ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

GENERATIVE ADVERSARIAL NETWORKS IN COMPUTER VISION APPLICATIONS

M.Sc. THESIS

Semih ÖRNEK

Department of Electronics and Communication Engineering

Telecommunication Engineering Programme

JANUARY 2021

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

GENERATIVE ADVERSARIAL NETWORKS IN COMPUTER VISION APPLICATIONS

M.Sc. THESIS

Semih ÖRNEK
(504181332)

Department of Electronics and Communication Engineering

Telecommunication Engineering Programme

Thesis Advisor: Prof. Dr. Ender Mete EKŞİOĞLU

JANUARY 2021

ISTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYARLI GÖRÜ UYGULAMALARINDA ÇEKİŞMELİ ÜRETİCİ AĞLAR

YÜKSEK LİSANS TEZİ

Semih ÖRNEK
(504181332)

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Telekomünikasyon Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Ender Mete EKŞİOĞLU

OCAK 2021

Semih Örnek, a M.Sc. student of ITU Graduate School of Science Engineering and Technology 504181332, successfully defended the thesis entitled "GENERATIVE ADVERSARIAL NETWORKS IN COMPUTER VISION APPLICATIONS", which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Prof. Dr. Ender Mete EKŞİOĞLU** İstanbul Technical University	....
Jury Members :	**Prof. Dr. Ahmet Hamdi KAYRAN** Istanbul Technical University	....
	Prof. Dr. Aydın KIZILKAYA Pamukkale University	....

Date of Submission : 30 December 2020
Date of Defense : 28 January 2021

To my family and friends,

FOREWORD

First of all, I would like to thank my advisor, Prof. Dr. Ender Mete Ekşioğlu, for giving me the opportunity to work on such a promising subject and providing me the necessary hardware for my project.

I would like to thank Arçelik for giving me the time and respect to finish my Master of Science degree.

Finally, I am very grateful to my family and friends for believing in me and helping me through this long and difficult period while continuing my Master of Science degree and working simultaneously at Arçelik.

December 2020 Semih ÖRNEK
(Software Engineer)

TABLE OF CONTENTS

ABBREVIATIONS

AI	: Artificial Intelligence
GAN	: Generative Adversarial Network
CNN	: Convolutional Neural Network
RNN	: Recurrent Neural Network
DL	: Deep Learning
OOP	: Object-Oriented Programming
GPU	: Graphical Processing Units
CPU	: Central Processing Unit
TPU	: Tensor Processing Unit
PIL	: Python Imaging Library
SRGAN	: Super Resolution Generative Adversarial Network
ESRGAN	: Enhanced Super Resolution Generative Adversarial Network
RRDB	: Residual-in-Residual Dense Block
DCGAN	: Deep Convolutional Generative Adversarial Networks
CoGAN	: Coupled Generative Adversarial Networks
SAGAN	: Self-Attention Generative Adversarial Networks
NLM	: Non-local Means
PSNR	: Peak Signal-to-Noise Ratio
MSE	: Mean Square Error
NLP	: Natural Language Processing
ITU	: Istanbul Technical University

SYMBOLS

Img_{src}	**:** Training dataset image
$kernel_{bic}$	**:** Bicubic kernel
Img_{ds}	**:** Downsampled image
Img_{hr}	**:** High-resolution image
G_{loss}	**:** Generator Loss
D_{loss}	**:** Discriminator Loss
Img_{org}	**:** Original Image
Img_{deg}	**:** Degraded Image
rows	**:** Number of rows in image matrix
cols	**:** Number of columns in image matrix

LIST OF TABLES

LIST OF FIGURES

Page

GENERATIVE ADVERSARIAL NETWORKS IN COMPUTER VISION APPLICATIONS

SUMMARY

Generative Adversarial Networks (GANs) are one of the examples of generative modelling which uses deep learning (DL) based methods. It is considered that GANs are the best way to train a generative model. GANs consist of two parts. The first one is the generator and the second one is the discriminator. Generator’s mission is to create fake data that is indistinguishable from the real data for the discriminator. Discriminator’s mission is to distinguish the real data from the fake data that has been generated by the generator.

Generative adversarial networks have two different neural network architectures to train. GANs should run discriminator training and generator training together. Because the generator training and the discriminator training heavily rely on each other, they are trained alternatingly in one iteration. Generative models are a branch of unsupervised machine learning, but the training of the GAN architecture, which relies on generative modelling, is considered as supervised machine learning.

This thesis demonstrates that generative adversarial network architectures can effectively tackle important computer vision problems. These problems include the generation of fake images, super-resolving images and denoising of noisy images.

In this thesis, we studied three different computer vision applications which use generative adversarial networks for solution. The results indicate that GANs can be very effectively used for these particular problems. Before GAN there were other architectures which used generative modelling as well, but with the founding of GAN those architectures that used generative modelling to solve the computer vision problems became pretty much insufficient. Also, some of the image processing techniques that were used to solve computer vision problems also fell out of use.

There were not many strategies to super-resolve images by using GANs until the last few years. Deep learning started to come handy for solving this task and the research for this problem started to grow. The super-resolution models are created, and different learning methods applied to these models to solve this task, but most of them failed on real world images which are taken by devices such as smartphones. The most widespread method for training the super-resolution deep learning models starts by downscaling the images that are inside the dataset with methods like nearest neighbor resampling, bicubic resampling and bilinear resampling. This process is applied in order to make a dataset that contains high-resolution and low-resolution training image pairs. The low-resolution images that are created by this process have almost no noise, in other words the images are clean. The main purpose of these super-resolution deep learning models is to increase the resolution of the images.

Enhanced Super Resolution Generative Adversarial Network (ESRGAN) has been used as a GAN architecture for this problem. ESRGAN is an improved version of the Super Resolution Generative Adversarial Network (SRGAN) architecture. As the

name SRGAN suggests, it uses a deep neural architecture with an adversarial network. Its main purpose is to super resolve the images to produce higher resolution images compared to the input images.

Fake image generation subject was started by Ian Goodfellow back in 2014. He and his colleagues founded the Generative Adversarial Network theory. Images have been generated from the datasets by using GAN architecture. These generated images look as if they are from the training dataset, but they are unique on their own because there are no such generated images in the training dataset. For this problem two different GAN architecture have been used. In 2015, Deep Convolutional Generative Adversarial Networks (DCGAN) were developed. It is a better version of the simple GAN. One more important thing that had come with DCGAN is the fact that traversing through the latent space of the generated image and changing the values in latent space dimensions can change the generated image drastically. For example, with using vector arithmetic in the latent space of the generated images, a new generated image can be produced. In 2018, BigGAN model was proposed. ResNet GAN architecture has been used for the BigGAN model. BigGAN benefitted from scaling and it provided bigger generative adversarial networks and larger batch sizes. Neural networks have been trained with two or four times more parameters and eight times more batch size then the previous implementations. As a result, the generated images looked indistinguishable from the real input images from the dataset that have been used to train the GAN.

Deep learning methods have been used to tackle image processing problems for quite some time. Denoising is one of the most known image processing problems to this date. There are different methods to attack this problem. The most traditional ones are the image processing methods. Denoising the images with the linear filters, non-linear filters, adaptive filters can be given as an example for the traditional image processing techniques. CNN architectures have been proposed to tackle this problem. In a recent method, a GAN was trained to learn the noise distribution and then CNN was used to denoise the images. In the literature there are no GAN architectures that are solely adapted to image denoising problem. For this problem, using an SRGAN like architecture was proposed. It has been understood that using the SRGAN like architecture not only works for improving the image resolution but to denoise the images as well. Also, it has been shown that DCGAN architecture can be used in more than one image processing problem. It has been shown that this architecture can be used not only for fake image generation, but also for problems such as image denoising, super resolving the images and deblurring the images as well. Nowadays deep learning methods are much more popular than the traditional image processing techniques, because these methods can learn from the datasets, create their own features, and preserve the image details better because of the learning aspect.

Updated form of the DCGAN architecture has been proposed and used for the denoising problem as a GAN architecture. In the generator of the neural network architecture some changes have been made to generate bigger images and also to reach better performance. Image size is kept the same while going forward in the layers of the generator but, the channel size is changed. An extra hidden layer was added to the generator to make the neural network denser and to keep the image size the same. Some changes have been made on the discriminator because of the changes in the generator.

From the practical standpoint, recent GAN architectures with the improved optimization techniques are easy to train, and the results are getting more accurate. The training, validation and testing parts are almost the same. The only differences are the loss functions and the optimizers. The procedure for making the dataset ready for training differs for each problem. Different pre-processing techniques and normalization techniques are used on those datasets. Also, the GAN part looks the same with two distinct networks, one being the generator and the other one being the discriminator. Although these two architectures do the same job every time, where the generator tries to generate fake images and the discriminator tries to distinguish the generated fake images from the real images, these architectures get changed from one problem to another to tackle the particular characteristics of the problem.

BİLGİSAYARLI GÖRÜ UYGULAMALARINDA ÇEKİŞMELİ ÜRETİCİ AĞLAR

ÖZET

Çekişmeli üretici ağlar, derin öğrenme tabanlı yöntemler kullanan üretken modelleme metotlarından biridir. Çekişmeli üretici ağların üretken modelleme eğitme yöntemleri içinde en iyi başarım sağlayan yöntem olduğu düşünülmektedir. Çekişmeli üretici ağlar iki ağdan oluşmaktadır. Birincisi üretici ağ, ikincisi ayrıştırıcı ağdır. Üretici ağın misyonu, ayrıştırıcı ağ için gerçek verilerden ayırt edilemeyen sahte veriler oluşturmaktır. Ayrıştırıcı ağın misyonu, gerçek verileri üretici ağ tarafından üretilen sahte verilerden ayırmaktır.

Çekişmeli üretici ağlarda eğitilmesi gereken iki farklı yapay sinir ağı mimarisi vardır. Çekişmeli üretici ağlar, ayrıştırıcı ağ eğitimini ve üretici ağ eğitimini birlikte yürütmelidir. Üretici ağın eğitimi ve ayrıştırıcı ağın eğitimi birbirlerinden geri bildirimler aldığından, tek bir yinelemede dönüşümlü olarak eğitim sağlanmaktadır. Üretken modeller, denetimsiz makine öğreniminin bir dalıdır. Ancak üretken modellemeye dayanan çekişmeli üretici ağı mimarisinin eğitimi, denetimli makine öğreniminin konusu olarak kabul edilir.

Bu tez, bilgisayarlı görü problemlerinin çözümü için birçok çekişmeli üretici ağ mimarisini incelemektedir. Bu bilgisayarlı görü problemleri sahte görüntülerin üretimi, görüntülerin çözünürlüğünün arttırılması ve görüntülerdeki gürültülerin yok edilmesidir.

Tezde, çekişmeli üretici ağları kullanan üç farklı bilgisayarlı görü uygulaması incelenmiştir. Sonuçlar, çekişmeli üretici ağların bu belirli problemler için çok etkili bir şekilde kullanılabileceğini göstermektedir. Çekişmeli üretici ağlardan önce, üretken modellemeyi kullanan başka mimariler de sunulmuştur. Ancak çekişmeli üretici ağların geliştirilmesiyle, bilgisayarlı görü problemlerini çözmek için üretken modellemeyi kullanan diğer mimariler giderek popülerliğini yitirmiştir ve çekişmeli üretici ağlara göre daha etkisiz kalmışlardır. Ayrıca, bilgisayarlı görü problemlerini çözmek için kullanılan görüntü işleme tekniklerinden bazıları da kullanım dışına itilmiştir.

Son birkaç yıla kadar çekişmeli üretici ağları kullanarak görüntülerin çözünürlüğünü arttırmak için fazla strateji yoktu. Derin öğrenmedeki gelişmelere paralel olarak, bu probleme yönelik araştırmalar giderek gelişmeye başladı. Yeni süper çözünürlük modelleri oluşturuldu ve bu modeller üzerinde farklı öğrenme yöntemleri uygulandı. Ancak bu öncül yöntemler herhangi bir cihazın kamerasından doğrudan alınan görüntülerde başarısız oldu. Süper çözünürlük için derin öğrenme modellerini eğitmenin en yaygın yöntemi, en yakın komşu yeniden örnekleme, çift kübik yeniden örnekleme ve çift doğrusal yeniden örnekleme gibi yöntemlerle veri kümesinin içindeki görüntülerin ölçeğini küçültmekle başlamaktadır. Bu işlem, yüksek çözünürlüklü ve düşük çözünürlüklü eğitim görüntü çiftleri içeren bir veri kümesi

oluşturmayı sağlar. Bu işlemden sonra elde edilen düşük çözünürlüklü görüntülerde gürültü büyük miktarda azalır yani görüntüler aynı zamanda temizlenmiş olur.

Süper çözünürlüklü derin öğrenme modellerinin temel amacı görüntülerin çözünürlüğünü artırmaktır. Bu probleme yönelik olarak tez kapsamında çekişmeli üretici ağı mimarilerinden olan ESRGAN gerçeklenmiştir. ESRGAN, SRGAN mimarisinin geliştirilmiş bir versiyonudur. Ana amacı, giriş görüntülerine kıyasla daha yüksek çözünürlüklü görüntüler üretmek için görüntülerin çözünürlüğünü arttırmaktır.

Çekişmeli üretici ağlarla sahte görüntü üretme konusu Ian Goodfellow tarafından 2014 yılında ortaya atıldı. Bu kapsamda çekişmeli üretici ağ eğitim veri kümeleriyle eğiterek, verisetlerinde olmayan yeni görüntülerin üretilmesi sağlandı. Bu üretilmiş görüntüler, eğitim veri kümesinden alınmış gibi görünmesine rağmen aslında benzersizdirler ve tamamen sıfırdan üretilmişlerdir.

Tez kapsamında bu problem için iki farklı çekişmeli üretici ağı mimarisi gerçeklenmiştir. Bu mimarilerden bir tanesi olan DCGAN, basit çekişmeli üretici ağı yapısının geliştirilmiş bir versiyonudur. Tez kapsamında sahte görüntü üretme için gerçeklenen ikinci GAN tabanlı yöntem ise yöntem BigGAN üretici ağını içermektedir. BigGAN ağ mimarisi ResNet ağ yapısına dayanmaktadır. BigGAN mimarisi görece olarak daha büyük çekişmeli üretici ağların eğitimini mümkün kılmaktadır. BigGAN mimarisinin kullanımı ağların önceki mimarilere göre dört kata kadar daha fazla parametre ve sekiz kata kadar daha fazla yığın boyutu ile eğitilebilmesini mümkün kılmıştır. Sonuç olarak üretilen görüntüler, çekişmeli üretici ağı eğitmek için kullanılan veri kümesindeki gerçek giriş görüntülerinden neredeyse ayırt edilemez görünmektedir.

Gürültü giderme, en popüler görüntü işleme problemlerinden bir tanesidir. Bu problemi çözmek için çok farklı yöntemler sunulmuştur. Görüntülerde gürültü gidermeye yönelik olarak sunulmuş çok sayıda geleneksel görüntü işleme yöntemi literatürde yer almaktadır. Bunlar arasında doğrusal filtreler, doğrusal olmayan filtreler, uyarlanabilir filtreler ve yerel olmayan yöntemler örnek olarak verilebilir.

Yakın zamanda literatürde sunulan bir araştırmada, gürültü dağılımını öğrenmek için bir çekişmeli üretici ağın kullanımı önerilmiştir. Bu adımın ardından görüntüleri gürültüden arındırmak için klasik evrişimli sinir ağı kullanılmıştır. Literatürde, görüntülerde gürültü giderme problemine özel olarak uyarlanmış bir çekişmeli üretici ağı mimarisi yer almamaktadır. Literatürde bu problem için, SRGAN benzeri bir mimarinin kullanılması önerilmiştir. Sunulan benzetim sonuçları SRGAN mimarisinin sadece görüntü çözünürlüğünü iyileştirmek için değil, aynı zamanda görüntülerdeki gürültüyü giderme problemine yönelik olarakta iyi sonuçlar verdiğini göstermiştir. Tez kapsamında ise DCGAN mimarisinin görüntülerde gürültü giderme için kullanımı incelenmiştir. DCGAN mimarisi sahte görüntü oluşturma, görüntülerin çözünürlüğünün arttırılması ve görüntülerin bulanıklığının giderilmesi gibi uygulamalarda kullanılmıştır. Tez kapsamında yapılan çalışma ile bu yapının görüntülerde gürültü giderme problemine yönelik olarak da kullanılabileceği anlaşılmıştır.

Günümüzde derin öğrenme yöntemleri geleneksel görüntü işleme tekniklerinden daha popüler hale gelmişlerdir. Derin ağlar çok büyük veri setleri kullanarak eğitimi ve analitik yöntemlerle ulaşılamayan özniteliklerin çıkarılmasını sağlamaktadır.

Tez kapsamında üç farklı bilgisayarlı görü problemine yönelik olarak çekişmeli üretici ağ derin öğrenme mimarilerinin kullanımı incelenmiştir. Literatüre yeni kazandırılmış

olan çekişmeli üretici ağ mimarilerinin eğitilmesi göreceli olarak kolaydır. Çekişmeli üretici ağlar için eğitim, doğrulama ve test kısımları çok benzerdir. Bu kısımların aralarındaki tek fark kullanılan kayıp fonksiyonları ve optimizasyon yöntemleridir. Tez kapsamında yapılan gerçeklemeler, görüntü çözünürlük yükseltme, sahte görüntü üretme ve görüntülerde gürültü giderme uygulamaları için çekişmeli üretici ağların başarıyla kullanılabileceğini göstermiştir.

1. INTRODUCTION

In this chapter, the theoretical background of the thesis is explained from the general subject to the specific subject, the aim of the thesis and the literature review are briefly introduced.

1.1 Theoretical Background

1.1.1 Machine learning

Machine learning brings together statistics and computer science to enable computers to learn how to do a given task without being programmed to do so.

What is really special about algorithms of machine learning is that they rely on the data, not executing some code blocks for given conditions. And of course, the more data the better the results.

The part of the statistics is for understanding dataset. The part of the computer science is to interpret and process this dataset in the most efficient way. Efficient algorithms and large datasets are the keys for getting better results from machine learning.

1.1.2 Machine learning techniques

A very common problem for machine learning is to make a prediction with using a model.

This requires couple of things such as: training dataset, which is used for training the model. In these datasets, there are inputs such as images and output labels. Output label shows what is inside the image. The other thing is model training, which consists of three steps. Model is trained with the inputs from the training dataset, making the model predict the output labels and correcting the model with optimizers and loss functions.

There are four main machine learning techniques:

- **Supervised Learning:** It is based on learning a mapping from inputs to the outputs with the given labeled dataset [1].

  The most common supervised learning problems are regression and classification. Some of the algorithms that help us solve these problems are linear regression, decision tree and random forest.

- **Unsupervised Learning:** In this type of learning approach, there are only inputs. In other meaning no outputs or labels. The goal is to find the undetected patterns between the data inside the dataset.

  In this kind of problems, we need to find the patterns and we cannot use any clear loss function because we do not have a prediction label for the predicted output [1]. The most common unsupervised learning problems are generative modeling, dimensionality reduction and clustering. Some of the algorithms that help us solve these problems are k-means clustering, generative adversarial networks and principal component analysis.

- **Semi-Supervised Learning:** It is a combination of both supervised learning and unsupervised learning. The main purpose of this learning approach is to classify unlabeled data using the labeled data [2].

- **Reinforcement Learning:** In this learning approach, the machine learning model learns to make a decision from the sequences of reinforcements. Reinforcement learning is like a game. Artificial intelligence either gets a reward or a punishment for the decision that it is making.

1.1.3 Machine learning models

- **Discriminative Models:** These models are a branch of supervised machine learning. Discriminative models discriminate the classes with a decision boundary using the training dataset. They are used for regression and classification.

- **Generative Models:** These models are a branch of unsupervised machine learning. At first generative models tries to learn the distribution of the data from the training dataset. Then using this data distribution as an input, new examples can be crated.

