contributions are two-fold. First, we show that a reparameterization of the variational lower bond yields a lower bound estimator that can be straightforwardly optimized using standard stochastic gradient methods. Second, we show that for i.i.d. datasets with continuous latent variables per data point, posterior inference can be made especially efficient by fitting an approximate inference model (also called a recognition model) to the intractable posterior using the proposed lower bound estimator. Theoretical advantages are reflected in experimental results.

**“Generative adversarial text to image synthesis,”**

Automatic synthesis of realistic images from text would be interesting and useful, but current AI systems are still far from this goal. However, in recent years generic and powerful recurrent neural network architectures have been developed to learn discriminative text feature representations. Meanwhile, deep convolutional generative adversarial networks (GANs) have begun to generate highly compelling images of specific categories such as faces, album covers, room interiors and flowers. In this work, we develop a novel deep architecture and GAN formulation to effectively bridge these advances in text and image modeling, translating visual concepts from characters to pixels. We demonstrate the capability of our model to generate plausible images of birds and flowers from detailed text descriptions.

**“Unsupervised representation learning with deep convolutional generative adversarial networks,”**

In recent years, supervised learning with convolutional networks (CNNs) has seen huge adoption in computer vision applications. Comparatively, unsupervised learning with CNNs has received less attention. In this work we hope to help bridge the gap between the success of CNNs for supervised learning and unsupervised learning. We introduce a class of CNNs called deep convolutional generative adversarial networks (DCGANs), which have certain architectural constraints, and demonstrate that they are a strong candidate for unsupervised learning. Training on various image datasets, we show convincing evidence that our deep convolutional adversarial pair learns a hierarchy of representations from object parts to scenes in both the generator and discriminator. Additionally, we use the learned features for novel tasks - demonstrating their applicability as general image representations

**3. SYSTEM ANALYSIS**

**3.1 Existing System**

A part of text to image synthesis is text to face generation. Along with the broad range of applications in the field of public safety, it has a significant impact on new research areas. The study on text to face generation is fairly limited as a result of the dearth of datasets. The most of the text's work the pre-trained text encoder has been utilized to extract the semantic properties of the input sentence, face generation up till now has been based on the partially trained generative adversarial networks. The image decoder has since been trained using these semantic features.

**Disadvantages of Existing System:**

1. Less Accuracy

**3.2 Proposed System**

GAN algorithm was used to decode images and BI-LSTM (Bidirectional-Long Short Term Memory) was used to encode TEXT. Both models will be combine to decode image based on input text sentences. BI-LSTM will map each input sentence to related image and then get trained to generate image decoding model. Input TEXT will be converted into VECTOR and this vector will be input to BI-LSTM model to encode vector and this vector will be input to GAN to decode image. In this paper author propose a fully trained generative adversarial network to generate realistic and natural images. The proposed work trained the text encoder as well as the image decoder at the same time to generate more accurate and efficient results

**Advantages of Proposed System**

1. More Accuracy.

**Modules Information:**

To implement this project we have designed following modules

1. Upload CelebA Dataset: using this module we will upload dataset images to application and this module we will read all images and description and then saved it as array
2. Generate Fully Trained GAN Model: images and description array will be input to Proposed Fully Trained GAN to generate TEXT to face conversion model. This model will differentiate gender using MALE and FEMALE and age will be differentiating using YOUNG and OLD words.
3. Generate Image from Text: this module will take sentences as TEXT and then input to GAN model which will covert TEXT into vector and then input this vector to GAN to decode vector to faces.

**FUNCTIONAL REQUIREMENTS:**

**SOFTWARE REQIREMENTS:**

**System Attributes:**

1. filename
2. gan\_model
3. encoder model
4. X, Y

**Data base Requirements:**

No need

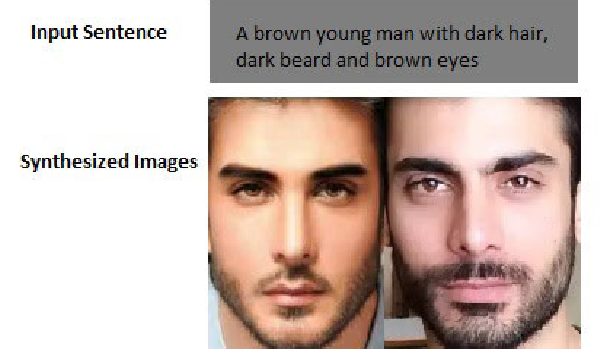
**USECASE:**

Use cases - Use cases describe the interaction between the system and external users that leads to achieving particular goals.

1. Upload CelebA Dataset
2. Generate Fully Trained GAN Model
3. Generate Image from Text

**User Stories:** In this paper author is using fully trained GAN (generative adversarial networks) algorithm to convert TEXT to Face. GAN algorithms are a deep learning algorithm which was introduced to generate Fake images by getting trained on sample images. GAN can be used to train on sample images and after training it can predict related fake images from given input images.

**Work down Structure:**

****

**Prototype:**

python 3.7.0 or 3.7.4

opencv-python==4.5.1.48

keras==2.3.1

tensorflow==1.14.0

protobuf==3.16.0

h5py==2.10.0

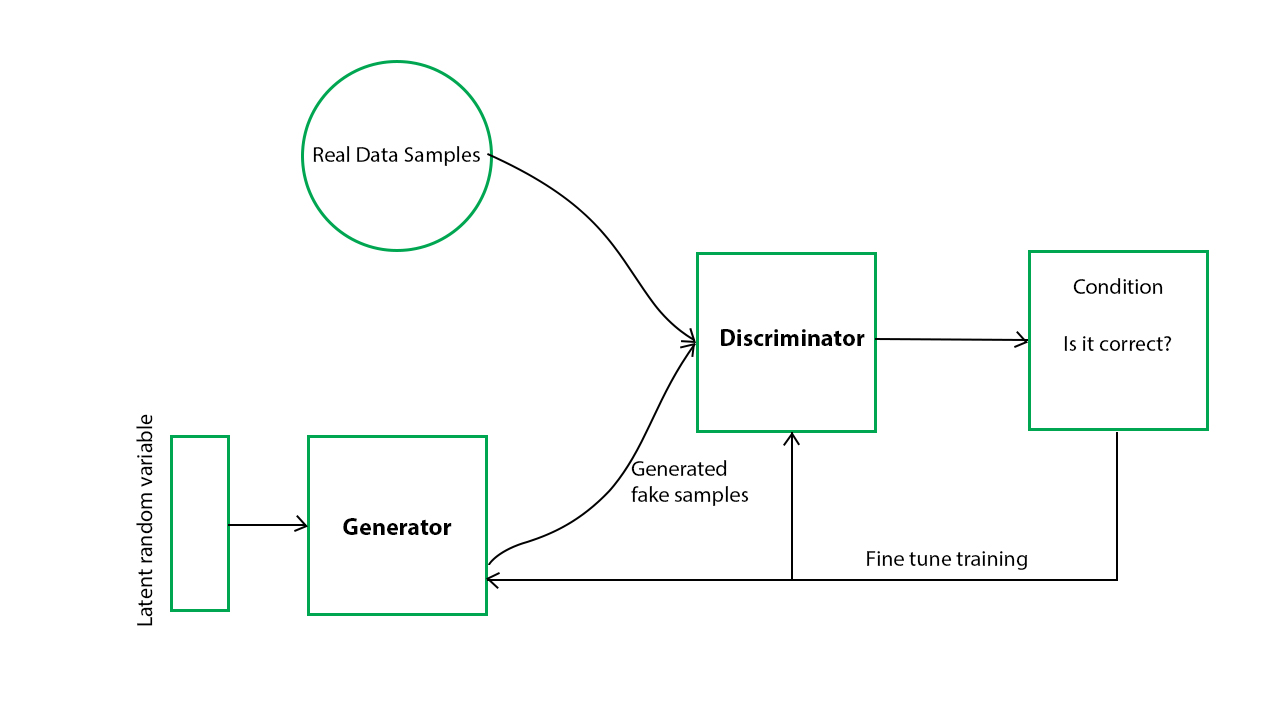
sklearn-extensions==0.0.2

scikit-learn==0.22.2.post1

Numpy

Pandas

**Models and Diagrams:**



**NON-FUNCTIONAL REQUIREMENT:**

**Usability:**  Usability is a quality attribute that assesses how easy user interfaces are to use. The word "usability" also refers to methods for improving ease-of-use during the design process.(how it was handle entire project easy)

**Security:** the quality or state of being secure: such as. a : freedom from danger : safety. b : freedom from fear or anxiety. c : freedom from the prospect of being laid off job security.

**Readability:** Readability is the ease with which a reader can understand a written text.

**Performance**: the execution of an action. : something accomplished : deed, feat. : the fulfillment of a claim, promise, or request : implementation. 3. : the action of representing a character in a play.

**Availability**: the quality or state of being available trying to improve the availability of affordable housing. 2 : an available person or thing.

**Scalability**: Scalability is the measure of a system's ability to increase or decrease in performance and cost in response to changes in application and system processing demands.

**3.3. PROCESS MODEL USED WITH JUSTIFICATION**

**SDLC (Umbrella Model):**

**Umbrella Activity**

**Umbrella Activity**

**Umbrella Activity**

1. Feasibility Study
2. TEAM FORMATION
3. Project Specification PREPARATION

Business Requirement Documentation

ANALYSIS & DESIGN

CODE

UNIT TEST

DOCUMENT CONTROL

ASSESSMENT

TRAINING

INTEGRATION & SYSTEM TESTING

DELIVERY/INSTALLATION

ACCEPTANCE TEST

Requirements Gathering

SDLC is nothing but Software Development Life Cycle. It is a standard which is used by software industry to develop good software.

**Stages in SDLC:**

* Requirement Gathering
* Analysis
* Designing
* Coding
* Testing
* Maintenance

**Requirements Gathering stage:**

The requirements gathering process takes as its input the goals identified in the high-level requirements section of the project plan. Each goal will be refined into a set of one or more requirements. These requirements define the major functions of the intended application, define operational data areas and reference data areas, and define the initial data entities. Major functions include critical processes to be managed, as well as mission critical inputs, outputs and reports. A user class hierarchy is developed and associated with these major functions, data areas, and data entities. Each of these definitions is termed a Requirement. Requirements are identified by unique requirement identifiers and, at minimum, contain a requirement title and textual description.



These requirements are fully described in the primary deliverables for this stage: the Requirements Document and the Requirements Traceability Matrix (RTM). The requirements document contains complete descriptions of each requirement, including diagrams and references to external documents as necessary. Note that detailed listings of database tables and fields are *not* included in the requirements document.

The title of each requirement is also placed into the first version of the RTM, along with the title of each goal from the project plan. The purpose of the RTM is to show that the product components developed during each stage of the software development lifecycle are formally connected to the components developed in prior stages.

In the requirements stage, the RTM consists of a list of high-level requirements, or goals, by title, with a listing of associated requirements for each goal, listed by requirement title. In this hierarchical listing, the RTM shows that each requirement developed during this stage is formally linked to a specific product goal. In this format, each requirement can be traced to a specific product goal, hence the term requirements traceability.

