ADAPTIVE LIGHTWEIGHT LICENSE PLATE IMAGE RECOVERY USING DEEP LEARNING BASED ON GENERATIVE ADVERSARIAL NETWORK

by

Wuttinan Sereethavekul

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of Doctor of Engineering in Microelectronics and Embedded Systems

Examination Committee:	Dr. Mongkol Ekpanyapong (Chairperson)
	Prof. Matthew N. Dailey
	Prof. Huynh Trung Luong
External Examiner:	Prof. Fausto Giunchiglia
	Faculty of Science
	University of Trento
	Trento, Italy
Nationality:	Thai
Previous Degree:	Master of Engineering in Microelectronics and Embedded Systems
	Asian Institute of Technology, Thailand
Scholarship Donor:	Royal Thai Government - AIT Fellowship

Asian Institute of Technology
School of Engineering and Technology
Thailand
May 2024

AUTHOR'S DECLARATION

I, Wuttinan Sereethavekul, declare that the research work carried out for this thesis was in accordance with the regulations of the Asian Institute of Technology. The work presented in it are my own and has been generated by me as the result of my own original research, and if external sources were used, such sources have been cited. It is original and has not been submitted to any other institution to obtain another degree or qualification. This is a true copy of the thesis including final revisions.

Date: January 16, 2024

Name: Wuttinan Sereethavekul

Signature: 

ACKNOWLEDGEMENTS

This research fund was supported in part by TSRI (Thailand Science Research and Innovation) project number RDG6250036. Furthermore, all of the hard work in this study would not have been completed without a support from Dr. Mongkol Ekpanyapong and his team at AI center, Asian Institute of Technology. They provided me with all datasets used in this work. A research fund supported by my institute (AIT) in purchasing research equipment, i.e., a computer machine and peripherals. Including thesis committees who gave me excellent advice and guidance. Also, my AIT friend gave me suggestions during the study process. Lastly, I would like to give special thanks to my family, who always supported me on living expenses throughout this entire study.

ABSTRACT

Many Convolutional Neural Networks (CNNs) methods have already surpassed traditional approaches to image restoration tasks. Those CNNs models were usually designed to enhance single tasks such as an image resolution (super-resolution) or image denoising, but we came up with unconventional goals, that is, multiple recovery tasks from a single network design. Although the Transformer design has recently gained attention in image recovery tasks, they are too slow. In order to work with license plate images from a traffic camera stream, the system has to be responsive. So, we proposed a fast and lightweight deep learning-based data recovery system using a Generative Adversarial Network (GAN) principle named License Plate Recovery GAN (LPRGAN). The design has a proposed encoder-decoder style inspired by an autoencoder aided by dual classification networks. This style suits problem-characteristic learning because strong contextual information is retrieved from the down-scaled representations. This proposed system has three main features such as identifying a problem, data recovery, and fail-safe mechanism. The core of system is a data recovery unit (LPRGAN), is used to recover license plate images from multiple degraded input images. Most existing image restoration systems do not have self-awareness, leading to an inefficiency problem. Unlike existing works, this system has anomaly detection and will only process on a degraded input, reducing workload overhead, improving efficiency and a fail-safe feature that prevents an unexpected bad output. Hence, the proposed algorithm requires less resource to deploy on a low-power machine such as edge computing devices, opening up new possibilities in on-device computing. Our proposed research can recover several degraded problems up to 720p resolution at 15 frames per second on a single graphic card, 256x128 resolution at 17 frames per second on a CPU-only workstation machine, or 7 frames per second on an ultra-low-power tablet PC.

Keywords: Data Recovery, Deep Learning, Generative Adversarial Networks, Image and Video Recovery, Machine Learning, Neural Networks, and Video Streaming.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABR = Adaptive Bitrate Streaming

AI = Artificial Intelligence

Array = An enumerated collection of identical entities

AVC/H.264 = Advanced Video Coding

Bit = A binary digit having a value of 0 or 1

CBP = Coded Block Pattern

CBR = Constant Bitrate

CG = Computer Graphic

CGAN = Conditional Generative Adversarial Networks

CNN = Convolutional Neural Network

CPU = Central Processing Unit

CQP = Constant Quantization Parameter

CRF = Constant Rate Factor

CTU(HEVC)/

Macroblock(AVC) = Coding Tree Unit

CycleGAN = Cycle Generative Adversarial Networks

DCGAN = Deep Convolution Generative Adversarial Networks

DCT = Discrete Cosine Transform

DNN = Deep Neural Network

FHD = Full High Definition

FLOPS = Floating Point Operations Per Second

FPS = Frames Per Second

GAN = Generative Adversarial Networks

GPGPU = General-Purpose Graphic Processing Unit

GPU = Graphic Processing Unit

HEVC/H.265 = High Efficiency Video Coding

HR = High-Resolution

HQ = High Quality

I/P/B Frame = Intra-coded/Predicted/Bidirectional Predicted Frame

LPRGAN = License Plate Recovery GAN

LQ = Low Quality

LR = Low-Resolution

MSE = Mean Squared Error

ML = Machine Learning

MB = Megabyte

OCR = Optical Character Recognition

PIP = Python Package Manager

PSNR = Peak Signal-to-Noise Ratio

QoE = Quality of Experience

QP = Quantization Parameter

ReLU = Rectified Linear Unit

RMSE = Root Mean Squared Error

SCC = Spatial Correlation Coefficient

Spatial = A space that one image consist of pixel values, coordinates, intensity, gradient and resolution

SR = Super-Resolution

SRCNN = Super-Resolution Convolutional Neural Network

SSIM = Structural Similarity Index

Temporal = A time that video consists of image frame sequences, correlations between the images that determines the dynamic changes of the object

U-Net = U-Net Convolutional Neural Network Architecture

UHD/4K = Ultra High Definition

UQI = Universal Image Quality Index

VIF = Visual Information Fidelity

VQM = Video Quality Metric

VMAF = Video Multi-Method Assessment Fusion

CHAPTER 1
INTRODUCTION

1.1 Overview

Traffic cameras are now becoming essential tools in part of transportation systems. They are used to monitor traffic activity and accidents or to detect illegal vehicles on the road. These cameras help in traffic police workforce reduction. Not only that, a traffic camera can be deployed in very remote areas where traffic police are hard to reach. The traffic monitoring system can provide a full country-wide road area coverage. This monitoring is a worldwide standard practice to enhance road security and safety. The most important aspect of traffic monitoring is vehicle license plate reading, such as in road accidents or traffic violation vehicles so that police officers can identify them. So this study provides a new approach to help restoring a degraded license plate image using a deep learning technique.

1.2 Problem Statement

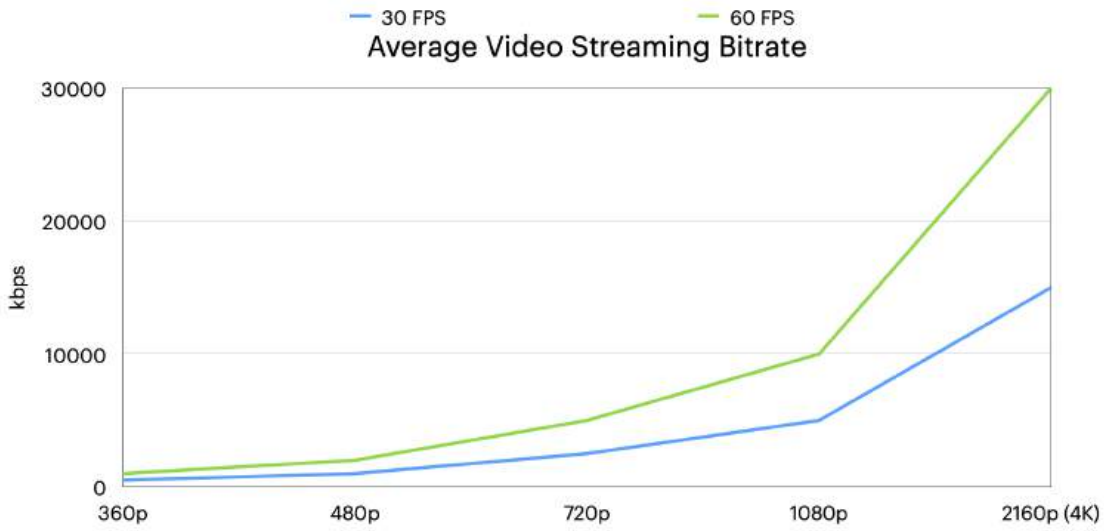
There are many shortcomings in reading a license plate. Examples are occasionally corrupted data within a streaming frame, low light area, slow shutter camera speed that is not fast enough to track a plate, or an intention to save disk space by reducing recording bitrate, resulting in low-quality media files. A simple form of data corruption can be seen as a blocky-looking or blocky artifact due to a low bitrate streaming. Fig.1.1 shows a bitrate requirement in a video streaming at different resolutions.

1.3 Objectives

This study aims to develop a new light and fast yet practical license plate image quality recovery, covering low bitrate, low light, motion blur and out-of-focus situations with a single network setup. This research relies on artificial intelligence to overcome those problems. It must be response to cope with real-time stream processing and light enough to deploy on a typical workstation machine to enhance a monitoring potential. In addition, it is designed to handle multiple scenarios from a single network setup. Hence, a training and using this work are effortlessly.

Figure 1.1

Bitrate Required for Video Streaming at Different Resolution



1.4 Limitations and Scope

This work features a GAN, contains a modified generator and discriminator, resulting in just under a million parameters (for 256x128 image recovery). The image recognition and image recovery systems are also introduced into the system, working together to detect and recover a degraded input, greatly improve system efficiency, instead of a simple barebone system. However, there are still some limitations that need to be concerned.

- Output dimension size is a fixed size (256x128x3). It cannot be used to create other larger or smaller sizes unless reconfiguration and retraining are required.
- Output quality is limited by the training dataset. It cannot create a better quality than the original dataset.
- Some small detail areas in the generated image could be less sharp, such as the province name in some problem cases. This drawback can be avoided by having a higher training dataset resolution i.e., collecting higher-resolution images or using a super-resolution technique.
- While the proposed model could be used in other countries rather than its origin, but its performance would not be on par. Hence, retraining in a target country is required.

CHAPTER 2
LITERATURE REVIEW

2.1 Related Works

This research has studied a deep learning principle which is a part of AI, to address such problems. It has a unique ability to learn and observe input patterns used for many complex tasks. This technique is suitable for improving low-quality images by learning from high-quality ones. There are many CNNs available that have these capabilities. For example, an autoencoder (Chollet, 2016) is useful when an input has a noise in the image or audio. It is used to remove noise in a signal. An autoencoder also has a convolutional version called a convolutional autoencoder. It has multiple convolution computation layers as part of a network. The U-Net (Ronneberger, Fischer, & Brox, 2015) is a network that shares similarities to an autoencoder but even more complex layers. It consists of two parts, a contracting path, and an expansive path. There are many GAN (Langr & Bok, 2019) variations such as Deep Convolutional GAN (*DCGAN. Deep Convolutional Generative Adversarial Network.*, 2023), Conditional GAN (*CGAN. Conditional GAN.*, 2022), and CycleGAN (Zhu, Park, Isola, & Efros, 2017). The DCGAN is usually used to generate a new image from random input data (normal distribution). A downside of DCGAN is that its output cannot be controlled, so it is impossible to specify an output appearance or class. This reason is why CGAN offers more control over DCGAN. The CycleGAN is used to swap between two input domains. A GAN relies heavily on CNNs because many CNNs models have posed a potential for image recovery and enhancement. They learn from a pattern from big datasets. Most CNNs setups usually have either an encoder-decoder style or a single-size style. In the first case, it utilizes a down-sampling method to map an input to a lower-resolution representation and then applies a reverse operation (up-sampling) to map to an original resolution. This operation is a good way to learn input context by down-sampling resolution, but a downside is that the fine spatial details are lost in the process. Thus, an output usually has lower details when compared to an original, making this style a lossy process. In the latter case, a single-scaling style utilizes feature processing. It does not contain any down-sampling operation, producing images with more fine details. Nevertheless, this single-scale style is commonly less effective in learning a pattern of contextual information due to a limited representative resolution. So these two examples both have their benefits and drawbacks.

It is a position-sensitive procedure where pixels from two sources need to be matched in a recovery learning process. The first source is a reference image, and the second is a distorted image. A slight shift in pixel position between them is undesired because a distorted pattern must be the only component in that image. Otherwise, a true pattern will be mixed up with a dislocation pixel, making it difficult to learn a degradation pattern for a specific problem. Therefore, both reference and degraded input images have to be perfectly aligned. Then the learning can even further benefit from a large context dataset, i.e., image scaling or problem variation.

Image processing has been developed over the past years, and one that benefits are traffic monitoring. Traffic monitoring involves content transmission from a camera unit to a monitoring unit. An interruption in transmission, such as unavailable bandwidth, can produce poor-quality video transmission. Many have tried to improve content transmission, such as Petrov et al. (Petrov, Kartalov, & Ivanovski, 2009) proposed a technique to reduce a blocking artifact by detecting a blocking artifact in a macroblock (8x8 pixels) and a displaced blocking effect. Then apply a blocking artifact reduction using their proposed filters. They targeted mobile platforms with low bitrate video focused on operation speed. Dar et al. (Dar & Bruckstein, 2015) presented a work that analyses only one slice of low bitrate video compression. This paper benefits from applying a spatio-temporal down-scaling, i.e., reduction of frame rate and frame size, before the compression and a corresponding up-scaling afterward. They left H.264 codec untouched. Their work covers 16x16 macroblock from very low (2 bits/slice) through low (around 30 bits/slice) and up to high (210 bits/slice) bitrate. So they presented that the downsampling video before compression took place is better. Li et al. (Li et al., 2017) proposed a new five layers CNN-based block up-sampling scheme for intra-frame coding. A block can be down-sampled before being compressed by normal intra-coding and then up-sampled to its original resolution. This way differs from the previous hand-craft down and up-sampling because this paper is based on training a CNN. A new CNN structure for up-sampling features the deconvolution of feature maps, multi-scale fusion, and residue learning, making the network compact and efficient. They also designed different networks for the up-sampling of luma and chroma components, respectively, where the chroma up-sampling CNN utilizes the luma information to boost its performance. This scheme is built into HEVC reference software. Resulting in an average 5.5% BD-rate reduction on common test sequences and an average 9.0% BD-rate reduction on ultra-

high definition (UHD) test sequences. Instead of using traditional downsampling, they presented a CNN-based sampling scheme. Lin et al. (Lin, He, & Qing, 2019) proposed an adaptive downsampling-based coding model to improve the low bitrate compression efficiency of high-efficiency video coding (HEVC). They use motion estimation to find the most similar blocks between upscaled Non-Key Frames (NKFs) and associated high-resolution Key Frames (KFs). Then, an adaptive patching-based method is used to warp the low-quality NKF blocks with the high-quality KF blocks. Their experimental results demonstrate significant improvements compared to existing methods but only work on HEVC. Yang et al. (Yang, Xu, Liu, Wang, & Guan, 2018) worked on enhancing low bitrate HEVC video quality. They enhanced the visual quality of HEVC videos on the decoder side. So they proposed a Quality Enhancement Convolutional Neural Network (QE-CNN) method that does not require any encoder modification to achieve quality enhancement for HEVC. In particular, their QE-CNN method learns QE-CNN-I and QE-CNN-P models to reduce the distortion of HEVC I and P/B frames, respectively. This method differs from the existing CNN-based quality enhancement approaches, which only handle intra-coding distortion and are thus unsuitable for P/B frames. They claimed their method validates that the QE-CNN method effectively enhances quality for both I and P/B frames of HEVC videos. These mentioned works feature both non-machine learning and machine learning forms. Despite the benefit of media transmissions that enhance streaming quality, resulting in a better video output quality, they are restricted to a specific video codec.

On the other hand, some previous works tried to improve the image processing aspect. For instance, Zhu et al. (Zhu, Park, Isola, & Efros, 2020) made use of the CycleGAN to do an image-to-image translation ($X \rightarrow Y$) which they called the "pix2pix" project (*Image-to-Image Translation using Pix2Pix.*, 2022). He and his team created this project to swap the texture of two things like, zebra and horse, and swap between two pictures style, such as a photograph and an art style. Although swapping two pictures could benefit some areas, such as swapping a noisy image with a clean one in the de-nosing task, it does not necessarily mean imagery improvement. Guo et al. (Guo, Yan, & Zhang, 2019) presented a way to improve image denoising with additive white Gaussian noise (AWGN) by training a convolutional blind denoising network (CBDNet) with a more realistic noise model and real-world noisy-clean image pairs. Also, they provided an interactive strategy to rectify denoising results conveniently. A noise estimation subnetwork

with asymmetric learning to suppress the underestimation of noise level is embedded into CBDNet. Wang et al. (Wang, Li, & Zhang, 2021) built a blind face restoration system. This Generative Facial Prior (GFP) is incorporated into the face restoration process via spatial feature transform layers, achieving a good balance of realness and fidelity. The GFP-GAN could jointly restore facial details and enhance colors with just a single forward pass. Kupyn et al. (Kupyn, Martyniuk, & Wu, 2019) presented DeblurGAN-v2, a newer version of DeblurGAN that considerably boosts state-of-the-art deblurring performance while being much more flexible and efficient. It was claimed to be faster and better than v1. It is made of GAN with a backend such as Inception ResNet v2. Zamir et al. (Zamir et al., 2020) presented the MIRNet, an image restoration model. A proposed architecture maintains high-resolution representations throughout the entire network and receives information from the low-resolution representations. Existing CNN-based methods usually operate just on full-resolution or low-resolution representations. Although this network packs much functionality, it also contains several modules. They used three Recursive Residual Groups (RRGs), each of which contains two Multi-scale Residual Block (MRBs), and each MRB also contains three streams. Zamir et al. (Zamir, Arora, & Khan, 2022) also proposed an efficient Transformer model for capturing long-range pixel interactions, while remaining applicable to large images. It can restore images on several tasks. Liang et al. (Liang et al., 2021) proposed a SwinIR model for image restoration based on the Swin Transformer. It consists of shallow feature extraction, deep feature extraction, and high-quality image reconstruction. Recently, a biomedical paper utilizing GAN from Zhang et al. (Zhang et al., 2022) presented a method of increasing contrast in CT scanning images for clinical diagnosis. Their MALAR system is based on CycleGAN. It has dual GANs that work on ultra-low-dose-ICM aorta CT (UDCT) and low-dose-ICM aorta CT (LDCT) images. However, this approach outputs DICOM format and does not work on standard RGB images. Wu et al. (Wu et al., 2021) presented an article for tomographic image reconstruction in a sparse-view CT scan. They proposed a Dual-domain Residual-based Optimization NEtwork (DRONE). It consists of three modules for embedding, refinement, and awareness. The results from the embedding and refinement modules in the data and image domains are regularized for optimized image quality in the awareness module, which ensures the consistency between measurements and images with the kernel awareness of compressed sensing. Wu et al. (Wu, Guo, Chen, Wang, & Chen, 2022) also presented a Deep

Embedding-Attention-Refinement (DEAR) network to achieve good images from high sparse-view levels in CT reconstruction tomography imaging. This study was based on the DRONE and released later. DEAR also consists of three modules including deep embedding, deep attention, and deep refinement. The results demonstrate the efficiency of the DEAR in edge preservation and feature recovery in deep tomographic reconstruction.

From the above referenced works, each one of them has its own strength and drawback but none focuses on real-time image processing. Since a live stream license plate recovery task is a time-crucial process. This is where this proposed system gears toward to. Finally, a group of selected novel approaches are evaluated by synthetically and visually benchmarks to set a baseline againts the proposed method. These results are shown in 4.4.

2.2 CUDA and CuDNN

CUDA(*Develop, Optimize and Deploy GPU-Accelerated Apps.*, 2023) is a licensed name of NVIDIA Corporation. It is an advanced computing platform using numerous general proposed processors inside GPU (GPGPU). It is aimed at parallel computing which largely supports popular development languages such as C, C++, Java, and Python. As a result, big performance gains from using CUDA core inside NVIDIA GPU. This helps to offload a workload from the main CPU to GPU, so the CPU could be used for something else. As a drawback, in order to gain a benefit from using CUDA cores, it is required only NVIDIA's GPU. CUDA, however, can be compared to OpenCL from The Khronos Group Inc which is also available on AMD Radeon GPU. CuDNN library is also a part of CUDA which is solely used for accelerating deep learning training.

2.3 Keras

Keras is an open software library that is used for artificial neural networks interface. It supports Python language. Keras acts as an interface for the TensorFlow library which is provided by Google. However, before Keras version 2.3, it supported many backends such as TensorFlow, Microsoft Cognitive Toolkit, Theano, and PlaidML. But after version 2.4, Keras now only supports TensorFlow. This allows for more user-friendly, faster speed, and more modular and extensible. Keras's official website can be found in (*Keras.*, 2023). This work also opts for Keras as a main framework.

2.4 Deep Neural Network

Neural networks are bio-neural copycat programming that enables a computer to learn from observational data just human learns something new. Deep learning is a powerful set of techniques for learning in neural networks. Deep Neural Network (DNN) is just a neural network with a lot of layers, normally network composed of 3 or more layers will be considered deep. It can be CNN or RNN. Neural networks and deep learning currently provide the best solutions to many problems in image recognition, speech recognition, and natural language processing. So, deep learning has become the most popular and powerful in solving many problems. For example, a Convolutional Neural network (CNN) can combine with these layers - The input layer, Convolution layer, ReLU layer, Pooling layer, Flattening layer, and Output layer to form a neural network.

2.4.1 Image Classification

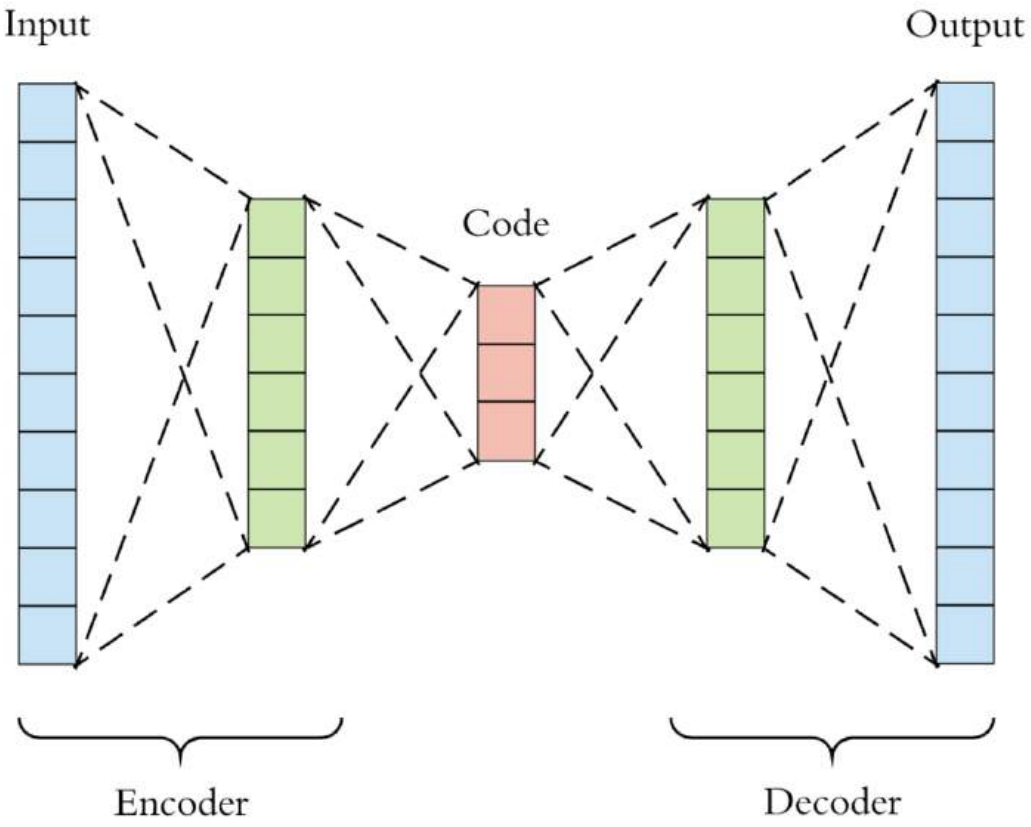
Image classification is one of many widely used DNNs. Its main purpose is to classify each input image into each class/category. For example, a network that classifies car images based on each type of car like a sedan, a truck, or a van. So image classification perfectly fits into this system, an in-house image classification created for sorting each input image into each problem type. Once trained, it can detect and identify the input image. This function is really helpful to a GAN since a pre-trained model selection relies on classification prediction. In addition, this classification is useful in final output qualification to ensure the best possible result.

2.4.2 Autoencoder

Autoencoder [Fig.2.1-Fig.2.2] is a part of machine learning. It is a data compression algorithm, different from a typical compression algorithm in which it learns automatically from examples rather than pre-built by a human. The idea is, it tries to discard data from an input (encoder/reduction) and rebuild data back to an output (decoder/reconstruction). This is useful when an input has a low quality (i.e. noise in image or audio) or the data is too big. Autoencoder will remove outlier data (that is noise signal) and carry over only significant data. This way, it can be used to reduce noise in an input or reduce an input size while preserving almost nearly its original quality. A downside of the autoencoder is data-specific, which means that it will only be able to compress data similar to what it has learned. Cannot be just used on any other data. For example, on an imagery task, an autoencoder trained on pictures of faces would produce a poor job of compressing

Figure 2.1

Autoencoder Layers



pictures of a car, because the features it would learn are face-specific. Autoencoder is a lossy process, its reconstruct outputs would be degraded compared to the original inputs.