  The goal here can be generating new examples that are indiscernible from the examples inside the training dataset or decreasing the noise and increasing the resolution of the images that are inside the testing dataset or in real world.

Mathematically we know the marginal probability of the input “P(input)”. We need to estimate the marginal probability of the output “P(output)” and conditional probability distribution “P (input | output)” (conditional probability of the input when output label is given) with generative modeling. Then by using the Bayes rule:

$$P(input)P(output|input) = P(output)P(input|output) \tag{1.1}$$

$$P(output|input) = P(output)\,P(output|input)\,/\,P(input) \tag{1.2}$$

From here, distribution of each class is modeled by finding the conditional probability distribution “P (output | input)” (conditional probability of output when input is given). New samples can be generated using this model.

Some of the examples for generative models are Naive Bayes, Deep Belief Network, Variational Autoencoder, and the Generative Adversarial Network (GAN).

1.1.4 Deep learning

Deep learning is a subset of machine learning that relies on neural networks with using large amounts of data. Structure of the neural networks is inspired by the human brain.

Some of the most common deep learning methods are Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Autoencoder and GAN.

The features are chosen by the neural network without human intervention. Neurons, the main asset of the neural networks, are the place that processing of information takes place. Neural networks have an input layer, hidden layers, and an output layer. As an example, the information which can be pixels of an image is given to the input layer of the neural network. In the last layer, which is the output layer, there are neurons that represent a digit. Between the input layer and output layer there are hidden layers. The geometry of the neural network is illustrated on the Figure 1.1.

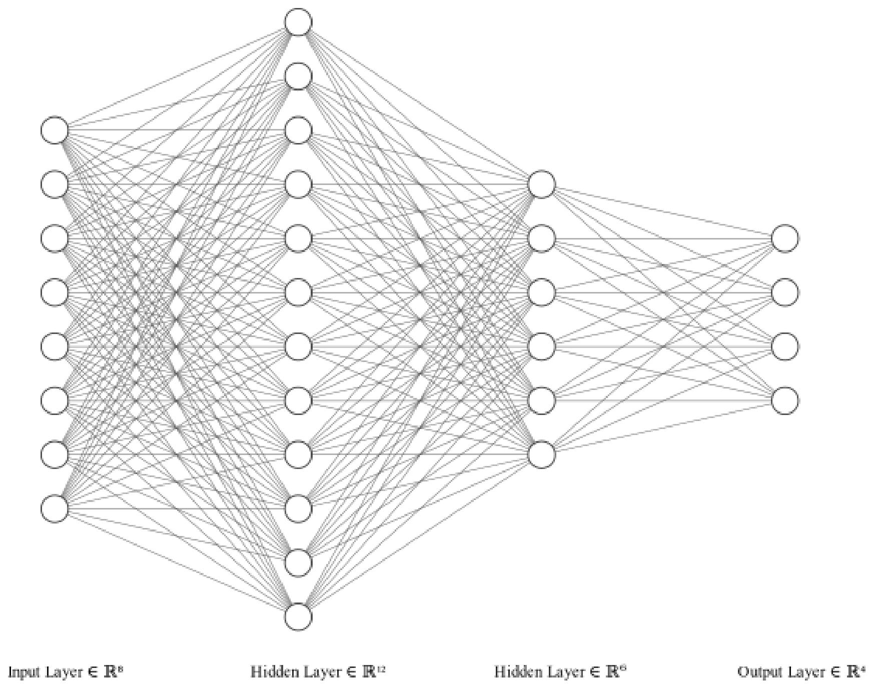

Figure 1.1 : Neural Network Example.

1.1.4.1 Working of neural networks

The information is transferred through the neural network layers with connecting channels. Each channel has a weight, so we can call them weighted channels. All the neurons may have its own bias. This bias value is added to the weighted sum of inputs that reaches to the neuron of the current neural network layer. Then this computation gets pass through the activation function. Activation function decides if the neuron will be activated or not by multiplying it with either a positive value or 0. Every activated neuron passes the information to the following layer. This continues until the last hidden layer. The neuron that is activated in the output layer corresponds to the inputs that is put through in input layer. The weights and biases are continuously adjusted through the process of neural network training.

1.2 Computer Vision Applications with GANs

In this thesis there are lots of generative adversarial network architectures and methods to tackle computer vision problems. These problems are generating fake images, super-resolving images and denoising the noisy images.

In this thesis, we studied three different computer vision applications which use generative adversarial networks for solution. The results indicate that GANs can be very effectively used for these particular problems. Before GAN there were other architectures which used generative modelling as well, but with the founding of GAN those architectures that used generative modelling to solve the computer vision problems became pretty much insufficient. Also, some of the image processing techniques that were used to solve computer vision problems also fell out of use.

From the practical standpoint, recent GAN architectures with the improved optimization techniques are easy to train, and the results are getting more accurate. The training, validation and testing parts are almost the same, the only differences are the loss functions and the optimizers. The procedure for making the dataset ready for training differs for each problem. Different pre-processing techniques and normalization techniques are used on those datasets. Also, the GAN part looks the same with two distinct networks, one being the generator the other one being the discriminator. Although these two architectures do the same job every time, where the generator tries to generate fake images and the discriminator tries to distinguish the generated fake images from the real images, these architectures get changed from one problem to another to tackle the particular characteristics of the problem.

1.2.1 Super resolution on images

GANs usage for this problem is to increase the resolution for the images and the videos. It has been used for different purposes in many areas. Some of them are as follows.

- Increasing the resolution for the old movies, commercials and shows to have better experience watching them.
- Increasing the resolution for the images and videos that are recorded with cameras that are not quite good to record high resolution images and videos. Mainly the cell phone cameras can be given as an example.
- Increasing the resolution in medical imaging for better medical imaging analysis.

1.2.2 Fake image generation

GANs usage for this problem is to create fake images that have not existed before. It has been used for different purposes in many areas. Some of them are as follows.

- Using fake image generation in the modeling agencies and commercial companies in order to not pay for the models or actors and actresses.
- Creating fake places that are not existed before in movies, thus no time or money is spent on finding new areas. In the future it is suggested that there will be no actors or actresses in movies, these people will be generated by the GAN.
- Creating datasets for the problems that are lacking enough data to solve the problem in artificial intelligence applications.

1.2.3 Image denoising

GANs usage for this problem is to denoise the images and the videos. It has been used for different purposes in many areas and these areas are so similar to super resolution problem. Some of them are as follows.

- Clearing the noises inside the video signals for the old movies, commercials and shows to have better experience watching them.
- Denoising the images and videos that are recorded with cameras that are not quite good to record noise free images and videos. The environment is also effective in this problem too. Low light and high heat can be given as an example for the environment problems. Mainly the cell phone cameras have this problem.
- Decreasing the noises in medical images for better medical imaging analysis.

1.3 Tools and Technologies

- **Language:** Python 3.6.9 has been used for this project. Python is an object-oriented programming (OOP) language which uses interpreter to turn the code into a machine code. It is one of the most used high-level languages in the world. It is used in web development, desktop application development, embedded software development and mostly in applications that uses data science.
- **Framework:** PyTorch 1.6.0 has been used in this project. PyTorch is a deep learning library mostly used for applications such as natural language processing and computer vision. It uses graphical processing units (GPUs) and central processing units (CPUs). Tensors are used in this library to make computation.

- **Environment:** Google Colab has been used to access all the technologies like Python programming language, PyTorch framework, machine learning libraries and GPUs like NVIDIA Tesla T4 GPU. Google Colab is an environment that allows us to write, run, save, and share code in Google Drive. It is basically a notebook which is composed of cells just like Jupyter Notebook. In those cells we can contain code, images, and text. Colab makes a connection with the cloud-based runtime with our notebook. Therefore, the Python code is executed without anything required locally. It uses GPUs, CPUs, and tensor processing units (TPUs) as a hardware accelerator.
- **Libraries:** NumPy and Python Imaging Library (PIL) are the third party-libraries that have been used in this project. PIL is an image processing library that allows us to open, manipulate and save images. NumPy is one of the most used libraries for scientific computation in data science applications. It is a third-party Python library that provides a multidimensional arrays and matrices, with a large collection of mathematical, basic, and advanced programming functions [3].

Tools and technologies that have been used in this project is summarized in the Table 1.1.

Table 1.1 : Tools and Technologies.

Tools	Tool Names
Programming Language	Python
Deep Learning Framework	PyTorch
Environment	Google Colab
3rd Party Libraries	PIL, NumPy
GPU	NVIDIA Tesla T4 GPU

1.4 Thesis Overview

This section provides an overview of the chapters within the thesis.

Chapter 1 is composed of theoretical background and tools and technologies. In the theoretical background section, the theory behind the project is explained from general to specific. In the tools and technologies section, the tools and technologies used in this project are explained.

Chapter 2 is composed of Generative Adversarial Networks (GAN). Structure and working mechanics of GAN are explained. Then the training of GAN parts and GAN is explained.

Chapter 3, 4 and 5 are composed in the order of the subjects called Super Resolution on Images, Fake Image Generation, and Image Denoising. Those chapters contain the same section titles. In literature review section, related works and methodologies about the application are explained briefly. In dataset section, the dataset used in training and testing for the application is introduced and the pre-processing operations used in the dataset are explained. In the neural network architecture section, the model used in training and testing for the application is explained.

Chapter 6 is composed of results. The results from Chapter 3, 4 and 5 are shown and explained here. In addition, the comparison of the results obtained from different models and methodologies has been made here.

Chapter 7 is composed of conclusion. The conclusions from the other chapters are explained here.

2. GENERATIVE ADVERSARIAL NETWORKS

2.1 Structure and Working Mechanics of GAN

Generative Adversarial Networks (GANs) are one of the examples of generative modelling which uses deep learning (DL) based methods. It is considered that GANs are the best way to train a generative model.

GANs consist of two parts. The first one is generator and the second one is discriminator.

- Generator’s mission is to create fake data that is indistinguishable from the real data for the discriminator. The generator uses feedback given by the discriminator when generating these fake data. These fake data are used in the training part of the discriminator.
- Discriminator’s mission is to distinguish the real data from the fake data that has been generated by the generator. If the generated fake data can be distinguished from the real data, then the discriminator gives a penalty to generator. The discriminator can be viewed as classifier in our situation. By looking at the data that the discriminator trying to classify, any deep learning based network architecture can be selected.

At first phases of the training, the generator creates fake data, and the discriminator learns easily to tell that the generated data from the generator is fake.

At the later stages of the training the generator starts to create fake data that can be indistinguishable for the discriminator.

Lastly, if the generator training is going bad then the discriminator will easily distinguish the real data from the generated fake data, but if the generator training goes well, the discriminator will struggle to distinguish the generated fake data from the real data and that can cause the discriminator to classify the generated fake data as a real data instead of classifying it as a fake data. This will make the discriminator’s accuracy decrease [4].

The working structure of training the GAN is in Figure 2.1 [5].

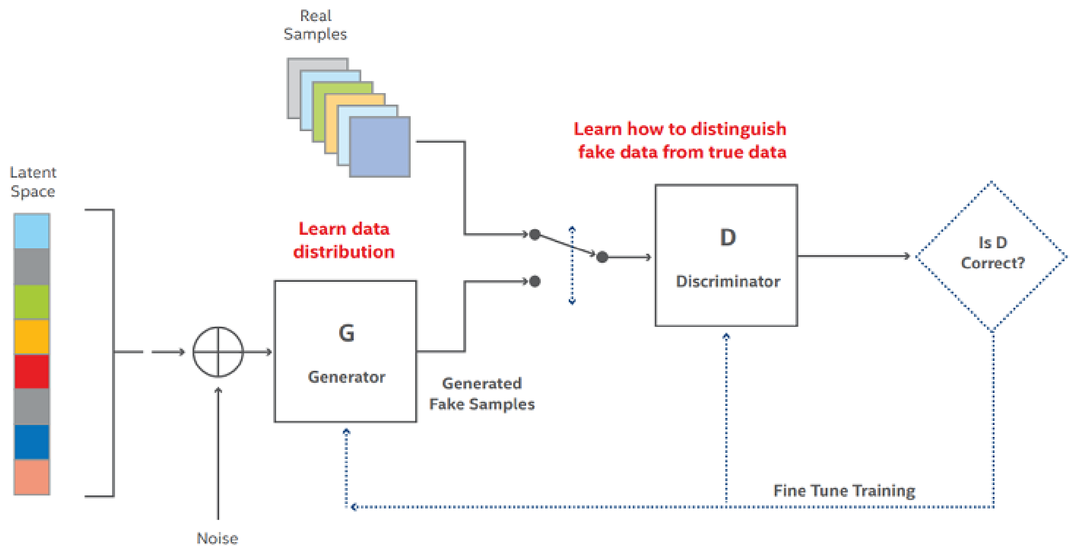

Figure 2.1 : Training the Generative Adversarial Network [5].

The working structure of testing the GAN is in Figure 2.2.

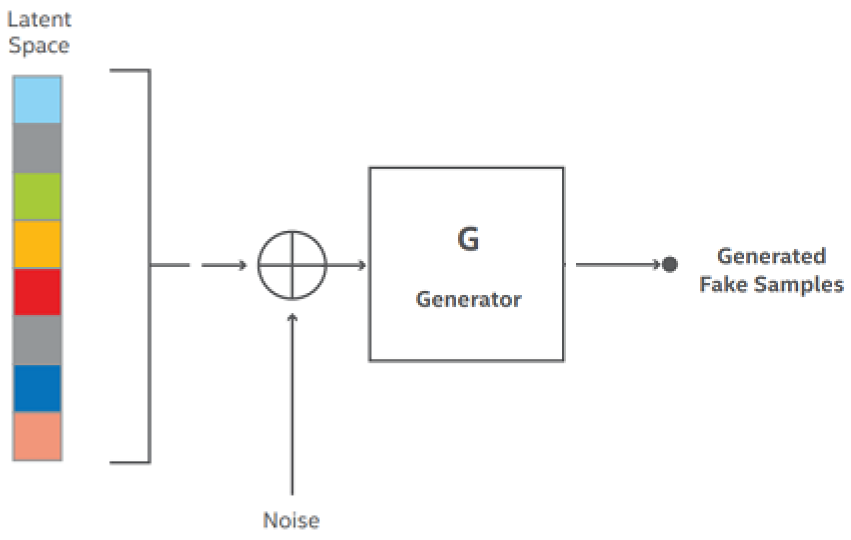

Figure 2.2 : Testing the Generative Adversarial Network.

In testing part of the GAN, generator loss and discriminator loss are not calculated with loss functions, the generator network and discriminator network are not getting optimized with the optimizer, discriminator does not distinguish the real data from the fake data. Hence, there is no backpropagation nor weight update in the neurons of the generator network and discriminator network. Only thing that happens in the testing part of the GAN is generator generating new samples from the given input samples.

2.2 GAN Training

In this section of the current chapter, the neural network training of the generative adversarial network is explained. First, parts of the GAN training as the generator training and the discriminator training, then the GAN training as a whole are explained both theoretically and practically.

2.2.1 Discriminator training

For the discriminator training there are two different data samples. The first one is called real data, which in our case images of the nature or people. The second one is called fake or generated data, which is the data that has been generated by the generator. Real data are called positive examples for the training of discriminator. Fake or generated data are called negative examples for the training of the discriminator.

Discriminator training procedure is as follows:

- Discriminator network tries to classify both the real data sample and the sample that has been generated by the generator.
- Loss function for the discriminator, calculates the losses for both real and fake data samples. Then the discriminator network gets penalized for misclassification between the real and generated samples by the loss function for the discriminator [6].
- The whole discriminator network is backpropagated with respect to the discriminator's loss function, and then the optimizer for the discriminator updates the model parameters. In short, the weights in each layer of the discriminator network is updated by backpropagation and optimization.
- Note: Different loss functions and optimizers can be used for the specific GAN problem.

2.2.2 Generator training

For the generator training there is one data sample which is a random input. This random input may be noise or a low-resolution image or a noisy image. If the random input is noise, we can understand that the distribution of noise is not that important by looking at the experiments, so choosing a distribution that is easy to sample is preferable [7]. As an example, uniform distribution can be chosen as a distribution that is easy to sample from.

Training the generator requires discriminator more than the discriminator training requires generator.

Generator training procedure is as follows:

- A random input is sampled, or a low-resolution image is gathered.
- Generator network creates an output from the random input.
- Real data and the fake data are put through the discriminator network and the validity predictions have been made as real and fake.
- Loss function for the generator, calculates the loss by using the classification made by the discriminator. Then the generator network gets penalized for creating data that cannot trick the discriminator.
- The whole generator network is backpropagated with respect to the generator loss function, and then the optimizer for the generator updates the model parameters. In short, the weights in each layer of the generator network is updated by backpropagation and optimization.
- Note: Different loss functions and optimizers can be used for the specific GAN problem.

2.2.3 GAN training as a whole

Generative adversarial networks have two different neural network architecture to train. GANs should run discriminator training and generator training together. Because the generator training and the discriminator training heavily rely on each other, they are trained alternatingly in one iteration. Generative models are a branch of unsupervised machine learning, but the training of the GAN architecture, which relies on generative modelling, is considered as supervised machine learning.

GAN training procedure is as follows:

- Train the generator to provide a batch of generated samples to the discriminator.
- Train the discriminator with the generated batch of samples from the generator and the real data samples.

- Repeat first 2 steps until the end of the all iterations in one epoch.
- Repeat the third step for one or more epochs.

There is a very common problem with GAN training, and it is called convergence of GAN. This problem occurs when the generator performance improves, and the discriminator performance decreases. This will cause the discriminator to hardly distinguish the difference between the generated data from the generator and the real data. In an ideal world, if the generator generates a batch of samples that are near perfect, then the discriminator could end up with having a 50% accuracy. This is often referred to as flipping a coin to make a prediction [8].

Therefore, the predictions of the discriminator lose its importance over the course of the GAN training and if the training continues from there then the discriminator will make completely random decisions and these decisions will return to the generator as a bad feedback to train the generator network with. This may cause generator quality to decrease as the training progresses to the later stages.

3. SUPER RESOLUTION ON IMAGES

3.1 Literature Review

There were not many strategies to super-resolve images by using GANs until the last few years. Deep learning started to come handy for solving this task and the research for this problem started to grow. The super-resolution models are created, and different learning methods applied to these models to solve this task, but most of them failed on real world images which are taken by devices such as smartphones.

The most widespread method for training the super-resolution deep learning models starts by downscaling the images that are inside the dataset with methods like nearest neighbor resampling, bicubic resampling and bilinear resampling. This process is applied in order to make a dataset that contains high-resolution and low-resolution training image pairs. The low-resolution images that are created by this process have almost no noise, in other words the images are clean.

The main purpose of these super-resolution deep learning models is to clean the images from the noises and increase the resolution of the images. But the given dataset for training the model contains clean images or noise reduced images. This leads the model to learn from the images that are almost noise-free.

After training super-resolution models with these strategies, the results are generally poor if the model is tested with real-world images that do not have any image processing methods applied by the developer, or in other words, images taken directly from the camera. The model leaves significant artifacts on the test images that are undesirable for super-resolving the images [9].

A real useful method that proposed in 2020 is real world super resolution with kernel estimation and noise injection [10]. Gathering a good dataset is very important for this problem, therefore a good strategy is proposed in this study.

- **Clean Up:** Bicubic downsampling was used for downsampling. The aim is to remove the high frequency noises from the training dataset images. Thus, higher resolution images were obtained in the training dataset [10, 11].

$$Img_{hr} = (Img_{src} * kernel_{bic}) \downarrow s \tag{3.1}$$

Img_{src} is the training dataset image, $kernel_{bic}$ is the bicubic kernel and s is the downsampling ratio.

