TABLE OF CONTENTS

CHAPTER **Page No.**

ACKNOWLEDGEMENT

We wish to express our profound gratitude to our respected Principal **Dr. B. GIRIRAJ**, P.S.G Polytechnic College for providing the opportunity and necessary facilities in carrying out this project.

We express our sincere thanks and deep sense of gratitude to our Head of the department **Dr. S. BRINDHA**, for her encouragement and support that she extended towards dissertation work.

We profusely thank our guide **Mrs. T. P. KAMATCHI**, for her constant support and encouragement throughout the course of the project. We are grateful to have inspiring guidance and invaluable suggestions which motivated us right from the inception of the project.

We take immense pleasure in conveying our thanks to teaching and non teaching staffs without whom this critique work would not have been completed successfully.

PHOTOREALISTIC IMAGE SYNTHESIS FROM TEXT DESCRIPTION USING MACHINE LEARNING

Photorealistic image synthesis has been difficult till date with the existing technologies however Artificial Intelligence makes this possible, we can benefit a lot from the wide application of this emerging technology. It can be employed to replace human labors in completing many tedious tasks. The objective of this project is to generate photorealistic images from a text description given by a user of a specific thing, object or being using machine learning. This project can be used by designers, engineers to generate designs based on concepts that they've thought.

We propose a system using a Generative Adversarial Network (GAN) which is a machine learning model which consists of a Generator and a Discriminator both of which are trained with the same dataset, the Generator is used to generate fake images based of the real images from the dataset that's used to train it. The Discriminator classifies the generated image as fake or real, when the Generator generates an image so compelling and realistic the Discriminator classifies it as real. This project is built using a stack GAN which consists of two GANs, the first GAN is called a stage 1 GAN which takes the sentences which are represented as word vectors and generates an image with primitive shapes and basic colors, it is a low resolution image, the second GAN is a stage 2 GAN which takes the image generated by the stage 1 GAN and the original text description as the input and generates a much higher resolution version of the image by completing the details.

This project will be developed in Python and Tensorflow, an open source machine learning library. The project will be supported on all platforms that support Python and the required dependencies. This project requires a system with a powerful 64 bit multicore processor.

LIST OF FIGURES

LIST OF ABBREVIATIONS

Acronyms	Description
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
GAN	Generative Adversarial Network
sGAN	Stacked Generative Adversarial Network
ESGAN	Enhanced Stacked Generative Adversarial Network
CUB	Caltech-UCSD Birds
tf	Tensorflow
pd	Pandas
np	Numpy
VAE	Variational Autoencoder
DCGAN	Deep Convolutional Generative Adversarial Network
CPU	Central Processing Unit
GPU	Graphics Processing Unit

CHAPTER 1
INTRODUCTION

Photorealistic image synthesis is the process of generating fake images that look realistic which is so compelling that it is very hard even for humans to identify the fake images. This can be used to employ very resource intensive work and can drastically reduce the cost of building prototypes for designs as our proposed system is capable of generating photorealistic images based of the user requirement. Homo sapiens are equipped with Synesthesia, a perceptual phenomenon in which stimulation of one sensory or cognitive pathway leads to automatic, involuntary experiences in a second sensory or cognitive pathway, which makes us human beings to visualize certain objects based of their description and our knowledge about basic shapes, colors and features, our proposed system is built to replicate this process of human beings in computers using Machine Learning.

The model we have proposed is a Stack Generative Adversarial Network which is built using two Generative Adversarial Networks which explains the naming “Stack”. The model needs to be trained using huge datasets of images and their text embeddings of the kind of image that needs to be generated.

1.1 **Machine Learning**

Machine Learning is a science that involves providing the ability for computers to learn and act like humans without being explicitly programmed with an exponentially greater accuracy. ML is closely related to computational statistics, which also focuses on prediction-making through the use of computers. It has strong ties to mathematical optimization, which delivers methods, theory and application domains to the field. Machine learning is sometimes conflated with data mining, where the latter subfield focuses more on exploratory data analysis and is known as unsupervised learning. Machine learning can also be unsupervised and be used to learn and establish baseline behavioral profiles for various entities and then used to find meaningful anomalies.

Within the field of data analytics, machine learning is a method used to devise complex models and algorithms that lend themselves to prediction; in commercial use, this is known as predictive analytics. These analytical models allow researchers, data scientists, engineers, and analysts to

"produce reliable, repeatable decisions and results" and uncover "hidden insights" through learning from historical relationships and trends in the data.

1.1.1 Types of Machine Learning Models

On the whole Machine Learning models can be generally a Classifier or a Prediction model, several models can be used for both classification and prediction of values. Classification is the process of feeding in data to the classifier model and the model predicts the class of data the input data belongs to, based on the training data and a Prediction model is used to predict values given the history of values on a particular field with its important features, but the models can also vary in the type of training data provided, what they do, and how they are trained, optimized. Based on the training data the models are provided the models can be classified as:

1. **Supervised Learning:** Supervised learning is a technique where the model is given a labelled dataset for training so the training data is accurate and the model need not analyze and classify the training data which will make the time taken for training a lot more and also making it resource intensive as the training dataset is generally humongous.
2. **Unsupervised Learning:** Unsupervised learning is a technique where the model is given a non- labelled dataset for training, the model needs to analyze and classify the training data as well and uses it as a base for the input data it receives to act upon it.
3. **Semisupervised Learning:** Semisupervised learning is a technique where the model is given dataset with labelled data which is correctly and accurately labelled and also unlabelled data. The ultimate goal of these types of model is to infer the correct labels for the given unlabeled data only.
4. **Reinforcement Learning:** Reinforcement learning is a technique where the model acts based on the environment and the rewards it receives based on the actions it performs and also the negative marks for the wrong actions performed. This type of learning is the one human beings use and it can be the future of Artificial Intelligence.

The models employ neural networks which can differ on what they’ve been trained to do making Machine learning a huge field.

1.1.2 Neural Networks

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of ANNs as well. Neural networks, with their remarkable ability to derive meaning from complicated or imprecise data, can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained neural network can be thought of as an "expert" in the category of information it has been given to analyze. This expert can then be used to provide projections given new situations of interest and answer "what if" questions.

Other advantages include:

1. Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.
2. Self Organization: An ANN can create its own organization or representation of the information it receives during learning time.
3. Real Time Operation: ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.
4. Fault Tolerance via Redundant Information Coding: Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage.

1.1.3 Convolutional Neural Network

A Convolutional Neural Network (CNN) is comprised of one or more convolutional layers (often with a subsampling step) and then followed by one or more fully connected layers as in a standard multilayer neural network. The architecture of a CNN is designed to take advantage of the 2D structure of an input image (or other 2D input such as a speech signal). This is achieved with local connections and tied weights followed by some form of pooling which results in translation invariant features. Another benefit of CNNs is that they are easier to train and have many fewer parameters than fully connected

networks with the same number of hidden units. In this article we will discuss the architecture of a CNN and the back-propagation algorithm to compute the gradient with respect to the parameters of the model in order to use gradient based optimization. See the respective tutorials on convolution and pooling for more details on those specific operations. A CNN consists of a number of convolutional and subsampling layers optionally followed by fully connected layers. The input to a convolutional layer is a m*m*r image where m is the height and width of the image and r is the number of channels the convolutional layer will have k filters (or kernels) of size n*n*q where n is the smaller dimension of the image and q can either be the same as the number of channels r or smaller and may vary for each kernel. The size of the filters gives rise to the locally connected structure which are each convolved with the image to produce k feature maps of size m - n + 1. Each map is then subsampled typically with mean or max pooling over p*p contiguous regions where p ranges between 2 for small images (e.g. MNIST) and is usually not more than 5 for larger inputs. Either before or after the subsampling layer an additive bias and sigmoidal nonlinearity is applied to each feature map. The figure below illustrates a full layer in a CNN consisting of convolutional and subsampling sublayers. Units of the same color have tied weights.

1.1.4 Recurrent Neural Networks

The idea behind RNNs is to make use of sequential information. In a traditional neural network, we assume that all inputs (and outputs) are independent of each other. But for many tasks that's a very bad idea. If you want to predict the next word in a sentence you better know which words came before it. RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations. Another way to think about RNNs is that they have a "memory" which captures information about what has been calculated so far. In theory RNNs can make use of information in arbitrarily long sequences, but in practice

they are limited to looking back only a few steps (more on this later). Here is what a typical RNN looks like:

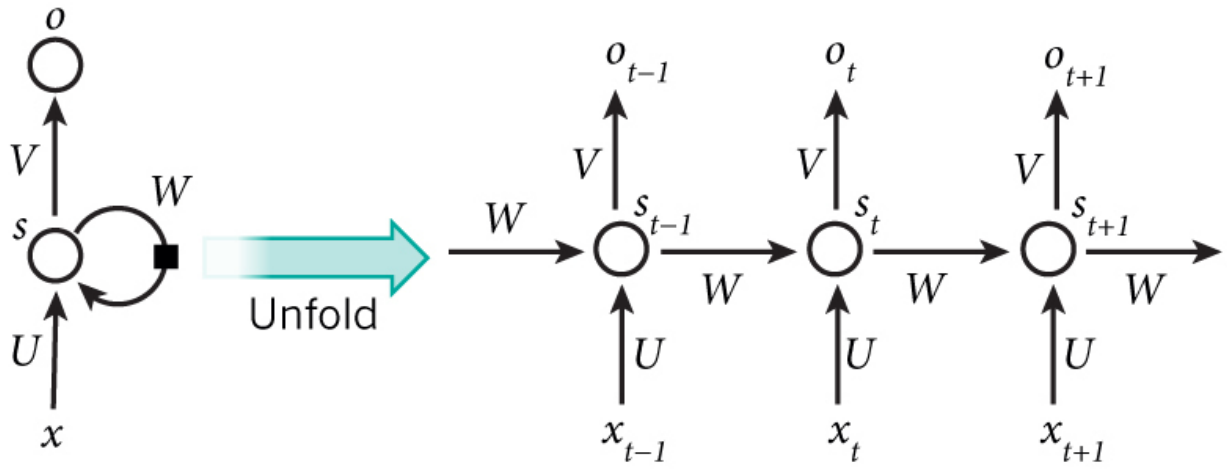


Figure 1.1 Recurrent Neural Network

The above diagram shows a RNN being *unrolled* (or unfolded) into a full network. By unrolling we simply mean that we write out the network for the complete sequence. For example, if the sequence we care about is a sentence of 5 words, the network would be unrolled into a 5-layer neural network, one layer for each word.