Convolutional Autoencoder

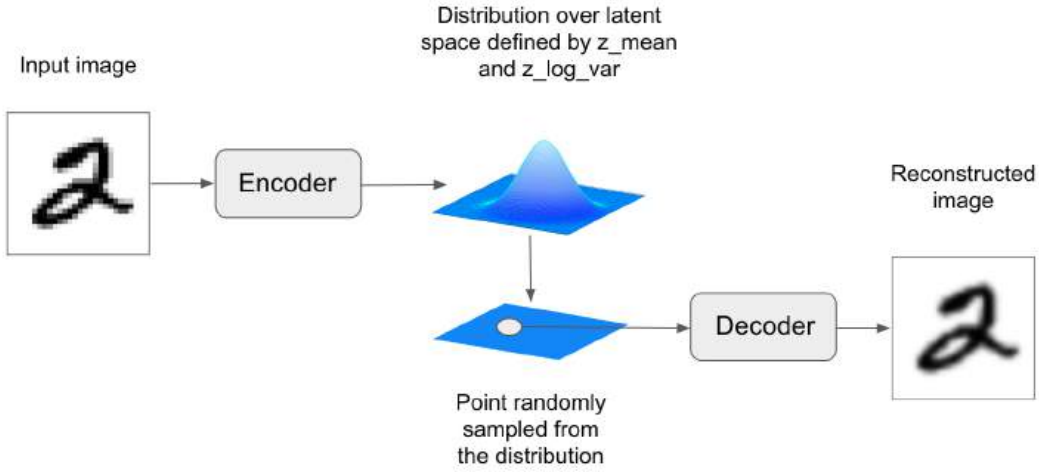
This is a more complex autoencoder. It is a more advanced version of the standard autoencoder because it has multiple convolution computation layers as a part of a network and is considered a deep network. The encoder consists of a stack of Conv2D and MaxPooling2D layers, while the decoder will consist of a stack of Conv2D and UpSampling2D layers, to do a reverse order of encoder.

2.4.3 U-Net

Developed by Computer Science Department of the University of Freiburg. U-Net is a convolutional neural network architecture semantic segmentation. It consists of two parts, a contracting path, and an expansive path. The contracting path follows the typical

Figure 2.2

Autoencoder Encodes Image into Z Space



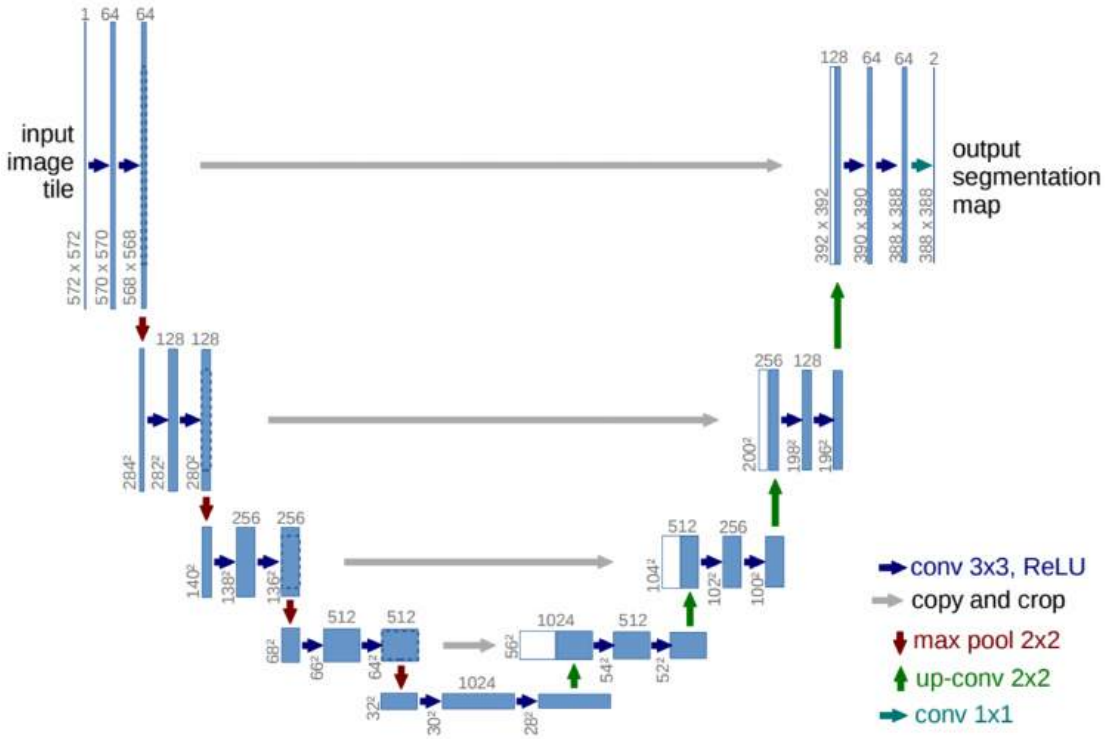
architecture of a convolutional network. It consists of the repeated application of two 3x3 convolutions, each followed by a rectified linear unit (ReLU) and a 2x2 max pooling operation with stride 2 for downsampling. At each downsampling step, we double the number of feature channels. Every step in the expansive path consists of an upsampling of the feature map followed by a 2x2 convolution that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the contracting path, and two 3x3 convolutions, each followed by a ReLU. The cropping is necessary due to the loss of border pixels in every convolution step. At the final layer, a 1x1 convolution is used to map each 64-component feature vector to the desired number of classes. In total the network has 23 convolutional layers. In Fig.2.3, each blue box corresponds to a multi-channel feature map. The number of channels is on top of the box. White boxes represent copied feature maps.

2.4.4 GAN

GAN stands for Generative Adversarial Network[Fig.2.4]. It is part of the DNN story. GAN is one exciting example of using a neural network. GAN is used in generating image, video, or audio data from a random input. It has two separate parts. A generator and a discriminator. A generator can be dubbed "the artist" and a discriminator is "the art critic". A generator's goal is to try to create the most realistic image from learning from real-world input images whereas the discriminator's goal is to tell whether that

Figure 2.3

U-Net Layer Configuration



input image is a real or fake image. During training[Fig.2.6], the generator progressively becomes better at creating images that look real, while the discriminator becomes better at telling them apart. The training process comes to an end when the discriminator can no longer distinguish real images from fakes.

In GAN, a minimax loss is used. It refers to the simultaneous optimization of the discriminator and generator models. The minimax is a strategy in which two players try to minimize the loss or cost for the worst case of the other player in turn-based games. In this case, the generator and discriminator take turns involving updating their model weights. The min and max refer to the minimization of the generator loss and the maximization of the discriminator loss. A discriminator and generator loss functions are derived as below.

$$D_{loss} = max(\log D(x) + \log(1 - D(G(z))))$$

and

$$G_{loss} = min(\log(1 - D(G(z))))$$

- $D(x)$ is a discriminator

Figure 2.4

A Briefed View of generic GAN Diagram

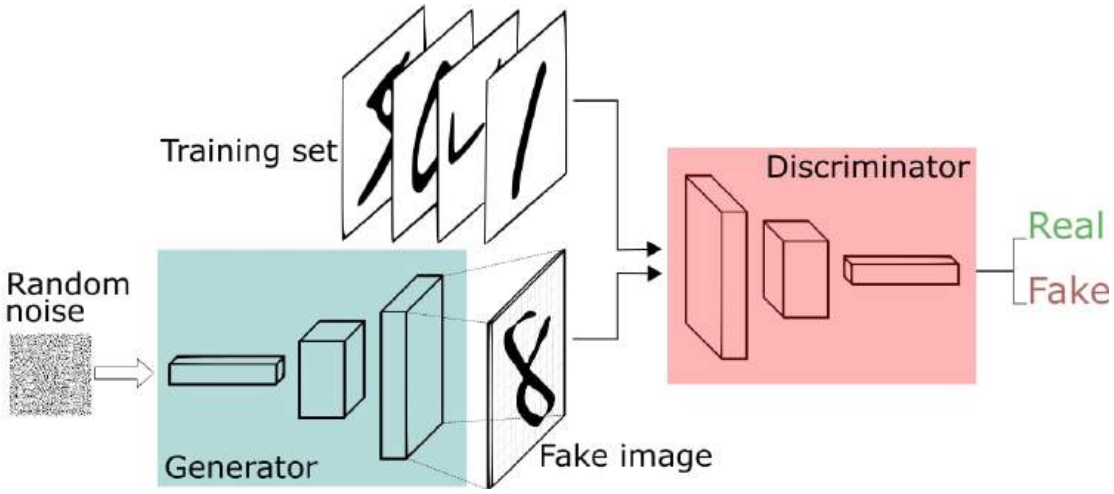


Figure 2.5

A Completed View of generic GAN Diagram

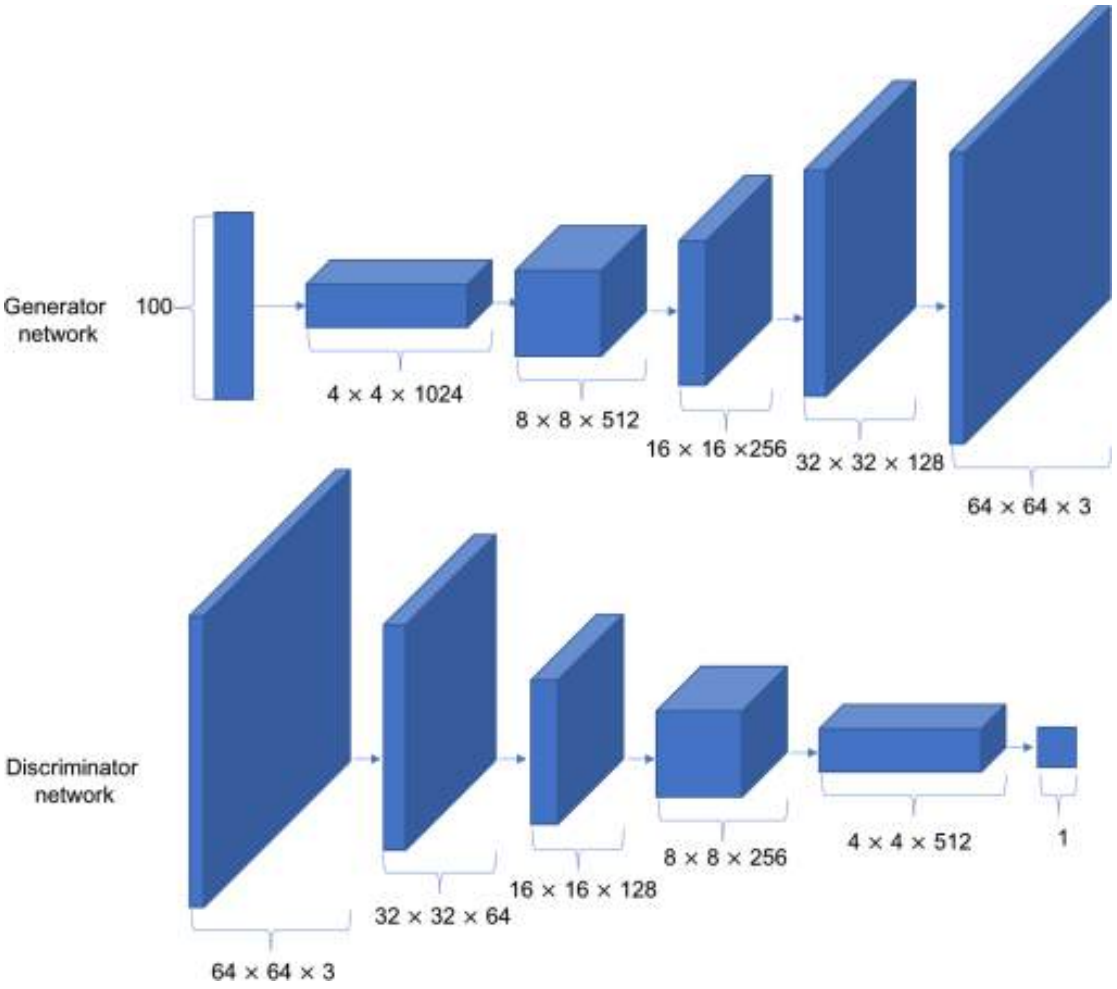
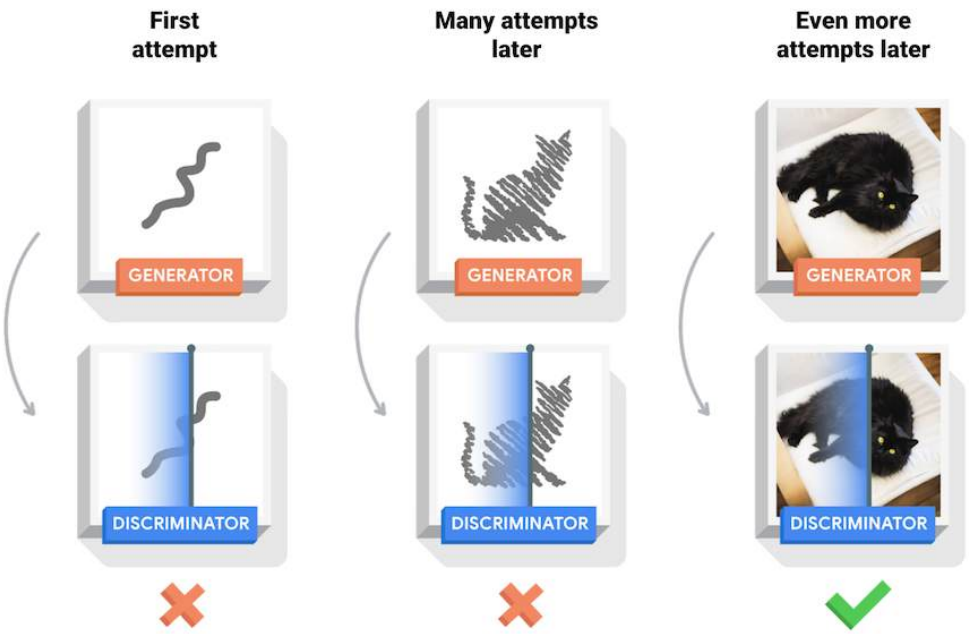


Figure 2.6

Generic GAN Training Process



- $G(z)$ is a generator
- z is an input signal

DCGAN

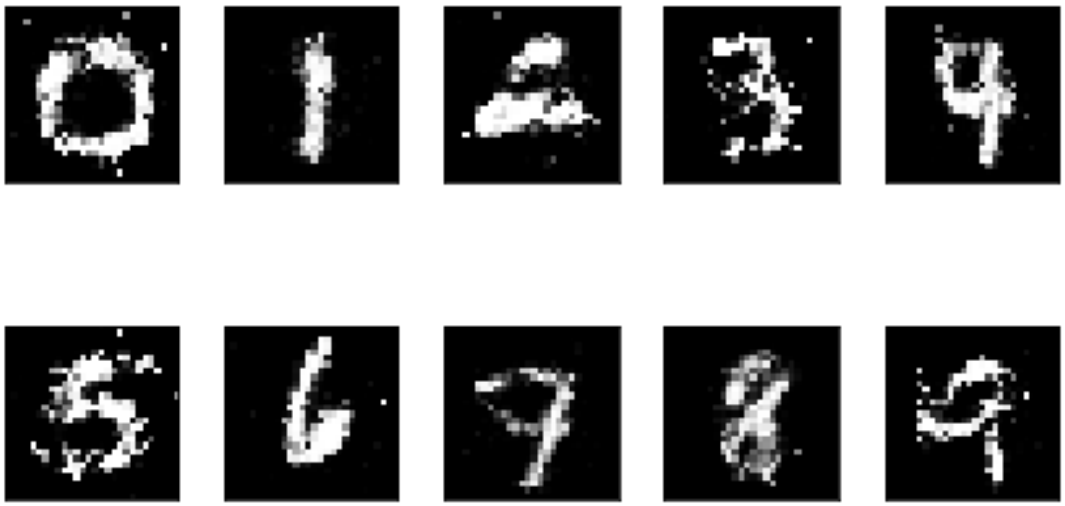
Deep Convolutional GAN is a variety of GAN systems. It is usually used to generate a new image from input random data (normal distribution). As a result, it is used to create a novel image that has never existed before. One downside of DCGAN is that because its output cannot be controlled so it is impossible to specify an output appearance or class. However, due to the fact that DCGAN is the most simple GAN form so it is easy to deploy.

CGAN

Due to the nature of DCGAN, its output cannot be controlled. On the other hand, CGAN (Conditional GAN) is able to do just that. It is used to generate a controllable output. For instance, with MNIST handwritten digits, CGAN can be controlled whether what

Figure 2.7

CGAN on MNIST Example



number the user needs to get. The output of CGAN is shown in Fig.2.7 where a label controls each generated number. A cumbersome of CGAN is that every data in the dataset requires to pair with its corresponding label, making the total number of files grow twice.

CycleGAN

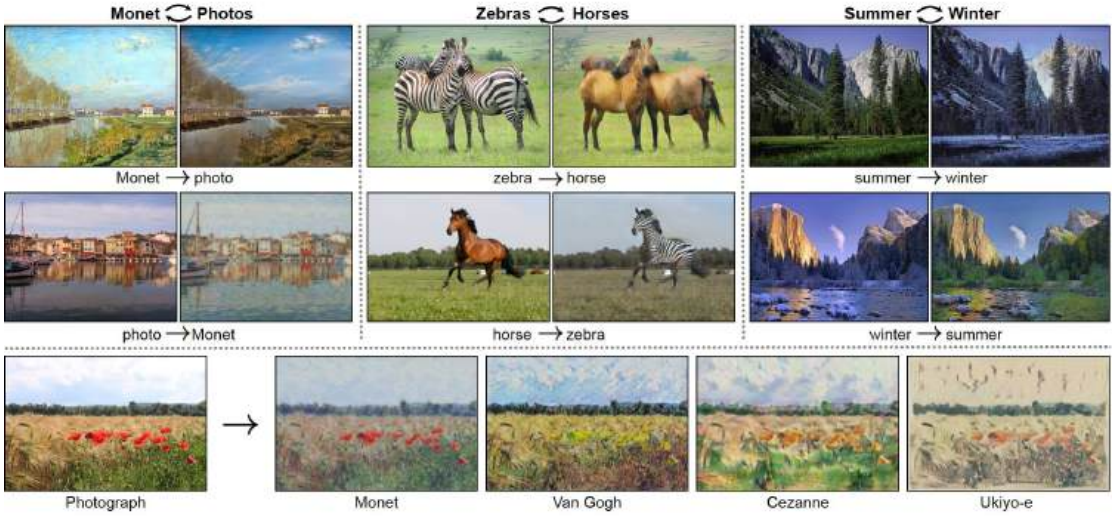
CycleGAN, or Cycle-Consistent GAN, specializes in swapping two domains like image-to-image translation. The image-to-image translation is a class of vision and graphics problems where the goal is to learn the mapping between an input image and an output image using a training set of aligned image pairs. In other words, a mapping between X domain and the Y domain is done by looking into each domain's particular characteristic, then transferring that information onto another and reverse. From this Fig.2.8 example, this GAN can be used to swap textures between horse and zebra.

2.4.5 Supervised VS Unsupervised Training

There are two types of machine learning training. One is supervised learning and unsupervised learning. Supervised learning requires the user to provide labels or information corresponding to training data. Provided data can be in a form of a label or class that groups a dataset. Unsupervised learning, on the other hand, does not need a label. It will try to find a pixel-based similarity and dissimilarity on each input data to form a

Figure 2.8

CycleGAN Example



group of data or data clusters. So these are applied to two applications, classification and clustering. The differences between those two are listed below (Sharma, 2022).

- Type - Clustering is an unsupervised learning method whereas classification is a supervised learning method
- Process – In clustering, data points are grouped as clusters based on their similarities. Classification involves classifying the input data as one of the class labels from the output variable.
- Prediction – Classification involves the prediction of the input variable based on the model building. Clustering is generally used to analyze the data and draw inferences from it for better decision-making.
- Splitting of data – Classification algorithms need the data to be split as training and test data for predicting and evaluating the model. Clustering algorithms do not need the splitting of data for their use.
- Data Label – Classification algorithms deal with labeled data whereas clustering algorithms deal with unlabelled data
- Stages – The classification process involves two stages, training and testing. The clustering process involves only the grouping of data.
- Complexity – As classification deals with a greater number of stages, the complexity of the classification algorithms is higher than the clustering algorithms whose aim is only to group the data

CHAPTER 3
METHODOLOGY

3.1 The Description of the Problems

As mentioned earlier, there are many common problems in traffic camera streams. Most seen problems were categorized into each group. A grouping is vital because the detection system can provide a corresponding description of an input to the recovery system precisely. Each group has its label and was used to train a detection system. Below is a list of problems that this study focuses on.

- Low Bitrate Dataset - Represents network congestion and low bandwidth network problems
- Low Light Dataset - Represents low light and nighttime situations
- Motion Blur - Horizontal Dataset - Represents slow camera shutter speed and speedy object problems
- Motion Blur - Vertical Dataset - Represents slow camera shutter speed and camera shaking due to vibration problems
- Normal Dataset (Normal/Good Condition) - Represents a high-quality, daylight situation in an ideal case

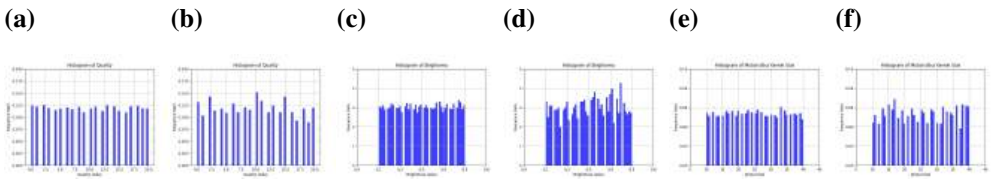
In the case of low bitrate problems where it is feasible to arrange them into sub-groups, these ranges refer to the JPG compression ratio range, which is mapped to a rating score system. This rating system will be useful in cases where a regular mathematical picture assessment is not possible to calculate.

- 1-Star: 0-20 JPG Quality Setting (Poorest Looking)
- 2-Star: 20-40 JPG Quality Setting
- 3-Star: 40-60 JPG Quality Setting
- 4-Star: 60-80 JPG Quality Setting
- 5-Star: 80-100 JPG Quality Setting (Best Looking)

The above descriptions are used in a deep learning classification, which is supervised training. It is trained to detect and categorize an occurring problem and acknowledges a difference between each compression ratio range to assess an output product.

Figure 3.1

Histogram Plot on each Problem Type Dataset - (a) Low Bitrate Train Set, (b) Low Bitrate Test Set, (c) Low Light Train Set, (d) Low Light Test Set, (e) Motion Blur Train Set and (f) Motion Blur Test Set



3.2 Dataset Preparation and Processing

Thai license plate images are the dataset used in this research and were provided by the AI Center, Asian Institute of Technology. There are 16,194 images in total, and they were separated into 14,500 train images and 1,694 evaluation images. However, due to ownership and privacy infringement, these images cannot be disclosed here. Due to the raw datasets being relatively small in resolution, all images after resizing were capped at 256x128 pixels, so a network is primarily designed to fit this image size. They were then processed into each category using a random filter to replicate real-world problems such as low JPG quality level for low bitrate problems or low brightness for a low light problem. These were prepared as the first step before training a model, and their category structure is shown below. In Fig.3.1a-Fig.3.1f, display histogram plots on each equal probability random distribution of filter level.

3.2.1 Low Bitrate Dataset

To create low bitrate/compressed images, OpenCV2 is used in writing a compressed image (a very low quality). In this way, this image can represent how each video frame in low bitrate video looks like. In the OpenCV2 document, It says that "For JPEG, it can be a quality (CV_IMWRITE_JPEG_QUALITY) from 0 to 100 (the higher is the better). Default value is 95.". But in this case, the quality value is set to 0 to 20 to match a 1-Star rating system range.

3.2.2 Low Light Dataset

To create low-light images, PIL ImageEnhance (Pillow library) is used to drop original image brightness from 100% to between 10% to 50%.

3.2.3 Out of focus Dataset

Using a random blur kernel to create a blur filter that varies between 7 - 15 kernel size. Then using this filter on the original dataset to simulate and creates out-of-focus dataset images.

3.2.4 Horizontal Motion Blur Dataset

A custom 2D kernel to create a motion blur filter (*How to Add Motion Blur to Numpy Array.*, 2016) is used with an image to create a motion blur image. Kernel filter size varies between 10x10-40x40. A blur kernel filter size is a pixel-shifting distance. For example, using 15 blur kernel size gives a 15-pixel shifting distance from the origin. This method creates a motion blur along the X-axis (0 degrees). A motion blur kernel filter has a formula below.

$$h = \frac{1}{m}$$

- h is the horizontal kernel value
- m is the size of the kernel

$$H_{m\times m} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ h_{m/2} & h_{m/2} & \cdots & h_{m/2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

3.2.5 Vertical Motion Blur Dataset

This vertical motion blur is the same idea as a horizontal motion blur but is different in the filter kernel. Again, kernel filter size varies between 10x10-40x40. This method creates a motion blur along the Y-axis (90 degrees). A motion blur kernel filter has a formula as below.

$$v = \frac{1}{m}$$

- v is a vertical kernel value
- m is the size of the kernel

$$V_{m\times m} = \begin{bmatrix} 0 & 0 & \cdots & v_{m/2} & \cdots & 0 \\ 0 & 0 & \cdots & v_{m/2} & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & v_{m/2} & \cdots & 0 \end{bmatrix}$$

3.2.6 Normal Dataset

A reference dataset, i.e., normal-looking, high-quality, and good-condition images. This dataset is used in recovery and measurement processes.

3.3 Proposed System, Model, and Layers

A proposed system is an end-to-end system combining two image classifications and one image recovery into one application. It helps traffic monitoring system to detect the anomaly and efficiently recover a bad ones when needed.

