EXPLORING GENERATIVE ADVERSARIAL NETWORKS USING TENSORFLOW

A PROJECT REPORT

Submitted by

AKASH GUPTA - 30316010002
GAGANDEEP SINGH - 30316010018
NAVONIL MUKHERJEE - 30316010024

of

MASTER OF COMPUTER APPLICATION

COMPUTER APPLICATION CENTRE, HERITAGE INSTITUTE OF TECHNOLOGY, KOLKATA

MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY, WEST BENGAL

DECEMBER, 2018

EXPLORING GENERATIVE ADVERSARIAL NETWORKS WITH TENSORFLOW

A PROJECT REPORT

Submitted by

AKASH GUPTA - 30316010002
GAGANDEEP SINGH - 30316010018
NAVONIL MUKHERJEE - 30316010024

of

MASTER OF COMPUTER APPLICATION

HERITAGE INSTITUTE OF TECHNOLOGY

MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY

DECEMBER, 2018

COMPUTER APPLICATION CENTRE

HERITAGE INSTITUTE OF TECHNOLOGY, KOLKATA

BONAFIDE CERTIFICATE

Certified that this project report **"EXPLORING GENERATIVE ADVERSARIAL NETWORKS WITH TENSORFLOW"** is the bonafide work of "**AKASH GUPTA, GAGANDEEP SINGH, NAVONIL MUKHERJEE"** who carried out the project work under my supervision.

SIGNATURE

Director,

Computer Application Centre,

Heritage Institute of Technology

SIGNATURE

SUPERVISOR

Computer Application Centre,
Heritage Institute of Technology

EXAMINER

TABLE OF CONTENTS

Acknowledgements

We would like to thank our advisor Professor Sumon Ghosh for his ongoing support in this project, Professor Dr. Jyotirmoy Ghosh, Professor Dr. Dinabandhu Bhandari and Professor Anirban Kundu for their valuable input during the project and also during the review sessions.

SIGNATURE

AKASH GUPTA - 30316010002

SIGNATURE

GAGANDEEP SINGH - 30316010018

SIGNATURE

NAVONIL MUKHERJEE - 30316010024

ABSTRACT

The field of artificial intelligence and machine learning is advancing at an alarming rate and shows great promise in various applications. Neural networks are being used in diverse areas such as medical imaging, self-driving automobiles, sentiment analysis from textual data etc. Neural networks have been traditionally used for the purposes of classifying data into pre-defined labels. Generative models on the other hand aim to generate new samples from a distribution (the population) given a set of test samples from the population. In our project we aim to explore and understand one such model; the **Generative Adversarial Network** or **GAN**. This novel model consists of two separate units; the **Discriminator** and the **Generator**. The discriminator is trained to recognize samples from a dataset. The generator is tasked with taking random noise as input and creating fake samples that are given to the discriminator. The discriminator then classifies this generated sample as being either real or fake. The generator takes this feedback and aims to ultimately fool the discriminator by creating samples that are indistinguishable from the real data set. In our project we hope to dive deep into the inner working of GANs and understand their qualities as well as short comings using the well-known **TensorFlow** framework.

LIST OF FIGURES

LIST OF SYMBOLS ABBREVIATIONS AND NOMENCLATURE

ABBREVIATIONS

1.	**GPU**	**-**	**Graphics Processing Unit**
2.	**ANN**	**-**	**Artificial Neural Network**
3.	**CNN**	**-**	**Convolutional Neural Network**
4.	**GAN**	**-**	**Generative Adversarial Network**
5.	**MLE**	**-**	**Maximum Likelihood Estimation**
6.	**MLP**	**-**	**Multi-Layer Perceptron**
7.	**ReLU**	**-**	**Rectified Linear Unit**
8.	**SGD**	**-**	**Stochastic Gradient Descent**
9.	**CUDA**	**-**	**Compute Unified Device Architecture**
10.	**cuDNN**	**-**	**CUDA Deep Neural Network**

SYMBOLS

1.	**D**	**-**	**Discriminator Function**
2.	**G**	**-**	**Generator Function**
3.	$\mathbf{X_{train}}$	**-**	**Training dataset**
4.	$\mathbf{X_{data}}$	**-**	**Real dataset or test dataset**
5.	**z**	**-**	**Activation value from each neuron in a neural network**
6.	**Z**	**-**	**Latent Space**
7.	**G(Z)**	**-**	**Generator Output**
8.	**X**	**-**	**Input to the discriminator, either from X_{train}, X_{data} or G(Z)**

