Emoji Synthesis from Text using Generative Adversarial Networks

By

Taanya Sunil

Student ID : 2345404

Supervisor : Ms. Ana Stroescu

A thesis submitted to the University of Birmingham
for the degree of MSc Artificial Intelligence and Machine Learning

School of Computer Science
University of Birmingham
Birmingham, UK

September 2022

Table of Contents

List of Figures

List of Tables

Abstract

Over the past two decades, there has been a rise in technology and subsequently a rise in the usage of social media. This has led to a drop in face-to-face human interactions and a rise in communication over social media platforms like Facebook, Instagram, and Snapchat. Presently, messaging over these platforms has become the most used form of communication. However, due to the lack of face-to-face interactions, messaging over mobile devices can lead to a lot of miscommunications. Hence, emojis were created around 20 years ago. They help by adding an emotive element to messages or texts which not only reduces misunderstandings but also makes the messaging experience more enjoyable.

As human beings, we emote daily and hence feel emotions that differ almost every few minutes based on the various situations we encounter. However, at present, there are not enough emojis that can be used while texting to help us express every emotion we feel. Hence, giving users the ability to create emojis on the fly is a vital task that helps improve the messaging experience. This is the main motivation for the project carried out by the author.

The research presented in this paper uses Generative Adversarial Networks (GANs) to carry out Text to Emoji Synthesis. It focuses on improving the results of previous researchers in this field. To do this, the author proposes a modified version of the Conditional Deep Convolutional Generative Adversarial Network (cDCGAN). This modified model generates better emojis than the previous research that has used the standard DCGAN architecture. The model was evaluated using a total of 7 evaluation metrics. This research also depicts the importance of the text used to carry out Text to Emoji Synthesis and shows how modifications made to the texts can help improve Text – Image Alignment which subsequently leads to better results.

Finally, this research also uses three Word Embedding algorithms – GloVe, Word2vec, and FastText to explore how these algorithms influence the model's ability to create emojis, by analysing their performance to see which one performs the best.

Keywords

Generative Adversarial Network; Conditional Generative Adversarial Network; Deep Convolutional Generative Adversarial Network; Word2vec; GloVe; FastText; Structural Similarity Index Measure; Peak Signal to Noise Ratio

Acknowledgement

I would like to express my heartfelt appreciation and gratitude for my supervisor, Ms. Ana Stroescu for her consistent support and guidance throughout this project. Her advice and mentorship over the past few months is what helped make this research a success.

I would also like to extend my sincere appreciation to my professors at the University of Birmingham as well as the Computer Science Department for equipping me with the necessary skills and tools required to carry out this project from start to finish.

Finally, I would also like to thank my parents and brother for their constant encouragement, patience, and support and for always pushing me to think outside of the box and motivating me to put my best foot forward.

CHAPTER I : OVERVIEW

1.1 Introduction

Over the past two decades, the digital and social media boom has led to an increasing desire for people to express themselves via social media platforms such as Facebook, Instagram, and Twitter. Over time, this desire has changed into a need for people to express their emotions quicker, easier, and more accurately. Although words can describe a person's feelings, text can sometimes be misleading due to the lack of facial expressions as we communicate over devices. A visual representation of the text that can express the user's emotions solved this problem. This visual representation is a pictogram called an Emoji.

Emojis help make messaging over social media platforms richer and more emotive. They have helped decipher what each text or tweet means and made our most used form of communication a fun and exciting process since using them for the first time over 20 years ago. They are a substitute for the tone of voice, gestures, and facial expressions [61]. Nowadays, all social media platforms come equipped with a set of emojis. At present, there are a total of 3,633 emojis in the Unicode standard [3]. While this is a large number, it is still not enough to express the wide variety of emotions we experience in the situations we encounter daily.

The only way to solve this issue is by creating more emojis that express a greater variety of emotions. [28] talks about how it is a demanding and tedious process to submit a proposal for a new emoji to the Unicode Emoji Subcommittee [3], which involves a long wait to receive approval which is not guaranteed. This is why previous researchers [1],[2] and [28] have attempted to create models capable of Text to Emoji Synthesis. Giving users the ability to create and customise emojis on the fly by inputting text of their choice is essential and is the primary motivation for this project.

1.2 Aims of the Research

This project aims to use Machine Learning, Deep Learning, and Natural Language Processing (NLP) to create a model capable of Text to Emoji synthesis. The following are the main goals that the project is driven by:

1. Creating a conditional DCGAN model capable of taking a text input from the user and producing a related emoji as an output. The model should be able to regenerate presently available emojis and generate non-existent emojis of good quality that consists of features that one would expect based on the text input.
2. Designing and implementing modifications to the standard DCGAN architecture (Chapter 4) to improve results by creating emojis that are better than the existing adversarial research [1,2] in the Text to Emoji Synthesis field.
3. This research will illustrate the importance of the text used to train the model. It depicts the effect the text has when creating non-existent emojis. It compares the difference in emoji synthesis when using Unicode standard captions [3] versus the modified captions proposed by the author to improve Text-Image Alignment (Chapter 4).
4. Evaluating the model's performance using multiple robust evaluation metrics (Chapter 5).
5. Finally, this study also compares multiple Word Embedding algorithms to find the best performing one by conducting a qualitative analysis of the emojis and a quantitative analysis of the evaluation metrics. (Chapter 6).

1.3 Outline

This research comprises three main parts. The first part involves designing and implementing a modified version of the Conditional Deep Convolutional Generative Adversarial Network (cDCGAN) that would improve the emojis synthesised by the standard DCGAN architecture [4] implemented by previous researchers [1,2] in the field of Text to Emoji Synthesis. The next part outlines the significance of the text we input during training and how modifying the text can improve Text-Image Alignment and

hence, create better-quality emojis. The third part discusses the standard evaluation techniques implemented by the author in this project. It also details two new evaluation techniques that were not previously used in the Text to Emoji Synthesis field. These metrics explore computer vision concepts such as SIFT [5] for Feature Matching, as well as concepts and models such as Transfer Learning and Long Short-Term Memory Networks (LSTM) [7] for Caption Loss.

Chapter I introduced the world of emojis and social media as well as the project's motivation and aim. It also talks about the experimental methodology to be used. The remainder of the research has been organised as written below:

Chapter II presents a literature review and discusses some common GANs and evaluation metrics used in the Text to Image Synthesis field. It then brings the reader's attention to the current methods implemented in the Text to Emoji Synthesis field and their achievements and failures.

Chapter III provides an in-depth background into this field and discusses the main algorithms, deep learning techniques, and Word Embedding algorithms used by the author in this research.

Chapter IV discusses the proposed model's architecture and details its implementation. It discusses critical aspects of the research, such as the dataset, pre-processing techniques and minute details related to the training of the conditional GAN such as the hyperparameters used, which allows the replication of this research for future analysis.

Chapter V discusses the common GAN and Image evaluation techniques the author has used in the research and the two additional evaluation metrics the author has used that were not implemented previously in the Text to Emoji Synthesis field. It also discusses the problems with the dataset and modifications made to the metrics (to make them applicable to the current research) used for evaluation.

Chapter VI discusses the results of the model. In this chapter, the author discusses the evaluation results and performance of the model compared to the standard DCGAN architecture. It gives justifications for the architecture used to build the model. Further, it compares the three Word Embedding algorithms to find the one that creates the best emojis. Finally, it also compares its results to previous work in the field of Text to Emoji Synthesis.

Chapter VII talks about the challenges faced by the author in the course of this project. It discusses the challenges faced while building the GAN and some remedies the author implemented to overcome them. Further, it provides some solutions which can help overcome some of the unsolved problems in the future.

Finally, Chapter VIII concludes the work and discusses the achievements and failures of this paper.

CHAPTER 2 : LITERATURE REVIEW

Introduction

This chapter provides a literature review of Text to Emoji synthesis. It starts by briefly outlining Text to Image Synthesis using Generative Adversarial Networks before delving into the previous research carried out in Text to Emoji Synthesis. Section 2.1 gives a brief introduction into Text to Image synthesis and is followed by Section 2.2, which briefly discusses Generative Adversarial Networks (GANs) and their various applications. Section 2.3 discusses some fascinating Generative Adversarial Networks used in Text to Image Synthesis. This section is followed by Section 2.4, which analyses the most popular evaluation metrics used in this field. Finally, Section 2.5 dives deeper and discusses the past research on Text to Emoji Synthesis and its achievements.

2.1 Text to Image Synthesis (T2I)

Text to Image Synthesis (T2I) is a task that converts natural language text inputted by the user into realistic images using conditional image generation neural networks. The model first recognises the text used to describe the image, and then searches for graphic elements related to the text which have the highest likelihood. The model then builds an image that is a combination of the elements described by the text. In earlier studies, a union of Search Strategy and Supervised Learning [8] carried out the task of converting text to images. However, the images generated by using the aforementioned method were unsatisfactory.

Over time, as the research in the field of Deep Learning improved, the task was carried out by neural networks such as the Variational Autoencoder (VAE) [9]. Although these models [9],[10] could successfully synthesise images from the text, the results were not very good. Generative Adversarial Networks (GANs) proposed by [11] in 2014 were able to produce images of great quality and were shown to have an excellent capability for Text to Image Synthesis. Comparing the performance of GANs with VAEs, the latter created faded and less realistic images. The following section briefly introduces the working of a Generative Adversarial Network.

2.2 Generative Adversarial Networks

Ever since the invention of the Generative Adversarial Network in 2014 [11], Text to Image Synthesis using Adversarial Networks has become a widely researched and a hot topic of interest. This section gives a brief introduction to Generative Adversarial Networks. Section 3.3.5.1 presents a more in-depth description into the working of a GANs.

Generative Adversarial Networks are deep learning models that consist of two neural networks, the Generator and the Discriminator. The Generator is responsible for creating realistic images from noise, and the Discriminator is a binary classifier that identifies if the images fed to it are real or fake. Both these neural networks are trained together and compete with each other. The Generator's objective is to create realistic samples such that the Discriminator fails to recognise them as fake samples. This Generative Adversarial Network has several applications such as image super-resolution [12], data augmentation [15], image in-painting [13], style transfer [14] and many more [63].

Over the years, improved research drove the advent of the Conditional Generative Adversarial Network in late 2014 [16]. They involved adding a conditional parameter 'y' to the input of the Generator, which allows the GAN to generate outputs based on the label ‘y’. This implies that, the Conditional Generative Adversarial Network has the huge advantage of giving the user the control over the kind of output they want the GAN to generate, rather than have multiple randomly generated outputs. One significant application of the Conditional GAN is Text to Image Synthesis. The following section details the most important and exciting research related to Adversarial Text to Image Synthesis that have made the most significant contributions to the advancement of the field.

2.3 Adversarial Text to Image Synthesis

The field of Text to Image (T2I) synthesis has been widely researched over the last few years and has seen a lot of significant improvements in both qualitative and quantitative evaluations. It was first researched in 2016 by Reed et al. [17]. They used Conditional GANs to generate realistic images based on the text descriptions input by the user. However, this model only worked well on restricted datasets.

Text to Image Synthesis using adversarial models [63] is divided into two main categories – Direct T2I methods and T2I methods with Additional Supervision. These categories are further divided into multiple sub-categories based on the architecture. The figure below (Figure 1) shows some of the various sub-categories of Text to Image Synthesis. The section following Figure 1 talks about some intriguing models in this field.

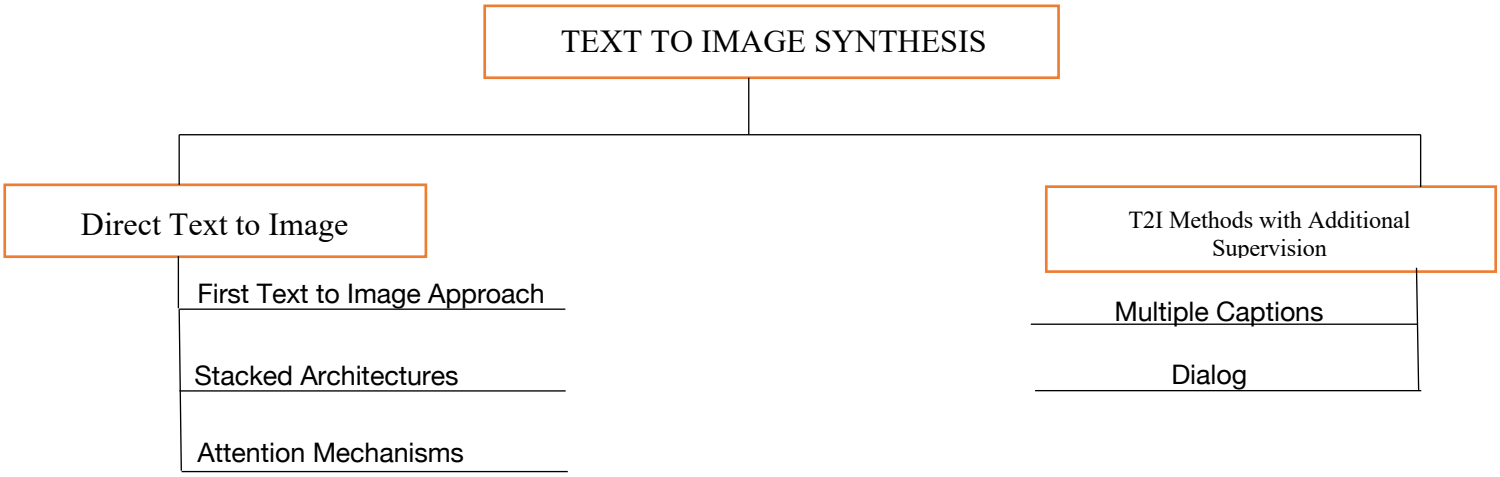


Fig 1 : The various types of Text to Image Synthesis Models

A. Direct Text to Image Synthesis

This section discusses the various Direct Text to Image Synthesis methods that have significantly contributed to this field. These are methods that use a single conditional input. As mentioned earlier, the First Text to Image model was implemented by Reed et al. [17]. This model was called the GAN-INT-CLS and used the Conditional GAN. The first step involves converting the text inputs into Word Embeddings with the help of a pre-trained text encoder. These Word Embeddings are the conditional input of the model.

Although the GAN-INT-CLS [17] was able to synthesise images from the text, these images had very low resolution. In order to have a model create images of greater resolution, many researchers implemented stacked architectures, which involved the use of multiple generators stacked over one another. One such architecture is the StackGAN [18]. This model consists of two stages. A rough image is created from noise by the first stage based on the text. This rough image is then input into a second generator along with the text embeddings created by a pre-trained embedding model. Using these inputs, the second stage creates an image which of much higher resolution than those created by the GAN-INT-CLS. A neural network is trained to distinguish between the real and the generated images. This is the Discriminator which is used in both stages of the StackGAN. StackGAN++ [19] is an improved version of the StackGAN, which involves connecting three Generators to one Discriminator.

Attention mechanisms work by assigning larger weights to valuable parts of an image and assigning lesser weights to not-so-important parts of the image. This is done to allow the network to focus its attention on specific parts of the input image. AttnGAN [20] is one such GAN that uses an attention mechanism and is built upon the StackGAN++. The attention mechanism allows the model to generate fine details of the image based on the relevant words and the vector of the complete sentence.

B. T2I Methods with Additional Supervision

The previous section discusses Direct Text to Image models conditioned on only one text input. However, various models involve additional supervision. These models tend to perform better than the Direct Text to Image models. However, they do require additional annotations during training.

Usually, datasets contain more than one text description for each image (For instance, the Flickr8k dataset]). Hence, this led to the hypothesis that using multiple captions can provide the model with extra data, which could in turn help create better images. One GAN that uses this is the C4Synth [21], which uses a cross caption cycle consistency to use numerous captions. The cross caption cycle ensures that the synthetic image created by the GAN is a good representation of the texts that are similar to each other based on semantics. It iterates over all the captions present in the dataset for each image. It improves image quality by taking information from all these captions and using them to create a better-quality image. Another GAN which does something similar is the RiFeGAN [22]. It uses the images and all their respective captions as a knowledge base and extracts necessary items using a caption matching mechanism. Unlike the C4Synth [21], it does not use an image captioning network.

In 2018, Sharma et al. proposed ChatPainter [23] to leverage Dialog Data, believing that a single sentence may not have enough information to create a clear and accurate image of a scene that consists of numerous objects. They used the Visual Dialog dataset. This dataset is made up of ten question-answer conversations assigned to each dialogue and are paired with text from the COCO dataset. The following section discusses the most used evaluation metrics for Text to Image Synthesis using GANs. [63]

2.4 Commonly Used Evaluation Metrics for Text to Image Synthesis

Generative Adversarial Networks are known to be very difficult to evaluate. This difficulty is because, unlike other neural networks, GANs do not have an objective loss function [64]. A popularly used method of evaluating GANs is manual evaluation. However, this is a time-consuming and tiresome process. Therefore, access to robust methods that can be used for evaluation is vital. This led to the development of a set of qualitative and quantitative evaluation metrics.

The two most important things researchers look for when creating GAN models are:

1. Diversity of images
2. Realism of the images, i.e. how realistic the images are

While the above two factors are valuable evaluators for all images created by GANs, it is essential to remember that for Text to Image Synthesis; it is vital to analyse the level of alignment (Text-Image alignment) between the captions and the images generated by the GAN semantically. In other words, it is vital to look at whether the model can create realistic images that are a pictorial representation of the text entered to create the image.

a. GAN Metrics

Two of the most frequently used GAN metrics are Inception Score [25] and Fréchet Inception Distance [26]. Both these evaluation metrics use the Inception V3 model trained on the ImageNet dataset, which has 1000 classes. The Inception Score is a numeric metric that is used to measure how diverse the images generated by the GAN are as well as how distinctive each image looks from each other. On the other hand, the Fréchet Inception Distance evaluates the similarity between the real and the generated images by creating a distribution of the extracted features of both the real and the generated images and finding the Wasserstein-2 distance between them. Chapter 5 discusses the working and implementations of both metrics in depth.

However, the drawback of both these scores is that they rely on the pre-trained Inception V3 model. Suppose this model is applied to a class of images that are not present in the ImageNet dataset. In that case, it will give a low Inception Score of 1.0 and a very high Fréchet Inception Distance, implying that the model performs poorly despite it being able to create realistic images with lots of diversity.

Due to its increased consistency in evaluating GANs [46] and because FID compares the distributions of the dataset to the generated images, thus assessing the similarity between the real and the fake images (which the Inception score does not do), it is considered a better metric than the Inception Score. Besides this, it is also better at capturing disturbances in the images than the Inception Score [63]. Despite this, the FID has a significant drawback. As mentioned earlier, the FID extracts features from the real and generated images and finds the Wasserstein-2 distance between their distributions. However, it assumes that the features extracted by the model follow a Gaussian Distribution which is not always the case. Besides this, [26] has recommended that a minimum of 10,000 [70,71] samples be used to find the Fréchet Inception Distance else, the actual Fréchet Inception Distance of the Generator is overestimated and hence, cannot be considered a reliable metric [46].
Even so, the Inception Score and Fréchet Inception Distance are popular metrics for evaluating Generative Adversarial Networks and are still used today.

b. Image-Text Alignment Metrics

As mentioned earlier, besides the quality and diversity of the generated images, it is vital to assess the Image - Text alignment for Text to Image Synthesis models. The following are two popular methods of testing Image - Text Alignment.

R-precision [27] measures the similarity between text descriptions and the generated images by ranking the retrieval results between text features and images. Along with the main caption used to produce the generated image, extra captions are sampled from the dataset on a random basis. The cosine similarity between the image features and text embedding of each caption defines its rank after it is arranged in a descending order of similarity. Suppose the text that was used to generate the image is placed in the top ‘n’ captions, then it is a favourable or successful outcome. [63]

Captioning Metrics – This method uses an image caption generator to find captions for the generated image. Following this, the caption used to generate the synthetic image is compared with the caption outputted by the image captioning model.

2.5 Text to Image for Emoji Synthesis

As mentioned in earlier sections, Text to Emoji Synthesis can help provide users with the ability to create emojis on the fly. As of 2022, there is not much research on Text to Emoji Synthesis. This scarcity of research is because there is a lack of emoji data to train the model. Most GANs use over 10,000 images for training in order to produce good results. At present, there are only 3,633 emojis available for use. This lack of data is the reason for the plodding progress of the research in this field. Presently, only three papers have implemented Text to Emoji Synthesis. Two papers use Generative Adversarial Networks [1,2], and the remaining paper uses Transformers [28].
Both the papers that use GANs [1,2] use the conditional Deep Convolutional Generative Adversarial Network. The EmotiGAN [1] was researched in 2017 and was the first paper to implement Text to Emoji Synthesis. It was successfully able to regenerate the emojis from the raw dataset. Although the emojis depicted the features one would expect from the text input, the emojis were of lacking quality. It was also unable to create non-existent emojis from the text inputted by the user.

On the other hand, [2] also uses Deep Convolutional Generative Adversarial Networks like [1] but achieves better results. It successfully regenerated the emojis in the raw dataset, the difference being that they were of better quality and depicted the features one would expect from the text. While the basic architecture used for both the models is the same, the aspect that could have led to the improvement in results for [2] could be due to the author's claim that, at times, they have trained the generator twice as compared to the discriminator. Besides this, neither of these papers has given details of the optimiser or other hyperparameters used. A difference in these variables could be the reason for worse results generated by [1]. Although this model [2] has performed well in regenerating training emojis, it was unable to provide a similar result for the generation of the non-existent emojis from the text. They have implemented an arithmetic operation for the word vectors, where they averaged the word vectors of the texts of two emojis to create a non-existent emoji to see if that would have all the features of both the emojis they used. While features of both emojis did successfully end up in the new emoji, it did not happen realistically and has generated emojis of very poor quality.

Besides this, the model also lacks Text-Image Alignment. Both [1] and [2] have used Word Embeddings created by the Word2vec model to convert their text to a vector format before inputting it into the GAN as a conditional input.

Finally, [28] makes use of transformers to create non-existent emojis. This Russian model focussed on fine-tuning the pre-trained Ru-DALL-E model to create an architecture that can create emojis from the text. This model used the Adam optimiser and was able to achieve great results. Not only were they able to regenerate existing emojis, but they were also able to create non-existent emojis of excellent quality that looked as realistic as real emojis. The emojis created were also great representations of the text that generated them. Besides this, the model would create multiple outputs for a single text prompt, thus giving the user a greater choice of emojis.

Summary

To conclude, Text to Image Synthesis has a long history. Several models with varying architectures have been tried and have generated excellent results. However, the same cannot be said when it comes to Text to Emoji Synthesis. Text to Emoji Synthesis has not been researched at the same level. The reason for minimal research in Text to Emoji Synthesis is the scarcity of data. At present, only three models have implemented Text to Emoji Synthesis, from which only one paper which uses Transformers [28] was able to produce good results. The remaining two papers [1,2] used GANs and produced poor and mediocre results respectively.
For this project, the author has trained a modified version of the Deep Convolutional Generative Adversarial Network to improve the results generated by [1] and [2]. In general, DCGANs are considered to be capable of producing better results than standard GANs as they use convolutional networks and do not have any pooling or fully connected layers [2]. Hence, the author aims to implement modifications to the existing DCGAN model to improve the emojis created.

CHAPTER 3 : BACKGROUND RESEARCH

Introduction

This research focuses on Text to Emoji Synthesis using GANs and hence focuses on two aspects – the text used to generate the emojis and the GAN that creates the emojis. This chapter introduces the Natural Language Processing techniques, main algorithms and Deep Learning techniques used in this research. Section 3.1 briefly introduces Natural Language Processing and the various pre-processing techniques used to clean the captions before using them as conditional input to the GAN for the Text to Emoji synthesis. Further, it also introduces Word Embeddings and discusses the three Word Embedding algorithms used in this project. Section 3.2 then introduces Artificial Neural Networks to the reader. Finally, Section 3.3 talks about Deep Learning and provides background from basic models like the Multilayer Perceptron to more complicated models such as the CNN, UNet, RNN, LSTM, and GANs (specifically DCGAN and the conditional GAN which are used in this project).