- **Degradation:** Degradation operation was applied to high resolution images taken from the last step. The degradation operation was done using the KernelGAN [12]. Degradation pool was created with blur kernels in it. Then, a blur kernel was randomly selected from this pool and applied to the images [10].

$$Img_{ds} = (Img_{hr} * blur\_kernel_i) \downarrow s, i \in \{1, 2, 3, \dots . \mathrm{k}\} \tag{3.2}$$

  Img_{ds} is the downsampled image.

- **Noise Injection:** While getting the high-resolution images, some of the information were lost due to bicubic downsampling. Noise was added to the downsampled images to create realistic low-resolution images. These noise patches were collected from the training dataset images. After that, a filtering rule was designed to pick the noise patches from a decided range. These patches were then added to the downsampled images to obtain low-resolution images [10, 11].

$$Img_{lr} = Img_{ds} + noise_i \; i \in \{1, 2, 3, \dots . \mathrm{l}\} \tag{3.3}$$

- **Neural Network Training:** Enhanced Super Resolution Generative Adversarial Network (ESRGAN) [13] was used as a network architecture. Low resolution images are the fake data, high resolution image are the real data. Pixel loss, adversarial loss and perceptual loss were applied as loss functions for the training.

3.2 Dataset

For super resolution problem the dataset that has been used is Large-scale CelebFaces Attributes (CelebA) Dataset [14]. This dataset contains more than 200,000 celebrity images. The reason for using this dataset is, it is easier to train with small amounts of training data to super-resolve the images in this dataset. Therefore, Google Colab can be used easily. Google Drive only gives 15 GBs of space while working with Google Colab and Google Colab only allows to work with datasets that our Google Drive contains. Therefore, to use the GPU and pre-installed machine learning and deep learning libraries that Google Colab offers freely, Google Colab and Google Drive need to be used.

3.3 Neural Network Architecture

In this part of the current chapter, the neural network architecture that has been used to tackle the super-resolution problem is explained.

ESRGAN architecture [13] was created with using the SRGAN architecture [15] as the starting point. The neural network architecture of the SRGAN is in the Figure 3.1 [15].

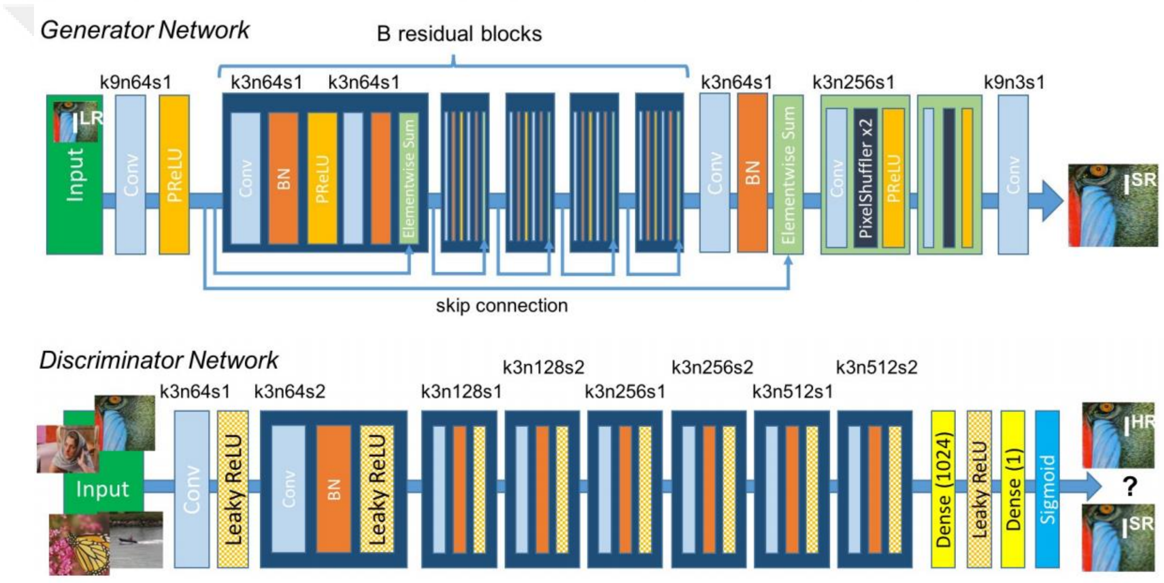

Figure 3.1 : Super Resolution GAN with Generator and Discriminator Network [15].

As the name Super Resolution Generative Adversarial Network (SRGAN) suggests it uses deep neural network with the adversarial network. Its main purpose is to super resolve the images to produce higher resolution images compared to the input images.

From the Figure 3.1 [15], the neural network architecture can be explained concisely as follows:

- **Convolutional Layers:** Convolutional layers are the main part of the convolutional neural networks. They're doing CNN's main job. It convolves the input, which can be image, with filter and bias and passes its result to the neurons in the next layer.

- **RELU:** Rectified Linear Unit (RELU) is an activation function for the neural networks. RELU outputs the negative values inside the neuron as 0 and outputs the positive values inside the neurons as it is. It is good for the vanishing gradient problem, but it might block gradient descent.
- **Leaky RELU:** To solve the gradient descent problem, Leaky Relu has been introduced as a new activation function. Leaky Relu outputs the negative values inside the neuron with multiplying with a constant and outputs the positive values inside the neurons as it is. The constant gets small values like 0.01.

$$Leaky\ RELU(x) = \max(ax, x),\ where\ a > 0 \tag{3.4}$$

- **Parametric RELU:** Leaky Relu made very small improvements on increasing the accuracy of the model. To increase the accuracy of the model, Parametric Relu has been introduced as another activation function. Parametric Relu outputs the negative values inside the neuron with multiplying with a learnable parameter during the training and outputs the positive values inside the neurons as it is.

$$Parametric\ RELU(x) = \max(0, x) + \min\ a_i(0, x) \tag{3.5}$$

- **Batch Normalization:** Batch normalization is a regularization function. It increases the training speed by reducing the number of epochs for the training. It stabilizes the neural network learning process.
- **Residual Blocks:** The residual blocks which is used for Deep Residual Learning [16] are used for easing the generator network training and to make the neural network deeper. These will result in an increase in neural network training performance.
- **Pixel Shuffler:** It is a sub-pixel convolutional layer. It learns the upscaling filter array during the training and upscales the map of low-resolution features to high resolution [17].
- **Dense Layer:** It is known as fully connected layer too. The values inside the neurons are multiplied by the weight of the channel and maybe if there is a bias, it is added to the multiplication.

Note: k9n64s1 means a convolutional layer with 9 kernels, 64 channels and it strides 1 pixel in the specified directions.

In order to improve the resolution of the output images, ESRGAN architecture [13] has been produced by making some modifications in SRGAN architecture [15].

In the generator architecture, all the batch normalization layers inside the residual blocks were deleted, and this resulted as an increase in the training performance of the architecture. Residual-in-Residual Dense Block (RRDB) was proposed instead of the basic block, which made it even deeper neural network architecture for the generator. The proposed method can be seen in the Figure 3.2 [13].

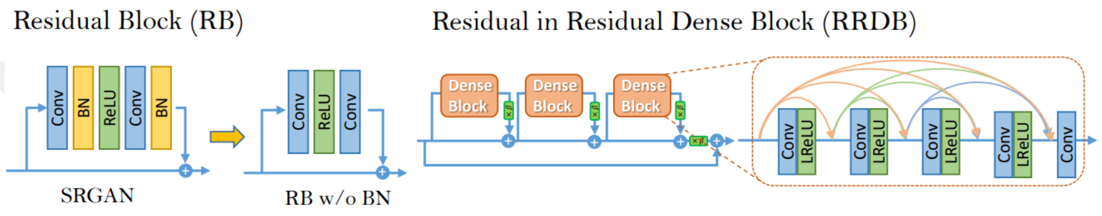

Figure 3.2 : RRDB in ESRGAN [13].

The working structure of training the ESRGAN [13] for super resolution is in Figure 3.3.

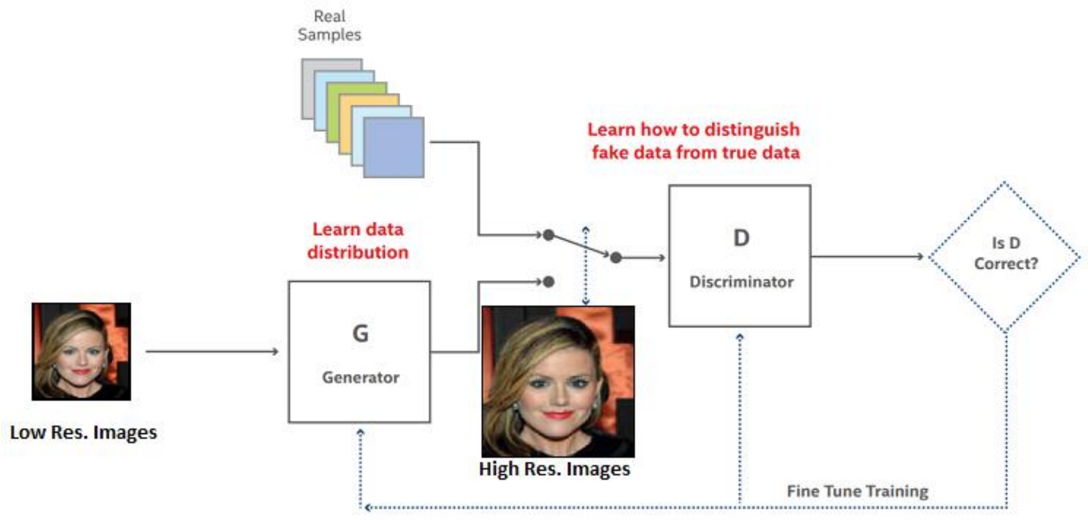

Figure 3.3 : Training the ESRGAN for Super Resolution.

The working structure of testing the ESRGAN [13] for super resolution is in Figure 3.4.

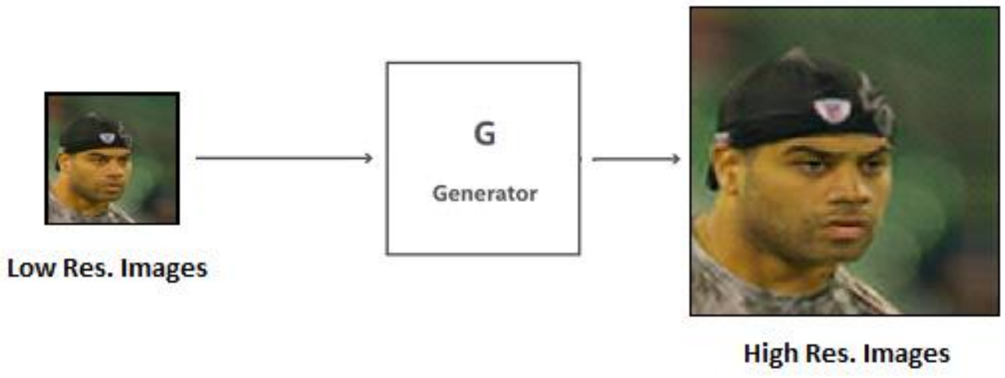

Figure 3.4 : Testing the ESRGAN for Super Resolution.

In both training and testing, the generator takes low resolution image as an input and generates high resolution image as an output.

3.4 Loss Functions and Optimizer

- **L1 Loss Function:** It means Least Absolute Deviations. L1 loss function [18] calculates the mean absolute error (MAE) between the generated image and the ground truth image.
- **Adversarial Loss Function:** For adversarial loss, a sigmoid layer combined with binary cross entropy [19] was used. This loss function uses sigmoid layer for activation function and then calculates the binary cross entropy between the output and the desired target.

Three different loss functions were used for generator training.

The first one is pixel-wise loss. For pixel-wise loss, L1 loss function was used. L1 loss function measured the pixel-wise loss between the generated high-resolution image and the high-resolution image.

The second one is content loss. For content loss, L1 loss function was used. L1 loss function measured the content loss between the extracted features of the generated high-resolution image and the extracted features of the high-resolution image. For feature extraction VGG19 model was used.

The third one is adversarial loss. For output parameter, high-resolution image and generated high-resolution image were put through the discriminator network and the validity predictions were made as real prediction and fake prediction. The difference between the fake prediction and real prediction gives the output. The desired target is a tensor filled with ones. Adversarial loss function used sigmoid layer for an activation function and then calculated the binary cross entropy between the output and the desired target.

The resulting loss function for the generator is as follows:

$$G_{loss} = content\ loss + 5e^{-3} * adversarial\ loss + \ e^{-2} * pixel\ loss \tag{3.6}$$

Two same loss functions were used for discriminator training.

The first one is adversarial loss for high resolution images. For output parameter, high-resolution image and generated high-resolution image were put through the discriminator network and the validity predictions were made as real prediction and fake prediction. The difference between the fake prediction and real prediction gives the output. The desired target is a tensor filled with ones. Adversarial loss function used sigmoid layer for an activation function and then calculated the binary cross entropy between the output and the desired target.

The second one is adversarial loss for generated high-resolution images. For output parameter, high-resolution image and generated high-resolution image were put through the discriminator network and the validity predictions were made as real prediction and fake prediction. The difference between the fake prediction and real prediction gives the output. The desired target is a tensor filled with zeros. Adversarial loss function used sigmoid layer for an activation function and then calculated the binary cross entropy between the output and the desired target.

The resulting loss function for the discriminator is as follows:

$$D_{loss} = \frac{(real\ image\ loss + generated\ image\ loss)}{2} \tag{3.7}$$

For the optimizer, which is an optimization algorithm, Adam was used for both generator network training and the discriminator network training.

4. FAKE IMAGE GENERATION

4.1 Literature Review

Fake image generation subject was started by Ian Goodfellow back in 2014. He and his colleagues founded the Generative Adversarial Network theory. Images have been generated from the datasets by using GAN architecture. These generated images look as if they are from the training dataset, but they are unique on their own because there are no such generated images in the training dataset.

In 2015, Deep Convolutional Generative Adversarial Networks (DCGAN) [21] were developed. It is a better version of the simple GAN. One more important thing that had come with DCGAN is the fact that traversing through the latent space of the generated image and changing the values in latent space dimensions can change the generated image drastically [21]. For example, with using vector arithmetic in the latent space of the generated images, a new generated image can be produced.

In 2016, Coupled Generative Adversarial Networks (CoGAN) [22] were developed. This GAN architecture consists of couple of GANs. Their responsibility is to generate images in only one domain. During their training process GANs share couple of parameters. As a result, GANs learn to generate similar images without the need of supervision of correspondence.

In 2017, progressive growing of GAN [23] was proposed. The key part on this method is the training methodology. Training is started with low resolution images and then the resolution of the input images is increased progressively with adding new layers to the generative adversarial network [23]. This method received a lot of attention at the time, as the generated images looked very close to real images.

In 2018, BigGAN [24] model was proposed. ResNet GAN architecture [25] has been used for the BigGAN model. BigGAN benefitted from scaling and it provided bigger generative adversarial networks and larger batch sizes. Neural networks have been trained with two or four times more parameters and eight times more batch size then the previous implementations [24]. As a result, the generated images looked indistinguishable from the real input images from the dataset that have been used to train the GAN.

4.2 Dataset

For fake image generation problem, two datasets have been used. The dataset, which is used for DCGAN [21], is Large-scale CelebFaces Attributes (CelebA) Dataset [14]. This dataset contains more than 200,000 celebrity images. The reasons for using this dataset are that there are lots of applications that uses this dataset and this dataset does not take too much space. Therefore, Google Colab can be used easily. Google Drive only gives 15 GBs of space while working with Google Colab and Google Colab only allows to work with datasets that our Google Drive contains. Therefore, to use the GPU and pre-installed machine learning and deep learning libraries that Google Colab offers freely, Google Colab and Google Drive need to be used. The dataset, which is used for BigGAN [24], is ImageNet dataset. This dataset contains more than 14,000,000 images. The reason for using this dataset is, this dataset is used with pre-trained BigGAN [24] models.

4.3 Neural Network Architectures

Two different generative adversarial network architectures have been implemented to solve the fake image generation problem. The first one is Deep Convolutional Generative Adversarial Networks (DCGAN) [21] and the second one is BigGAN [24].

4.3.1 Deep convolutional generative adversarial network

DCGAN architecture [21] was created with using the GAN architecture [20] for starting point. Generator of the DCGAN architecture is in the Figure 4.1 [21].

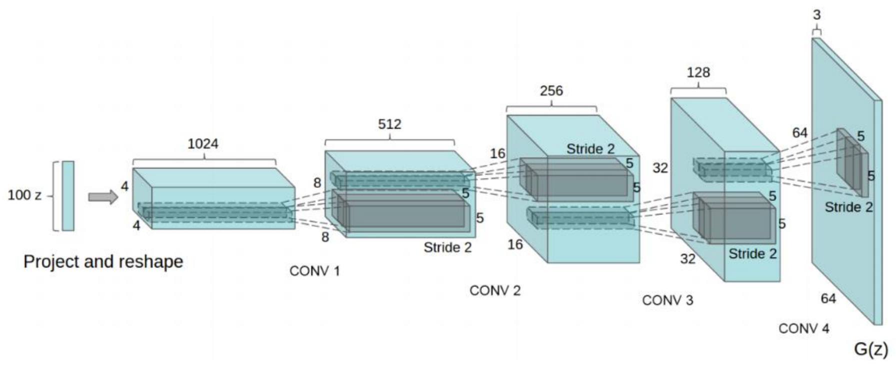

Figure 4.1 : Generator of the DCGAN [21].

Previous generative model architectures were checked, and drastic rule changes were made on them [21]. Those rules are as follows:

- Delete all the pooling layers and add strided deconvolutional layers on generator and add strided convolutional layers on discriminator.
- Use batch normalization as a regularization function both in generator and discriminator.
- For making the neural network architecture deeper, delete all the dense layers in hidden layers.
- Use Relu as an activation function for each layer in the generator except the last layer. In the output layer use Tanh as an activation function.
- Use LeakyRelu as an activation function for each layer in the discriminator.

In the last layer of the discriminator a sigmoid layer has been used.

- **Sigmoid:** Sigmoid is an activation function for the neural networks. Sigmoid function is used because it outputs a value between 0 and 1. It is useful for the discriminator because it predicts a probability as an output. If the output value is close to 1, it means the data is real, if it is close to 0, it means the data is fake.

In the last layer of the generator a Tanh layer has been used.

- **Tanh:** Tanh is an activation function for the neural networks. Tanh function is like sigmoid function. It outputs a value between -1 and 1, therefore the negative inputs are also mapped too. It has been shown that using an activation function that is bounded, makes the model learn faster.

4.3.2 BigGAN

BigGAN [24] model was created with using the Self-Attention Generative Adversarial Networks (SAGAN) [25]. As the name suggests BigGAN's main focus is on bigger generative adversarial network. Generator of the BigGAN's neural network architecture is in the Figure 4.2 [24].