LIST OF EQUATIONS AND FUNCTIONS

1 INTRODUCTION

Machine learning is a diverse field of applied computer science and is quickly gaining momentum nowadays. Neural networks are a prime example of machine learning and can approximate a wide variety of functions. They are often used for classification problems. Although neural networks have existed for a while now, they have recently attracted a lot of attention due to the significant increases in computation power that is a natural result of developments in the chip manufacturing processes (**Moore's Law**). With the advent of powerful and efficient graphics processing units or GPUs, the running time for training neural networks has been reduced by several orders of magnitude. Neural networks are units of computation that are loosely based on the neural structures of the brains seen in the organisms in the natural world. Just as we learn from our experiences by trial and error, so too do these artificial neural networks (ANN) by consuming data known as training datasets. The learning process can be broadly classified into two types – **supervised** and **unsupervised** learning. In supervised learning, the network is trained using labeled datasets. For example, let us consider a network that classifies images of dogs and cats. These types of image classification problems are best handled by a class of neural networks called **Convolutional Neural Networks** (CNN). The CNN is trained using lots of images of dogs and cats. During the training process, the network is fed images from the training dataset and is asked to classify them. Every time it incorrectly classifies a dog as a cat or vice versa, it "**learns**" from the label provided with the image. The network is composed of nodes and weights. It is the value of these weights that play the main role in the learning process.

But what if we train a network using lots of images of cats and then ask it to give us new images of cats? This cannot be done using regular CNNs and require some kind of generative model. Generative models try to create samples that fit a known distribution. GAN is one such model and is the topic of our project.

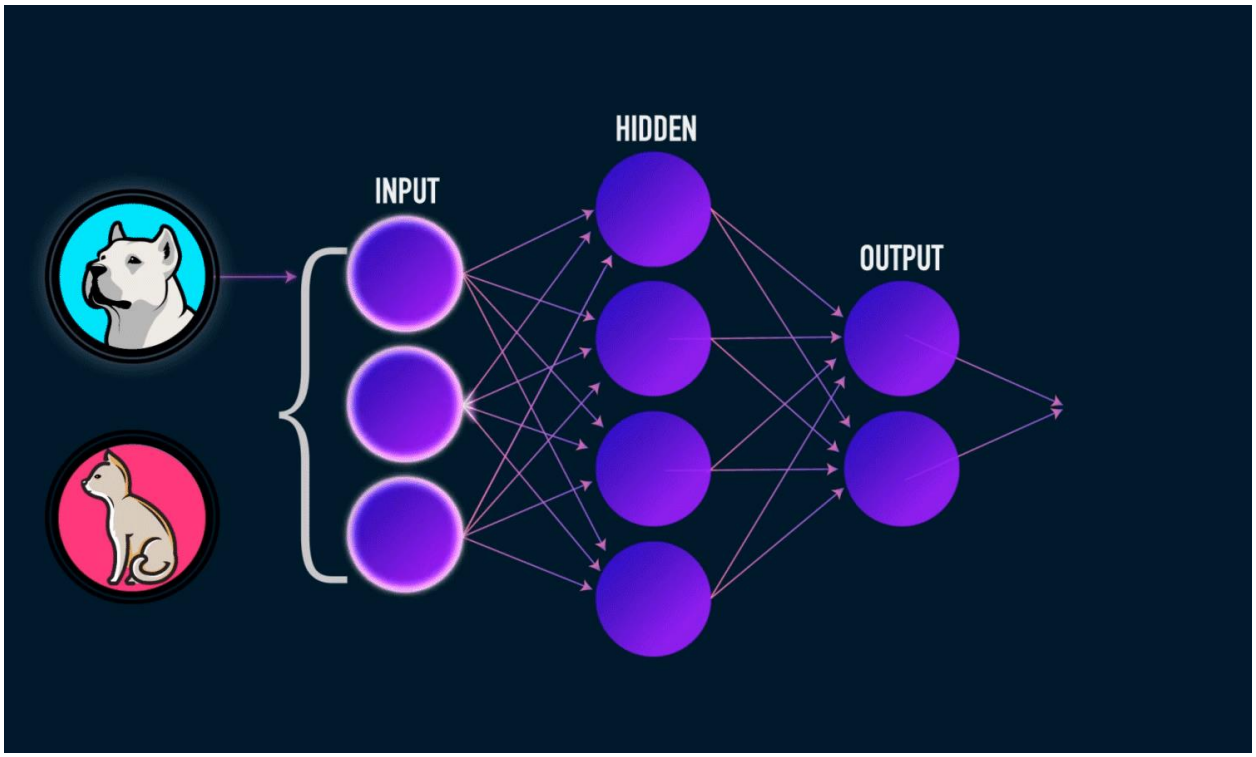

Fig 1.1 A Neural Network classifier trained using images of cats and dogs

In figure 1.1, we see a high-level diagram of a neural network that classifies cat and dog images. The edges between the nodes are the weights which are responsible for the actual learning process.

GANs are a novel solution which can generate some very convincing samples that are very close to the ground truth seen in the training data set. Generative Adversarial Networks were invented by Ian Goodfellow et al. in 2014. We will now take a closer look at GANs and their components.