3.1 Natural Language Processing (NLP)

Natural Language Processing is a subfield of Computing Sciences and AI [80]. It solves problems with the aim of decreasing the gap between electronic machines and the languages spoken by humans. This section discusses the multiple facets of Natural Language Processing implemented in this project. It details the popular text pre-processing techniques of Natural Language Processing used in this project. Further, it briefly introduces Word Embeddings and then delves deeper into the three Word Embedding algorithms used in this project (Section 3.1.2).

3.1.1 Text Pre-processing Techniques

Text Pre-processing is the first and most crucial step when working with models that deal with Natural Language Processing. It ensures that the text is in the most digestible form so that Machine Learning models can use the data easily. Mentioned below are some of the standard text pre-processing techniques used in NLP projects [29]:

1. Lower Case: All text used in NLP projects are converted to the same case (usually lowercase) to ensure that they are all on a level playing field. This is because the models tend to consider words like 'Books' and 'books' as two different words despite the fact that the two words have the same semantics and only differ due to the capitalising of the first letter. To ensure that the model considers capitalised or lowercase words, in the same manner, all captions are converted to lowercase.

2. Tokenisation: The sentences or phrases entered by the user are tokenised. This entails converting the sentences into tokens which are the words separated by white space— this aids in further cleaning of the data (Points 3 and 4 below).

3. Removal of Punctuation: Punctuations do not add any value for Text to Emoji synthesis tasks as they do not have semantic value. Hence, they are eliminated from the texts.

4. Removing Stop words: Stop words are words that occur very frequently in the text. Words such as 'a', 'the', 'an', and 'is' are the most common words in texts and hence do not add much value to the phrase or sentence. Hence, these are dropped from the sentences.

5. Removal of White Space: Sometimes, due to noise, there is extra white space before the first word of the text and after the last word. These are additional white spaces that do not add any value and are removed from the captions to ensure that they all start and end in the same manner.

The methods and libraries used to implement the Data Pre-processing steps mentioned above are detailed in Section 4.3.

3.1.2 Word Embeddings

Language Modelling and Feature Learning methods in NLP are collectively described by a term called Word Embedding [82]. Deep Learning models are not capable of dealing with words. They can only deal with data in a numerical format. Word Embeddings fulfil this by converting texts to vector data such that semantically similar words are assigned positions that are close to each other in the vector space.

There are two main types of Word Embedding algorithms – Predictive and Count-based [72]. Predictive algorithms predict a word or set of words from the given input. On the other hand, Count-based algorithms depict the target word based on the words that occur with the target word in a variety of contexts. The three Word Embedding algorithms used in this project have been detailed below.

1. Word2vec :

Word2vec is a Predictive Word Embedding model introduced in 2013 by Google [30] that can learn and represent words in a vector format. It is the standard for developing pre-trained Word Embeddings. It has a shallow neural network architecture (a single hidden layer) and follows two algorithms: Skip-gram and Continuous Bag of Words (CBOW). The Skip-Gram model takes a particular word as input and then predicts the surrounding words as the output. On the other hand, the continuous bag of words (CBOW) model predicts a word based on its surrounding words [73]. Figure 2 illustrates the CBOW architecture on the left and the Skip-gram architecture on the right.

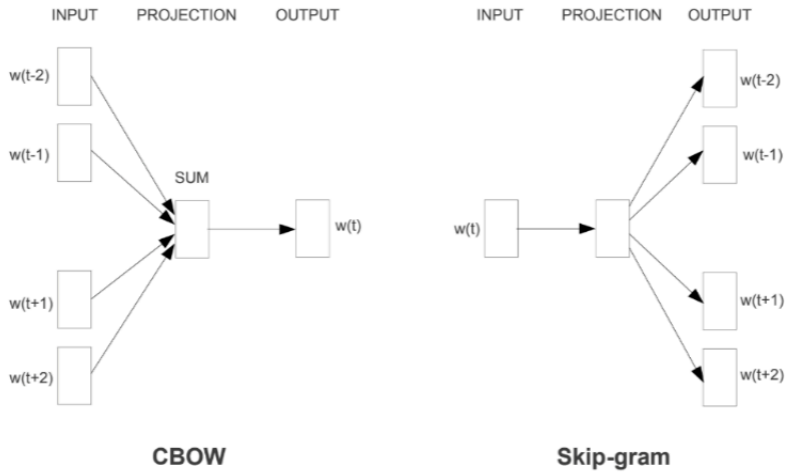


Fig 2 : Word2vec : CBOW architecture (left) and Skip-gram architecture (right)

2. Global Vectors for Word Representation (GloVe) :

Stanford University created Global Vectors for Word Representation or GloVe in the year 2014 [32]. It is an unsupervised, count-based algorithm that generates Word Embeddings. The output embeddings show the linear substructures of words in the vector space (Figure 3).

The co-occurrence (frequency of each word in some context) probability ratio between two words is encoded as vector differences. First, a large matrix is constructed that consists of co-occurrence information. The GloVe model works on the idea that ratios of word-to-word co-occurrence probabilities can encode meaning as vector differences. Hence, the objective is to learn word vectors such that the dot product of these vectors will be equal to the logarithm of the word's probability of co-occurrence. The ratios of co-occurrence probabilities is equal to the difference between the word vectors in the word vector space. This is because the logarithm of a ratio is equal to the difference of their individual logarithms. [74,75]

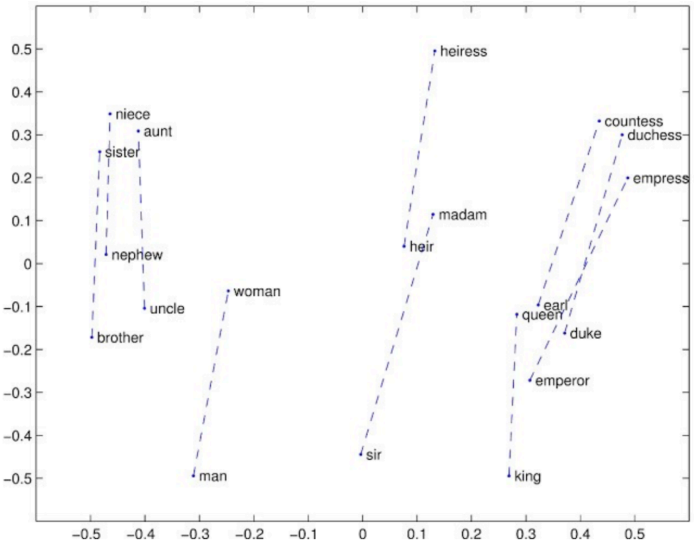


Fig 3 : GloVe : Relationships between similar words

3. FastText :

FastText [31] is a predictive Word Embedding model similar to Word2vec [30]. The difference between the two is that FastText works at a character level, whereas Word2vec works at a word level. The Skip-gram and CBOW architecture of Word2vec working at a character n-gram level is known as FastText.

Character n-grams are a set of co-occurring characters within a given window. An example of Character n-grams where n = 3 is shown in Figure 4. In the case of FastText, a word is represented by the sum of a set of character n-grams. Character n-grams allow the model to learn embeddings for morphologically rich languages like Arabic.
An example for this is : the word 'table tennis' translates to 'tischtennis' in German. In Word2vec, the model would learn the embeddings of tennis and tischtennis separately as two different words, lacking the ability to realise that the two words are in fact related to each other. However, with character n-grams, the probability of the model understanding that the two words are related increases. This is because the two words would share overlapping n-grams. It is also useful as we can use words that are not seen by the model. For example, if the word ‘foolish’ is not in our dataset but the word ‘fool’ is, the model will be able to understand that the two words are related due to their overlapping character n-grams. [76]

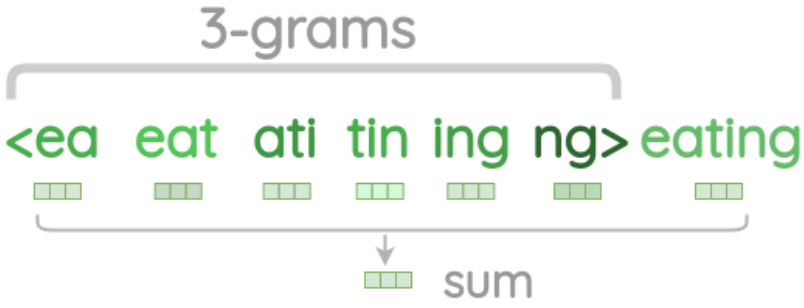


Fig 4 : FastText Character n-grams (n = 3)

All three Word Embeddings perform well, depending on the dataset and the task to be completed. However, FastText has the edge over GloVe and Word2vec due to the latter two method's inability to deal with words that are not part of the corpora. FastText uses character n-grams and can find relationships between words in and outside of the corpus.

3.2 Artificial Neural Networks (ANN)

An Artificial Neural Network (ANN) is a group of nodes that are connected with one another in an attempt to allow machines to carry out decision-making in a human-like manner. ANNs were created after taking inspiration from the human brain. Similar to the neuron of the human brain, the smallest unit of the ANN is the artificial neuron known as a node. ANNs can have between twenty to a million nodes arranged in a series of layers (Figure 5).

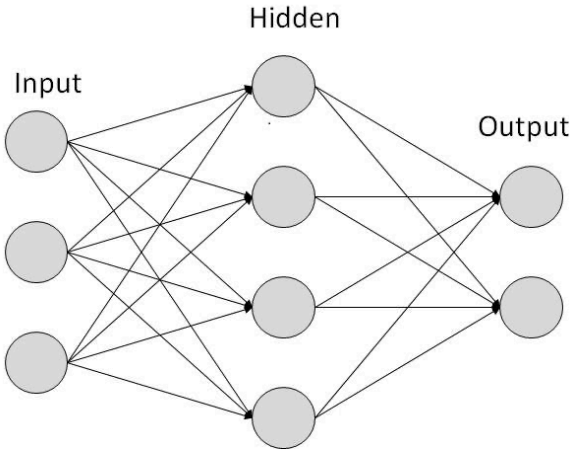


Fig 5 : Architecture of Artificial Neural Networks

All neural networks have three types of layers:

1. Input Layer: This is the first layer of the neural network. It takes in the external data inputted and passes it on to the rest of the network.
2. Hidden Layers: These are the layers that accept the external data from the input layer. ANNs can have one or multiple hidden layers. Various mathematical operations are carried out amongst these layers, and the last hidden layer's final output is transferred to the output layer at the end. Figure 5 illustrates an Artificial Neural Network that has only one hidden layer.
3. Output Layer: This layer holds the final output of the Neural Network

[77] Artificial Neural Networks can be used for various tasks and can reduce or eliminate the amount of manual work done by human beings. For instance, to build an image classifier that would take an image of a dog or a cat and classify it by giving it a label that holds the value 'Dog' or 'Cat', we feed the raw images of the animal to the Input layer, after which they get passed on to the hidden layers. The neurons in the hidden layers process the raw data. Figure 6 illustrates an artificial neuron, where X1 and X2 are the inputs and f(x) is the mathematical operation used to process the data.

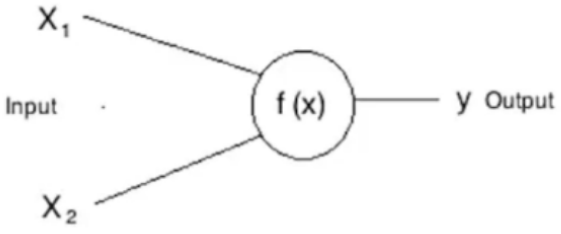


Fig 6 : Node/Artificial Neuron

These neurons will fire if the mathematical operation f(x) fulfils a condition. For instance, if f(x) represents a subtraction operation, and the condition states that the neuron will fire if its value exceeds N, then it will fire only if X1 - X2 (Figure 6) is greater than the threshold of N. Thus, the neuron makes a contribution to the final output on satisfying the condition. The negative of this threshold is called Bias.

X1 + X2 - Threshold > 0 <=> X1 + X2 + Bias > 0

All layers in an ANN have the same Bias. Each connection between the neurons has a Weight. This Weight differs based on the importance of a particular neuron's value on the output.

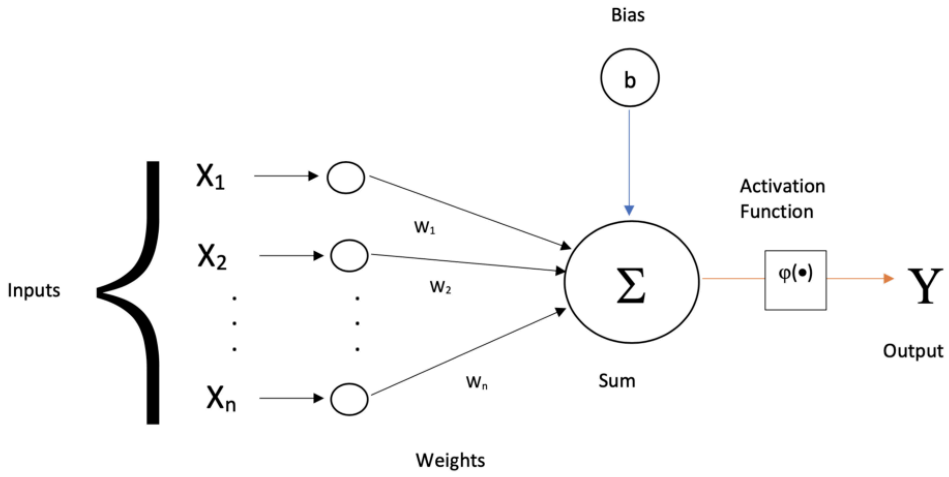


Fig 7 : Mathematical Computation of a Neuron

Figure 7 illustrates a neuron whose output will be the sum of the Bias and the product of the inputs and their Weights. Equation 3.1 (below) represents this:

$$output = X1W1 + X2W2 + ... + XnWn + b \quad (3.1)$$

As visible from the illustration in Figure 7, this output is passed through an activation function to give the final Output 'Y'. Activation functions transform the weighted sum of the inputs and the Bias to an output value passed on to the next layer. This sum (Equation 3.1) is a linear equation. Therefore, regardless of the number of hidden layers the researcher or programmer adds to their network, the model will not be able to carry out complicated tasks. Activation functions add value to the neural network by adding nonlinearity to the function. Choosing an appropriate activation function depends on the task. It is vital as it decides whether a neuron should be activated, i.e., whether the neuron's Input is important and makes a difference to the prediction process. There are various types of activation functions. The following are the most used activation functions: sigmoid, Tanh, ReLU and Leaky ReLU. Usually, all layers have the same activation function. However, depending on the task the neural network tries to fulfil, the final layer is assigned a different activation function.

This algorithm calculates the outputs of each node as the data passes through the layers. This is done until the final output is obtained from the output layer. It is called Forward Propagation. This output is then compared with the true output to give the neural network loss. In the case of the image classifier, the model's predictions are compared with the true labels of the image to return the loss. This loss is then propagated backwards through the network to change the weights between the nodes of the neural network to fine-tune the model to reduce the loss as much as possible.

3.3 Deep Neural Networks (DNN)

A Deep Neural Network is an Artificial Neural Network with an architecture that is several layers deep. The network can have multiple hidden layers but must have a minimum of two hidden layers. These networks can carry out complex tasks and deal with large datasets.

Some of the most popular Deep Neural Networks are described in the upcoming sections :

3.3.1 Multi-Layer Perceptron (MLP)

Multi-Layer Perceptron, otherwise known as Feed Forward Neural Network, is the most basic Deep Neural Network. Its architecture consists of a series of fully connected layers of neurons. In fully connected layers, each neuron of each layer is connected to every neuron of the adjacent layer.[83] This is illustrated in Figure 8 (below).

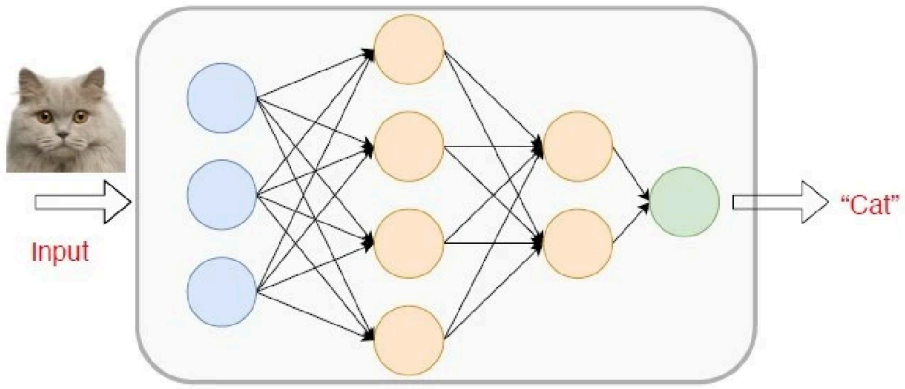


Fig 8 : Multi-layer Perceptron

3.3.2 Convolutional Neural Network (CNN)

[81] Convolutional Neural Networks are a top-rated class of deep neural networks. This is due to their excellent performance when dealing with audio data as well as various computer vision tasks. Some popular applications are classification, object detection and semantic segmentation. When given an image, CNNs are capable of feature extraction that makes computer vision tasks possible. They have three main types of layers:

1. Convolutional Layer: This is the core building block and the first layer of the CNN. A feature detector called a Kernel or a Filter will move across the input image. Following this, the dot product of the input pixels and the filter are taken. This process is called convolution and is used to find the features of an image.

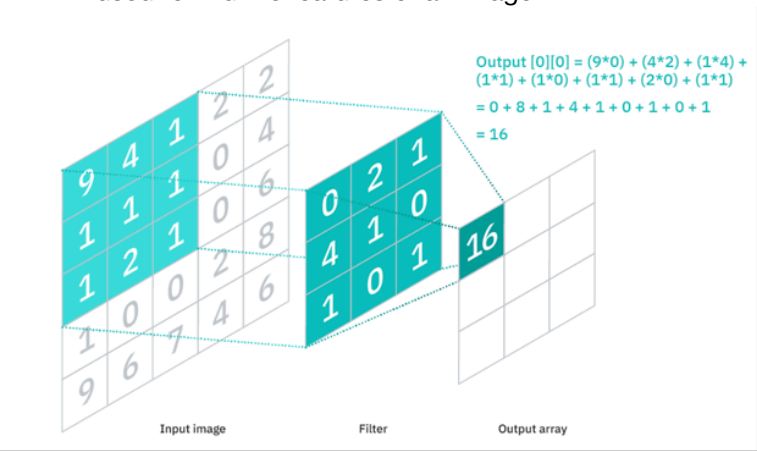


Fig 9 : Convolution of input image with filter

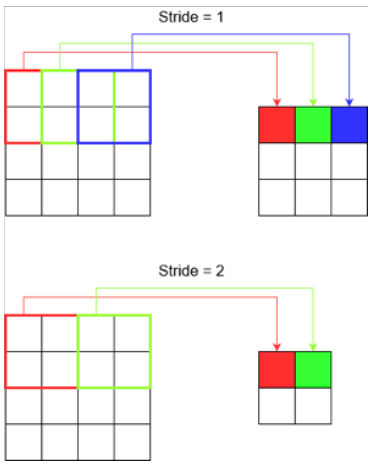


Fig 10 : Stride

The first round of convolution is carried out between the filter and a section of the image called the Receptive Field (Figure 9). The filter is then shifted based on the stride, which the programmer decides. Stride defines the number of pixels the kernel shifts over after completing one round of convolution with the receptive field before moving on to the next section. The new receptive field is then convolved with the filter (Figure 10). This process takes place until the filter has swept across the entire image.

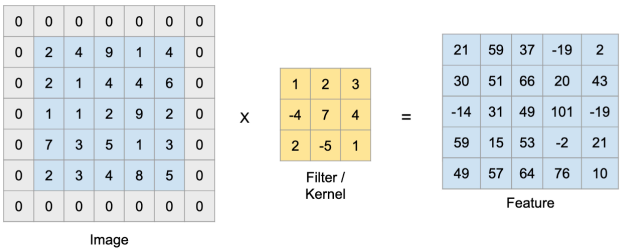


Fig 11 : Zero Padding

At times, the filter may not fit the image. Figure 11 illustrates such a case, where a 3x3 filter cannot fit a 5x5 image with a minimum possible stride of 1. In such cases, zero padding is applied to the input image. Zero padding involves the addition of a layer of zero elements to the input image to make the kernel fit the input image, producing an equal or larger-sized output.
After each convolution, the ReLU activation function is applied to add nonlinearity to the model. The convolutional layer can be followed by another convolutional layer or the pooling layer, which is detailed below.

1. Pooling Layer: The pooling layers, otherwise known as Down Sampling layers, are responsible for reducing the number of parameters of the input. Like the convolutional layer, the pooling layer sweeps a kernel across the output of the convolutional layer. However, the difference here is that the kernel does not have any weights. Instead of the weights, it applies an aggregation function to the input values. Two pooling layers are mainly used in neural networks. These are detailed below:

    a. Max Pooling: Max pooling is where the filter picks the maximum value of the part of the input image that the filter sweeps over. (Figure 12 Left)
    b. Average Pooling: Average pooling is where the filter calculates the average of the values of the input image that the filter sweeps over. (Figure 12 Right)

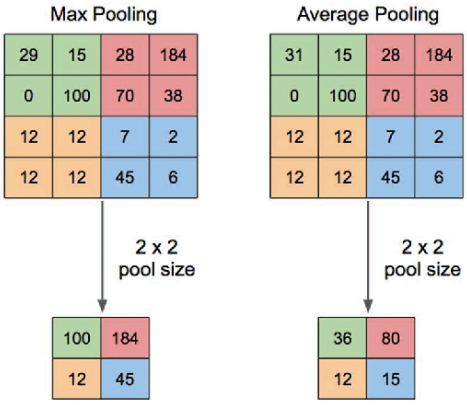


Fig 12 : Max Pooling (left) and Average Pooling (right)

Although the pooling operation does mean that some of the information from the input will be lost, it helps by improving efficiency and reducing overfitting.

2. Fully Connected Layer: The Fully Connected Layer is used if we want our CNN for classification purposes. After the final convolution or pooling layer, it's output is flattened and sent into a fully connected layer. The fully connected layers are simply Feed Forward Neural Networks, which usually have a softmax function that allows the classification of the inputs by producing a probability between 0 and 1.

In general, CNNs are preferred over Multilayer Perceptrons as the latter tends to be quite expensive due to its full connectivity and, hence they have too many weights/parameters. On the other hand,

CNNs are computationally efficient as they use convolutional and pooling layers that perform parameter sharing as they are only 'partially connected' layers. CNNs have been successful in various applications related to computer vision. Below listed are three of the most common and popular applications where CNN has been used :

a. Classification

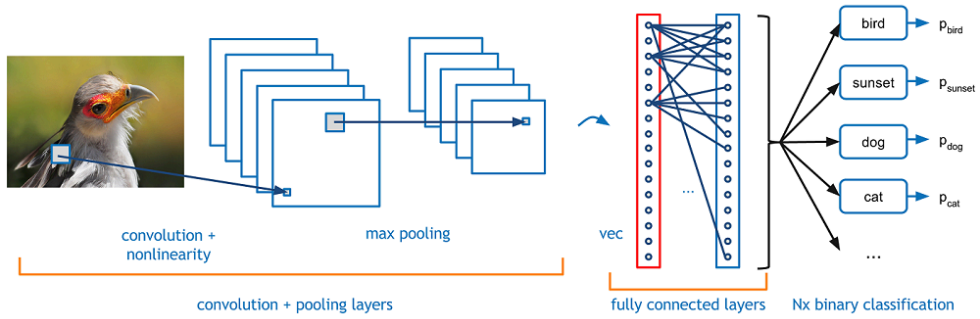


Fig 13 : Classification

As mentioned in earlier sections, to classify images, the input image goes through a series of convolutional and pooling layers to learn the features of the input image and then downsample them to reduce overfitting and complexity. Following this, as illustrated in Figure 13, the output of the convolutional/pooling layers goes through a series of fully connected layers (as mentioned in the earlier section) that represent the output into probabilities which predicts the class of the image. A popular CNN used for image classification is the VGG-16 [37].

b. Segmentation

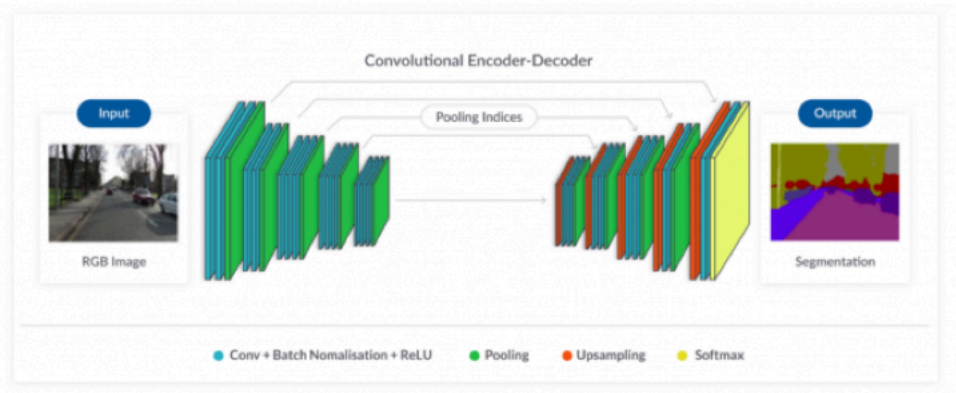


Fig 14 : Segmentation

Image Segmentation is a deep learning task where the model classifies the pixels of the image into various categories. The CNN architecture for segmentation consists of encoder-decoder models (Figure 14), where the encoder is responsible for encoding the image into a form that is sent through the network. Encoding allows the model to learn the features of the image. The decoder then converts it back into its original representation.