A detection system is built from CNN image classification and used to handle an incoming stream frame, detect a degraded frame and select a matching pre-trained model for a recovery system. This detection system can be found as a "Detector" in Fig.3.2. It is based on VGG-16 (*VGG-16 CNN Model.*, 2023) network but with a reduced layers count, resulting in three convolutional levels. The kernel size used in this network is 2. Its output (Image Description) is used in "Model Selector". It currently has five description output classes plus five star-rating classes in low-bitrate situations. This unit matches an input description with a predefined description found in the description of the problems to select a proper recovery model for that input, i.e., low-bitrate input needs a trained low-bitrate model. A selected model will then be passed to a recovery system.

A recovery system (LPRGAN) is a pixel-based license plate image recovery system. This proposed network features a reduction layer count configuration, including replacing the max pooling layer with a convolutional stride to speed up model performance (Springenberg, Dosovitskiy, Brox, & Riedmiller, 2015). After receiving an input image and its corresponding trained model from a model selector, the LPRGAN uses a trained model to recreate a high-quality version of the degraded input. Thus, this step is called the recovery process. The LPRGAN has two parts, a generator, and a discriminator. A generator is based on a convolutional autoencoder but optimized with a less complicated configuration. The main benefit of an autoencoder is that it can down-sampling data while preserving a significant representation of original data. All max-pooling layers from the original version are replaced with striding. An upscale process is also replaced with a convolutional transpose layer instead of the original upsampling layer. A generator layer configuration can be found in Fig.3.3 and Fig.3.4. A discriminator [Fig.3.5] is configured with a light VGG-16 version that also features a max-pooling replacement.

Both generator and discriminator have a kernel size of 3 in the main layer, except the last one (output layer) in the generator has a kernel size equal to 1. They also feature a sigmoid activation function instead of a hyperbolic tangent (tanh) for better output value coverage. These setups make a network more compact and responsive. A generated product from the LPRGAN will later be fed to a qualifier for evaluation purposes.

After a recovery step, a qualifier will validate a result with a rating score according to its problem type. This validation comes in handy when a degraded input is severely distorted and cannot be recovered. Since most existing works do not report on this case where the input is too distorted beyond GAN recovery capability, It is usually because the output will result in even worse quality. This occasion can sometimes happen when GAN could not reproduce the desired output image due to too much damage in an input image. Because of this inadequacy, our proposed system has one final touch, a fail-safe mechanism using a "Qualifier". It will determine which final result should be presented, either from a degraded input or an unrecovered result. This action is to prevent the end user gets an unpleasant disaster output. A qualifier has the same CNN image classification setup but is trained with a different purpose, featuring three convolutional levels. The kernel size used in this network is 2. It is used to evaluate and validate a result at the end of the process.

3.3.1 MaxPooling VS Stride

There are two ways in reducing data size between each layer, using a max pool layer or a stride which can be configured right in the Conv2D layer, it is a number of a grid that the kernel needs to skip in a convolution. For example, setting stride to 2 means a kernel will slide in 2D data by 2 slots. On the other hand, the Maxpool layer is used to select a maximum value on each kernel window to represent a data signature. As in Fig.3.7, the output layer is shrunken down while retaining a maximum number of blocks from the previous layer. So the difference is that a striding does a computational on the input but max pooling does not. Thus, using the stride method could gain a speed over max pooling because no additional layer is required.

3.4 System Flowchart

The proposed system flowchart shows in Fig.3.8. In the training stage (left chart), prepared low-quality images is fed into QRGAN for training purpose. Then a series of measurements (one is mathematics and another visual measurement) takes place before

Figure 3.2

Overview of Proposed System Block Diagram

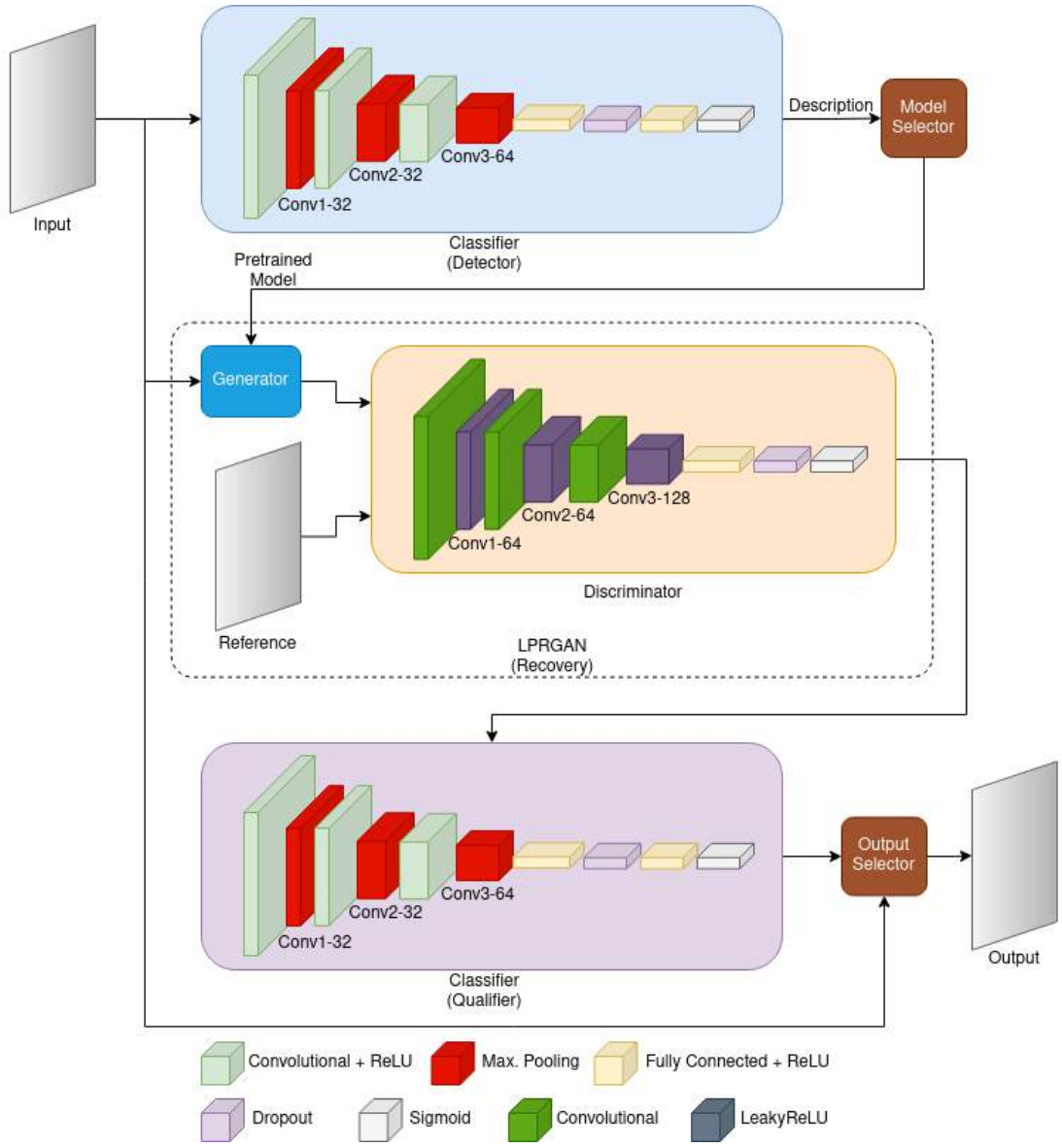
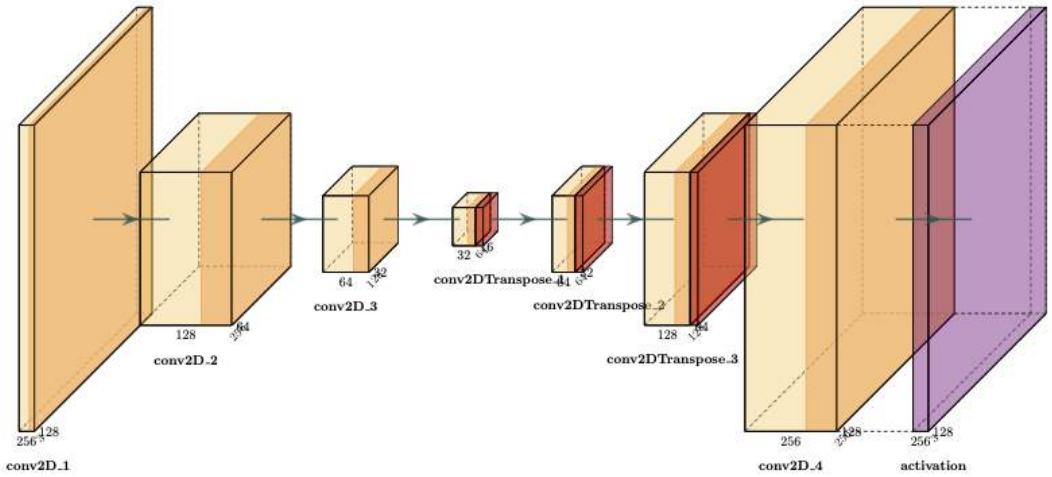


Figure 3.3

LPRGAN Generator Layers Visualization



the generated image reaches the final step. In the testing stage or real-world application usage, a proposed system monitors an input if it has a blockiness effect and falls below a threshold then that image will be fed into QRGAN for data recovery. After a recovery step, an item will be passed to output as the reconstructed image for display. If the input image is visually acceptable then it is simply bypassed to output (skip QRGAN).

3.5 Data Reconstruction using LPRGAN

The basic idea in any image compression (*Image Compression.*, 2022) is that the more the compression ratio is, the more space-saving, but it results in a blocky, bad-looking image. In the JPEG compression stage (*JPEG.*, 2022), once an image is translated from the spatial 2D domain into the frequency domain via DCT (Discrete Cosine Transformation), a low to mid-frequency is usually discarded to reduce file size, leaving out a high-frequency area untouched. This high-frequency signal comes from a sharp edge in the image. The reason to leave a sharp edge area in the image is that human eyes are sensitive to them. Removing the rest would not affect on final image in terms of visuals. This mentioned principle is also adopted for video compression with inter and intra-coding to save a bitrate.

On the other hand, GAN has the unique ability to generate fake data based on training in the deep learning world. GAN is used to learn a data loss pattern from a compression mechanism in this case. In the training process, pairs of good/ordinary-looking images

Figure 3.4

LPRGAN Generator Layer Diagram

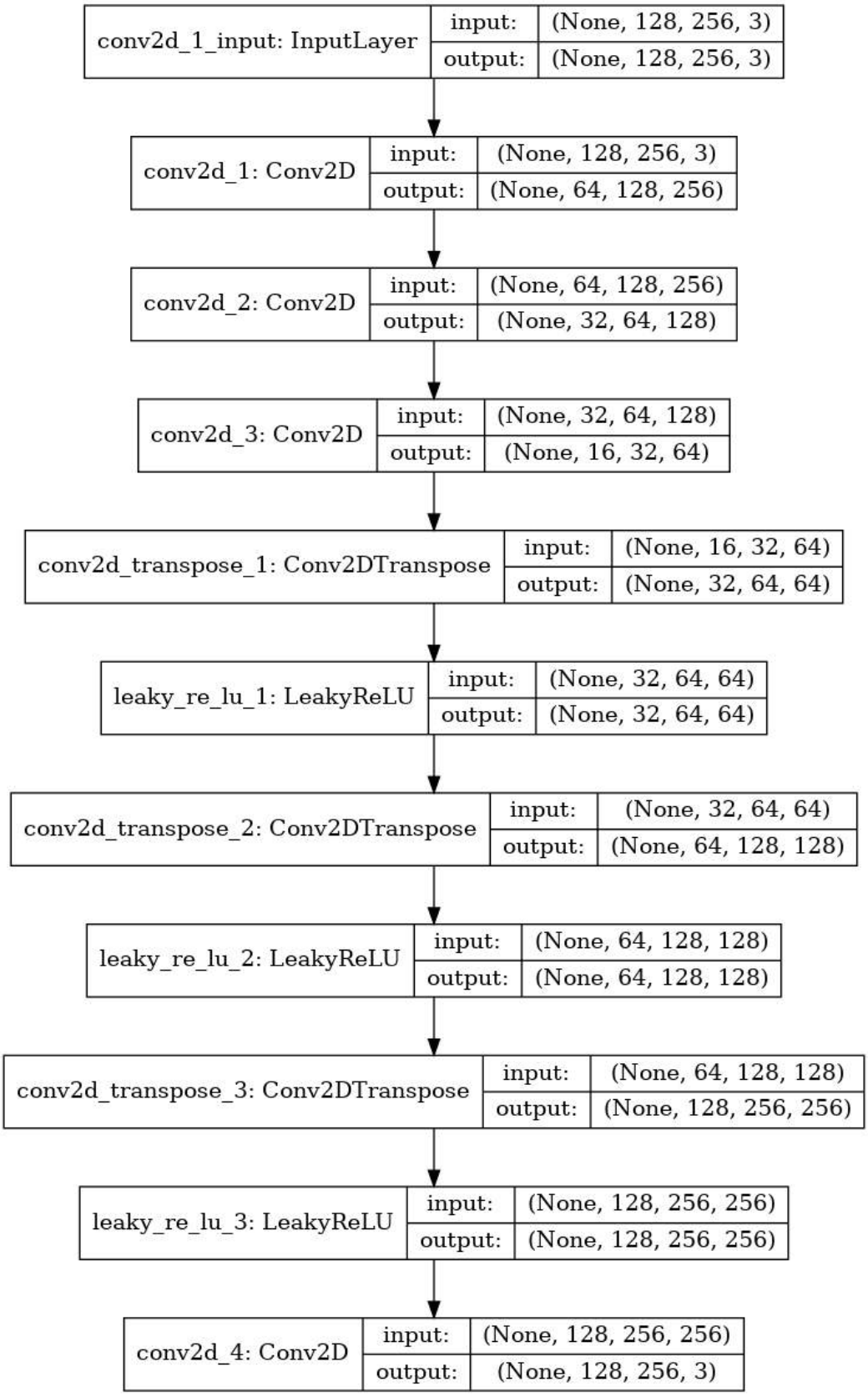


Figure 3.5

LPRGAN Discriminator Layer Diagram

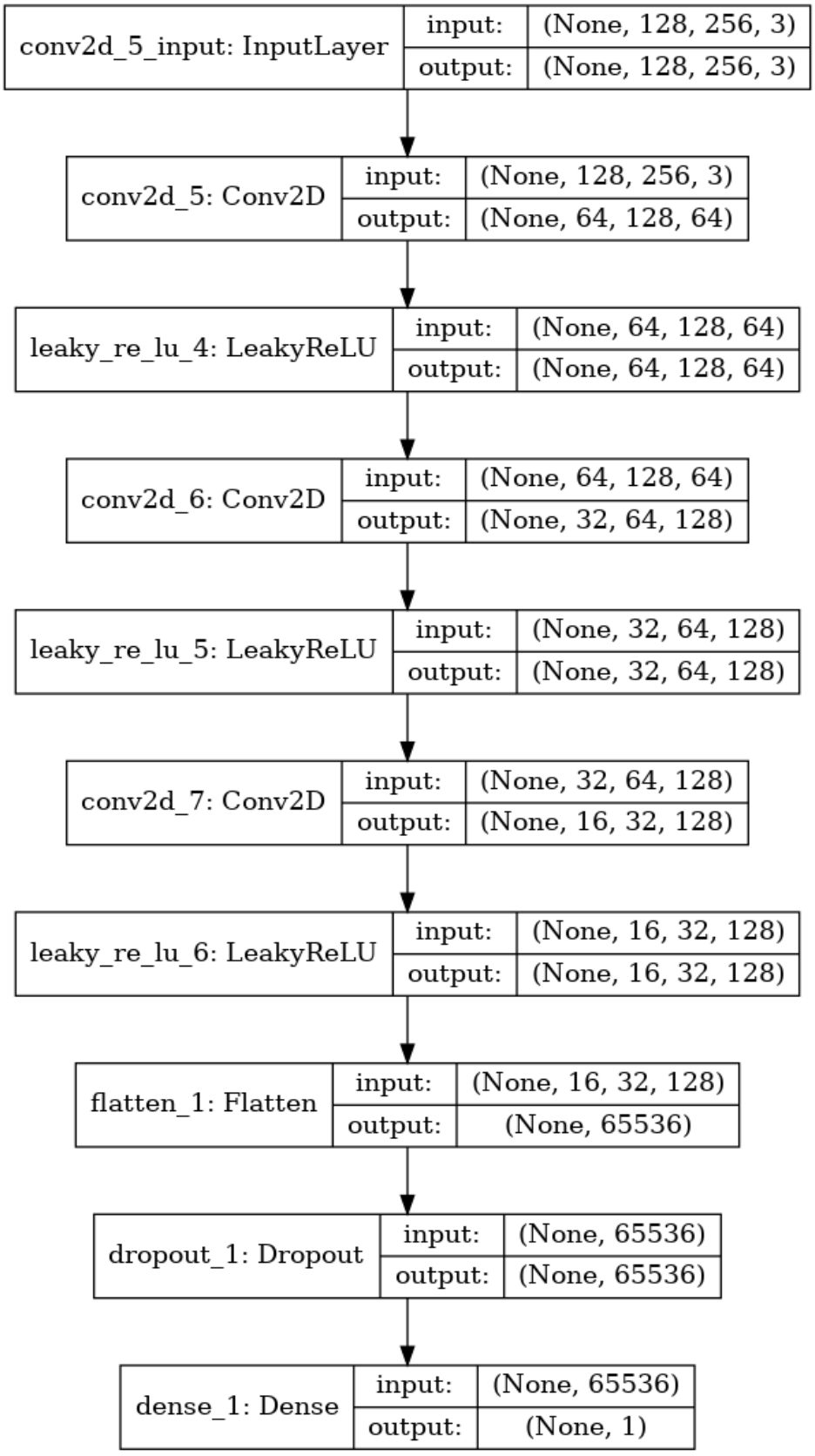


Figure 3.6

LPRGAN GAN Layer Diagram

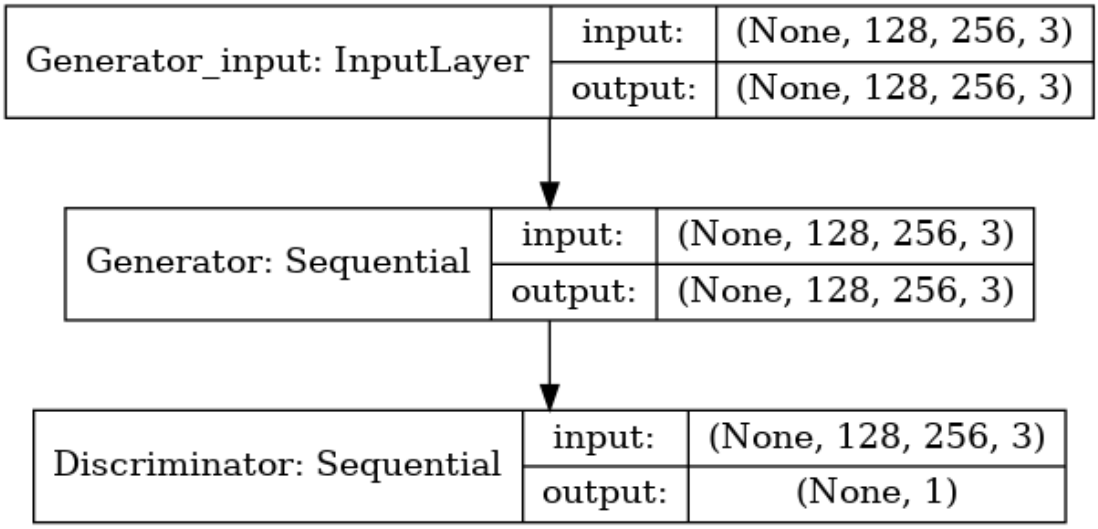


Figure 3.7

Maxpooling Operation

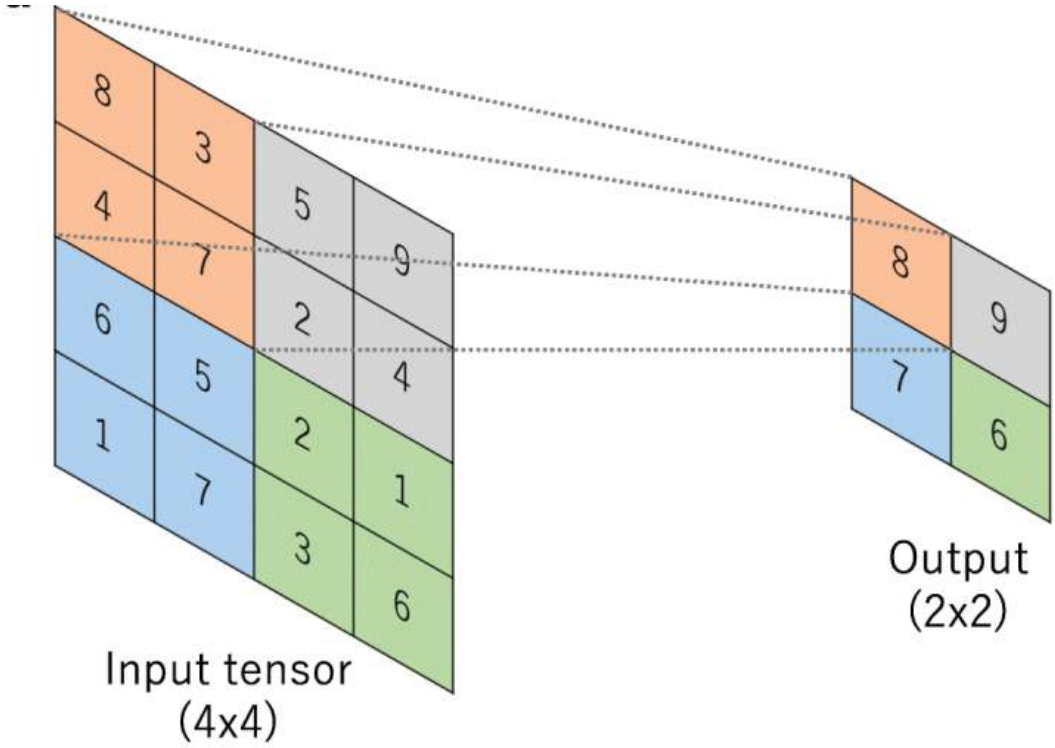
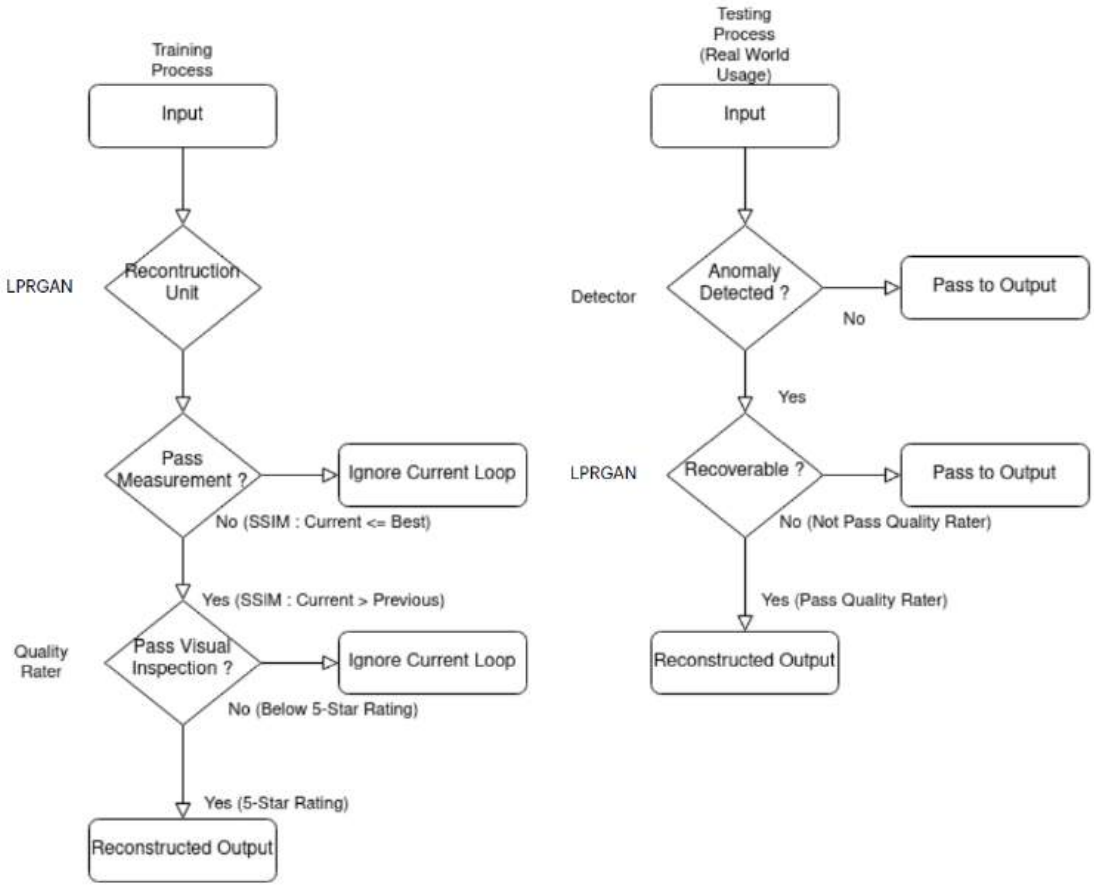


Figure 3.8

Logic System Flowchart



(references) (x) alongside real labels (y) and pairs of fake images (x^*) with fake labels (y^*) are used to train a discriminator (D) to learn on how to differentiate a good and a bad. Then, prepared degraded input images (X) with inverted real labels (y) is used in a generator to generate a fake image but due to the fact that a generator cannot be trained. So to train a GAN (G), the input (X) is passed through a series of Convolutional 2D layers (Conv2D), and their dimensions are halved in every Conv2D layer until they start to be upscaled back in Convolutional 2D Transpose layers (Conv2DTranspose). Conv2D is used to extract an input feature, called feature extraction, to represent its distinctive. In contrast, Conv2DTranspose, an invert of Conv2D, is used to rebuild pixel information based on the extracted feature. After a generator generates a new image, a discriminator will compare a generated image (x^*) with an original/reference image (x) to decide whether a generated image looks realistic enough. This way, a discriminator helps improve a generator's performance to generate an even better result as if it was trained. Ultimately, a generator target is to generate a realistic image that a discriminator can no longer tell them apart. This recovery system predicts and recovers data from compression or degraded loss to reduce a blocky artifact, smooth out a color gradient, and brighten or reduce a blurry image. Thus, this concept can help to repair degraded problems found in real life.