The outputs of the requirements definition stage include the requirements document, the RTM, and an updated project plan.

* Feasibility study is all about identification of problems in a project.
* No. of staff required to handle a project is represented as Team Formation, in this case only modules are individual tasks will be assigned to employees who are working for that project.
* Project Specifications are all about representing of various possible inputs submitting to the server and corresponding outputs along with reports maintained by administrator.

**Analysis Stage:**

The planning stage establishes a bird's eye view of the intended software product, and uses this to establish the basic project structure, evaluate feasibility and risks associated with the project, and describe appropriate management and technical approaches.



The most critical section of the project plan is a listing of high-level product requirements, also referred to as goals. All of the software product requirements to be developed during the requirements definition stage flow from one or more of these goals. The minimum information for each goal consists of a title and textual description, although additional information and references to external documents may be included. The outputs of the project planning stage are the configuration management plan, the quality assurance plan, and the project plan and schedule, with a detailed listing of scheduled activities for the upcoming Requirements stage, and high level estimates of effort for the out stages.

**Designing Stage:**

The design stage takes as its initial input the requirements identified in the approved requirements document. For each requirement, a set of one or more design elements will be produced as a result of interviews, workshops, and/or prototype efforts. Design elements describe the desired software features in detail, and generally include functional hierarchy diagrams, screen layout diagrams, tables of business rules, business process diagrams, pseudo code, and a complete entity-relationship diagram with a full data dictionary. These design elements are intended to describe the software in sufficient detail that skilled programmers may develop the software with minimal additional input.

  
When the design document is finalized and accepted, the RTM is updated to show that each design element is formally associated with a specific requirement. The outputs of the design stage are the design document, an updated RTM, and an updated project plan.

**Development (Coding) Stage:**

The development stage takes as its primary input the design elements described in the approved design document. For each design element, a set of one or more software artifacts will be produced. Software artifacts include but are not limited to menus, dialogs, and data management forms, data reporting formats, and specialized procedures and functions. Appropriate test cases will be developed for each set of functionally related software artifacts, and an online help system will be developed to guide users in their interactions with the software.



The RTM will be updated to show that each developed artifact is linked to a specific design element, and that each developed artifact has one or more corresponding test case items. At this point, the RTM is in its final configuration. The outputs of the development stage include a fully functional set of software that satisfies the requirements and design elements previously documented, an online help system that describes the operation of the software, an implementation map that identifies the primary code entry points for all major system functions, a test plan that describes the test cases to be used to validate the correctness and completeness of the software, an updated RTM, and an updated project plan.

**Integration & Test Stage:**

During the integration and test stage, the software artifacts, online help, and test data are migrated from the development environment to a separate test environment. At this point, all test cases are run to verify the correctness and completeness of the software. Successful execution of the test suite confirms a robust and complete migration capability. During this stage, reference data is finalized for production use and production users are identified and linked to their appropriate roles. The final reference data (or links to reference data source files) and production user list are compiled into the Production Initiation Plan.



The outputs of the integration and test stage include an integrated set of software, an online help system, an implementation map, a production initiation plan that describes reference data and production users, an acceptance plan which contains the final suite of test cases, and an updated project plan.

* **Installation & Acceptance Test:**

During the installation and acceptance stage, the software artifacts, online help, and initial production data are loa ded onto the production server. At this point, all test cases are run to verify the correctness and completeness of the software. Successful execution of the test suite is a prerequisite to acceptance of the software by the customer.

After customer personnel have verified that the initial production data load is correct and the test suite has been executed with satisfactory results, the customer formally accepts the delivery of the software.



The primary outputs of the installation and acceptance stage include a production application, a completed acceptance test suite, and a memorandum of customer acceptance of the software. Finally, the PDR enters the last of the actual labor data into the project schedule and locks the project as a permanent project record. At this point the PDR "locks" the project by archiving all software items, the implementation map, the source code, and the documentation for future reference.

**Maintenance:**

Outer rectangle represents maintenance of a project, Maintenance team will start with requirement study, understanding of documentation later employees will be assigned work and they will undergo training on that particular assigned category. For this life cycle there is no end, it will be continued so on like an umbrella (no ending point to umbrella sticks).

**3.4. Software Requirement Specification**

**3.4.1. Overall Description**

A Software Requirements Specification (SRS) – a [requirements specification](http://en.wikipedia.org/wiki/Requirements_specification) for a [software system](http://en.wikipedia.org/wiki/Software_system) is a complete description of the behavior of a system to be developed. It includes a set of [use cases](http://en.wikipedia.org/wiki/Use_case) that describe all the interactions the users will have with the software. In addition to use cases, the SRS also contains non-functional requirements. [Nonfunctional requirements](http://en.wikipedia.org/wiki/Non-functional_requirements) are requirements which impose constraints on the design or implementation (such as [performance engineering](http://en.wikipedia.org/wiki/Performance_engineering) requirements, [quality](http://en.wikipedia.org/wiki/Quality_%28business%29) standards, or design constraints).

System requirements specification: A structured collection of information that embodies the requirements of a system. A [business analyst](http://en.wikipedia.org/wiki/Business_analyst), sometimes titled [system analyst](http://en.wikipedia.org/wiki/System_analyst), is responsible for analyzing the business needs of their clients and stakeholders to help identify business problems and propose solutions. Within the [systems development lifecycle](http://en.wikipedia.org/wiki/Systems_development_life_cycle) domain, the BA typically performs a liaison function between the business side of an enterprise and the information technology department or external service providers. Projects are subject to three sorts of requirements:

* [Business requirements](http://en.wikipedia.org/wiki/Business_requirements) describe in business terms what must be delivered or accomplished to provide value.
* Product requirements describe properties of a system or product (which could be one of several ways to accomplish a set of business requirements.)
* Process requirements describe activities performed by the developing organization. For instance, process requirements could specify .Preliminary investigation examine project feasibility, the likelihood the system will be useful to the organization. The main objective of the feasibility study is to test the Technical, Operational and Economical feasibility for adding new modules and debugging old running system. All system is feasible if they are unlimited resources and infinite time. There are aspects in the feasibility study portion of the preliminary investigation:
* **ECONOMIC FEASIBILITY**

A system can be developed technically and that will be used if installed must still be a good investment for the organization. In the economical feasibility, the development cost in creating the system is evaluated against the ultimate benefit derived from the new systems. Financial benefits must equal or exceed the costs. The system is economically feasible. It does not require any addition hardware or software. Since the interface for this system is developed using the existing resources and technologies available at NIC, There is nominal expenditure and economical feasibility for certain.

* **Operational Feasibility**

Proposed projects are beneficial only if they can be turned out into information system. That will meet the organization’s operating requirements. Operational feasibility aspects of the project are to be taken as an important part of the project implementation. This system is targeted to be in accordance with the above-mentioned issues. Beforehand, the management issues and user requirements have been taken into consideration. So there is no question of resistance from the users that can undermine the possible application benefits. The well-planned design would ensure the optimal utilization of the computer resources and would help in the improvement of performance status.

* **TECHNICAL FEASIBILITY**

Earlier no system existed to cater to the needs of ‘Secure Infrastructure Implementation System’. The current system developed is technically feasible. It is a web based user interface for audit workflow at NIC-CSD. Thus it provides an easy access to .the users. The database’s purpose is to create, establish and maintain a workflow among various entities in order to facilitate all concerned users in their various capacities or roles. Permission to the users would be granted based on the roles specified. Therefore, it provides the technical guarantee of accuracy, reliability and security.

**3.4.2. External Interface Requirements**

**User Interface**

The user interface of this system is a user friendly python Graphical User Interface.

**Hardware Interfaces**

The interaction between the user and the console is achieved through python capabilities.

**Software Interfaces**

The required software is python.

**SYSTEM REQUIREMENT:**

**HARDWARE REQUIREMENTS:**

# Processor - Intel i3(min)

* Speed - 1.1 GHz
* RAM - 4GB(min)
* Hard Disk - 500 GB
* Key Board - Standard Windows Keyboard
* Mouse - Two or Three Button Mouse
* Monitor - SVGA

**SOFTWARE REQUIREMENTS:**

* Operating System - Windows10(min)
* Programming Language - Python

**4. SYSTEM DESIGN**

**CLASS DIAGRAM:**

The class diagram is the main building block of object oriented modeling. It is used both for general conceptual modeling of the systematic of the application, and for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main objects, interactions in the application and the classes to be programmed. In the diagram, classes are represented with boxes which contain three parts:

* The upper part holds the name of the class
* The middle part contains the attributes of the class
* The bottom part gives the methods or operations the class can take or undertake



**USECASE DIAGRAM:**

A **use case diagram** at its simplest is a representation of a user's interaction with the system and depicting the specifications of a use case. A use case diagram can portray the different types of users of a system and the various ways that they interact with the system. This type of diagram is typically used in conjunction with the textual use case and will often be accompanied by other types of diagrams as we



**SEQUENCE DIAGRAM**

A **sequence diagram** is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called **event diagrams**, **event scenarios**, and timing diagrams.



**COLLABORATION DIAGRAM:**

A collaboration diagram describes interactions among objects in terms of sequenced messages. Collaboration diagrams represent a combination of information taken from class, sequence, and use case diagrams describing both the static structure and dynamic behaviour of a system.

****

**COMPONENT DIAGRAM:**

In the Unified Modelling Language, a component diagram depicts how components are wired together to form larger components and or software systems. They are used to illustrate the structure of arbitrarily complex systems.

Components are wired together by using an assembly connector to connect the required interface of one component with the provided interface of another component. This illustrates the service consumer - service provider relationship between the two components.



**DEPLOYMENT DIAGRAM:**

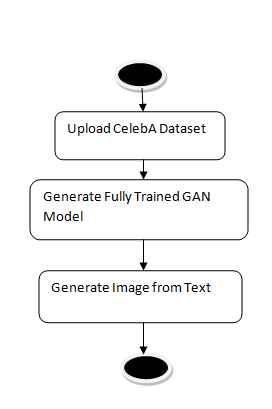
A **deployment diagram** in the Unified Modeling Language models the *physical* deployment of artifacts on nodes. To describe a web site, for example, a deployment diagram would show what hardware components ("nodes") exist (e.g., a web server, an application server, and a database server), what software components ("artifacts") run on each node (e.g., web application, database), and how the different pieces are connected (e.g. JDBC, REST, RMI).

The nodes appear as boxes, and the artifacts allocated to each node appear as rectangles within the boxes. Nodes may have sub nodes, which appear as nested boxes. A single node in a deployment diagram may conceptually represent multiple physical nodes, such as a cluster of database servers.

