6-2020

Deep Learning Methods for Efficient Image Coding

Zachary Daniel Bellay

Deep Learning Methods for Efficient Image Coding

by

Zachary Bellay

THESIS

Submitted to the School of Engineering,

of Santa Clara University,

Santa Clara, California

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

2020

MAJOR: Computer Science and Engineering

Approved By:

Ying Liu

Thesis Advisor Dr. Ying Liu

Thesis Reader Dr. Yi Fang

N. Ling

Department Chair Dr. Nam Ling

DEDICATION

To John, Kumi, and Abby.

ACKNOWLEDGEMENTS

I would like to thank Dr. Ying Liu for her patience and guidance throughout the two years I have pursued research with her. I have learned a lot under her guidance, and I am grateful that she has been kind to me through all stages of learning.

ABSTRACT
Deep Learning Methods for Efficient Image Coding

by

Zachary Daniel Bellay

June 2020

Advisor: Dr. Ying Liu
Major: Computer Science and Engineering
Degree: Master of Science

Video data makes up 58% of all internet traffic and is growing as self-driving car cameras, 4K televisions, and video surveillance systems continue to come online. Traditional heuristics based image and video codecs such as JPEG and HEVC have been successful thus far, however, these approaches lack the ability to leverage big data to gain massive insights. Six deep learning based approaches are proposed to tackle efficient image/video compression and image compression for machine classification.

Keywords: Image Compression, Video Compression, Image Classification, GAN, CNN, Deep Learning, Coding for Machines

TABLE OF CONTENTS

Dedication ii

Acknowledgements iii

Abstract iv

List of Tables vii

List of Figures viii

Chapter 1 Introduction 1

1.1 Motivation 1

1.2 Background 1

1.2.1 Traditional Methods: JPEG and HEVC 1

1.2.2 Deep Learning Methods: CNNs and GANs 4

Chapter 2 Literature Survey 6

2.1 GANs for Extreme Learned Image Compression 6

2.1.1 Architecture 6

2.1.2 Loss 6

2.2 Adversarial Video Compression Guided by Soft Edge Detection 7

2.3 An End-to-End Compression Framework Based on Convolutional Neural Networks 9

2.4 Generative Compression 12

Chapter 3 GAN-Based Image Compression 13

3.1 Objective 13

3.2 Methodology 13

3.2.1 Full Image Based Compression 13

3.3 Image Block Based Compression 16
3.4 Experiments 20
3.4.1 Full Image Based Compression 20
3.4.2 Image Block Based Compression 21
Chapter 4 Image Coding for Machine Classification 26
4.1 Objective 26
4.2 Methodology 26
4.2.1 Concatenated Autoencoder and Classifier 26
4.2.2 Separated Autoencoder and Classifier 28
4.2.3 Hydra-net Based Autoencoder and Classifier 29
4.2.4 VGG16 with Batch Normalization Baseline Classifier 31
4.3 Experiments 32
4.3.1 Dataset 32
4.4 Training Details 33
4.4.1 Results 33
Chapter 5 Conclusion 37
5.1 Suggestions for Improvement 37
References 39

LIST OF TABLES

Table 1 Generator architecture by Agustsson *et al.* The residual blocks are the same as those specified by He *et al* with specific parameters specified [10]. 8

Table 2 Generator architecture inspired by Agustsson *et al.* The residual blocks are the same as those specified by He *et al* with specific parameters specified [10]. 15

Table 3 Discriminator architecture inspired by Aguston *et al* with Spectral Normalization improvement [14]. 15

Table 4 The WGAN + Spectral Normalization method has higher perceptual quality metrics than solely training by using an autoencoder. 21

Table 5 Performance of encoding/decoding an image. 21

Table 6 The 20 most populous classes selected from the Caltech 256 dataset used to construct an evenly distributed dataset. 32

LIST OF FIGURES

Figure 1 Top network traffic sources in various regions of the world. [1] . . . 2