3.6 Training Process

A training result can be poor if there are insufficient resources such as computation power, dataset, or time available. A sufficient amount of training datasets is crucial since they directly impact training performance. The more variety of datasets available, usually the more system performance is likely to be. Also, a reasonable time is needed for the model to fit perfectly (balanced fit). This research also features a decay learning rate (*Learning Rate Decay and methods in Deep Learning.*, 2022) to accelerate the beginning of the training process.

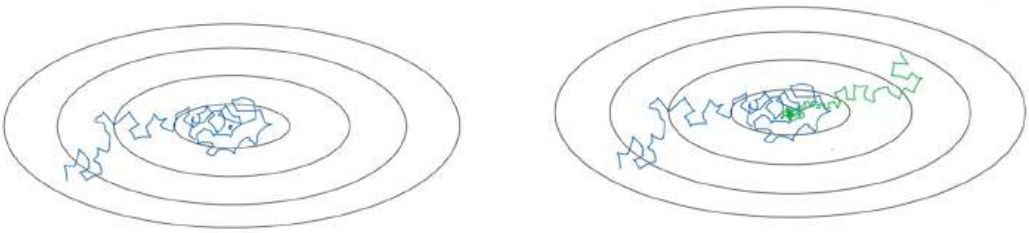
3.6.1 Fixed Learning Rate VS Decay Learning Rate

These are two common approaches in training any neural network. Fixed learning rate means the user set one constant value to a network and that value will be used through the entire training process whereas decay learning rate will use the same constant value as an initial value and slowly reduce a learning rate over time throughout a training step. This approach can help to prevent overshooting due to a big jump while the network is

Figure 3.9

Different Between Fixed and Decay Learning Rate Path - (a) Fixed Learning Rate Converging Path and (b) Decay Learning Rate Converging Path

(a) **(b)**



trying to converge toward a balanced point. A decay learning rate is a bit more advanced technique to optimize and generalize DNN when compared to a constant fixed learning rate. Both the constant learning rate and decay learning rate could be visualized in Fig.3.9a-Fig.3.9b. A decay learning rate has a formula below.

$$LR = LR_0 * \frac{1}{(1 + LR_{Decay} * N)}$$

- LR is the current learning rate in that iteration
- LR_0 is an initial constant learning rate value
- LR_{Decay} is a decay value used to control how fast the learning rate decreases
- N is the current iteration number

3.6.2 Detector/Qualifier Training

To train an image classifier to detect and categorize (qualify) incoming input, these are training parameters listed below.

- Iterations = 8
- Initial Learning Rate = 0.001
- Optimizer = RMSprop
- Activation = Sigmoid

To train an image classifier to scale an output rating score, these are training parameters listed below.

- Iterations = 4
- Initial Learning Rate = 0.001
- Optimizer = RMSprop
- Activation = Sigmoid

Algorithm 1 Detector/Qualifier Training

procedure TRAIN CLASSIFIER
  for $iteration = 1, 2, 3, \ldots$ **do**
    for $batch = 1, 2, 3, \ldots$ **do**
      Load random image samples x in a mini-batch manner
      Train the classifier on loaded samples (x)
      Compute the classifier loss $L(x)$ from predicted output $\hat{y}$ and actual output y and backpropagation a total error $\theta^{(L)}$ to minimize a loss
      Validate a model with validated samples (x')
      Compute and update Loss $L(x')$ to minimize a loss
    end for
  end for
  Save Model
  Save Final Iteration Result
  Plot a Training Graph
end procedure

3.6.3 Recovery System Training

The training parameters except the β_1 value are common to all experiments. The detail of values used in these studies is below.

- Main Kernel Size = 3
- Iterations = 200
- Initial Learning Rate = 0.0001
- Optimizer = Adam (Low Bitrate β_1 = 0.5, Low Light β_1 = 0.3, Horizontal Motion Blur β_1 = 0.7, Vertical Motion Blur β_1 = 0.5)
- Activation = Sigmoid

3.6.4 Fine Tuning Hyper Parameters

Finding the right value for each parameter used in a training process is one of the most difficult and time-consuming steps. Since most of them could only be found by trial-error approach. Parameters that need to be adjusted are listed below.

- Batch Size - It is the number of how many images in one batch. Because of the nature of neural network training, a small number of data is preferred.
- Optimizer - These are some DNN most popular optimizers to choose from such

Algorithm 2 LPRGAN Training

procedure TRAIN GAN
  for $iteration = 1, 2, 3, \ldots$ **do**
    for $batch = 1, 2, 3, \ldots$ **do**
      Load random image samples x in a mini-batch manner
      Train the discriminator on loaded samples with its real labels (x, y)
      Compute the discriminator loss $D(x)$ and backpropagation a total error $\theta^{(D)}$ to minimize a loss
      Using the generator to generate fake images from input $G(x') = x^*$
      Train the discriminator on fake images and fake labels sample (x^*, y^*)
      Compute the discriminator loss $D(x^*)$ and backpropagation a total error $\theta^{(D)}$ to minimize a loss
      Train the GAN by using low-quality input images with real labels sample (x', y)
      Compute and update GAN Loss $\theta^{(G)}$ to maximize a loss
    end for
    $LR \leftarrow LR_{iteration}$
    Load different random image samples x_0 in a mini-batch
    Evaluate the discriminator on real images set (x_0, y)
    Generate fake images from input $G(x'_0) = x^*_0$
    Evaluate the discriminator on fake images set with fake labels (x^*_0, y^*)
    $psnr \leftarrow PSNR(x, x^*)$ and $psnr_0 \leftarrow PSNR(x_0, x^*_0$
    $scc \leftarrow SCC(x, x^*)$ and $scc_0 \leftarrow SCC(x_0, x^*_0)$
    $ssim \leftarrow SSIM(x, x^*)$ and $ssim_0 \leftarrow SSIM(x_0, x^*_0)$
    $vif \leftarrow VIF(x, x^*)$ and $vif_0 \leftarrow VIF(x_0, x^*_0)$
    if $psnr_0 > saved\_psnr_0 \vee scc_0 > saved\_scc_0 \vee ssim_0 > saved\_ssim_0 \vee vif_0 > saved\_vif_0$ **then** $saved\_alue \leftarrow new\_value$
    end if
  end for
  $fid \leftarrow FID(x_0, x^*_0)$
end procedure

as Stochastic gradient descent (SGD), RMSprop, and Adam

- Initial Learning Rate and Decay Learning Rate - In case of using variable learning rate value, these two values are needed but if using constant learning rate, it will be only one learning rate value to be set.
- Total Iteration - Control how many loops/epochs in one training
- Each Network Layer Convolution Size including Kernel Size and Stride Size
- Activation Function - Two common functions that can be used in the final layer of the model here are sigmoid and tanh. The final layer is a decisive layer that determines the output result. The difference between these two in principle is that the former has a range scaled from 0 to 1 while the latter has -1 to 1. So tanh function gives better scaling and a better gradient. A graph plot is shown in Fig.3.10. Sigmoid and tanh formulas are as followed.

$$sigmoid = \frac{1}{1 + e^{-x}}$$

$$tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

or

$$tanh(x) = 2sigmoid(2x) - 1$$

3.6.5 Evaluation Process

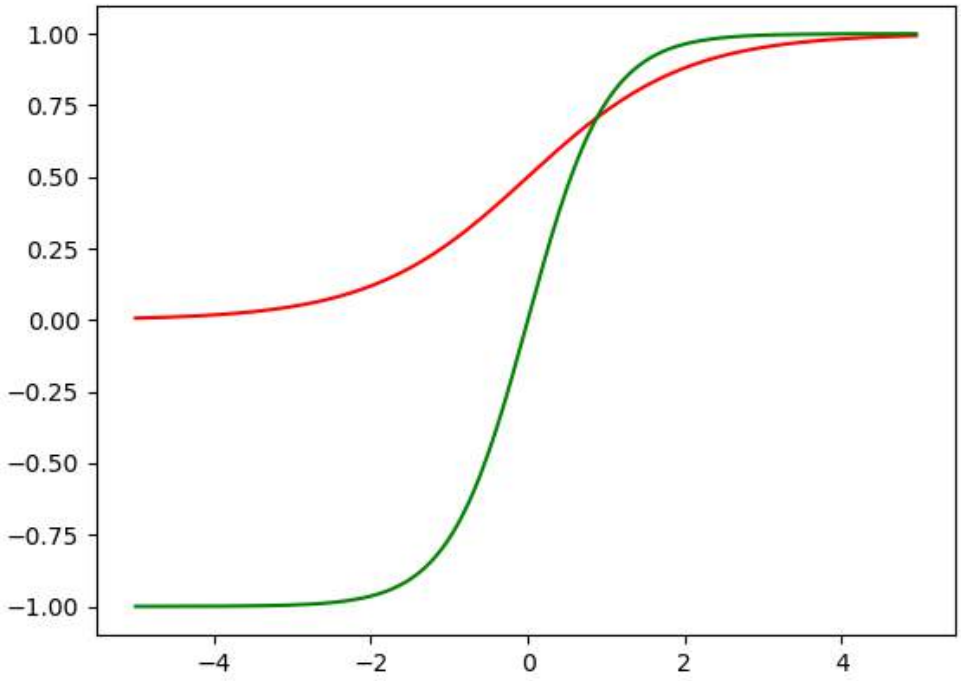
In between training, at the end of each epoch, there is an evaluation takes place. It can be considered one of the training stages. This is also an important step in the training process since it could tell how well the system can learn on seen dataset once applied to an unseen dataset. To evaluate the network model at end of each iteration, will be two methods, an evaluation image set is used to calculate measurement metrics (further on the Metric Approach topic) and record the best score for each metric to be later used in Testing Process.

3.7 Testing Process

Once the system has been trained and evaluated, then it is time to test a system with a separated, unseen image set. They can be either got from computer processing or real-world record. This test image will be fed into the fully trained system to generate an output which then will be judged based on two criteria, one is a visual inspection from both human and image classification network and another is metric measurement same as in the evaluation step. There are two common image quality measurement or

Figure 3.10

Sigmoid (Red) and Tanh (Green) Function Output



distortion assessment types. First is a human visual system (HVS) to assess perceptual quality measures. The second one is mathematical measurements. In this study, the first method is implemented in the "Visual Approach" and the second method is implemented in the "Metric Approach" which is using python package named "Sewar" to measure an output quality in a qualitative style.

3.7.1 Visual Approach

Due to a problem that visual inspection from humans might have a bias because a human is hard to get a consistent result since each person has a different perception, and the human eye is hard to tell a slight change between images. So using well-trained deep learning can help estimate an output quality is a better method. If a reconstructed image is justified as a normal/good image by training it with a bunch of images in each problem type. Once got output from GAN is, then feed this output to the trained image classification network to measure the quality of the output from the GAN system and whether it is flagged as normal/good label. Otherwise, the result will fall into one of the bad labels. These are two metrics used in visual approach.

- Quality Rater (Qualifier)
- Prediction Confidence (Recogniser)

3.7.2 Synthetic Metric Approach

A mathematical assessment in this study includes these measurements, FID, PSNR, SCC, SSIM, and VIF. FID is explicitly designed to assess a non-authentic, generated image. This FID score is offered from Pytorch-fid (*FID score for PyTorch.*, 2022) package. The rest of the metrics are offered by Sewar (*Sewar Python Package.*, 2022) package. Below list are all metrics conducted in this research.

- FID (Fréchet Inception Distance) (*How to Implement the Frechet Inception Distance (FID) for Evaluating GANs.*, 2022)
- PSNR (Peak Signal-to-Noise Ratio) (*Peak to Signal Noise Ratio.*, 2022)
- SCC (Spatial Correlation Coefficient) (Vallejos, Perez, Ellison, & Richardson, 2019)
- SSIM (Structural Similarity Index) (*Structural Similarity Index.*, 2022)
- VIF (Visual Information Fidelity) (*Visual Information Fidelity.*, 2022)
- File size
- Time Usage

- Render Speed
- Memory Usage

FID

The Fréchet inception distance (FID) is a metric used to assess the quality of images specifically created by the GAN. It compares the distribution of generated images with the distribution of real images used to train the generator. In other words, this score tells how well the GAN is from comparing generated dataset with a training dataset. FID compares the mean and standard deviation of one of the deeper layers in the Inception v3 network. These near-end layers are near output nodes that correspond to real-world objects. Thus, it can mimic the human perception of similarity in images. The FID value will be 0 if paired datasets are identical, and the value will go higher when there is more difference (deviation) between two input datasets. The lower it is, the better.

$$FID = ||\mu_1 - \mu_2||^2 + tr(\Sigma_1 + \Sigma_2 - 2(\Sigma_1^{\frac{1}{2}} \cdot \Sigma_1 \cdot \Sigma_2^{\frac{1}{2}})^{\frac{1}{2}})$$

- mu_1 and mu_2 refer to the feature-wise mean of the real and generated images, e.g., 2,048 element vectors where each element is the mean feature observed across the images.
- Σ_1 and Σ_2 are the covariance matrix for the real and generated feature vectors
- $||\mu_1 - \mu_2||^2$ refers to the sum squared difference between the two mean vectors. tr is the trace linear algebra operation, e.g., the sum of the elements along the main diagonal of the square matrix.

PSNR

PSNR takes two inputs to calculate a signal power using the MSE of the reference image from the original image. Its range is usually between 25-48dB for an 8-bit image, where higher is better. PSNR has a formula as below.

$$PSNR = 10 * log10(\frac{256^2}{MSE})$$

- MSE is the mean squared error that measures the average of the squares of the errors, the average squared difference between estimated values and actual value

SCC

The spatial Correlation Coefficient calculates the spatial correlation coefficient score from paired images. SCC is defined as a spatial concordance coefficient for second-order stationary processes. This problem has been widely addressed in a non-spatial context, but here a coefficient that for a fixed spatial lag allows one to compare two spatial sequences along a 45°line. The proposed coefficient was explored for the bivariate Matérn and Wendland covariance functions. The asymptotic normality of a sample version of the spatial concordance coefficient for an increasing domain sampling framework was established for the Wendland covariance function. To work with large digital images, a local approach was proposed for estimating the concordance that uses local spatial models on non-overlapping windows. Monte Carlo simulations were used to gain additional insights into the asymptotic properties of finite sample sizes. The analysis showed that the local approach helped to explain a percentage of the non-spatial concordance and provided additional information about its decay as a function of the spatial lag. The generalized SCC formula is shown below.

$$\rho^c(h) = \frac{2\sigma_X\sigma_Y}{\sigma_X^2 + \sigma_Y^2}\rho_{XY}R(h, \phi_{XY})$$

- $|\rho^c(h)| \leq |\rho_{XY}(h)| \leq 1$
- $|\rho^c(h)| = 0$ if $|\rho_{XY}(h)| = 0$
- σ_X is a standard deviation of X, and σ_Y is a standard deviation of Y. Given that X and Y are two random variables.
- $R(h, \phi_{XY})$ is a correlation function with parameter vector ϕ in which a covariance function is defined

SSIM

This image quality assessment techniques rely on quantifying errors between a reference and a sample image, which needs two images to do a calculation. A common metric is to quantify the difference in the values of each of the corresponding pixels between the sample and the reference images but relying on the human visual perception system (HVS), on the other hand, is highly capable of identifying structural information from a scene and hence identifying the differences between the information extracted from a

reference and a sample scene. Hence, a metric that replicates this behavior will perform better on tasks that involve differentiating between a sample and a reference image. As mentioned, it takes two inputs, one is original and another is reference then break down into 3 parts, luminance, contrast, and structural comparison. The outcome value varies between 0 to 1 where 1 is identical and 0 is not identical where higher is better. SSIM has a general formula as below.

$$SSIM(i_1, i_2) = [l(i_1, i_2)]^{\alpha} \cdot [c(i_1, i_2)]^{\beta} \cdot [s(i_1, i_2)]^{\gamma}$$

- i_1 is the first image
- i_2 is the second image
- $l(i_1, i_2)$ is luminance comparison function
- $c(i_1, i_2)$ is contrast comparison function
- $s(i_1, i_2)$ is structural comparison function
- α > 0, β > 0, γ > 0 denote the relative importance of each of the metrics

VIF

Visual Information Fidelity calculates pixel-based visual information fidelity. The purpose of the HVS model in the information fidelity setup is to quantify the uncertainty that the HVS adds to the signal that flows through it. As a matter of analytical and computational simplicity, all sources of HVS uncertainty merged into one additive noise component that serves as a distortion baseline in comparison to which the distortion added by the distortion channel could be evaluated, called lumped HVS distortion visual noise and model it as a stationary, zero mean, additive white Gaussian noise model in the wavelet domain. Thus, the HVS noise is modeled in the wavelet domain as stationary. The VIF is computed for a collection of wavelet coefficients that could represent either an entire subband of an image or a spatially localized set of subband coefficients. In the former case, the VIF is a single number that quantifies the information fidelity for the entire image, whereas in the latter case, a sliding-window approach could be used to compute a quality map that could visually illustrate how the visual quality of the test image varies over space. In the VIF system, the higher score is the better where 1 is the best case. A general term of the VIF formula is below.

$$VIF = \frac{\Sigma_{j(subbands)} I(\vec{C}^{N_j}; \vec{F}^{N_j} | s^{N_j})}{\Sigma_{j(subbands)} I(\vec{C}^{N_j}; \vec{E}^{N_j} | s^{N_j})}$$

- $I(\vec{C}^N; \vec{F}^N|s^N)$ and $I(\vec{C}^N; \vec{E}^N|s^N)$ represent the information that could ideally be extracted by the brain from a particular subband of the reference and test images, respectively
- $\vec{C}^{N_j}$ represents N elements of the RF that describe the coefficients from subband j, and so on

File Size

Usually, any image file size will be larger when its resolution and compression quality value is bigger. However, suppose all images have the exact resolution and compression ratio. In that case, their file size can reflect how much that image holds information (image detail). The bigger size, the more image contains fine detail. The file size in this study has a unit as kilobytes (kB).

Time Usage

Time usage in this study is a training time usage per training epoch. It indicates how fast a model setup is, and the faster approach is always preferable. A unit for the time used in this study is second.

Render Speed

A render speed has a unit of frames per second (FPS). It indicates how fast a model setup is. The faster approach is always preferable. This metric is used in a testing process. This FPS can also tell if a testing model is viable for real-time operation since a CCTV camera typically operates from 15 to 30 FPS.

Memory Usage

Memory usage can ultimately be a deciding factor if any approach could be deployed on edge computing devices since most of them have a very limited amount of memory. Usually, these embedded systems have around 32MB to 512MB in memory capacity.

So a model should use as the least RAM as possible.

3.7.3 Hardware

Below is a custom build model training PC running Ubuntu 18.04. This PC was used throughout the entire research.

- AMD Ryzen 7 2700X 8 Cores 16 Threads CPU
- Asrock B450 Gaming K4 Motherboard
- Galax NVIDIA GeForce RTX 2080 Ti 11GB GDDR6 352-bit 260 Watts GPU
- Corsair 32GB DDR4 Memory
- WD Green SATA SSD 120GB
- Cooler Master 80PLUS Gold Full Modular 750W Power Supply

This is a workstation PC used in testing.

- Intel Xeon Processor E3-1200 v6 72 Watts CPU
- 8GB DDR4 Memory
- 2x 3.5" Enterprise SATA 7.2k 1TB

The Microsoft Surface Go 3 tablet PC is an ultra-low-power PC used in testing.

- Intel Core i3 10100Y 5 Watts Ultra Low Power CPU
- 1866MHz 8GB DDR3 Memory
- 128GB SSD PCIe Storage

This is a specification of a camera used in this study.

- Lilin ZR8022EX10 1080p 2MP CMOS Sensor IP Camera

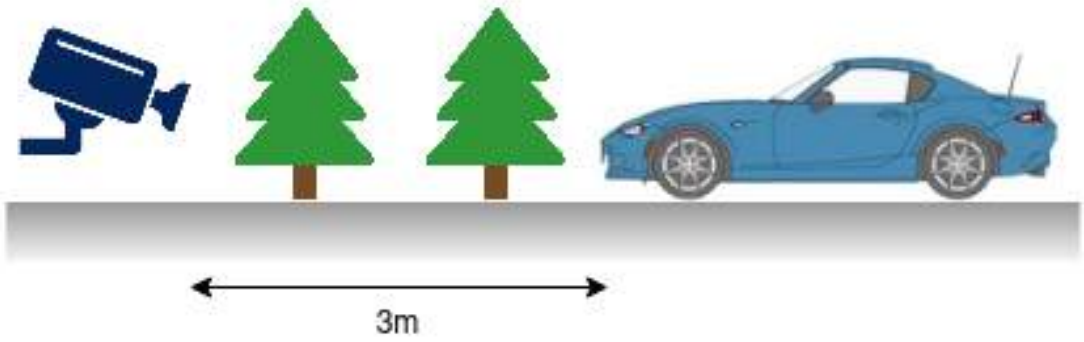
3.7.4 Software

Below is a software list used in this thesis.

- 64-bit Ubuntu 18.04.3 LTS
- NVIDIA Driver V.430.26
- CUDA Toolkit V.10.2 + 10.0
- CuDNN V.7.6.2
- Python V.3.6.8
- PIP3 V.9.0.1
- Sublime Text 3
- Tensorflow-gpu V.1.14.0 (Not Support GPU Acceleration on CUDA10.1+)
- Keras V.2.3.0
- OpenCV2 V.4.1.1.26

Figure 3.11

Test Scene Demonstration



- Matplotlib V.3.1.3
- Numpy V.1.18.1
- Pillow V.7.0.0
- Scipy V.1.4.1
- Scikit-Image V.0.16.2
- h5py V.2.10.0

3.7.5 Test Scene

A figure shown in Fig.3.11, represents how a test scene was set up. A camera model used in the testing is Lilin ZR8022EX10. The distance between a camera and the front bumper of a car where a license plate is located is 3 ± 0.5 meters. A car is stationary in front of the camera to be safe and avoid an accident. There is no physical restriction on the distance between a camera and a plate since all collected images are cropped to fit a license plate area in post-processing.

CHAPTER 4
RESULT

4.1 Classification Training Result

Result on training an image classification as a degrading detector/qualifier is in Fig.4.1. It is obvious that training beyond 6 epochs is useless in this case since model starts to overfit a data. The optimum point here is at the 4th epoch. Due to training dataset is rather large and learning rate is quite big so it needs only a few epochs to reach its balanced point. The same story to a rater training [Fig.4.2], a loss is relatively low at the 4th epoch. So a training is saturated.

4.2 Classification Testing Result

As shown result, a quality rating works well in predicting each JPEG compression-level image. This could not be done by humans since a human could not be able to tell a difference down to a very tiny level. A set of image varieties is used to test this ML. Below are sample results from quality ratings on different JPEG compression level images [Fig.4.3a-Fig.4.3b].

The below image is an original/reference image[1] found in Fig.4.4. It will be used to compare against each problem type generated images.

4.3 Optimization

There are many ways to optimize a result. The most effective one is getting more training data with more variety but if a dataset is already at its limit then trying changing layer configuration or parameter in the model is an another option. Table.4.1 represents the combination between each layer and the last layer kernel size settings. This experiment studies the result difference between each set and finds the best kernel size setting for a current generator setup. Furthermore, a main layer kernel size setting is used in both a generator and a discriminator.

4.3.1 Kernel Size

Kernel size in each layer can affect the final output. Normally kernel size in a layer is usually odd numbers such as 3, 5, or 7. This result shows an effect on the final output

[1]This license plate is the author's ownership, and it does not violate third party privacy or rights.