****

**ACTIVITY DIAGRAM:**

Activity diagram is another important diagram in UML to describe dynamic aspects of the system. It is basically a flow chart to represent the flow form one activity to another activity. The activity can be described as an operation of the system. So the control flow is drawn from one operation to another. This flow can be sequential, branched or concurrent



Upload CelebA Dataset

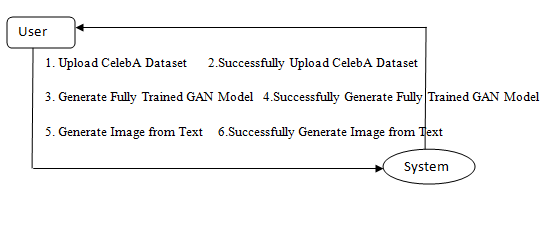
Generate Fully Trained GAN Model

Generate Image from Text

**Data flow :**

Data flow diagrams illustrate how data is processed by a system in terms of inputs and outputs. Data flow diagrams can be used to provide a clear representation of any business function. The technique starts with an overall picture of the business and continues by analyzing each of the functional areas of interest. This analysis can be carried out in precisely the level of detail required. The technique exploits a method called top-down expansion to conduct the analysis in a targeted way.

As the name suggests, Data Flow Diagram (DFD) is an illustration that explicates the passage of information in a process. A DFD can be easily drawn using simple symbols. Additionally, complicated processes can be easily automated by creating DFDs using easy-to-use, free downloadable diagramming tools. A DFD is a model for constructing and analyzing information processes. DFD illustrates the flow of information in a process depending upon the inputs and outputs. A DFD can also be referred to as a Process Model. A DFD demonstrates business or technical process with the support of the outside data saved, plus the data flowing from the process to another and the end results.



**5. IMPLEMETATION**

**5.1 Python**

Python is a general-purpose language. It has wide range of applications from Web development (like: Django and Bottle), scientific and mathematical computing (Orange, SymPy, NumPy) to desktop graphical user Interfaces (Pygame, Panda3D). The syntax of the language is clean and length of the code is relatively short. It's fun to work in Python because it allows you to think about the problem rather than focusing on the syntax.

**History of Python:**

Python is a fairly old language created by Guido Van Rossum. The design began in the late 1980s and was first released in February 1991.

**Why Python was created?**

In late 1980s, Guido Van Rossum was working on the Amoeba distributed operating system group. He wanted to use an interpreted language like ABC (ABC has simple easy-to-understand syntax) that could access the Amoeba system calls. So, he decided to create a language that was extensible. This led to design of a new language which was later named Python.

**Why the name Python?**

No. It wasn't named after a dangerous snake. Rossum was fan of a comedy series from late seventies. The name "Python" was adopted from the same series "Monty Python's Flying Circus".

**Features of Python:**

**A simple language which is easier to learn**

Python has a very simple and elegant syntax. It's much easier to read and write Python programs compared to other languages like: C++, Java, C#. Python makes programming fun and allows you to focus on the solution rather than syntax.

If you are a newbie, it's a great choice to start your journey with Python.

**Free and open-source**

You can freely use and distribute Python, even for commercial use. Not only can you use and distribute software’s written in it, you can even make changes to the Python's source code.

Python has a large community constantly improving it in each iteration.

**Portability**

You can move Python programs from one platform to another, and run it without any changes.

It runs seamlessly on almost all platforms including Windows, Mac OS X and Linux.

**Extensible and Embeddable**

Suppose an application requires high performance. You can easily combine pieces of C/C++ or other languages with Python code.

This will give your application high performance as well as scripting capabilities which other languages may not provide out of the box.

**A high-level, interpreted language**

Unlike C/C++, you don't have to worry about daunting tasks like memory management, garbage collection and so on.

Likewise, when you run Python code, it automatically converts your code to the language your computer understands. You don't need to worry about any lower-level operations.

**Large standard libraries to solve common tasks**

Python has a number of standard libraries which makes life of a programmer much easier since you don't have to write all the code yourself. For example: Need to connect MySQL database on a Web server? You can use MySQLdb library using import MySQLdb .

Standard libraries in Python are well tested and used by hundreds of people. So you can be sure that it won't break your application.

**Object-oriented**

Everything in Python is an object. Object oriented programming (OOP) helps you solve a complex problem intuitively.

With OOP, you are able to divide these complex problems into smaller sets by creating objects.

**Applications of Python:**

**1. Simple Elegant Syntax**

Programming in Python is fun. It's easier to understand and write Python code. Why? The syntax feels natural. Take this source code for an example:

a = 2

b = 3

sum = a + b

print(sum)

**2. Not overly strict**

You don't need to define the type of a variable in Python. Also, it's not necessary to add semicolon at the end of the statement.

Python enforces you to follow good practices (like proper indentation). These small things can make learning much easier for beginners.

**3. Expressiveness of the language**

Python allows you to write programs having greater functionality with fewer lines of code. Here's a link to the source code of Tic-tac-toe game with a graphical interface and a smart computer opponent in less than 500 lines of code. This is just an example. You will be amazed how much you can do with Python once you learn the basics.

**4. Great Community and Support**

Python has a large supporting community. There are numerous active forums online which can be handy if you are stuck.

**5.2 Sample Code:**

from tkinter import messagebox

from tkinter import \*

from tkinter import simpledialog

import tkinter

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

from tkinter import simpledialog

from tkinter import filedialog

from keras.models import model\_from\_json

from random import randrange

from numpy.random import randn

from keras.models import load\_model

from matplotlib import pyplot

import pandas as pd

import cv2

import os

from numpy import expand\_dims

from numpy import zeros

from numpy import ones

from numpy import vstack

from numpy.random import randn

from numpy.random import randint

from keras.optimizers import Adam

from keras.models import Sequential

from keras.layers import Dense,Reshape,Flatten,Conv2D,Conv2DTranspose,LeakyReLU,Dropout

from matplotlib import pyplot

import numpy as np

import cv2

from keras.layers import LSTM

from keras.layers import Bidirectional

from sklearn.metrics import roc\_curve

from sklearn.metrics import roc\_auc\_score

import pickle

import re

from numpy import dot

from numpy.linalg import norm

import json

from sklearn.feature\_extraction.text import TfidfVectorizer

main = tkinter.Tk()

main.title("A Realistic Image Generation of Face From Text Description Using the Fully Trained Generative Adversarial Networks") #designing main screen

main.geometry("1300x1200")

global filename

global gan\_model

global encoder\_model

global X, Y

# function to generate discriminator model

def define\_discriminator(in\_shape=(32,32,3)):

model = Sequential()

# normal

model.add(Conv2D(64, (3,3), padding='same', input\_shape=in\_shape))

model.add(LeakyReLU(alpha=0.2))

model.add(Bidirectional(LSTM(32))) #adding bilstm for text encoding

# downsample

model.add(Conv2D(128, (3,3), strides=(2,2), padding='same'))

model.add(LeakyReLU(alpha=0.2))

# downsample

model.add(Conv2D(128, (3,3), strides=(2,2), padding='same'))

model.add(LeakyReLU(alpha=0.2))

# downsample

model.add(Conv2D(256, (3,3), strides=(2,2), padding='same'))

model.add(LeakyReLU(alpha=0.2))

# classifier

model.add(Flatten())

model.add(Dropout(0.4))

model.add(Dense(1, activation='sigmoid'))

# compile model

opt = Adam(lr=0.0002, beta\_1=0.5)

model.compile(loss='binary\_crossentropy', optimizer=opt, metrics=['accuracy'])

return model

# function to generate standalone generator model

def define\_generator(latent\_dim):

model = Sequential()

# foundation for 4x4 image

n\_nodes = 256 \* 4 \* 4

model.add(Dense(n\_nodes, input\_dim=latent\_dim))

model.add(LeakyReLU(alpha=0.2))

model.add(Reshape((4, 4, 256)))

# upsample to 8x8

model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))

model.add(LeakyReLU(alpha=0.2))

# upsample to 16x16

model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))

model.add(LeakyReLU(alpha=0.2))

# upsample to 32x32

model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))

model.add(LeakyReLU(alpha=0.2))

# output layer

model.add(Conv2D(3, (3,3), activation='tanh', padding='same'))

return model

# define the combined generator and discriminator model, for updating the generator

def define\_gan(g\_model, d\_model):

# make weights in the discriminator not trainable

d\_model.trainable = False

# connect them

model = Sequential()

# add generator

model.add(g\_model)

# add the discriminator

model.add(d\_model)

# compile model

opt = Adam(lr=0.0002, beta\_1=0.5)

model.compile(loss='binary\_crossentropy', optimizer=opt)

return model

#load real dataset samples

def load\_real\_samples():

XX = np.load('model/Y.npy')

return XX

# select real samples to generate fake or related images

def generate\_real\_samples(dataset, n\_samples):

# choose random instances

ix = randint(0, dataset.shape[0], n\_samples)

# retrieve selected images

X = dataset[ix]

# generate 'real' class labels (1)

y = ones((n\_samples, 1))

return X, y

def upload():

text.delete('1.0', END)

global filename

global X, Y

filename = filedialog.askdirectory(initialdir=".")

text.delete('1.0', END)

text.insert(END,filename+" loaded\n\n")

if os.path.exists("model/X.npy"):

X = np.load("model/X.npy")

Y = np.load("model/Y.npy")

else:

X = []

Y = []

desc\_file = "raw\_2.0.jsonl"

for line in open(desc\_file, 'r'):

if len(X) < 10000:

data = json.loads(line)

fileName = data['filename']

description = data['description']

if os.path.exists("Dataset/img\_align\_celeba/"+fileName):

img = cv2.imread("Dataset/img\_align\_celeba/"+fileName)

img = cv2.resize(img, (128,128))

data = re.sub('[^A-Za-z]+', ' ',description.lower().strip())

X.append(data)

Y.append(img)

print(data+" "+str(len(X))+" "+str(img.shape))

else:

break

vectorizer = TfidfVectorizer(use\_idf=True, smooth\_idf=False, norm=None, decode\_error='replace')

X = vectorizer.fit\_transform(X).toarray()

with open('model/vector.txt', 'wb') as file:

pickle.dump(vectorizer, file)

file.close()

X = np.asarray(X)

Y = np.asarray(Y)

np.save('model/X',X)

np.save('model/Y',Y)

text.insert(END,"Total images found in dataset : "+str(Y.shape[0])+"\n\n")

text.insert(END,"Total descriptions found in dataset : "+str(X.shape[0])+"\n\n")

def generate\_latent\_points(latent\_dim, n\_samples):

x\_input = randn(latent\_dim \* n\_samples)

x\_input = x\_input.reshape(n\_samples, latent\_dim)

print(x\_input.shape)

return x\_input

# use the generator to generate n fake examples, with class labels

def generate\_fake\_samples(g\_model, latent\_dim, n\_samples):

# generate points in latent space

x\_input = generate\_latent\_points(latent\_dim, n\_samples)

# predict outputs

XX = g\_model.predict(x\_input)

# create 'fake' class labels (0)

y = zeros((n\_samples, 1))

return XX, y

# evaluate the discriminator, plot generated images, save generator model

def summarize\_performance(epoch, g\_model, d\_model, dataset, latent\_dim, n\_samples=150):