1.2 Digital Images

A digital image refers to the binary representation of a two-dimensional image, each pixel of the image is given a value which specifies the pixel's color and its intensity. Machine learning models analyze the image only by the raw pixels input. The most popular image classification and recognition machine learning model is the Convolutional Neural network. The image is analyzed and features of the object represented in the image are recognized and the neural network is trained on it allowing it to know what the representation actually represents rather than the traditional hard coded rules which has a high chance of failure of recognition if it is given an altered image. The model generates images by sampling the required image representation parameters according to the user requirement. The model knows what and how the parts of the image representation needs to be sampled, thanks to word vectors.

1.3 Feature Vectors

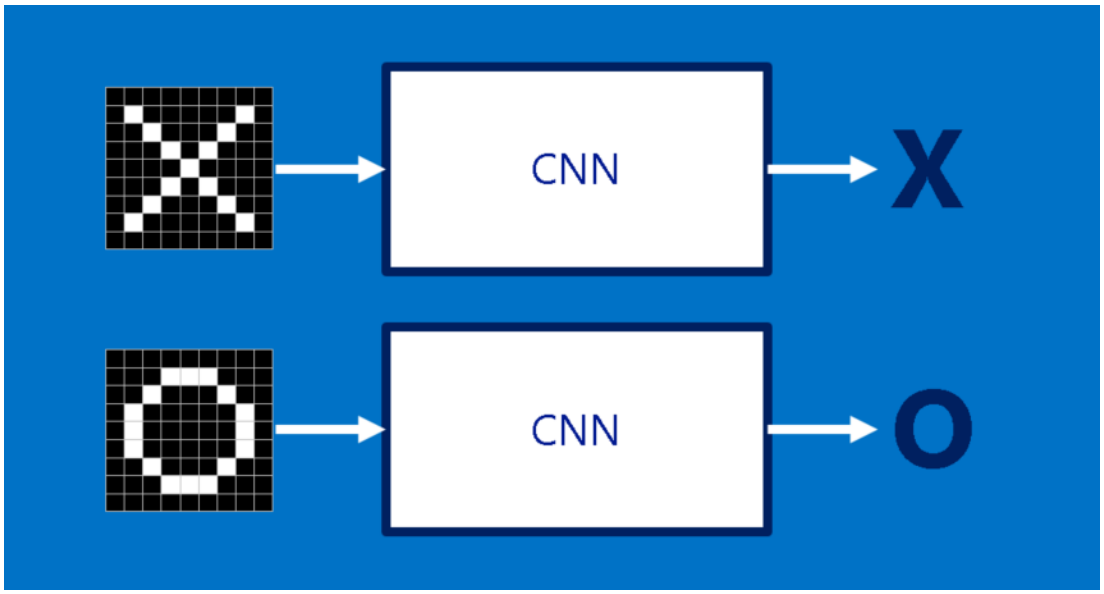


Figure 1.2 - Image Recognition

In the field of machine learning a feature vector is an n-dimensional vector of numerical features that represent some object. Many algorithms in machine learning require a numerical representation of objects, since such representations facilitate processing and statistical analysis. When representing images, the feature values might correspond to the pixels of an image, while when representing texts, the features might be the frequencies of occurrence of textual terms. Feature vectors are equivalent to the vectors of explanatory variables used in statistical procedures such as linear regression. Feature vectors are often combined with weights using a dot product in order to construct a linear predictor function that is used to determine a score for making a prediction. The vector space associated with these vectors is often called the feature space. In order to reduce the dimensionality of the feature space, a number of dimensionality reduction techniques can be employed.

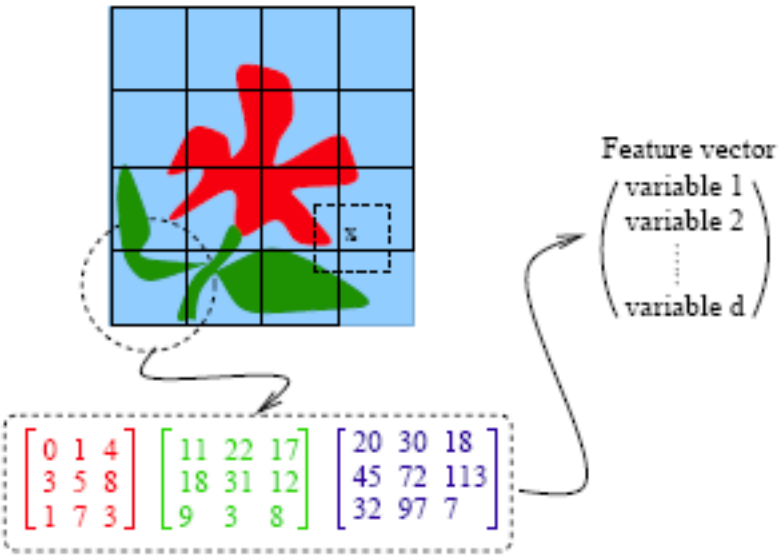


Figure 1.3 - Feature Vectors

1.4 Generative models

Generative models are a class of machine learning models that is used to generate based of the data it has been trained on. It uses the Joint Probability Distribution over the observations. Although Generative Models have short term Applications but in the long run they actually have the potential and power to automatically learn the features from a dataset, categories or dimensions or anything and generate data. Generative Adversarial Networks is a generative model as the name suggests.

1.5 Generative Adversarial Networks

Generative Adversarial Network is a machine learning generative model that we are going to use for Photorealistic image synthesis. GANs have a Generator and a Discriminator. The Generator is used to generate images based of the data gathered from training from the image datasets and its associated feature vector text embeddings and the user requirement. The discriminator is used to classify the image generated by the generator as a real image (The generated image is not from the training data) or a fake image (The generated image is from the training data). Both the generator and the discriminator are trained using the same dataset, the Generator needs to generate an image so compellingly realistic that the discriminator needs to be outsmarted and the generated image is classified as real and passed on as the output. Deep generative models have had less of an impact, due to the difficulty of approximating many intractable probabilistic computations that arise in maximum likelihood estimation and related

strategies, and due to difficulty of leveraging the benefits of piecewise linear units in the generative context where GAN is the solution. This framework can yield specific training algorithms for many kinds of model and optimization algorithm. the generative model generates samples by passing random noise through a multilayer perceptron, and the discriminative model is also a multilayer perceptron.

1.6 Stack Generative Adversarial Networks

Stack GANs include two GANs, the Stage 1 GAN and the Stage 2 GAN. The Stage 1 GAN takes the user requirement in the form of a text description and generates a low-resolution image with basic shapes and colors whereas the Stage 2 GAN takes the low resolution image generated by the Stage 1 GAN and also the text description from the user as the input and generates a much higher resolution image as the output.

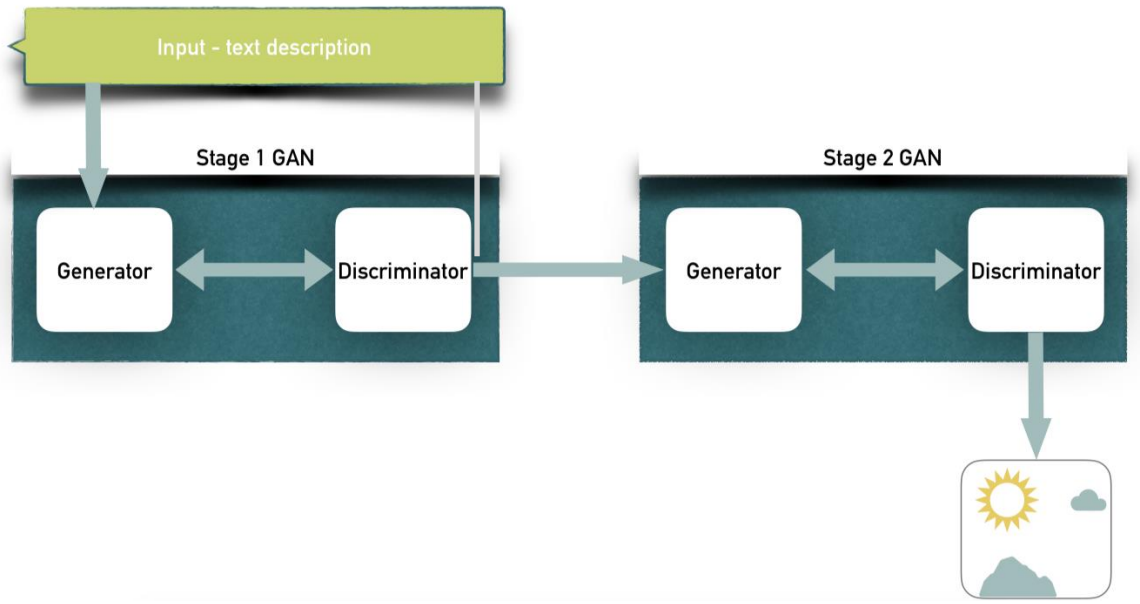


Figure 1.4 - sGAN model

1.7 Torch

Torch is a scientific computing framework with wide support for machine learning algorithms that puts GPUs first. It is easy to use and efficient, thanks to an easy and fast scripting language, LuaJIT, and an underlying C/CUDA implementation.