Figure 4.1

A Detector/Qualifier Training Result

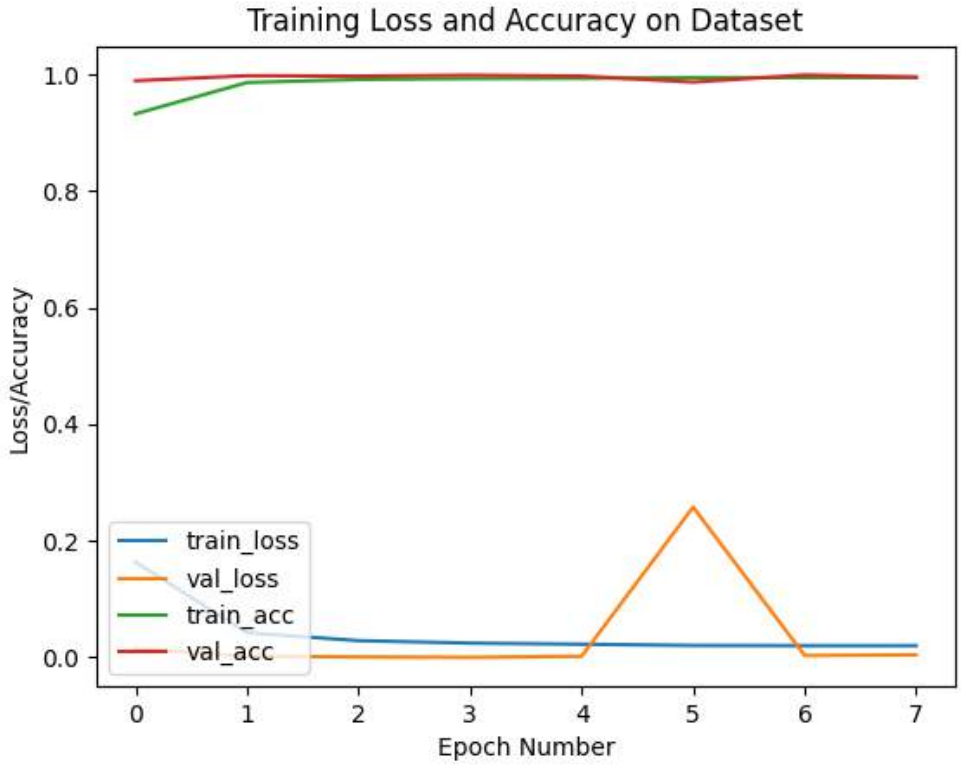


Figure 4.2

A Rater Training Result

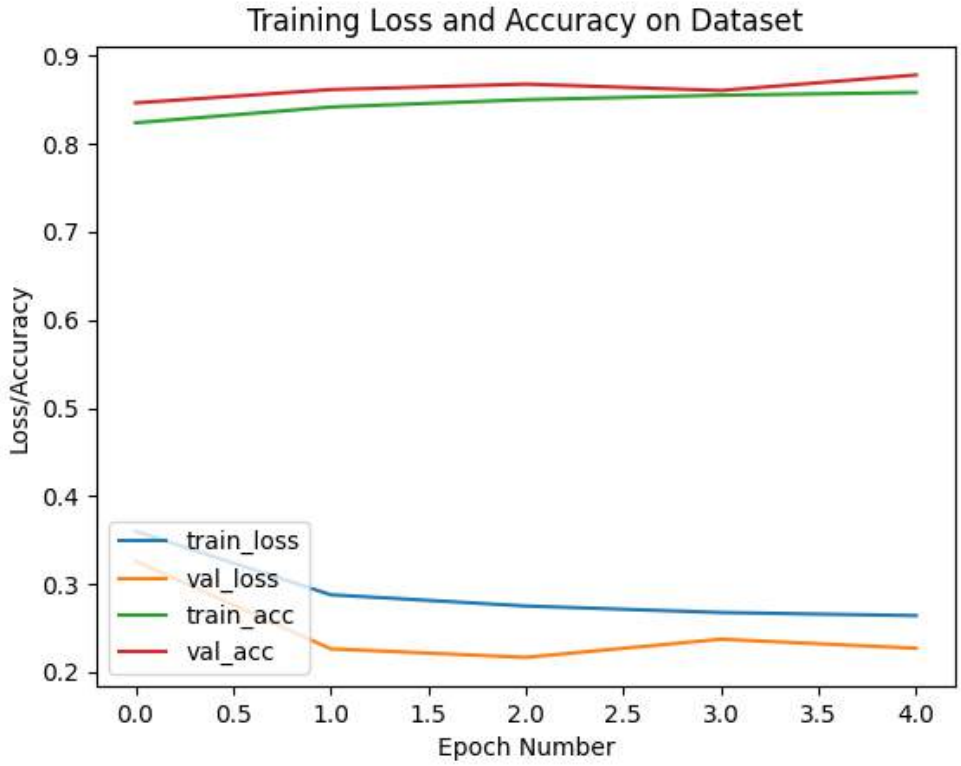


Figure 4.3

Classifier Prediction Result, (a) JPEG Quality 0 = 1-Star and (b) JPEG Quality 50 = 3-Star

(a) **(b)**

Figure 4.4

Original/Reference Image

Figure 4.5

Comparison Between Each Kernel Size Setting - (a) K=3, (b) K=4, (c) K=5, (d) K=7 and (e) K=9

(a) **(b)** **(c)** **(d)** **(e)**

by selecting a different kernel size [Fig.4.5a-Fig.4.5e].

Another value that can be tweaked is the last layer's kernel size. Below is a comparison between setting the last generator layer's kernel size from 1, 3, and 5 while keeping the other's kernel size as 4. This test result shows a better result when using the last layer kernel size = 1 [Fig.4.6a-Fig.4.6c].

4.3.2 Layer Depth Configuration

Table.4.2 shows ablation studies between 11, 13, and 15 of the generator's layers configurations. A training time is a time usage per epoch, unit in seconds. As shown below, the more layer counts the more computational time is needed making overall performance drop. Also, at the current stage, using 11 layers count produces the best outcome for both metrics.

Figure 4.6

Last Layer Kernel Size Configuration - (a) K=1, (b) K=3 and (c) K=5

(a) **(b)** **(c)**

Table 4.1

SSIM Score on Each Kernel Size Table

Kernel Size	Main K=3	Main K=4	Main K=5	Main K=7	Main K=9
Last K=1	**0.788**	0.782	0.693	0.756	0.742
Last K=3	0.764	0.629	0.779	0.756	0.748
Last K=5	0.773	0.762	0.763	0.766	0.695

Table 4.2

SSIM Score on Each Layer Depth Configuration Table

Layer Counts	SSIM	Training Time
11	**0.786**	**141.9**
13	0.731	146.9
15	0.135	148.6

Figure 4.7

Stride vs MaxPooling Result - (a) Striding (SSIM 0.787), (b) Maxpooling (SSIM 0.554)

(a) **(b)**

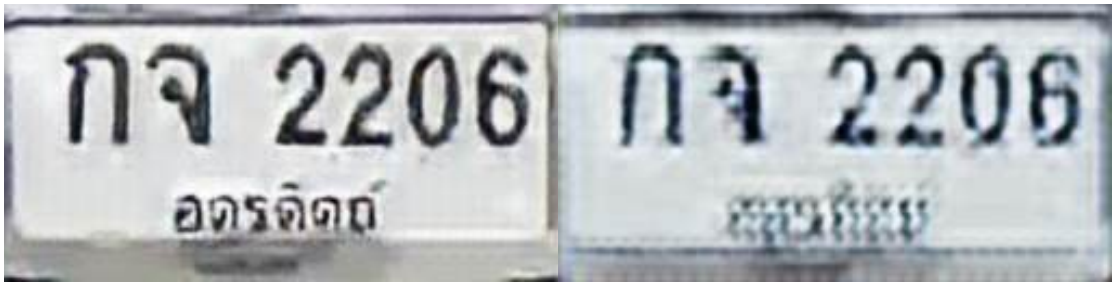


Figure 4.8

Result from Using Sigmoid VS Tanh Function - (a) Sigmoid (SSIM 0.787), (b) Tanh (SSIM 0.784)

(a) **(b)**

4.3.3 Stride VS Maxpooling2D

As mentioned earlier, this research uses stride instead of max pool layer to gain more speed but to support a decision, here is a result compares between using stride set to 2 without Maxpooling2D and using Maxpooling2D with stride set to 1. In the end, max pooling method did not work well here, it has too many artifacts on the output image.

4.3.4 Sigmoid VS Tanh as Activation Function

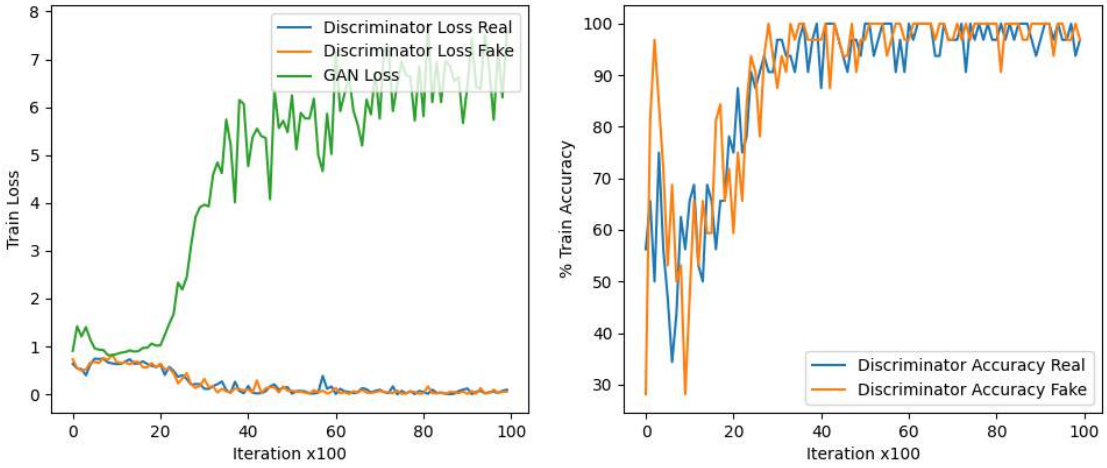
As discussed earlier, these proposed models' activation functions have been modified. It is slightly different from selecting the Sigmoid to the Tanh function. However, an experiment result shows that the Sigmoid function performs better in the SSIM score.

4.3.5 Learning Rate Adjustment

This subsection demonstrates how different when selecting a learning rate. In Fig.4.9, picking a learning rate value too high would make a training weight overshoots, mean-

Figure 4.9

Training Performance of Learning Rate = 0.001



ing that a loss has gone saturated and accuracy hit almost 100% (optimal accuracy in GAN training is 50%), including those measurement metrics started to decline, in very early point of the training. On the other hand, when picking a proper learning rate value [Fig.4.10], a training loss stays consistent, and an accuracy value hangs in between the middle, especially near the end of the training, accuracy is stable at around 50% (balanced-fit) as well as measurement metrics that reach a very high peak in the later results.

4.3.6 Decay Rate Adjustment

Here shows how the decay rate has an impact on training values. In Fig.4.11, when using a bigger decay rate, a training enters equilibrium after around the 80th iteration. But when using a smaller decay rate number [Fig.4.12], training enters a steady state faster at around the 20th loop onward because a smaller decay value makes the learning rate bigger. Thus training goes faster. When using decay rate value too high, it would need more iterations to reach a peak, making the training process much slower.

- At (0,0) - Training Loss
- At (0,1) - Training Accuracy
- At (0,2) - Training PSNR
- At (1,0) - Training SCC
- At (1,1) - Training SSIM
- At (1,2) - Training VIF

Figure 4.10

Training Performance of Learning Rate = 0.0001

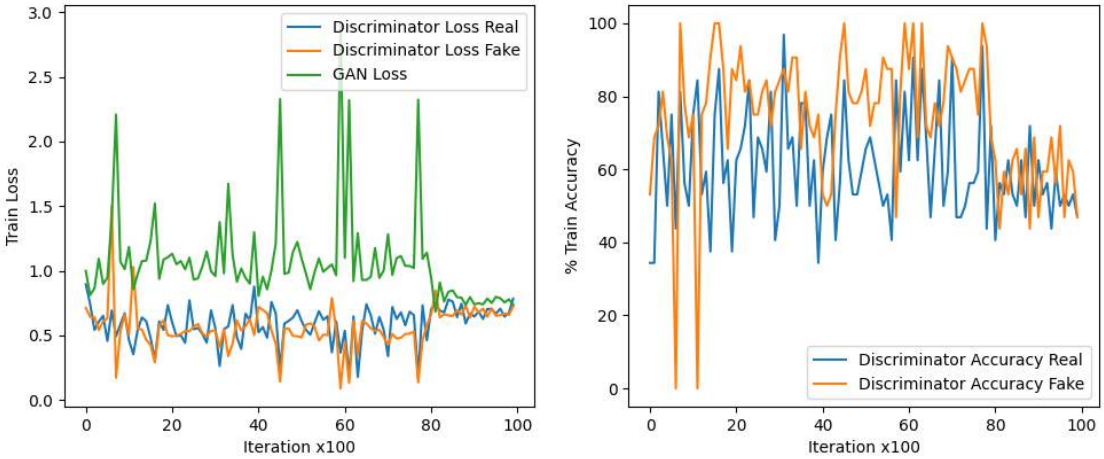


Figure 4.11

Training Performance of Decay Rate = 0.1

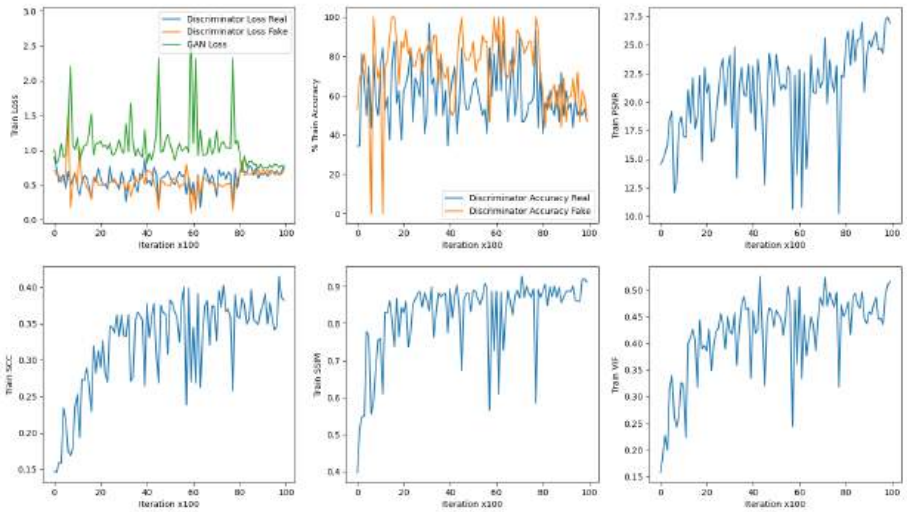
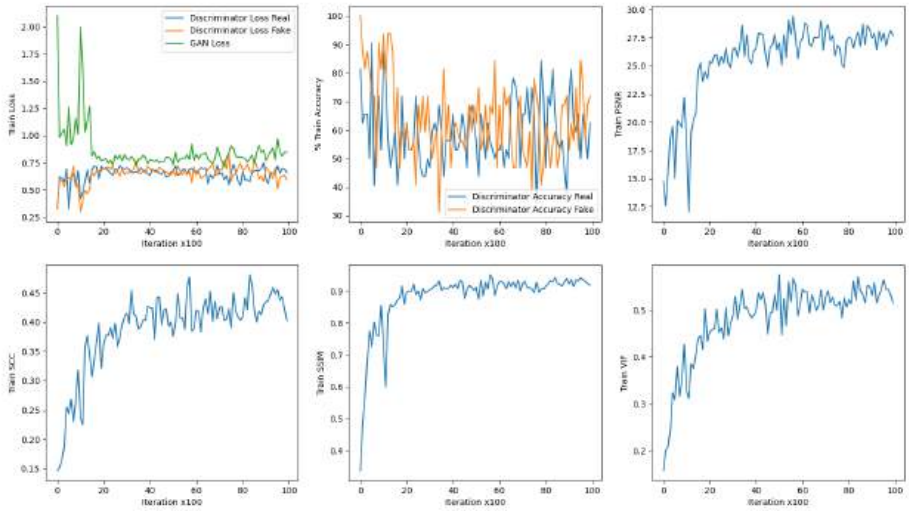


Figure 4.12

Training Performance of Decay Rate = 0.01



4.3.7 ADAM Optimizer Adjustment

An optimizer used in this work is an Adaptive Moment Estimation (ADAM) optimizer (*Adam Optimizer in Tensorflow.*, 2023). A β_1 is one of the hyperparameters and adjustable value. It is the initial decay rate used when estimating the first moment of the gradient while training, which is multiplied at the end of each training step or batch. Decreasing β_1 will slow down a training process and increasing a value will result in the opposite way. This value needs to be adjusted according to batch size. Normally, a large batch size will result in faster learning and a small batch will result in slower learning. Once using a very large or very low batch size could lead to non-optimal learning, adjusting this β_1 value can help in these situations. In this showing case, using β_1 = 0.5 (default is 0.9) is a middle ground that matches the other training values used in this learning process and results in the best balance point between training speed and performance. However, each training may also require tuning and has its own optimal β_1 value.

4.3.8 Best Saved Weight Selection

Picking the right weight from the best-fit point is very crucial here since the system would not ever perform well from a bad weight. So using the right weight makes the

Figure 4.13

Training Performance of $\beta_1 = 0.1$

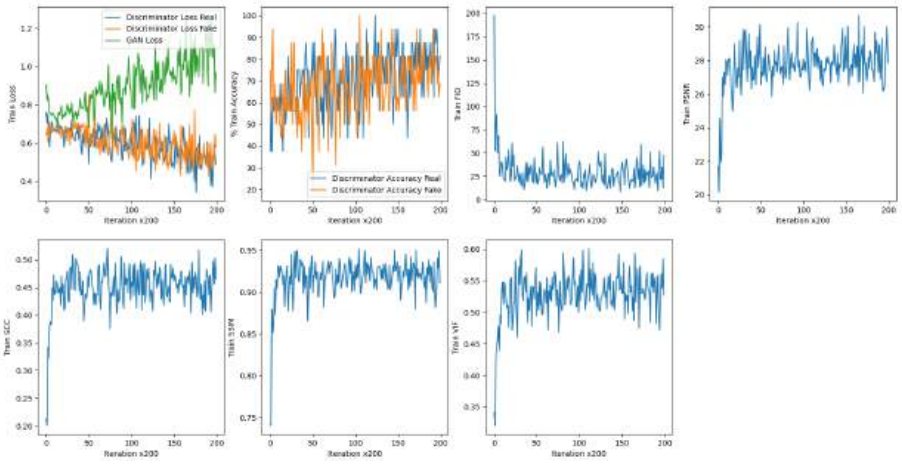


Figure 4.14

Training Performance of $\beta_1 = 0.5$

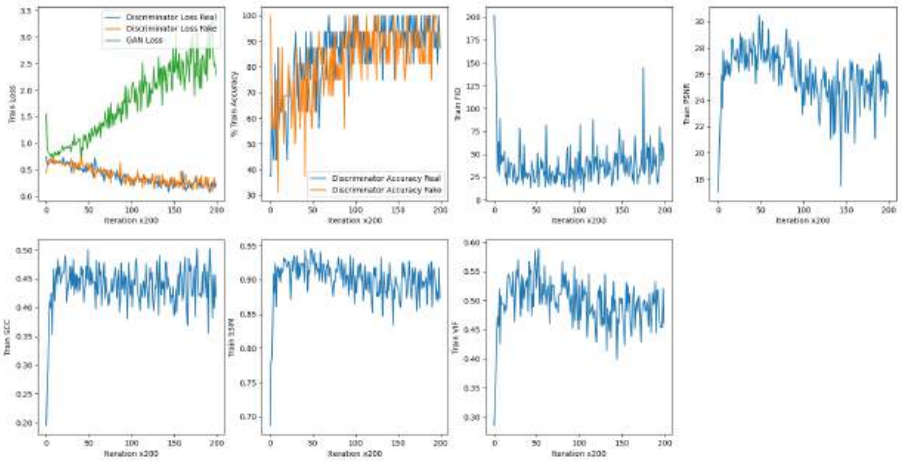
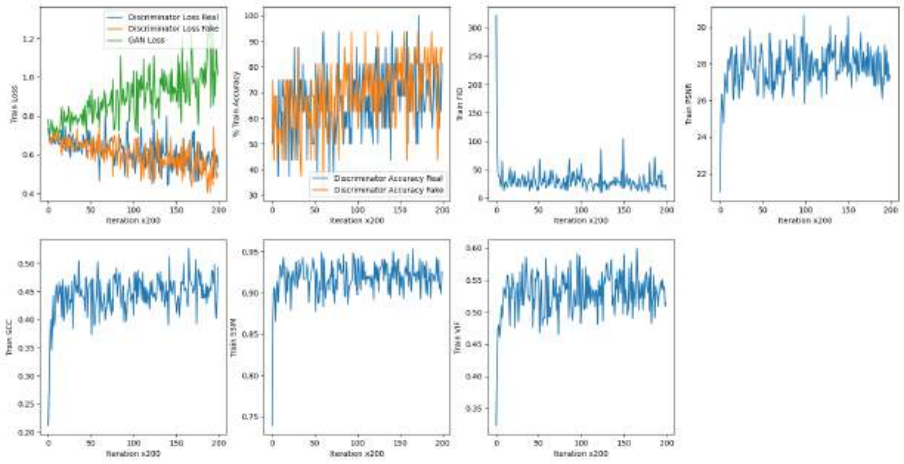


Figure 4.15

Training Performance of $\beta_1 = 0.9$



system perform at its finest level. This is a comparison between picking the right saved weight and unfit weight [Fig.4.16a-Fig.4.16b].

4.4 LPRGAN Testing Result

This section presents all testing results from the recovery system including several situations. The separated unseen images are collected for testing purposes. This is an effective way to measure its performance on a real world scale.

Figure 4.16

Result on Different Weight Selection - (a) Fit Weight (SSIM 0.787) and (b) Unfit Weight (SSIM 0.465)

(a) **(b)**

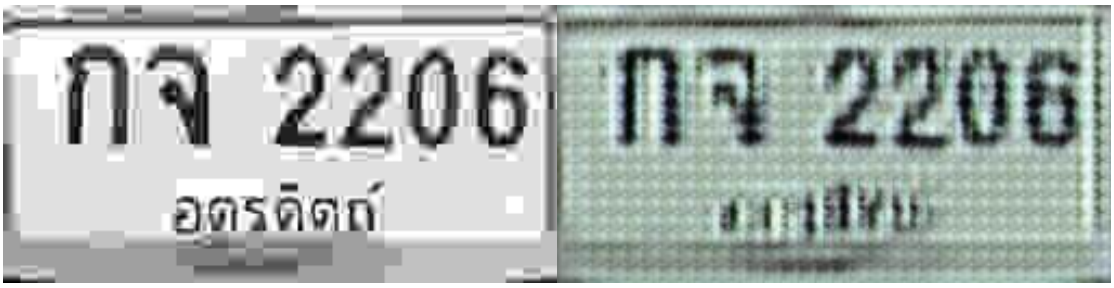
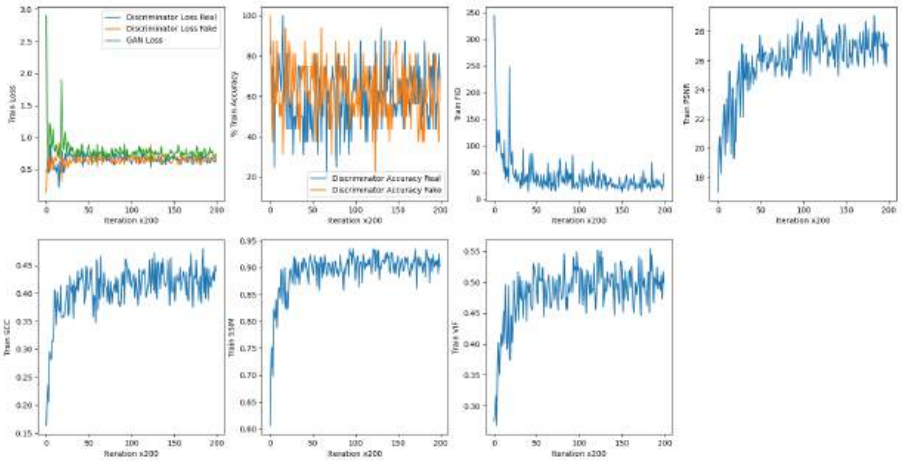


Figure 4.17

Overall Training Performance on Low Bitrate Problem



4.4.1 Low Bitrate Problem

This subsection shows the LPRGAN testing results in low bitrate conditions. Overall performance graphs are displayed in Fig.4.17-Fig.4.18.

In this case, a poor-quality version of the reference image is the input image. The input image can be found in Fig.4.19a[2]. Output results from each network are shown in Fig.4.19b-Fig.4.19g respectively. As a result, the CBDNet, the GFPGAN-SR, and the SwinIR do not work in this case, but the LPRGAN can recover most of the lost data in input images, especially in low-detail areas like a grey plate background. At the same time, a convolutional autoencoder has a smooth output but fails to remove a blocky artifact, and the GAN+U-Net has a not-so-sharp image.

Here is an example of using the LPRGAN on a low bitrate video, an actual use case, instead of a JPG compression image. A video was recorded with a very low bitrate, 8kbps, H264 format. It was also resized to a 256x128 frame size, matching a proposed network configuration. Unfortunately, there is no original/reference image to compare with the output in an actual situation, so only the input [Fig.4.20a] and outputs [Fig.4.20b-Fig.4.20g] results are available. When zoomed in, all results [Fig.4.21a-Fig.4.21f] are

[2]Some PDF viewers (i.e., Preview in macOS) have an antialiasing feature that smooths out a rough rendered object in a document. In order to see raw result images, this feature must be turned off.

Figure 4.18

Overall Evaluating Performance on Low Bitrate Problem

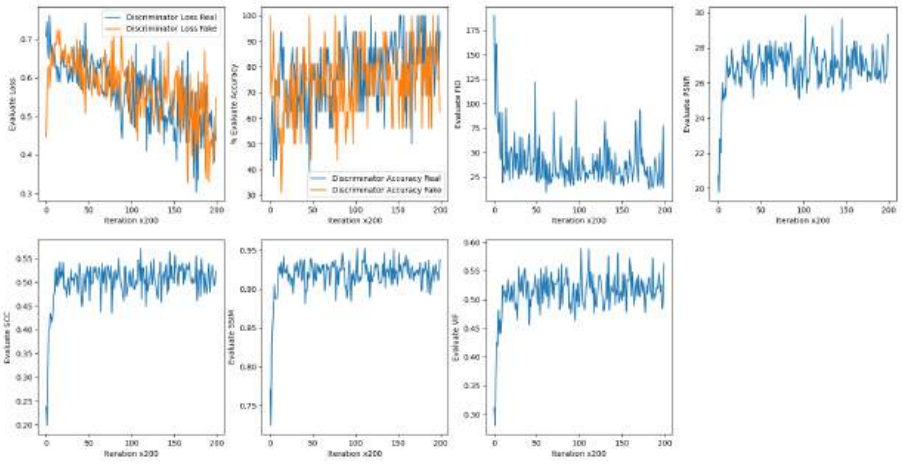


Figure 4.19

Result on Simulated Low Bitrate Problem - (a) Input Image, (b) CBDNet Output Image, (c) GFPGAN-SR Output Image, (d) Convolutional Autoencoder Output Image, (e) GAN+U-Net Output Image, (f) SwinIR Output Image and (g) LPRGAN Output Image

(a) **(b)** **(c)** **(d)**

(e) **(f)** **(g)**

Figure 4.20

Result on Actual Low Bitrate Video Problem - (a) Input Image, (b) CBDNet Output Image, (c) GFPGAN-SR Output Image, (d) Convolutional Autoencoder Output Image, (e) GAN+U-Net Output Image, (f) SwinIR Output Image and (g) LPRGAN Output Image

visible, showing that the LPRGAN gives out the best result over the rest. It produces more detail and contrasts. Although the SwinIR could remove compression artifact, it also removes some fine detail from the image, making it looks less sharp. The GFPGAN-SR has an aspect ratio distortion due to the nature of the super-resolution technique. The rest of the outputs are dull and blurry.