1.1 GENERAL

A Generative Adversarial Network has two units; a **discriminator** D and a **generator** G. Although D and G can be implemented using any model of computation that satisfies GAN's properties, usually they are created using neural networks. Generative Adversarial Networks are so called because D and G are pitted against each other in a zero-sum game where the generator tries to create samples from random noise that are as close to the samples from the real dataset as possible. The discriminator on the other hand tries to differentiate these generated images from the real ones. It is much like a counterfeiter and a cop where one tries to outsmart the other in a game of cat and mouse.

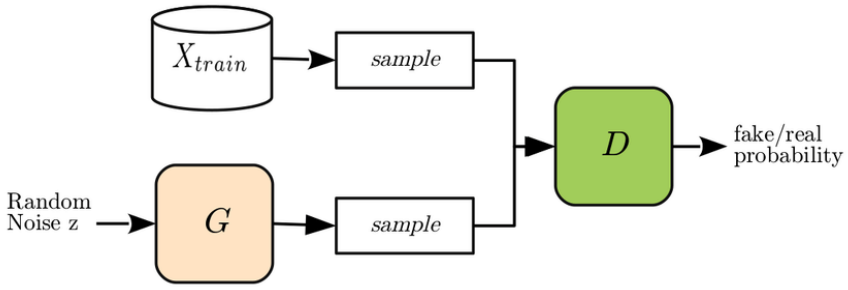

Fig 1.2 Basic architecture of a Generative Adversarial Network

The discriminator is initially only "partially" trained and the generator initially sends random noise as the input to the discriminator. The discriminator is usually fed one image from the real set followed by one made by G in alternating fashion although different training strategies are possible. With each iteration the discriminator gets better at spotting the fake images whereas the generator gets better at creating fake images that look real. This goes on until the discriminator is no longer able to reliably differentiate between the two. One thing to note is that even though we are talking about images here, it is completely possible for GANs to work with other types of data like text, music etc.

1.2 TAXONOMY OF GENERATIVE MODELS

Internally, GANs use **Maximum Likelihood Estimation** (MLE) whereby the network tries to estimate a set of parameters from the sample size such that the chance that these parameters come from the real data set is maximized.

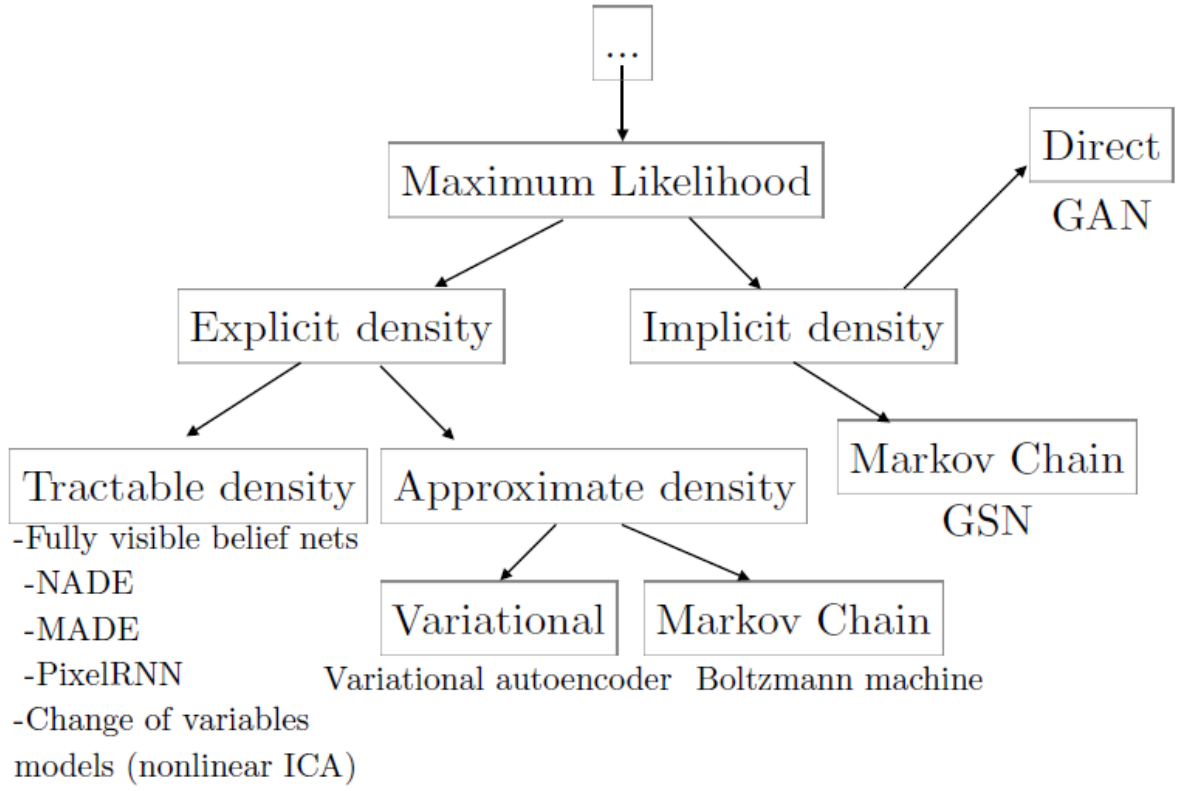

Fig 1.3 Taxonomy of Generative Models

As we can see from Figure 1.3, MLE models can be broadly classified into two types. Models which have explicit analytical functions for specifying a distribution and those which implicitly learn the distribution from the training data set. GANs follow such an unsupervised learning process.