Figure 2 GAN architecture proposed by Agustsson *et al.* 6

Figure 3 High level architecture proposed by Kim *et al.* 9

Figure 4 Compressed edge map pipeline proposed by Kim *et al.* 9

Figure 5 Down sampled edge map input to cGAN outputs recovered $\hat{X}$. 10

Figure 6 Down sampled edge map input to cGAN outputs recovered $\hat{X}$. 11

Figure 7 Down sampled edge map input to cGAN outputs recovered $\hat{X}$. 11

Figure 8 Architecture of image based image compression experiment. C represents the variable number of feature maps at the bottleneck layer. . 14

Figure 9 Architecture of block based image compression experiment. C represents the variable number of feature maps at the bottleneck layer. The recovered image does not resemble the input image, and as a result is a major shortcoming of this GAN based approach. 18

Figure 10 The GAN-based approach has generally sharper results, however, the GAN images are also more subject to various forms of warping. For example, the rightmost photo of an emu is certainly sharper in the GAN image, however, it is also more distorted. 21

Figure 11 The GAN based approach is only able to learn very simple representations, such as the all white or all black video frame blocks. Otherwise, the GAN based approach exhibits symptoms of mode collapse and training collapse, whereas the autoencoder based approach recovers a blurry version of the image. This model's PSNR has a value of 8.144 dB, which is among the highest in the GAN based approach.The hyperparameters are $\lambda_{MSE} = 10$, $\lambda_{GAN} = 1$, autoencoder head start is used, and $C = 16$, so the compression ratio is 12. 23

Figure 12 Almost immediately after the discriminator is introduced, the visual quality plummets dramatically. 24

Figure 13 Almost immediately after the discriminator is introduced, the visual quality plummets dramatically. 24

Figure 14 Almost immediately after the discriminator is introduced, the discriminator is able to distinguish between real and fake images at rates near 100% accuracy, thus providing no useful feedback to the generator. . . 25

Figure 15 The architecture of the proposed architecture for both the concatenated and separated models. 27

Figure 16 The architecture of the proposed hydra-net model. 30

Figure 17 The architecture of VGG 16 with Batch Normalization. 31

Figure 18 A sample of images from the training set. 32

Figure 19 Bits per pixel versus the test classification accuracy for each model proposed for image coding for machine classification. 34

Figure 20 A bitrate of 0.00781 represents a compression ratio of $\tilde{1}$024x. Despite this massive compression, VGG net is able to achieve 89.25% test accuracy. This demonstrates the impressive ability of the VGG net feature extractor. 36

CHAPTER 1 INTRODUCTION

1.1 Motivation

As more people come online, so too do IoT devices for video surveillance, self-driving car cameras, 4K televisions, and other high bandwidth video and image related services. This big data must be efficiently transmitted, stored, and computed upon to ensure that the internet remains accessible, affordable, and available to all users. Although various advancements in increasing broadband and cell network bandwidth are being pursued, such as gigabit ethernet and 5G cell, reducing the footprint of video and image data stands to dramatically impact internet traffic. Figure 1 demonstrates how in 2018, 58% of all internet traffic globally came from video streaming [1]. The implications of increased video coding efficiency means reduction in network traffic, reduction in energy consumption per video, cheaper storage per video, and more bandwidth available to users. In order to achieve this goal, new video and image compression and computation techniques must be adopted over today's standards such as HEVC and JPEG. While these standards remain widely utilized, the dramatic growth of big data will require that new techniques, such as those in machine learning and deep learning, be adopted to tackle the difficult problem of video and image compression. This paper will explore various methods in deep learning applied toward video and image compression, as well as image analysis.

1.2 Background

1.2.1 Traditional Methods: JPEG and HEVC

JPEG and HEVC are among the most commonly used image and video compression formats, respectively. They are both deterministic algorithms primarily based around multiple layers of transformations to reduce images into as sparse a representation as possible. JPEG exploits spatial redundancy in images, while HEVC is able to take advantage of spatial-temporal redundancies in videos.