# prepare real samples

X\_real, y\_real = generate\_real\_samples(dataset, n\_samples)

# evaluate discriminator on real examples

\_, acc\_real = d\_model.evaluate(X\_real, y\_real, verbose=0)

# prepare fake examples

x\_fake, y\_fake = generate\_fake\_samples(g\_model, latent\_dim, n\_samples)

# evaluate discriminator on fake examples

\_, acc\_fake = d\_model.evaluate(x\_fake, y\_fake, verbose=0)

# summarize discriminator performance

print('>Accuracy real: %.0f%%, fake: %.0f%%' % (acc\_real\*100, acc\_fake\*100))

# save the generator model tile file

filename = 'model/generator\_model\_%03d.h5' % (epoch+1)

g\_model.save(filename)

# train the generator and discriminator

def train(g\_model, d\_model, gan\_model, dataset, latent\_dim, n\_epochs=100, n\_batch=128):

bat\_per\_epo = int(dataset.shape[0] / n\_batch)

half\_batch = int(n\_batch / 2)

# manually enumerate epochs

for i in range(n\_epochs):

# enumerate batches over the training set

for j in range(bat\_per\_epo):

# get randomly selected 'real' samples

X\_real, y\_real = generate\_real\_samples(dataset, half\_batch)

# update discriminator model weights

d\_loss1, \_ = d\_model.train\_on\_batch(X\_real, y\_real)

# generate 'fake' examples

X\_fake, y\_fake = generate\_fake\_samples(g\_model, latent\_dim, half\_batch)

# update discriminator model weights

d\_loss2, \_ = d\_model.train\_on\_batch(X\_fake, y\_fake)

# prepare points in latent space as input for the generator

X\_gan = generate\_latent\_points(latent\_dim, n\_batch)

# create inverted labels for the fake samples

y\_gan = ones((n\_batch, 1))

# update the generator via the discriminator's error

g\_loss = gan\_model.train\_on\_batch(X\_gan, y\_gan)

# summarize loss on this batch

print('>%d, %d/%d, d1=%.3f, d2=%.3f g=%.3f' %(i+1, j+1, bat\_per\_epo, d\_loss1, d\_loss2, g\_loss))

# evaluate the model performance, sometimes

if (i+1) % 10 == 0:

summarize\_performance(i, g\_model, d\_model, dataset, latent\_dim)

def ganModel():

global gan\_model

text.delete('1.0', END)

if os.path.exists('model/generator\_model\_001.h5'):

gan\_model = load\_model('model/generator\_model\_001.h5')

latent\_points = generate\_latent\_points(200, 200)

X = gan\_model.predict(latent\_points)

text.insert(END,'Fully-GAN model generated\n')

text.insert(END,'GAN generated latent generated points size : '+str(X.shape)+"\n\n")

else:

# size of the latent space

latent\_dim = 200

# create the discriminator

d\_model = define\_discriminator()

# create the generator

g\_model = define\_generator(latent\_dim)

# create the gan

gan\_model = define\_gan(g\_model, d\_model)

# load image data

dataset = load\_real\_samples()

XX = []

for i in range(len(dataset)):

img = dataset[i]

img = cv2.resize(img, (32,32))

XX.append(img)

XX = np.asarray(XX)

# train model

train(g\_model, d\_model, gan\_model, XX, latent\_dim)

def generateImage(data, features):

global X, Y

latent\_points = generate\_latent\_points(200, 200) #making array of 200 to ask GAN to generate 200 images from train model

predict = gan\_model.predict(latent\_points) #calling GAN predict model with 200 array size to generate image

predict = Y

return predict, X

def textToImage():

text.delete('1.0', END)

with open('model/vector.txt', 'rb') as file:

tfidf = pickle.load(file)

file.close()

data = tf1.get()

data = re.sub('[^A-Za-z]+', ' ',data)

embed = tfidf.transform([data]).toarray()

embed = embed.ravel()

print(embed.shape)

data = embed[0:1083]

data = data.reshape(19,19,3)

data = cv2.resize(data, (32,32))

generated\_image, X = generateImage(data,data)

max\_accuracy = 0

index = 0

for i in range(len(X)):

predict\_score = dot(X[i], embed)/(norm(X[i])\*norm(embed))

if predict\_score > max\_accuracy:

max\_accuracy = predict\_score

index = i

if max\_accuracy >= 0.45:

text.insert(END,"Input Text: "+tf1.get()+"\n\n")

text.insert(END,"Prediction Accuracy: "+str(max\_accuracy))

text.update\_idletasks()

predict = generated\_image[index]

predict = cv2.resize(predict,(250,250))

cv2.imshow("Generated Image",predict)

cv2.waitKey(0)

else:

text.insert(END,"Unable to predict face from given sentence")

def close():

main.destroy()

font = ('times', 16, 'bold')

title = Label(main, text='A Realistic Image Generation of Face From Text Description Using the Fully Trained Generative Adversarial Networks')

title.config(bg='LightGoldenrod1', fg='medium orchid')

title.config(font=font)

title.config(height=3, width=120)

title.place(x=0,y=5)

font1 = ('times', 12, 'bold')

text=Text(main,height=20,width=100)

scroll=Scrollbar(text)

text.configure(yscrollcommand=scroll.set)

text.place(x=520,y=100)

text.config(font=font1)

font1 = ('times', 13, 'bold')

uploadButton = Button(main, text="Upload CelebA Dataset", command=upload)

uploadButton.place(x=50,y=100)

uploadButton.config(font=font1)

ganButton = Button(main, text="Generate Fully Trained GAN Model", command=ganModel)

ganButton.place(x=50,y=150)

ganButton.config(font=font1)

l1 = Label(main, text='Enter Text Here')

l1.config(font=font1)

l1.place(x=50,y=200)

tf1 = Entry(main,width=50)

tf1.config(font=font1)

tf1.place(x=50,y=250)

encoderButton = Button(main, text="Generate Face from Text", command=textToImage)

encoderButton.place(x=50,y=300)

encoderButton.config(font=font1)

predictButton = Button(main, text="Exit", command=close)

predictButton.place(x=50,y=350)

predictButton.config(font=font1)

main.config(bg='OliveDrab2')

main.mainloop()

**6. TESTING:**

**Implementation and Testing:**

Implementation is one of the most important tasks in project is the phase in which one has to be cautions because all the efforts undertaken during the project will be very interactive. Implementation is the most crucial stage in achieving successful system and giving the users confidence that the new system is workable and effective. Each program is tested individually at the time of development using the sample data and has verified that these programs link together in the way specified in the program specification. The computer system and its environment are tested to the satisfaction of the user.

## Implementation

## The implementation phase is less creative than system design. It is primarily concerned with user training, and file conversion. The system may be requiring extensive user training. The initial parameters of the system should be modifies as a result of a programming. A simple operating procedure is provided so that the user can understand the different functions clearly and quickly. The different reports can be obtained either on the inkjet or dot matrix printer, which is available at the disposal of the user. The proposed system is very easy to implement. In general implementation is used to mean the process of converting a new or revised system design into an operational one.

## Testing

Testing is the process where the test data is prepared and is used for testing the modules individually and later the validation given for the fields. Then the system testing takes place which makes sure that all components of the system property functions as a unit. The test data should be chosen such that it passed through all possible condition. Actually testing is the state of implementation which aimed at ensuring that the system works accurately and efficiently before the actual operation commence. The following is the description of the testing strategies, which were carried out during the testing period.

### System Testing

Testing has become an integral part of any system or project especially in the field of information technology. The importance of testing is a method of justifying, if one is ready to move further, be it to be check if one is capable to with stand the rigors of a particular situation cannot be underplayed and that is why testing before development is so critical. When the software is developed before it is given to user to use the software must be tested whether it is solving the purpose for which it is developed. This testing involves various types through which one can ensure the software is reliable. The program was tested logically and pattern of execution of the program for a set of data are repeated. Thus the code was exhaustively checked for all possible correct data and the outcomes were also checked.

**Module Testing**

To locate errors, each module is tested individually. This enables us to detect error and correct it without affecting any other modules. Whenever the program is not satisfying the required function, it must be corrected to get the required result. Thus all the modules are individually tested from bottom up starting with the smallest and lowest modules and proceeding to the next level. Each module in the system is tested separately. For example the job classification module is tested separately. This module is tested with different job and its approximate execution time and the result of the test is compared with the results that are prepared manually. The comparison shows that the results proposed system works efficiently than the existing system. Each module in the system is tested separately. In this system the resource classification and job scheduling modules are tested separately and their corresponding results are obtained which reduces the process waiting time.

**Integration Testing**

After the module testing, the integration testing is applied. When linking the modules there may be chance for errors to occur, these errors are corrected by using this testing. In this system all modules are connected and tested. The testing results are very correct. Thus the mapping of jobs with resources is done correctly by the system.

**Acceptance Testing**

When that user fined no major problems with its accuracy, the system passers through a final acceptance test. This test confirms that the system needs the original goals, objectives and requirements established during analysis without actual execution which elimination wastage of time and money acceptance tests on the shoulders of users and management, it is finally acceptable and ready for the operation.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case Id** | **Test Case Name** | **Test Case Desc.** | **Test Steps** | | | | **Test Case Status** | **Test Priority** |
| **Step** | **Expected** | | **Actual** |
| 01 | Upload CelebA Dataset | Verify Upload CelebA Dataset or not | If CelebA Dataset may not upload | we cannot do any further operations | we can do further operations | | High | High |
| 02 | Generate Fully Trained GAN Model | Verify Generate Fully Trained GAN Model or not | If Fully Trained GAN Model may not Generate | we cannot do any further operations | we can do further operations | | High | High |
| 03 | Generate Image from Text | Verify Generate Image from Text or not | If Image from Text not be Generate | we cannot do any further operations | we can do further operations | | High | High |

**7. SCREENSHOTS:**

A Realistic Image Generation of Face From Text Description Using the Fully Trained Generative Adversarial Networks

In this paper author is using fully trained GAN (generative adversarial networks) algorithm to convert TEXT to Face. GAN algorithms is a deep learning algorithm which was introduced to generate Fake images by getting trained on sample images. GAN can be used to train on sample images and after training it can predict related fake images from given input images.