4.4.2 Low Light Problem

This subsection demonstrates two types of testing images in a low-light situation. One simulates a low light by reducing image brightness, and the other captures an actual low light nighttime. Overall performance graphs are displayed in Fig.4.22-Fig.4.23.

In the simulation [Fig.4.24a], both the LPRGAN and the GAN+U-Net show improved brightened images but not the rest. In addition, the GAN+U-Net output has some yellow tint and is not as sharp whereas the MIRNet and the LPRGAN produce the correct color temperature results. Fig.4.24b-Fig.4.24g show all outputs. In the real-world scene test [Fig.4.25a], only the GAN+U-Net, the MIRNet, and the LRPGAN outputs have improved brightness from the input but the GAN+U-Net also has a weird tint in the output, only the MIRNet and the LPRGAN produce the most realistic and corrected color tone images [Fig.4.25b-Fig.4.25g]. Ultimately, the LPRGAN and the MIRNet are the first and second candidates in low light recovery, but the CBDNet, the GFPGAN-SR, and a

Figure 4.21

Result on Actual Low Bitrate Video Problem with 3X Zoom on Last 3 Digits - (a) CBDNet Output Image, (b) GFPGAN-SR Output Image, (c) Convolutional Autoencoder Output Image, (d) GAN+U-Net Output Image, (e) SwinIR Output Image and (f) LPRGAN Output Image

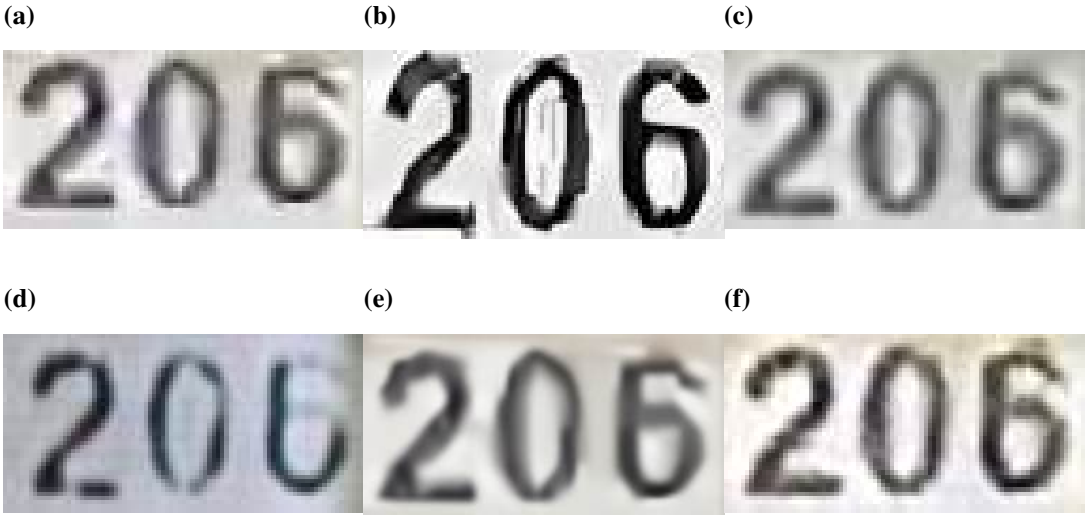


Figure 4.22

Overall Training Performance on Low Light Problem

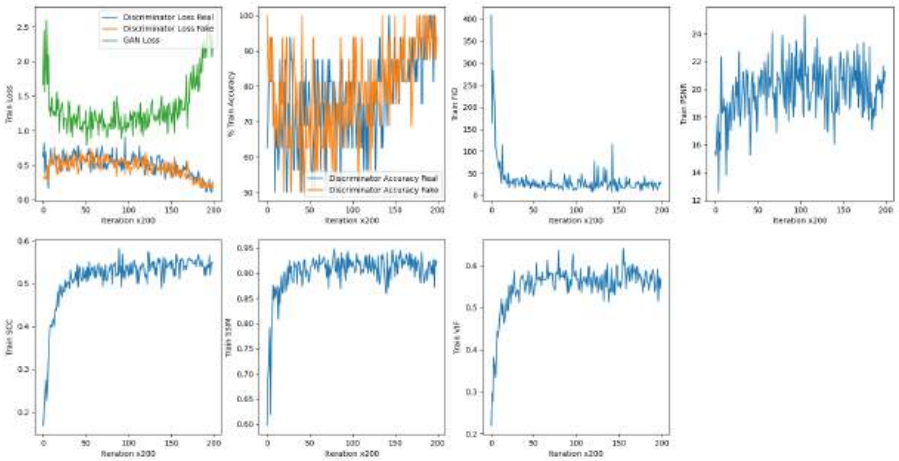


Figure 4.23

Overall Evaluating Performance on Low Light Problem

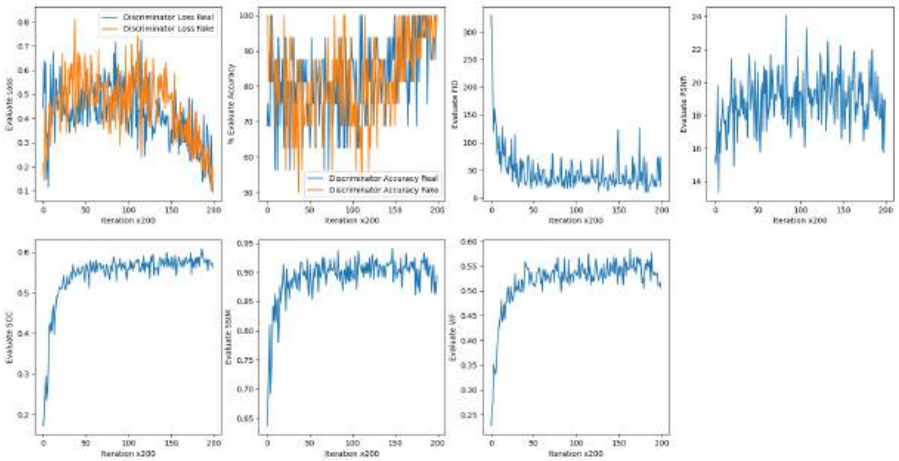


Figure 4.24

Result on Simulated Low Light Problem - (a) Input Image, (b) CBDNet Output Image, (c) GFPGAN-SR Output Image, (d) Convolutional Autoencoder Output Image, (e) GAN+U-Net Output Image, (f) MIRnet Output Image and (g) LPRGAN Output Image

(a) **(b)** **(c)** **(d)** **(e)** **(f)** **(g)**

convolutional autoencoder failed completely.

4.4.3 1-Axis Motion Blur Problem

These are examples of solving one-directional motion blur problems. Both horizontal (0 degrees) and vertical (90 degrees) motion blur were studied in this research. Overall performance graphs are displayed in Fig.4.26-Fig.4.29.

A blurred input image in the horizontal blur problem is shown in Fig.4.30a for a simulated test case. Simulation output results for horizontal blur demonstrated from each network are shown in Fig.4.30b-Fig.4.30h. However, in order to get actual motion blur images, a high-speed panning camera in left-right directions creates a horizontal blur [Fig.4.31a]. This action is a much safer measurement than driving a car speeding to-

Figure 4.25

Result on Actual Low Light Problem - (a) Input Image, (b) CBDNet Output Image, (c) GFPGAN-SR Output Image, (d) Convolutional Autoencoder Output Image, (e) GAN+U-Net Output Image, (f) MIRnet Output Image and (g) LPRGAN Output Image

(a) **(b)** **(c)** **(d)** **(e)** **(f)** **(g)**

Figure 4.26

Overall Training Performance on Horizontal Motion Blur Problem

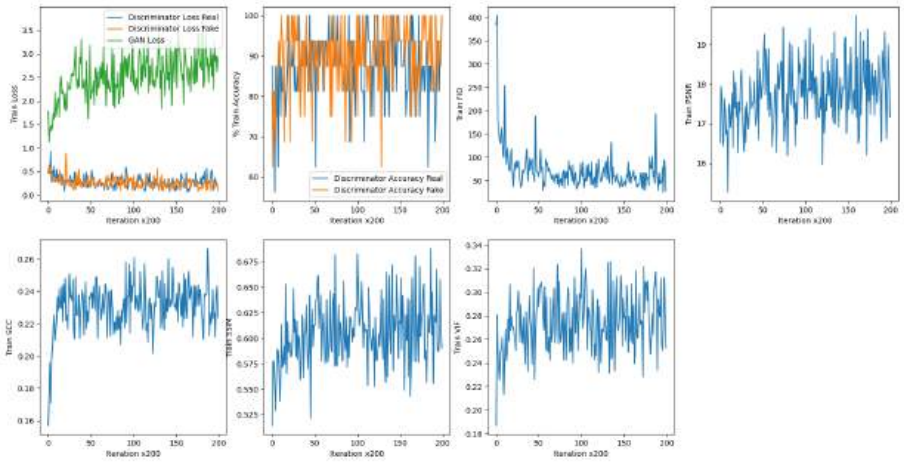


Figure 4.27

Overall Evaluating Performance on Horizontal Motion Blur Problem

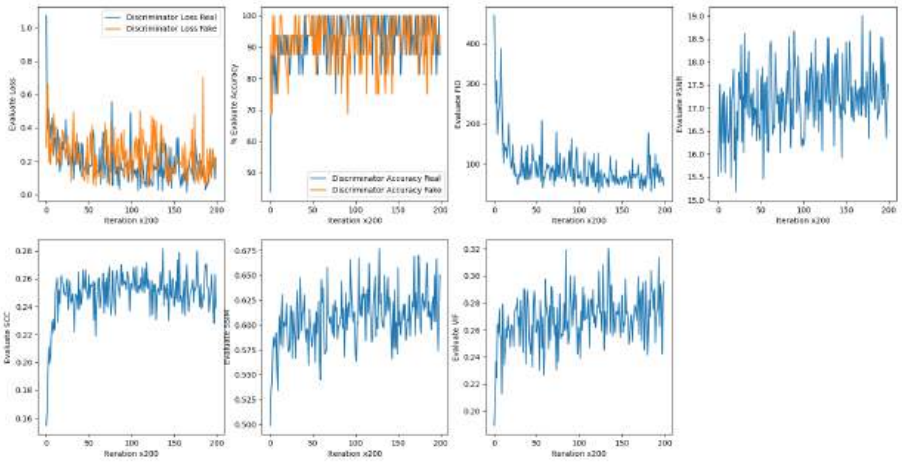


Figure 4.28

Overall Training Performance on Vertical Motion Blur Problem

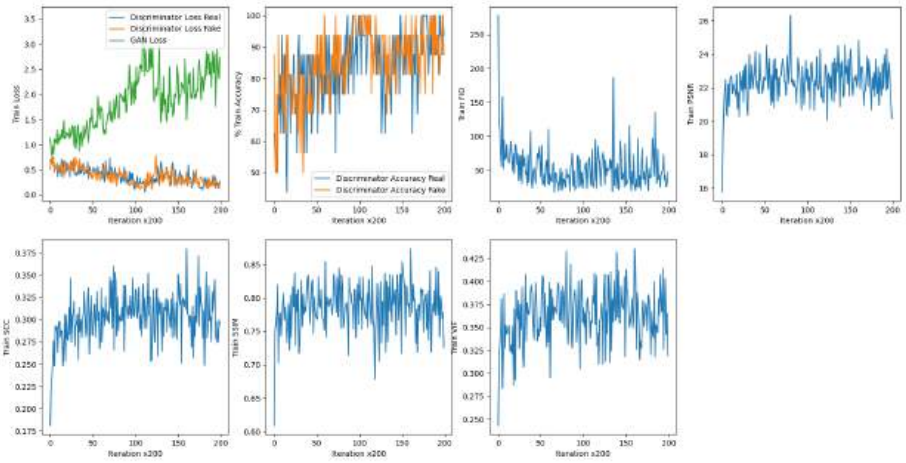
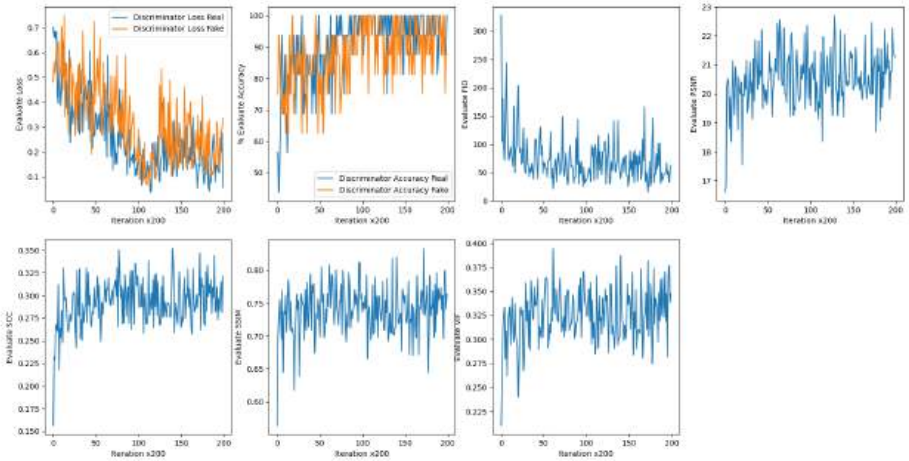


Figure 4.29

Overall Evaluating Performance on Vertical Motion Blur Problem



ward a camera. The actual case output results for horizontal blur demonstrated from each network are shown in Fig.4.31b-Fig.4.31h. Also, in a vertical motion blur problem, a simulated vertical blur image is in Fig.4.32a, and these Fig.4.32b-Fig.4.32h are the output. Once again, in order to get an actual vertical motion blur to quickly pan a camera in up-down directions to create a vertical blur [Fig.4.33a]. The outputs are shown in Fig.4.33b-Fig.4.33h. In the end, the DeblurGANv2 and the Restormer can fix a motion blur problem only in a simulation case but not a real-world one, and only the LPRGAN can recover blurred images in both cases.

4.4.4 Out Of Focus Problem

This problem can be seen when an object is not in a camera focus range. Typically, any camera would have a specific focus range at any single time. This range can be shifted around when changing the camera focal length or aperture. A high focal length when paired with a large aperture value will result in a narrow focus range. Thus, once a camera does not focus at the right place there is a large chance that the object is likely to be out of focus. Overall performance graphs are displayed in Fig.4.34-Fig.4.35.

Here demonstrate two types of out-of-focus testing images, one is simulated using a blur filter and another is an actual out-of-focus image achieved by forced focusing in front of a license plate location. Results from the CBDNet, the GFPGAN, a convolutional

Figure 4.30

Result on Simulated Horizontal Motion Blur Problem - (a) Input Image, (b) CBDNet Output Image, (c) GFPGAN-SR Output Image, (d) Convolutional Autoencoder Output Image, (e) GAN+U-Net Output Image, (f) DeblurGANv2+MobileNet Output Image, (g) Restormer Output Image and (h) LPRGAN Output Image

Figure 4.31

Result on Actual Horizontal Motion Blur Problem - (a) Input Image, (b) CBDNet Output Image, (c) GFPGAN-SR Output Image, (d) Convolutional Autoencoder Output Image, (e) GAN+U-Net Output Image, (f) DeblurGANv2+MobileNet Output Image, (g) Restormer Output Image and (h) LPRGAN Output Image

Figure 4.32

Result on Simulated Vertical Motion Blur Problem - (a) Input Image, (b) CBDNet Output Image, (c) GFPGAN-SR Output Image, (d) Convolutional Autoencoder Output Image, (e) GAN+U-Net Output Image, (f) DeblurGANv2+MobileNet Output Image, (g) Restormer Output Image and (h) LPRGAN Output Image

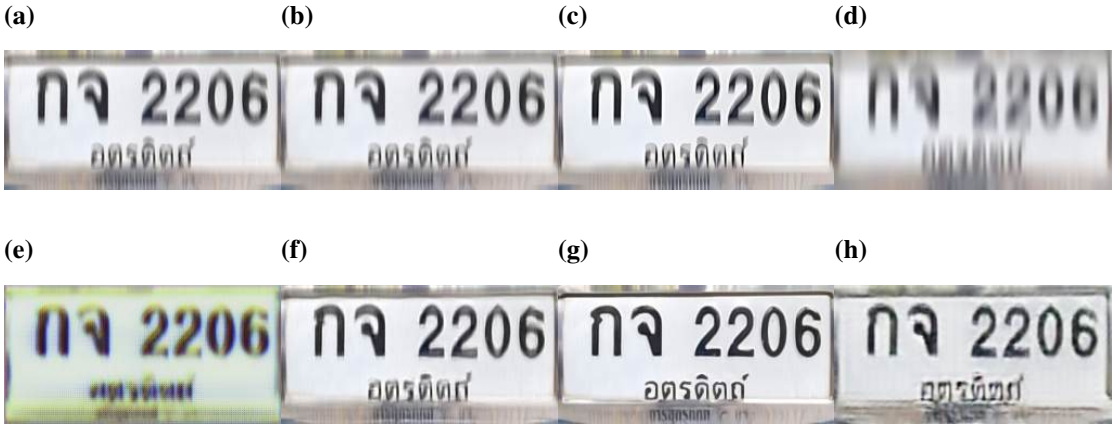


Figure 4.33

Result on Actual Vertical Motion Blur Problem - (a) Input Image, (b) CBDNet Output Image, (c) GFPGAN-SR Output Image, (d) Convolutional Autoencoder Output Image, (e) GAN+U-Net Output Image, (f) DeblurGANv2+MobileNet Output Image, (g) Restormer Output Image and (h) LPRGAN Output Image

Figure 4.34

Overall Training Performance on Out Of Focus Problem

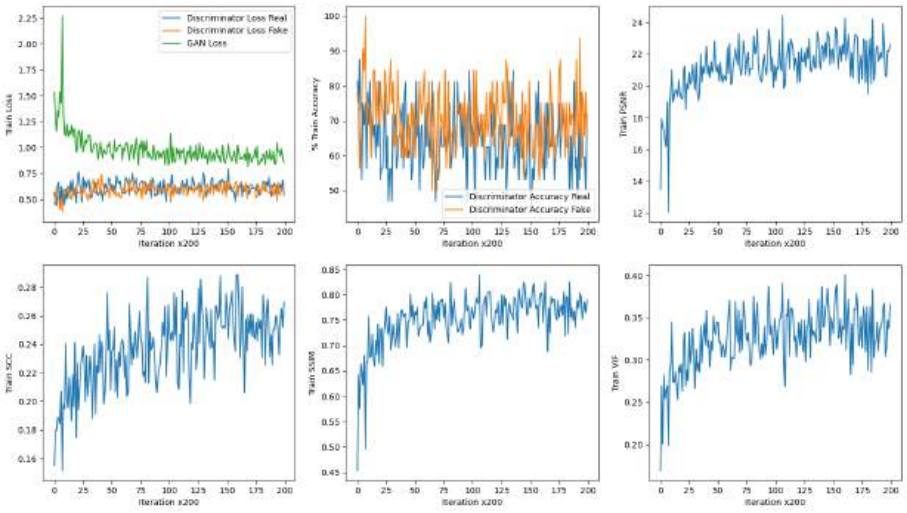


Figure 4.35

Overall Evaluating Performance on Out Of Focus Problem

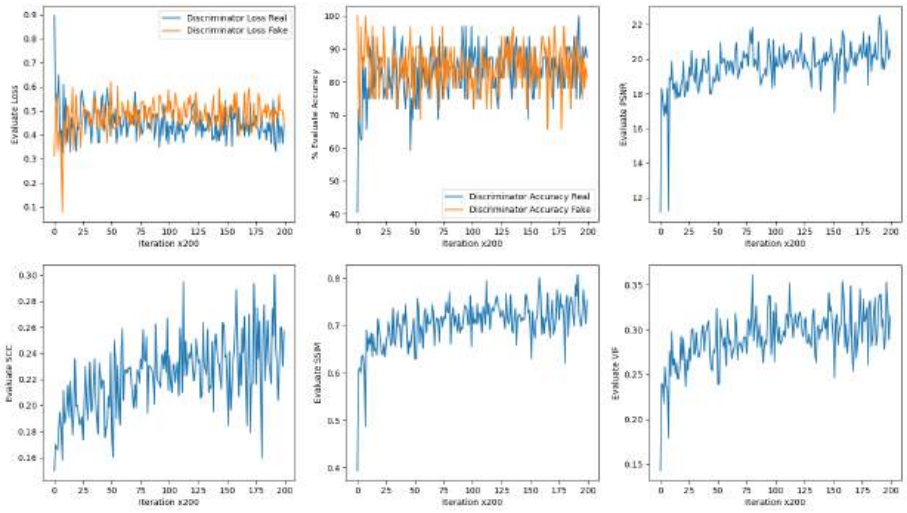


Figure 4.36

Result on Simulated Out of Focus Problem - (a) Input Image, (b) CBDNet Output Image, (c) GFPGAN-SR Output Image, (d) Convolutional Autoencoder Output Image, (e) GAN+U-Net Output Image and (f) LPRGAN Output Image

(a) **(b)** **(c)** **(d)** **(e)** **(f)**

Figure 4.37

Result on Actual Out of Focus Problem - (a) Input Image, (b) CBDNet Output Image, (c) GFPGAN-SR Output Image, (d) Convolutional Autoencoder Output Image, (e) GAN+U-Net Output Image and (f) LPRGAN Output Image

(a) **(b)** **(c)** **(d)** **(e)** **(f)**

autoencoder, and the U-Net are not good. Only the LPRGAN model can fix an out-of-focus problem. The Out of Focus results are shown in Fig.4.36a-Fig.4.37f.

4.4.5 International License Plate Test

Although the current LPRGAN model has been trained on the Thai license plate dataset without knowing other countries' plate appearances, this model can still recover a poor-quality plate at a reasonable level thanks to its generalization. Of course, its performance would not be near a Thai license plate recovery on which it was trained, but it can be solved by retraining with a target country dataset. These figures [Fig.4.38a-Fig.4.38b] show a US (*Jeff's License Plates. Jeffsplates.*, 2022) and UK (*UK European License Plate. European License Plates.*, 2022) license plate samples.

4.4.6 Metric Measurements Result

There are two types of measurement in this research. First, using an image qualifier that gives out a Star Rating, and second, mathematical measurements (Rating score is an output from classifier both detector and qualifier. These two models have an accuracy of over 99%. All mathematical measurement values were computed with a pair of original/reference and distorted images. All delta values in each case were calculated

Figure 4.38

Original US and UK License Plates - (a) US and (b) UK

(a) **(b)**

Figure 4.39

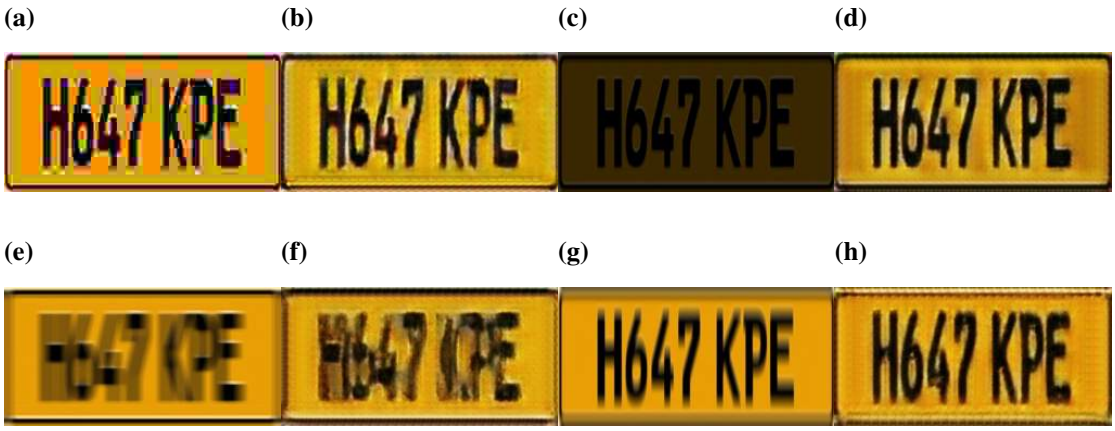
Result on US Plate from LPRGAN (a) Low Bitrate Input Image, (b) Low Bitrate Output Image, (c) Low Light Input Image, (d) Low Light Output Image, (e) Horizontal Motion Blur Input Image, (f) Horizontal Motion Blur Output Image, (g) Vertical Motion Blur Input Image and (h) Vertical Motion Blur Output Image

(a) **(b)** **(c)** **(d)**

(e) **(f)** **(g)** **(h)**

Figure 4.40

Result on UK Plate from LPRGAN (a) Low Bitrate Input Image, (b) Low Bitrate Output Image, (c) Low Light Input Image, (d) Low Light Output Image, (e) Horizontal Motion Blur Input Image, (f) Horizontal Motion Blur Output Image, (g) Vertical Motion Blur Input Image and (h) Vertical Motion Blur Output Image



against each own low-quality image. However, it is impossible to compute these metrics values in actual test cases due to a lack of reference material. In the motion blur problem case, HB is a horizontal blur, and VB is a vertical blur.) including five metrics (FID, PSNR, SCC, SSIM, and VIF) and three synthetic benchmarks (file size, training time usage per epoch, and recovery render speed). A proposed model tested up against other approaches[3] is presented in this subsection.