2 COMPONENTS OF A GAN

We will now be looking at the components of a generative adversarial network. The primary pieces of a GAN are the discriminator and generator networks and the loss function which we choose to optimize.

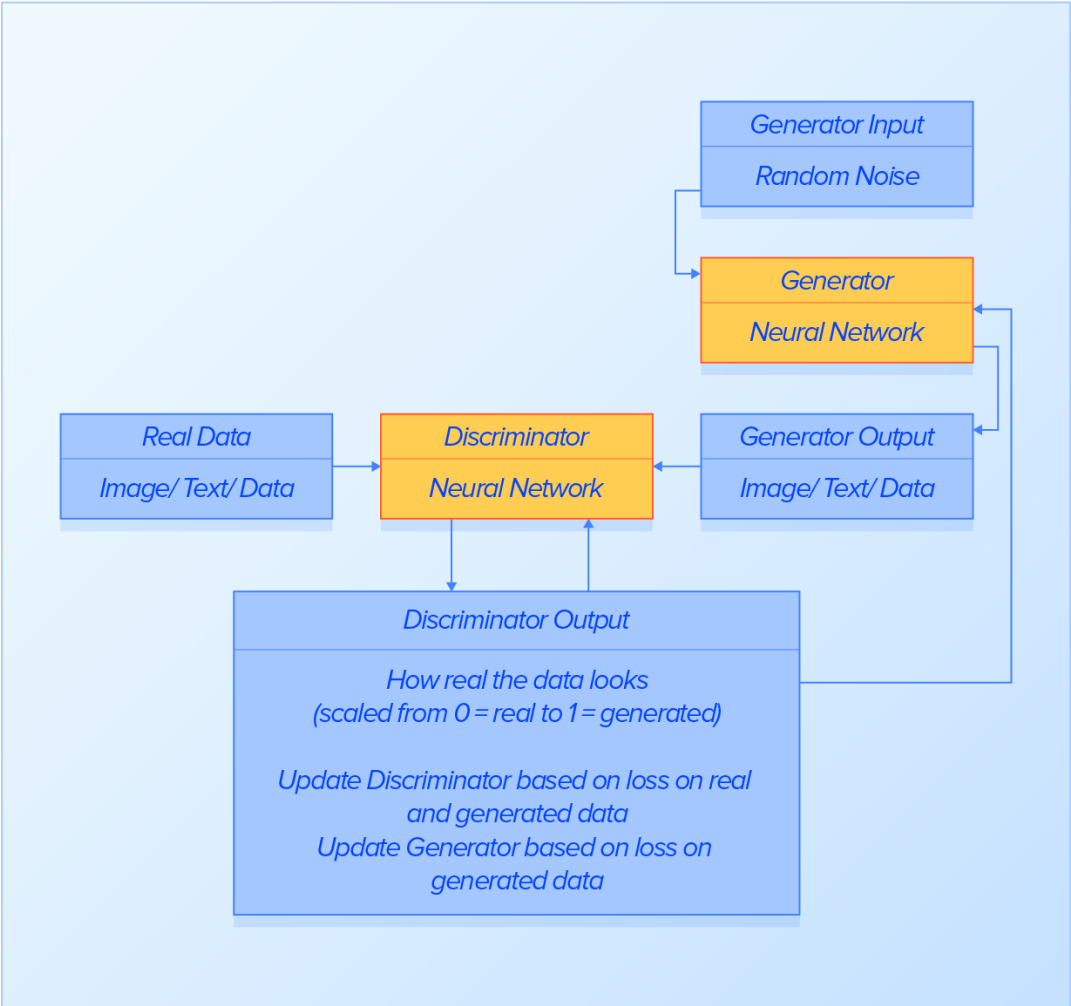

Fig 2.1 Flow diagram of a GAN

Figure 2.1 gives us a view of the flow of information in the network. As we can see, the error function of the discriminator is fed into the generator. Since it is a zero-sum game, the loss function that D aims to reduce, G aims to maximize.

2.1 DISCRIMINATOR

The discriminator D is a classifier that is partially trained using supervised learning. It classifies whether an image is real (1) or fake (0). So, the discriminator is a standard convolutional neural network that learns from the training and generated dataset and makes a binary choice about their authenticity.