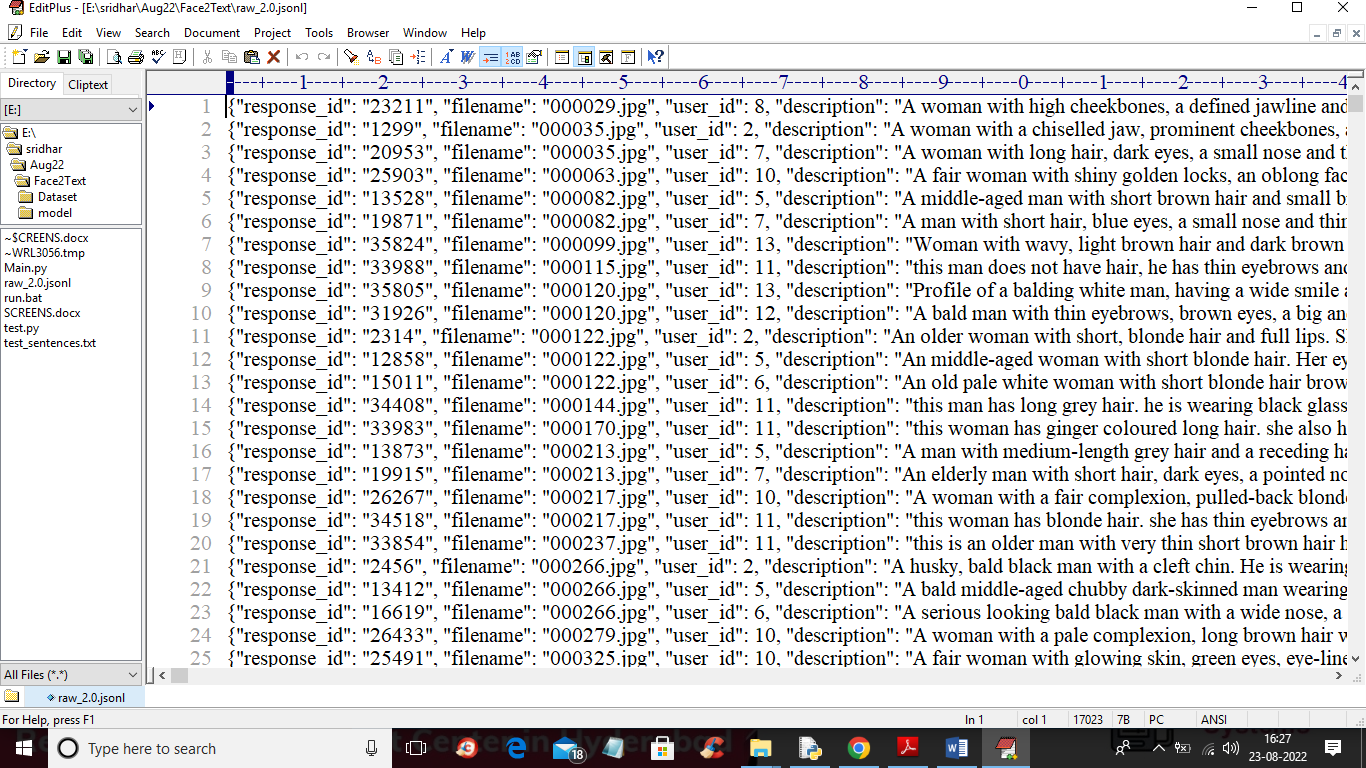
Later GAN was used to convert TEXT to FACES but this existing GAN model was training separately with images and pre-trained TEXT encoder and due to separate training, face generated from pre-trained TEXT model was not accurate and to overcome from this problem, author of this paper is simultaneously training TEXT encoder and image decoder.

GAN algorithm was used to decode images and BI-LSTM (Bidirectional-Long Short Term Memory) was used to encode TEXT. Both models will be combine to decode image based on input text sentences. BI-LSTM will map each input sentence to related image and then get trained to generate image decoding model. Input TEXT will be converted into VECTOR and this vector will be input to BI-LSTM model to encode vector and this vector will be input to GAN to decode image.

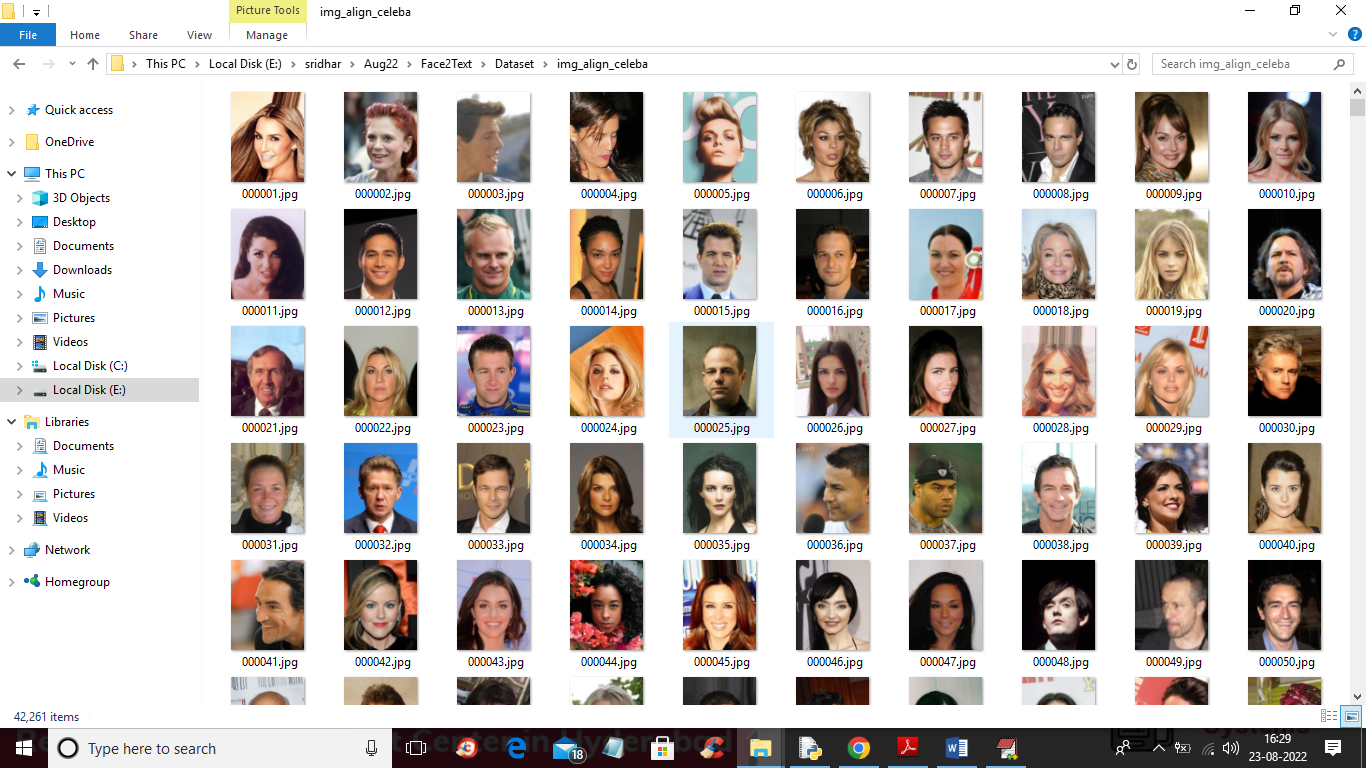
In this paper author propose a fullytrained generative adversarial network to generate realistic and natural images. The proposed work trainedthe text encoder as well as the image decoder at the same time to generate more accurate and efficient results.

To trained GAN and BI-LSTM model author has used CELEBA dataset which consists of DESCRIPTION and IMAGES. GAN get trained on IMAGES and BI-LSTM get trained on DESCRIPTION.