A summary of core features:

- a powerful N-dimensional array
- lots of routines for indexing, slicing, transposing, …
- amazing interface to C, via LuaJIT
- linear algebra routines
- neural network, and energy-based models
- numeric optimization routines
- Fast and efficient GPU support
- Embeddable, with ports to iOS and Android backends

The goal of Torch is to have maximum flexibility and speed in building your scientific algorithms while making the process extremely simple. Torch comes with a large ecosystem of community-driven packages in machine learning, computer vision, signal processing, parallel processing, image, video, audio and networking among others, and builds on top of the Lua community.

At the heart of Torch are the popular neural network and optimization libraries which are simple to use, while having maximum flexibility in implementing complex neural network topologies. You can build arbitrary graphs of neural networks, and parallelize them over CPUs and GPUs in an efficient manner.

1.8 Objective of the project

The objective of the project is to develop a sGAN machine learning model that can generate photorealistic images based upon the user requirement. The model can be trained with different datasets and can generate images only based of the training data of images and its associated text embeddings, the model can be trained with a generic dataset or different datasets. This has a very high capability and is very powerful, it can save huge amounts of money for hardware producing companies where they need not build physical prototypes to look at how the design is in flesh or designing a concept wasting hours and hours of time making it possible in a much better way in a matter of seconds. The model can also be used in a variety of fields such as fashion designing and research.

CHAPTER 2
LITERATURE SURVEY

It’s easy to forget just how much you know about the world: you understand that it is made up of 3D environments, objects that move, collide, interact; people who walk, talk, and think; animals who graze, fly, run, or bark; monitors that display information encoded in language about the weather, who won a basketball game, or what happened in 1970.

This tremendous amount of information is out there and to a large extent easily accessible — either in the physical world of atoms or the digital world of bits. The only tricky part is to develop models and algorithms that can analyze and understand this treasure trove of data.

Generative models are one of the most promising approaches towards this goal. To train a generative model we first collect a large amount of data in some domain (e.g., think millions of images, sentences, or sounds, etc.) and then train a model to generate data like it.

The trick is that the neural networks we use as generative models have a number of parameters significantly smaller than the amount of data we train them on, so the models are forced to discover and efficiently internalize the essence of the data in order to generate it.

Generative models have many short-term applications. But in the long run, they hold the potential to automatically learn the natural features of a dataset, whether categories or dimensions or something else entirely.

Generative models are a rapidly advancing area of research. As we continue to advance these models and scale up the training and the datasets, we can expect to eventually generate samples that depict entirely plausible images or videos. This may by itself find use in multiple applications, such as on-demand generated art, or Photoshop++ commands such as “make my smile wider”. Additional presently known applications include image denoising, inpainting, super-resolution, structured prediction, exploration in reinforcement learning, and neural network pretraining in cases where labeled data is expensive.

An alternative to directed graphical models with latent variables are undirected graphical models with latent variables, such as restricted Boltzmann machines (RBMs), deep Boltzmann machines (DBMs) and their numerous variants. The interactions within such models are represented as the product of unnormalized potential functions, normalized by a global summation/integration over all states of the random variables. This quantity (the partition function) and its gradient are intractable for all but the most trivial instances, although they can be estimate d by Markov chain Monte Carlo (MCMC) methods. Mixing poses a significant problem for learning algorithms that rely on MCMC.

Deep belief networks (DBNs) are hybrid models containing a single undirected layer and several directed layers. While a fast-approximate layer-wise training criterion exists, DBNs incur the computational difficulties associated with both undirected and directed models.

Alternative criteria that do not approximate or bound the log-likelihood have also been proposed, such as score matching and noise-contrastive estimation (NCE). Both of these require the learned probability density to be analytically specified up to a normalization constant. Note that in many interesting generative models with several layers of latent variables (such as DBNs and DBMs), it is not even possible to derive a tractable unnormalized probability density. Some models such as denoising auto-encoders and contractive autoencoders have learning rules very similar to score matching applied to RBMs. In NCE, as in this work, a discriminative training criterion is employed to fit a generative model. However, rather than fitting a separate discriminative model, the generative model itself is used to discriminate generated data from samples a fixed noise distribution. Because NCE uses a fixed noise distribution, learning slows dramatically after the model has learned even an approximately correct distribution over a small subset of the observed variables.

Finally, some techniques do not involve defining a probability distribution explicitly, but rather train a generative machine to draw samples from the desired distribution. This approach has the advantage that such machines can be designed to be trained by back-propagation. Prominent recent work in this area includes the generative stochastic network (GSN) framework, which extends generalized denoising auto-encoders: both can be seen as defining a parameterized Markov chain, i.e., one learns the parameters of a machine that performs one step of a generative Markov chain. Compared to GSNs, the adversarial nets framework does not require a Markov chain for sampling. Because adversarial nets do not require feedback loops during generation, they are better able to leverage piecewise linear units, which improve the performance of backpropagation but have problems with unbounded activation when used in a feedback loop. More recent examples of training a generative machine by back-propagating into it include recent

work on auto-encoding variational Bayes and stochastic backpropagation. Other related work includes:

Autoencoders is a type of machine learning model that uses unsupervised learning which means the model can be trained on unlabelled data. Autoencoders sequentially deconstruct input data into hidden representations, then use those same representations to sequentially reconstruct outputs that resemble their originals. Autoencoding is considered as a data compression algorithm, but they're data specific so it can only compress data similar to what it's been trained on.They're also relatively lossy so the decompressed outputs will be a bit degraded compared to the original inputs. It can be used for data denoising, where we train an autoencoder to reconstruct the input from a corrupted version of it, so that given some similar data that's corrupted, it can denoise it. It can also be used to generate similar but unique data. There are different autoencoder types.

Disadvantages of Autoencoders: You have to train an autoencoder. That's a lot of data, processing time, hyperparameter tuning, and model validation before you even start building the real model. An autoencoder learns to efficiently represent a manifold on which the training data lies. If your training data is not representative of your testing data, then you wind up obscuring information rather than clarifying it. An autoencoder learns to capture as much information as possible rather than as much *relevant* information as possible. That means that if the information most relevant to your problem makes up only a small (in magnitude) part of the input, the autoencoder may lose a lot of it. Using an autoencoder also destroys any interpretability your model may otherwise have had. That may or may not be important to you. Another point about the autoencoder not being able to determine what information is relevant. Suppose you plan to use a neural network for your model and an autoencoder to reduce the dimensionality. First, the autoencoder reduces dimensionality while keeping as much information as possible. Then the neural network model extracts as much *relevant* information as possible from that. The result is essentially transfer learning. It's the same as if you took the encoding layers of the autoencoder and put the model on top. The problem, however, is that the encoding layers try to preserve *quantity* of information rather than *quality* of information. You could instead train a network from scratch with the same architecture as the combined structure and expect better results (with proper convergence).

Pixel-RNN: Modeling of the distribution of natural images is a landmark problem in unsupervised learning. This task requires an image model that is at once expressive, tractable and scalable. A deep neural network that sequentially predicts the pixels in an image along the two spatial dimensions. Our method models the discrete probability of the raw pixel values and encodes the complete set of dependencies in the image. Architectural novelties include fast two- dimensional recurrent layers and an effective use of residual connections in deep recurrent net- works. We achieve log-likelihood scores on natural images that are considerably better than the previous state of the art. Our main results also provide benchmarks on the diverse ImageNet dataset. Samples generated from the model appear crisp, varied and globally coherent.

DCGAN: 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), that 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.

Disadvantages of DCGAN: Training a DCGAN requires finding a Nash equilibrium of a game. Sometimes gradient descent does this, sometimes it doesn't. We don't really have a good equilibrium finding algorithm yet, so GAN training is unstable compared to VAE or PixelRNN training. I'd argue that it still feels a lot more stable than Boltzmann machine training in practice. It's hard to learn to generate discrete data, like text. Compared to Boltzmann machines, it's hard to do things like guess the value of one pixel given another pixel. DCGANs are really trained to do just one thing, which is generate all the pixels in one shot. You can fix this by using a BiGAN, which lets you guess missing pixels using Gibbs sampling, the same as in a Boltzmann machine.

Variational auto-encoder : The variational Bayesian (VB) approach involves the optimization of an approximation to the intractable posterior. Unfortunately, the common mean-field approach requires analytical solutions of expectations w.r.t. the approximate posterior, which are also intractable in the general case. We show how a reparameterization of the variational lower bound

yields a simple differentiable unbiased estimator of the lower bound; this SGVB (Stochastic Gradient Variational Bayes) estimator can be used for efficient approximate posterior inference in almost any model with continuous latent variables and/or parameters, and is straightforward to optimize using standard stochastic gradient ascent techniques.

For the case of an i.i.d. dataset and continuous latent variables per data point, we propose the Auto- Encoding VB (AEVB) algorithm. In the AEVB algorithm we make inference and learning especially efficient by using the SGVB estimator to optimize a recognition model that allows us to perform very efficient approximate posterior inference using simple ancestral sampling, which in turn allows us to efficiently learn the model parameters, without the need of expensive iterative inference schemes (such as MCMC) per data point. The learned approximate posterior inference model can also be used for a host of tasks such as recognition, denoising, representation and visualization purposes. When a neural network is used for the recognition model.