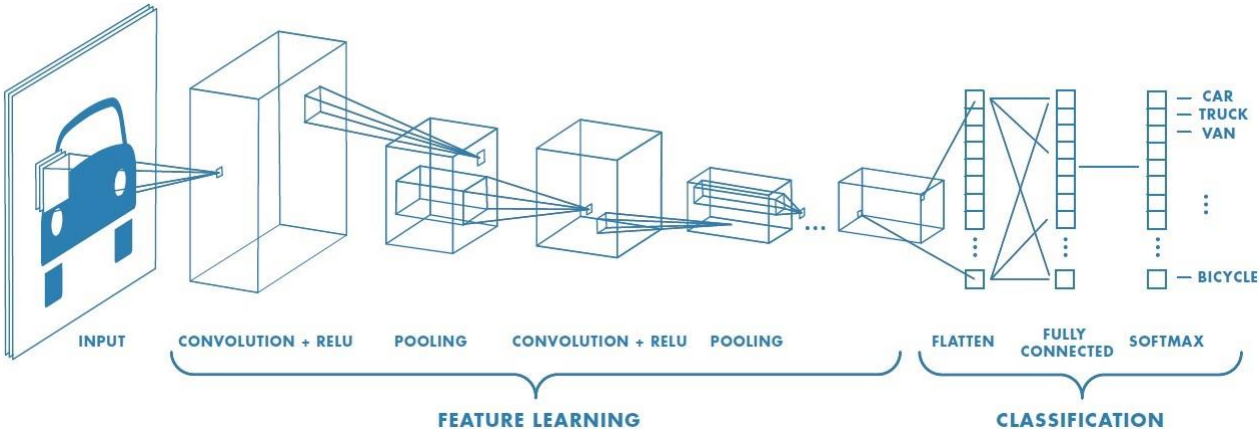

Fig 2.2 Structure of a CNN

As we can see from the figure above, a CNN first extracts the properties of an image that it is tasked to recognize and then uses this knowledge for further classification purposes. Let us now take a closer look at the layers in a ConvNet.

2.1.1 THE CONVOLUTION LAYER

As the name suggests, the convolution operation is at the heart of the CNN. These networks have some unique features which make them ideal candidates for pattern recognition tasks. Unlike fully connected multi-layer perceptrons (MLP), CNNs don't match every pixel of the pattern to every pixel of the candidate image. Instead, they match pieces of the images. Therefore, CNNs are rotation invariant; which means they can detect a pattern even if the

candidate image has been rotated. During the convolution operation, a small square of k x k pixels is slid across the image of n x n pixels where k < n and the resulting pixel values are averaged. This square set of pixels is called the convolution filter and k is the kernel size. The number of pixels the filter is slid after each iteration is called the **stride**. The result is a **feature map**. So, there can be multiple convolution layers with different filters for different features that need to be extracted.

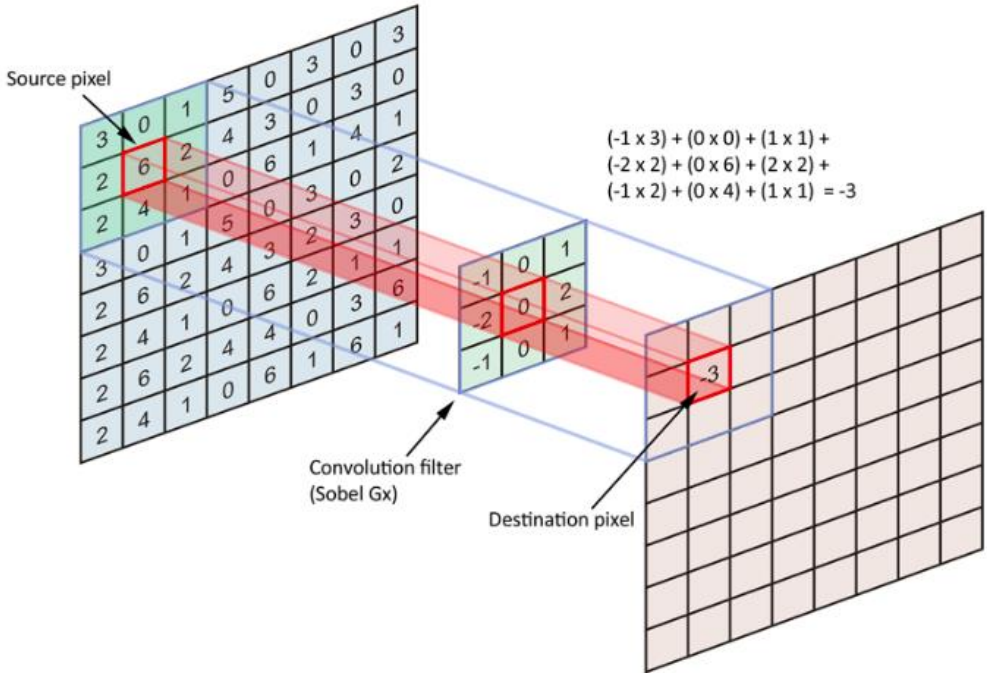

Fig 2.3 The convolution operation

2.1.2 THE ReLU LAYER

After the convolution layer we have the **rectified linear unit** or ReLU layer. This layer provides removes all pixels that have a non-negative value as a result of the convolution operation.

$$f(x) = x^{+} = \max(0, x)$$

THE ReLU FUNCTION

Sometimes, instead of regular ReLU, a slightly modified version of the function called a Leaky ReLU is also used. Leaky ReLU allows a small positive gradient when a unit is not active so that the learning process during backpropagation doesn't stall.