UNet: Figure 15 (below) illustrates a modification of the CNN called the UNet [36], commonly used for image segmentation purposes. The standard CNN architecture was first used for image segmentation. Although it gave decent results for simple segmentation problems, it did not work well for more complicated image segmentation problems. This is where the UNet came into the picture. It was designed for medical image segmentation but has, over time, been used in several other fields.

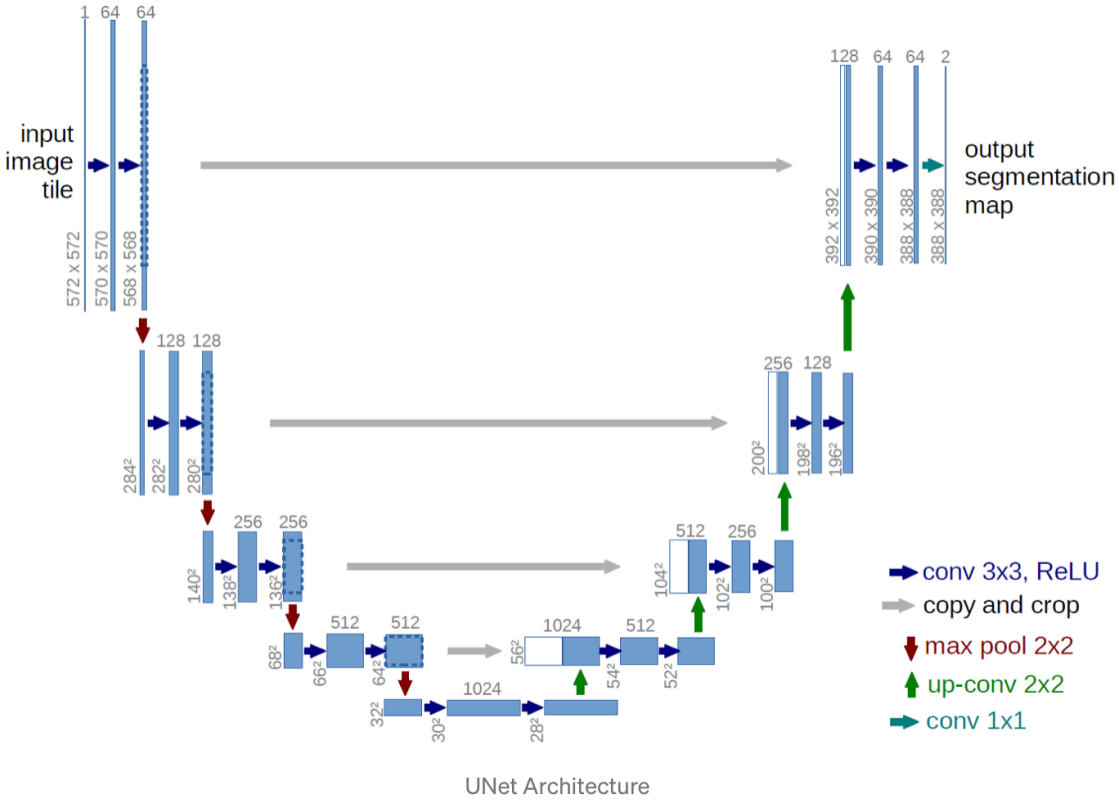


Fig 15 : UNet

As visible from the illustration in Figure 15 (above), the architecture gets its name from the 'U' shape it has. This architecture has three sections:

1. Encoder (Contraction): The encoder part of the UNet consists of multiple contraction blocks where each contraction block has two convolutional layers followed by a layer of max pooling. After each block, the number of kernels doubles to allow the model to learn complex features.
2. Bottleneck: The Bottleneck mediates between the encoder and the decoder, and it does this by using two convolutional layers followed by one upsampling layer.
3. Decoder (Expansion): This is the final section of the UNet architecture. Unlike the encoder, the decoder consists of several expansion blocks. Each block consists of two transposed convolutional layers followed by one upsampling layer. The number of expansion blocks is always the same as the number of contraction blocks. Following this, the result is passed through another convolutional mapping where the number of feature maps is the number of segments required.

As visible in Figure 15, the layers in the contraction section (encoder) of the UNet are connected and concatenated with the layers in the expansion section (decoder) of the UNet through skip connections. These skip connections allow the UNet to use fine details that it learns during encoding in the decoder part to reconstruct the image.

c. Object Detection

Object detection involves two concepts: Localisation as well as Classification. Object detection models are trained to find and classify multiple objects that appear in an image. Some of the most popular object detection models are R-CNN (Figure 16), Fast R-CNN, and YOLO, all of which use CNNs to carry out the task.

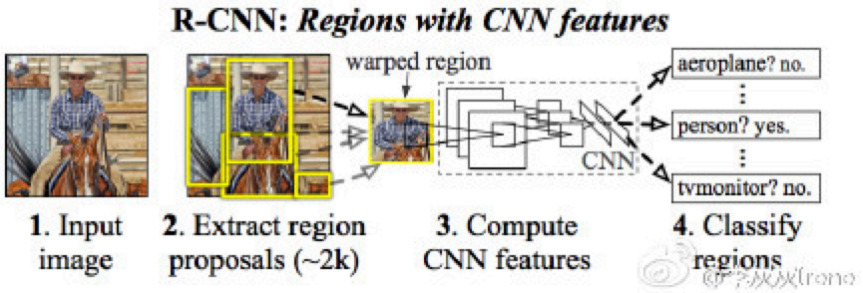


Fig 16 : Object Detection

3.3.3 Recurrent Neural Network (RNN) [78]

Recurrent Neural Networks work with speech and text data used in Natural Language Processing. These networks use sequential or time series data to predict what comes next. They are useful when the context is vital in predicting the outcome. The factor differentiating them from other DNNs is that they have feedback loops that process a sequence of data by maintaining a history/memory, thus allowing them to retain previous information and use this context to make more accurate predictions.

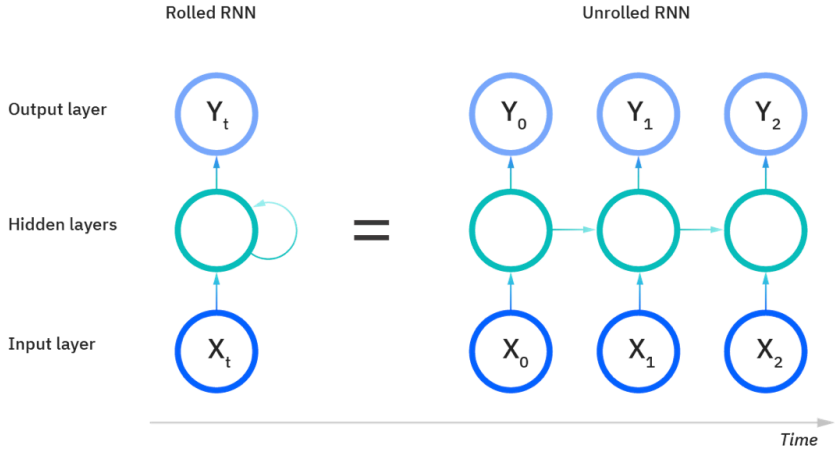


Fig 17 : Recurrent Neural Networks

Figure 17 illustrates the feedback loop Recurrent Neural Networks use that helps them maintain previous data in their memory. It uses this to make predictions. These networks have a hidden state, 'h_t' which is updated each time a sequence is processed. Each output Y is created by processing the input data at time t and the memory from t-1 which is h_{t-1} (Equation 3.2).

$$y_t = f(X_t , h_{t-1}) \tag{3.2}$$

The hidden state at time t, h_t is updated using the current input at time t and the hidden state at time t-1. (Equation 3.3)

$$h_t = f(X_t , h_{t-1}) \tag{3.3}$$

Another factor that differentiates Recurrent Neural Networks from other neural networks is that, in the former, the weights are shared across all parameters. Three sets of weights are defined in the beginning, and it remains the same throughout the network.

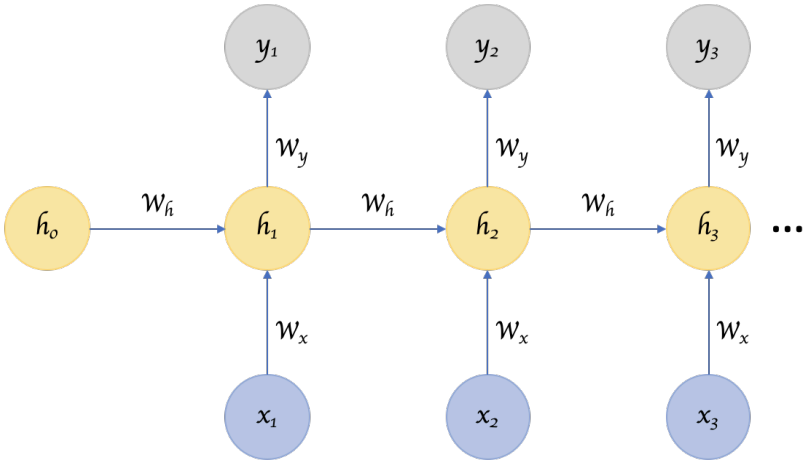


Fig 18 : Weights of RNNs

Figure 18 illustrates the three weights of the RNN that are shared across all its cells. The three weights are W_x, W_h and W_y. The weights are used to obtain the hidden state h_t (Equation 3.4) and the output y_t (Equation 3.5) at the time 't'. Often, people use tanh as an activation function to add nonlinearity as it produces better results.

$$h_t = \tanh(W_h^T h_{t-1} + W_x^T X_t) \quad (3.4)$$

$$y_t = W_y^T h_t \quad (3.5)$$

Once the predictions are made, the loss functions can be defined through backpropagation. The weights can then be altered to minimise the loss of the model.

3.3.4 Long Short-Term Memory Networks (LSTM) [84]

Standard RNNs face two problems, vanishing gradients and exploding gradients. RNNs tend to fail in the presence of significant time lags, i.e. the range of information that a standard RNN can analyse to create a context to make a good prediction is limited in practical situations. The Long Short-Term Memory Networks solve these issues.

[84] Long Short-Term Memory Networks or LSTMs were introduced in 1997 [6]. They proposed an architecture which involves replacing the RNN cell with a memory cell. This version of the LSTM was later modified in 1999 [7] to include a Forget gate. The LSTMs we know now use gates to control the memorising process, thus improving their ability to track information through many time steps. Gates are structures that optionally allow data to pass through. Figure 20 (below) illustrates the structure of a gate. It comprises a sigmoid neural net layer and a point-wise multiplication operation. The sigmoid layer outputs numbers between 0 and 1 that signal the amount of information to be kept and forgotten.

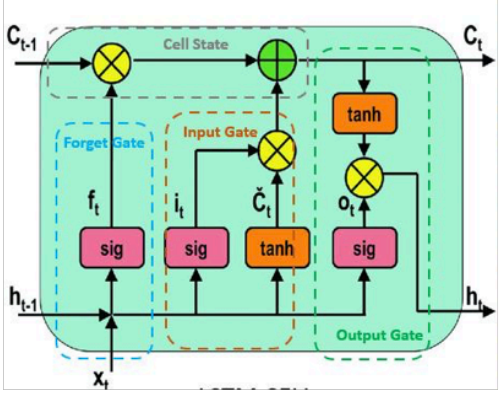


Fig 19 : LSTM Cell

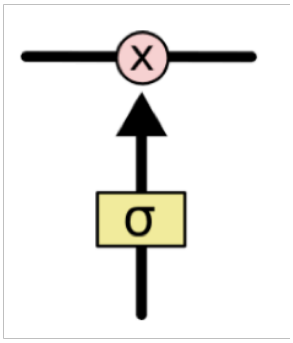


Fig 20 : Gates

Figure 19 (above) illustrates the inner workings of a single LSTM cell. The LSTM uses four gates, namely - Forget Gate, Store Gate, Update Gate and Output Gate. Below, the workings of the four gates are explained in more detail.

1. Forget Gate: The Forget Gate is responsible for deciding what information from the cell state is to be kept and forgotten.

$$f_t = \sigma(W_{fh}{}^T h_{t-1} + W_{fx}{}^T x_t) \in [0,1] \tag{3.6}$$

Equation 3.6 represents the inner workings of the Forget Gate where h_{t-1} represents the hidden state and C_{t-1} represents the cell state of the previous time step 't-1'. The cell state controls the flow of information. It contains information that may not be important to the current time state 't' but needs to be transmitted through time. The Forget Gate checks the input 'x_t' as well as the previous hidden state 'h_{t-1}' and applies a sigmoid function which outputs 0 and 1 for each number in the cell state C_{t-1}. A 1 represents that the information must be kept, whereas a 0 represents that it must be forgotten.

2. Store Gate: The Store Gate is responsible for deciding the new information that must be stored in the cell state.

$$i_t = \sigma(W_{ih}{}^T h_{t-1} + W_{ix}{}^T x_t) \in [0,1] \tag{3.7}$$

$$C_{tbar} = \tanh(W_{ch}{}^T h_{t-1} + W_{cx}{}^T x_t) \in [-1,1] \tag{3.8}$$

It consists of two parts. The first part is called the 'input gate layer', which decides the values that will be updated. The next part is a tanh layer that outputs prospective values (C_{tbar}), which are added to the cell state.

3. Update Gate: The Update Gate is responsible for updating the previous time state C_{t-1} into the new cell state C_t.

$$C_t = f_t \times C_{t-1} + i_t \times C_{tbar} \tag{3.9}$$

Here, we multiply the values we decided to forget, f_t (which we found earlier using Equation 3.6), with the previous cell state to forget it. It is then added to the new candidate values found in the Store Gate by multiplying i_t and C_{tbar}.

4. Output Gate: The output gate controls what is sent to the next time step. The output is a 'filtered cell state', C_t.

$$o_t = \sigma(W_{oh}{}^T h_{t-1} + W_{ox}{}^T x_t) \in [0,1] \tag{3.10}$$

$$h_t = o_t \times \tanh(C_t) \tag{3.11}$$

This involves running it through a sigmoid layer, which tells us what parts of the cell state we will output (Equation 3.10). Further, the cell state is passed through a tanh layer which normalises the values between -1 and 1. These two are multiplied (Equation 3.11) so that the parts we want to output are outputted.

RNNs and LSTMs can be used for tasks such as Sentiment Classification, Machine translation, Music Generation and Image Captioning [38].

3.3.5 Generative Models

Generative models are Deep Neural Networks trained to generate data. A generative model describes how a dataset is generated using a probabilistic model. Further, sampling from this probabilistic model leads to the generation of new data.

Training a generative model is done via density estimation.

$$g : Z \longrightarrow X$$

Here, g represents the generative model, which, given the latent characteristics Z, can generate new realistic data X.

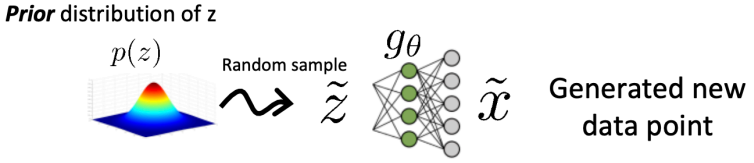


Fig 21 : Generative Models

Figure 21 illustrates the prior distribution P(z) from which a random sample z is chosen and passed through the generative model returning a new data point x.
Suppose the training data follows a probability distribution $P_{data}(x)$. We can then create a generative model having parameters θ such that, for an input x, it will have a probability distribution of $P_\theta(x)$. The goal of the generative model is to learn parameters θ for all the data such that

$$P_\theta(x) \sim P_{data}(x)$$

This ensures that the new data created is as realistic as possible and carries features of $P_{data}(x)$ while the generative model has a minimal loss.

3.3.5.1 Generative Adversarial Networks (GAN)

Generative Adversarial Networks (GANs) are a new class of generative models created in 2014 [11]. They have made a significant impact in the world of machine learning due to their ability to create new data instances that are similar to the training data.

The GAN starts from a prior, a standard normal gaussian with zero mean and a standard deviation of one. The main goal of the GAN is to create realistic images from data sampled from this prior distribution. To accomplish this, a Generative Adversarial Network comprises two neural networks:

1. Generator : The Generator is the neural network that creates fake images from the samples taken from the prior distribution.

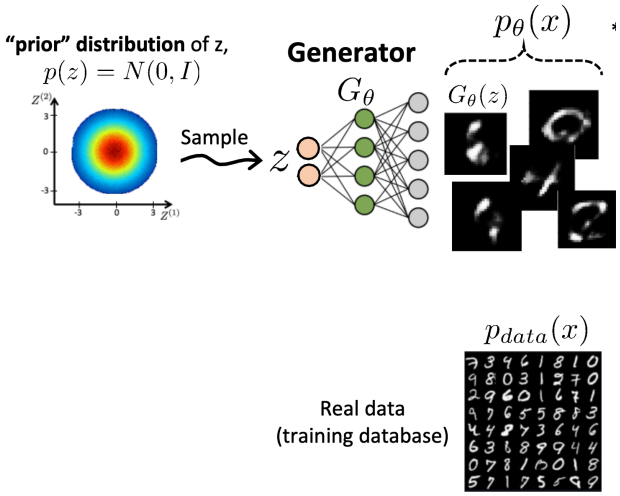


Fig 22 : Generator of the GAN

Figure 22 illustrates the Generator of the GAN. Here, P(z) represents the prior distribution from which we sample 'z'. The sample is passed through the generator network G_θ, which acts as a decoder to create realistic fake images. 'θ' represents the parameters of the Generator. As mentioned in Section 3.3.5, the goal of generative models is to create realistic images by ensuring that the probability density function of the model is as similar as possible to the probability density function of the training data. The discriminator aids in accomplishing this goal.

2. Discriminator : The discriminator, D_Φ, is a binary classifier (Multilayer Perceptron) trained to differentiate whether the input is a real data point from the training dataset or a fake data point created by the generator.

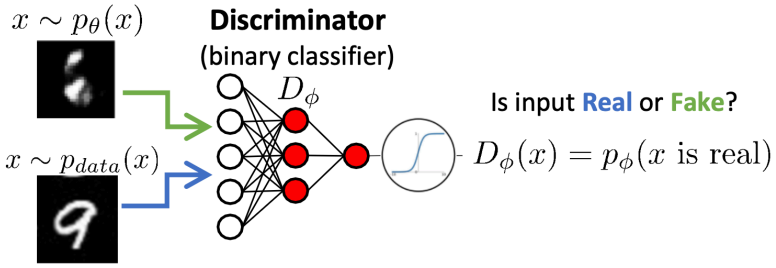


Fig 23 : Discriminator of the GAN

Figure 23 depicts the discriminator of a GAN. $D_\Phi(x)$ is the output of the discriminator that predicts the probability of the input being real. The discriminator's loss must be minimised to differentiate between real and fake images accurately.

Suppose $D_\Phi(x) = 1$; the discriminator is certain that the input is real. On the other hand, if $D_\Phi(x) = 0$, the discriminator is certain that the input is a fake/generated image. In this way, for each input, it predicts the probability of the input being real. Since the outputs must be probabilities, a sigmoid function is used over the activation function of the MLP. Through training, the discriminator learns a decision boundary between real and fake samples.

To maximise the accuracy of the discriminator, we need to find the optimal value of Φ, such that it will accurately classify the real and fake images. This is done by maximising the terms in the objective function below (Equation 3.12).

$$\Phi' = \arg\max_{\Phi} E_{x \sim Pdata(x)}[\log D_\Phi(x)] + E_{x \sim P_\theta(x)}[\log(1 - D_\Phi(x))] \quad (3.12)$$

This equation has two terms that represent two possible cases. The first case is when the input x is from the training database. In this case, the model should be maximising the expected value of the logarithm of the probability of the discriminator. Since logarithm is a monotonically increasing function, it means that to maximise the log, we need to maximise $D_\Phi(x)$, i.e. when sampling from the training database, we should be maximising $D_\Phi(x)$ to make it 1. On the other hand, the second term says that if we are sampling from the model, the model should be maximising $\log(1-D_\Phi(x))$, i.e. we have to minimise $D_\Phi(x)$ and push it to zero.

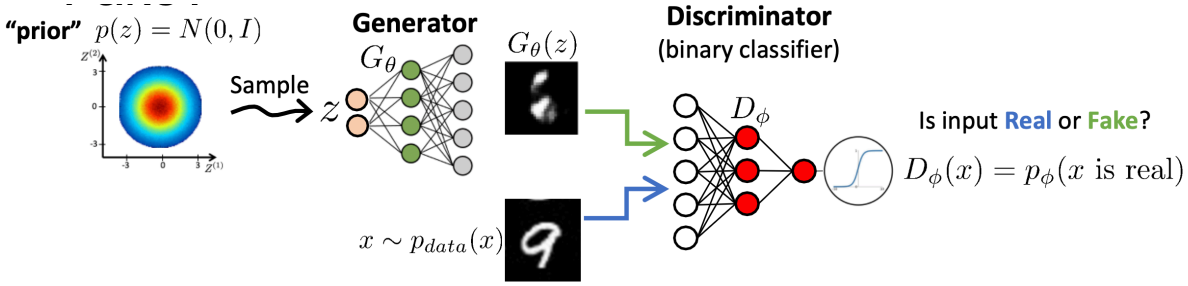


Fig 24 : Architecture of the GAN

Therefore, the generator and the discriminator have two opposing goals. The generator's goal is to create realistic fake images so that the discriminator will misclassify them as real images. On the other hand, the goal of the discriminator is to accurately classify the images as real if they are raw images from the dataset or fake if they are images created by the generator.

Since the goal is to have the generator create good images, we need to minimise the loss of the generator. However, as Figure 24 shows, the generator is not directly connected to the loss at the end, the discriminator is. However, the generator is connected to the discriminator. Hence, this issue is solved using backpropagation. The loss is back propagated back to the generator to change its model parameters such that the generator makes such realistic images that the discriminator is unable to differentiate between the real and fake images.

The objective function above can be modified to become the equation below (Equation 3.13) :

$$\Phi' = \arg\min_{\Phi} E_x \sim P_{data}(x)[-\log D_\Phi(x)] + E_z \sim P(z)[-\log(1 - D_\Phi(G_\theta(z)))] \quad (3.13)$$

These changes can be made because x in the second term represents fake data. In that case, $x = G_\theta(z)$, which is data created by the generator. The function has been changed from a maximising function to a minimising one by making all the terms in the objective function negative. The second term is a function of θ, which is a parameter of the generator. Hence, we can backpropagate the gradients back to the generator and change the generator's parameters.

$$\{\theta',\Phi'\} = \arg\min_{\theta}\max_{\Phi} E_x \sim P_{data}(x)[\log D_\Phi(x)] + E_z \sim P(z)[\log(1 - D_\Phi(G_\theta(z)))] \quad (3.14)$$

The equation above (Equation 3.14) represents the Two Player Min-Max Game for Optimisation. It states the following:

1. For a fixed generator, the discriminator tries to maximise the accuracy of separating the real from the fakes.
2. For a fixed discriminator, the generator tries to minimise the discriminator's accuracy in separating the real and fake images.