A quality rating test using an image qualifier shows that the LPRGAN has no problem fixing poor-quality input images. It generated a 5-Star output image in the low bitrate case and normal-looking images in the rest of the test cases. At the same time, U-Net architecture with GAN also did a reasonable job in most cases. However, a convolutional autoencoder did not perform any good. The reason behind this measurement is that all mathematical measurements do not work when there is no reference image (its counterpart) to calculate a value in an actual situation. In reality, only a degraded image is presented.

[3]All methods experimented in this study were under the same environment and parameters, such as the same computer machine, an equal number of iterations, learning rate, batch size, and the same input image under each test. Also, all images used in this study were encoded at 100% Quality to avoid a compression loss, revealing an actual image data size.

Next, a visual quality inspection using mathematical calculation tests shows that the GAN+U-Net is superior to a convolutional autoencoder. However, the LPRGAN has the best outcome in most cases among convolutional autoencoder and GAN+U-Net approaches. As in a low bitrate scenario [Table.4.3], the LPRGAN generated the biggest output file size. It also has the highest (as high as SwinIR) SSIM score on low bitrate recovery tests. The file size value is essential here since they tell how much an image holds a piece of information. The bigger the file size, the more fine detail is generated. SSIM score indicates that the reconstruction image has a similar structure to the original. These outcomes prove that the LPRGAN helps predict lost data back, producing a richer detailed image. In a low light situation [Table.4.4], the LPRGAN beats out the other models on SSIM score, but when comparing the LPRGAN to the MIRNet in this scenario (MIRNet cannot reconstruct other cases, only a low light case), a competitor has three out of five metrics (FID, SCC, and VIF) score higher than the LPRGAN. Even though the LPRGAN has only two metrics (PSNR and SSIM) wins over the MIRNet. Because of the low light situation, these PSNR and SSIM metrics are crucial. Since PSNR is a Power Signal to Noise Ratio, the more PSNR is, the more power signal and the lesser noise, producing a cleaner image, and SSIM is also calculated based on image luminosity factors meaning that SSIM is high when the output has a structural and luminosity close to an original one. A higher SSIM is a brighter image because a reference is bright. Thus, the LPRGAN output has a lower noise yet a brighter image as the result. In motion blur cases [Table.4.5], the LPRGAN has SCC and file size (in horizontal blur case) metrics higher than the rest. The SCC is the concerning metric here since it is designed to detect any pixel location shifting, which is suitable for this case. A better SCC value means the lesser pixel is shifted from a reference image. In other words, the lesser blurry image. In the last case, LPRGAN can recover a simulated out-of-focus problem to a normal-looking image with 4 out of 6 metrics wins. In the actual out-of-focus problem, LPRGAN also has the biggest output file size.

On the other hand, US and UK were chosen for this international plate test. The result from Table.4.7 shows that every problem case passes a visual inspection test. Even though the LPRGAN could not always produce better metric values than the original input, all generated output always contains more data than the original ones by looking at the file size metric. The outputs from the LPRGAN surprisingly have consistent file sizes on every problem test in a margin of ± 1.4 kB.

Move over to a speed test, a speed comparison between each approached method [Table.4.8] shows that although a convolutional autoencoder is the fastest in an FPS count due to a non-GAN design it does not produce an output well at all. The CBDNet is also a non-GAN design but it performs slowly at the same level as GAN-based designs. In contrarily, comparing GAN-based models, results show that the LPRGAN is the fastest in every render speed test. When compared to the GAN+U-Net approach in 256x128 resolution, the LPRGAN is 2.93X speedup in training speed and more than twice as fast (2.42X) in a render test. When compared to the Restomer and the SwinIR in the render test, the LPRGAN is 1.77X and 9.32X speedup. At a higher resolution (600x400), comparing the LPRGAN to the MIRNet, results are as expected because the MIRNet has a much bigger network (one complex parallel stream vs. one simplified serial stream) and is very limited in resolution (only producing 600x400 output), so the LPRGAN is way faster than the MIRNet by 25.26X in the render test while the DeblurGANv2+MobileNet is a relatively compact network, there is still a gap between itself and the LPRGAN in render speed (1.71X difference), while the GAN+U-Net and the Restomer are dropped behind the LPRGAN. The same story at HD resolution, the LPRGAN is slightly faster than the DeblurGANv2+MobileNet (1.36X) in the same test scenario due to fewer parameter counts. The GAN+U-Net is even lacking behind these two GAN-based models while the SwinIR could not finish a test due to painfully low speed. This makes the LPRGAN the fastest.

Furthermore, there are three videos used in the LPRGAN speed on a different type of processor test [Table.4.9]. The first video is 21.5 seconds long, 932 frames, full-length (100% degradation) degraded video file named Video#1. The second and the third test case are 21.5 seconds long, 932 frames, half-length (50%, 466 frames degradation) degraded video file and 21.5 seconds long, 932 frames, 25% length (233 frames degradation) degraded video files, named Video#2 and Video#3. This test represents real-world usage in a mixing environment because not every video frame would be degraded all the time, so running recovery on a good frame is a waste. This test shows a difference between the plain LPRGAN and the adaptive LPRGAN (detection+recovery) performance in action. However, the adaptive LPRGAN did not actually double the frame rate in Video#2 even though a test video contains only half degraded frames due to an additional detection workload, but it is still close to doubling a frame rate than an unequipped detection system by 1.71X and 3.3X in Video#2 and #3 on ultra-low power CPU, 1.76X

and 3.4X in Video#2 and #3 on workstation CPU and 1.7X and 2.92X in Video#2 and #3 on a single GPU.

The last one is the memory usage test [Table.4.10] (unit in MB), this test is a measurement of memory usage on a single frame recovery from each method. Separated into two groups, a non-GAN and a GAN based. Although, non-GAN methods tend to use less RAM as result in this article shows that they do not work well in many situations. On the other hand, the LPRGAN still uses the least amount of RAM in GAN based group. Thus, it is possible to be used in an edge computing device where it has a limited amount of RAM.

4.4.7 Real World License Plates Test

These are real-world samples [Fig.4.41a-Fig.4.41t] recovery using the proposed system. However, these plates in this subsection do not belong to the author, so to protect their owner's privacy, they cannot be exposed to the full license plate area (*Use of Disclosure of Personal Data, Section 24 and 27. Thai Government Gazette.*, 2022). The data recovery result in Fig.4.42 shows that in every degraded type, average file sizes have gained more data after a recovery in low bitrate, low light, horizontal blur, vertical blur, and out of focus situations at rate 1.51X, 1.25X, 1.61X, 1.63X, and 2.06X, respectively.

Difficult Real World Situation

In extremely poor input images where input information is too much distorted, a generated result could be confusing as in Fig.4.41k-4.41l. This problem comes from the generator applying deblurring aggressively from its learning which can cause misleading information. It can be solved by selecting a different weight from a lower number of iterations where the generator is not overpowered by the discriminator. When comparing the LPRGAN to both the DeblurGANv2+MobileNet and the Restormer, only the LPRGAN output produces a sharper result while the rest do not deblur well since those remain unsharp in this real license plate test.

4.4.8 Real World License Plate Recognition Test

This test is another aspect of the LPRGAN helping license plate recognition. A recognizer is a recognition system similar to a classifier, a typical VGG-16 network found

Table 4.3

A Metric Measurement Score Table on Low Bitrate Problem

Method	**Rating**	**FID (△FID)**	**PSNR (△PSNR)**	**SCC (△SCC)**	**SSIM (△SSIM)**	**VIF (△VIF)**	**File Size (△File Size)**
Original/ Baseline	5-Star	0.0	48.0	1.0	1.0	1.0	31.7
Simulated Low Bitrate	1-Star	239.07	20.66	0.202	0.763	0.271	7.9
Conv. Autoencoder	3-Star	188.67 (-50.4)	20.67 (+0.01)	0.245 (+0.043)	0.776 (+0.013)	0.273 (+0.002)	12.0 (+4.1)
CBDNet	1-Star	220.64 (-18.43)	20.79 (+0.13)	0.209 (+0.007)	0.765 (+0.002)	0.273 (+0.002)	10.3 (+2.4)
GFPGAN-SR	3-Star	160.72 (-78.35)	20.01 (-0.65)	0.338 (+0.136)	0.784 (+0.021)	0.303 (+0.032)	22.6 (+14.7)
GAN+U-Net	5-Star	184.30 (-54.77)	16.61 (-4.05)	0.158 (-0.044)	0.669 (-0.094)	0.195 (-0.076)	22.2 (+14.3)
SwinIR	2-Star	206.46 (-32.61)	21.29 (+0.37)	0.261 (+0.059)	0.787 (+0.024)	0.293 (+0.022)	18.3 (+10.4)
LPRGAN	**5-Star**	**102.15 (-136.92)**	**20.81 (+0.15)**	0.302 (+0.100)	**0.787 (+0.024)**	0.285 (+0.014)	**27.5 (+19.6)**

Method	**Rating**	**FID (△FID)**	**PSNR (△PSNR)**	**SCC (△SCC)**	**SSIM (△SSIM)**	**VIF (△VIF)**	**File Size (△File Size)**
Actual Low Bitrate	3-Star	-	-	-	-	-	16.5
Conv. Autoencoder	4-Star	-	-	-	-	-	18.5 (+2.0)
CBDNet	3-Star	-	-	-	-	-	16.7 (+0.2)
GFPGAN-SR	5-Star	-	-	-	-	-	23.0 (+6.5)
GAN+U-Net	5-Star	-	-	-	-	-	22.9 (+6.4)
SwinIR	5-Star	-	-	-	-	-	18.2 (+1.7)
LPRGAN	**5-Star**	-	-	-	-	-	**25.9 (+9.4)**

Table 4.4

A Metric Measurement Score Table on Low Light Problem

Method	**Rating**	**FID (△FID)**	**PSNR (△PSNR)**	**SCC (△SCC)**	**SSIM (△SSIM)**	**VIF (△VIF)**	**File Size (△File Size)**
Original/ Baseline	Normal	0.0	48.0	1.0	1.0	1.0	31.7
Simulated Low Light	Low Light	179.30	3.33	0.835	0.128	0.395	15.4
Conv. Autoencoder	Low Light	190.14 (+10.84)	2.36 (-0.97)	0.087 (-0.748)	0.024 (+0.104)	0.078 (-0.317)	7.4 (-8.0)
CBDNet	Low Light	126.43 (-52.87)	3.31 (-0.02)	0.512 (-0.323)	0.120 (-0.008)	0.330 (-0.065)	9.9 (-5.5)
GFPGAN-SR	Low Light	122.86 (-56.44)	3.31 (-0.02)	0.586 (-0.249)	0.124 (-0.004)	0.351 (-0.044)	12.8 (-2.6)
GAN+U-Net	Low Light	162.18 (-17.12)	8.60 (+5.27)	0.332 (-0.503)	0.604 (+0.476)	0.301 (-0.094)	23.8 (+8.4)
MIRNet	Normal	70.73 (-108.57)	8.66 (+5.33)	0.781 (-0.054)	0.642 (+0.514)	0.470 (+0.075)	25.0 (+9.6)
LPRGAN	**Normal**	102.99 (-76.31)	**11.16 (+7.83)**	0.330 (-0.505)	**0.729 (+0.601)**	0.313 (-0.082)	24.3 (+8.9)

Method	**Rating**	**FID (△FID)**	**PSNR (△PSNR)**	**SCC (△SCC)**	**SSIM (△SSIM)**	**VIF (△VIF)**	**File Size (△File Size)**
Actual Low Light	Low Light	-	-	-	-	-	24.5
Conv. Autoencoder	Low Light	-	-	-	-	-	11.7 (-12.8)
CBDNet	Low Light	-	-	-	-	-	18.4 (-6.1)
GFPGAN-SR	Low Light	-	-	-	-	-	23.0 (-1.5)
GAN+U-Net	Normal	-	-	-	-	-	31.1 (+6.6)
MIRNet	Normal	-	-	-	-	-	29.5 (+5.0)
LPRGAN	**Normal**	-	-	-	-	-	**32.5 (+8.0)**

Table 4.5

A Metric Measurement Score Table on Motion Blur Problem

Method	**Rating**	**FID (△FID)**	**PSNR (△PSNR)**	**SCC (△SCC)**	**SSIM (△SSIM)**	**VIF (△VIF)**	**File Size (△File Size)**
Original/ Baseline	Normal	0.0	48.0	1.0	1.0	1.0	31.7
Simulated Horizontal Blur	HB	138.53	16.90	0.059	0.666	0.162	20.8
Conv. Autoencoder	HB	273.42 (+134.89)	14.55 (-2.35)	0.046 (-0.013)	0.458 (-0.208)	0.076 (-0.086)	14.8 (-6.0)
CBDNet	HB	153.14 (+14.61)	16.90 (+0.0)	-0.034 (-0.093)	0.665 (-0.001)	0.162 (+0.0)	16.3 (-4.5)
GFPGAN-SR	HB	192.12 (+53.59)	16.38 (-0.52)	-0.021 (-0.080)	0.660 (-0.006)	0.156 (-0.010)	21.2 (+0.4)
GAN+U-Net	Normal	263.96 (+125.43)	9.22 (-7.68)	0.088 (+0.029)	0.426 (-0.240)	0.113 (-0.049)	26.4 (+5.6)
Deblur-GANv2-+Mobile-Net	Normal	121.03 (-17.5)	18.41 (+1.51)	0.055 (-0.004)	0.764 (+0.098)	0.217 (+0.055)	20.3 (-0.5)
Restormer	Normal	88.73 (-49.8)	19.19 (+2.29)	0.085 (+0.026)	0.800 (+0.134)	0.240 (+0.78)	23.6 (+2.8)
LPRGAN	**Normal**	128.49 (-10.04)	17.82 (+0.92)	**0.180 (+0.121)**	0.740 (+0.074)	0.236 (+0.074)	**28.2 (+7.4)**

Method	**Rating**	**FID (△FID)**	**PSNR (△PSNR)**	**SCC (△SCC)**	**SSIM (△SSIM)**	**VIF (△VIF)**	**File Size (△File Size)**
Actual Horizontal Blur	HB	-	-	-	-	-	22.2
Conv. Autoencoder	HB	-	-	-	-	-	15.3 (-6.9)
CBDNet	HB	-	-	-	-	-	18.2 (-4.0)
GFPGAN-SR	HB	-	-	-	-	-	23.4 (+1.2)
GAN+U-Net	Normal	-	-	-	-	-	26.7 (+4.5)
Deblur-GANv2-+Mobile-Net	HB	-	-	-	-	-	19.3 (-2.9)
Restormer	HB	-	-	-	-	-	23.1 (+0.9)
LPRGAN	**Normal**	-	-	-	-	-	**28.9 (+6.7)**

Method	**Rating**	**FID (△FID)**	**PSNR (△PSNR)**	**SCC (△SCC)**	**SSIM (△SSIM)**	**VIF (△VIF)**	**File Size (△File Size)**
Simulated Vertical Blur	VB	217.31	16.99	0.090	0.649	0.155	21.2
Conv. Autoencoder	VB	446.92 (+229.61)	14.50 (-2.49)	-0.028 (-0.118)	0.414 (-0.235)	0.068 (-0.087)	15.4 (-5.8)
CBDNet	VB	237.96 (+20.65)	16.99 (+0.00)	-0.070 (-0.160)	0.650 (+0.001)	0.154 (-0.001)	16.7 (-4.5)
GFPGAN-SR	VB	249.09 (+31.78)	16.52 (-0.47)	-0.041 (-0.131)	0.639 (-0.010)	0.147 (-0.008)	21.8 (+0.6)
GAN+U-Net	Normal	277.13 (+59.82)	14.35 (-2.64)	0.010 (-0.080)	0.496 (-0.153)	0.105 (-0.050)	29.8 (+8.6)
Deblur-GANv2-+Mobile-Net	Normal	184.46 (-32.85)	17.55 (+0.56)	-0.024 (-0.114)	0.712 (+0.063)	0.180 (+0.025)	19.5 (-1.7)
Restormer	Normal	108.18 (-109.13)	18.52 (+1.53)	0.079 (-0.011)	0.799 (+0.150)	0.230 (+0.075)	24.2 (+3.0)
LPRGAN	**Normal**	155.57 (-61.74)	18.39 (+1.4)	**0.155 (+0.065)**	0.713 (+0.064)	0.210 (+0.055)	23.1 (+1.9)

Method	**Rating**	**FID (△FID)**	**PSNR (△PSNR)**	**SCC (△SCC)**	**SSIM (△SSIM)**	**VIF (△VIF)**	**File Size (△File Size)**
Actual Vertical Blur	VB	-	-	-	-	-	25.3
Conv. Autoencoder	VB	-	-	-	-	-	15.3 (-10.0)
CBDNet	VB	-	-	-	-	-	19.5 (-5.8)
GFPGAN-SR	VB	-	-	-	-	-	25.1 (-0.2)
GAN+U-Net	Normal	-	-	-	-	-	30.3 (+5.0)
Deblur-GANv2-+Mobile-Net	VB	-	-	-	-	-	20.9 (-4.4)
Restormer	VB	-	-	-	-	-	25.4 (+0.1)
LPRGAN	**Normal**	-	-	-	-	-	29.5 (+4.2)

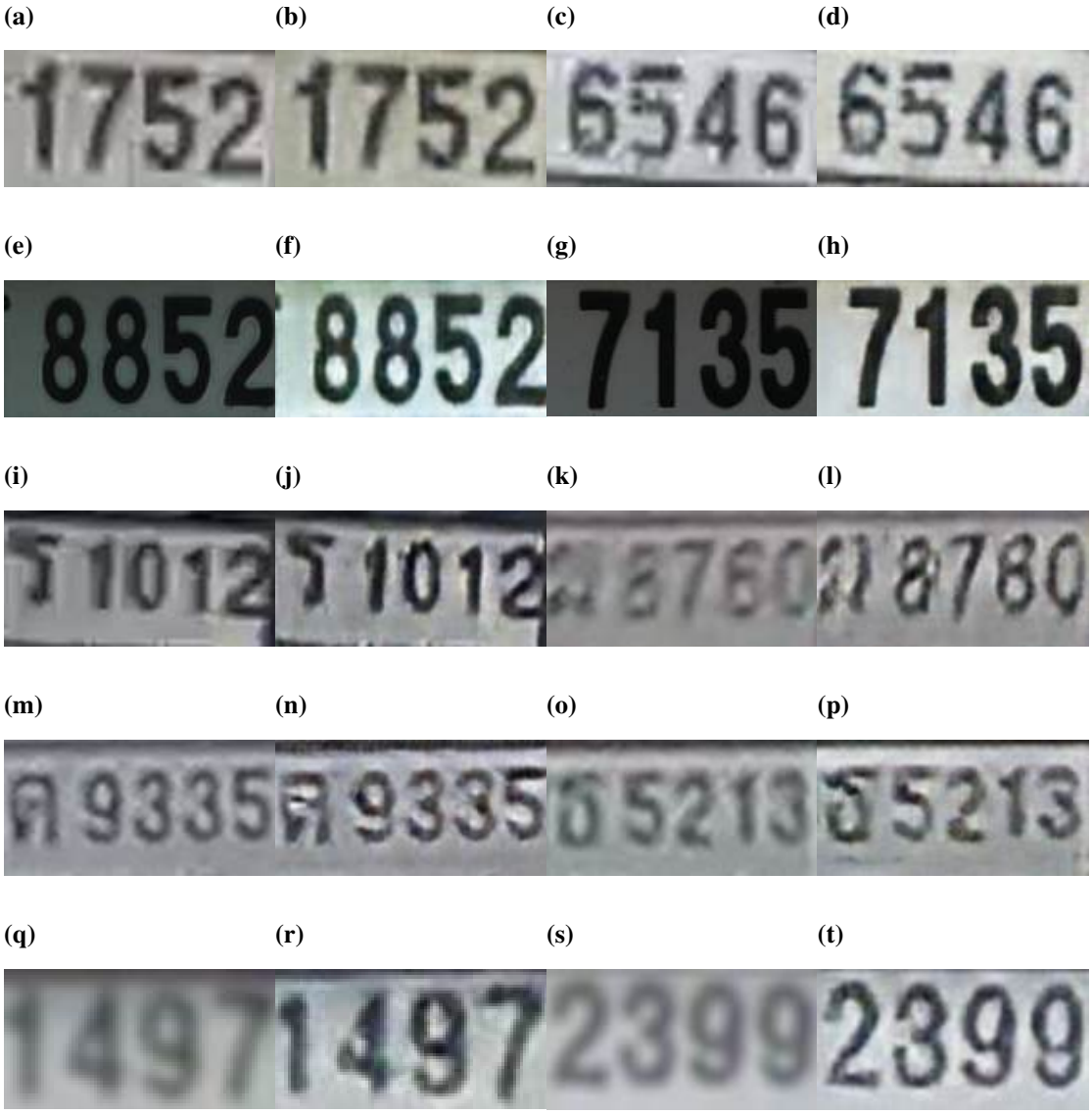
Table 4.6

A Metric Measurement Score Table on Out of Focus Problem

Method	**Rating**	**FID (△FID)**	**PSNR (△PSNR)**	**SCC (△SCC)**	**SSIM (△SSIM)**	**VIF (△VIF)**	**File Size (△File Size)**
Original/ Baseline	Normal	0.0	48.0	1.0	1.0	1.0	31.7
Simulated OOF	OOF	191.77	16.87	-0.044	0.614	0.172	15.7
Conv. Autoencoder	OOF	228.56 (+36.79)	15.27 (-1.6)	0.012 (+0.056)	0.472 (-0.142)	0.120 (-0.052)	13.1 (-2.6)
CBDNet	OOF	191.03 (-0.74)	16.78 (-0.09)	-0.052 (-0.008)	0.608 (-0.006)	0.170 (-0.002)	8.1 (-7.6)
GFPGAN-SR	Normal	221.20 (+29.43)	14.71 (-2.16)	0.102 (+0.146)	0.649 (+0.035)	0.232 (+0.060)	27.2 (+11.5)
GAN+U-Net	Normal	**132.36 (-59.41)**	13.96 (-2.91)	0.095 (+0.139)	0.642 (+0.028)	0.195 (+0.023)	**28.3 (+12.6)**
LPRGAN	**Normal**	169.63 (-22.14)	**17.82 (+0.95)**	**0.175 (+0.219)**	**0.702 (+0.088)**	0.210 (+0.038)	**28.3 (+12.6)**

Figure 4.41

Result on Additional Actual Thai License Plates from LPRGAN (a) Low Bitrate Input Image#1, (b) Low Bitrate Output Image#1, (c) Low Bitrate Input Image#2, (d) Low Bitrate Output Image#2, (e) Low Light Input Image#1, (f) Low Light Output Image#1, (g) Low Light Input Image#2, (h) Low Light Output Image#2, (i) Horizontal Blur Input Image#1, (j) Horizontal Blur Output Image#1, (k) Horizontal Blur Input Image#2, (l) Horizontal Blur Output Image#2, (m) Vertical Blur Input Image#1, (n) Vertical Blur Output Image#1, (o) Vertical Blur Input Image#2, (p) Vertical Blur Output Image#2, (q) Out of Focus Input Image#1, (r) Out of Focus Output Image#1, (s) Out of Focus Input Image#2 and (t) Out of Focus Output Image#2



Method	Rating	FID (ΔFID)	PSNR (ΔPSNR)	SCC (ΔSCC)	SSIM (ΔSSIM)	VIF (ΔVIF)	File Size (ΔFile Size)
Actual OOF	OOF	-	-	-	-	-	17.9
Conv. Autoencoder	OOF	-	-	-	-	-	12.3 (-5.6)
CBDNet	OOF	-	-	-	-	-	8.2 (-9.7)
GFPGAN-SR	OOF	-	-	-	-	-	18.5 (+0.6)
GAN+U-Net	Normal	-	-	-	-	-	24.0 (+6.1)
LPRGAN	**Normal**	-	-	-	-	-	**26.9 (+9.0)**