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$$

THE LEAKY ReLU FUNCTION

2.1.3 THE POOLING LAYER

After the relu layer, we have the pooling layer, although the order of relu and pooling layer can be switched. If instead of a CNN, we were to use a fully connected MLP, where we represented each pixel of an image of size n x n by a node in the neural network, the number of nodes would be n^2 and the number of connections between then would be even higher. Training such a network could take prohibitively long. For this reason, CNNs perform dimensionality reduction, where the features are extracted and reduced in size for efficiency. This reduction in size takes place in the pooling layer. One pooling operation is **max-pooling** where the maximum valued pixel in a k x k map is chosen to represent the map.

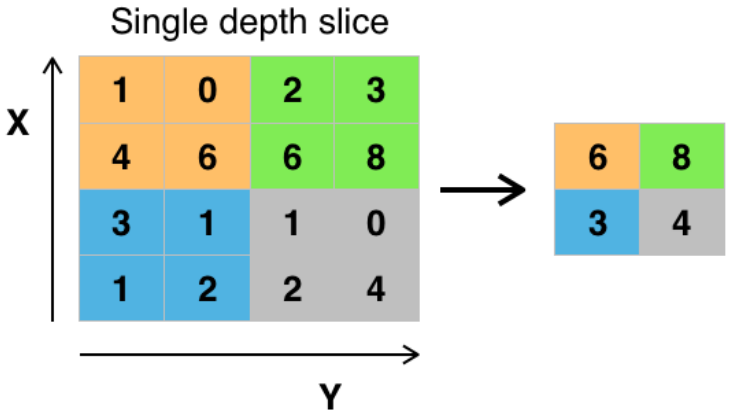

Fig 2.4 Max pooling with a 2x2 filter and stride = 2

2.1.4 THE SOFTMAX LAYER

After the fully connected layers where the features are matched, we have the penultimate layer called the **Softmax** layer. The classes for a CNN can be overlapping in which case the output values won't add up to 1. The softmax function fixes this problem for us. The output values will all lie in the interval [0,1] and their sum will be 1.

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

THE SOFTMAX FUNCTION

Here **z** is the output from each neuron that acts as an input to the softmax layer. There are k such neurons.

2.2 GENERATOR

As we have already mentioned, the generator G is a function that creates new samples from random noise. The set from which the random noise is taken is called the **latent space**. So the generator creates a mapping from this latent space **Z** to **X** where X is the input space to the discriminator D. If G does a good job, then D can be fooled into thinking that G(z) is a sample from X. Just like the discriminator is basically a regular CNN, the generator is actually an inverse of that, sometimes called a **deconvolutional network**. A CNN takes in an image and gives us a vector that is used for classification. The generator takes an input random noise and returns an image based on the parameters it learns from.

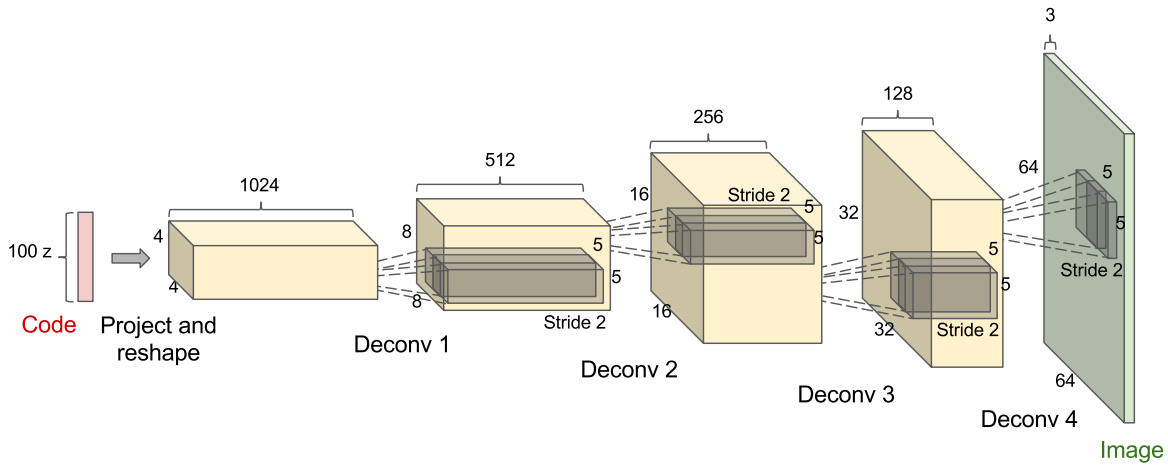

Fig 2.5 The deconvolutional network in the generator

2.3 THE LOSS FUNCTION

The loss function for the discriminator is the standard **cross-entropy** function. The discriminator D aims to maximize this function. Since G and D are engaged in a minimax game, G aims to minimize the same loss function that D has to maximize i.e. G wants to minimize the times that D is correct meaning the generated samples G(z) do not represent X_{data}. We train D to maximize the probability of assigning correct labels to samples from X_{data} and those from G(z). Similarly, we train G alternately to minimize the probability that D correctly labels samples from G(z). So, D and G are engaged in a zero-sum game with the value function V (D, G) given by: -

$$\min_G \max_D V(D,G)$$

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

THE CROSS-ENTROPY LOSS FUNCTION

The cross-entropy function considers the degree of misclassification on the part of D. This means