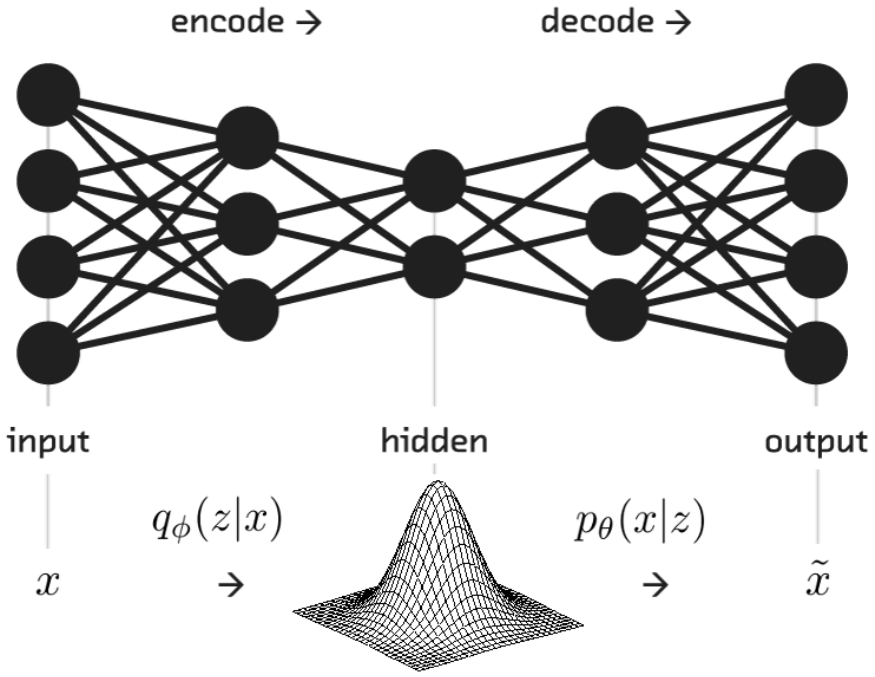


Figure 2.1 - Variational Autoencoder

Disadvantages of Variational auto-encoder: Because of the injected noise and imperfect reconstruction, and with the standard decoder (with factorized output distribution), the generated samples are much more blurred than those coming from GANs. The fact that VAEs basically

optimize likelihood while GANs optimize something else can be viewed both as an advantage or a disadvantage for either one. Maximizing likelihood yields an estimated density that always bleeds probability mass away from the estimated data manifold. GANs can be happy with a very sharp estimated density function even if it does not perfectly coincide with the data density (i.e. some training examples may come close to the generated images but might still have nearly zero probability under the generator, which would be infinitely bad in terms of likelihood). GANs tend to be much more finicky to train than VAEs, not to mention that we do not have a clear objective function to optimize, but they tend to yield nicer images.

WaveNet: A deep generative model of raw audio waveforms. WaveNets are able to generate speech which mimics any human voice and which sounds more natural than the best existing Text-to-Speech systems, reducing the gap with human performance by over 50%. The same network can be used to synthesize other audio signals such as music, and present some striking samples of automatically generated piano pieces. Researchers usually avoid modelling raw audio because it ticks so quickly: typically, 16,000 samples per second or more, with important structure at many time-scales. Building a completely autoregressive model, in which the prediction for every one of those samples is influenced by all previous ones (in statistics-speak, each predictive distribution is conditioned on all previous observations), is clearly a challenging task. It is a fully convolutional neural network, where the convolutional layers have various dilation factors that allow its receptive field to grow exponentially with depth and cover thousands of timesteps. At training time, the input sequences are real waveforms recorded from human speakers. After training, we can sample the network to generate synthetic utterances. At each step during sampling a value is drawn from the probability distribution computed by the network. This value is then fed back into the input and a new prediction for the next step is made. Building up samples one step at a time like this is computationally expensive, but we have found it essential for generating complex, realistic-sounding audio.

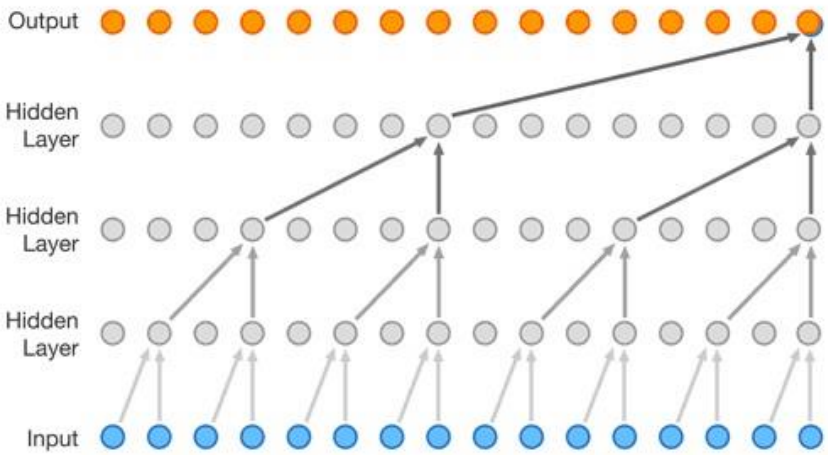


Figure 2.2 - Google Wavenet model

Latent-feature discriminative model (M1): A commonly used approach is to construct a model that provides an embedding or feature representation of the data. Using these features, a separate classifier is thereafter trained. The embeddings allow for a clustering of related observations in a latent feature space that allows for accurate classification, even with a limited number of labels. Instead of a linear embedding, or features obtained from a regular auto-encoder, we construct a deep generative model of the data that is able to provide a more robust set of latent features. The generative model we use is:

$p(z) = N(z|0,I)$; $p\theta(x|z) = f(x;z,\theta)$, (1) where f (x; z, θ) is a suitable likelihood function (e.g., a Gaussian or Bernoulli distribution) whose probabilities are formed by a non-linear transformation, with parameters θ, of a set of latent variables z. This non-linear transformation is essential to allow for higher moments of the data to be captured by the density model, and we choose these non-linear functions to be deep neural networks. Approximate samples from the posterior distribution over the latent variables $p(z|x)$ are used as features to train a classifier that predicts class labels y, such as a (transductive) SVM or multinomial regression. Using this approach, we can now perform classification in a lower dimensional space since we typically use latent variables whose dimensionality is much less than that of the observations. These low dimensional embeddings should now also be more easily separable since we make use of independent latent Gaussian posteriors whose parameters are formed by a sequence of non-linear transformations of the data. This simple approach results in improved performance for SVMs.

Generative semi-supervised model (M2): We propose a probabilistic model that describes the data as being generated by a latent class variable y in addition to a continuous latent variable z. The data is explained by the generative process:

$p(y) = Cat(y|\pi); p(z) = N(z|0, I); p\theta(x|y, z) = f(x; y, z, \theta), (2)$

where $Cat(y|\pi)$ is the multinomial distribution, the class labels y is treated as latent variables if no class label is available and z are additional latent variables. These latent variables are marginally independent and allow us, in case of digit generation for example, to separate the class specification from the writing style of the digit. As before, $f(x;y,z,\theta)$ is a suitable likelihood function, e.g., a Bernoulli or Gaussian distribution, parameterized by a non-linear transformation of the latent variables. In our experiments, we choose deep neural networks as this non-linear function. Since most labels y are unobserved, we integrate over the class of any unlabelled data during the infer- ence process, thus performing classification as inference. Predictions for any missing labels are obtained from the inferred posterior distribution $p\theta(y|x)$. This model can also be seen as a hybrid continuous-discrete mixture model where the different mixture components share parameters.

Stacked generative semi-supervised model (M1+M2): We can combine these two approaches by first learning a new latent representation z1 using the generative model from M1, and subsequently learning a generative semi-supervised model M2, using embeddings from z1 instead of the raw data x. The result is a deep generative model with two layers of stochastic variables: $p\theta(x, y, z1, z2) = p(y)p(z2)p\theta(z1|y,z2)p\theta(x|z1)$, where the priors p(y) and p(z2) equal those of y and z above, and both $p\theta(z1|y, z2)$ and $p\theta(x|z1)$ are parameterized as deep neural networks.

AEGAN: Auto-encoding generative adversarial networks (GANs) combine the standard GAN algorithm, which discriminates between real and model-generated data, with a reconstruction loss given by an auto-encoder. Such models aim to prevent mode collapse in the learned generative model by ensuring that it is grounded in all the available training data. This also specified the principle upon which auto-encoders can be combined with generative adversarial networks by exploiting the hierarchical structure of the generative model. The underlying principle shows that variational inference can be used a basic tool for learning, but with the in- tractable likelihood replaced by a synthetic likelihood, and the unknown posterior distribution replaced by an implicit distribution; both synthetic likelihoods and implicit posterior distributions can be learned using

discriminators. This allows us to develop a natural fusion of variational auto-encoders and generative adversarial networks, combining the best of both these methods. We describe a unified objective for optimization, discuss the constraints needed to guide learning, connect to the wide range of existing work, and use a battery of tests to systematically and quantitatively assess the performance of our method.

Deep Autoregressive Networks: A deep, generative autoencoder capable of learning hierarchies of distributed representations from data. Successive deep stochastic hidden layers are equipped with autoregressive connections, which enable the model to be sampled from quickly and exactly via ancestral sampling. We derive an efficient approximate parameter estimation method based on the minimum description length (MDL) principle, which can be seen as maximizing a variational lower bound on the log-likelihood, with a feedforward neural network implementing approximate inference. We demonstrate state-of-the-art generative performance on a number of classic data sets, including several UCI data sets, MNIST and Atari 2600 games.