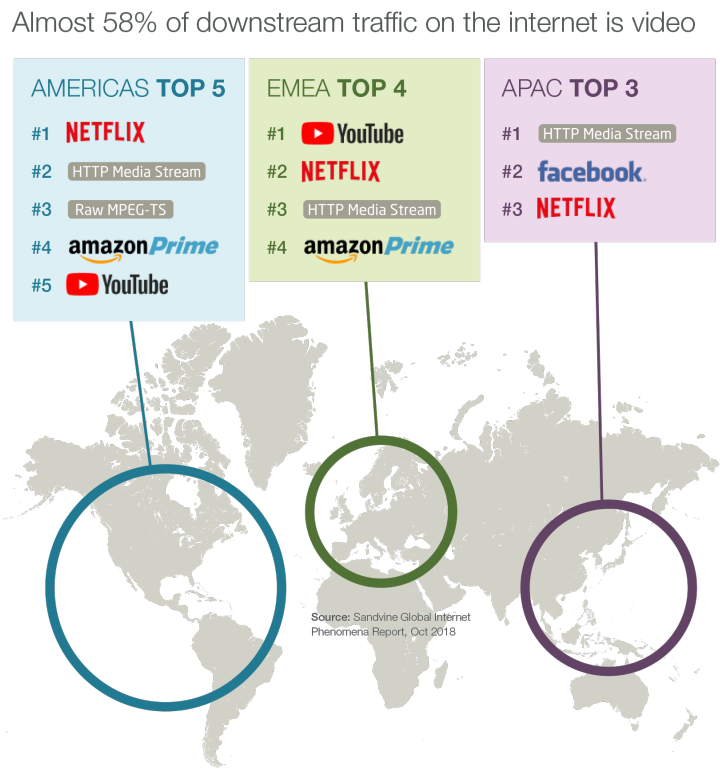

Figure 1: Top network traffic sources in various regions of the world. [1]

JPEG As mentioned, JPEG is able to compress images by taking advantage of the fact that most images are locally redundant. For example, if you were to take a picture of a landscape, the sky does not contain a great amount of detail, and as a result can be dramatically reduced to save space. JPEG compresses images by first converting an image from the RGB colorspace to the YUV colorspace. Then the two chroma components (U and V) are then downsampled because the human eye is not particularly sensitive to these two channels. Next, each channel of the image is then subdivided into 8x8 pixel blocks, also known as macroblocks. This macroblock then undergoes the Discrete Cosine Transform, which decomposes this 8x8 block from a set of pixels into a sum of a set of cosines with different frequencies and amplitudes. A macroblock that is generated from the sky in our landscape photo will only contain low frequency information, so the DCT will be able to represent this macroblock with only a few coefficients, as opposed to the entire 8x8 grid of pixels. Finally, this set of coefficients is quantized by being rounded to the nearest integer, and then fed through a custom lossless entropy coding technique to minimize the representation. To recover the image, this process is then conducted in reverse, with the exception of the quantization as this information has been destroyed through this process. [3]

HEVC HEVC is similar to JPEG in that it is able to take advantage of spatial redundancy, however, because it is a video format it is also able to take advantage of temporal redundancies as well. The process for video compression with HEVC involves intraprediction, in which macroblocks are estimated based on surrounding macroblocks. It also involves interprediction in which macoblocks are estimated based on previously recovered macroblocks from previous frames. The approach HEVC uses for intraprediction is similar to that of JPEG in that the macroblock is decomposed into its DCT and quantized. However, the main difference is that on the encoder side, HEVC calculates a prediction and compares its prediction to the ground truth it has available. This residual between the prediction and the ground truth is then quantized, compressed, and sent to the decoder side. For temporally-based intraprediction, HEVC decomposes a set of video frames into a background and the motion on top of that background. This approach allows HEVC to not

have to send each frame over and over, and instead, HEVC can send only the parts that are changing. [7]

1.2.2 Deep Learning Methods: CNNs and GANs

While JPEG and HEVC have and continue to perform well, their main limitation is that they are essentially highly tuned heuristic algorithms. Although such algorithms perform well, in the age of big data, we are capable of training machines to learn efficient representations of images. This approach stands to be much more performant than a heuristic based approach because machine learning models are capable of making generalizations in high dimensional spaces which is not possible with a heuristic based approach.