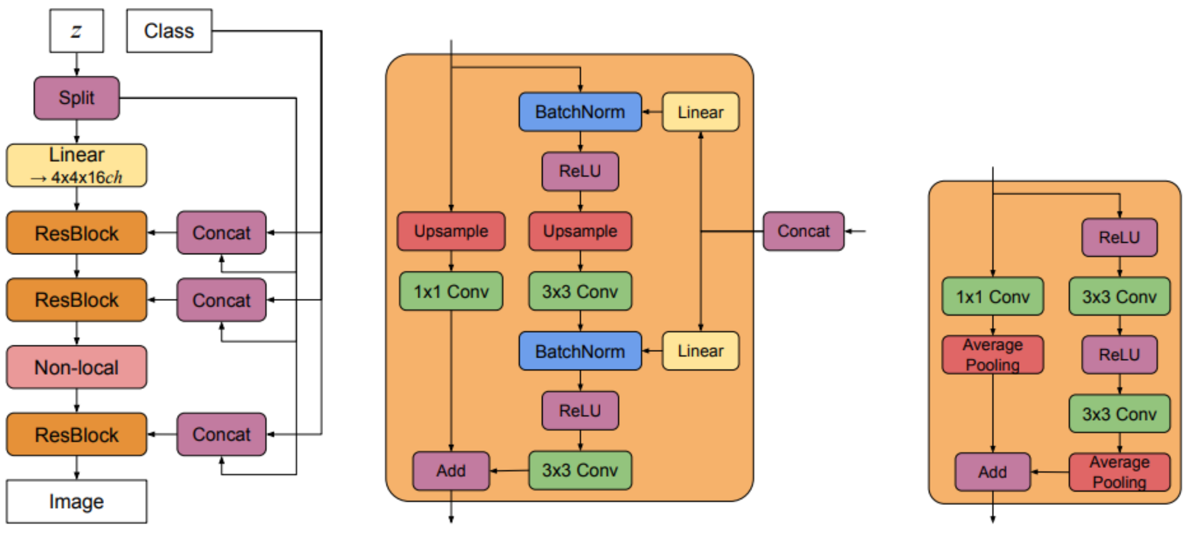

Figure 4.2 : (a) Generator of the BigGAN. (b) Residual Block in generator. (c) Residual Block in discriminator [24].

SAGAN architecture [25] was checked and drastic rule changes were made on them [21]. Those rules are as follows:

- Improve the batch size. Experiments suggested increasing the batch size by multiplying with 8 gives the best result [24].
- Increase the number of channels contained in each layer by about 50% [24].
- Utilize a one shared class embedding instead of using many in the generator. Experiments suggested that it can improve the training performance of the architecture by improving the training speed and reducing the memory usage and computation. Its implementation was shown in the Figure 4.2 [24]. Split latent vector, shown as z in the Figure 4.2 [24], into one stack per pixel. Concatenate each stack to the shared class. Pass them to the residual blocks. From there pass it to the linear layer and then to the batch norm layer.

In the residual block part, there are new layer types that are added to the architecture.

- **Linear:** Linear layer applies a linear transformation to the input data passing through this layer. Input data is multiplied with the weight of the neuron and from there the bias of the neuron is added to the result from the multiplication.
- **Average Pooling:** Average pooling layer is a pooling layer. Pooling layers are added after the convolutional layers. Its function is to avoid the overfitting by reducing the size of the neural network, in this way the computation in the neural

network training and the parameter amount is reduced. Average pooling layer divides the input into pooling regions by performing down-sampling and computes the average values of all the inputs in every single region [26].

The working structure of training the DCGAN [21] and BigGAN [24] for fake image generation is in Figure 4.3.

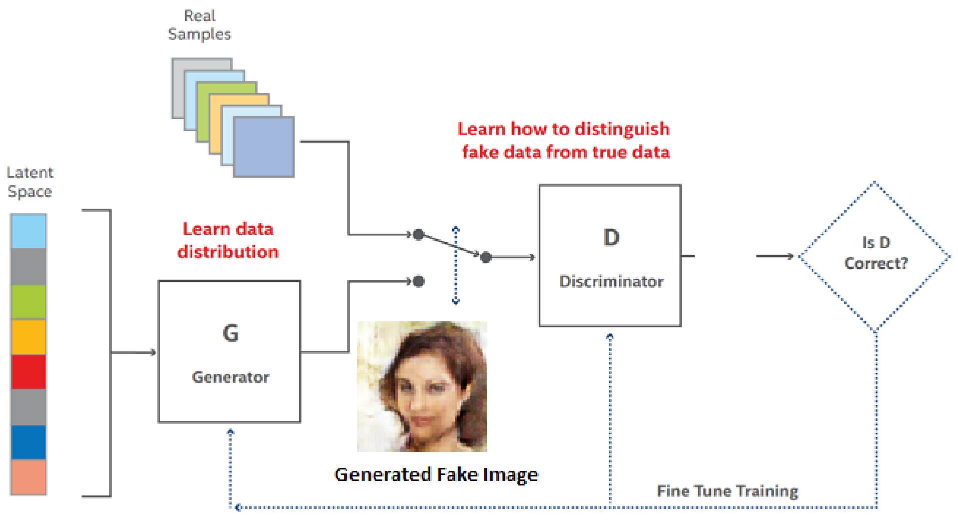

Figure 4.3 : Training the DCGAN and BigGAN for Fake Image Generation.

The working structure of testing the DCGAN [21] and BigGAN [24] for fake image generation is in Figure 4.4.

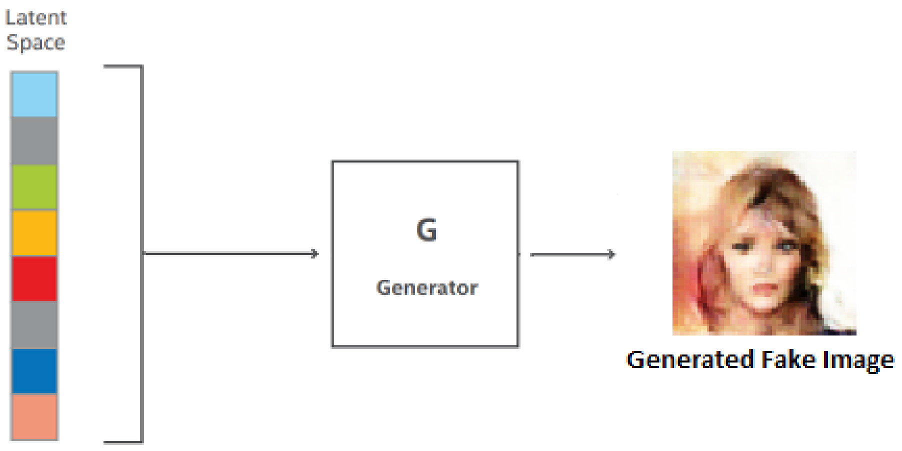

Figure 4.4 : Testing the DCGAN and BigGAN for Fake Image Generation.

In both training and testing, the generator takes a noise vector as an input and generates fake image as an output.

4.4 Loss Functions and Optimizer

In this part of the chapter, loss functions and the optimizer for the DCGAN [21] and BigGAN [24] are explained.

4.4.1 DCGAN loss function and optimizer

- **BCE Loss Function:** It means binary cross entropy loss function [27]. BCE loss function calculates the binary cross entropy between the output and the desired target.

Two same loss functions were used for discriminator training.

First one is binary cross entropy loss for real images. For output parameter, real image was put through the discriminator network and the validity prediction was made as real or fake prediction. The desired target is a tensor filled with ones. Then, binary cross entropy between the output and the desired target was calculated.

The second one is binary cross entropy loss for generated images. For output parameter, generated image was put through the discriminator network and the validity prediction was made as real or fake prediction. The desired target is a tensor filled with zeros. Then, binary cross entropy between the output and the desired target was calculated.

Gradients of the both two loss function were calculated separately. In other meaning, whole discriminator network had to be backpropagated with respect to the discriminator loss functions twice.

The resulting loss function for the discriminator is as follows:

$$D_{loss} = (real\ image\ loss + generated\ image\ loss) \tag{4.1}$$

One loss function was used for generator training.

The loss function used for the generated images is the binary cross entropy loss. For output parameter, generated image was put through the discriminator network and the validity prediction was made as real or fake prediction. The desired target is a tensor filled with zeros. Then, binary cross entropy between the output and the desired target was calculated.

$$G_{loss} = \ generated\ image\ loss \tag{4.2}$$

For the optimizer, which is an optimization algorithm, Adam was used for both generator network training and the discriminator network training.

4.4.2 BigGAN loss functions and optimizer

- **Hinge Loss Function:** Hinge loss function is used for classification. This loss function tries to find the best decision boundary for the classification task.

Two same loss functions were used for discriminator training.

First one is hinge loss for real images. Real image was put through the discriminator network and the validity prediction was made as real or fake prediction. Hinge loss function used the prediction as an input and calculated the real image loss.

The second one is hinge loss for generated images. Generated image was put through the discriminator network and the validity prediction was made as real or fake prediction. Hinge loss function used the prediction as an input and calculated the generated image loss.

The resulting loss function for the discriminator is as follows:

$$D_{loss} = \frac{(real\ image\ loss + generated\ image\ loss)}{2} \tag{4.3}$$

One loss function was used for generator training.

The loss function used for the generated images is the hinge loss. Generated image was put through the discriminator network and the validity prediction was made as real or fake prediction. Hinge loss function used the prediction as an input and calculated the generated image loss. The calculation of the generator loss is the same as for DCGAN generator loss.

5. IMAGE DENOISING

5.1 Literature Review

Deep learning methods have been used to tackle image processing problems for quite some time. Denoising is one of the most known image processing problems to this date.

There are different methods to tackle this problem. The most traditional ones are the image processing methods. Denoising the images with the linear filters, non-linear filters, adaptive filters can be given as an example for the traditional image processing techniques.

CNN architectures have been proposed to tackle this problem. In [28] a GAN was trained to learn the noise distribution and then CNN was used to denoise the images.

In literature there are not GAN architectures that are solely adapted to image denoising problem. For this problem, using an SRGAN [15] like architecture was proposed in [29]. It has understood that using the SRGAN like architecture not only works for improving the image resolution but to denoise the images as well. It has been shown in [30] that the DCGAN architecture [21] can be used in more than one image processing problem. It has been understood that this architecture can be used not only for fake image generation, but also for problems such as image denoising, super resolving the images and deblurring the images as well.

Nowadays, deep learning methods are much more popular than the traditional image processing techniques because, these methods can learn from the datasets, create their own features that people cannot understand and preserve the image details better because of the learning aspect.

5.2 Dataset

For image denoising problem the dataset that has been used is Large-scale CelebFaces Attributes (CelebA) Dataset [14]. This dataset contains more than 200,000 celebrity images. The reason for using this dataset is, it is easier to train with small amounts of training data to denoise the images in this dataset. Therefore, Google Colab can be used easily. Google Drive only gives 15 GBs of space while working with Google Colab and Google Colab only allows to work with datasets that our Google Drive contains. Therefore, to use the GPU and pre-installed machine learning and deep learning libraries that Google Colab offers freely, Google Colab and Google Drive need to be used.

5.3 Neural Network Architecture

Updated form of the DCGAN [21] architecture has been proposed and used for the denoising problem as a GAN architecture. In the generator of the neural network architecture some changes have been made to generate bigger images and also to generate slightly better images.

Generator of the DCGAN's architecture is in the Figure 4.1 [21].

Here are the some of the changes that have been made on the DCGAN architecture [21].

- Image size kept the same while going forward in the layers of the generator but, the channel size had changed. To keep the image size the same; kernel size was set to 2, striding was set to 1 and padding was set to 1 in first, third and fifth convolutional layers and 0 in second, fourth and sixth convolutional layers.
- An extra hidden layer was added to the generator to make the neural network denser and to keep the image size as the same. This hidden layer consists of convolutional layer and batch normalization. Batch normalization has been used as an activation function for the output of the convolutional layer.
- Some changes have been made on the discriminator because of the changes in the generator.

The working structure of training the updated DCGAN [21] for image denoising is in Figure 5.1.

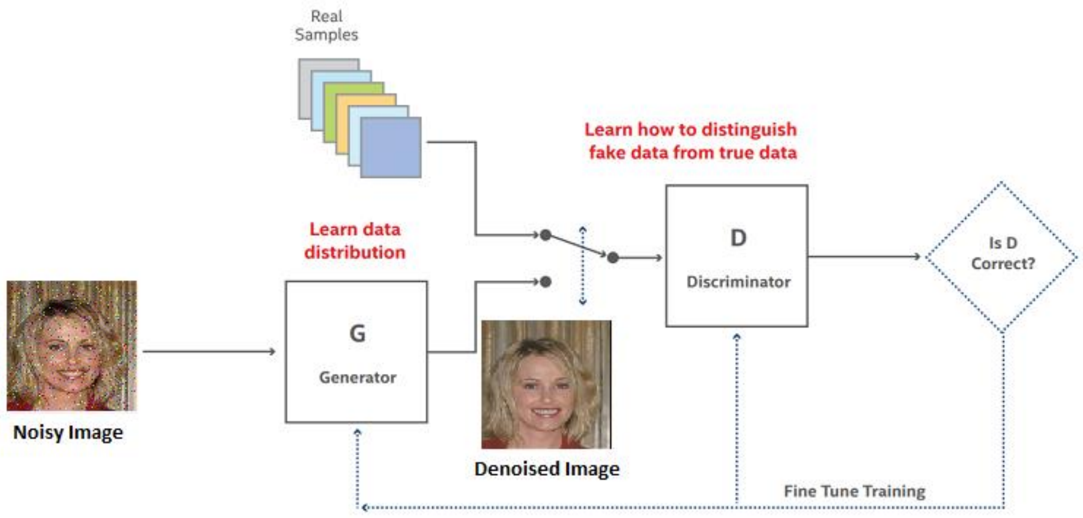

Figure 5.1 : Training the Updated DCGAN for Image Denoising.

The working structure of testing the updated DCGAN [21] for image denoising is in Figure 5.2.

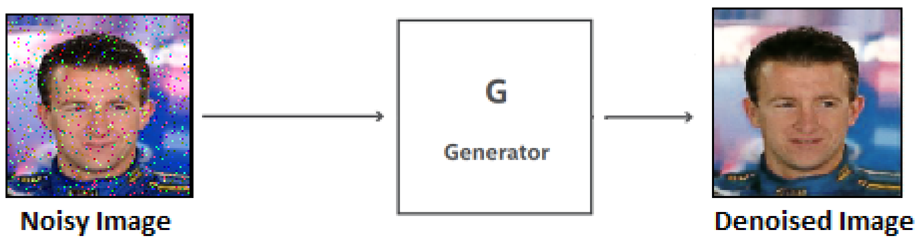

Figure 5.2 : Testing the Updated DCGAN for Image Denoising.

In both training and testing, the generator takes noisy image as an input and generates denoised image as an output.

5.4 Loss Functions and Optimizer

Two same loss function were used for discriminator training.

First one is binary cross entropy loss for real images. For output parameter, real image was put through the discriminator network and the validity prediction was made as real or fake prediction. The desired target is a tensor filled with ones. Then, binary cross entropy between the output and the desired target was calculated.

The second one is binary cross entropy loss for generated images. For output parameter, generated image was put through the discriminator network and the validity prediction was made as real or fake prediction. The desired target is a tensor filled with zeros. Then, binary cross entropy between the output and the desired target was calculated.

Gradients of the both two loss function were calculated separately. In other meaning, whole discriminator network had to be backpropagated with respect to the discriminator loss functions twice.

The resulting loss function for the discriminator is as follows:

$$D_{loss} = (real\ image\ loss + generated\ image\ loss) \tag{5.1}$$

One loss function was used for generator training.

The loss function used for the generated images is the binary cross entropy loss. For output parameter, generated image was put through the discriminator network and the validity prediction was made as real or fake prediction. The desired target is a tensor filled with zeros. Then, binary cross entropy between the output and the desired target was calculated.

$$G_{loss} = generated\ image\ loss \tag{5.2}$$

For the optimizer, which is an optimization algorithm, Adam was used for both generator network training and the discriminator network training.

In addition to the loss functions that were used during the training process, Peak signal-to-noise ratio (PSNR) values were also calculated.

- **Peak Signal-to-Noise Ratio:** PSNR is the ratio between an image's highest potential power and the power of corrupting noise that influences its representation accuracy [31]. PSNR is used as a quality metric in image processing applications. It is calculated between the clean image and the noisy image or clean image and denoised image. PSNR is expressed in terms of the logarithmic decibel scale. To calculate the PSNR, the mean square error (MSE) must be calculated first.

$$MSE = \frac{1}{rows * cols} \sum_{i=0}^{rows-1} \sum_{j=0}^{cols-1} (Img(i,j)_{org} - Img(i,j)_{deg})^2 \tag{5.3}$$

Then the MSE is used in the PSNR calculation.

$$PSNR = 20\, log_{10}(\frac{max\ intensity}{\sqrt{MSE}}) \tag{5.4}$$

Max intensity can be taken as 255 because, 255 is the maximum pixel value.

PSNR values were calculated only in the testing part.

In the GAN method, PSNR calculations were made between the generated images and the ground truth images and also between the noisy input images and the ground truth images.

In the image processing method, PSNR calculations were made between the denoised images and the ground truth images and also between the noisy input images and the ground truth images.

6. RESULTS

In this chapter, the results for all the GAN problems in this thesis were shown and discussed. These results contain input images from the dataset, generated output images from the input images, model, and method comparisons, and plottings of the training loss curves.

6.1 Super Resolution on Images

For super resolution two different methods were used. The first one is ESRGAN [13], trained to generate high-resolution images from the low-resolution images. The second one is nearest neighbour interpolation algorithm, used to upsample the low-resolution images.

6.1.1 ESRGAN

Input images were used in neural network training with four images per batch in one iteration. Only two images per one batch were shown below.

In Figure 6.1 the first examples of low-resolution images have been given. In Figure 6.2, generated output images for every 2000 iteration have been given. Generated output images have been created from the low-resolution images in Figure 6.1.

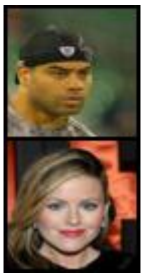

Figure 6.1 : First Example of Low-Resolution Images.

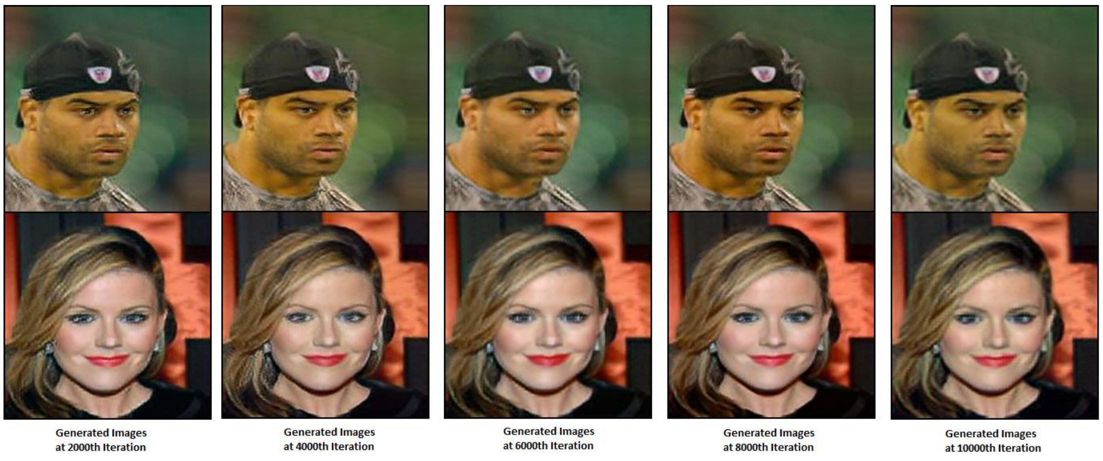
Generated Images at 2000th Iteration
Generated Images at 4000th Iteration
Generated Images at 6000th Iteration
Generated Images at 8000th Iteration
Generated Images at 10000th Iteration

Figure 6.2 : First Example of Generated Output Images for every 2000 Iteration.

In Figure 6.3 the second examples of low-resolution images have been given. In Figure 6.4 generated output images for every 2000 iteration have been given. Generated output images have been created from the low-resolution images in Figure 6.3.

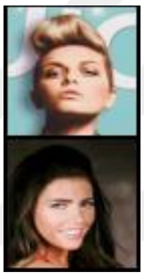

Figure 6.3 : Second Example of Low-Resolution Images.