Below screen showing description file which is available in JSON format

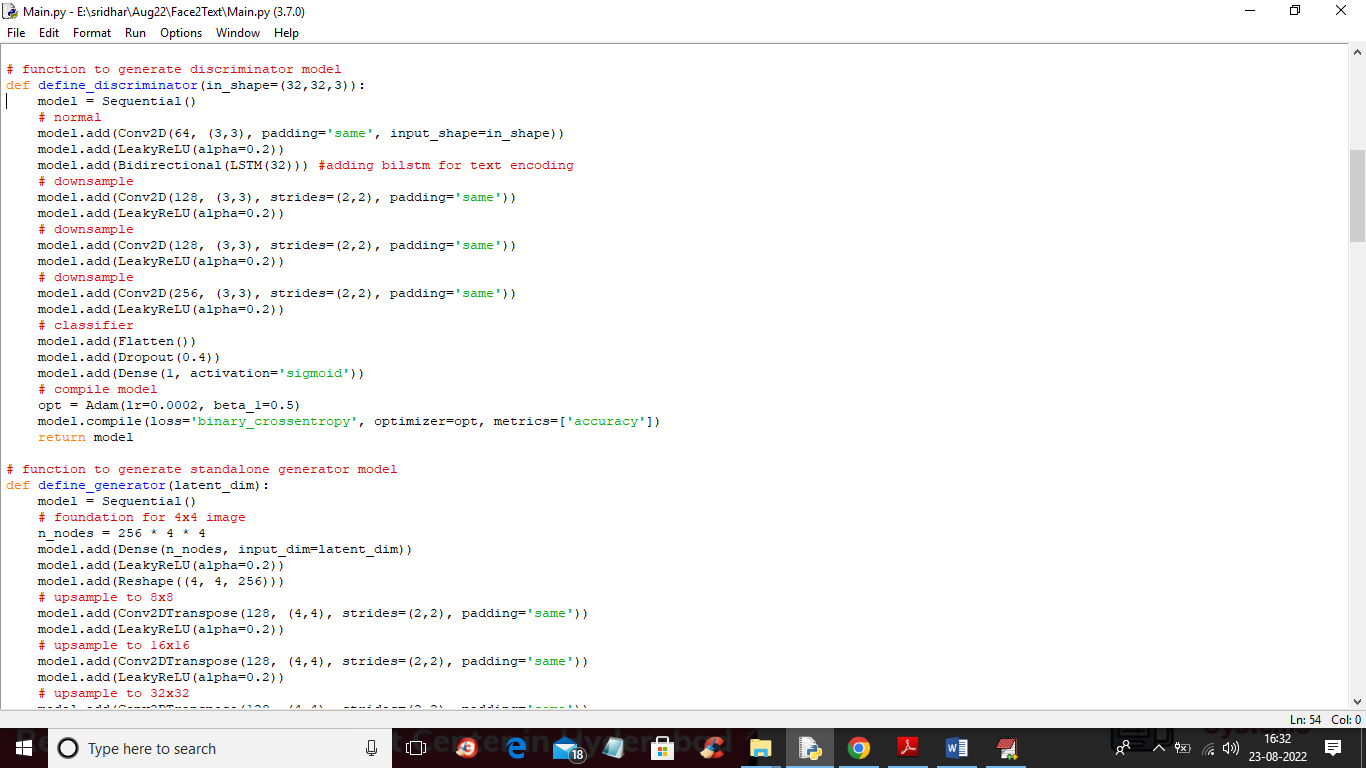


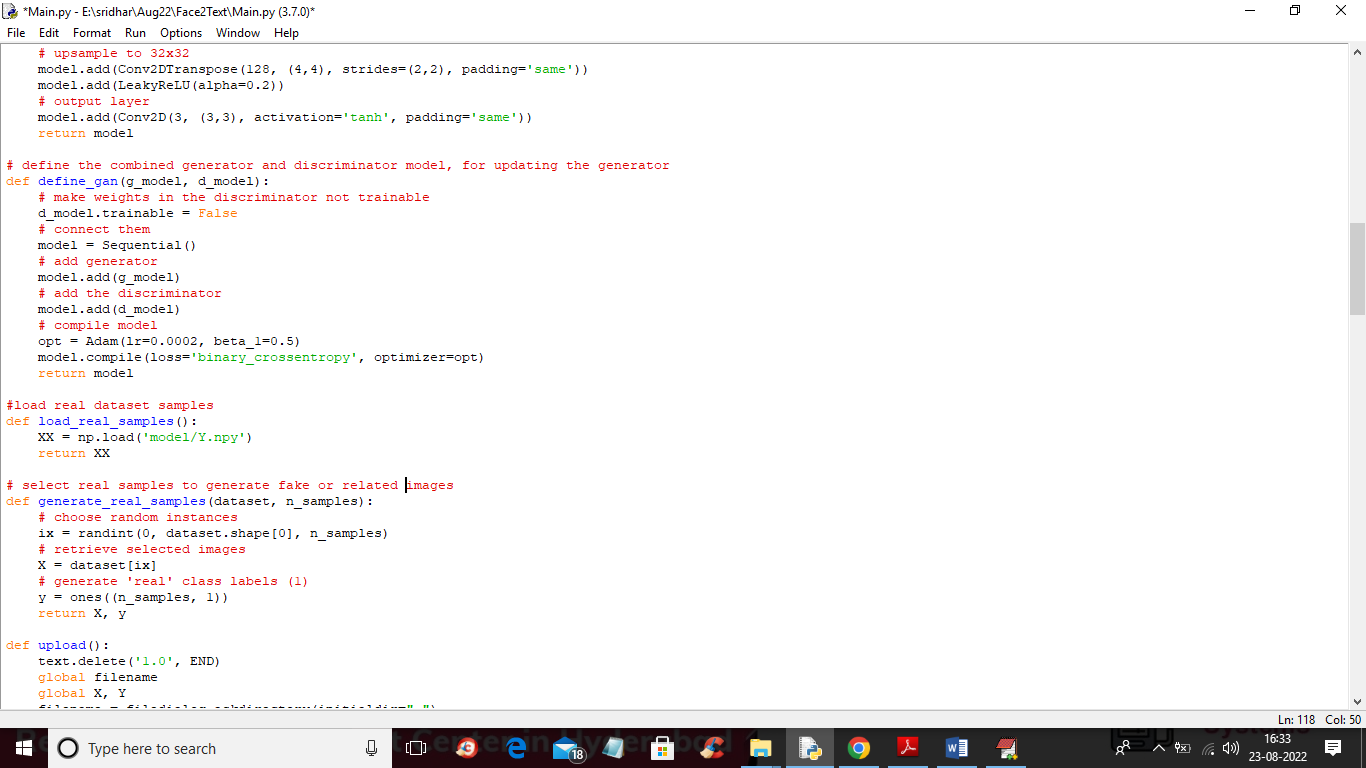
In above JSON file we can see IMAGE NAME and Description and this images we can find inside ‘Dataset/img\_align\_celeba’ folder and I am showing those images in below screen.



So by using above description and images we will train GAN and BI-LSTM model. GAN model consists of Discriminator which will take REAL dataset images and Generator will generate fake or alternate images by taking trained features from Discriminator.

In below screen I am showing code with comments for Generator and discriminator





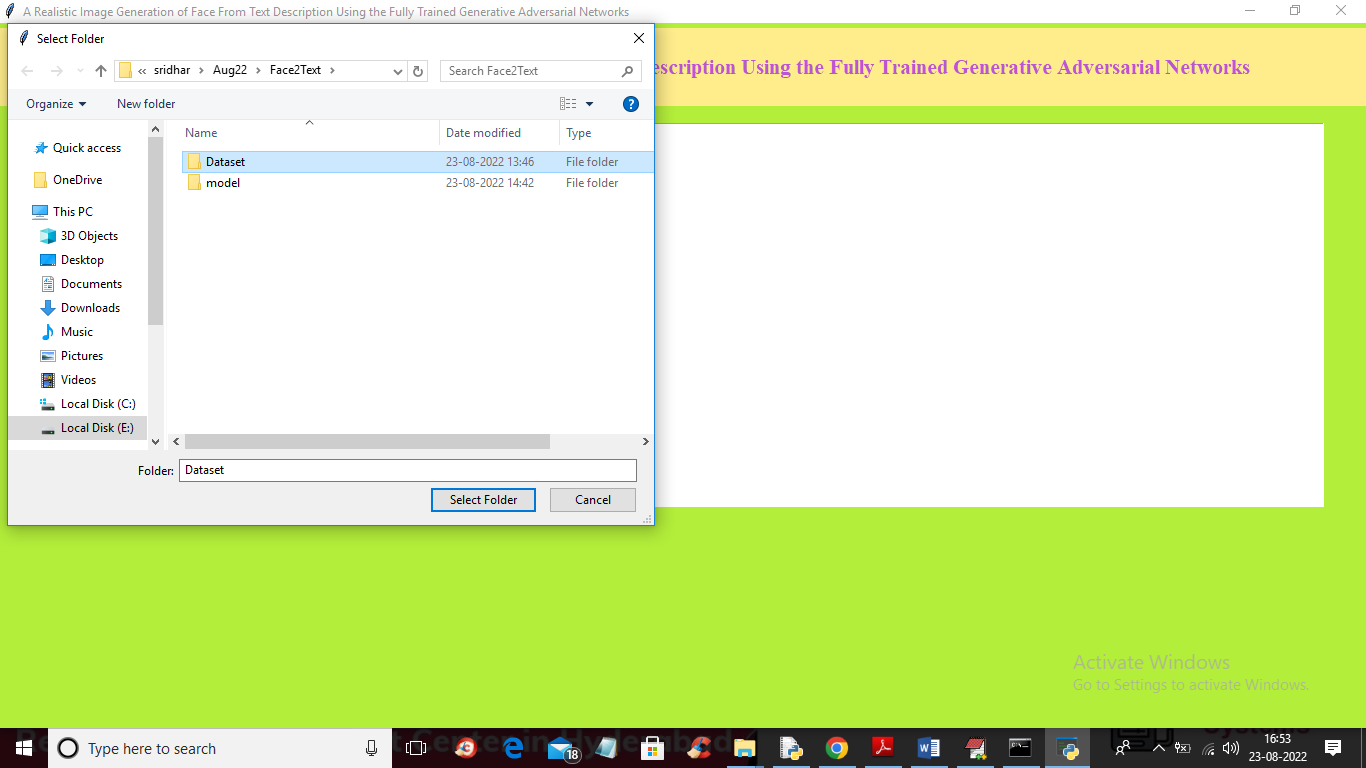
In above screen read red colour comments to know about GAN training with Discriminator, Generator and BI-LSTM.

SCREEN SHOTS

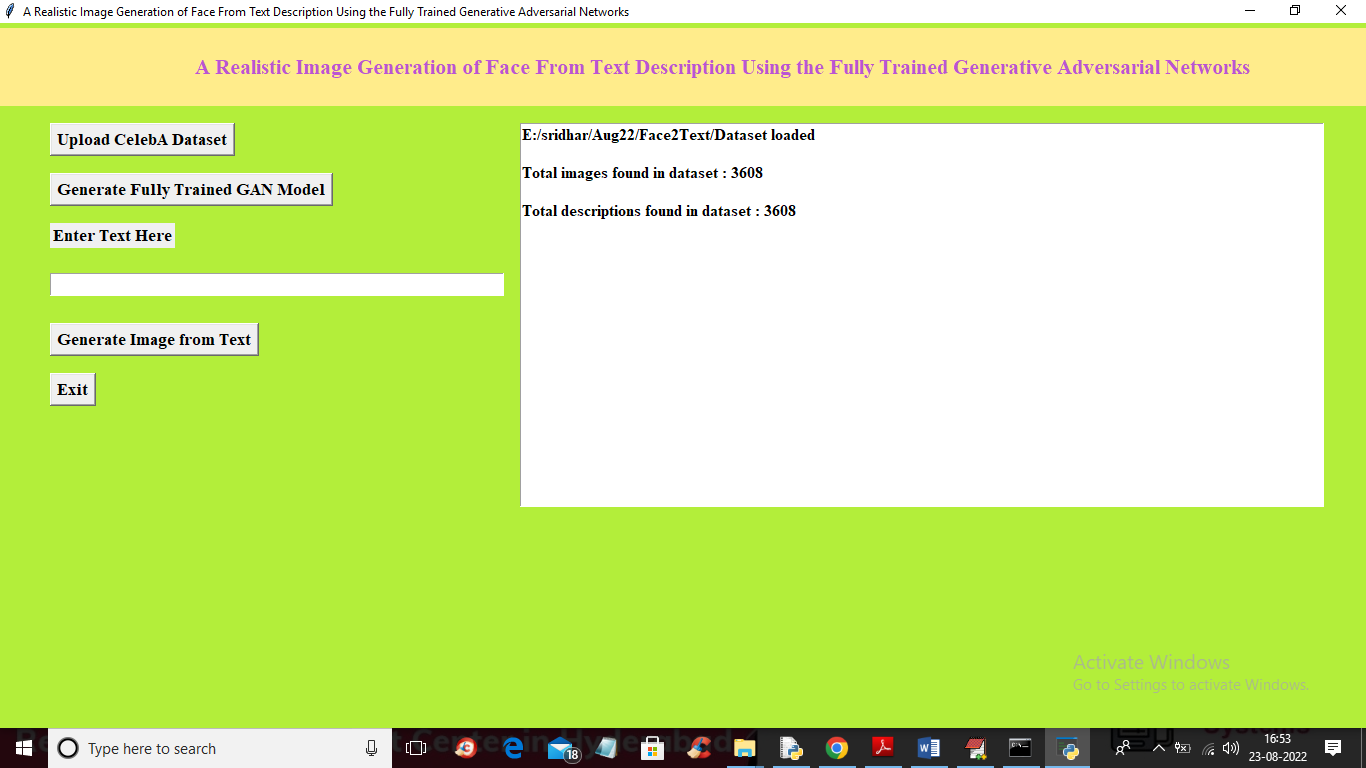
To run project double click on ‘run.bat’ file to get below screen



In above screen click on ‘Upload CelebA Dataset’ button to upload dataset and get below screen