The discriminator changes its parameters to maximise the above objective function, whereas the generator changes its parameters to minimise the objective function. Only the second term in the objective function is a function of θ, so while training the generator, we drop the first term of the objective function.

$$\theta' = \arg\min_{\theta} (1/N_g)\Sigma_i[\log(1 - D_\Phi(G_\theta(z_i)))] \quad (3.15)$$

Equation 3.15 is a representation of this. Here, N_g represents the batch size used when dealing with the practical usage of a GAN. The discriminator loss will include both terms of Equation 3.14 and will have a negative sign as it aims to maximise the objective function as opposed to the generator which minimises the objective function
Therefore, the equations below represent the theoretical loss of GANs.

$$L_D(\Phi,\theta) = -1/N_{D,r}\,\Sigma_i[\log D_\Phi(x_i)] - (1/N_{D,f})\Sigma_i[\log(1 - D_\Phi(G_\theta(z_i)))] \quad (3.16)$$

$$L_G(\Phi,\theta) = (1/N_g)\Sigma_i[\log(1 - D_\Phi(G_\theta(z_i)))] \quad (3.17)$$

where, $L_D(\Phi,\theta)$ represents the discriminator loss, $L_G(\Phi,\theta)$ represents the generator loss and $N_{D,r}$ and $N_{D,f}$ represents the batch size of real and fake samples respectively.

$$\theta \longrightarrow \theta - \frac{\beta \partial L_G(\Phi,\theta)}{\partial \theta}$$

In order to optimise the generator to create realistic data, the generator loss is then backpropagated through the discriminator back to the generator to change its parameters.

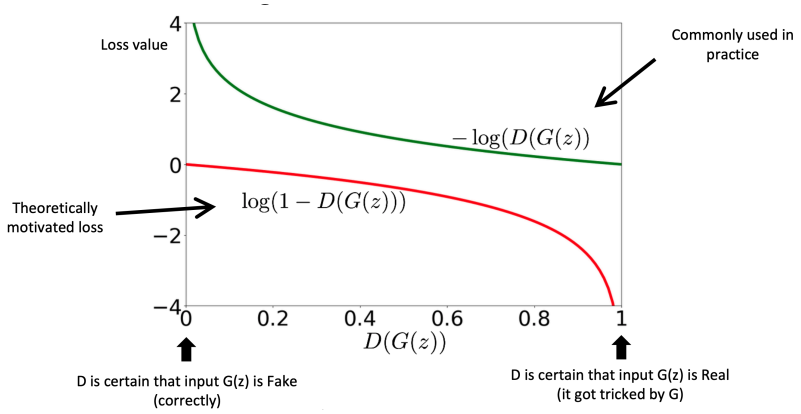


Fig 25 : Theoretical VS Practical Loss of a GAN

Training a GAN involves training the generator and discriminator simultaneously. When we train one, the other is kept constant. GAN training takes place in alternating periods. First, the discriminator is trained for one or more epochs. Following this, the generator is trained for a couple of epochs, after which the cycle continues. Hence, since the generator is not trained, it creates terrible samples during the beginning of training. Since the discriminator has already been optimised, it can easily differentiate between real and fake samples. Therefore, at the beginning of training, the loss will be on the left side of the red loss curve (theoretical loss) illustrated in Figure 25. However, that area is almost flat and has a small gradient. Therefore, the model will not be able to learn well from bad samples. Hence, in the beginning, small gradients are being backpropagated to the generator, due to which it learns very slowly. Following the same concept, if the generator was to create good samples, the discriminator gets confused and assigns it a probability of 1. In such a case, we would like the loss to saturate as we no longer want the model to change the way it creates data as the sample is already good and capable of fooling the discriminator. However, in the theoretical graph, the gradients are large when the probability is near 1, i.e. the generator can fool the discriminator. Hence, the loss function is modified as below.

Modified Loss function:

$$L_G(\Phi,\theta) = (1/N_g)\Sigma_i - \log D_\Phi(G_\theta(z_i)) \tag{3.18}$$

Using the modified loss function (Equation 3.18) above solves this issue. The loss curve would follow a trend similar to the green curve illustrated in Figure 25, and the model can learn well from poor samples as they have larger gradients and will not learn much from the good samples as the curve saturates off.

After training, the generator can map any point z from the prior P(z) to create a realistic output x = G(z) where all G(z) form $P_\theta(x) \sim P_{data}(x)$.

Through a lot of research, different types of GANs have been invented, which work better in some applications than others, such as the DCGAN [4], Pix2Pix [33], CycleGAN [34], StyleGAN [35] and so on.

3.3.5.2 Deep Convolutional Generative Adversarial Networks (DCGAN) [79]

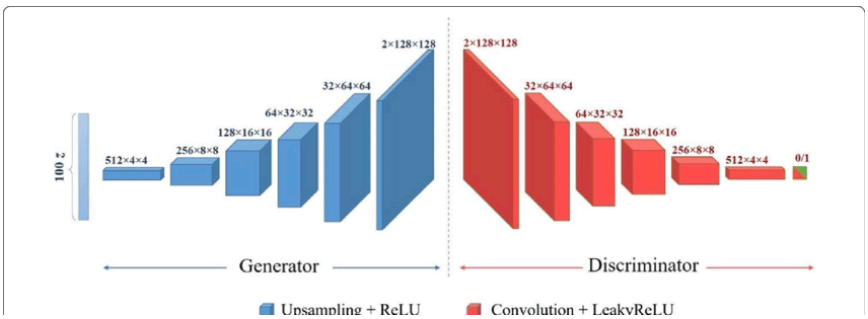


Fig 26 : Deep Convolutional Generative Adversarial Networks (DCGAN)

Over time, the GAN was researched more and advanced further to create the Deep Convolutional Generative Adversarial Network (DCGAN) in 2016 [4]. Deep Convolutional Generative Adversarial Networks have the same function as the GAN - it generates new, realistic data. Figure 25 shows the architecture of the DCGAN, which is quite different from that of a standard GAN.

1. All pooling layers of the standard GAN are replaced with strided convolutional layers for the discriminator and transposed convolutions for the generator.
2. Batch Normalisation is used in both the generator and the discriminator. Batch normalisation is a technique used to standardise or normalise the inputs to a network. It has been found to improve the performance of neural networks and accelerate training.
3. Vanilla or Standard GANs rely on feed-forward networks. All these fully connected layers are removed in the DCGAN. As shown in Figure 26, DCGANs have multiple convolutional layers that upsample the vector to higher and higher dimensionality until it reaches the dimensionality of the image in the generator. In contrast, the discriminator takes the real and fake images and passes them through a series of convolutional and downsampling layers before passing them through a fully connected layer to distinguish between the real and the fake images.
4. Uses ReLU as the activation function for all layers in the generator except the last layer where we use tanh.
5. LeakyReLU has been used as the activation function for all layers in the discriminator.

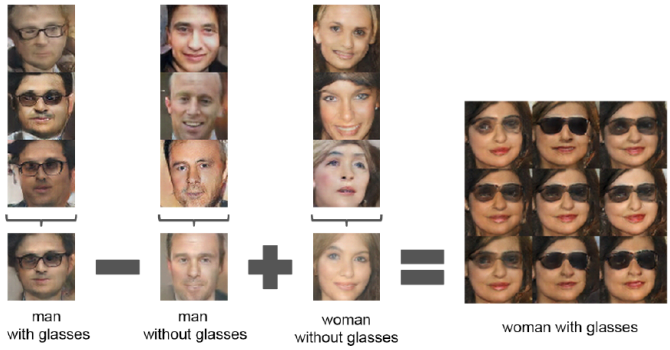


Fig 27 : Interpolation with DCGAN

Interestingly, DCGANs also have the ability to create outputs that show features of 2 or more input images through interpolation. This is illustrated in Figure 27, where the model can learn various features of the images and create new images.

3.3.5.3 Conditional Generative Adversarial Networks (cGAN)

Conditional Generative Adversarial Networks, otherwise known as cGANs, were introduced in 2014 [16]. They are a class of Generative Adversarial Networks which generates new data based on a particular condition.
As mentioned in section 3.3.5.1, GANs use a generator network to create new images and a discriminator network to distinguish real from synthetic images created by the generator.
In conditional GANs, a condition such as class labels or data is enforced upon the GAN. This means that both the generator and the discriminator are conditioned on these labels, which gives the model the advantage of creating specific outputs that the user wants by simply inputting the label of a particular class.

For instance, in the case of hand-written (MNIST) images, standard GANs can produce a wide variety of outputs, and we cannot control what output we want. However, with the conditional GAN, by assigning groups of images of the same numbers with labels ranging from 0 to 9, it can generate output images based on the user's request by simply assigning the label of the image they want the model to generate.

This is done by adding an extra layer as an input which would contain the labels encoded in a vector format. Popular methods of encoding these labels into vectors have been outlined in the next section

(Section 3.4). This additional layer guides the generator to generate examples that match a specific label. As illustrated in Figure 28 (below), these labels are also inputted to the discriminator so that it learns to distinguish fake image-label pairs from real image-label pairs. Hence the discriminator in a conditional GAN is trained to reject mismatched pairs and fake image-label pairs and only accepts real image-label pairs.

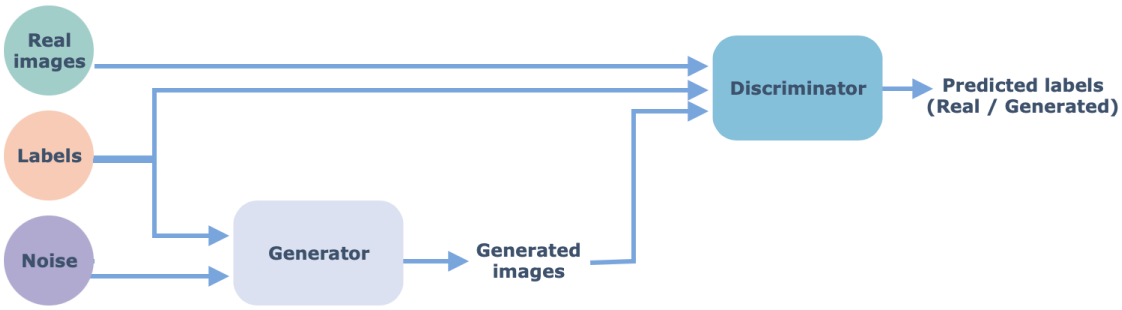


Fig 28 : Conditional GANs

Conditional GANs are useful for various applications. A few of them are listed below:

1. Text to Image Synthesis
2. Video Generation
3. Image-to-Image Translation
4. Convolutional Face Generation

Summary

To summarise, this chapter introduced some important terminologies and machine learning models used in this research. It starts by listing the popular pre-processing techniques in NLP-related projects and the Word Embedding algorithms implemented in this project. It then goes on to introduce ANNs, DNNs and talks about specific deep learning models such as UNets and LSTMs. Finally, it talks about GANs, DCGANs and Conditional GANs. The next Chapter gives an insight into the Methodology and implementation of the model used in this project.

CHAPTER 4: IMPLEMENTATION

Introduction

This chapter offers a detailed explanation of the methodologies and tools used to train the Generative Adversarial Network used in this project. Section 4.1 describes the source of the dataset as well as its features. Section 4.2 talks about the various tools and libraries used to import the Word Embeddings and build the GAN. Further, the chapter briefly describes the data processing techniques used in Section 4.3. Section 4.4 talks about the source of the Word Embeddings created by the three algorithms used in this project. Section 4.5 gives an in-depth explanation of the modified architecture of the DCGAN implemented by the author and then details the optimiser and hyperparameters used for training. Section 4.6 talks about the caption modifications made by the researcher to improve Text-Image Alignment. It details the qualitative improvement shown in the emojis by implementing these modifications. Finally, Section 4.7 talks about the User Interface created for this project.

4.1 Dataset Description

This project uses the emojis in the JoyPixels 5.0.1 dataset, procured from [39]. It is a dataset that consists of 3,633 emojis of size 64x64 available to download for free. Each of these emojis is accompanied by a caption used to describe them as shown in the images below. These captions are of Unicode standard [3], which is the world standard for texts and emojis.

sad but relieved face

cowboy hat face

books

face with hand over mouth

4.2 Tools

The programming language used to build this project was Python and it was built using Google Colaboratory. However, due to the lack of RAM and the need for a good GPU, it was upgraded to Google Colaboratory Pro which provides more memory, High RAM runtimes, and access to faster GPUs. The Generative Adversarial Network was built using Keras which acts as an interface for Artificial Neural Networks and the Word Embeddings used in this project were imported using the Gensim Library. Besides this, the project also makes use of the Natural Language Toolkit (NLTK) library for text pre-processing. Finally, the project also uses the Gradio library [40] to build the user interface. Several other libraries such as Open CV, scikit-learn and PILLOW were also used throughout the project.

4.3 Data Pre-processing

Data Pre-processing is a crucial step for building any Deep Learning model. This project synthesises images from text using a conditional Generative Adversarial Network [16]. Hence, the data pre-processing stage involves two main components:

1. Image Pre-processing

The dataset obtained from JoyPixels 5.0.1 contains 3,633 emojis. These 3,633 emojis are all one of a kind and do not share labels. This means that there is a one-to-one mapping between the emojis and the captions. Previous studies have suggested that replicating images up to a certain limit, helps GANs produce better images when data is scarce [41]. Yuki Ezuika et al., attempts at replicating a small amount of data to see if the model they used (SeqGAN) performs better. They used a small dataset of 95 emoticons and a replication constant of N = 10 and found that their model was able to output better results. They also found that setting the duplication-magnification value (N) appropriately was important. Setting N as a very high value could lead to overfitting. Due to the lack of emoji data present,

the author of the proposed model has implemented replication of data in this study in an attempt to produce better results. However, due to a lack of computational power, it was not possible to replicate all the emojis in the dataset and train the GAN on them. Hence, 114 emojis of varying shapes, sizes and colours were carefully hand-picked to create a model that can generate all sorts of emojis. These 114 emojis were replicated seven times (N = 7) to create a total of 798 emojis for the model to learn from. It is important to ensure that the value of N is chosen carefully, in order to avoid overfitting. This is justified further into the paper, in Section 6.1.

An Excel sheet was compiled, which contained the captions of all 798 emojis saved under the column 'text'. Besides this, they were also assigned a number between 1 and 798 under the 'image_id' column. Finally, the complete dataset of emojis was compiled into a folder and all files were renamed to match the corresponding numbers in the 'image_id' column of the Excel sheet. All the emojis were also resized to 64x64 pixels to ensure that they are of the same sizes. It is important to note that these emojis are four-channel images. This means that they have a mode of RGBA where 'A' stands for alpha and represents the transparency of the pixels. These are 8-bit emojis, where each pixel, takes a value between 0 and 255. Finally, the emojis were converted into arrays and were normalised to have pixel values between -1 and 1. This was done as it was found that normalising the image data to have values between -1 and 1 improved the performance of the GAN and made the model stable [42].

2. Caption Pre-processing Image

Pre-processing captions is vital to ensure that the phrases/sentences are ready to be input into the model and do not contain any unnecessary or repeating words. However, this project does not use the Unicode captions provided by the Joypixels 5.0.1 dataset. Instead, the author implements caption modifications with the aim of improving Text-Image Alignment detailed below in Section 4.6. These captions are then pre-processed by using the steps mentioned earlier in Section 3.1.1. Once the captions are converted to lowercase and all punctuation and extra white spaces are removed, they are tokenised, and the stop words are extracted from the captions. The Natural Language Toolkit (NLTK) is a widely used open-source library in Python used for Natural Language Processing. The NLTK library consists of a list of stop words and is used in this project to remove the stop words from the captions after tokenisation.
Once the captions are pre-processed, they are converted from text format into vectors using the Word Embedding algorithms detailed in the following section. These embeddings are then used as conditional input in the model.

4.4 Word Embeddings

Machine learning models cannot process words in the textual format. They can only process data in numerical format. Hence, the image captions which are free of stop words, capitalised letters, and punctuations are tokenised and represented in a vector format using Word Embedding algorithms. As mentioned in Section 2.4, previous studies related to Emoji Synthesis from Text have only used the Word2vec algorithm. However, for this study, the author has used Word Embeddings from three algorithms to test which of the three algorithms would provide the best results. The embedding size used for each of these models was 300 dimensions. The inner workings of the algorithms are explained in Section 3.1.2. The details of the pre-trained algorithms used for this project are detailed below:

1. Global Vectors for Word Representation (GloVe): For this study, the pre-trained Word Embeddings of the GloVe 300d were downloaded from [43]. The GloVe 300d has 6 billion tokens and was trained on data extracted from Wikipedia 2014 and Gigaword 5.

2. Word2vec: The Word2vec model was implemented using the 'Gensim' library. The pre-trained model 'word2vec-google-news' was used for this study. This pre-trained dataset was trained on a part of the Google News dataset that contains about 100 billion words.

3. FastText: The FastText model was also implemented using the 'Gensim' library. The pre-trained model used for this purpose was the 'fasttext-wiki-vec' downloaded from the official FastText website. These Word Embeddings were trained on Common Crawl and Wikipedia using CBOW with a dimension of 300 and character n-grams of length 5.

These Word Embedding algorithms represent each word in a vector format. Hence, to vectorise a sentence or a phrase, the Word Embeddings of individual tokens in a pre-processed sentence is found and then summed up to create a final Word Embedding of dimension 300. This sum is the final vector that represents the sentence. These are then used as the conditional input in the model.

4.5 Architecture

This research involves the use of Conditional Generative Adversarial Networks that allow the model to synthesise Emojis based on the Text that the user chooses to input. As mentioned in Section 2.4, at present there is a lack of research on Text to Emoji synthesis. Apart from one research paper that uses transformers [28], all previous papers make use of the Deep Convolutional Generative Adversarial Networks [1,2]. Since this study focuses on Generative Adversarial Networks, the author has chosen to work with DCGANs in an attempt to improve the existing results and accomplish the goals mentioned in Section 1.2.

The author proposes a modified version of the Deep Convolutional Generative Adversarial Network that provides better results than a standard DCGAN. The architecture of the model used in this project is detailed below and the results have been presented in Chapter 6.

Generator

The main difference between the standard DCGAN and the author's model lies in the architecture of the generator. The generator of the standard DCGAN [4] consists of multiple convolutional and upsampling layers. However, the model proposed by the author uses a UNet [36] instead of the generator of the DCGAN. This replacement was done because the UNet's encoder-decoder architecture allows it to extract a lot more information from the images than the generator of the DCGAN. The DCGAN generator also lacks the skip connections between the encoder and the decoder of the UNet that transfer the information learned during encoding for the decoder to use. Besides this, the UNet (used mainly for segmentation) is known to have a good performance despite being trained on limited training samples [65,66]. These reasons make the UNet a great choice for a replacement.

The generator has a latent dimension of 512. This is concatenated with the Word Embeddings (conditional input) of dimension 300. The concatenated vector is then used as an input into the UNet. The encoder section of the UNet comprises several Convolutional layers with a kernel size of 3x3. Just like the standard UNet architecture, the convolutional layers are accompanied by the Leaky ReLU activation function. The decoder comprises several layers of Transposed Convolution with a kernel size of 3x3. The transposed convolutional layers in the decoder are accompanied by the ReLU activation function (as in the standard UNet architecture). Although the standard UNet architecture, does not contain Batch Normalisation layers, they were added to the proposed model, due to their increased stability and regularising effect. This model is trained on only 798 emojis and hence, batch normalisation helps reduce overfitting. Besides this, several dropout layers of 0.5 were also added to the decoder to reduce overfitting. Just like the standard UNet, the model comprises skip connections between the encoder and decoder layers to allow smooth transfer of information obtained by the encoder to the decoder to use during image reconstruction (As mentioned in Section 3.3.2). The encoder consists of 5 layers followed by 2 bottleneck layers followed by 5 decoder layers. The final layer of the generator uses the Tanh activation function.

It is important to note that, unlike the UNet architecture proposed in [36] the author's proposed model does not use max-pooling layers in the encoder. It instead uses Convolutional layers with stride 2. Using a stride of 2 helps to reduce the size of the output. Hence, this works as a replacement for max-pooling layers. Research has shown that replacing max-pooling layers with convolutional layers of stride 2 increases the model's learning ability [44,45]. Similarly, upsampling layers in the decoder have been replaced by Transposed Convolutional Layers of Stride 2. This removes the need for upsampling and max-pooling layers that are fixed operations and replaces them with strided transposed convolutions and strided convolutions respectively that were found to improve the overall accuracy of the model and reduce overfitting.

Discriminator

The discriminator of a GAN is a binary classifier that is responsible for differentiating between the real and synthetic samples presented to it. The author has used the discriminator architecture of the standard DCGAN [4] which has a kernel size of 3x3 for this project. It consists of multiple convolutional layers and uses Leaky ReLU as the activation function as suggested by [4]. Several Dropout layers of 0.25 (chosen through trial and error) were also added to avoid overfitting. Besides this, Gaussian Noise of standard deviation 0.2 was also added to the input which helped stabilise the GAN [42]. The real and generated images pass through several layers of convolution and are concatenated with the Word Embeddings of the sentence (conditional input). This output is then passed through a fully connected network with 1 node which will help the discriminator classify the input images as real or fake.

The kernel size of both the Generator and the Discriminator of the GAN was chosen through trial and error. It was found that a kernel size of 4x4 led to worse results than a kernel size of 3x3. [60] talks about how increasing the latent dimension, improves the GAN performance up to a certain limit. Following the limit, the performance of the model remains stable. Hence, dimensions of 100, 256 and 512 were tested and a dimensionality of 512 was chosen as it led to the best results. The results for this are shown in Section 6.1.

Optimisers

Both the generator and the discriminator are first built, following which the discriminator is compiled using the Adam optimiser with a learning rate of 0.0001 and beta_1 = 0.5. Then a combined model is created which combines both the generator and the discriminator for which the discriminator is no longer trainable. This is because when a GAN is trained, the generator and discriminator are trained alternatively with the discriminator being trained first. The combined model is then compiled, and it also uses the Adam optimiser with a learning rate of 0.0001 and beta_1 = 0.5. All hyperparameters of this model were chosen through trial and error.

Training the model

The model is then trained for a total of 980 epochs, with a batch size of 27. For each epoch, the discriminator is trained first. Following this, the combined model is trained where the discriminator is no longer trainable. The training data of size 771 is divided into 28 batches of batch size 27. The training data is then trained in these batches. This cycle carries on until all 28 batches have been trained in the first epoch, after which the testing of the first epoch (size of test set = 27) takes place. Then the training process moves on to the next epochs where the same cycle is carried out for all 980 epochs. The loss function used is the Binary Cross Entropy Loss which is used to compute the loss equations 3.16 and 3.18.

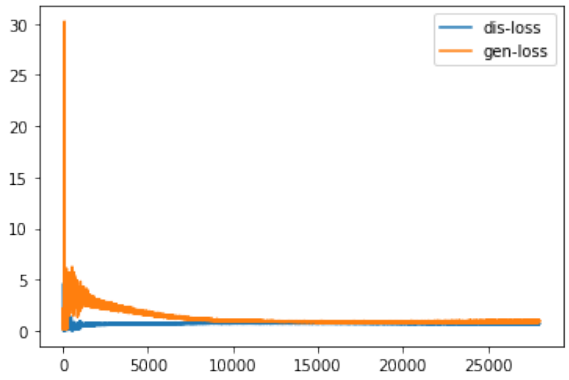


Fig 29: Discriminator and Generator Loss Curves

[85] talks about how stable GANs usually have a discriminator loss between 0.5 and 0.8 and the generator loss is usually slightly higher and can hover around 1.0. Both the discriminator and generator losses fall into these ranges, with the discriminator loss taking values around 0.7 and the generator loss taking values around 0.9.

Generation of existent emojis

Fig 30 : Raw Emojis used for training (Left) and Generated Emojis using captions used for training (Right)

Figure 30 shows the raw emojis used to train the GAN (left) and the regeneration of the training emojis by the GAN using the text (right). The emojis created by the generator (right) are of good quality with minimal noise and are similar to the raw emojis that were used to train the GAN.