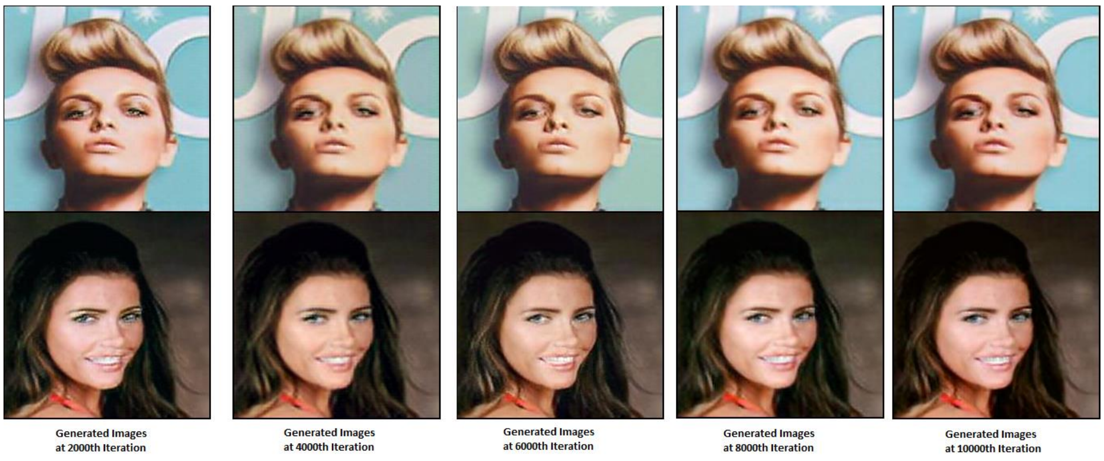
Generated Images at 2000th Iteration
Generated Images at 4000th Iteration
Generated Images at 6000th Iteration
Generated Images at 8000th Iteration
Generated Images at 10000th Iteration

Figure 6.4 : Second Example of Generated Output Images for every 2000 Iteration.

In Figure 6.5 the third examples of low-resolution images have been given. In Figure 6.6 generated output images for every 2000 iteration have been given. Generated output images have been created from the low-resolution images in Figure 6.5.

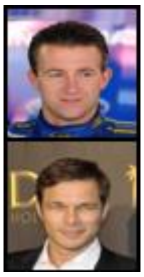

Figure 6.5 : Third Example of Low-Resolution Images.

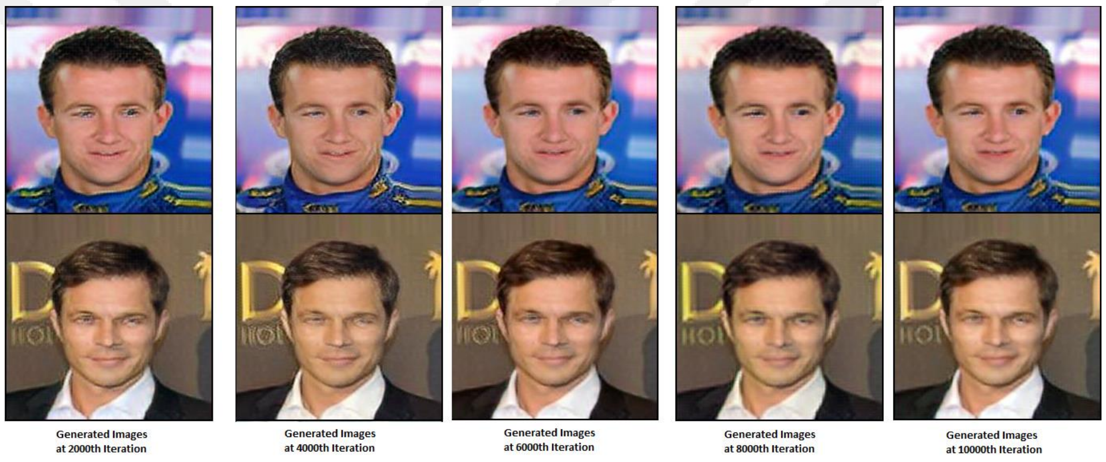
Generated Images at 2000th Iteration
Generated Images at 4000th Iteration
Generated Images at 6000th Iteration
Generated Images at 8000th Iteration
Generated Images at 10000th Iteration

Figure 6.6 : Third Example of Generated Output Images for every 2000 Iteration.

From the images above, it can be observed that the results get better as the training continues.

Plotting of the generator’s training loss curve is in the Figure 6.7.

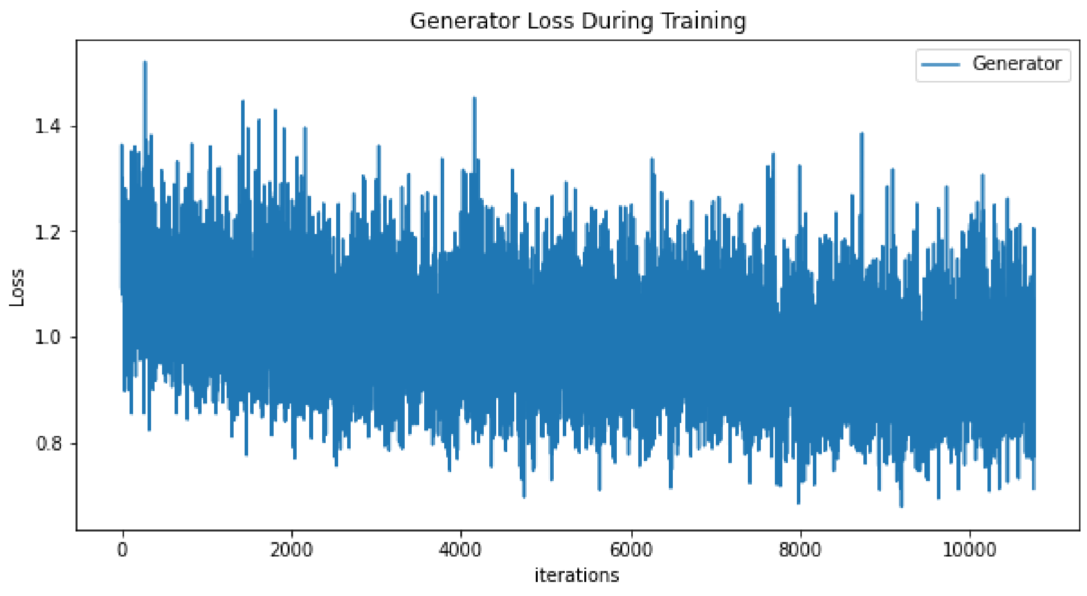

Figure 6.7 : Generator Loss for Super-resolving.

Plotting of the discriminator’s training loss curve is in the Figure 6.8.

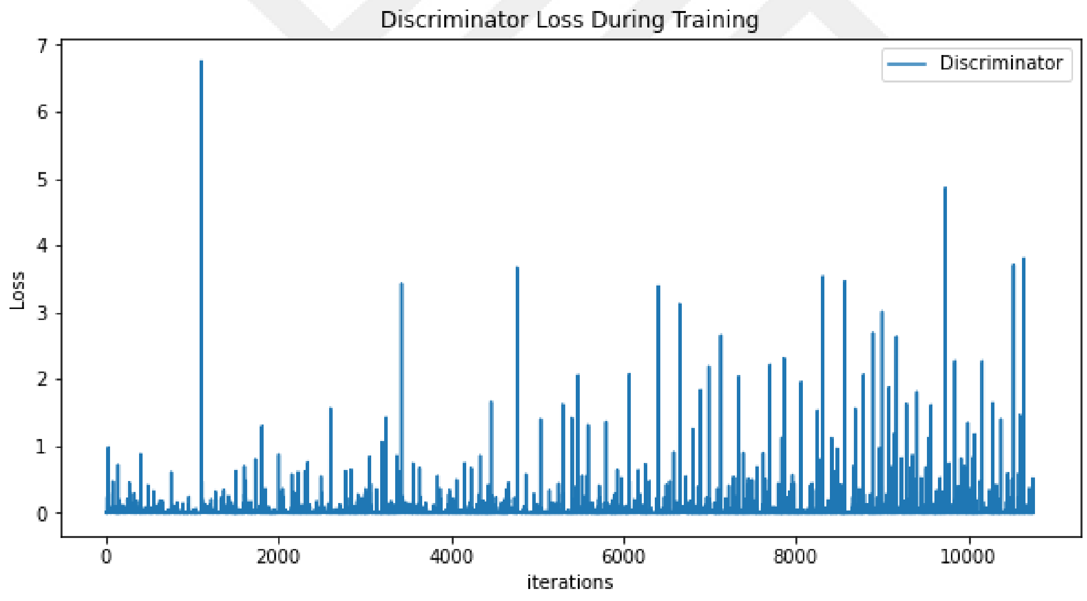

Figure 6.8 : Discriminator Loss for Super-resolving.

First epoch’s iterations are considered as warm-up iterations and only the pixel-wise loss gets calculated and the whole generator network is backpropagated with respect to the pixel-wise loss function. Then the optimizer for the generator, updates the model parameters. During warm-up iterations, discriminator network never gets to backpropagate with respect to any loss function and never updates its model parameters. In the results above warm-up iterations have not shown.

6.1.2 Nearest neighbour interpolation

This is an image processing technique to up sample the low-resolution images. Therefore, there are no loss functions for this part.

In Figure 6.9 the first examples of low-resolution images and the upsampled images with nearest neighbour interpolation algorithm have been given. In Figure 6.10, upsampled images and generated output images have been given. Upsampled images and generated output images have been created from the low-resolution images in Figure 6.9.

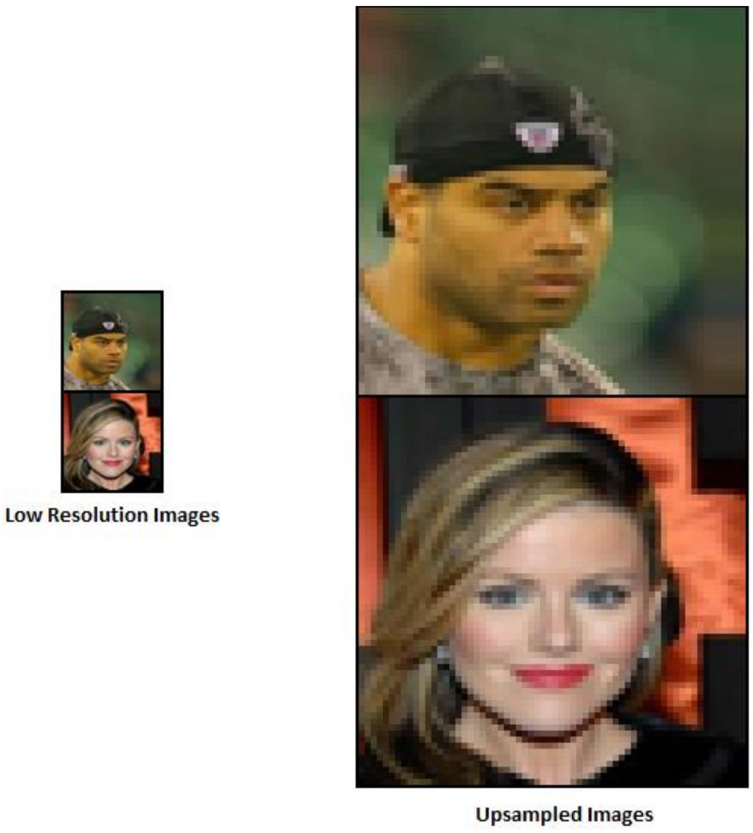

Figure 6.9 : First Example of Low-Resolution Images and Upsampled Images.

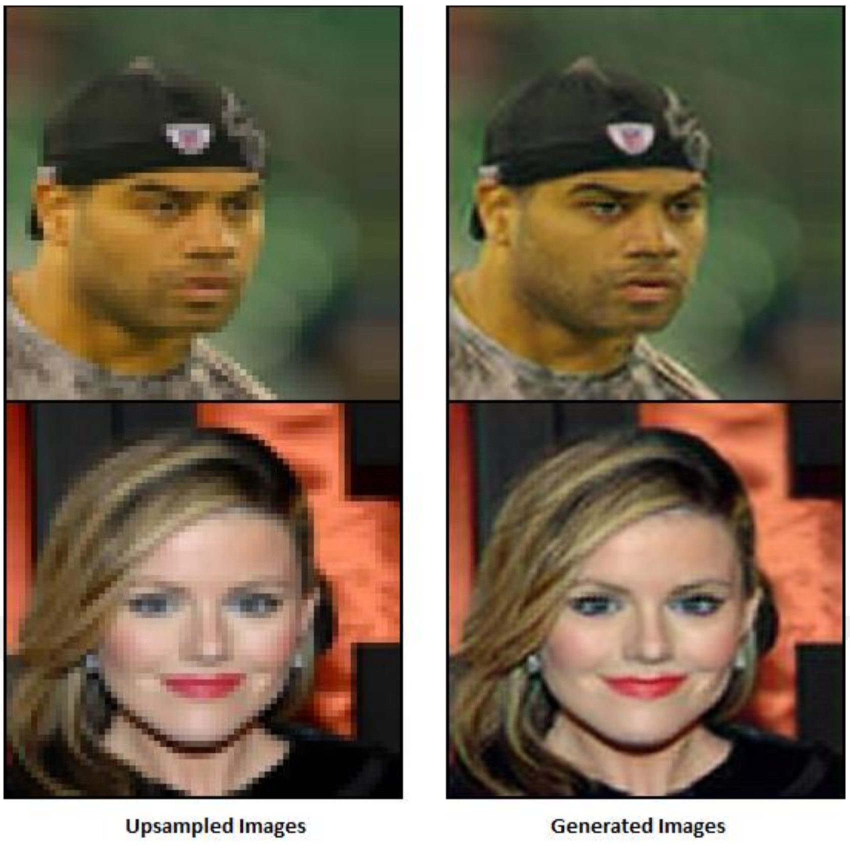

Figure 6.10 : First Example of Upsampled Images and Images Generated by GAN.

In Figure 6.11 the second examples of low-resolution images and the upsampled images with nearest neighbour interpolation algorithm have been given. In Figure 6.12, upsampled images and generated output images have been given. Upsampled images and generated output images have been created from the low-resolution images in Figure 6.11.

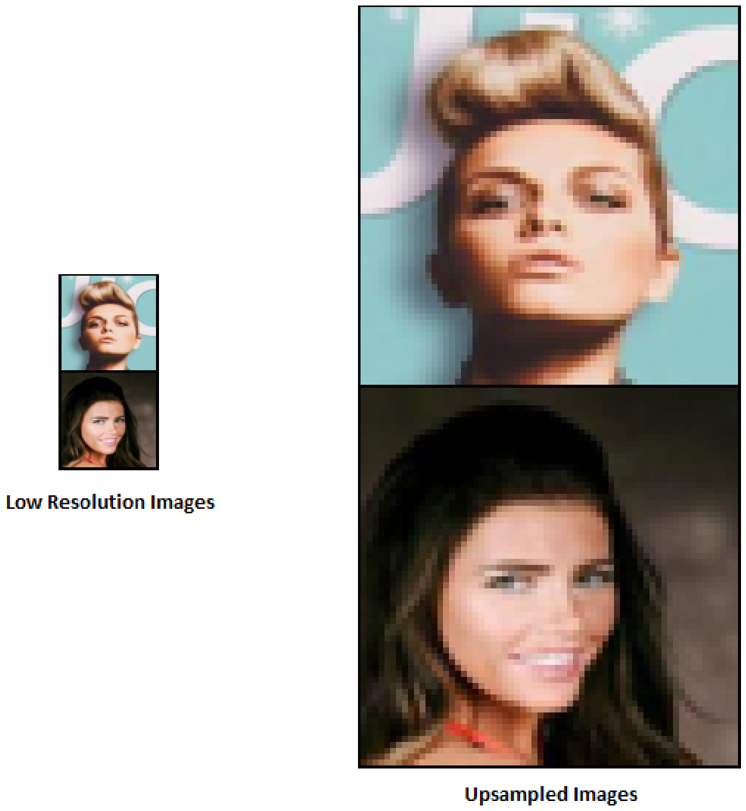

Figure 6.11 : Second Example of Low-Resolution Images and Upsampled Images.

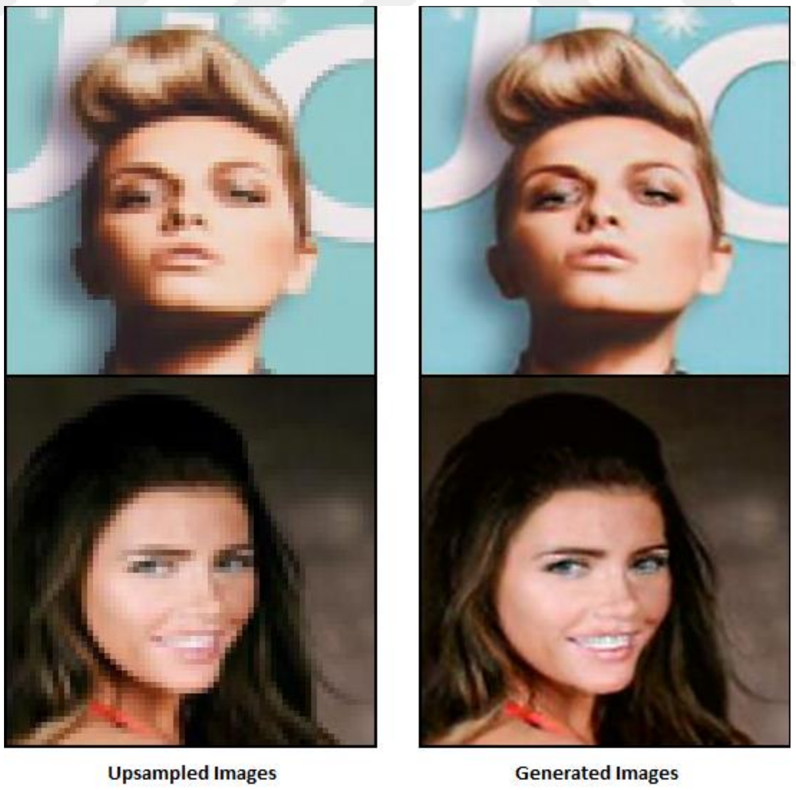

Figure 6.12 : Second Example of Upsampled Images and Images Generated by GAN.

In Figure 6.13 the third examples of low-resolution images and the upsampled images with nearest neighbour interpolation algorithm have been given. In Figure 6.14, upsampled images and generated output images have been given. Upsampled images and generated output images have been created from the low-resolution images in Figure 6.13.

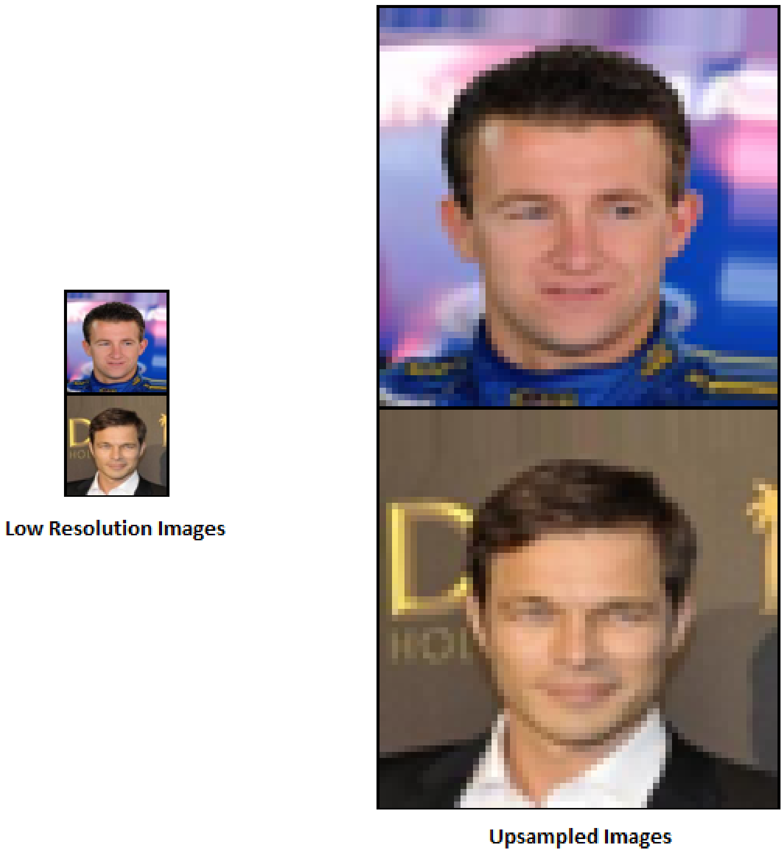

Figure 6.13 : Third Example of Low-Resolution Images and Upsampled Images.