DRAW: A Recurrent Neural Network for Image Generation: Deep Recurrent Attentive Writer (DRAW) neural network architecture for image generation. DRAW networks combine a novel spatial attention mechanism that mimics the foveation of the human eye, with a sequential variational auto-encoding framework that allows for the iterative construction of complex images. The system substantially improves on the state of the art for generative models on MNIST, and, when trained on the Street View House Numbers dataset, it generates images that cannot be distinguished from real data with the naked eye. The core of the DRAW architecture is a pair of recurrent neural networks: an encoder network that compresses the real images presented during training, and a decoder that reconstitutes images after receiving codes. The combined system is trained end-to-end with stochastic gradient de- scent, where the loss function is a variational upper bound on the log-likelihood of the data. It therefore belongs to the family of variational auto-encoders, a recently emerged hybrid of deep learning and variational inference that has led to significant advances in generative modelling.

CHAPTER 3
SYSTEM STUDY AND ANALYSIS

3.1 REQUIREMENT ANALYSIS

Operating System: macOS / Windows / Linux

Language: Python

Processor: 64-bit multicore processor preferred

RAM: 4 GB Recommended

Disk Space: 30 GB

3.1.1 HARDWARE REQUIREMENT

The proposed project doesn't involve any special hardware to be run but a PC with a powerful 64-bit multi core processor and at least 4 GB RAM is is recommended as the training of the model is resource intensive.

GPU (Optional): Using a Graphical Processing Unit with Tensorflow can exponentially increase the performance and minimize the time consumed in performing the computation, training the model, etc.

3.1.2 SOFTWARE REQUIREMENT

The project is designed to run on any operating system that supports Python and the project dependencies such as Tensorflow (Currently supports all major operating system).

Python: Python is a widely used high-level programming language for general-purpose programming, features a dynamic type system and automatic memory management and supports multiple programming paradigms, including object-oriented, imperative, functional programming, and procedural styles. It has a large and comprehensive standard library. Python interpreters are available for many operating systems, allowing Python code to run on a wide variety of systems.

Tensorflow: Tensorflow is an open source software library for implementing machine learning, it provides a framework for building and training neural networks, it can be used to perform numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API.

PIL: Python Imaging Library (abbreviated as PIL) (in newer versions known as Pillow) is a free library for the Python programming language that adds support for opening, manipulating, and saving many different image file formats. Pillow offers several standard procedures for image manipulation. These include:

- per-pixel manipulations,
- masking and transparency handling,
- image filtering, such as blurring, contouring, smoothing, or edge finding,
- image enhancing, such as sharpening, adjusting brightness, contrast or color,
- adding text to images and much more.

SciPy: SciPy is an open source Python library used for scientific computing and technical computing. SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering. SciPy builds on the NumPy array object and is part of the NumPy stack which includes tools like Matplotlib, pandas and SymPy, and an expanding set of scientific computing libraries. This NumPy stack has similar users to other applications such as MATLAB, GNU Octave, and Scilab. The NumPy stack is also sometimes referred to as the SciPy stack. The SciPy package of key algorithms and

functions core to Python's scientific computing capabilities. Available sub-packages include:

- **constants**: physical constants and conversion factors (since version 0.7.0)
- **cluster**: hierarchical clustering, vector quantization, K-means
- **fftpack**: Discrete Fourier Transform algorithms
- **integrate**: numerical integration routines
- **interpolate**: interpolation tools
- **io**: data input and output
- **lib**: Python wrappers to external libraries
- **linalg**: linear algebra routines
- **misc**: miscellaneous utilities (e.g. image reading/writing)
- **ndimage**: various functions for multi-dimensional image processing
- **optimize**: optimization algorithms including linear programming
- **signal**: signal processing tools
- **sparse**: sparse matrix and related algorithms
- **spatial**: KD-trees, nearest neighbors, distance functions
- **special**: special functions
- **stats**: statistical functions
- **weave**: tool for writing C/C++ code as Python multiline strings

NumPy: NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The core functionality of NumPy is its "ndarray", for *n*-dimensional array, data structure. These arrays are strided views on memory. In contrast to Python's built-in list data structure (which, despite the name, is a dynamic array), these arrays are homogeneously typed: all elements of a single array must be of the same type. Such arrays can also be views into memory buffers allocated by C/C++, Cython, and Fortran extensions to the CPython interpreter without the need to copy data around, giving a degree of compatibility with existing numerical libraries. This functionality is exploited by the SciPy package, which wraps a number of such libraries (notably BLAS and LAPACK). NumPy has built-in support for memory-mapped ndarrays.

Pandas: pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for

manipulating numerical tables and time series. The library can be used in DataFrame object for data manipulation with integrated indexing, Tools for reading and writing data between in-memory data structures and different file formats, Data alignment and integrated handling of missing data, Reshaping and pivoting of data sets, Label-based slicing, fancy indexing, and subsetting of large data sets, Data structure column insertion and deletion, Group by engine allowing split-apply-combine operations on data sets, Data set merging and joining, Hierarchical axis indexing to work with high-dimensional data in a lower-dimensional data structure, Time series-functionality: Date range generation and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging. The library is highly optimized for performance, with critical code paths written in Cython or C.

Prettytensor: Pretty Tensor provides a high level builder API for TensorFlow. It provides thin wrappers on Tensors so that you can easily build multi-layer neural networks.Pretty Tensor provides a set of objects that behave likes Tensors, but also support a chainable object syntax to quickly define neural networks and other layered architectures in TensorFlow. Pretty Tensors can be used (almost) everywhere that a tensor can.Just call pt.wrap to make a tensor pretty. You can also add any existing TensorFlow function to the chain using apply. apply applies the current Tensor as the first argument and takes all the other arguments as normal. It also uses standard TensorFlow idioms so that it plays well with other libraries, this means that you can use it a little bit in a model or throughout.

3.2 Feasibility Study

The project involves training of a Generative Adversarial Network with a Generator and Discriminator with around 18 Gigabytes of a dataset including the “Caltech-UCSD Birds 200” which is for the images and text embeddings and the skipthoughts model. Training the model is the most time consuming and resource consuming phase of the project with **600 epochs** of training.

CHAPTER 4
SYSTEM DEVELOPMENT

4.1 PROPOSED SYSTEM

The block diagram of the proposed model is shown in Figure 4.1. The different blocks are explained in the subsections.

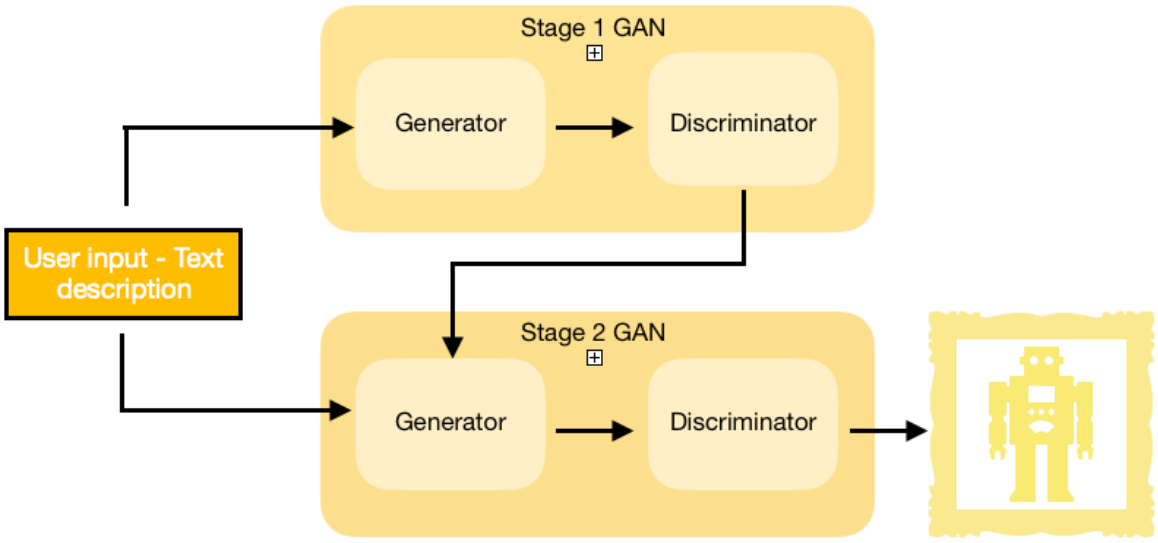


Figure 4.1 - Tesseract Workflow

4.1.1 User input - Text description

The models accept the user input in the form of a text description describing the image representation, and the user needs to note that the model is only capable of generating images based of what it has been trained on and cannot generate something it doesn't know for example if it is trained for generating photorealistic images on chairs based of the user's description it wont be able to generate images on tables. The text description can be passed on as the input in a natural way as well and doesn't require any special formatting and the user can give the desired amount of descriptions together, the given input text description is processed for word

embeddings. Word embedding is the collective name for a set of language modeling and feature learning techniques in natural language processing (NLP) where words or phrases from the vocabulary are mapped to vectors of real numbers. Conceptually it involves a mathematical embedding from a space with one dimension per word to a continuous vector space with much lower dimension. Methods to generate this mapping include neural networks, dimensionality reduction on the word co-occurrence matrix, probabilistic models, and explicit representation in terms of the context in which words appear. Word and phrase embeddings, when used as the underlying input representation, have been shown to boost the performance in NLP tasks such as syntactic parsing and sentiment analysis. In linguistics word embeddings were discussed in the research area of distributional semantics. It aims to quantify and categorize semantic similarities between linguistic items based on their distributional properties in large samples of language data. Thought vectors are an extension of word embeddings to entire sentences or even documents. Some researchers hope that these can improve the quality of machine translation.