4.6 Caption Modifications for Improved Text-Image Alignment

The captions used to train the model were those extracted from the JoyPixels 5.0.1 dataset. These correlate with the captions present on www.unicode.org, which is the world standard for texts and emojis. However, while trying to create new emojis, it was found that the model's performance was unsatisfactory. This was because the model was not able to correlate the features of the emoji with the captions used to describe them. This is a major drawback of the Unicode captions for Text to Emoji Synthesis.

Standard Text to Image Synthesis models are trained on thousands of images, each accompanied by multiple captions (E.g.: Flickr30k dataset which contains 30,000 images and each image has 5 captions). However, for this project, only 114 emojis are used, each of them having one caption. Emojis have a lot of repeating features. Hence, mapping the features of each emoji to specific words helps make the creation of non-existent emojis easier and improves the performance of the model. This is done by modifying the captions to improve the Text-Image Alignment.

Hence, the captions were manually changed for all 798 emojis, to best reflect the features present in the emoji. While this does mean that the modified captions would be longer than the standard Unicode captions, using more descriptive text was found to improve the quality of the emojis by reducing the noise and showing features expected from the caption.

Text-Image Alignment was improved by making the following modifications:

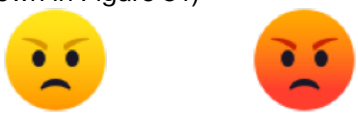
a. Assigning different terms for different features in an emoji and using them to create captions. At times, the Unicode Captions used the same word to describe multiple features of the emoji. The modified captions assign unique terms to different features to give the user more control over the type of emojis they want to create and aid in creating clear and precise features in the emoji. This is described in Point 1 below.
b. Emojis that have similar expressions, but different colours are given the same caption with an additional term that describes the colour of the two emojis. This not only makes the model capable of understanding the expression, but it also allows the model to learn colours well. This is depicted in points 2 and 4 below.
c. Using descriptive text to ensure all features of the emoji are mapped to a word. This is explained in points 3 and 5 below.

Hence, the modified text improves Text-Image Alignment by creating a mapping between the features of the emojis and the words used to describe them. Hence, emojis having similar features are linked

to each other by the text used to describe them. Some examples of the modifications made to the captions and their implications are detailed below :

1. Several emojis such as those shown below have dripping sweat. However, the captions used to describe them are 'sad but relieved face' and 'anxious face with sweat'. These texts are ambiguous and do not allow the model to learn what 'sweat' is. Hence, the captions were modified to ensure that such emojis have the word 'sweat' in them. This change has allowed the generator to associate the word 'sweat' with the blue drop dripping down the face. The modified captions also allow the user to differentiate between sweat dripping off the side of the face and sweat dripping from the forehead by assigning two different phrases, 'sweat' and 'forehead sweat' unlike the Unicode captions which assign the same word (sweat) to both these features. (Shown in Figure 31)

2. The two emojis shown below were captioned 'angry face' and 'pouting face' by the Unicode Captions. However, both the emojis have the same expression with the colour being the only difference. Hence, these two emojis were renamed as 'yellow angry' and 'red angry'. This has allowed the model to correlate the expression with the word 'angry'. It also allows the model to learn the colours 'red' and 'yellow' as the only difference between the two emojis is the colour. This also allows the model to recreate emojis from the training set by only changing the colour of the emoji. (Shown in Figure 31)



3. The first emoji below was originally captioned as 'smiling face with sunglasses'. However, the middle emoji was named 'smirking face'. Once again, this does not allow the model to learn what sunglasses are and what a smirk is. The first emoji is not smiling, it has a similar expression to the smirking emoji. Hence, this was renamed as 'smirking and wearing sunglasses' and the second emoji was renamed as 'smirking'. Doing this allowed the model to understand what sunglasses are and the model was able to generate emojis wearing the sunglasses. Similarly, the third emoji was originally named 'winking face'. This was renamed as 'flirty wink and smirking'. (Shown in Figure 31)

4. There are also images of three books available in the dataset. All three emojis look the same except for the colour. The first two emojis were captioned 'blue book' and 'orange book'. However, the third book was captioned 'closed book' despite everything about the emoji being the same as the former two emojis except for the colour. This leads to ambiguity. Hence the 'closed book' was renamed as 'red book'.

5. The emoji below was originally captioned as ‘nerd face’. However, this was renamed as 'neutral and happy smile wearing spectacles and braces'. Although the text is longer, using this modified caption has led to better results as it mentions that the emoji is wearing spectacles. This allows the model to understand what spectacles are. Modifying the caption in this manner allows the model to generate non-existent emojis with spectacles which were not previously possible. (Shown in Figure 31)

Similar to the changes presented above, the captions of all 798 images were changed to create links between the emojis that have similar features. This allows the model to learn the correlation between the words in the captions and the features that they represent.
The emojis below represent the new emojis that the model creates before and after changing the captions. Please note: The text used to generate the emojis before and after changing the captions are different to reflect the captions the model has been trained on.

Figure 31 represents the difference in the generated emojis before implementing the text modifications (right), versus, after implementing the modifications that created a mapping between the features and the words used to describe them (left). It is visible from the images below that the model creates emojis of better quality after the caption modifications and the emojis contain features one would expect from the text input indicating improved Text-Image Alignment.
For instance, comparing the 'cat face with sweat' emoji with the 'cat with sweat' emoji, we can see that earlier, the model was confused about where the sweat goes on the emoji. This is because in the previous dataset the term 'sweat' was used to describe sweat dripping off the face as well as sweat dripping off the forehead, but once these were given different terms as mentioned in Point 1 above, the model can create much clearer and more precise emojis. The lack of descriptive text explained in Point 5 can be seen in the 'grinning face with glasses' emoji. Previously the model was unable to create glasses, whereas now that the texts have been changed, the model clearly understands what 'spectacles' are, and which feature they represent. Overall, using the new texts, the emojis created by the model have less noise and clearly show the features expected.

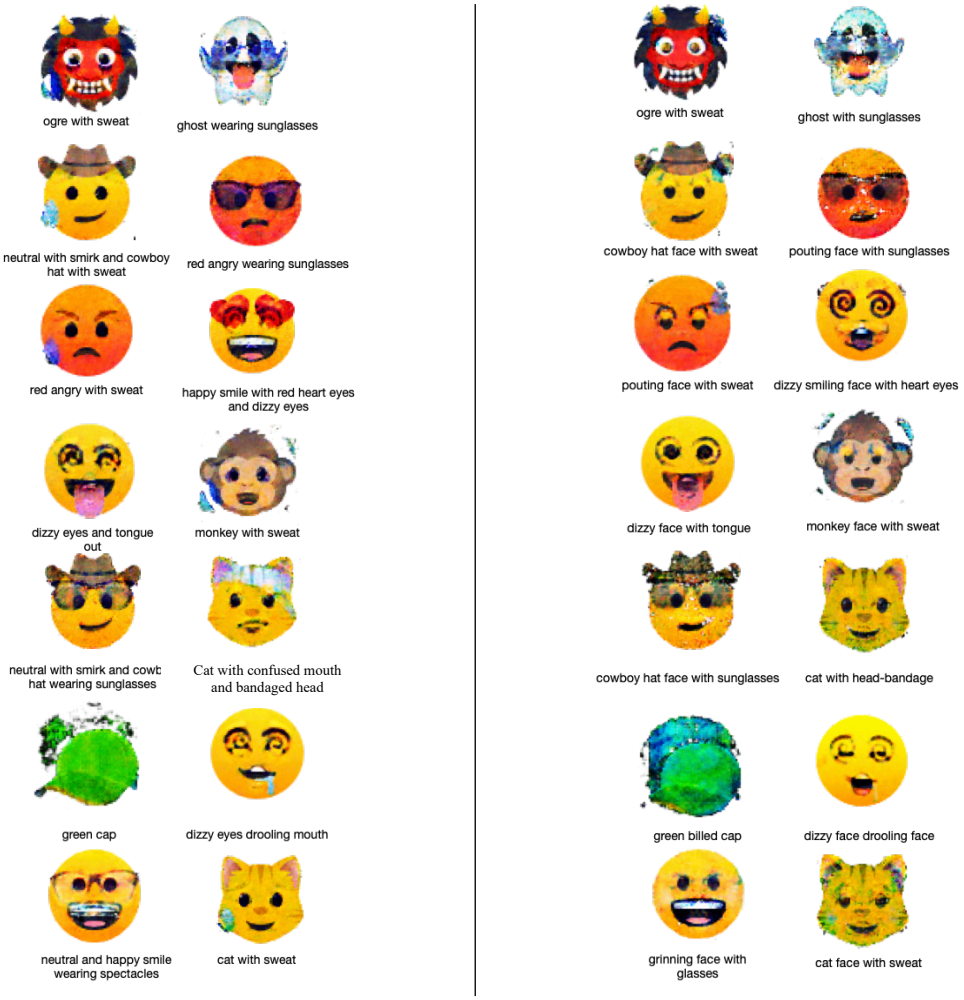


Fig 31 : Emojis generated by the GAN when using Unicode texts (right) and when using modified text (left)

In fact, these captions even allow the models to understand minute features like ‘blushing’ as shown in the image below (Figure 32).

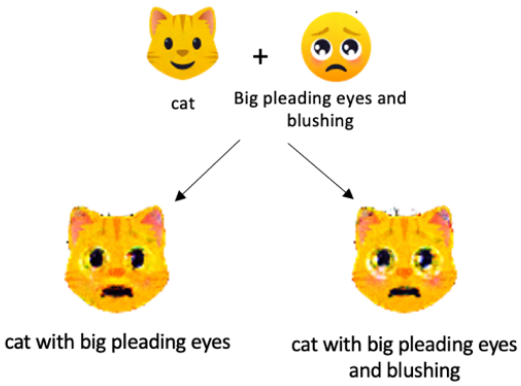


Fig 32 : Minute details that the modified texts allow the model to learn

Besides this, it also gives the user more control over what kind of emojis they want to create:

Fig 33 : Minute details that the modified texts allow the model to learn

Mixing up the two emojis using the Unicode captions on the left, leaves the model capable of only creating one emoji which shows a disfigured emoji that shows features of all aspects of the two emojis no matter how we might change the text. On the other hand, the modified captions allow the model to map features of the emoji to text rather than map emojis as a whole to the text and this means the model is capable of creating different types of emojis based on the text we input. For instance, the two emojis created by the modified captions can be used to describe two completely different emotions. Hence modifying the captions does make the model better and more intelligent (Figure 33).

Thus, the modifications made to the text achieves the following:

a. It creates a better mapping between the features of the emoji and the words used to describe them. This implies these modifications improve Text-Image alignment.
b. It helps the model understand this mapping better, thus creating emojis that have less noise and that clearly depict the features the text describes.
c. The model is capable of picking and choosing features to create multiple emojis by mixing up the features of two emojis unlike the Unicode Captions which usually is only capable of creating one emoji which contains all the features of the two emojis we are attempting to mix.
d. The user has more control over the type of emojis they want to create by simply changing the terms they use in the caption.

4.7 User Interface (UI)

To present the emojis well and provide the user with a platform to enter text in and instantly see the emojis they generate, a User Interface or UI was created. This was created using the Gradio [40] library.

Gradio [40] is a library that allows a user to demonstrate their machine learning models using a user-friendly web interface.

The Gradio interactive app below was built using Blocks. Blocks are low-level APIs that are used to demonstrate our machine learning models flexibly. They offer flexibility to do multiple things such as feature multiple demos at a time as well as give the programmer the freedom to control where their components appear on the app and are thus, more customisable [40].

This Gradio App is a 'Text to Image Synthesis' application which offers the user, the freedom to type in the text of their choice into the text box as shown in Figure 34. The app is coded to ensure that all text entered is pre-processed. This involves making all text lowercase, removing all punctuation marks, as well as stripping the text of spaces before and after it.

Suppose the text entered by the user belongs to an image in the training set, then the raw image, its caption, and the GAN-generated image will be displayed. Besides these features, the Gradio app also showcases two evaluation metrics, the SSIM and the Percentage of matching Key Points detailed in Chapter 5. Alongside this, the generated image is classified into one of the 114 unique training captions using a classifier created through transfer learning detailed in Chapter 5, Section B.2.3. This acts as an additional evaluation metric for the generated images. If the text entered by the user does not belong to an image in the training set, the Gradio app will only output the Synthetic image.

Figure 34 (below) shows the Gradio application created by the author to present the machine learning model.

Fig 34 : 'Text to Emoji Synthesis' Application using Gradio

Summary

To summarise, this chapter talks about the dataset used for the project as well as the major tools and methods used to train the Generative Adversarial Network. It also provided an in-depth explanation and sources of the three algorithms used to create the Word Embeddings. It then goes on to talk about the text modifications made in the dataset and how it has led to synthesis of better emojis through improved Text-Image Alignment. Finally, the library 'Gradio' that was used to build the UI for this deep learning model was introduced.

CHAPTER 5 : EVALUATION METRICS

Introduction

Evaluation is a vital part of training a neural network which helps us gain a deeper understanding of the model's performance. Based on the results, the model's performance can be improved by tweaking its architecture and hyperparameters. This chapter talks about the multiple evaluation metrics used to quantify the performance of the GAN and provides a detailed overview of the implementation of these metrics in this project.

The author has used a total of 7 metrics to evaluate the GAN. These comprise one Qualitative Metric (Section A), outlined in Section A.1 and 6 Quantitative Metrics detailed in Section B. Section B is divided into three sub-sections discussing the three types of quantitative metrics used for evaluation. Section B.1 details the Image Quality Metrics implemented. Section B.2 dives into the popular GAN evaluation metrics implemented by the author. Finally, Section B.3 talks about two metrics which have not been used in previous research in Text to Emoji field [1,2,28] and have hence been classified as 'Additional Metrics'. Section 5.1 below, outlines each of the metrics used in this project in more detail.

5.1 Evaluation Metrics Used in The Project

Usually, neural networks are trained with an objective loss function. This loss function helps the researcher understand the progress of the model's performance during training. However, GANs do not have an objective loss function [64]. As mentioned in Section 3.3.5.1, the generator of the GAN is not trained directly and is trained using the discriminator model. It does this by using the discriminator loss that defines how well the discriminator (a binary classifier) can differentiate the real and fake images. The lack of an objective loss function means that there is no way to assess the model and how the training of the GAN progresses. Although several metrics have been developed which can be used to evaluate the performance of a GAN, they have some limitations due to which they cannot provide an accurate assessment of the proposed model in this project. Due to these limitations, the evaluation of Generative Adversarial Networks is still an active and open area of research.
The author has evaluated the proposed model, using a mixture of qualitative and quantitative metrics to provide a robust assessment of the model.

Section A.1 talks about the most popular qualitative GAN evaluation metric - Manual Inspection. Section B.1.1 and B.1.2 explain two popular Image quality metrics, Structural Similarity Index Measure (SSIM) and Peak Signal to Noise Ratio (PSNR). Sections B.2.1 to B.2.3 dive into two popular GAN evaluation metrics, Inception Score (IS) and Fréchet Inception Distance (FID). These sections also detail the modified implementation of these two metrics with the help of Transfer Learning to make them applicable to this project. Finally, the author has also implemented two additional methods, Feature Matching and Caption Loss, detailed in Section B.3.1 and B.3.2 respectively.

A. <u>Qualitative Metrics</u>

A.1 Manual Inspection

Manual assessment of images is one of the most popular methods used to evaluate GANs and is usually performed by the researcher. The researcher manually goes through the images created by the GAN and evaluates the quality as well as the variation among the generated images. Although visual inspection is the simplest and most popular evaluation method, it has several limitations. The evaluations may be a product of some form of bias of the researcher about the model. Manual evaluations are also time-consuming and cumbersome as GANs do produce a large number and variety of images and require knowledge of what is realistic or not. [46]

All 114 images generated by the GAN were manually inspected by the researcher and were found to be of good quality and were good representations of the captions they were trained with. The images created by the GAN are presented in the Appendix (Appendix B) along with the raw images for comparison.

B. Quantitative Metrics

B.1 Image Quality Metrics

These evaluation methods are used to assess the quality of images in the form of a numerical metric after they go through some kind of image degradation. The author uses two Image Quality metrics which are the Structural Similarity Index measure (SSIM)[47] and Peak Signal to Noise Ratio (PSNR). Both metrics are Full Reference Quality Metrics which create a comparison between the generated image with the raw images from the dataset [67].

B.1.1 Structural Similarity Index Measure (SSIM)

The Structural Similarity Index Measure (SSIM) [47] is a common evaluation technique that checks how similar two images are. It does this by finding the difference between their luminance, contrast, and structure and combining them into a single numerical value. SSIM ranges from 0 to 1, where a score of 1 would represent two images that are the same.

For this project, SSIM was found by using the 'scikit-image' library which is equipped with a function called 'structural similarity' which is used to compute the SSIM of two images. To use this function, both the real and generated image are first converted into grayscale as SSIM is designed to only measure the differences between two luminance or gray signals. The conversion to grayscale is done by using the OpenCV library.

$$\mathrm{SSIM}(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{5.1}$$

The equation above (Equation 5.1) is used to calculate the SSIM of two images, where μ_x and μ_y are the means of x and y respectively. σ_x^2 and σ_y^2 represent the variances of x and y, and the covariance of x and y is represented by σ_{xy}. c_1 and c_2 are variables that are used to stabilise the division, where $c_1 = (k_1L)^2$ and $c_2 = (k_2L)^2$ and L is the range of pixel values, which is given by: $(2^{\#\text{bits per pixel}} - 1)$ and $k_1 = 0.01$ and $k_2 = 0.03$ (default).

B.1.2 Peak Signal to Noise Ratio (PSNR)

The Peak Signal to Noise Ratio (PSNR), as the name suggests is the ratio of the power of the signal (the image) to the noise (what degrades the image). The equation below (Equation 5.2) is used to find the PSNR of an image where f represents the real image, MAX_f represents the maximum pixel value in the real image and MSE is the Mean Squared Error (Equation 5.3).

$$PSNR = 20\log_{10}\left(\frac{MAX_f}{\sqrt{MSE}}\right) \tag{5.2}$$

$$MSE = \frac{1}{mn}\sum_{0}^{m-1}\sum_{0}^{n-1}\|f(i,j) - g(i,j)\|^2 \tag{5.3}$$

PSNR is calculated using the Mean Squared Error (MSE) which measures the average squared difference between the real image and the generated image (Equation 5.3). This project uses 8-bit images and hence, the highest possible PSNR value is 48dB.

Although both SSIM and PSNR are widely used to evaluate projects that work with images, they come with certain limitations. They are not very reliable metrics as research has found that there are cases where they assign images with significant blurring or distortion differences with the same SSIM and

PSNR scores [68,54]. Hence, the reliability of these metrics is questionable. Even so, they are still popularly used image metrics.

B.2 GAN Evaluation Metrics

As mentioned in Section 5.1, the author has implemented two GAN evaluation metrics, namely Inception Score [25] and Fréchet Inception Distance [26]. Both metrics were designed to quantitatively evaluate the performance of generative models whereas the metrics mentioned in Section B.1 (SSIM and PSNR) can be used in a variety of projects dealing with images.

B.2.1 Inception Score (IS)

The Inception Score [25] is an algorithm that automatically assesses the quality of images created by generative models like GANs. It uses the pre-trained Inception V3 model [48] to classify the images generated by the GAN. The Inception V3 [48] model was trained on the ImageNet dataset containing 1000 classes of images. This model is used to predict the conditional probabilities P(y|x) i.e. the probability of a particular class belonging to a certain image. These conditional probabilities are then used to find the marginal probability which is then applied to the Inception Score formula (Equation 5.4) which outputs the score. The formula is used to calculate the Kullback-Leibler (KL) Divergence (Equation 5.5) between the conditional and marginal probabilities. Following this, the KL Divergence is then summed over all classes and averaged over all images before taking the exponential which produces the final score.

$$\mathrm{IS}(G) = \exp\left(\mathbb{E}_{\mathbf{x}\sim p_g} \, D_{KL}(\, p(y|\mathbf{x}) \parallel p(y) \,) \right), \tag{5.4}$$

$$D_{KL}(\, p(y|x) \parallel p(y) \,) = p(y|x)*(\log(p(y|x)) - \log(p(y))) \tag{5.5}$$

The conditional probability defines the quality of the images, and the marginal probability defines the variety of the images. Hence, overall, the Inception Score is used to measure two things:

a. The variety of images present in the generated images.
b. Each image should distinctly look like a particular class.

The Inception Score [25] ranges from the lowest score of 1.0 to the highest score of the number of classes that the classification model can classify. In the case of the Inception V3 model [48], the highest possible score would be 1000. The higher the inception score, the better the generative model performs.

B.2.2 Fréchet Inception Distance (FID)

Although the Inception Score [25] is a widely used metric for evaluating the performance of GANs, it is incapable of assessing the similarity between the real and the generated images. This gap is filled by the Fréchet Inception Distance (FID) [26]. The Inception Score is only applied to the set of the generated images whereas the Fréchet Inception Distance compares both the real and the generated images.

Similar to the Inception Score, the Fréchet Inception Distance [26], also makes use of the Inception V3 classifier. The difference between the two is that the FID does not use the classification layers of the Inception V3 model. These layers are removed, and the model is used as a Feature Extractor. Using the model, the features of the real and generated images are extracted, and the distributions (assumed to be gaussian) are found by finding the mean and the standard deviation. Following this, the distance between the two distributions is found. This distance is the Wasserstein-2 distance which is found by using the formula shown below (Equation 5.6) where x and y represent the real and generated images respectively and Σ_x and Σ_y represent the covariance matrices of the real and generated feature vectors. If the two distributions are perfectly identical, the distance between them would be zero which means that the images are the same. Hence, the FID can range from the lowest value of 0 and can take higher values as the Wasserstein-2 distance between the distributions of the real and the generated images increases.

$$\mathrm{FID} = ||\mu_X - \mu_Y||^2 - \mathrm{Tr}(\sum_X + \sum_Y - 2 \quad \sum_X \sum_Y) \tag{5.6}$$

B.2.3 Modified Inception Score and Fréchet Inception Distance using Transfer Learning

As mentioned in the earlier sections, the Inception V3 classifier [48] is used to calculate the Inception Score [25] and the Fréchet Inception Distance [26]. However, this classifier is incapable of classifying emojis. This is because it has been trained on the ImageNet dataset which contains images belonging to 1000 classes [49] but does not include emojis. Hence, using this model will not provide an accurate score for these two metrics. Using the Inception V3 classifier to find the Inception Score and Fréchet Inception Distance will lead to a very low score for the former and a very high score for the latter. Hence transfer learning was used to create a classifier that was used instead of the Inception V3 classifier to find these scores.

Transfer Learning is a method that is widely used in machine learning, where knowledge and information obtained from solving one problem is then applied to solve a different, but related problem. This means that a model that was originally used to accomplish a particular task is then slightly modified to utilise it for a similar task.

In this project, the author has implemented transfer learning on the discriminator of the GAN. This is possible because the discriminator is a neural network that is trained on the real dataset of raw emojis and is capable of classifying the emojis as real and fake. In other words, it is a binary classifier. Hence, transfer learning was implemented on the discriminator to convert it into a classifier that would now be able to classify emojis into one of the 114 categories of unique emojis. This classifier was then used to find the conditional probabilities of the generated emojis and then find the Inception Score [25] using Equation 5.4. Further, the classification layers from the classifier were removed and then it was used as a feature extractor to find the Fréchet Inception Distance [26]. The activations are reduced to 1024 features due to a lack of computational power. This means that each generated image is reduced to 1024 activations features. A 1024 feature vector is predicted for all the real and synthetic images and then the distance between their gaussian distributions is found by using the equation mentioned above (Equation 5.6)

Transfer Learning to create the classifier was done by using the steps below:

1. Loading the saved discriminator model and weights using Keras
2. Setting the first 14 layers of the model as untrainable. This includes all layers of the discriminator except for the classification layers of the discriminator which make it capable of classifying the images as real or fake. This is a vital step done in transfer learning to bring the weights of the original discriminator to the new model we are now creating.
3. Removing classification layers of the discriminator, which were used to classify the emojis as real or fake.
4. Adding a fully connected layer that has 114 neurons, thus making it capable of classifying all 114 classes of unique emojis.
5. Adding a softmax activation function at the end that would make the model capable of outputting conditional probabilities of the image belonging to each of the labels.