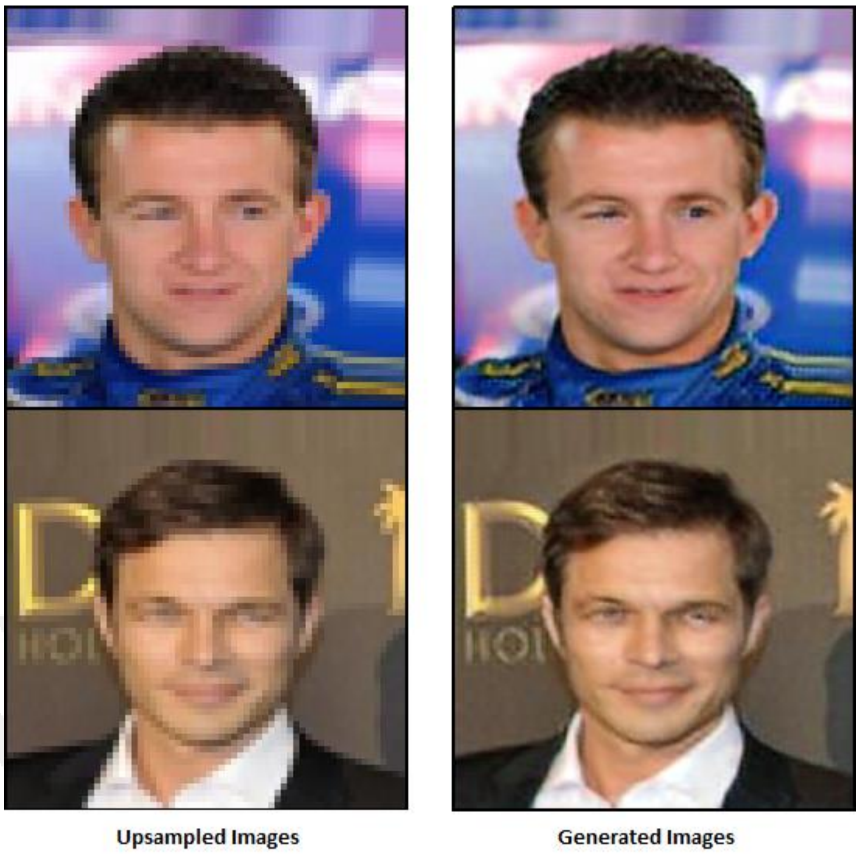

Figure 6.14 : Third Example of Upsampled Images and Images Generated by GAN.

As it has shown above in the figures, ESRGAN [13] increased the resolution of the images better than the nearest neighbour interpolation algorithm. As for the result it can be understood that GAN performed better than the image processing technique.

6.2 Fake Image Generation

For fake image generation two different models were used. The first model, which is DCGAN [21], trained with different batch sizes to generate fake images. For the second model, which is BigGAN [24], a pre-trained model was used to generate fake images.

6.2.1 DCGAN

Input images were used in neural network training with four images per batch in one iteration and one image per batch in one iteration.

For the training with four images per one batch, input images and the generated output images were shown below in the following Figure 6.15 and Figure 6.16.

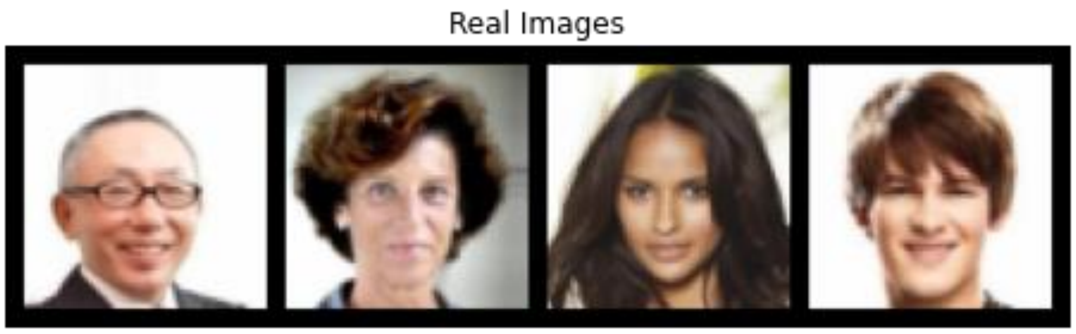

Figure 6.15 : Input Image Examples with the Batch Size of 4 (CelebA).

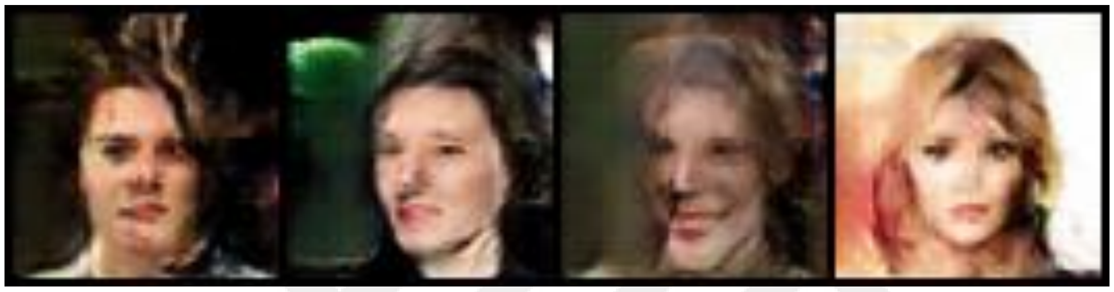

Figure 6.16 : Generated Images with the Batch Size of 4 (CelebA).

Plotting of the generator's training loss curve with the batch size of 4 is in the Figure 6.17.

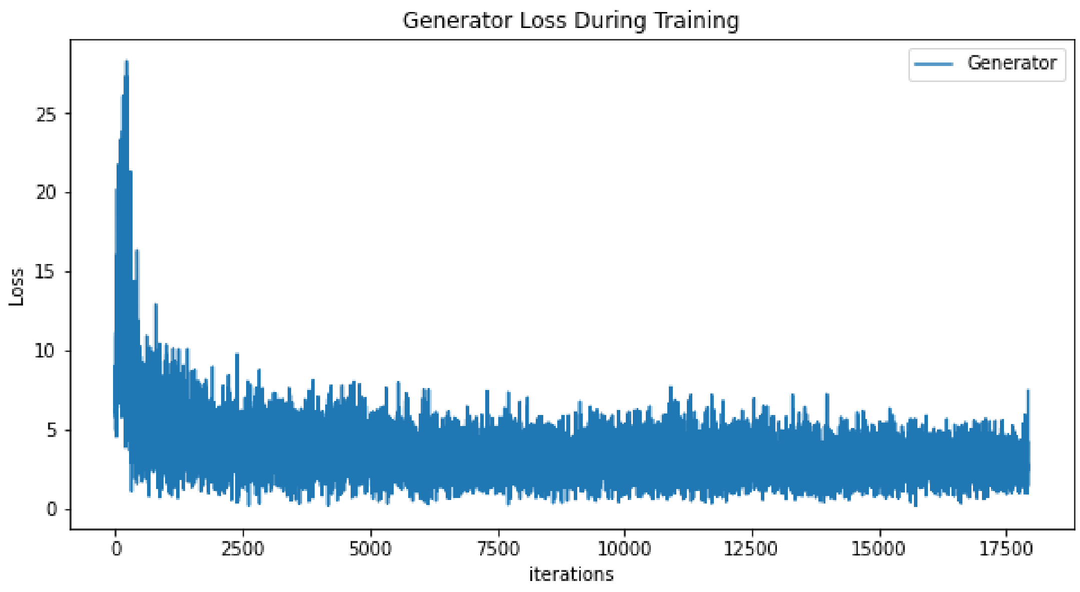

Figure 6.17 : Generator Loss for Image Generation with the Batch Size of 4 (DCGAN).

Plotting of the discriminator's training loss curve with the batch size of 4 is in the Figure 6.18.

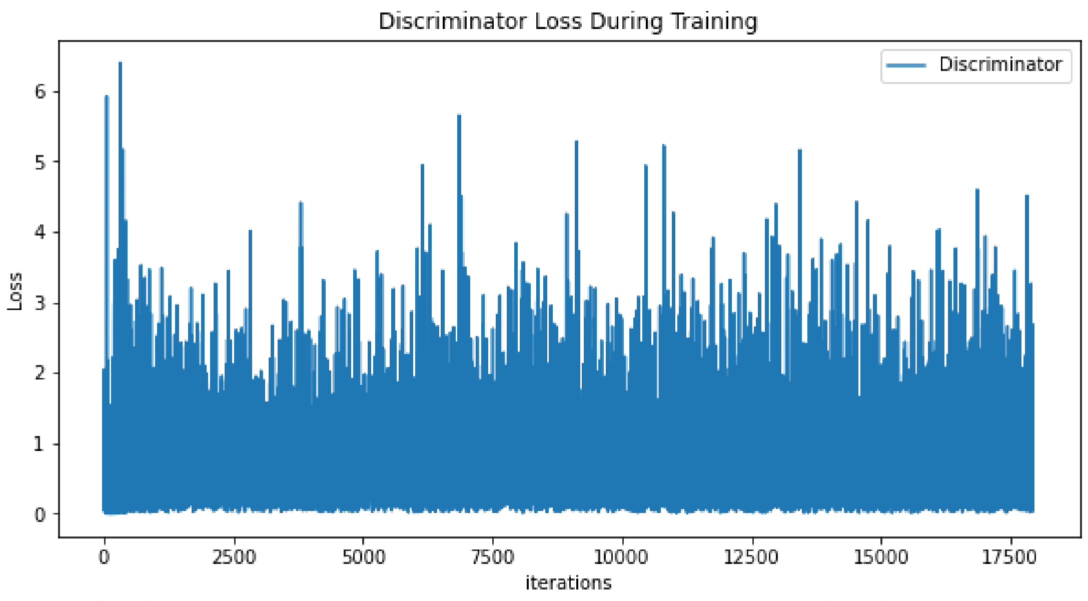

Figure 6.18 : Discriminator Loss for Image Generation with the Batch Size of 4 (DCGAN).

For the training with one image per one batch, input image and the generated output image were shown below in the following Figure 6.19 and Figure 6.20.

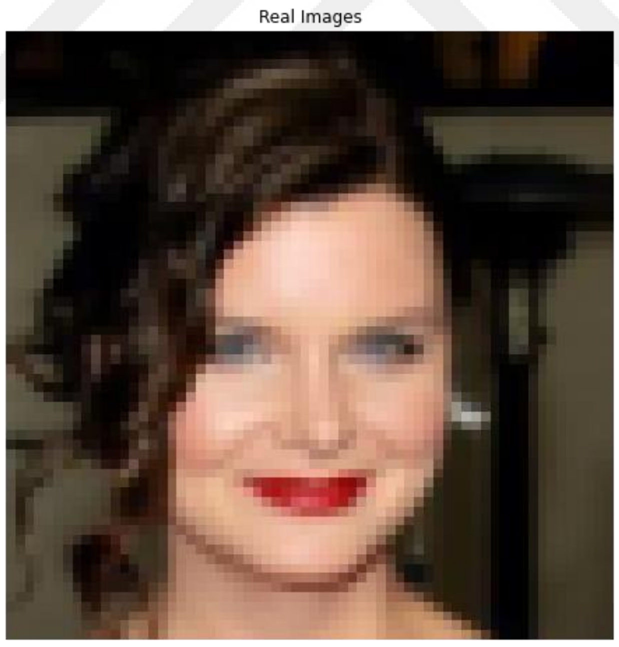

Figure 6.19 : Input Image Example with the Batch Size of 1 (CelebA).

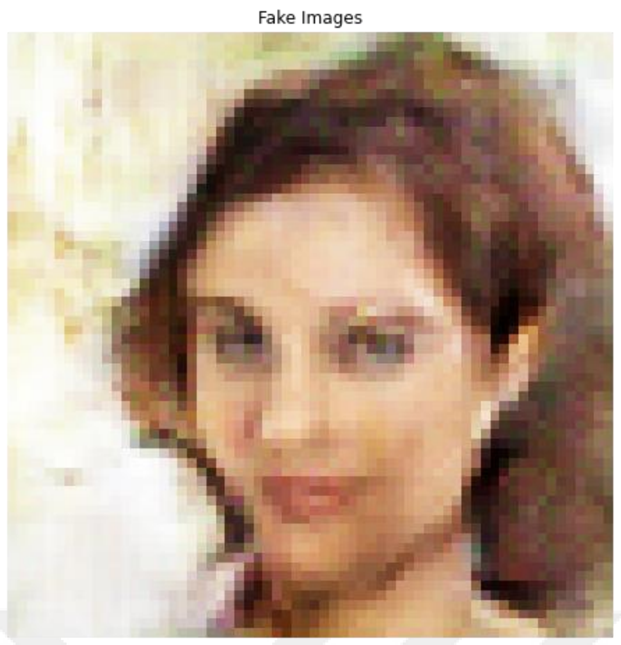

Figure 6.20 : Generated Image with the Batch Size of 1 (CelebA).

Plotting of the generator's training loss curve with the batch size of 1 is in the Figure 6.21.

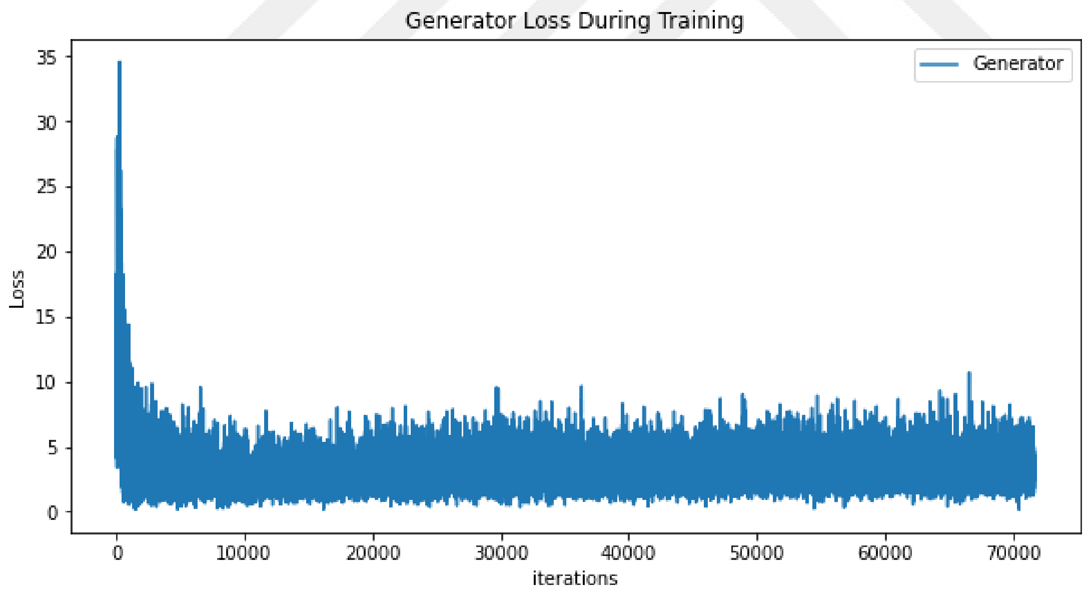

Figure 6.21 : Generator Loss for Image Generation with the Batch Size of 1 (DCGAN).

Plotting of the discriminator's training loss curve with the batch size of 1 is in the Figure 6.22.

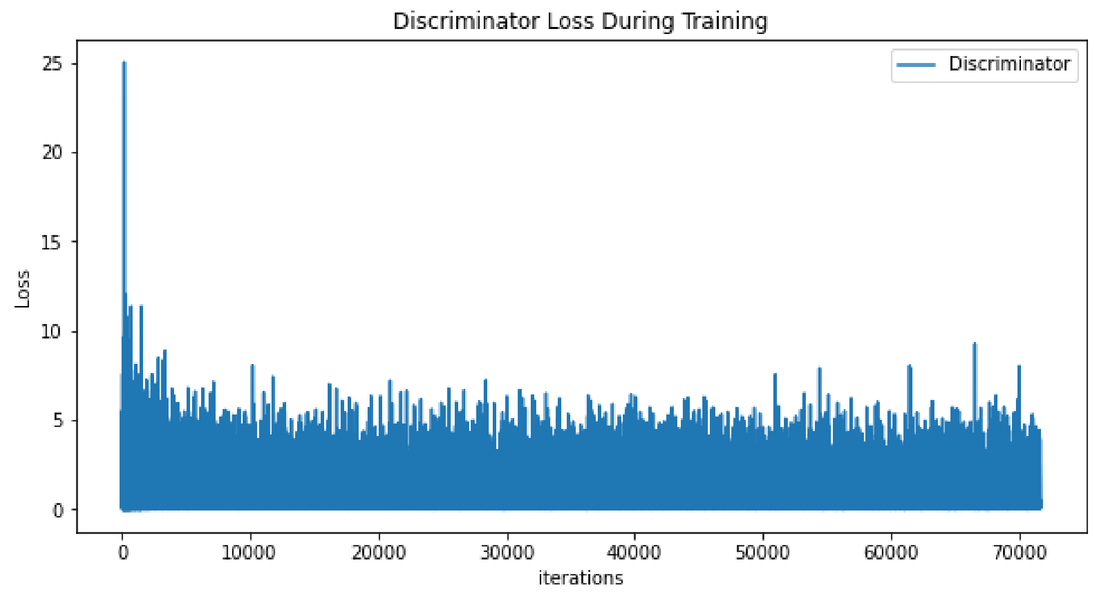

Figure 6.22 : Discriminator Loss for Image Generation with the Batch Size of 1 (DCGAN).

As it was shown above training with the batch size of one compared to training with the batch size of four, gives the best result for generating fake images with using DCGAN architecture as a model [21]. This statement can also be understood by looking at the discriminator's loss curves being more stable in training with the batch size of 1.

6.2.2 BigGAN

For BigGAN [24], a pre-trained model was used to generate fake images. Therefore, there are no input images and loss functions for this part.

Generated output images are shown below in the following Figure 6.23, Figure 6.24, and Figure 6.25. The network was trained by using images from the ImageNet dataset.

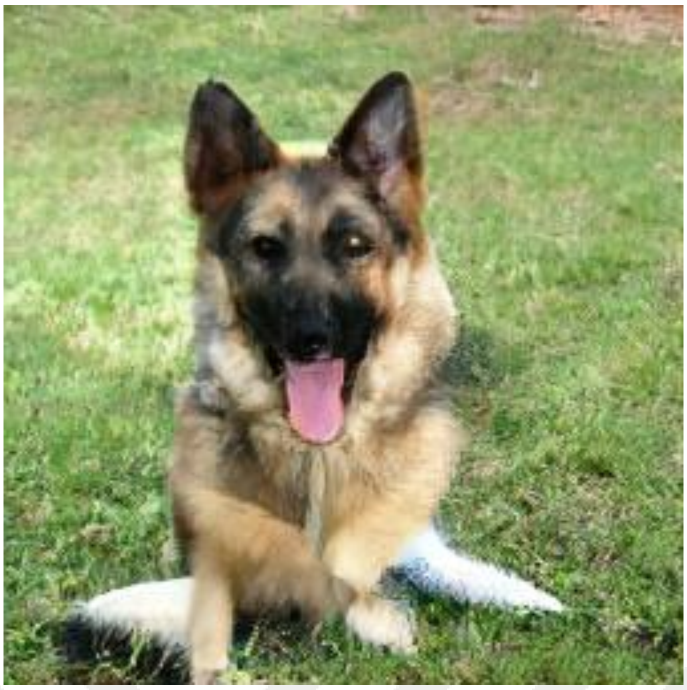

Figure 6.23 : First Example for Images Generated by BigGAN.