4.1.2 Stage 1 GAN (Generator)

The user's input is passed on to the Stage 1 Generative Adversarial Network's Generator, from which it takes the word vectors and based of the training from the images and its associated text embeddings it samples a low-resolution image with basic shapes and primitive colors. The generated low-resolution image is passed on as the input to the Discriminator

4.1.3 Stage 1 GAN (Discriminator)

The Discriminator takes the input image from the Generator and classifies it as a real image or a fake image, if the generator generates a compelling fake image the discriminator classifies it as a real image. The GANs Generator and Discriminator are trained using the same dataset. If the generated image is classified as real, the image and the original user input in the form of text description are passed on as the input to the Stage 2 Generative Adversarial Network's Generator

4.1.4 Stage 2 GAN (Generator & Discriminator)

The Generator takes the original text description and the Stage 1 GANs output image as the input and Generates a much higher resolution image by completing the details of the image, a random noise is also introduced to the generated image for variations such as in the viewing angle, etc. The discriminator in the Stage 2 GAN has the same purpose of the Stage 1 GAN, if the Discriminator is bypassed by the Generator the Generated image is the final output.

CHAPTER 5
SYSTEM IMPLEMENTATION

5.1 SYSTEM IMPLEMENTATION

A text file is maintained for the user input of text descriptions, different image's descriptions can be separated by a new line, the program opens and reads the file containing the user input and it splits each image's description based of a new line and stores the descriptions in a Python List variable, A variable is initialized with the path to store the generated images in, the program then loads the Skipthoughts text encoder model and then processes each text description in the list variable for the embeddings, the number of word embeddings in the description is stored in a variable, the trained GAN model is loaded and initialized into a new tensorflow session, 16 images are generated (8 for each Stage), the images are drawn onto a single image using PIL (Python Imaging Library).

5.2 Python File I/O

Before you can read or write a file, you have to open it using Python's built-in open() function. This function creates a file object, which would be utilized to call other support methods associated with it. Once a file is opened and you have one file object, you can get various information related to that file. The close() method of a file object flushes any unwritten information and closes the file object, after which no more writing can be done. Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the close() method to close a file. The *write()* method writes any string to an open file. It is important to note that Python strings can have binary data and not just text. The write() method does not add a newline character ('\n') to the end of the string. The *read()* method reads a string from an open file. It is important to note that Python strings can have binary data. apart from text data.

5.3 Skipthoughts text encoder

An approach for unsupervised learning of a generic, distributed sentence encoder. Using the continuity of text from books, we train an encoder - decoder model that tries to reconstruct the surrounding sentences of an encoded passage. Sentences that share semantic and syntactic properties are thus mapped to similar vector representations and also a simple vocabulary expansion method to encode words that were not seen as part of training, allowing us to expand our vocabulary to a million words. After training our model, we extract and evaluate our vectors with linear models on 8 tasks: semantic relatedness, paraphrase detection, image-sentence ranking, question-type classification and 4 benchmark sentiment and subjectivity datasets. The end result is an off-the-shelf encoder that can produce highly generic sentence representations that are robust and perform well in practice. An encoder maps words to a sentence vector and a decoder is used to generate the surrounding sentences. Encoder- decoder models have gained a lot of traction for neural machine translation. In this setting, an encoder is used to map e.g. an English sentence into a vector. The decoder then conditions on this vector to generate a translation for the source English sentence. Several choices of encoder-decoder pairs have been explored, including ConvNet-RNN, RNN-RNN and LSTM-LSTM. The source sentence representation can also dynamically change through the use of an attention mechanism to take into account only the relevant words for translation at any given time. In our model, we use an RNN encoder with GRU activations and an RNN decoder with a conditional GRU. This model combination is nearly identical to the RNN encoder-decoder of used in neural machine translation. GRU has been shown to perform as well as LSTM on sequence modelling tasks while being conceptually simpler. GRU units have only 2 gates and do not require the use of a cell. While we use RNNs for our model, any encoder and decoder can be used so long as we can backpropagate through it.

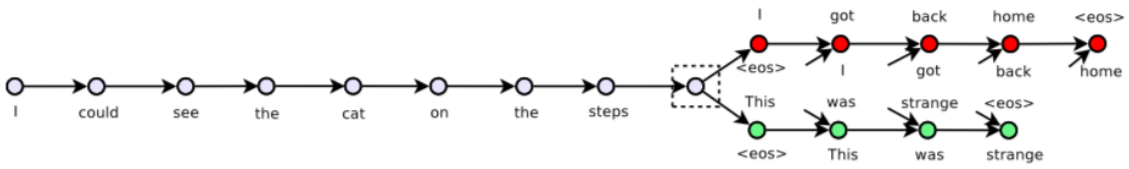


Figure 5.1 - Word Encoding

The skip-thoughts model. Given a tuple (si−1, si, si+1) of contiguous sentences, with si the i-th sentence of a book, the sentence si is encoded and tries to reconstruct the previous sentence si−1 and next sentence si+1. In this example, the input is the sentence triplet I got back home. I could see the cat on the steps. This was strange. Unattached arrows are connected to the encoder output. Colors indicate which components share parameters. ⟨ eos⟩ is the end of sentence token. Language modeling is a fundamental task in artificial intelligence and natural language processing (NLP), with applications in speech recognition, text generation, and machine translation. A language model is formalized as a probability distribution over a sequence of strings (words), and traditional methods usually involve making an n-th order Markov assumption and estimating n-gram probabilities via counting and subsequent smoothing (Chen and Goodman 1998). The count-based models are simple to train, but probabilities of rare n-grams can be poorly estimated due to data sparsity (despite smoothing techniques). Neural Language Models (NLM) address the n-gram data sparsity issue through parameterization of words as vectors (word embeddings) and using them as inputs to a neural network (Bengio, Ducharme, and Vincent 2003; Mikolov et al. 2010). The parameters are learned as part of the training process. Word embeddings obtained through NLMs exhibit the property whereby semantically close words are likewise close in the induced vector space (as is the case with nonneural techniques such as Latent Semantic Analysis

5.4 Tesseract Model (sGAN)

A 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 everywhere. In the case where G and D are defined 2 by multilayer perceptrons, the entire system can be trained with backpropagation. There is no need for any Markov chains or unrolled approximate inference net- works during either training or generation of samples. Experiments

demonstrate the potential of the framework through qualitative and quantitative evaluation of the generated samples.

5.5 Tensorflow session

A Session object encapsulates the environment in which Operation objects are executed, and Tensor objects are evaluated. A session may own resources, such as tf.Variable, tf.QueueBase, and tf.ReaderBase. It is important to release these resources when they are no longer required. To do this, either invoke the tf.Session.close method on the session, or use the session as a context manager. The ConfigProto protocol buffer exposes various configuration options for a session. For example, to create a session that uses soft constraints for device placement, and log the resulting placement decisions, create a session If no graph argument is specified when constructing the session, the default graph will be launched in the session. If you are using more than one graph (created with tf.Graph() in the same process, you will have to use different sessions for each graph, but each graph can be used in multiple sessions. In this case, it is often clearer to pass the graph to be launched explicitly to the session constructor. you can use with tf.Session(): to create a session that is automatically closed on exiting the context, including when an uncaught exception is raised. In simple words, a session allows to execute graphs or part of graphs. It allocates resources (on one or more machines) for that and holds the actual values of intermediate results and variables.

5.6 PIL (ImageDraw)

The ImageDraw module provides simple 2D graphics for Image objects. You can use this module to create new images, annotate or retouch existing images, and to generate graphics on the fly for web use. The graphics interface uses the same coordinate system as PIL itself, with (0, 0) in the upper left corner. To specify colors, you can use numbers or tuples just as you would use with PIL.Image.Image.new() or PIL.Image.Image.putpixel(). For “1”, “L”, and “I” images, use integers. For “RGB” images, use a 3-tuple containing integer values. For “F” images, use integer or floating-point values. For palette images (mode “P”), use integers as color indexes. In 1.1.4 and later, you can also use RGB 3-tuples or color names (see below). The drawing layer will automatically assign color indexes, as long as you don’t draw with more than 256 colors. PIL

can use bitmap fonts or OpenType/TrueType fonts. Bitmap fonts are stored in PIL's own format, where each font typically consists of two files, one named. pil and the other usually named .pbm. The former contains font metrics, the latter raster data. To load a bitmap font, use the load functions in the ImageFont module. To load a OpenType/TrueType font, use the truetype function in the ImageFont module. Note that this function depends on third-party libraries, and may not available in all PIL builds.

5.7 Datasets and evaluation metrics:

CUB contains 200 bird species with 11,788 images. Since 80% of birds in this dataset have object-image size ratios of less than 0.5, as a pre-processing step, we crop all images to ensure that bounding boxes of birds have greater-than-0.75 object-image size ratios. Oxford-102 contains 8,189 images of flowers from 102 different categories. To show the generalization capability of our approach, a more challenging dataset, MS COCO is also utilized for evaluation. Different from CUB and Oxford- 102, the MS COCO dataset contains images with multiple objects and various backgrounds. It has a training set with 80k images and a validation set with 40k images. Each image in COCO has 5 descriptions, while 10 descriptions are provided by for every image in CUB and Oxford- 102 datasets. Following the experimental setup in, we directly use the training and validation sets provided by COCO, meanwhile we split CUB and Oxford-102 into class-disjoint training and test sets.