This model was then compiled using the Adam optimiser, which is the same optimiser used to train the discriminator. The loss function chosen was ‘categorical cross entropy’ which is commonly used for multi-class classification tasks. The model was then trained and tested for 1 epoch only to avoid overfitting due to the small dataset and because it is already trained on the raw emojis. After training, the transfer learned model now outputs probabilities for each of the 114 classes, and the label having the highest probability is assigned to the image.

Inception Score and Fréchet Inception Distance have some limitations too. These are discussed in Chapter 6 during the analysis of the results.

B.3 Additional Metrics

The author has implemented two additional metrics, other than the previously mentioned 5 metrics. These two metrics have not been used previous research related to text to emoji synthesis but can serve as good evaluation metrics for the images generated by the GAN.

B.3.1 Feature Matching

Feature Matching [50,51] is an important concept in Computer Vision as well as Object Detection. It involves using algorithms to find the main keypoints of the images and then matching the closest keypoints of the two images based on a search distance algorithm.

The author has carried out Feature Matching between the real and the generated images by using the Scale Invariant Feature Transform (SIFT) algorithm [5,51]. SIFT is an algorithm that is used to find interest or keypoints in an image. Keypoints are locations or points that are interesting and stand out in an image. They are locations which show significant intensity change in all directions. These are usually corners in a rectangle or the nose tip, eyebrows, or eye corners in a human face. This is illustrated in Figure 35.

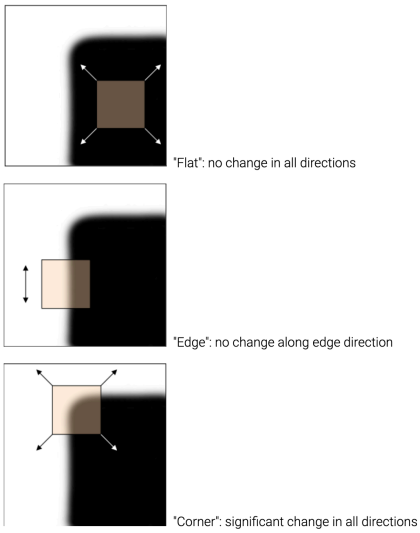


Fig 35: Corners are good keypoints.

One of the most used keypoint detectors is the Harris Detector. However, the author has chosen to use the SIFT [5] detector over the Harris detector as the latter is only capable of finding keypoints that are rotation invariant, whereas the former finds keypoints that are both scale and rotation invariant. The Harris detector is capable of finding keypoints between two images even if one is a rotation of the other, while the SIFT algorithm is not only capable of doing the aforementioned task but also of finding keypoints between images regardless of their scales. Besides this, the SIFT algorithm is also partially invariant to illumination, noise, and minor change in viewpoint.

The following are the steps involved in SIFT [69]:

1. Constructing a Scale Space

This step involves identifying the features that stand out and ensuring that they are scale independent. This is done by first applying Gaussian Blurring to the image to reduce the noise and then creating a scale space. A scale space is a set of images that have been scaled to different sizes from a single image. Each scaled image is then blurred multiple times. Usually, the image is scaled 4 times, which are called Octaves and each octave is then blurred 5 times.

Further, the Difference of Gaussian of these images are taken to enhance the features. This is done by subtracting the blurred version of images within an octave from each other. Therefore, each of the 5

blurred images in an octave will create four Difference of Gaussian images. This is done for each of the octaves.

2. Keypoint Localisation

This step is done to find the keypoints in the Scale Space. This is done by finding the local maxima and local minima of the images which is done by comparing every pixel with its 9 neighbouring pixels of that image and the previous and next image of a particular octave. The pixel is then selected as a keypoint if it is the highest or lowest pixel amongst its neighbours. This returns keypoints that are scale invariant. However, some of these keypoints may not be robust to noise. Hence, it is vital to eliminate the keypoints that have low contrast or lie very close to the edge. The keypoints that have low contrast are eliminated by implementing the 2nd order Taylor Expansion for each keypoint and rejecting those that are lower than a particular threshold. On the other hand, keypoints that are close to the edge are identified by using the 2nd-order Hessian matrix.

3. Orientation Assignment

This step is done to assign an orientation to each of the scale-invariant keypoints found earlier, to ensure that they are rotation invariant. For this, we find the magnitude and orientation (angle) of the keypoints using the formulas below (Equations 5.7 and 5.8)

$$\text{Magnitude} = \sqrt{[(G_x)^2+(G_y)^2]} \tag{5.7}$$

$$\Phi = \tan^{-1}(Gy / Gx) \tag{5.8}$$

Once the magnitudes and orientations are found, a histogram is created that lists the magnitude (y) vs angle (x) for all pixels. The points where the graph peaks are keypoints that are rotation invariant.

4. Keypoint Descriptor

This step is used to assign a unique fingerprint to each of the keypoints found earlier that are scale and rotation invariant. This is done by using the neighbouring pixels' magnitude and orientation. Since we utilise the neighbouring pixels, the descriptors are partially invariant to illumination.

For this project, the author has used Open CVs SIFT functionalities [51] to implement SIFT on both the generated and real images. The keypoints of each generated emoji is matched with the keypoints of all the emojis in the real dataset. These matches are found by Brute Force where every keypoint is matched with every other keypoint (Figure 36). Further, the Manhattan distance between them is found following which they are arranged in increasing order. The closest keypoints are matched to each other. This is done for all real and generated emojis, and the one with the greatest percentage of keypoint matches would be assigned as the closest match. The percentage was found by dividing the number of keypoints matched to each other by the total number of keypoints in both emojis. This is depicted in Equation 5.9 below. The Euclidean distance can be used in place of the Manhattan distance to find the closest matches. Although the Euclidean distance is more accurate, the author has chosen to use the Manhattan distance as it is easier and faster to compute than the Euclidean distance.

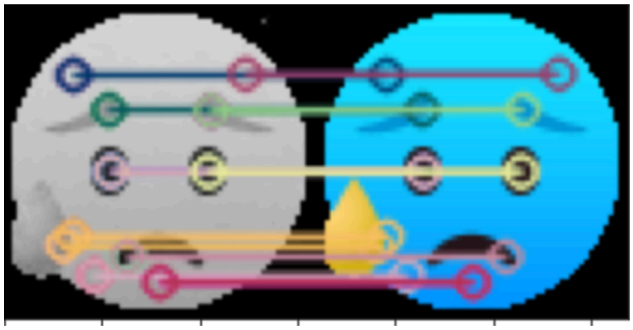


Fig 36: Keypoints using SIFT matched through Brute Force between real and generated image

Percentage of keypoint matches = Number of Keypoints matched / Total number of keypoints (5.9)

B.3.2 Caption Loss

This is an evaluation metric that was proposed by [28] for future research. It involves training a neural network for the inverse process. In other words, this involves training a neural network to create captions from an image. Hence, this technique involves generating captions from the generated image and seeing how similar they are to the real captions.

The following steps were taken to build a caption generator:

1. Pre-process images by resizing them so that they can be used with the Inception V3 model.
2. Clean captions of images using pre-processing techniques mentioned in Section 4.3
3. Create a vocabulary of all the unique words in the captions.
4. Add the terms 'startseq' and 'endseq' to all captions so that the model recognises the start and end of the captions and is able to process the captions accordingly.
5. Remove classification layers from the Inception V3 classifier to use it as a feature detector to extract the features of the images.
6. Create a training and a test set to train and evaluate the model.
7. Encode image to features by using the Inception V3 model.
8. Assign every word in the vocabulary an integer value starting from 1.
9. Training this neural network in a supervised manner involves giving the neural network two inputs which leads to the production of one output. This means that the model would take the encoded features of the images as input along with the integer encoded words of the captions to have the model predict what the next word in the text sequence would be. The model outputs the probabilities of all the words being the next word in the sequence. Hence, it predicts the integer assigned to the next word that is predicted by the model. These are encoded by using one-hot encoding. It is important to note that the input text sequence has been padded with the length of the actual caption.
10. The features of the image are passed through a fully connected layer to reduce the number of activations and the caption inputs are passed through an 'Embedding' layer to convert the integer values into dense vectors of fixed size. The output of this layer is then passed through an LSTM, after which both the image features and the processed text inputs are concatenated before using a dense layer equipped with a softmax layer which will predict probabilities of the words in the vocabulary being the next word.
11. An embedding matrix is created which contains the Word Embeddings of the words in the vocabulary. This is then used to set the weights of the model after which the model is compiled and trained.
12. The trained model is then used to obtain the captions of the generated images.

Once the captions are generated, evaluation metrics like the BLEU score [53] can be used to evaluate the performance of the caption generator model.

Summary

This chapter discussed the multiple evaluation metrics used by the author in this project. These evaluation metrics focus on both Qualitative and Quantitative evaluation of the images generated by the GAN. Manual Inspection, Inception Score, Fréchet Inception Distance and Feature Matching are some of the metrics that are used by the author evaluate the quality of the images created by the GAN as well as the performance of the GAN. As mentioned earlier in Section 2.3 , Text to Image Synthesis not only focusses on the quality of the generated images, but also focusses on the Text-Image Alignment metrics. Caption Loss is the metric used by the author to test the models' performance in Text-Image Alignment.

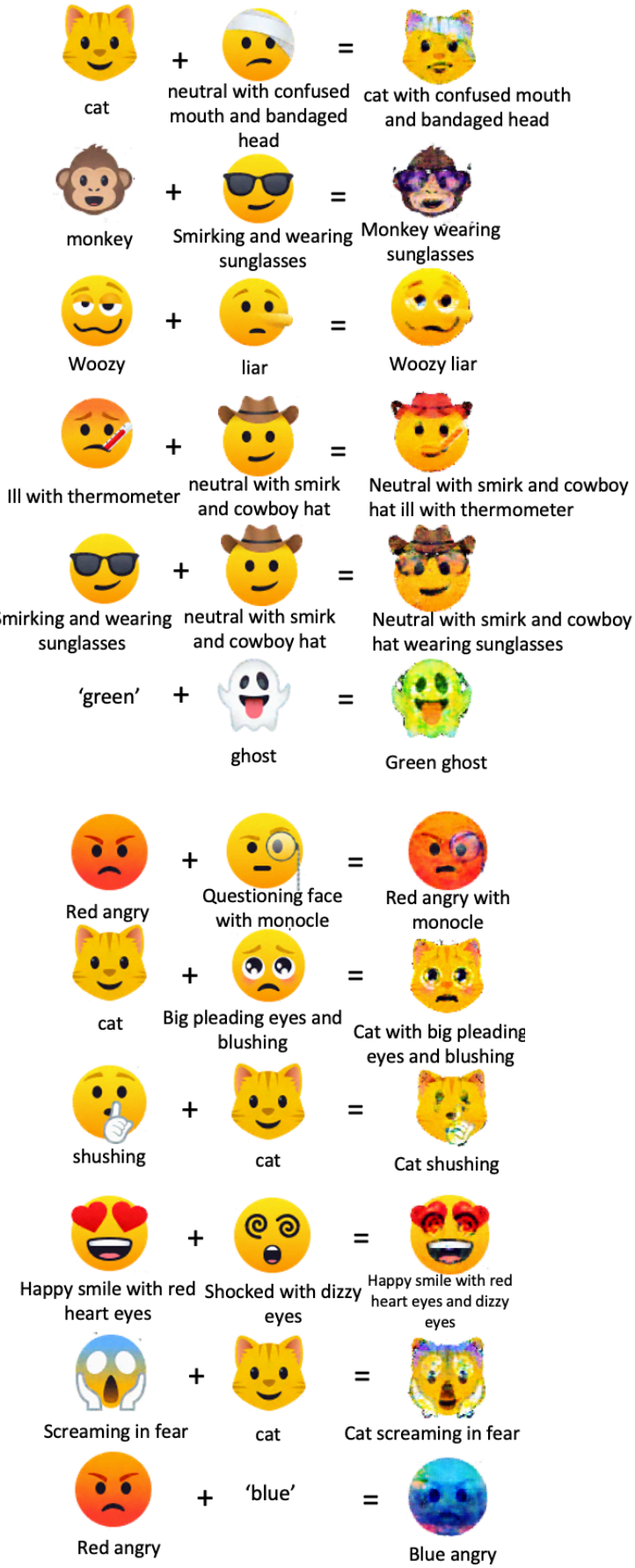
CHAPTER 6 : RESULTS

Introduction

This chapter presents the results of the author's proposed Text to Emoji Synthesis model. Section 6.1 displays the emojis generated by the proposed model. This comprises both, the regeneration of the existing emojis as well as the synthesis of non-existent emojis from the text of the user's choice. The following section, Section 6.2 explains the choice of the architecture of the model i.e., why the researcher chose to replace the generator of the DCGAN architecture with a UNet. It also justifies the choice of kernel size and latent dimension. This is done by analysing the quality of the emojis as well as conducting a quantitative analysis of the evaluation metrics which shows how these implementations have improved the performance of the model. The next section, Section 6.3 provides the reader with a quantitative and qualitative analysis of the three Word Embedding models – GloVe, Word2vec and FastText, and reaches a conclusion about which one leads to the best performance of the model. Section 6.4 talks about the transfer learning model that was implemented to find the Inception score and the Fréchet Inception Distance and analyses how the scores of these two evaluation metrics have improved after implementing transfer learning to create a classifier that replaced the Inception V3 model. The following section, Section 6.5 talks about the Caption Loss model implemented and the results obtained. Finally, this chapter also provides a comparison of the author's proposed model with the models created by previous researchers in the Text to Emoji Synthesis field.

6.1 Emojis created by the proposed model

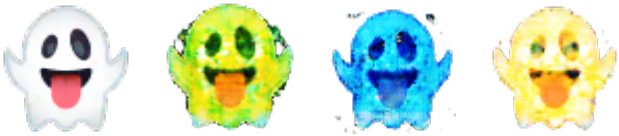
This section provides the reader with the emojis created by the proposed model. Some of the training emojis generated by the model have already been compared with the raw emojis from the dataset and are presented in Section 4.5. All the 114 unique emojis in the training set that have been generated by the model have been compared with their respective raw emojis and are presented in Appendix B. The first step in evaluating GANs is qualitative evaluation through manual inspection. The author has manually inspected all of the training emojis generated by the model and compared them with the emojis in the raw dataset. The model is capable of recreating these emojis very well with minimal noise. Below presented, are some of the non-existent emojis that the model has generated along with the text used to create them.


cat
+
neutral with confused mouth and bandaged head
=
cat with confused mouth and bandaged head
monkey
+
Smirking and wearing sunglasses
=
Monkey wearing sunglasses
Woozy
+
liar
=
Woozy liar
Ill with thermometer
+
neutral with smirk and cowboy hat
=
Neutral with smirk and cowboy hat ill with thermometer
Smirking and wearing sunglasses
+
neutral with smirk and cowboy hat
=
Neutral with smirk and cowboy hat wearing sunglasses
'green'
+
ghost
=
Green ghost
Red angry
+
Questioning face with monocle
=
Red angry with monocle
cat
+
Big pleading eyes and blushing
=
Cat with big pleading eyes and blushing
shushing
+
cat
=
Cat shushing
Happy smile with red heart eyes
+
Shocked with dizzy eyes
=
Happy smile with red heart eyes and dizzy eyes
Screaming in fear
+
cat
=
Cat screaming in fear
Red angry
+
'blue'
=
Blue angry

As mentioned earlier in Section 4.3, [41] found that replicating the dataset helps improve results by creating less noisy images when the data available for training is limited. However, the duplication-magnification constant (number of replications), N has to be chosen carefully to avoid overfitting. A duplication-magnification value (N) of 7 was carefully chosen through trial and error to ensure that the model is not overfitting. As visible from the emojis above, the model is able to create new emojis based on the text inputted. This implies that the model is able to generalise and create new outputs. Therefore, replicating the emojis 7 times does not lead to overfitting and instead improves the performance of the model by creating less noisy images.

As visible from the emojis above, the model has successfully been able to create non-existent emojis that consist of a mix of the features of training emojis. More importantly, the emojis consist of features that one would expect from the text inputted to create them, implying good Text-Image Alignment. This is possible due to the modifications made to the captions (Section 4.5) and the UNet architecture that replaced the generator of the DCGAN (Section 4.5). The results of the UNet vs DCGAN architecture are presented in Section 6.2 (below) and prove that like the modified captions, it aids in improving Text-Image Alignment by creating a better mapping between the words and the features they represent.

It is also interesting to note that the model can learn colours well. For instance, the model was successful in changing the colours of 'red angry' and 'ghost' to blue and green when changing the text to 'blue angry' and 'green ghost' respectively as shown earlier. Below presented is the 'ghost' emoji that has been changed to multiple colours. Even though these emojis may not be perfect, it proves that the model is capable of associating colours of the emojis with the text used to describe them.



ghost green ghost blue ghost yellow ghost

Overall, this model has also been able to create good quality emojis and has depicted a strong Text-Image Alignment. It has outperformed the previous adversarial research on Text to Emoji Synthesis. This comparison is presented in Section 6.6.

6.2 Proposed Architecture VS DCGAN

This section presents the qualitative and quantitative analysis of the emojis, and the results of the evaluation metrics when using the proposed model architecture versus the standard DCGAN architecture. It also justifies the choice of kernel size and latent dimension used. The GloVe algorithm was used to generate the Word Embeddings for both DCGAN and the proposed architecture to create a fair ground of comparison. The DCGAN model had 5 convolutional layers for the Generator (same as the decoder in the UNet) and had the same architecture as the proposed model for the Discriminator. The DCGAN and the proposed model both were run ten times to provide a more robust estimation of the results that were generated by using the evaluation metrics mentioned in Chapter 5. It is important to note that these evaluation metrics only assess the training emojis regenerated by the model as these metrics require raw emojis that are used to compare the regenerated emojis with.

As mentioned above, each of the models (Proposed architecture and the DCGAN) were run ten times and the results were compiled into an excel sheet. The tests could only be run ten times due to time constraints and lack of computational power. Once the tests were run and the results were compiled, a distribution was created from which the mean and the standard deviation of the evaluation metrics were found in order to provide results that are as robust as possible.

Below are the two formulas (Equation 6.1 and Equation 6.2) used to find the mean and the standard deviation of the evaluation metrics:

$$\overline{x} \equiv \frac{1}{N}\sum_{i=1}^{N} x_i. \tag{6.1}$$

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2}, \tag{6.2}$$

Since the number of samples (N) is a small value (N = 10), the divisor used to find the standard deviation (Equation 6.2) is (N-1) instead of the usual N. This is because, when the sample size is small (less than 30), the observed values on average, fall closer to the sample mean than the population mean. Using N as the divisor in such cases would lead to an underestimation of the standard deviation. Hence, a divisor of N-1 is used to provide a better approximation to correct this issue (Equation 6.2).

The tables below (Table 1, Table 2) represent the evaluation results of the ten runs for each of the models. It shows the SSIM, PSNR, Percentage of Keypoint Matches, Inception Score (IS) and the Fréchet Inception Distance (FID). Following this, the mean and standard deviation was found for the above observations using Equation 6.1 and 6.2.
The confidence intervals for the metrics were then calculated using the formula below (Equation 6.3)

$$CI = \bar{x} \pm z\frac{s}{\sqrt{n}} \tag{6.3}$$

The Confidence Interval is the range of values that we can estimate, our values to fall within, with a certain level of confidence. [86]
For this project, the author has estimated 95% confidence. In order to estimate the range of values that the metrics fall into we use Equation 6.3 where, $\bar{x}$ represents the mean of all the observations, $z\cdot$ is a pre-defined value that varies based on the percentage of confidence we choose to have. For instance, for 95 percent confidence, z = 1.960, s is the standard deviation and n is the sample size (n =10).

DCGAN					
Test Number	**SSIM**	**PSNR**	**Keypoint**	**IS**	**FID**
1	0.7628	38.86	88.49	1.34	0.001
2	0.7627	38.68	84.21	1.1801	0.0007
3	0.762	39.11	85.08	1.13	0.0005
4	0.758	38.56	82.45	1.152	0.0013
5	0.756	38.45	88.59	1.189	0.001
6	0.7612	39.05	87.71	1.185	0.001
7	0.766	38.56	87.71	1.16	0.0009
8	0.7698	38.63	81.57	1.184	0.0006
9	0.755	38.12	83.33	1.251	0.0014
10	0.7538	38.3	82.45	1.111	0.0011
MEAN	0.76073	38.632	85.159	1.18821	0.00095
STD. DEV	0.005	0.312	2.743	0.0654	0.00029
Confidence Interval	0.757 TO 0.763	38.43 TO 38.82	83.45 TO 86.85	1.14 TO 1.22	0.0007 TO 0.001

Table 1: Quantitative Analysis of Ten Runs of DCGAN model

UNET					
Test Number	SSIM	PSNR	Keypoint	IS	FID
1	0.7511	37.58	84.21	21.3	0.009
2	0.7625	37.47	85.96	34.04	0.022
3	0.7517	37.14	90.35	24.86	0.0109
4	0.7588	37.77	87.71	30.11	0.017
5	0.7577	37.68	85.08	13.36	0.008
6	0.748	37.1	80.7	17.72	0.008
7	0.7554	37.88	82.45	15.4	0.008
8	0.7416	37.4	86.84	8.39	0.005
9	0.7482	37.67	87.71	34.13	0.013
10	0.7461	37.23	86.84	8.53	0.007
MEAN	0.75211	37.492	85.785	20.784	0.01079
STD. DEV	0.0065	0.2703	2.7981	9.7465	0.0052
Confidence Interval	0.748 TO 0.756	37.32 TO 37.65	84.05 TO 87.51	14.74 TO 26.82	0.007 TO 0.014

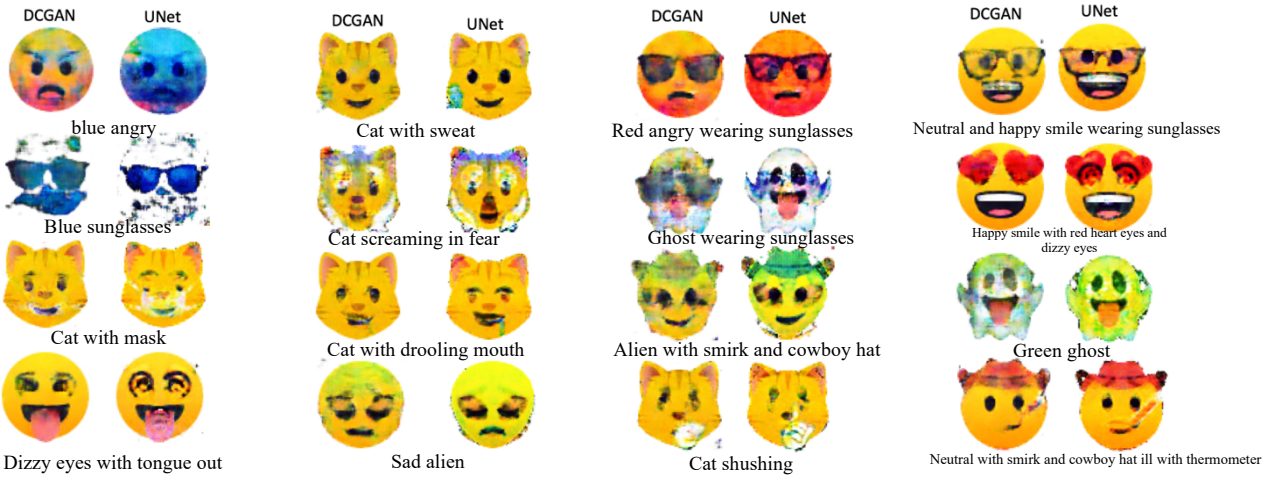
Table 2: Quantitative Analysis of Ten Runs of UNet model

Table 1 and 2 depict the quantitative results of the metrics of the DCGAN and the proposed model architecture respectively. From the results of the metrics, we can see that although the differences are minute, the DCGAN model does have better scores than the proposed architecture for SSIM, PSNR and Fréchet Inception Distance (FID). It can be assumed that these differences might get balanced out for SSIM and PSNR when more test runs are carried out. For example, if 30 test runs are carried out, it is highly likely that the SSIM and PSNR scores will be similar. This is visible from the manual inspection carried out on the regenerated training emojis for both models. Both models create similar emojis and there are no significant distortion differences between the emojis generated by the two models.