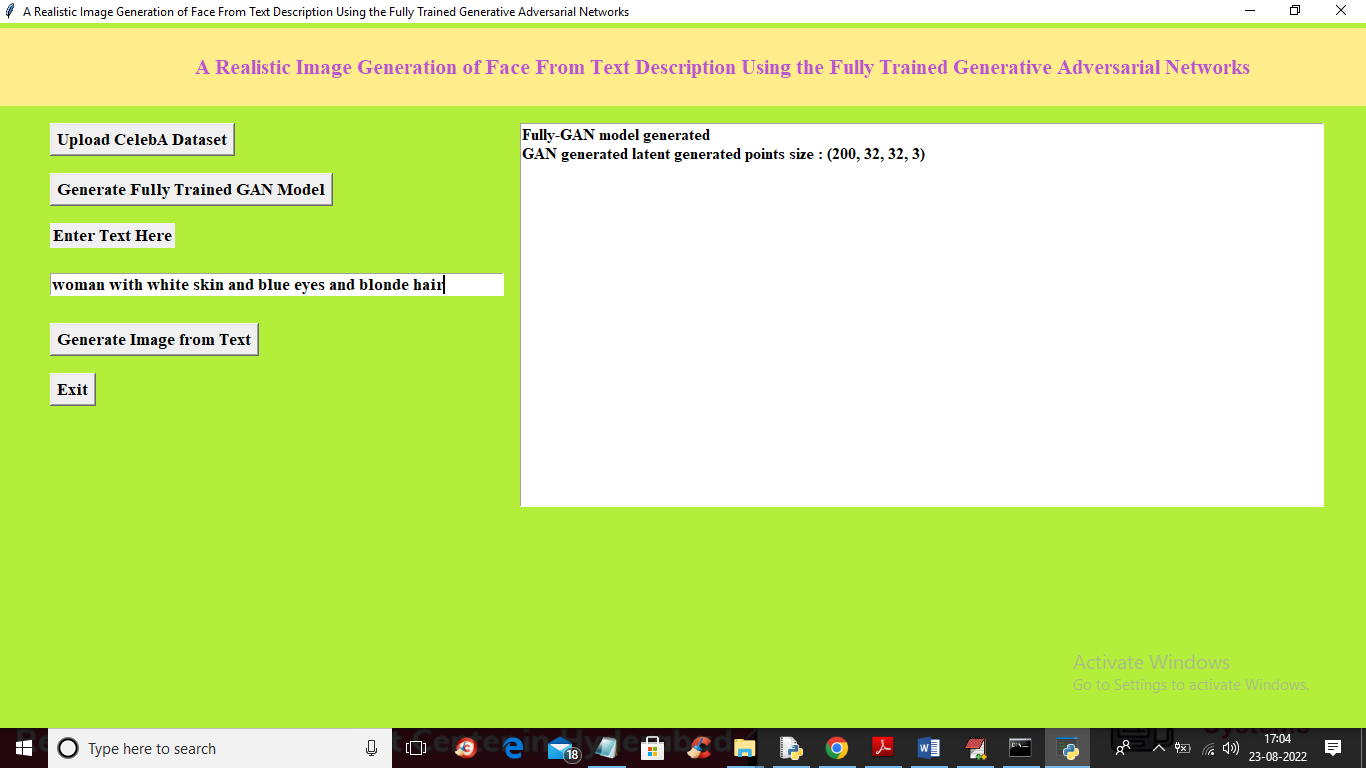
In above screen selecting and uploading ‘Dataset’ folder and then click on ‘Select Folder’ button to load dataset and get below output



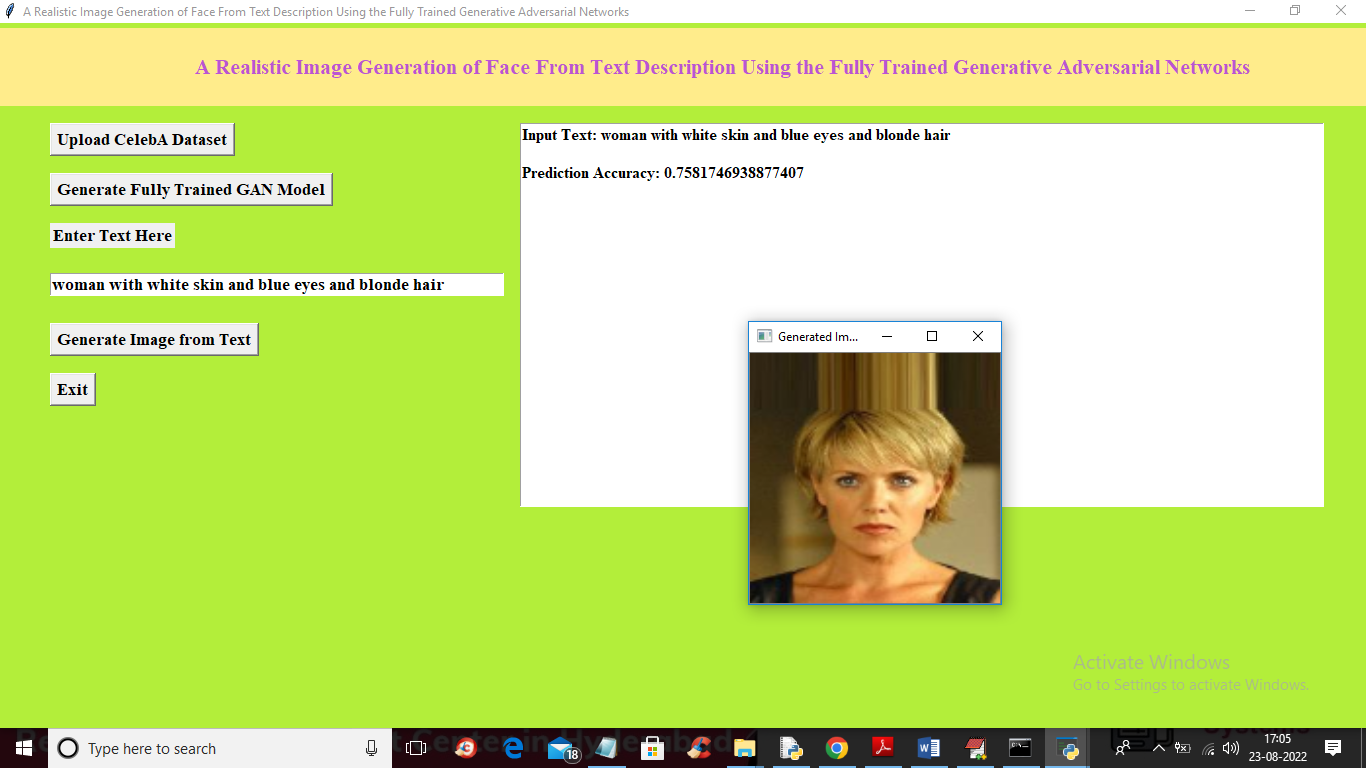
In above screen dataset loaded and application read 3608 images and description from dataset and now click on ‘Generate Fully Trained GAN Model’ button to train GAN with above images and description to get below output



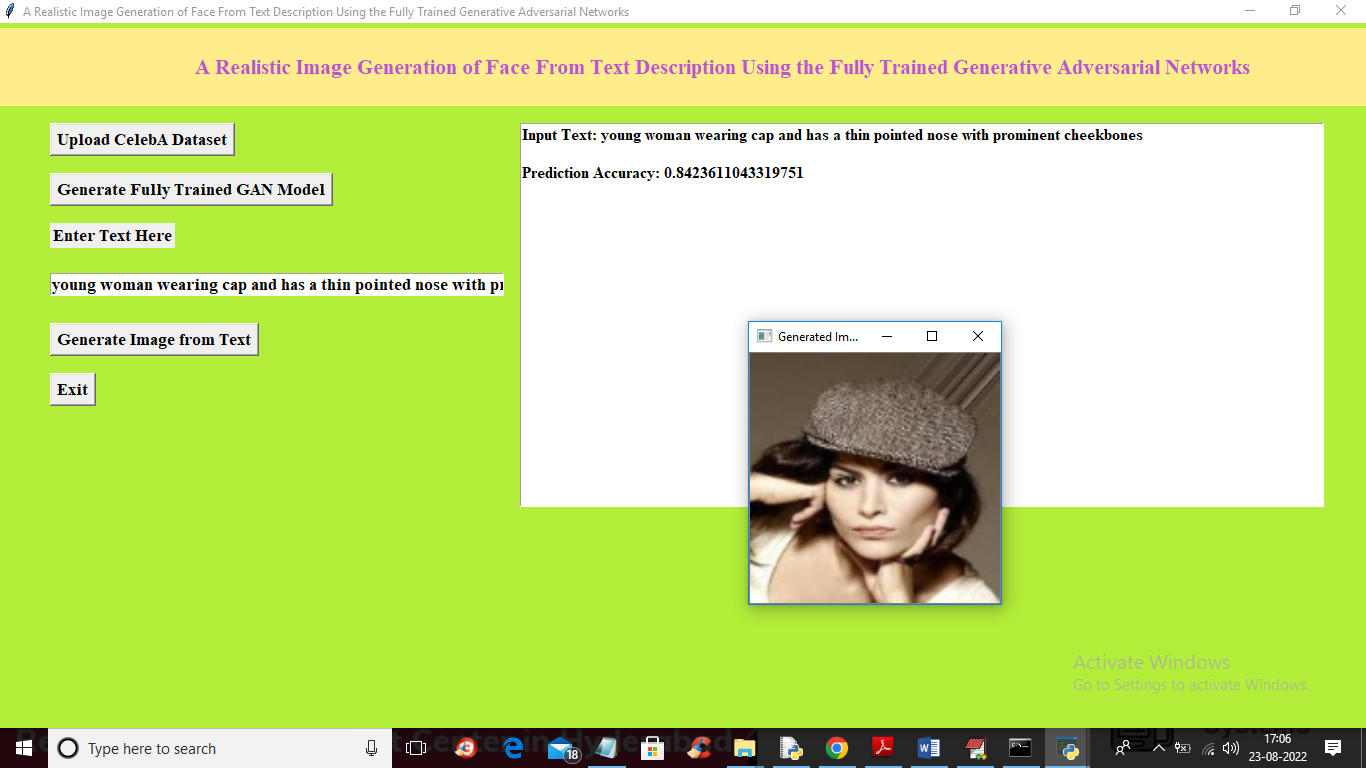
In above screen GAN model generated and this model can generated 200 faces with 32 X 32 dimension with 3 as RGB colour images from TEXT sentences and then extract face with maximum similarity from input encoded text. Now enter some TEXT inside text box and then click on ‘Generate Image from Text’ button to get below output