CHAPTER 6
TESTING AND EXPERIMENTAL RESULT

6.1 TESTING PHASE

Testing is the phase of evaluating a system or its components to find whether it satisfies the specified requirements or not. Testing phase is executed in order to identify the errors or missing requirements. Testing can be started from the requirements gathering phase and continued till the deployment of the project. The analysis, verification and reviewing the flow is also considered as testing.

During testing phase, 5 test text descriptions are given as the input and is tested whether the near expected output is gained or not.

6.1.1 INCEPTION IMAGE RECOGNITION TEST

Using Google's Inception, a machine learning model for image recognition can be used to get the conditional label distribution of every generated image. Images that contain meaningful objects should have conditional label distribution with low entropy, while the marginal images have higher entropy.

6.1.2 HUMAN ANNOTATION

GANs lack of objective function make it harder to compare performance of different proposed models. Human annotators can be used to judge the visual quality of samples.

6.2 RESULT ANALYSIS

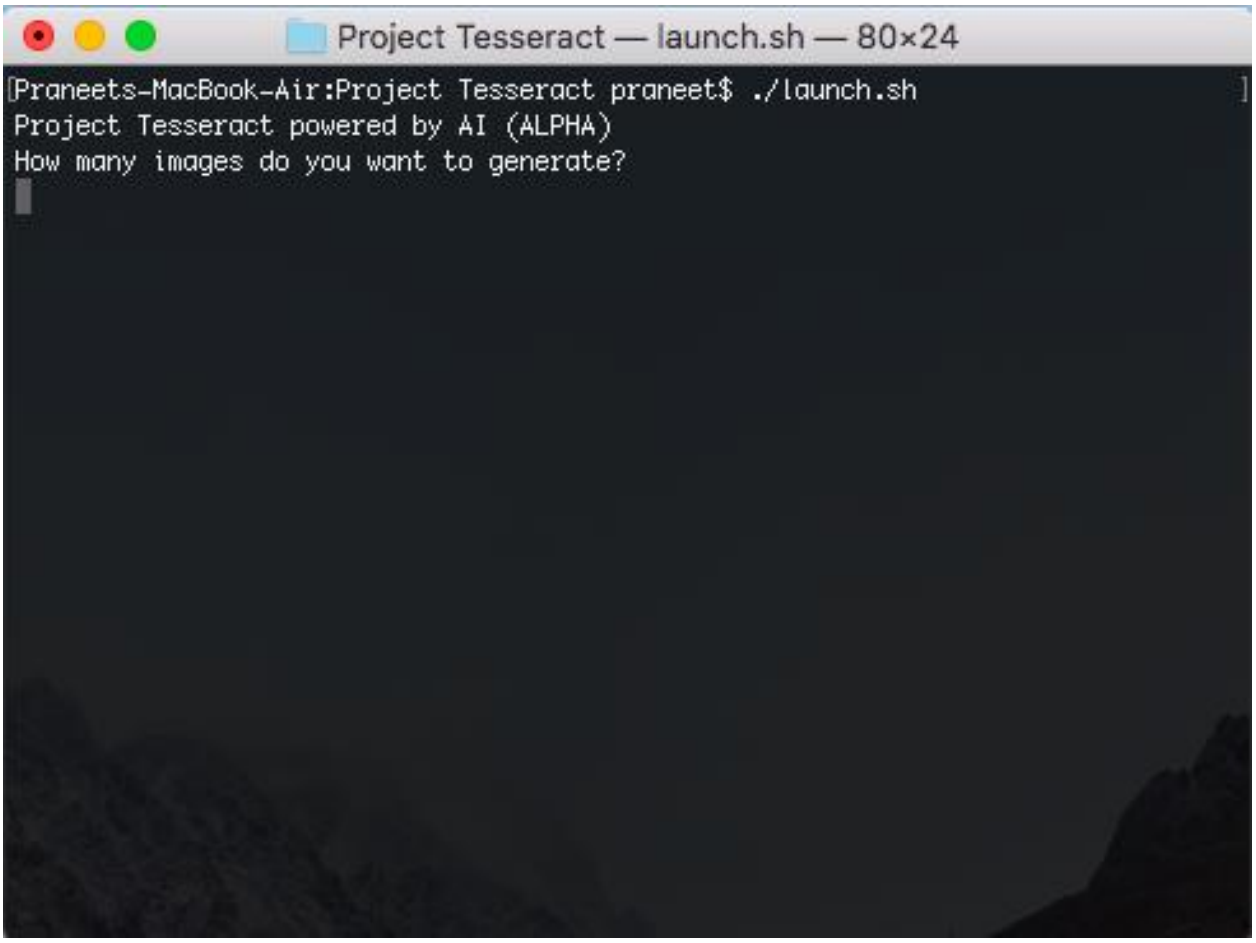
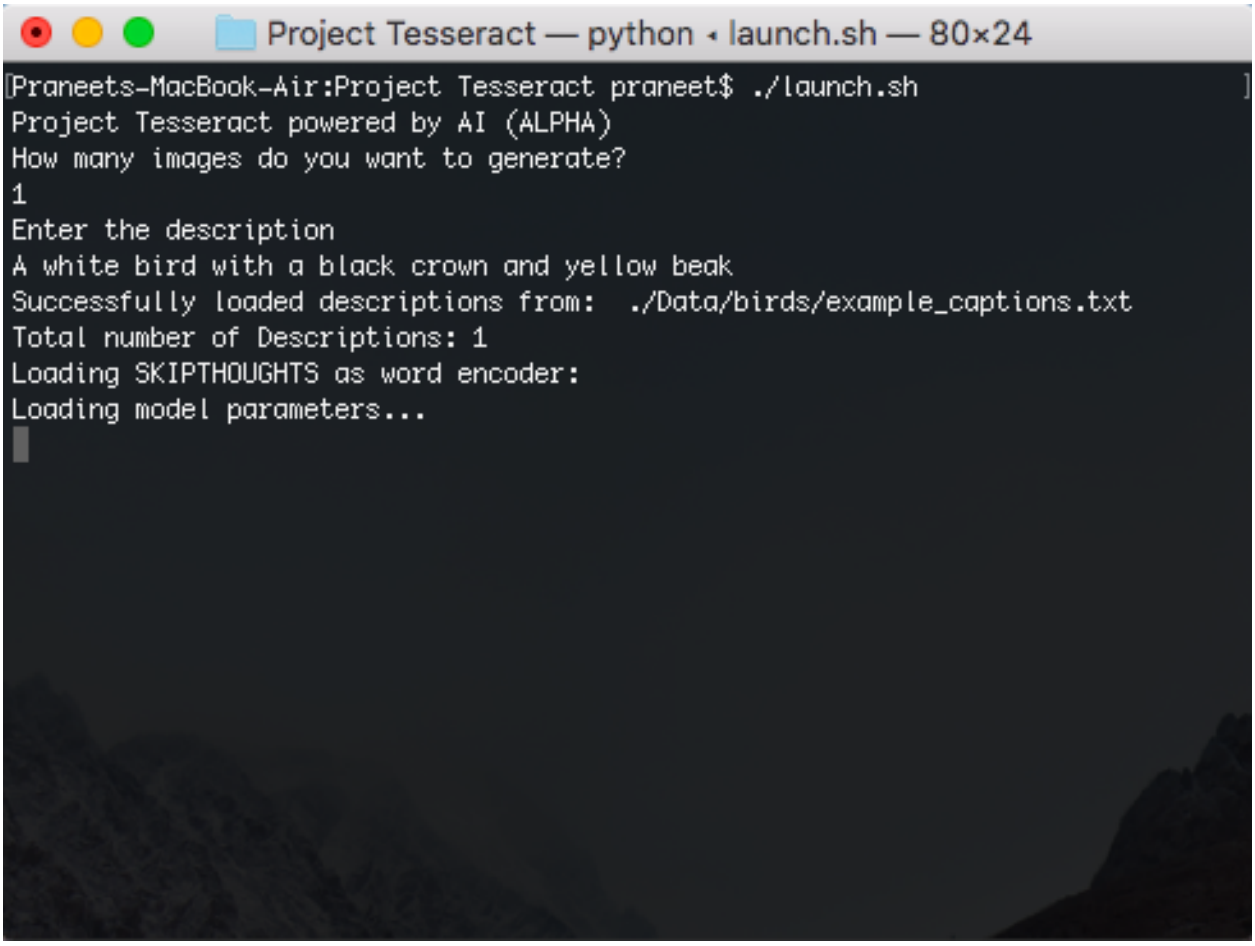


Figure 6.1 - Application running in the terminal

A BASH script has been written for the project, running the script in the terminal, it asks for the total number of images to be generated.



Giving “1” as the number of images to be generated, the terminal waits for the description input and stores it into a variable and writes the descriptions to the captions file, the program begins execution after the virtual environment is activated, then the program loads the descriptions from the file and then loads the Skipthoughts word encoder and processes the input text descriptions.