However, on analysing the Inception Score (IS) results, we can see that the proposed model's Inception Score is a lot higher than the DCGAN's Inception Score. The proposed model has a relatively high IS that has a mean of 20.784 whereas the DCGAN has a mean Inception Score of 1.18 when the least possible inception score that a model can have is 1.0. The inception score is known to be quite sensitive to noise [55,46] and small changes in noise can lead to large differences in the scores. Hence, it is safe to assume that the emojis created by the author's proposed architecture are lower in noise. This assumption is coherent with the keypoint matching percentages of the proposed architecture being higher than the DCGAN. A larger keypoint matching percentage implies less random keypoints in the image. Random keypoints are found by the SIFT algorithm when there are intensity changes due to noise. Hence, the DCGAN algorithm having a lower feature matching percentage implies that the images generated by the model have more noise. However, once again, it can be assumed that the DCGAN model might perform as well as the proposed architecture over 30 test runs. Therefore the only metric that implies the proposed model performs better than the DCGAN architecture is the Inception Score.

However, the conclusion that the DCGAN model creates images with slightly more noise than the proposed model does not align with the results of the Fréchet Inception Distance scores. The table shows that the Fréchet Inception Distance of the DCGAN is lower than the proposed model. This means that the distribution of the fake images generated by the DCGAN are a closer match than the distribution of the fake images created by the UNet. However, the Fréchet Inception Distance requires a minimum of 10,000 samples to be considered as a reliable metric [70,71]. Hence, this is not a reliable score. Yet, this was implemented in this project as it is a very popular evaluation metric for GANs.

However, the difference between the regenerated training emojis created by the DCGAN and the Proposed architecture is minimal and the re-generated emojis of the two models also do not have any large visual defects and do look very similar to each other and seem to be of good quality. Therefore, it would not be wise to switch architectures solely based on the Inception Score results. Hence, the qualitative analysis of the non-existent emojis was also carried out.



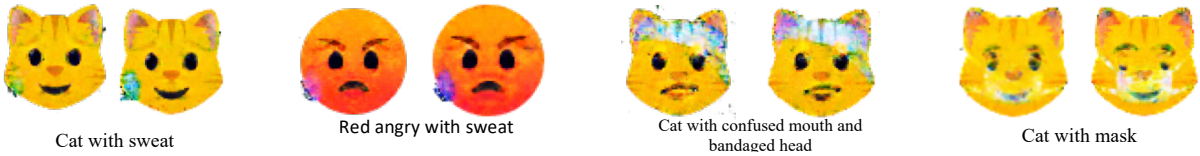
On conducting qualitative analysis of the non-existent emojis created by the two models, it can be confidently said that the author's proposed model is able to create emojis that have lesser noise and has better Text-Image Alignment capabilities. It is also visible that the DCGAN is unable to learn minute features of the emojis. Examples of these are the emojis created by using the text 'cat with drooling mouth', 'cat with sweat' and 'cat with mask'. On the other hand, the proposed architecture is able to learn the features very well. Besides this, the DCGAN model is also not able to learn colours as well as the proposed architecture. Examples of these are visible in the 'blue angry' emoji or the 'green ghost' emoji. This proves that unlike the DCGAN architecture, the proposed model is able to learn and associate colours to the words used to describe them. This implies that the proposed model has better Text-Image Alignment capabilities which help improve the quality of the emojis massively.
Hence, on conducting both qualitative and quantitative analysis of the emojis and the evaluation metrics generated, it can be confidently said that the author's proposed model performs better than the DCGAN model.

Choice of Kernel Size

Kernels of size 3 and 4 were implemented in the model to see which kernel size would lead to a better performance. It was found that, a kernel size of 4 created similar emojis to those generated when a kernel size of 3 was implemented. Quantitative evaluations also showed similar scores for all metrics to when a kernel size of 3 was chosen. However, the model took extremely long to train. The author's proposed model takes approximately 1 hour and 30 minutes to train over 980 epochs with a kernel size of 3. However, when a kernel size of 4 is used, the model takes 3 hours to run the model for 980 epochs. Due to this large computational time, a kernel size of 3 was chosen for the model.

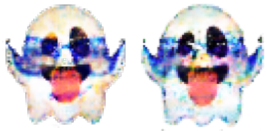
Choice of Latent Dimensionality

As mentioned in Section 4.5, [60] is a research carried out that talks about how increasing the latent dimension of GANs helps improve performance up to a certain limit after which the performance remains stable. Hence, a latent dimensionality of 100, 256 and 512 was tested to compare the quality of emojis generated by the model. It was found that regardless of whether a latent dimensionality of 256 or 512 was used, the model was able to recreate the training emojis well. Hence, D = 256 and D = 512 were tested in creating non-existent emojis. The emojis below are some of the non-existent emojis created by the model when using dimensionality of 256 (left) and 512 (right).



As visible from the emojis above, when a latent dimension of 256 was used, the model was able to create emojis of good quality too. However, the emojis created when dimensionality of 512 was used were slightly less noisy. It is important to note that there were instances where a few emojis created with latent dimensionality = 256, that were slightly better and less noisy than those created when latent dimensionality = 512. The emojis below show the emojis generated with D = 256 (left) and D = 512

(right) when the former performed slightly better than the latter. However, these differences were minute.

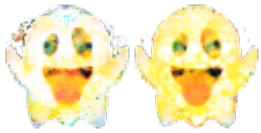

Ghost wearing sunglasses

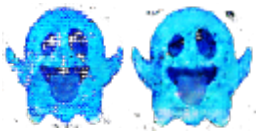

Cat screaming in fear

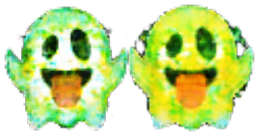

Monkey wearing sunglasses

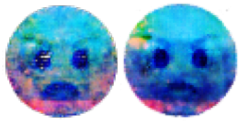

Cat shushing

However, it was noticed that when the dimensionality is 256, the model is unable to learn colours as well as when a dimensionality of 512 was used. The results are presented below where the emojis on the left are those created with a latent dimension of 256 and those on the right are created with a latent dimension of 512. The emojis on the left do not contain colours that are as bright as those on the right. They are also slightly noisier whereas those on the right have a smoother 'emoji-like' appearance.


Yellow ghost


Blue ghost


Green ghost


Blue angry

Quantitative assessments showed that all the metrics (Table 3 and 4) are approximately the same with the exception of Inception score. However, the qualitative assessments of the emojis showed that when a latent dimension of 512 was used the model had an improved ability to learn colours better thus implying better Text-Image Alignment. Since this is a major aspect and goal of this project, a latent dimension of 512 was chosen. Similar results were observed when a dimensionality of 100 was used.

UNET/D = 256							
Test Number	SSIM	PSNR	Keypoint	IS	IS w/o TL	FID	FID w/o TL
1	0.7503	37.16	85.96	18.61	1.00	0.027	381.91
2	0.7502	37.33	85.08	14.1	1.00	0.007	379.87
3	0.751	37.38	84.21	25.33	1.00	0.032	380.62
4	0.7603	37.55	85.96	16.42	1.00	0.009	382.37
5	0.7468	37.14	87.71	19.07	1.00	0.009	381.54
6	0.7554	37.93	82.45	24.27	1.00	0.011	382.74
7	0.7489	37.21	77.19	28.02	1.00	0.033	383.69
8	0.7534	37.75	88.59	20.7	1.00	0.02	381.59
9	0.758	37.82	83.33	37.91	1.00	0.013	381.95
10	0.7545	37.8	85.08	36.27	1.00	0.012	380.35
MEAN	0.75288	37.507	84.556	24.07	1.00	0.0173	381.663
STD. DEV	0.0042	0.3008	3.1803	8.0395	0.0000	0.0100	1.1527
Confidence Interval	0.7502 TO 0.7554	37.32 TO 37.69	82.58 TO 86.52	19.08 TO 29.05	1	0.011 TO 0.023	380.94 TO 382.37

Table 3: Quantitative Analysis of model with Latent Dimension of 256

UNET/ D = 512							
Test Number	SSIM	PSNR	Keypoint	IS	IS w/o TL	FID	FID w/o TL
1	0.7511	37.58	84.21	21.3	1.00	0.009	381.34
2	0.7625	37.47	85.96	34.04	1.00	0.022	379.53
3	0.7517	37.14	90.35	24.86	1.00	0.0109	381.16
4	0.7588	37.77	87.71	30.11	1.00	0.017	380.8
5	0.7577	37.68	85.08	13.36	1.00	0.008	380.07
6	0.748	37.1	80.7	17.72	1.00	0.008	381.61
7	0.7554	37.88	82.45	15.4	1.00	0.008	379.47
8	0.7416	37.4	86.84	8.39	1.00	0.005	380.74
9	0.7482	37.67	87.71	34.13	1.00	0.013	380.73
10	0.7461	37.23	86.84	8.53	1.00	0.007	381.06
MEAN	0.75211	37.492	85.785	20.784	1.00	0.01079	380.651
STD. DEV	0.0065	0.2703	2.7981	9.7465	1.00	0.0052	0.7338
Confidence Interval	0.748 TO 0.756	37.32 TO 37.65	84.05 TO 87.51	14.74 TO 26.82	1.00	0.007 TO 0.014	380.19 TO 381.10

Table 4: Quantitative Analysis of model with Latent Dimension of 512

6.3 GloVe VS Word2vec VS FastText

This section presents the qualitative and quantitative analysis of the emojis, and the evaluation metrics produced by the proposed model when using each of the three Word Embeddings used in this project which are GloVe, Word2vec and FastText.

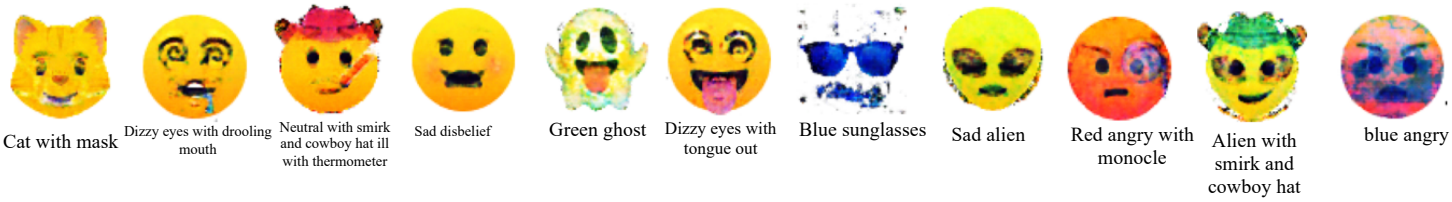
Presented below are the non-existent emojis generated when using each of these Word Embedding algorithms as well as the results of the evaluation metrics. Following this, an analysis is done on each of these group of emojis.

- Manual Inspection of non-existent emojis (Qualitative Analysis)

1. GloVe

2. Word2vec

3. FastText



On analysing the emojis generated by the proposed model when using each of the three word embedding models, it can be easily said that the GloVe model performs the best. The non-existent emojis created by the GloVe model have all the features one would expect from the text inputted. For instance, in the 'cat with mask' emoji, the mask is clearly visible for the GloVe model, but the same cannot be said for the Word2vec and FastText model. Similarly, for the 'neutral with smirk and cowboy hat ill with thermometer' emoji, the hat is well defined for the GloVe model, but it is slightly deformed for the Word2vec and FastText models. The same can be said for the 'sad alien' emoji where, the alien looks neutral for the Word2vec and FastText models, but the expression of sadness is clearly visible for the emoji created by the GloVe model.
Besides this, the GloVe model is also found to be able to learn colours the best. For instance, in the 'blue angry' emoji, both Word2vec and GloVe are able to generate the emoji well. However, the Word2vec 'blue angry' emoji has a slight deformity which makes it hard to see the angry expression of the emoji. However, the FastText model completely fails to learn the colour blue. This result is also visible for the 'green ghost' emoji. In this case, the GloVe model creates the best emoji whereas the remaining two word embedding algorithms fail to learn the colours as well.

Finally, the GloVe model also creates emojis that have lesser noise when compared to Word2vec and FastText. This is visible in the 'blue sunglasses', 'drooling mouth with dizzy eyes' and the 'dizzy eyes with tongue out' emoji. Overall, the GloVe model performs better than the Word2vec and FastText models when it comes to creating non-existent emojis.

- Quantitative Analysis

The author's proposed architecture was run ten times for each of the word embeddings (GloVe, FastText and Word2vec) and the results were compiled, and the mean, standard deviation and confidence intervals of the observations were found in the same way as mentioned in Section 6.2.

UNET/Glove						
Test Number	SSIM	PSNR	Keypoint	IS	FID	FID w/o TL
1	0.7511	37.58	84.21	21.3	0.009	381.34
2	0.7625	37.47	85.96	34.04	0.022	379.53
3	0.7517	37.14	90.35	24.86	0.0109	381.16
4	0.7588	37.77	87.71	30.11	0.017	380.8
5	0.7577	37.68	85.08	13.36	0.008	380.07
6	0.748	37.1	80.7	17.72	0.008	381.61
7	0.7554	37.88	82.45	15.4	0.008	379.47
8	0.7416	37.4	86.84	8.39	0.005	380.74
9	0.7482	37.67	87.71	34.13	0.013	380.73
10	0.7461	37.23	86.84	8.53	0.007	381.06
MEAN	0.75211	37.492	85.785	20.784	0.01079	380.651
STD. DEV	0.0065	0.2703	2.7981	9.7465	0.0052	0.7338
Confidence Interval	0.748 TO 0.756	37.32 TO 37.65	84.05 TO 87.51	14.74 TO 26.82	0.007 TO 0.014	380.19 TO 381.10

Table 5: Quantitative Analysis of Ten Runs of GloVe Word Embeddings

UNET/Word2Vec						
Test Number	SSIM	PSNR	Keypoint	IS	FID	FID w/o TL
1	0.7596	38	82.45	64.34	0.031	378.45
2	0.7634	37.82	83.33	61.47	0.115	381.97
3	0.7567	37.95	83.33	55.44	0.1	380.3
4	0.7633	38.33	83.33	81.52	0.049	378.8
5	0.7625	37.78	88.59	78.82	0.142	379.09
6	0.7502	37.85	80.7	65.87	0.04	380.68
7	0.7551	37.43	85.96	77.8	0.23	379.82
8	0.7561	37.7	90.35	75.92	0.21	381.85
9	0.7661	37.91	85.08	58.92	0.108	383.94
10	0.756	37.67	85.08	62.12	0.148	382.15
MEAN	0.7589	37.844	84.82	68.222	0.1173	380.705
STD. DEV	0.0049	0.2365	2.8960	9.3896	0.0678	1.7555
Confidence Interval	0.755 TO 0.761	37.69 TO 37.99	83.02 TO 86.61	62.40 TO 74.04	0.075 TO 0.159	379.616 TO 381.793

Table 6: Quantitative Analysis of Ten Runs of Word2vec Word Embeddings

UNET/FastText						
Test Number	SSIM	PSNR	Keypoint	IS	FID	FID w/o TL
1	0.7545	37.88	90.35	105.86	0.279	379
2	0.7536	37.62	88.59	91.74	0.213	381.82
3	0.7624	37.96	87.71	88.73	0.647	381.15
4	0.7519	38.13	83.33	93.39	0.063	380.49
5	0.7549	37.49	89.47	99.25	0.043	381.45
6	0.758	38.05	88.59	92.115	0.064	381.11
7	0.7436	37.69	86.84	90.13	1.114	382.74
8	0.7626	37.75	85.96	105.14	0.14	378.05
9	0.7619	38.7	86.84	101.18	0.0343	383.88
10	0.749	38.27	83.33	95.84	0.309	383.2
MEAN	0.75524	37.954	87.101	96.3375	0.29063	381.289
STD. DEV	0.0062	0.3559	2.3770	6.1860	0.3439	1.8012
Confidence Interval	0.751 TO 0.759	37.73 TO 38.17	85.62 TO 88.57	92.50 TO 100.17	0.077 TO 0.503	380.172TO 382.405

Table 7: Quantitative Analysis of Ten Runs of FastText Word Embeddings

As visible from the confidence intervals of the evaluation metrics, we can see that all three Word Embedding algorithms have similar performance for all evaluation metrics except for Inception Score and Fréchet Inception Distance. On conducting manual inspection of the emojis regenerated by the models, there were no extreme defects caused by the three Word Embedding algorithms, in fact they looked similar to each other. However, as mentioned earlier, the Inception Score is known to be very sensitive to changes in images like blurring or small artifacts in synthetic images [55,46]. This serves as an explanation for the large Inception Score difference amongst the three models. Hence, the large difference in the Inception scores, does not imply that the emojis created by the GloVe model which has the least inception score are of extremely poor quality. It is solely because the Inception score is quite sensitive to these changes. On the other hand, the Fréchet Inception Distance is known to be more stable to such changes, due to its increased consistency at evaluating GANs [46] and this is visible from the results obtained. Comparing the standard deviations of the FID scores of each of the

three word embedding techniques, we can see that there is not a very large difference like the Inception Score where the scores differ a lot over the ten runs. This is due to the greater stability and lesser sensitivity of FID over IS. This is one of the reasons why the FID score is preferred over the Inception Score.
It is also visible that the FID score of the GloVe model is the least. However, as mentioned earlier, this is not a reliable score as a minimum of 10,000 samples is needed to make a robust estimation of FID. Hence, it is not taken into consideration [70,71].
During manual inspection of regenerated training emojis, it was found that, GloVe was successfully able to generate all emojis well and all emojis were properly linked to the captions they were trained with. However, in the case of Word2vec and FastText, although both the models created clear images with minimal artifacts, there were times when the model was unable to link the right emoji to the text.

Overall, considering the quantitative scores mentioned in Table 5, 6 and 7, it is safe to say that all models performed well. The performance of the three models differs based on the dataset. For some datasets GloVe performs well whereas for others Word2vec or FastText may perform better. However, considering that GloVe was able to generate all emojis well unlike Word2vec and FastText and was also able to create much better non-existent emojis which have a lot lower noise, depict the features one would expect from the text clearly and precisely and are also able to learn colours better, indicating better Text-Image Alignment, it is safe to say that the GloVe Word Embedding algorithm performs the best. Word2vec and FastText can be considered to have similar performances based on the quantitative and qualitative analysis carried out by the researcher. However, as mentioned in Section 3.1 since FastText is based on character n-grams, it is able to learn words that are not present in the corpora too. Due to this feature of the FastText, it can be considered that FastText may be able to create emojis well when using certain text that Word2vec may not be able to do.

6.3 Results using Transfer Learning

The following table (Table 6) depicts the Inception Score and Fréchet Inception Distance of the author's proposed architecture with the GloVe word embedding models when using the Inception V3 model, and when implementing Transfer Learning.

UNET							
Test Number	SSIM	PSNR	Keypoint	IS	IS w/o TL	FID	FID w/o TL
1	0.7511	37.58	84.21	21.3	1.00	0.009	381.34
2	0.7625	37.47	85.96	34.04	1.00	0.022	379.53
3	0.7517	37.14	90.35	24.86	1.00	0.0109	381.16
4	0.7588	37.77	87.71	30.11	1.00	0.017	380.8
5	0.7577	37.68	85.08	13.36	1.00	0.008	380.07
6	0.748	37.1	80.7	17.72	1.00	0.008	381.61
7	0.7554	37.88	82.45	15.4	1.00	0.008	379.47
8	0.7416	37.4	86.84	8.39	1.00	0.005	380.74
9	0.7482	37.67	87.71	34.13	1.00	0.013	380.73
10	0.7461	37.23	86.84	8.53	1.00	0.007	381.06
MEAN	0.75211	37.492	85.785	20.784	1.00	0.01079	380.651
STD. DEV	0.0065	0.2703	2.7981	9.7465	1.00	0.0052	0.7338
Confidence Interval	0.748 TO 0.756	37.32 TO 37.65	84.05 TO 87.51	14.74 TO 26.82	1.00	0.007 TO 0.014	380.19 TO 381.10

Table 8: Inception Score and Fréchet Inception Distance with and without Transfer Learning

As mentioned in Section B.2 in Chapter 5, Transfer Learning was implemented due to the Inception V3 model's incapability to classify emojis. This is because the Inception V3 model was trained on the ImageNet dataset that had 1000 classes, but emojis were not one of them.

Due to this, as visible from the results above, we can see that the model was assigned a very low Inception Score of 1.0, which is the least possible score. Along with this, the emojis were a Fréchet Inception Distance of approximately 380.6. This measures the distance between the distributions of the real and the generated emojis and the model has been assigned a very large score.
When we implement transfer learning on the discriminator model and convert it into a classifier capable of classifying all 114 emojis, there is a major improvement in the scores produced. Hence, transfer learning was implemented [1].

6.4 Caption Loss Results

The caption loss model was generated using the steps mentioned in Chapter 5, Section B.3.2. However, on conducting the caption generation process on the generated image, it was found that the model was unable to produce a caption that was a depiction of the features in the image. This was due to the lack of data. Only 114 emojis from the real dataset were used to create the model. This lack of data led to overfitting of the model.

Usually, a caption generator is created by using large datasets like the Flickr8K or the Flickr30K. The Flickr8K is a dataset that contains 8000 images, each of which are assigned 5 captions that describe it. Hence, it consists of a total of 8000 images and 40,000 captions. Despite the large number of images and captions, this model is prone to overfitting and the LSTM used in the model trains very quickly [52].
Several remedies to overfitting were implemented such as implementing kernel regularisers and recurrent regularisers. However, despite these additions, the model was still overfitting.

The model is successfully able to create captions. However, these captions do not reflect the features of the image. Hence, the same model was then run on the real images to check the captions generated. It was found that, there was some level of similarity between the captions generated for the real and the generated image.
Caption Loss was also attempted using the Features extracted by the Transfer Learned model. In the case of the transfer learned model also the model was found to be overfitting. Due to this, the model was unable to create captions that represent the features of the image. Because the Transfer Learned model is better than the Inception V3 model at extracting features of emojis (because the Inception V3 is not trained to classify emojis), the former model is preferred.

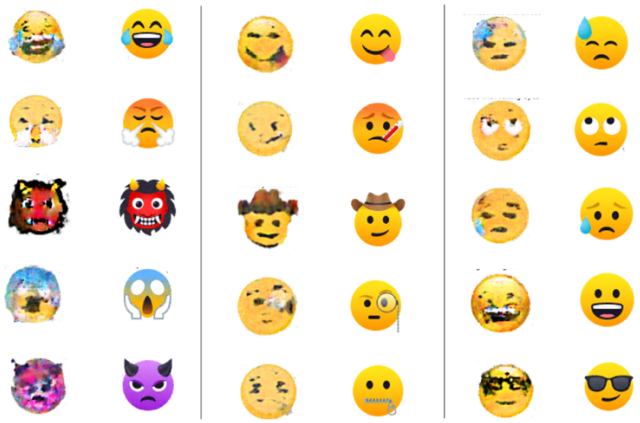
6.5 Comparison with previous Research

This section compares the results of the previous Adversarial networks that have attempted Text to Emoji Synthesis with the emojis created by the authors proposed model. As mentioned in Section 2.4, so far only two papers have implemented Text to Emoji Synthesis using Adversarial Networks. The following shows the comparison of the two papers with the synthetic emojis created by the proposed model.

Paper 1:

This paper is titled ‘EmotiGAN: Emoji Art using Generative Adversarial Networks’ [1] by Marcel Puyat and was written in 2017. The author of this paper implemented a standard Deep Convolutional Generative Adversarial Network using the Word2vec Word Embedding model and have used Inception score as an evaluation metric and have utilised the transfer learning method used by the author in the present paper.

Below shown is a comparison of the training emojis recreated by the model (left) and the emojis created by the proposed model (right).