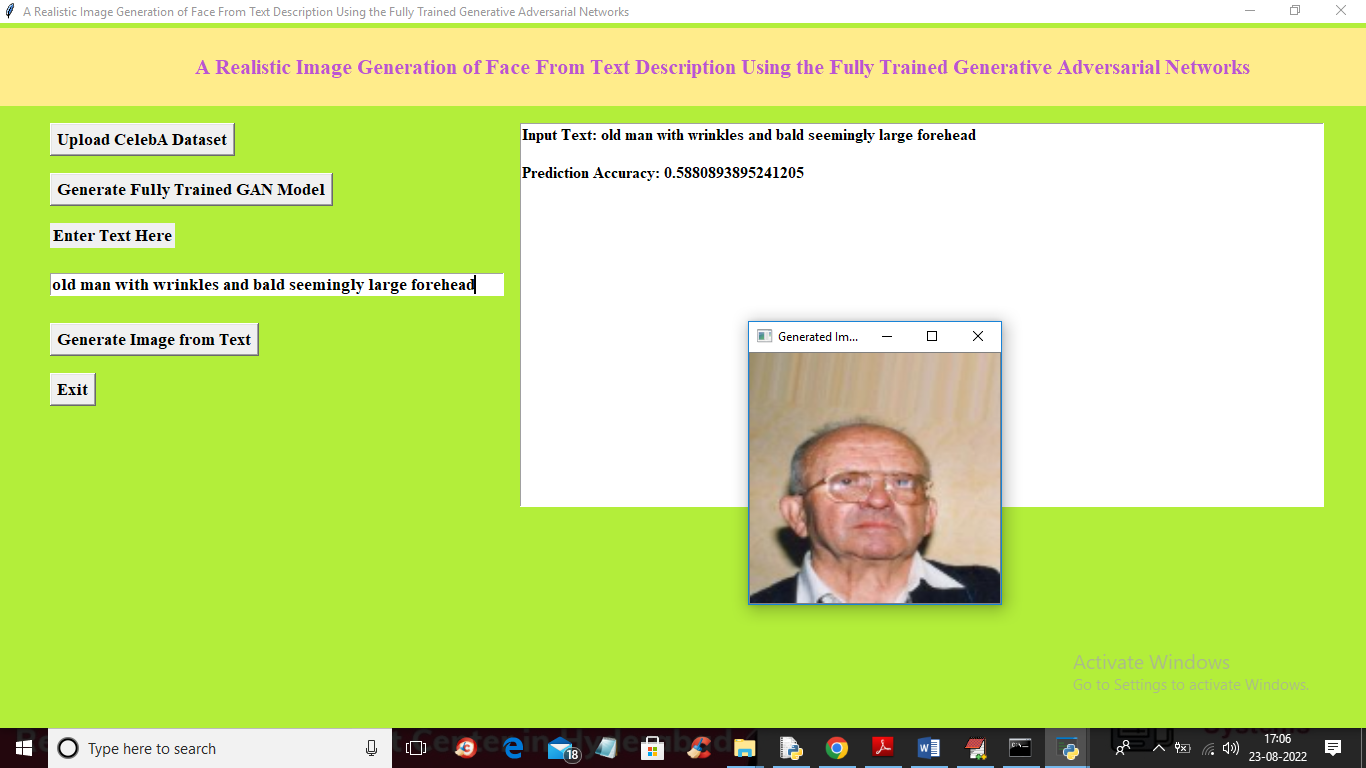
In above screen I entered some message in the TEXT BOX and then click on ‘Generate Image from Text’ button to get below output



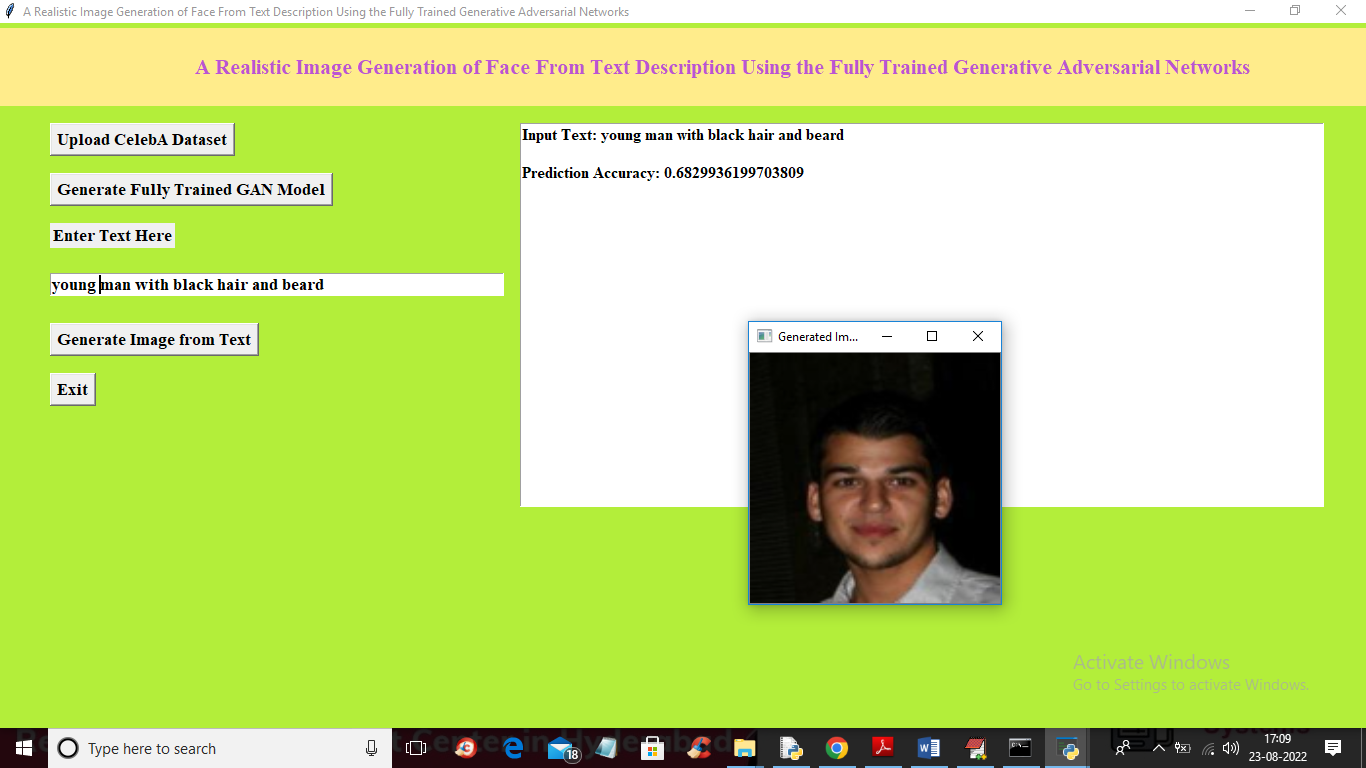
In above screen we can see face generated from given text and now try another TEXT sentences



In above screen you can read sentence from TEXT AREA and then can see generated image



In above screen you can see given TEXT sentence and generated image and we can see prediction accuracy also. This accuracy refers to MAX matching similarity between GAN predicted image and input TEXT sentence vector.



Similarly you can input any text to generate faces

**8. CONCLUSION:**

In this paper, we have proposed the fully trained generative adversarial network for text to face image synthesis. The study presents network, that trained both text encoder and image decoder for generating good quality images relative to the input sentences. By performing extensive experiments on the publicly available dataset, the superiority of our proposed methodology is proved. Moreover, in this novel task, we have also contributed towards the text to face generation dataset. Different publically available dataset along with the locally generated images have been combined. After that manual labelling of each image with defined categories have been performed. The study also presents the details of similarity between the generated faces and the ground-truth input description sentences. Experiments have shown that our proposed generative adversarial network generates natural images with good quality along with the similar face compared to the ground truth labels and faces. We compared proposed method with state of the art methods using FID and FSD scores. Proposed model achieved FSD score of 1.118 and FID score of 42.62 that is comparatively less than other benchmark algorithms. Additionally, human ratings for our generated images are also plausible. In future, to further improve the quality of images and to increase to similarity between the description and the generated faces, we have will focus on denser and precise information related to face for the proposed architecture. This proposed work has a huge impact on security related domains like forensic analysis and public safety domain etc.

**9. REFERENCES:**

1. I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in Advances in neural information processing systems, pp. 2672–2680, 2014.

2. S. Hong, D. Yang, J. Choi, and H. Lee, “Inferring semantic layout for hierarchical text-to-image synthesis,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 7986– 7994, 2018.

3. A. Van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, et al., “Conditional image generation with pixelcnn decoders,” in Advances in neural information processing systems, pp. 4790–4798, 2016.

4. H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. N. Metaxas, “Stackgan++: Realistic image synthesis with stacked generative adversarial networks,” IEEE transactions on pattern analysis and machine intelligence, vol. 41, no. 8, pp. 1947–1962, 2018.

5. C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, “The caltech-ucsd birds-200-2011 dataset,” 2011.

6. M.-E. Nilsback and A. Zisserman, “Automated flower classification over a large number of classes,” in 2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing, pp. 722–729, IEEE, 2008.

7. T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Doll´ar, and C. L. Zitnick, “Microsoft coco: Common objects in context,” in European conference on computer vision, pp. 740–755, Springer, 2014.

8. D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” arXiv preprint arXiv:1312.6114, 2013.

9. S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, “Generative adversarial text to image synthesis,” arXiv preprint arXiv:1605.05396, 2016.

10. A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” arXiv preprint arXiv:1511.06434, 2015.

11. H. Dong, S. Yu, C. Wu, and Y. Guo, “Semantic image synthesis via adversarial learning,” in Proceedings of the IEEE International Conference on Computer Vision, pp. 5706–5714, 2017.

12. H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. N. Metaxas, “Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks,” in Proceedings of the IEEE international conference on computer vision, pp. 5907–5915, 2017.

13. S. E. Reed, Z. Akata, S. Mohan, S. Tenka, B. Schiele, and H. Lee, “Learning what and where to draw,” in Advances in neural information processing systems, pp. 217–225, 2016.

14. S. Sharma, D. Suhubdy, V. Michalski, S. E. Kahou, and Y. Bengio, “Chatpainter: Improving text to image generation using dialogue,” arXiv preprint arXiv:1802.08216, 2018.

15. T. Xu, P. Zhang, Q. Huang, H. Zhang, Z. Gan, X. Huang, and X. He, “Attngan: Fine-grained text to image generation with attentional generative adversarial networks,” in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1316–1324, 2018.

16. T. Qiao, J. Zhang, D. Xu, and D. Tao, “Mirrorgan: Learning texttoimage generation by redescription,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1505– 1514, 2019.

17. Z. Zhang, Y. Xie, and L. Yang, “Photographic text-to-image synthesis with a hierarchically-nested adversarial network,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 6199–6208, 2018.

18. A. Gatt, M. Tanti, A. Muscat, P. Paggio, R. A. Farrugia, C. Borg, K. P. Camilleri, M. Rosner, and L. Van der Plas, “Face2text: collecting an annotated image description corpus for the generation of rich face descriptions,” arXiv preprint arXiv:1803.03827, 2018.

19. Y. Guo, L. Zhang, Y. Hu, X. He, and J. Gao, “Ms-celeb-1m: A dataset and benchmark for large-scale face recognition,” in European conference on computer vision, pp. 87–102, Springer, 2016.

20. G. B. Huang, M. Mattar, T. Berg, and E. Learned-Miller, “Labeled faces in the wild: A database forstudying face recognition in unconstrained environments,” 2008.

21. Z. Liu, P. Luo, X. Wang, and X. Tang, “Deep learning face attributes in the wild,” in Proceedings of the IEEE international conference on computer vision, pp. 3730–3738, 2015.

22. M. Mirza and S. Osindero, “Conditional generative adversarial nets,” arXiv preprint arXiv:1411.1784, 2014.