Finally, we use stochastic gradient descent (SGD) to ascend the gradient in case of D and descend the gradient in case of G.

2.4 THE TRAINING

<u>The algorithm for training the GAN is as follows</u>: -

for number of training iterations do

for k steps do

Sample minibatch of m noise samples { Z^1, Z^2, … Z^m } from Z

Sample minibatch of m examples { X^1, X^2, … X^m } from X_{data}

Update the discriminator D by ascending its stochastic gradient

end for

Sample minibatch of m noise samples { Z^1, Z^2, … Z^m } from Z

Update the generator G by ascending its stochastic gradient

end for

The training continues until D and G reach Nash equilibrium when the output from D converges to a probability value of 0.5, i.e. when the discriminator cannot reliably differentiate between the real and the fake images anymore. At this point, we have a generator that can be used to generate new samples that are representative of the training set.

2.5 IMPLEMENTATION USING TENSORFLOW

We used the immensely popular TensorFlow framework from Google to implement a GAN that is trained on the MNIST dataset of handwritten digits from 0 through 9. TensorFlow is written in C/C++ and is highly optimized for parallel computation. Using NVIDIA CUDA, we were able to gain execution speeds that were orders of magnitude faster compared to the CPU only build of TensorFlow. NVIDIA CUDA is a GPGPU platform for using graphics processing units for general purpose computation whereas cuDNN is a library that allows fast math operations needed for deep learning to run on GPUs.

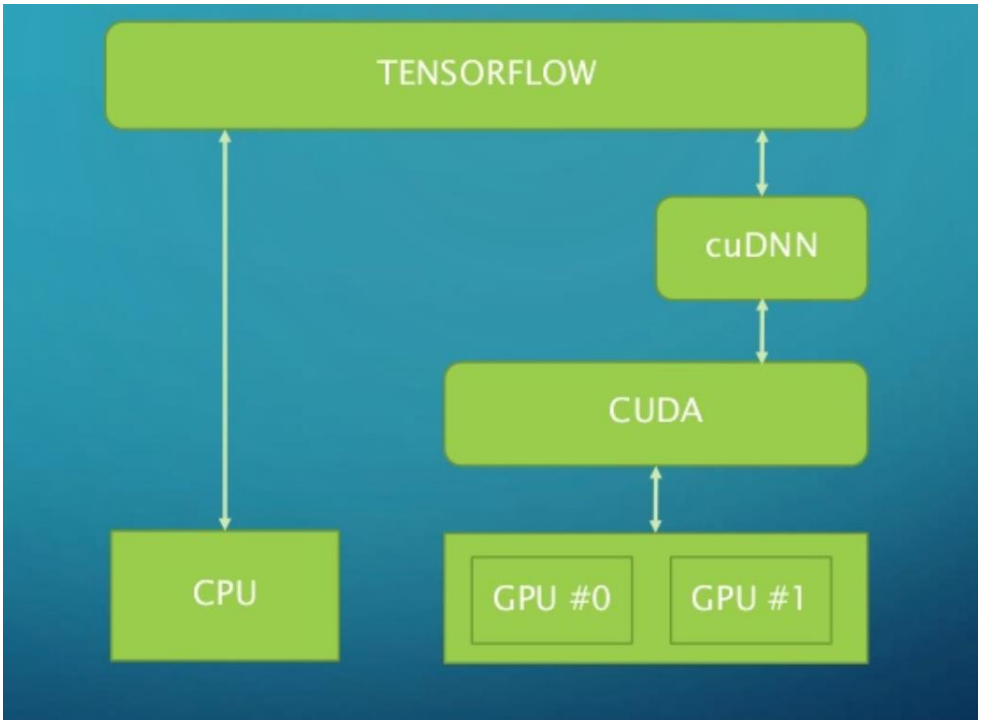

Fig 2.6 The platform architecture of TensorFlow

2.5.1 RESULTS

All of our tests were run on a 6-core workstation with Intel Xeon x5670 CPU along with 24 GB of main memory and an NVIDIA GTX 1080 graphics card. We used Python 3.5 with TensorFlow version 1.5 running on 64-bit Windows 10 Professional with GPU driver version 399.24. TensorFlow was running under Anaconda in its own virtual environment.