Figure 4.42

Average Real World Data Recovery Performance Result

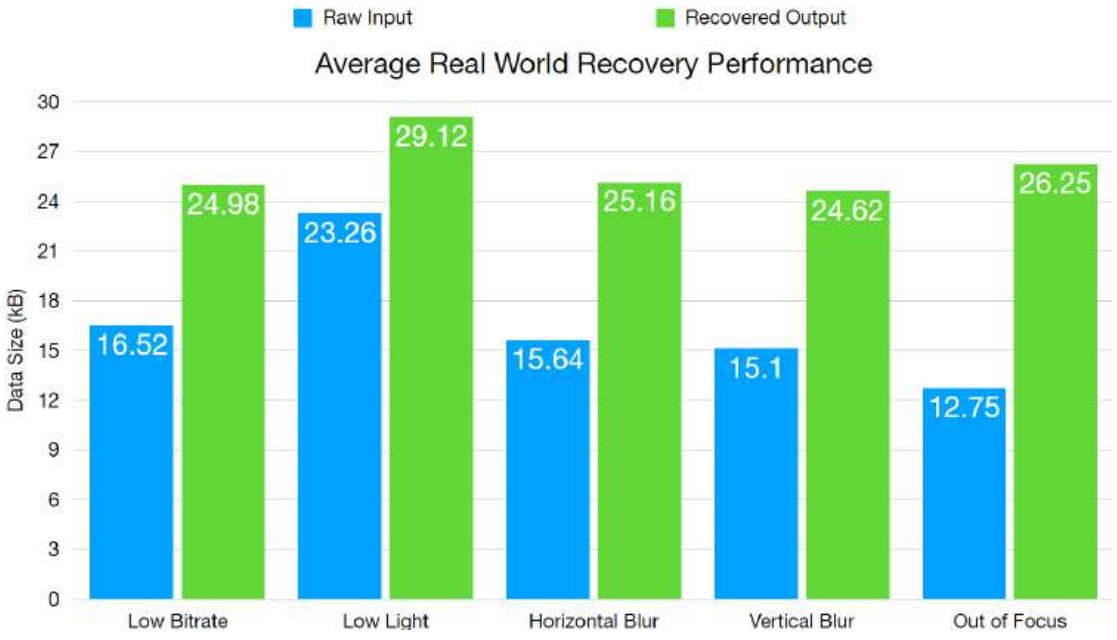


Table 4.7

A Metric Measurement Score Table on International Plate

Method	**Rating**	**FID (△FID)**	**PSNR (△PSNR)**	**SCC (△SCC)**	**SSIM (△SSIM)**	**VIF (△VIF)**	**File Size (△File Size)**
US Original/ Baseline	5-Star/ Normal	0.0	48.0	1.0	1.0	1.0	34.5
US Sim. Low Bitrate	1-Star	142.27	20.02	0.300	0.797	0.306	2.2
US LPRGAN	**5-Star**	157.80 (+15.53)	19.32 (-0.7)	**0.340 (+0.04)**	0.789 (-0.008)	0.289 (-0.017)	**30.4 (+28.2)**
US Sim. Low Light	Low Light	66.79	4.45	0.859	0.271	0.561	21.3
US LPRGAN	**Normal**	160.02 (+93.23)	**13.75 (+9.3)**	0.325 (-0.534)	**0.749 (+0.478)**	0.285 (-0.276)	**29.5 (+8.2)**
US Sim. HB	HB	164.79	15.21	0.270	0.519	0.114	21.5
US LPRGAN	**Normal**	230.80 (+66.01)	14.5 (-0.71)	0.091 (-0.179)	0.476 (-0.043)	0.104 (-0.010)	**29.0 (+7.5)**
US Sim. VB	VB	182.41	17.09	0.301	0.632	0.206	23.3
US LPRGAN	**Normal**	237.56 (+55.15)	16.91 (-0.18)	0.161 (-0.140)	**0.645 (+0.013)**	0.192 (-0.014)	**29.3 (+6.0)**

Method	**Rating**	**FID (△FID)**	**PSNR (△PSNR)**	**SCC (△SCC)**	**SSIM (△SSIM)**	**VIF (△VIF)**	**File Size (△File Size)**
UK Original/ Baseline	5-Star/ Normal	0.0	48.0	1.0	1.0	1.0	26.0
UK Sim. Low Bitrate	1-Star	346.09	18.55	0.294	0.580	0.214	1.9
UK LPRGAN	**5-Star**	369.93 (+23.84)	17.73 (-0.82)	**0.334 (+0.04)**	**0.639 (+0.59)**	**0.236 (+0.022)**	**33.7 (+31.8)**
UK Sim. Low Light	Low Light	64.30	7.91	0.813	0.306	0.521	17.2
UK LPRGAN	**Normal**	247.79 (+183.49)	**16.61 (+8.7)**	0.393 (-0.42)	**0.665 (+0.359)**	0.275 (-0.246)	**32.4 (+17.0)**
UK Sim. HB	HB	242.53	15.47	0.234	0.540	0.196	18.4
UK LPRGAN	**Normal**	378.32 (+135.79)	14.31 (-1.16)	0.116 (-0.118)	0.432 (-0.108)	0.114 (-0.082)	**32.4 (+14.0)**
UK Sim. VB	VB	246.28	16.76	0.346	0.690	0.297	20.1
UK LPRGAN	**Normal**	357.76 (+111.48)	15.32 (-1.44)	0.139 (-0.207)	0.567 (-0.123)	0.181 (-0.116)	**33.8 (+13.7)**

Table 4.8

Methods Speed Comparison Table

Model	Total Params	Training Time	Render Speed
Conv. Autoencoder @256x128 Video#1	0.26M	42.9	249
CBDNet @256x128 Video#1	-	-	24
GFPGAN-SR @256x128 Video#1	-	-	14
GAN+U-Net @256x128 Video#1	31.55M	369.2	73
DeblurGANv2+MobileNet @256x128 Video#1	3.01M	-	49
Restormer @256x128 Video#1	-	-	100
SwinIR @256x128 Video#1	-	-	19
LPRGAN @256x128 Video#1	0.96M	125.8	177

Figure 4.43

(a) Input Image, (b) DeblurGANv2+MobileNet Output Image, (c) Restormer Output Image and (d) LPRGAN Output Image

(a) **(b)** **(c)** **(d)**

Model	Total Params	Training Time	Render Speed
Conv. Autoencoder @600x400 Video#1	0.26M	314.8	68
CBDNet @600x400 Video#1	-	-	12
GFPGAN-SR @600x400 Video#1	-	-	4
GAN+U-Net @600x400 Video#1	32M	759.1	20
MIRNet @600x400 Video#1	-	-	1.9
DeblurGANv2+MobileNet @600x400 Video#1	3.01M	-	28
Restormer @600x400 Video#1	-	-	20
SwinIR @600x400 Video#1	-	-	3
LPRGAN @600x400 Video#1	1.37M	464.3	48

Model	Total Params	Training Time	Render Speed
Conv. Autoencoder @1280x720 Video#1	0.26M	906.2	24
CBDNet @1280x720 Video#1	-	-	6
GFPGAN-SR @1280x720 Video#1	-	-	1
GAN+U-Net @1280x720 Video#1	33M	1567.5	8
DeblurGANv2+MobileNet @1280x720 Video#1	3.01M	-	11
Restormer @1280x720 Video#1	-	-	5
SwinIR @1280x720 Video#1	-	-	0
LPRGAN @1280x720 Video#1	2.73M	1294.1	15

Table 4.9

LPRGAN Speed Test Table

Mode	Total Params	Training Time	Render Speed
Ultra Low Power CPU			
LPRGAN @256x128 Video#1	0.96M	-	7
Adaptive LPRGAN @256x128 Video#1	0.96M	-	7 (+0)
LPRGAN @256x128 Video#2	0.96M	-	7
Adaptive LPRGAN @256x128 Video#2	0.96M	-	12 (+5)
LPRGAN @256x128 Video#3	0.96M	-	7
Adaptive LPRGAN @256x128 Video#3	0.96M	-	23 (+11)
Workstation CPU			
LPRGAN @256x128 Video#1	0.96M	-	17
Adaptive LPRGAN @256x128 Video#1	0.96M	-	17 (+0)
LPRGAN @256x128 Video#2	0.96M	-	17
Adaptive LPRGAN @256x128 Video#2	0.96M	-	30 (+13)
LPRGAN @256x128 Video#3	0.96M	-	17
Adaptive LPRGAN @256x128 Video#3	0.96M	-	58 (+41)

Mode	Total Params	Training Time	Render Speed
GPU			
LPRGAN @256x128 Video#1	0.96M	-	176
Adaptive LPRGAN @256x128 Video#1	0.96M	-	176 (+0)
LPRGAN @256x128 Video#2	0.96M	-	176
Adaptive LPRGAN @256x128 Video#2	0.96M	-	298 (+122)
LPRGAN @256x128 Video#3	0.96M	-	176
Adaptive LPRGAN @256x128 Video#3	0.96M	-	512 (+336)

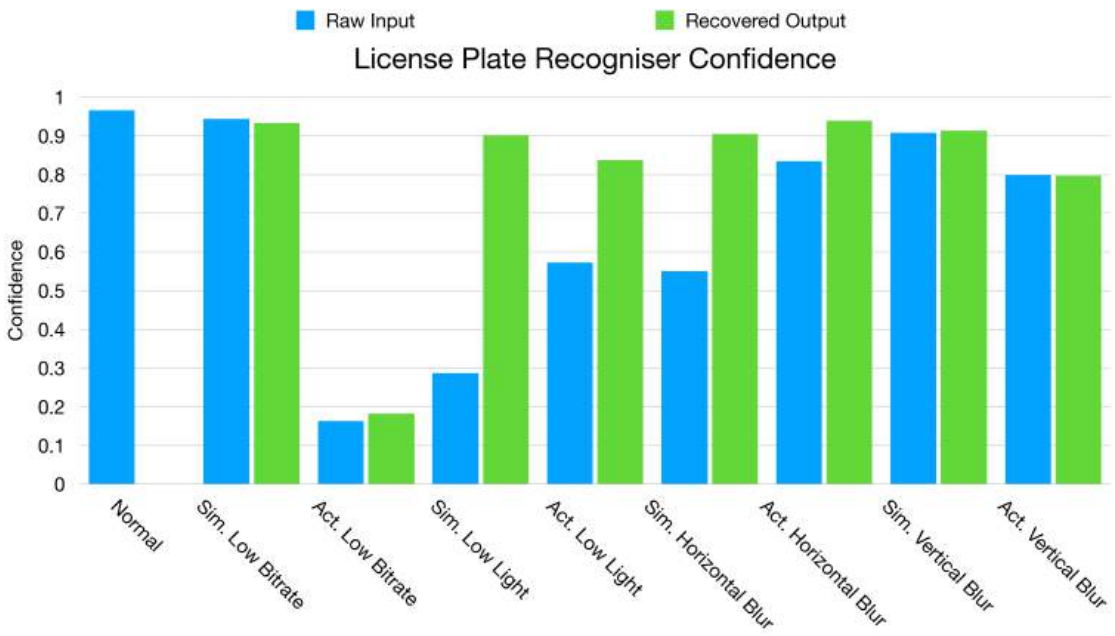
Table 4.10

Memory Usage Comparison Table

Model	RAM Usage
Non GAN Based	
Conv. Autoencoder	109.5
CBDNet	182.3
MIRNet	580.6
Transfomer Based	
Restormer	3136.0
SwinIR	3208.8
GAN Based	
GFPGAN-SR	767.4
GAN+U-Net	576.7
DeblurGANv2+MobileNet	633.7
LPRGAN	306.7

Figure 4.44

Recognizer Prediction Confidence Result



in Fig.3.2 which has three classes, EU, TH (*Offence Relating to Documents. Chapter 3, Section 264. Thailand Penal Code Thai Criminal law.*, 2022) and US plates (EU-Belgian and US datasets were from (*Belgian License Plates. Kaggle.*, 2022) and (*US License Plates. Kaggle.*, 2022)), in total 1,237 training images. These training images differ from the LPRGAN dataset, making predictions the most neutral. However, we only focus on the TH license plate class to prove that with the help of LPRGAN can make a recognizer has more confidence in recognizing a license plate. In Fig.4.44 is a result from Fig.4.45 images set, there is not much different result in low bitrate and vertical blur problems but in low light and horizontal blur, cases result in a great benefit from using the LPRGAN system, whereas normal is a good quality image so its confidence is the highest. The next test is an average confidence value result by sampling a set of each degraded type from real-world images (some of them were shown in Fig.4.41a-Fig.4.41t). The result in Fig.4.46 proves that recovery images help to increase an average prediction performance in low bitrate by 1.1X, low light by 1.41X, horizontal blur by 1.52X, horizontal blur by 1.16X, and out of focus by 1.29X, respectively.

Figure 4.45

Images Set Used in Recognition Test

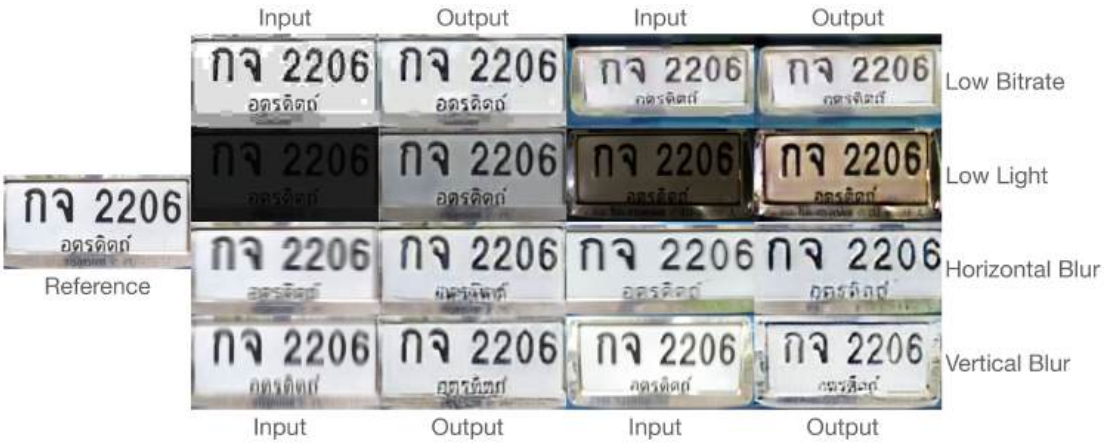
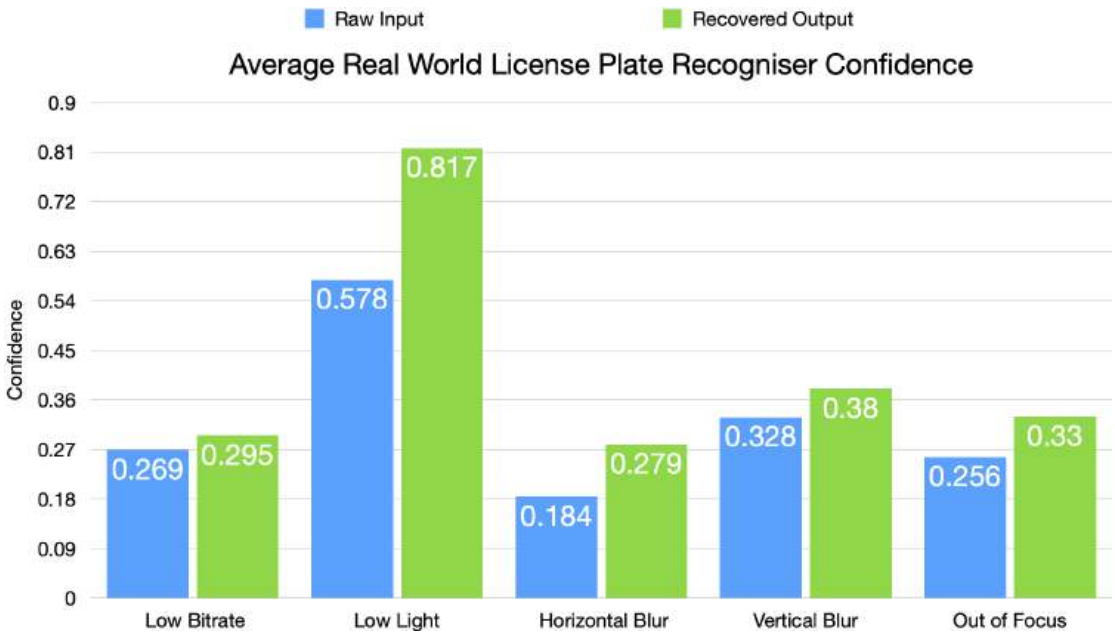


Figure 4.46

Average Real World Prediction Confidence Result



CHAPTER 5
CONCLUSION

The research presented in this article studied a way to implement a fast license plate image quality recovery for traffic monitoring in various poor situations. The proposed framework uses the optimized lightweight encoder-decoder style CNN architecture built inside a GAN model to do a recovery job alongside image classifications that detect inputs and verify outputs, helping the LPRGAN in a much more efficient and effective way. This study proved that it could improve low bitrate, low light, and motion blur problems from a single design network in many test cases. Not only that, this system is able to outpace or be at the same quality level as other complex networks while performing the task quickest. As a result, the proposed system can run on less computational power machines like most typical workstation PCs without a discreet graphic card at a reasonable pace and is possible to deploy in embedded systems such as edge computing devices. This study opens a new door for many power-constrained image recovery applications. Such benefits make this framework easy to be deployed on traffic officer computers or even embedded within camera recording boxes, aiding them in identifying vehicle licenses in inadequate conditions. Thus removing the need for a high-performance server machine and greatly reducing network bandwidth usage between devices.

At this stage, the LPRGAN can render a real-time frame recovery up to 1280x720@15fps, which is sufficient for most typical CCTV/IP cameras, for example, Merit Lilin ZG1232EX3 (3MP, 15FPS) or Merit Lilin LR832 (2MP, 15FPS). As time passes, license plates can be collected more, giving a model retraining even more performance gain. However, this study demonstrated a few applications that this system could handle. It depends on how the user provides a dataset for training because the system uses a good dataset as a template and learns how a distorted dataset differs, so it would theoretically work in other situations too.

REFERENCES

Adam optimizer in tensorflow. (2023, Jan). Retrieved from https://www.geeksforgeeks.org/adam-optimizer-in-tensorflow

Belgian license plates. kaggle. (2022, Nov). Retrieved from https://www.kaggle.com/datasets/aladdinss/license-plate-annotated-image-dataset

Cgan. conditional gan. (2022, Jul). Retrieved from https://keras.io/examples/generative/conditional_gan

Chollet, F. (2016, May). *Autoencoder. building autoencoders in keras.* Retrieved from https://blog.keras.io/building-autoencoders-in-keras.html

Dar, Y., & Bruckstein, A. M. (2015, Apr). Improving low bit-rate video coding using spatio-temporal down-scaling. *Multimedia cs.MM*. Retrieved from https://arxiv.org/abs/1404.4026v2

Dcgan. deep convolutional generative adversarial network. (2023, Nov). Retrieved from https://www.tensorflow.org/tutorials/generative/dcgan

Develop, optimize and deploy gpu-accelerated apps. (2023, Mar). Retrieved from https://developer.nvidia.com/cuda-toolkit

Fid score for pytorch. (2022, Jul). Retrieved from https://pypi.org/project/pytorch-fid

Guo, S., Yan, Z., & Zhang, K. (2019, Jun). Toward convolutional blind denoising of real photographs. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Retrieved from https://ieeexplore.ieee.org/document/8954448

How to add motion blur to numpy array. (2016, Oct). Retrieved from https://stackoverflow.com/questions/40305933/how-to-add-motion-blur-to-numpy-array

How to implement the frechet inception distance (fid) for evaluating gans. (2022, Jul). Retrieved from https://machinelearningmastery.com/how-to-implement-the-frechet-inception-distance-fid-from-scratch

Image compression. (2022, Jul). Retrieved from https://en.wikipedia.org/wiki/Image_compression

Image-to-image translation using pix2pix. (2022, Jun). Retrieved from https://www.geeksforgeeks.org/image-to-image-translation-using-pix2pix

Jeff's license plates. jeffsplates. (2022, Aug). Retrieved from http://

www.jeffsplates.ca/wp-content/uploads/2018/07/E5E421DF-5108-456C-AE5B-A479BA65A1B2.jpeg

Jpeg. (2022, Jul). Retrieved from https://en.wikipedia.org/wiki/JPEG

Keras. (2023, Mar). Retrieved from https://keras.io

Kupyn, O., Martyniuk, T., & Wu, J. (2019, Aug). Deblurgan-v2: Deblurring (orders-of-magnitude) faster and better. *IEEE/CVF International Conference on Computer Vision (ICCV)*. Retrieved from https://ieeexplore.ieee.org/document/9008540

Langr, J., & Bok, V. (2019). *Gans in action - deep learning with generative adversarial networks*. Manning.

Learning rate decay and methods in deep learning. (2022, Jul). Retrieved from https://medium.com/analytics-vidhya/learning-rate-decay-and-methods-in-deep-learning-2cee564f910b

Li, Y., Liu, D., Li, H., Li, L., Wu, F., Zhang, H., & Yang, H. (2017, Jul). Convolutional neural network-based block up-sampling for intra frame coding. *IEEE Transactions on Circuits and Systems for Video Technology*. Retrieved from https://arxiv.org/abs/1702.06728v3

Liang, J., Cao, J., Sun, G., Zhang, K., Gool, L. V., & Timofte, R. (2021, Aug). Swinir: Image restoration using swin transformer. *Image and Video Processing (eess.IV), Computer Vision and Pattern Recognition (cs.CV)*. Retrieved from https://ieeexplore.ieee.org/document/9878962

Lin, H., He, X., & Qing, L. (2019, May). Improved low bitrate hevc video coding using deep learning based super-resolution and adaptive block patching. *IEEE Transactions on Multimedia*. Retrieved from https://ieeexplore.ieee.org/document/8723517

Offence relating to documents. chapter 3, section 264. thailand penal code thai criminal law. (2022, Nov). Retrieved from https://www.samuiforsale.com/law-texts/thailand-penal-code.html#3

Peak to signal noise ratio. (2022, Jul). Retrieved from https://sonalsart.com/what-is-psnr

Petrov, A., Kartalov, T., & Ivanovski, Z. (2009, Nov). Blocking effect reduction in low bitrate video on a mobile platform. presented at ieee international conference on image processing. *IEEE International Conference on Image Processing*. Retrieved

from https://ieeexplore.ieee.org/abstract/document/5414031

Ronneberger, O., Fischer, P., & Brox, T. (2015, May). U-net: Convolutional networks for biomedical image segmentation. *MICCAI 2015*. Retrieved from https://arxiv.org/abs/1505.04597v1

Sewar python package. (2022, Jul). Retrieved from https://pypi.org/project/sewar

Sharma, R. (2022, Aug). *Clustering vs classification: Difference between clustering & classification.* Retrieved from https://www.upgrad.com/blog/clustering-vs-classification

Springenberg, J. T., Dosovitskiy, A., Brox, T., & Riedmiller, M. (2015, Apr). Striving for simplicity : The all convolutional net. *ICLR 2015*. Retrieved from https://arxiv.org/abs/1412.6806v3

Structural similarity index. (2022, Jul). Retrieved from https://medium.com/srm-mic/all-about-structural-similarity-index-ssim-theory-code-in-pytorch-6551b455541e

Uk european license plate. european license plates. (2022, Aug). Retrieved from https://www.customeuropeanplates.com/images/uk-license-plate.jpg

Use of disclosure of personal data, section 24 and 27. thai government gazette. (2022, Aug). Retrieved from https://data.opendevelopmentmekong.net/dataset/78c90118-6671-4c19-afe1-7bfbace4d46a/resource/ec616be5-9fbf-4071-b4b5-cb1f3e46e826/download/entranslation_of_the_personal_data_protection_act_0.pdf

Us license plates. kaggle. (2022, Nov). Retrieved from https://www.kaggle.com/datasets/tolgadincer/us-license-plates

Vallejos, R., Perez, J., Ellison, A. M., & Richardson, A. D. (2019, May). A spatial concordance correlation coefficient with an application to image analysis. *Methodology (stat.ME)*. Retrieved from https://arxiv.org/abs/1905.05016

Vgg-16 cnn model. (2023, Jan). Retrieved from https://www.geeksforgeeks.org/vgg-16-cnn-model

Visual information fidelity. (2022, Jul). Retrieved from https://www.sciencedirect.com/topics/computer-science/visual-information-fidelity

Wang, X., Li, Y., & Zhang, H. (2021, Jun). Towards real-world blind face restoration

with generative facial prior. *CVPR 2021*. Retrieved from https://arxiv.org/abs/2101.04061v2

Wu, W., Guo, X., Chen, Y., Wang, S., & Chen, J. (2022, Nov). Deep embedding-attention-refinement for sparse-view ct reconstruction. *IEEE Transactions on Instrumentation and Measurement*. Retrieved from https://ieeexplore.ieee.org/document/9944644

Wu, W., Hu, D., Niu, C., Yu, H., Vardhanabhuti, V., & Wang, G. (2021, May). Drone: Dual-domain residual-based optimization network for sparse-view ct reconstruction. *IEEE Transactions on Medical Imaging*. Retrieved from https://ieeexplore.ieee.org/document/9424618

Yang, R., Xu, M., Liu, T., Wang, Z., & Guan, Z. (2018, Jul). Enhancing quality for hevc compressed videos. *IEEE Transactions on Circuits and Systems for Video Technology (2018)*. Retrieved from https://arxiv.org/abs/1709.06734

Zamir, S. W., Arora, A., & Khan, S. (2022, Sep). Restormer: Efficient transformer for high-resolution image restoration. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Retrieved from https://ieeexplore.ieee.org/document/9878962

Zamir, S. W., Arora, A., Khan, S., Hayat, M., Khan, F. S., Yang, M.-H., & Shao, L. (2020, Jul). Learning enriched features for real image restoration and enhancement. *European Conference on Computer Vision (ECCV) 2020*. Retrieved from https://arxiv.org/abs/2003.06792

Zhang, W., Zhou, Z., Gao, Z., Yang, G., Xu, L., Wu, W., & Zhang, H. (2022, Oct). Multiple adversarial learning based angiography reconstruction for ultra-low-dose contrast medium ct. *IEEE Journal of Biomedical and Health Informatics*. Retrieved from https://ieeexplore.ieee.org/document/9916111

Zhu, J.-Y., Park, T., Isola, P., & Efros, A. A. (2017, Mar). Unpaired image-to-image translation using cycle-consistent adversarial network. *ICCV*. Retrieved from https://arxiv.org/abs/1703.10593v7

Zhu, J.-Y., Park, T., Isola, P., & Efros, A. A. (2020, Aug). Unpaired image-to-image translation using cycle-consistent adversarial networks. *ICCV*. Retrieved from https://arxiv.org/abs/1703.10593v7

APPENDIX
LICENSE PLATE DATASET

Thai license plates dataset used in this work contains both actual plates and dummy plates but only the latter is available upon request due to privacy infringement on personal information disclosure.