Figure 6.2 - Loading the Skipthought model

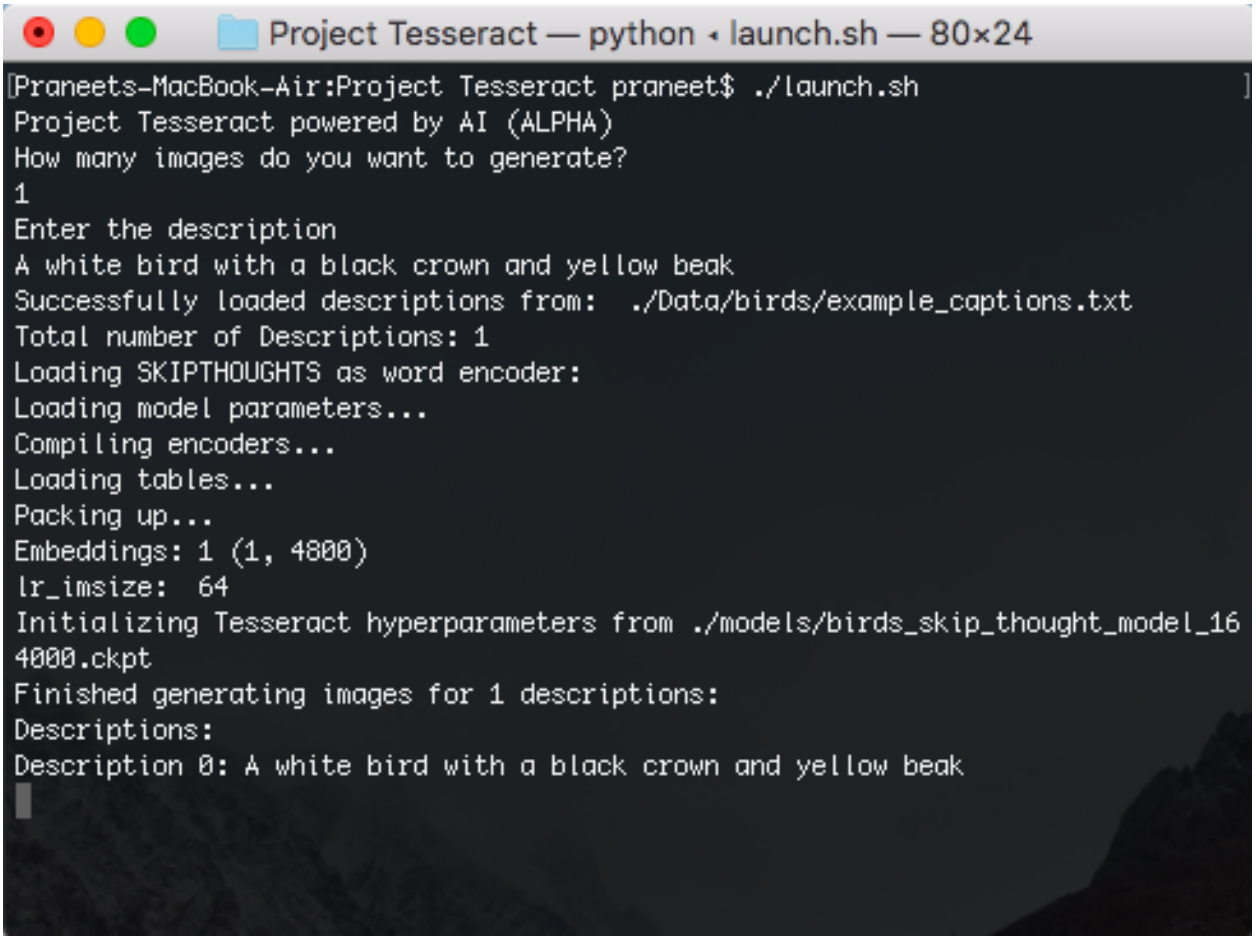


Figure 6.3 - Loading the tesseract model and generating the image

The Tesseract model is initialized into the Tensorflow session and begins Generating the images for each description and draws it onto a single formatted image using Python Imaging Library.

Given the input “A white bird with a black crown and yellow beak” as the text description, the model generates the following set of images:

Figure 5.4 - Output for the given description

The 4th image and the 6th image from the set of images generated by the Stage - II GAN have high amount of noise but the rest of the images have the desired output.

Evaluation of generative models till date: Generative models have many applications and can be evaluated in many ways. For density estimation and related tasks, log-likelihood (or equivalently Kullback-Leibler divergence) has been the de-facto standard for training and evaluating generative models. However, the likelihood of many interesting models is computationally intractable. For example, the normalization constant of unnormalized energy-based models is generally difficult to compute, and latent-variable models often require us to solve complex integrals to compute the likelihood. These models may still be trained with respect to a different objective that is more or less related to log-likelihood, such as contrastive divergence (Hinton, 2002), score matching (Hyvarinen, 2005), lower bounds on the log-likelihood ¨ (Bishop, 2006), noise-contrastive estimation (Gutmann & Hyvarinen, 2010), probability flow (Sohl- ¨ Dickstein et al., 2011), maximum mean discrepancy (MMD) (Gretton et al., 2007; Li et al., 2015), or approximations to the Jensen-Shannon divergence (JSD) (Goodfellow et al., 2014). For computational reasons, generative models are also often compared in terms of properties more readily accessible than likelihood, even when the task is density estimation. Examples include visualizations of model samples, interpretations of model parameters (Hyvarinen et al., 2009), Parzen ¨ window estimates of the model's log-likelihood (Breuleux et al., 2009), and evaluations of model performance in surrogate tasks such as denoising or missing value imputation. Just as choosing the right training method is important for achieving good performance in a given application, so is choosing the right evaluation metric for drawing the right conclusions. In the following, we first continue to discuss the relationship between average log-likelihood and the visual appearance of model samples. Model samples can be a useful diagnostic tool, often allowing us to build an intuition for why a model might fail and how it could be improved. However, qualitative as well as quantitative analyses based on model samples can be misleading about a model's density estimation performance, as well as the probabilistic model's performance in applications other than image synthesis.

CONCLUSION

Thus, we have developed a sGAN model “Tesseract” for photorealistic image synthesis from text descriptions, currently the model is trained using the “**Caltech-UCSD Birds 200**” dataset for generating photorealistic images based of the user’s descriptions on birds even if the bird doesn’t exist on earth.

By using machine learning we have created a novel approach in image generation which can create breakthroughs in many fields like hardware prototyping, fashion designing, etc. The model can be further trained on various other datasets for making the model more generic and even being able to generate any image based of the user’s description. The randomness introduced given new perspective to the generated images making it even more photorealistic.

FUTURE ENHANCEMENTS

Accuracy is a very important aspect in the field of machine learning and AI as it is the factor which decides if the produced output is reliable, worth or not. There are a lot of techniques which can be used to improve the machine learning model's accuracy but they are either resource intensive or very generalized meaning it works well with one kind of a model and performs just slightly better which is not satisfying. In this research paper we are proposing a system for enhancing and improving the accuracy of Stacked Generative Adversarial Networks where we employ a double discriminative system on the generated result. Current techniques for minimization of cost functions and enhancements for GANs include finding the Nash equilibrium which is a very difficult problem and is difficulty to implement apart from the traditional Gradient based minimization and it has its own drawbacks. And Feature matching for minimizing the instability of GANs by preventing the generator from overfitting the discriminator, minibatch discrimination where the discriminator processes on several generations together rather than individually preventing the collapse of the generator and several other general enhancements in the field of neural nets. Stacked Generative Adversarial Networks, the existing system, it has 2 Generative Adversarial Networks which have a Generator and a Discriminator, the Generator and the Discriminator are trained using the same dataset, GAN's are useful for generating Photorealistic images or can be used for semisupervised learning. The Generator generates images based of the images it has been trained with and the discriminator is used to distinguish if the generated image is real or fake, if the generated image is so compelling the discriminator passes it on as the real output in the case of photorealistic image synthesis. Our propose system, an "Enhanced Stacked Generative Adversarial Networks" has 2 GANs, the stage 1 GAN is unmodified with just one Generator and a Discriminator while the stage 2 GAN is modified with the addition of a combination of a Convolutional Neural Network (CNN) and a Bidirectional Recurrent Neural Network (BRNN) for captioning of the generated image, the generated Caption is semantically compared with the input text with a semantic string comparison module, if the comparison results with a score greater than a threshold value then the image is passed on as the output image, if the comparison's score results with a value lesser than the threshold value the image is rejected and the model has to generate a more compelling image or retrain a model to produce a higher accuracy compared to the existing GAN. The proposed system can be

considered as a GAN with two Discriminators where one works as a classifier (to classify the image based on if it looks similar to the trained dataset that the discriminator is fooled thinking it as a real image) while the other Discriminator is used to generate captions of the generated image and it is semantically compared to the input text given to find if the image generated is an accurate output for the input given. For training the Enhanced Stacked Generative Adversarial Networks Online learning can be employed instead of Batch Learning, as Online learning is considered as On the Go learning as new data can be trained without the old data and the trained dataset need not be stored on the disk. Bidirectional Recurrent Neural Networks are used for image captioning with Convolutional Neural networks as they give importance to the order of the input which is fed into the cell which is very important, as the phrase in a wrong order of words can change the complete meaning of a phrase. Convolutional Neural Networks are the state of the art in image feature extraction. If the semantic comparison of the sentences results in a value greater than the threshold value the image is passed on as the output, if not we can reject the image for one of the two possibilities where both work but the rate of accuracies differ: The Stage 1 model of the proposed ESGAN can be used to generate the image from scratch or the Stage 2 model can be directly used to regenerate the image but the input from the Stage 1 ESGAN influences the Stage 2 ESGAN's output. If the Image is regenerated from the scratch the accuracy is higher compared to rebuilding the basic image from the Stage 1 ESGAN.

BIBILIOGRAPHY

Books:

Fundamentals of Deep Learning by Nikil Buduma

Introduction to machine learning with Python by Muller & Guido

Python Data Science Handbook by Jake VanderPlas

Hands-On Machine Learning with Scikit-Learn & Tensorflow by Aurelien Geron

Websites:

https://blog.openai.com/generative-models/

https://deepmind.com/search/?query=generative