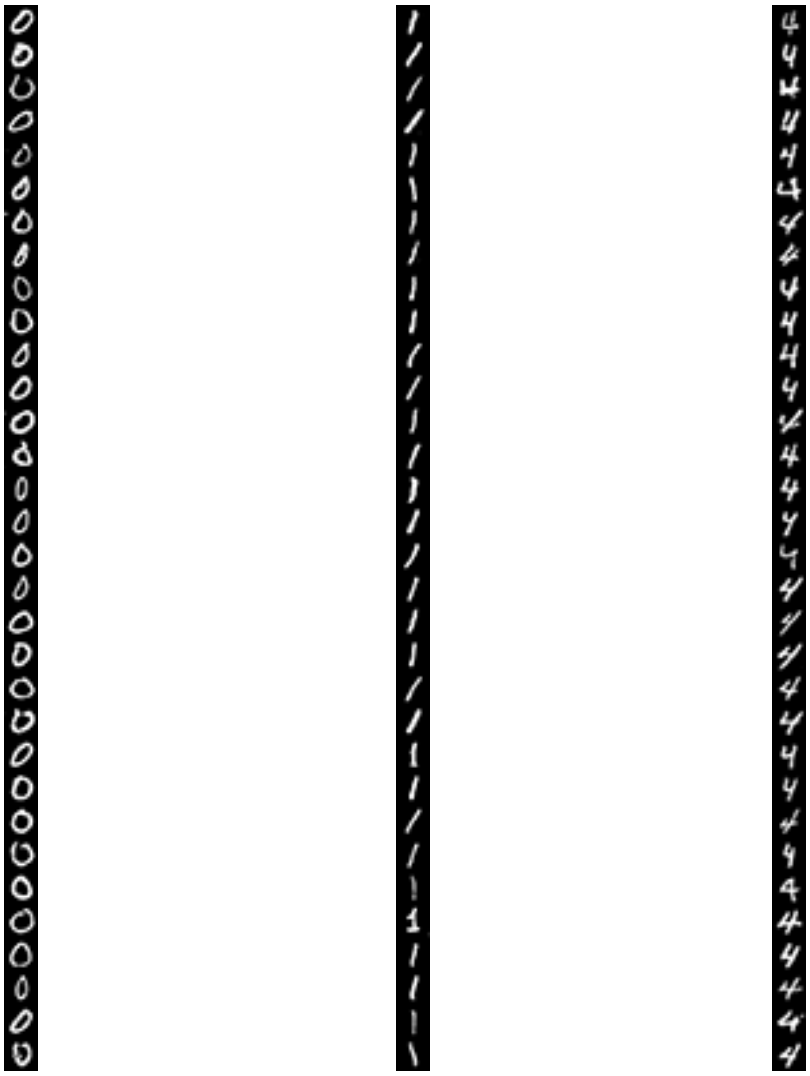

Fig 2.7 Generated outputs of the digits 0, 1 and 4 based on the MNIST dataset each after 20000 iterations

3 CONCLUSION AND FUTURE WORK

Generative Adversarial Networks have quickly become an exciting sub-field of study finding applications in diverse fields. Their proven ability to mimic seemingly complex and arbitrary distributions make them an ideal candidate for creating samples where very large-scale data acquisition is difficult or infeasible. Companies like Google, Facebook, NVIDIA are all actively conducting research using GANs and they find applications in commercial software as

well. NVIDIA recently proposed a novel method for using style-transfer in their GAN for generating human faces and achieved startlingly realistic results. Ongoing research focuses on solving problems like **mode collapse** where generated samples stop being diverse because of the generator failing to learn to produce new features based on the training data.

Fig 3.1 NVIDIA's results with their "style transfer" GAN

We can see that it is a very exciting time to be working on generative adversarial networks and research on this topic looks to be useful and promising.

References

[1] Cody Nash - https://www.toptal.com/machine-learning/generative-adversarial-networks - "Create Data from Random Noise with Generative Adversarial Networks"

[2] Ian Goodfellow et al. (2014) - Generative Adversarial Nets

[3] Ian Goodfellow, OpenAI - NIPS 2016 Tutorial - Generative Adversarial Networks

[4] Ian Goodfellow, Yoshua Bengio and Aaron Courville – Deep Learning – MIT Press 2016

[5] Jamie Hayes et al. (2014) "LOGAN: Evaluating Privacy Leakage of Generative Models Using Generative Adversarial Networks"

[6] Michael Nielsen - http://neuralnetworksanddeeplearning.com/

[7] Robert DiPietro - https://rdipietro.github.io/friendly-intro-to-cross-entropy-loss/ - "A Friendly Introduction to Cross-Entropy Loss"

[8] Tariq Rashid – "Make Your Own Neural Network"

[9] William Feller – "An Introduction to probability theory and its applications" – Wiley 2014

[10] Thomas M. Cover – "Elements of Information Theory" – Wiley 2017

[11] https://www.mathworks.com/discovery/convolutional-neural-network.html

[12] https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2