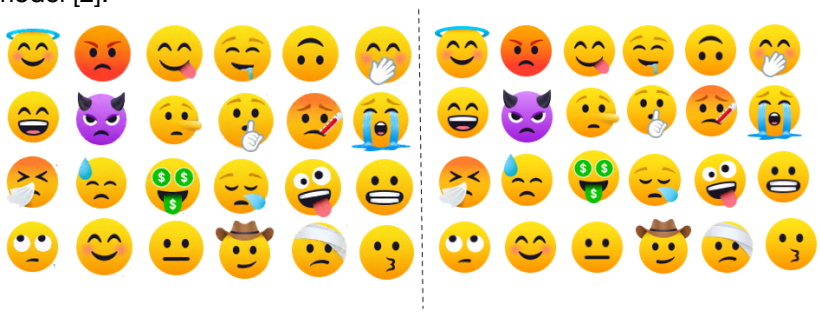


It is clearly visible that the emojis created by the proposed model is a lot less noisy and more precise than the emojis recreated by the DCGAN model [1]. Besides this, [1] is unable to create any non-existent emoji. The superior performance of the proposed model in this paper, can be owed to the modified captions and increased latent dimension that improved Text-Image Alignment, the replication of emojis used during training as well as the replacement of the DCGAN generator with a UNet which has led to better results as shown in Section 6.3. The proposed model has also used GloVe instead of Word2vec as it has been shown to create a better mapping between the text and the features of the emoji, thus leading to creation of better, more precise emojis that match the text used to describe them perfectly. The model [1] was able to get an inception score of 4.9 as compared to the proposed model which has an inception score of approximately 20.784.

Paper 2:

This paper is titled 'Conditional Generative Adversarial Networks for Emoji Synthesis with Word Embedding Manipulation' by N. Dianna Radpour et al [2] in 2017. This paper also made use of the DCGAN architecture like [1] and was successfully able to recreate emojis of good quality as shown in the image below. The emojis on the left are the synthetic images created by the model and the emojis on the right are the emojis used for training.

Although they do present better results than [1], the emojis recreated by the proposed model are of much better quality. As visible from the emojis in the image below, the synthetic emojis created by the model (left) are almost similar to the training emojis on the right and do not contain noise like the ones created by the model [2].



The paper [2] also attempted to create non-existent emojis, and although they were able to create emojis that contained features of the two emojis that they tried to mix, they are quite noisy and cannot be used as actual emojis. On the other hand, the proposed model has been successful at creating better emojis, which are less noisy and all the features that one would expect from the text inputted are clearly visible. This is because of three reasons : a) The modified captions implemented by the proposed method has helped improve results when creating non-existent emojis by improving Text-Image Alignment b) the modified architecture of the proposed model and c) The model [2] uses a method of averaging the word embeddings of the text of the two emojis to create a new emoji. This creates a new emoji that shows all the features of the two emojis that were averaged together whereas the proposed model sums up the word embeddings of the individual words used to create the new emoji and helps create an emoji with only those features that the user wants. This is a better alternative as it gives the user more control and generates a clearer and more accurate emoji as visible in the images shown below.

The emojis on the left are the emojis created by [2] whereas the emojis on the right are the emojis created by the proposed model.

As discussed in earlier sections, replication of emojis helps improve emoji quality [41]. Comparing the non-existent emojis of the DCGAN architecture (without UNet generator) created by the author (Section 6.2) and the DCGAN architecture implemented by [2], one can clearly see that the former model creates much better emojis. Other than the modification of captions which helps improve Text-Image Alignment, the significant difference between the implementation of the author's DCGAN model and [2] is the replication of emojis. This proves that replication of emojis have played a role in improving image quality. This aligns with the research carried out by [41].

Overall, the proposed model (UNet generator) outperforms all previous research. This is owed to the combination of the UNet generator architecture and the Replication of emojis which contributes to an improvement in Image Quality , as well as the combination of the modifications made to the caption as well as the increase in Latent Dimension which has improved Text-Image Alignment.

Summary

This chapter provides the reader with the results of the proposed model's emojis and creates a comparison among the various choices the author has considered before reaching the final architecture and choice of Word Embeddings in both a qualitative and quantitative manner. Finally, it also compares the results of the previous research papers with the proposed models results and it is clearly evident from the results that the proposed method outperforms all previous research.

CHAPTER 7 : CHALLENGES AND FUTURE WORK

Introduction

This chapter discusses the various challenges and difficulties faced during the course of this project. It talks about the various challenges faced while training the GAN, creating a model with strong Text-Image alignment using Word Embeddings, problems with the data and so on. It then goes on to talk about how some of these challenges were overcome by the author and what can be done in the future to solve the problems that could not be solved due to lack of resources.

7.1 Challenges faced while training the GAN

Generative Adversarial Networks (GANs) are known to be notoriously difficult to train. This is due to the fact that it is important to create a balance between the generator and the discriminator during training in order to ensure that one does not overpower the other. Below are some of the challenges faced by the author while training the GAN:

1. Mode Collapse : Mode collapse occurs when the generator learns a particular output with which it is able to successfully fool the discriminator and hence, maps all values of z (noise) to the same output. This means that, regardless of the text inputted by the user or the noise used to generate the images, the model learns to output only one type of image. The model presented in this project also faced this issue and was overcome by adding Gaussian Noise that has a standard deviation of 0.2 to the input of the discriminator [57]. This noise helps stabilise the model, reduces overfitting and helps eliminate mode collapse. It can also be solved by increasing the dimension of the latent space [56,58]. For the proposed model, the problem of mode collapse was solved by solely adding gaussian noise to the discriminator. However, the latent dimension was also increased as it helped improve results by creating less noisy images.
2. Convergence Failure : Convergence Failure occurs when the GAN is not successful at finding an equilibrium between the discriminator and the generator. This usually occurs when the discriminator loss goes to zero, i.e., when the discriminator overpowers the generator. This means that the generator creates unrealistic outputs that the discriminator is easily able to identify as real or fake. It can usually be identified when the generator creates very noisy, static images. In order to solve this problem, it is vital to ensure that the generator architecture is not very weak compared to the discriminator so that the discriminator does not overpower the generator.
3. Batch Size and Learning Rate : Choosing an appropriate Batch Size and Learning Rate to train the GAN is vital as it was found that using very large batch sizes could lead to convergence failure as during the initial stages of training, the discriminator would be exposed to a lot of training data making it very strong, thus overpowering the generator [58] . On the other hand, using a very small batch size means that the model learns very slowly. Similarly, using a large learning rate could mean that the model is learning very quickly, and it could affect the training process negatively. Hence, both the learning rate and the batch size was chosen through trial and error

7.2 Text-Image Alignment for Text to Emoji Synthesis

As mentioned earlier, Text-Image Alignment is vital for Text to Image synthesis as the goal is not just to create good quality, realistic images, but also to be able to create images that are related to the text inputted. This was a big hurdle faced during the course of this project.

This issue was mostly solved by modifying the captions and replication of emojis outlined in Section 4.6 and Section 4.3 respectively. However, the author believes that the Word Embeddings used can also play a major role in this. Using embedding algorithms like BERT or ELMO which are context aware embeddings could lead to better results This allows each word to have multiple representations based on the context. This can lead to a better mapping between the text and the emoji features and could ultimately lead to better generation of non-existent emojis. This was attempted in this study, however,

BERT and ELMO both have embeddings of a dimension of 768 which is double the dimensions used for Word2vec, FastText and GloVe (dimension = 300). This dimensionality can be reduced using Principal Component Analysis (PCA). However, due to the lack of computational power and time, this could not be implemented at the current stage of the project. With greater computational power and higher RAM, this can be implemented in the future to see how it would affect the quality of emojis as well as the Text-Image Alignment.

7.3 Issues with Data

The biggest issue faced while carrying out this project, was the lack of data. This lack of images, and the fact that there is only one text for each emoji has led to issues in training the GAN as well as carrying out evaluation using methods like Fréchet Inception Distance which needs a minimum of 10,000 samples [70,71] to be a reliable score as well as Caption Loss for which the model could not be trained due to lack of data. This lack of data explains why this topic has not been researched as much as most Text to Image Synthesis models.
As mentioned earlier, due to lack of processing power all emojis available in the JoyPixels dataset could not be replicated. Only 114 emojis were used. In the future, with more processing power, the amount of data used can be increased, which will not only lead to the creation of greater number and more diverse emojis but will also reduce the overfitting in the Caption Loss model thereby making it possible to use it as an evaluation metric. It will also make Fréchet Inception Distance a more reliable metric.

7.4 Issues with Evaluation

As mentioned in Section 5.1, GANs are extremely difficult to evaluate due to the lack of an objective function. However, through research several metrics have been researched that can be used for the purpose of evaluation. However, unlike other neural networks there is no one evaluation metric that proves to be a robust method of quantitatively and qualitatively evaluating the performance of the GAN. This is the reason the author has chosen to use multiple evaluation methods to evaluate the GAN. The metrics used in this GAN are not perfect and do not provide scores that are always reflective of the quality of the image and the performance of the GAN. However, using this mixture of evaluation metrics allows the researcher to carefully evaluate the scores while keeping the problems of each evaluation metric in mind to decide which architecture or which word embedding performs the best.

CHAPTER 8 : CONCLUSION

In 2022, a total of 5.03 billion people around the world use the internet and hence have access to social media. Emojis are an important aspect of digital communication. By adding an emotive element to texts or messages, they help users understand the emotions or intention behind each text. Besides this, with the rise of trends on social media platforms like TikTok or Instagram, people want to use emojis that relate to certain trends. Hence, giving users the ability and the power to make emojis as and when they wish is a huge advantage to any social media platform. This project aimed at solving this issue. The research uses Deep Learning and Natural Language Processing to build a model that is capable of carrying out Text to Emoji Synthesis using Adversarial Networks.

The following are the significant findings and contributions made by this research:

1. This research successfully implemented a modified version of the Deep Convolutional Generative Adversarial Network (DCGAN). This involved replacing the generator of the DCGAN with a UNet. It was found that this modification helped improve the quality of emojis created as well as aided in improving the Text-Image Alignment which is an import aspect of Text to Image Synthesis models.
2. This research implemented the research carried out by [41]. This involved replicating emojis when only a limited dataset is available. It was found that doing this improved the quality of emojis created.
3. Most Text to Image Synthesis models use a latent dimension of 100. This model was trained using a latent dimension of 512. It was found that using this dimension helped improve results and allowed the model to learn features better. A latent dimension of 512 was able to learn colours better than lower latent dimensions. This implies that using a dimensionality of 512 allows the model to learn features in a better manner.
4. This research also focussed on improving Text-Image Alignment which is a major aspect of any Text to Image Synthesis model. The caption modifications, UNet generator architecture and increase of latent dimensionality were found to improve the Text-Image alignment thereby improving the emoji generated.
5. A study was also conducted where three Word Embedding Algorithms were compared to find the best performing algorithm for Text to Emoji Synthesis. It was found that the GloVe Word Embeddings created the best emojis. This study was vital as all previous research in this field implemented the Word2Vec model. This study proves that for the task of Text to Emoji Synthesis, the GloVe model has been found to have a far superior performance when compared to the other two models.
6. GANs are known to be very difficult to evaluate and do not have a robust evaluation metric that can be used to judge its performance. The presently available metrics do have various drawbacks such as the need for a large sample size due to which they cannot be used as a robust metric for this project. Hence, a total of 7 metrics were implemented in order to make the analysis of the best performing models or Word Embeddings as accurate as possible. These are Manual Evaluation, SSIM, PSNR, Inception Score, Fréchet Inception Distance, Feature Matching and Caption Loss. Feature Matching using SIFT is an evaluation metric that has not been used in previous research and has shown great promise through its implementation this research. Caption Loss is another metric that showed great promise, but due to the lack of data the model was overfitting and was unable to produce the expected output.

Overall, this project was an interesting experiment, and the model was successfully able to create new emojis of good quality. Although the emojis created by the model are not perfect, the proposed model has outperformed previous adversarial research and generated emojis that are good representations of the text inputted. The author believes that this work is worth further research. It would be interesting to see how the model will perform on replacing the GloVe embeddings with context-aware word embeddings like BERT or ELMO. The author is confident that with access to more processing power, a greater number of emojis can be replicated. This will not only allow the model to learn more features, but it will also lead to the generation of better quality emojis than those created by the model at present.

References

[1] Marcel Puyat, EmotiGAN: Emoji Art using Generative Adversarial Networks , 2017
[2] N. Dianna Radpour et al, Conditional Generative Adversarial Networks for Emoji Synthesis with Word Embedding Manipulation, 2017
[3] https://home.unicode.org, The official Unicode website that is the standard for text and emojis
[4] Alec Radford, Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, 2016
[5] David G. Lowe, Distinctive Image Features from Scale-Invariant Keypoints, 2004
[6] Sepp Hochreiter et al, Long Short-Term Memory, 1997
[7] Felix A. Gers et al, Learning to Forget : Continual Prediction with LSTM, 1999
[8] Xiaojin Zh et al, A Text-to-Picture Synthesis System for Augmenting Communication, 2007
[9] Diederik P. Kingma et al, Auto-Encoding Variational Bayes, 2013
[10] Karol Gregor et al, DRAW: A Recurrent Neural Network For Image Generation, 2013
[11] Ian J. Goodfellow et al, Generative Adversarial Nets, 2014
[12] C. Ledig et al, Photo-realistic single image super-resolution using a generative adversarial network 2016
[13] Raymond A et al, Semantic Image Inpainting with Deep Generative Models, 2016
[14] L.A.Gatys et al, Image style transfer using convolutional neural networks, 2016
[15] M.Frid-Adar et al, Gan-based synthetic medical image augmentation for increased cnn performance in liver lesion classification, 2018
[16] Mehdi Mirza, Conditional Generative Adversarial Nets, 2014
[17] S. E. Reed et al, Generative adversarial text to image synthesis, 2016
[18] Han Zhang et al, StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks, 2016
[19] Han Zhang et al, StackGAN++: Realistic Image Synthesis with Stacked Generative Adversarial Networks, 2019
[20] Tao Xu et al, AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks, 2017
[21] K J Joseph et al, C4Synth: Cross-Caption Cycle-Consistent Text-to-Image Synthesis, 2018
[22] Jun Cheng et al, RiFeGAN: Rich Feature Generation for Text-to-Image Synthesis from Prior Knowledge, 2020
[23] Shikhar Sharma et al, ChatPainter: Improving Text to Image Generation using Dialogue, 2018
[24] L. Theis et al, A note on the evaluation of generative models, 2016
[25] Tim Salimans et al, Improved Techniques for Training GANs, 2016
[26] Martin Heusel et al, GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium, 2017
[27] T. Xu et al, Attngan: Fine-grained text to image generation with attentional generative adversarial networks, 2017 (R-precision)
[28] Alex Shonenkov et al, Emojich - zero-shot emoji generation using Russian Language: A technical report, 2022
[29] Dan Jurafsky et al, Speech and Language Processing 3rd edition draft, Chapter 2 : Regular Expressions, Text Normalizations, Edit Distance, 2021
[30] Tomas Mikolov et al, Efficient Estimation of Word Representations in Vector Space, 2013
[31] Piotr Bojanowski et al, Enriching Word Vectors with Subword Information, 2016
[32] Jeffrey Pennington et al, GloVe: Global Vectors for Word Representation, 2014
[33] Phillip Isola et al, Image-to-Image Translation with Conditional Adversarial Networks, 2016
[34] Jun-Yan Zhu et al, Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, 2017
[35] Tero Karras et al, A Style-Based Generator Architecture for Generative Adversarial Networks, 2018
[36] Olaf Ronneberger et al, U-Net: Convolutional Networks for Biomedical Image Segmentation, 2015
[37] Karen Simonyan et al, Very Deep Convolutional Networks for Large-Scale Image Recognition, 2014
[38] M. Z. Hossain et al, A comprehensive survey of deep learning for image captioning, 2019
[39] https://joypixels.com - Dataset used by author for the project

[40] https://gradio.app
[41] Yuki Eizuka et al, Impact of Duplicating Small Training Data on GANs, 2021
[42] https://github.com/soumith/ganhacks
[43] https://nlp.stanford.edu/projects/glove/
[44] Jost Tobias Springenberg et al, Striving for Simplicity: The All Convolutional Net, 2015
[45] https://pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/
[46] Ali Borji, Pros and Cons of GAN Evaluation Measures: New Developments, 2021
[47] Zhou Wang et al, Image Quality Assessment: From Error Visibility to Structural Similarity, 2004
[48] Christian Szegedy et al, Rethinking the Inception Architecture for Computer Vision, 2014
[49] https://image-net.org/download.php
[50] Feng Guo et al, Research on Image Detection and Matching Based on SIFT Features, 2018
[51] https://docs.opencv.org/3.4/dc/dc3/tutorial_py_matcher.html
[52] https://medium.com/@raman.shinde15/image-captioning-with-flickr8k-dataset-bleu-4bcba0b52926
[53] Kishore Papineni, BLEU: a Method for Automatic Evaluation of Machine Translation, 2002
[54] https://videoprocessing.ai/metrics/ways-of-cheating-on-popular-objective-metrics.html
[55] Yaniv Benny et al, Evaluation Metrics for Conditional Image Generation, 2021
[56] https://neptune.ai/blog/gan-failure-modes
[57] https://developers.google.com/machine-learning/gan/problems
[58] https://medium.com/intel-student-ambassadors/tips-on-training-your-gans-faster-and-achieve-better-results-9200354acaa5
[59] S. Reed, Z. Akata, S. Mohan, S. Tenka, B. Schiele, and H. Lee. Learning what and where to draw. In NIPS, 2016.
[60] Ivana Marin et al, The Effect of Latent Space Dimension on the Quality of Synthesized Human Face Images, 2021
[61] https://www.makeuseof.com/how-emojis-have-changed-the-way-we-communicate/
[62] Rui Zhou et al., A survey on generative adversarial network-based text-to-image synthesis, 2021
[63] Stanislav Frolov et al., Adversarial text-to-image synthesis: A review, 2021
[64] https://machinelearningmastery.com/how-to-evaluate-generative-adversarial-networks/
[65] Nahian Siddique et al., U-Net and its variants for medical image segmentation : theory and applications, 2020
[66] Pius Kwao Gadosey et al., SD-UNet : Stripping down U-Net for Segmentation of Biomedical Images on Platforms with Low Computational Budgets.
[67] https://www.mathworks.com/help/images/image-quality-metrics.html
[68] Valery Starovoitov, Comparative analysis of the SSIM index and the Pearson coefficient as a criterion for image similarity, 2020
[69] https://www.analyticsvidhya.com/blog/2019/10/detailed-guide-powerful-sift-technique-image-matching-python/
[70] https://github.com/bioinf-jku/TTUR
[71] https://wandb.ai/ayush-thakur/gan-evaluation/reports/How-to-Evaluate-GANs-using-Frechet-Inception-Distance-FID---Vmlldzo0MTAxOTI
[72] http://semanticgeek.com/technical/a-count-based-and-predictive-vector-models-in-the-semantic-age/
[73] https://machinelearningmastery.com/what-are-word-embeddings/
[74] https://nlp.stanford.edu/projects/glove/
[75] https://towardsdatascience.com/light-on-math-ml-intuitive-guide-to-understanding-glove-embeddings-b13b4f19c010
[76] https://kavita-ganesan.com/fasttext-vs-word2vec/#.YygBxy0RpQI
[77] https://www.analyticsvidhya.com/blog/2021/03/basics-of-neural-network/
[78] https://www.ibm.com/cloud/learn/recurrent-neural-networks
[79] https://jonathan-hui.medium.com/gan-dcgan-deep-convolutional-generative-adversarial-networks-df855c438f
[80] https://en.wikipedia.org/wiki/Natural_language_processing
[81] https://www.ibm.com/cloud/learn/convolutional-neural-networks
[82] https://towardsdatascience.com/word-embeddings-deep-dive-hands-on-approach-a710eb03e4c5
[83] https://www.mygreatlearning.com/blog/types-of-neural-networks/
[84] https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[85] https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/
[86] https://www.scribbr.com/statistics/confidence-interval/

CODE REFERENCES

The following are the references the author has used to write the code for this project:
[1] https://github.com/codekaust/emotigan/blob/master/MODEL.ipynb
[2] https://github.com/nithintata/image-caption-generator-using-deep-learning/blob/master/image-captioning.ipynb
[3] https://machinelearningmastery.com/how-to-implement-the-frechet-inception-distance-fid-from-scratch/
[4] https://www.thepythoncode.com/article/sift-feature-extraction-using-opencv-in-python
[5] https://ourcodeworld.com/articles/read/991/how-to-calculate-the-structural-similarity-index-ssim-between-two-images-with-python
[6] https://www.geeksforgeeks.org/python-peak-signal-to-noise-ratio-psnr/
[7] https://github.com/legokichi/keras-pix2pix/blob/master/model_unet.py
[8] https://github.com/gucci-j/emoji-gan/blob/master/inception_score.py
[9] https://towardsdatascience.com/saving-and-loading-keras-model-42195b92f57a
[10] https://gradio.app/

All data pre-processing (image and text) was implemented by the author and the code for this was written by themselves. [1] was used as a reference to build the model architecture and to carry out the training as well. However, several modifications were made to the code that was used to build the architecture. [7] was used as a reference to build the UNet architecture for the generator. However, once again, modifications were made to this code as well. All code written to import the Word Embeddings from the genism library was also written by the author. The SSIM and PSNR architecture were implemented by referring to [5] and [6] respectively. The code to evaluate the model using Feature Matching was written by the author themselves after some online research from [4]. The entire code for implementing Transfer Learning to convert the discriminator into a multi-class classifier was written by the author themselves with the exception of a small chunk of code used to save and load Keras models which was adapted from [9]. Inception Score and Fréchet Inception Distance were implemented by using [8] and [3] as references respectively. This code was then modified to make it applicable to the Transfer Learned model when implementing these metrics using Transfer Learning. The caption loss model using the Inception V3 model was implemented by referring to [2] and the modifications made to apply the model to the transfer learned model were done entirely by the author. Finally, the Gradio application was also coded entirely by the author after reading the documentations present in [10].

Appendix A : GitLab Repository

To run the proposed model developed in this research, access the GitLab Repository using the link:
https://git-teaching.cs.bham.ac.uk/mod-msc-proj-2021/txs104/

The repository includes the following files:

1. A python notebook named 'FinalProject.ipynb' which contains the entire code for this project. It includes the code used to train and evaluate the model.
2. A link to a folder containing all 798 emojis in the file named 'Dataset'
3. Excel File named 'full_emoji 2 3 3.csv' which contains the captions for all 798 emojis.
4. Weights of the Generator and the Discriminator of the model after training under the files named 'generator_weights_980.h5' and 'discriminator_weights_980.h5' respectively.
5. It also contains the weights of the transfer learned model in the file named 'classification_model_weights.h5' as well as the weights.
6. It comprises of the weights of the Caption Loss model trained with and without Transfer Learning under the file names 'image-caption-weightss0.h5' and 'image-caption-weights1.h5' respectively.
7. Finally, an excel sheet that contains the results of the Quantitative Analysis of all the models and Word Embedding Algorithms has been compiled.

Running the model :

Please Note : This notebook requires High RAM and a GPU. The model was built using Colab Pro. Running all components of the project requires High RAM and a GPU.
Files 1 through 6 must be downloaded and saved on to Google Drive if the project is being run on Colab Pro. If not, download and save the aforementioned files in a location where it can be accessed from.
The emojis are present in the file named 'Dataset'. The link in the file directs the user to a drive folder that holds all 798 emojis.
Paths of all the files in the file 'FinalProject.ipynb' must be changes to reflect the path of folder where the files are stored so that they can be accessed through the code. Besides this, all paths where emojis generated during training as well as non-existent emojis would be saved must be modified based on where the user wishes to save these images.

Once these changes are made, run the file titled 'FinalProject.ipynb'. The model should now be able to create emojis based on the text input. The emojis generated by the model can be viewed in the section titled 'Creating Emoji from Text' or by scrolling further below where the UI was built using the Gradio app. The code also allows the user to change Word Embedding Algorithms by running the code in the cells titled 'GloVe', 'Word2vec' or 'FastText'. Finally, the model can be evaluated by running the code that outputs the metrics used to quantitatively analyse the performance of the model as mentioned in this paper. The Word2vec and FastText models can be directly implemented on the 'FinalProject.ipynb' file. However, the GloVe model (glove.6B.300d) must be downloaded from https://nlp.stanford.edu/projects/glove/. It could not be added to GitLab due to its large size.

Appendix B : Comparison of generated emojis with raw emojis

This section shows all the 114 emojis generated by the model in comparison with all the raw emojis from the dataset used during training.
The emojis above represent the raw emojis used for training and the emojis below are the emojis generated by the author's proposed model.

---------- RAW EMOJIS FROM THE REAL DATASET----------

---------- EMOJIS GENERATED BY THE MODEL ----------