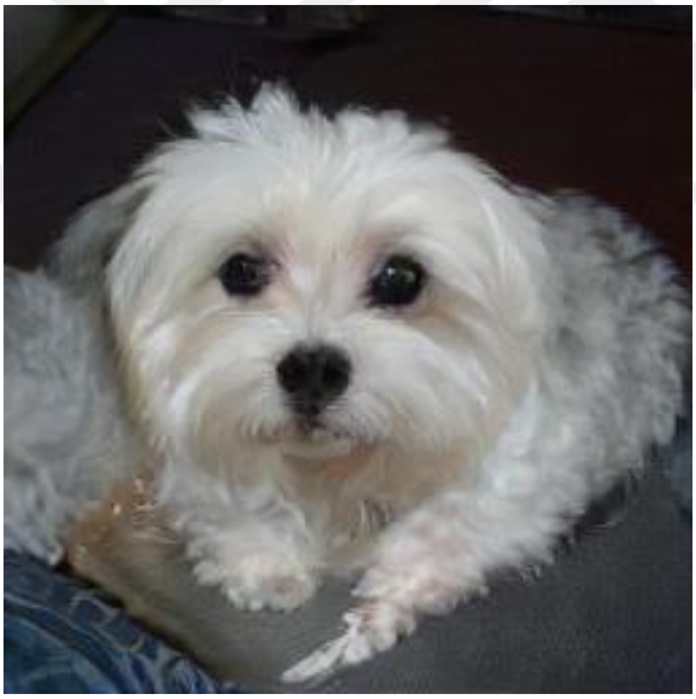

Figure 6.24 : Second Example for Images Generated by BigGAN.

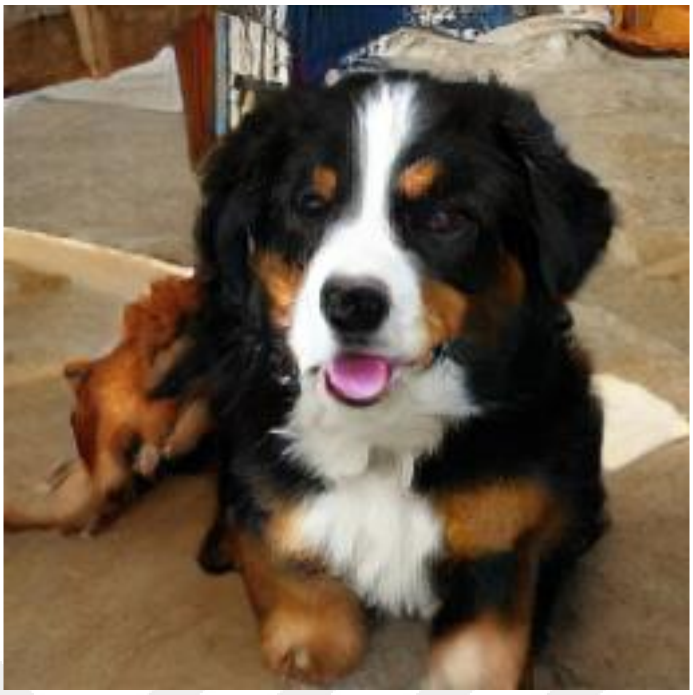

Figure 6.25 : Third Example for Images Generated by BigGAN.

As it was shown above BigGAN architecture [24] generated more realistic images than DCGAN architecture [21].

6.3 Image Denoising

For image denoising two different methods were used. The first one is an updated form of the DCGAN [21] architecture, trained to generate a denoised version of the input images. The second one is Non-local Means (NLM) Denoising algorithm, used to clean the noises inside images.

The noise added to the input images is salt and pepper noise. Salt and pepper noise replace some of the random pixel values in all the channels of the image with a 1 or 0. Salt noise and pepper noise ratio was kept equal. The noise density was chosen as 0.05. This value shows that the added noise effects only 5% of the pixel values of the image.

6.3.1 Updated dcgan architecture

Input images were used in neural network training with two images per batch in one iteration.

In Figure 6.26 the first examples of ground truth images and first examples of noisy input images have been given. In Figure 6.27, denoised images with GAN for every

7000 iteration have been given. Denoised images with GAN have been created from the noisy input images in Figure 6.26.

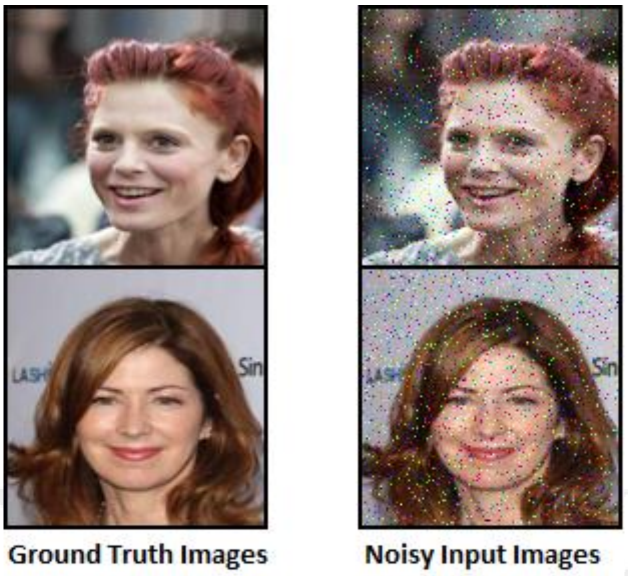

Figure 6.26 : First Example of Ground Truth Images and Noisy Input Images.

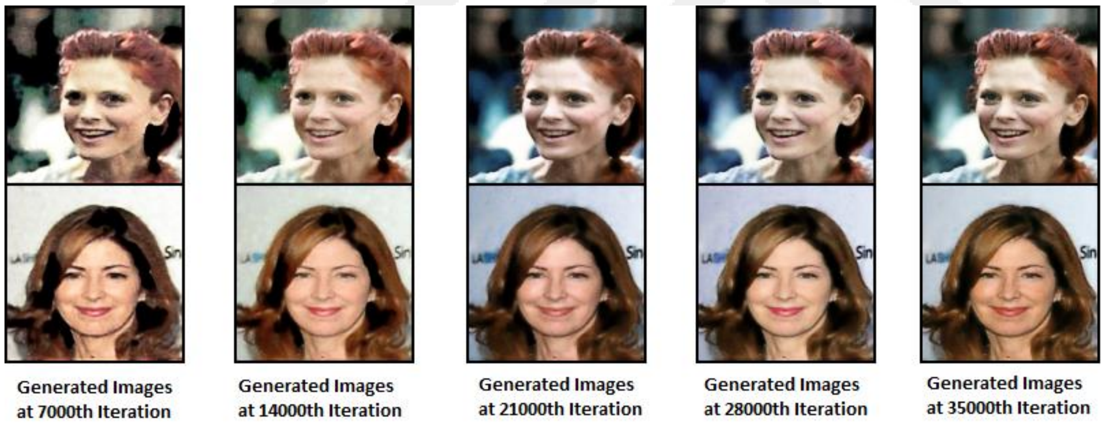
Generated Images at 7000th Iteration
Generated Images at 14000th Iteration
Generated Images at 21000th Iteration
Generated Images at 28000th Iteration
Generated Images at 35000th Iteration

Figure 6.27 : First Example of Denoised Images with GAN for every 7000 Iteration.

In Figure 6.28 the second examples of ground truth images and second examples of noisy input images have been given. In Figure 6.29, denoised images with GAN for every 7000 iteration have been given. Denoised images with GAN have been created from the noisy input images in Figure 6.28.

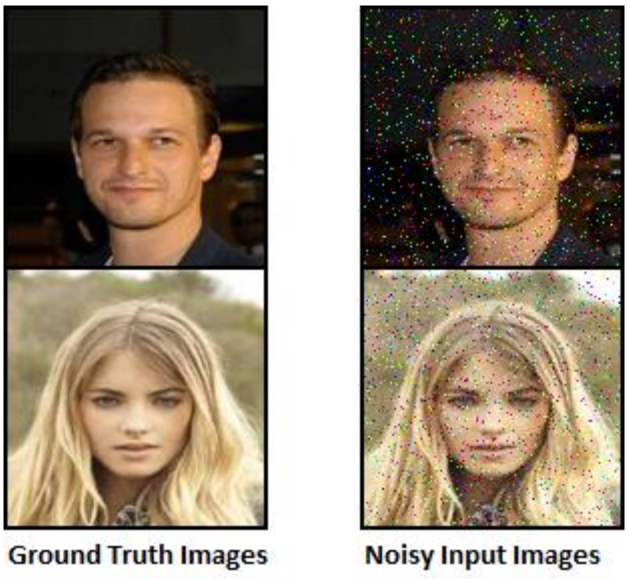

Figure 6.28 : Second Example of Ground Truth Images and Noisy Input Images.

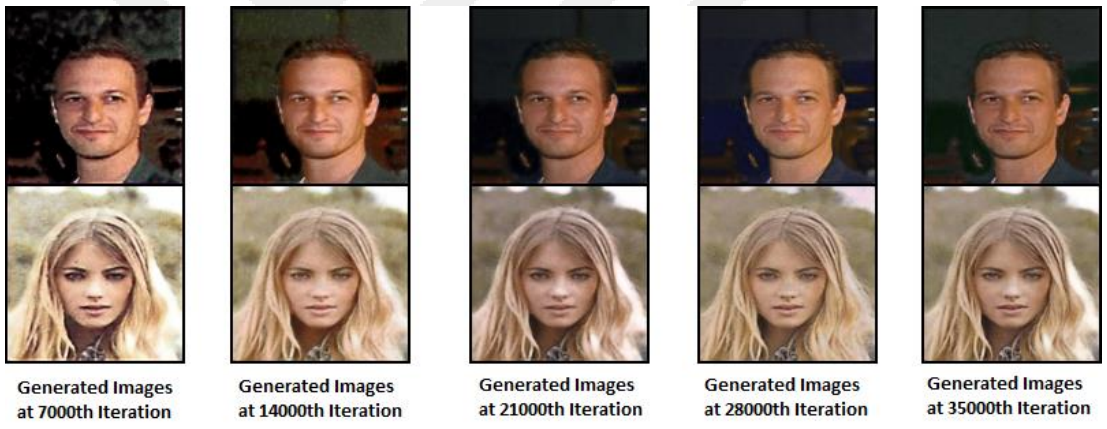
Generated Images at 7000th Iteration
Generated Images at 14000th Iteration
Generated Images at 21000th Iteration
Generated Images at 28000th Iteration
Generated Images at 35000th Iteration

Figure 6.29 : Second Example of Denoised Images with GAN for every 7000 Iteration.

In Figure 6.30 the third examples of ground truth images and third examples of noisy input images have been given. In Figure 6.31, denoised images with GAN for every 7000 iteration have been given. Denoised images with GAN have been created from the noisy input images in Figure 6.30.

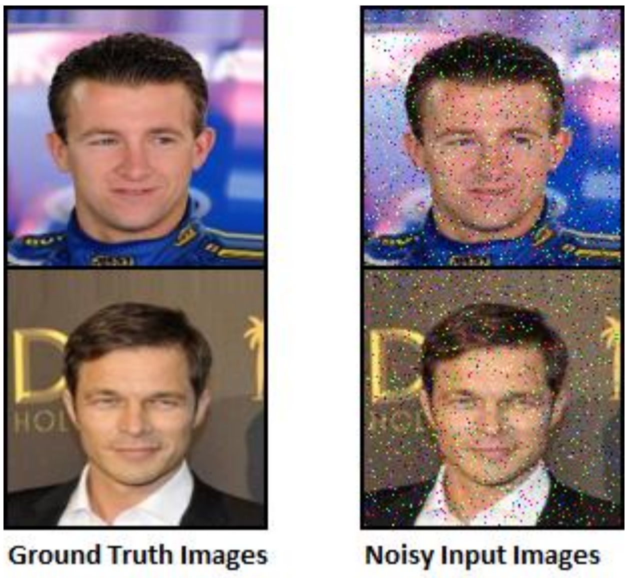

Figure 6.30 : Third Example of Ground Truth Images and Noisy Input Images.

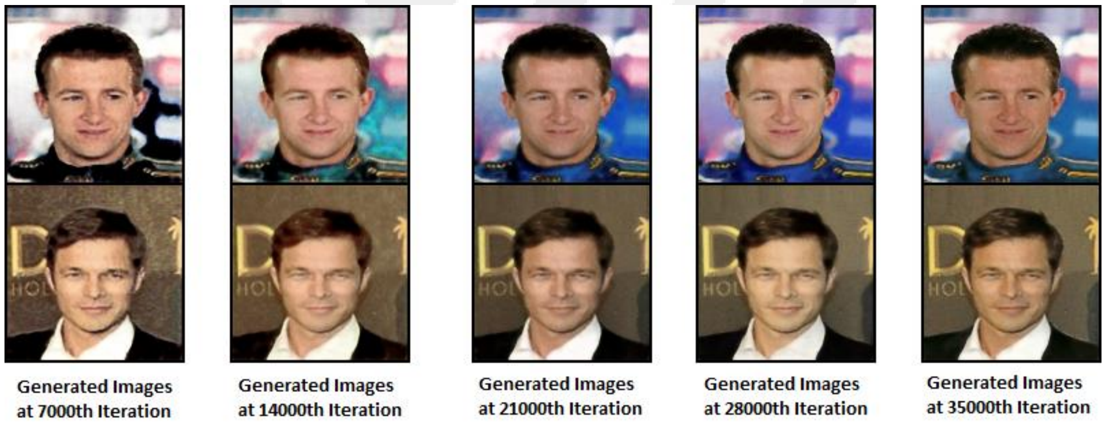
Generated Images at 7000th Iteration
Generated Images at 14000th Iteration
Generated Images at 21000th Iteration
Generated Images at 28000th Iteration
Generated Images at 35000th Iteration

Figure 6.31 : Third Example of Denoised Images with GAN for every 7000 Iteration.

From the images above, it can be observed that the results get better as the training continues.

Plotting of the generator's training loss curve is in the Figure 6.32.

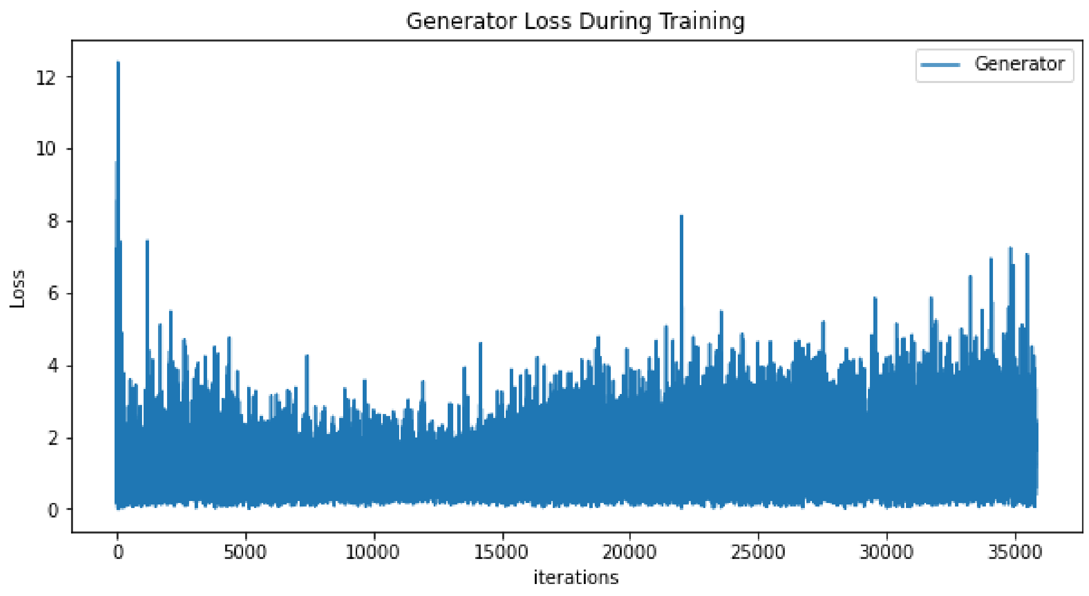

Figure 6.32 : Generator Loss for Image Denoising.

Plotting of the discriminator’s training loss curve is in the Figure 6.33.

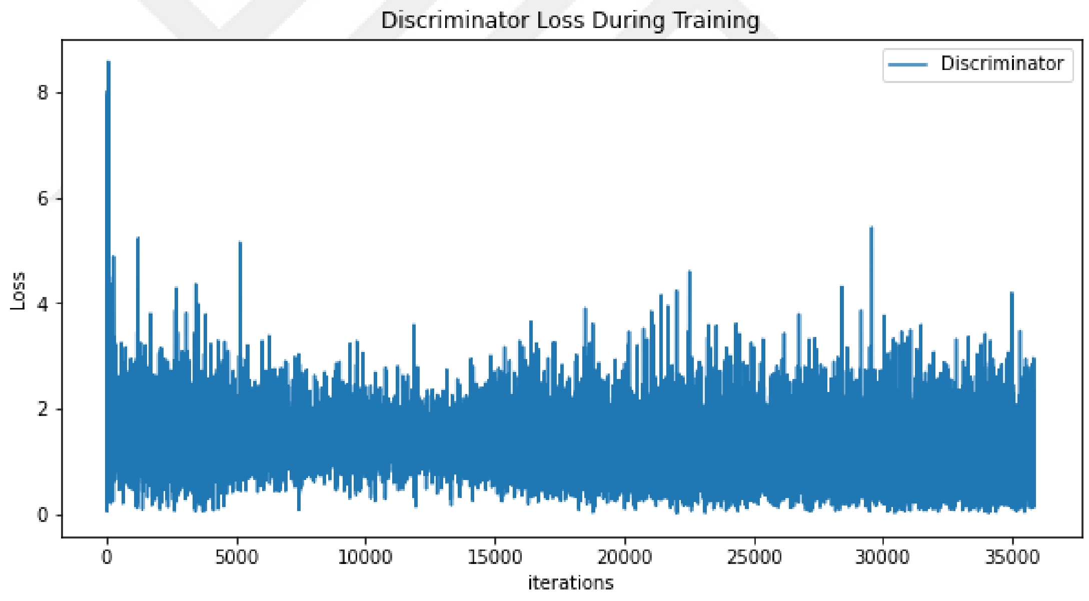

Figure 6.33 : Discriminator Loss for Image Denoising.

As it was shown above as the training progresses to the later stages, the denoised images with GAN looked more plausible. This statement can also be understood by looking at the discriminator’s and generator’s loss curves being stable in the training progress. This stability shows convergence of GAN has been prevented.

Peak signal-to-noise ratio (PSNR) values of the denoised images with GAN and noisy input images that can be seen in the Figure 6.26, Figure 6.27, Figure 6.28, Figure 6.29, Figure 6.30 and Figure 6.31 compared to the ground truth images are in the Table 6.1.

Table 6.1 : PSNR Values of the Denoised Images with GAN.

Example No	Denoised Images with GAN/Ground-truth Image	Input Image/Ground-truth Image
1	29.92 dB	40.86 dB
2	28.44 dB	41.26 dB
3	29.32 dB	41.15 dB

It can be seen from Table 6.1 that the PSNR values between the denoised images with GAN and the ground truth images are lower than the PSNR values between the noisy input images and ground truth images. Even though the results from the GAN show lower PSNR values, denoised images with GAN look more appealing than the noisy input images.

From there, it can be understood that evaluation of the problems like super-resolution and image denoising can be done by human perception in a better way than the generator and discriminator loss function results and the PSNR values between the denoised images and the ground truth images.

6.3.2 Non-local means denoising

This is an image processing technique to remove the noises in the noisy images. Therefore, there are no loss functions for this part.

In Figure 6.34 the first examples of ground truth images, first examples of noisy input images and the denoised images with non-local means denoising algorithm have been given. In Figure 6.35, denoised images with NLM and denoised images with GAN have been given. Denoised images with NLM and denoised images with GAN have been created from the noisy input images in Figure 6.34.

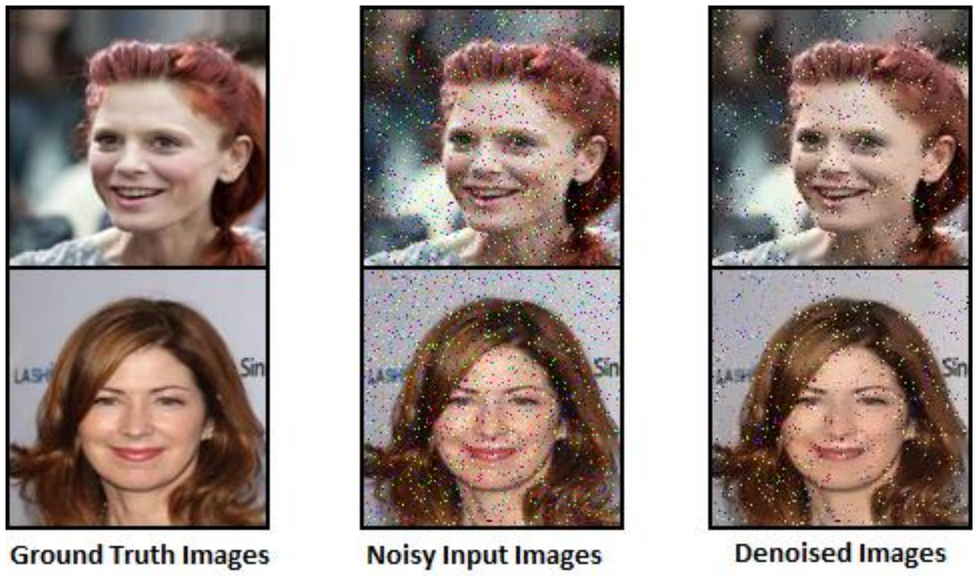

Figure 6.34 : First Example of Ground Truth Images, Noisy Input Images and Denoised Images with NLM.

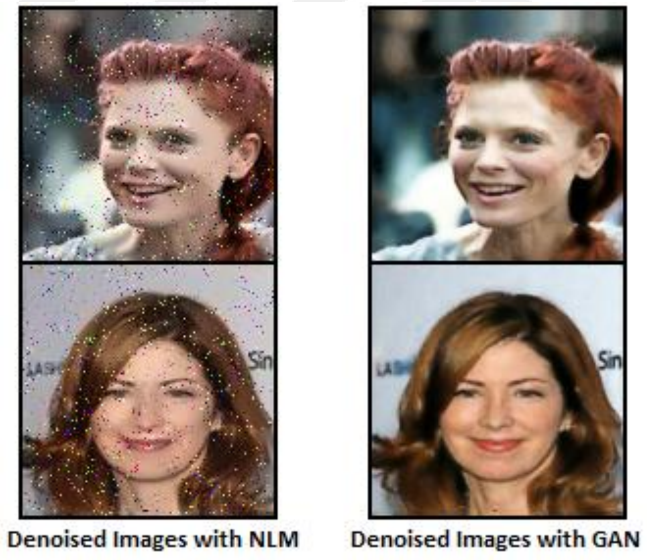

Figure 6.35 : First Example of Denoised Images with NLM and Denoised Images with GAN.

In Figure 6.36 the second examples of ground truth images, second examples of noisy input images and the denoised images with non-local means denoising algorithm have been given. In Figure 6.37, denoised images with NLM and denoised images with GAN have been given. Denoised images with NLM and denoised images with GAN have been created from the noisy input images in Figure 6.36.

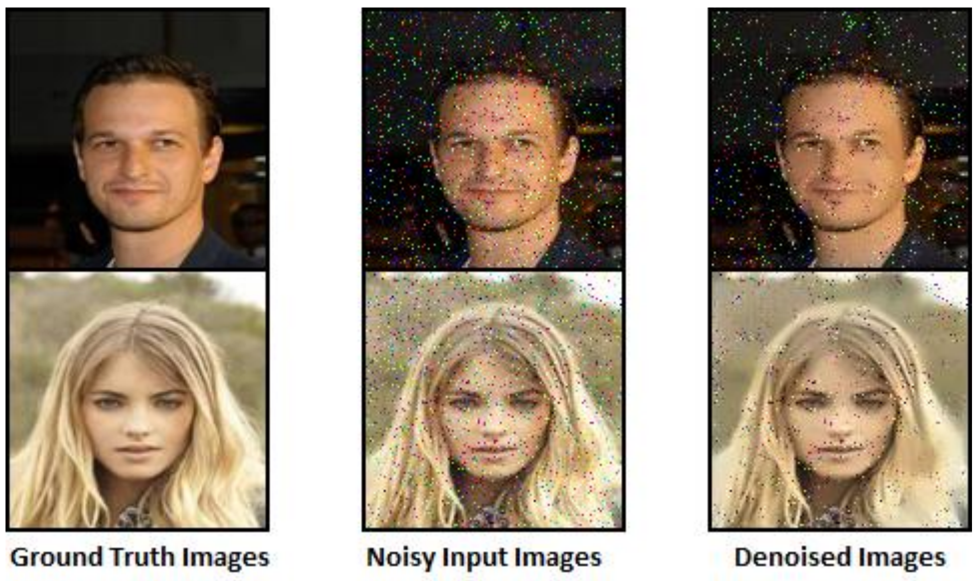

Figure 6.36 : Second Example of Ground Truth Images, Noisy Input Images and Denoised Images with NLM.

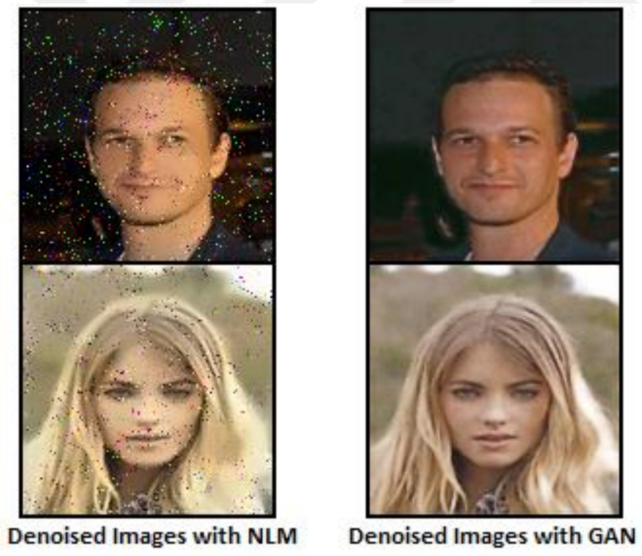

Figure 6.37 : Second Example of Denoised Images with NLM and Denoised Images with GAN.

In Figure 6.38 the third examples of ground truth images, third examples of noisy input images and the denoised images with non-local means denoising algorithm have been given. In Figure 6.39, denoised images with NLM and denoised images with GAN have been given. Denoised images with NLM and denoised images with GAN have been created from the noisy input images in Figure 6.38.

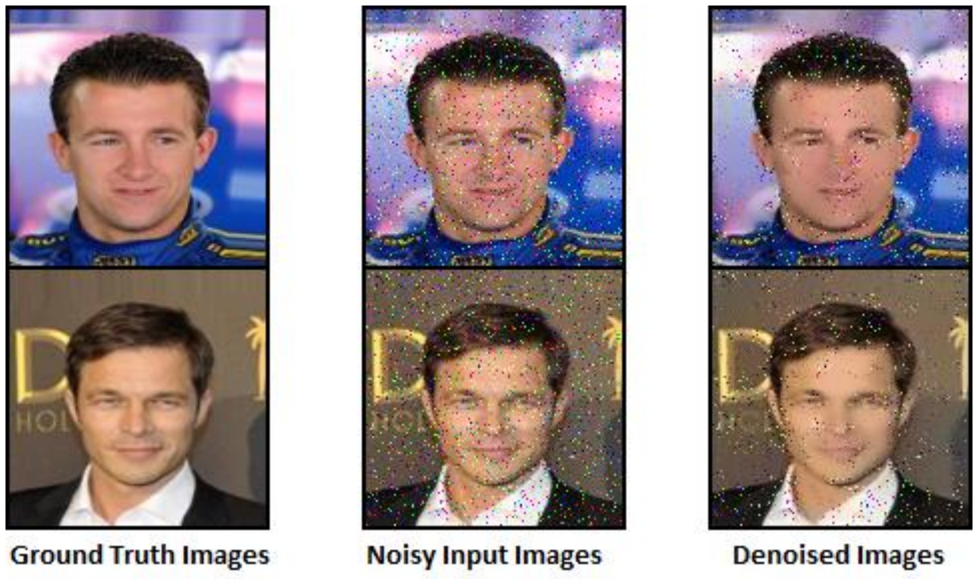

Figure 6.38 : Third Example of Ground Truth Images, Noisy Input Images and Denoised Images with NLM.

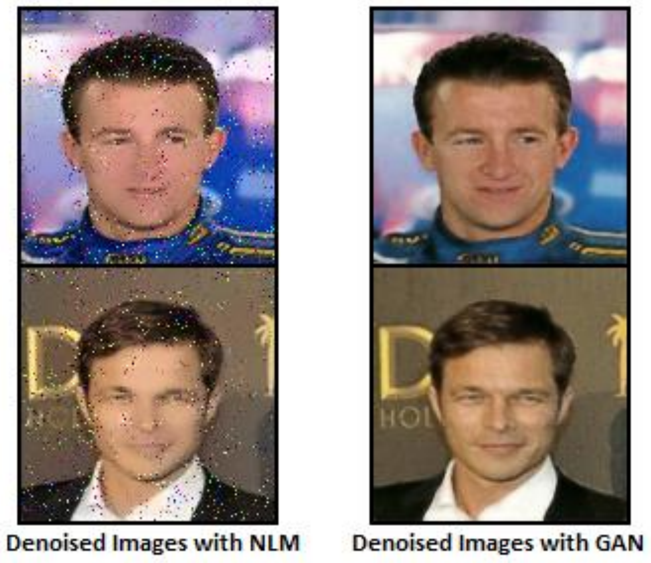

Figure 6.39 : Third Example of Denoised Images with NLM and Denoised Images with GAN.

As it was shown above in the figures, an updated form of the DCGAN [21] architecture denoised the images better than the non-local means denoising. As for the result it can be understood that GAN performed better than the image processing technique.

Peak signal-to-noise ratio (PSNR) values of the denoised images with NLM and noisy input images that can be seen in the Figure 6.34, Figure 6.35, Figure 6.36, Figure 6.37, Figure 6.38 and Figure 6.39 compared to the ground truth images are in the Table 6.2.

Table 6.2 : PSNR Values of the Denoised Images with Non-local Means Denoising.

Example No	Denoised Image with NLM/Ground-truth Image	Input Image/Ground-truth Image
1	33.43 dB	40.86 dB
2	33.26 dB	41.26 dB
3	33.38 dB	41.15 dB

It can be seen from Table 6.2 that the PSNR values between the denoised images with NLM and the ground truth images are lower than the PSNR values between the noisy input images and ground truth images. Even though the results from the non-local means denoising algorithm show lower PSNR values, denoised images with non-local means denoising algorithm look more appealing than the noisy input images.

As it was shown above in the Table 6.1 and Table 6.2, PSNR values between the denoised images with GAN and the ground truth images are lower than the PSNR values between the denoised images with non-local means denoising and the ground truth images. Even though the results from the GAN show lower PSNR values in contrast to the image processing method, denoised images with GAN look more appealing than the denoised images with non-local means denoising algorithm. Therefore, the PSNR value is not a feasible metric to make a comparison with different methods while one of these methods is GAN.

7. CONCLUSION

This thesis demonstrated that generative adversarial network architectures can effectively tackle important computer vision problems. These problems can vary from generating fake images to super-resolving images and denoising of noisy images. This thesis contains some of the popular GAN architectures in computer vision applications, but there are other GAN architectures which are used in other fields of machine learning. Image to image translation, text to image synthesis, text to speech translation etc. can be given as possible examples. Hence, it can be understood that GAN is not only used in the computer vision applications but also in natural language processing (NLP).

In this thesis, we studied three different computer vision applications which use generative adversarial networks for solution. The results indicate that GANs can be very effectively used for these particular problems. Before GAN there were other architectures which used generative modelling as well, but with the founding of GAN [20] those architectures that used generative modelling to solve the computer vision problems became pretty much insufficient. Also, some of the image processing techniques that were used to solve computer vision problems also fell out of use.

From the practical standpoint, recent GAN architectures with the improved optimization techniques are easy to train, and the results are getting more accurate. The training, validation and testing parts are almost the same, the only differences are the loss functions and the optimizers. The procedure for making the dataset ready for training differs for each problem. Different pre-processing techniques and normalization techniques are used on those datasets. Also, the GAN part looks the same with two distinct networks, one being the generator the other one being the discriminator. Although these two architectures do the same job every time, where the generator tries to generate fake images and the discriminator tries to distinguish the generated fake images from the real images, these architectures get changed from one problem to another to tackle the particular characteristics of the problem.

REFERENCES

[1] **K. P. Murphy** (2013) *Machine Learning: A Probabilistic Perspective 1st edition*. Moorpark, CA: Cram101.

[2] **Rodriguez, J.** (2017). Understanding Semi-supervised Learning. *Medium.* Retrieved September 20, 2020, from https://medium.com/@jrodthoughts/understanding-semi-supervised-learning-a6437c070c87

[3] ***What is NumPy?.*** *(n.d.). Retrieved October 18, 2020, from https://numpy.org/doc/stable/user/whatisnumpy.html*

[4] ***Overview of GAN Structure | Generative Adversarial Networks.*** *(n.d.). Retrieved October 3, 2020, from https://developers.google.com/machine-learning/gan/gan_structure*

[5] ***Explore the Possibilities of Generative Modeling.*** *(2018, June 22). Retrieved October 3, 2020, from https://software.intel.com/content/www/us/en/develop/articles/explore-the-possibilities-of-generative-modeling.html*

[6] ***The Discriminator | Generative Adversarial Networks.*** *(n.d.). Retrieved October 24, 2020, from https://developers.google.com/machine-learning/gan/discriminator*

[7] ***The Generator | Generative Adversarial Networks | Google Developers.*** *(n.d.). Retrieved October 30, 2020, from https://developers.google.com/machine-learning/gan/generator*

[8] ***GAN Training | Generative Adversarial Networks | Google Developers.*** *(n.d.). Retrieved October 31, 2020, from https://developers.google.com/machine-learning/gan/training*

[9] ***Competition.*** *(n.d.). Retrieved October 11, 2020, from https://competitions.codalab.org/competitions/22220#learn_the_details*

[10] **Ji, X., Cao, Y., Tai, Y., Wang, C., Li, J., & Huang, F.** (2020). Real-World Super-Resolution via Kernel Estimation and Noise Injection. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW).* doi:10.1109/cvprw50498.2020.00241

[11] **Lugmayr, A., Danelljan, M., Timofte, R., Ahn, N., Bai, D., Cai, J., ... Zou, X.** (2020). NTIRE 2020 Challenge on Real-World Image Super-Resolution: Methods and Results. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW).* doi:10.1109/cvprw50498.2020.00255

[12] **Sefi Bell-Kligler, Assaf Shocher, and Michal Irani.** (2019) Blind super-resolution kernel estimation using an internal-gan. *In NeurIPS*, pages 284–293, 2019.

[13] **Wang, X., Yu, K., Wu, S., Gu, J., Liu, Y., Dong, C., . . . Loy, C. C.** (2019). ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks. *Lecture Notes in Computer Science Computer Vision – ECCV 2018 Workshops*, 63-79. doi:10.1007/978-3-030-11021-5_5

[14] **Liu, Z., Luo, P., Wang, X., & Tang, X.** (2015). Deep Learning Face Attributes in the Wild. *In Proceedings of International Conference on Computer Vision (ICCV).*

[15] **Ledig, C., Theis, L., Huszar, F., Caballero, J., Cunningham, A., Acosta, A., . . . Shi, W.** (2017). Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* doi:10.1109/cvpr.2017.19

[16] **He, K., Zhang, X., Ren, S., & Sun, J.** (2016). Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* doi:10.1109/cvpr.2016.90

[17] **Shi, W., Caballero, J., Huszar, F., Totz, J., Aitken, A. P., Bishop, R., . . . Wang, Z.** (2016). Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* doi:10.1109/cvpr.2016.207

[18] ***L1Loss.*** *(n.d.). Retrieved November 4, 2020, from https://pytorch.org/docs/stable/generated/torch.nn.L1Loss.html*

[19] ***BCEWithLogitsLoss.*** *(n.d.). Retrieved November 4, 2020, from https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html*

[20] **Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., . . . Bengio, Y.** (2020). Generative adversarial networks. *Communications of the ACM, 63*(11), 139-144. doi:10.1145/3422622

[21] **Radford, A., Metz, L., & Chintala, S.** (2016). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *CoRR, abs/1511.06434.*

[22] **Liu, M., & Tuzel, O.** (2016). Coupled Generative Adversarial Networks. *NIPS.*

[23] **Karras, T., Aila, T., Laine, S., & Lehtinen, J.** (2018). Progressive Growing of GANs for Improved Quality, Stability, and Variation. *ArXiv, abs/1710.10196.*

[24] **Brock, A., Donahue, J., & Simonyan, K.** (2019). Large Scale GAN Training for High Fidelity Natural Image Synthesis. *ArXiv, abs/1809.11096.*

[25] **Zhang, H., Goodfellow, I.J., Metaxas, D., & Odena, A.** (2019). Self-Attention Generative Adversarial Networks. *ArXiv, abs/1805.08318.*

[26] ***Average pooling layer - MATLAB.*** *(n.d.). MathWorks. Retrieved November 18, 2020, from*

https://www.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.averagepooling2dlayer.html

[27] ***BCELoss.*** *(n.d.). Retrieved November 18, 2020, from https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html*

[28] **Chen, J., Chen, J., Chao, H., & Yang, M.** (2018). Image Blind Denoising with Generative Adversarial Network Based Noise Modeling. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition.* doi:10.1109/cvpr.2018.00333

[29] **Alsaiari, A., Rustagi, R., Alhakamy, A., Thomas, M. M., & Forbes, A. G.** (2019). Image Denoising Using A Generative Adversarial Network. *2019 IEEE 2nd International Conference on Information and Computer Technologies (ICICT).* doi:10.1109/infoct.2019.8710893

[30] **Yan, Q., Wang, W.** (2017). DCGANs for image super-resolution, denoising and debluring.

[31] **Saha, A.** (2020). Python: Peak Signal-to-Noise Ratio (PSNR). Retrieved December 12, 2020, from https://www.geeksforgeeks.org/python-peak-signal-to-noise-ratio-psnr/

CURRICULUM VITAE

Name Surname	**: Semih Örnek**
Place and Date of Birth	**: Eskişehir 23.07.1995**
E-Mail	**:**

EDUCATION **:**

- **B.Sc.** : 2018, Anadolu University, Faculty of Engineering, Electrical - Electronics Engineering

PROFESSIONAL EXPERIENCE AND REWARDS:

- February 2021 – Present Software Engineer at Arçelik A.Ş
- April 2019 – January 2021 Project Engineer at Arçelik A.Ş.
- June 2018 –September 2018 Embedded Software Engineer at Piton Ar-Ge ve Yazılım Evi
- 2018 Graduated as Honor Student at Anadolu University