CNNs The breakthrough moment for convolutional neural networks (CNN) came from the 2012 ImageNet Large Scale Visual Recognition Contest (ILSVRC). When AlexNet, a CNN, outperformed all other entries by 10% in top-1 accuracy, many researchers took note and began developing their own CNNs. The key contribution of AlexNet was the insight to train deep networks by utilizing GPUs for accelerating training. Years later, many higher performing CNN models have been introduced to ILSVRC. These now famous architectures include GoogLeNet, VGG Net, ResNet and Inception Net. This dramatic increase in top-1 accuracy has sparked a revolution in computer vision related artificial intelligence tasks. While ILSVRC was focused on image classification, many researchers began developing approaches for other tasks such as image compression. Many researchers have already demonstrated that CNNs stand to benefit the field of image and video compression. [2]

GANs As the dramatic improvement of CNNs began to slow down, Ian Goodfellow *et al* introduced a new approach to training a CNN called generative adversarial networks (GANs). This idea was to use one neural network to train another. In theory, this would allow neural networks to achieve even greater performance. In practice, GANs have proven to be difficult to train. However, if trained properly, GANs are capable of achieving impressive results. GANs are particularly interesting for the task of image and video compression because GANs are designed to 'hallucinate' images into its output. This means that GANs

are able to learn what a realistic image should look like a is then capable of producing something that one might actually observe in an image. In image and video compression, because data is thrown away during compression, GANs offer the ability to reintroduce the data that was thrown away by sampling a prior knowledge from images and videos it has trained on.

CHAPTER 2 LITERATURE SURVEY

2.1 GANs for Extreme Learned Image Compression

In this paper, the authors propose multiple conditional and a non-conditional GANs for image compression. The first conditional approach is one in which a binary map of which areas should be preserved and generated is fed only to the discriminator. The other is one in which the binary map is fed both to the generator and discriminator. Both of these conditional methods show promise but are left to be fully developed in future works. The main focus of the paper is a non-conditional GAN. This GAN uses an encoder-decoder as the generator, a quantizer to reduce the size of the latent compressed representation, a custom entropy coding technique to compress the quantized representation, and a discriminator used during training to guide the generator to produce higher quality images. The high level overview of this architecture is shown in Figure 2. [4]

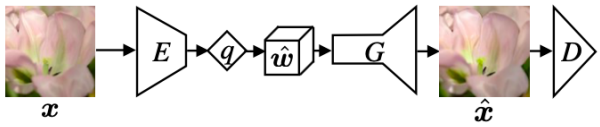

Figure 2: GAN architecture proposed by Agustsson *et al.*

Some key contributions by this paper include the architecture and the loss function.

2.1.1 Architecture

The overview of the generator architecture is provided in Table 1. While this architecture is a fairly standard deep encoder-decoder, this is one of first applications of this architecture to generative compression with great success. The bottleneck at C is varied to be $C \in \{2, 4, 8\}$ and thus supports variable compression at the architectural level.

2.1.2 Loss

The GAN loss function used as seen in Equation 2.1 by Agustsson *et al.* is unique in two approaches. First, they add a distortion loss function, which is simply the Mean Squared

Error (MSE) as seen in Equation 2.2 as the term d. This term is added to help preserve similar perceptual qualities from the input image and the generated output image.

$$\begin{aligned} \min_{E,G} \max_{D} \quad & \mathbb{E}[f(D(\boldsymbol{x}))] + \mathbb{E}[g(D(G(\boldsymbol{z})))] \\ & + \lambda \mathbb{E}[d(\boldsymbol{x}, G(\boldsymbol{z}))] + \beta H(\hat{\boldsymbol{w}}) \end{aligned} \tag{2.1}$$

$$\lambda \mathbb{E}[d(\boldsymbol{x}, G(\boldsymbol{z}))] \tag{2.2}$$

The authors also add an entropy loss term as seen in Equation 2.3. The definition of entropy can be seen in Equation 2.4. The purpose of this loss function is to penalize the encoder if it does not efficiently store information in the latent vector. This way by penalizing lack of efficiency, fewer bits can be used to represent the same data resulting in a higher compression ratio.

$$\beta H(\hat{\boldsymbol{w}}) \tag{2.3}$$

$$H(X) = -\sum_{i=1}^{n} P\left(x_i\right) \log_b P\left(x_i\right) \tag{2.4}$$

2.2 Adversarial Video Compression Guided by Soft Edge Detection

In this paper, Kim *et al.* propose a novel video compression scheme. Their main contribution is in essence, a more efficient inter-key frame interpolator for traditional video coding schemes such as H.264. The high level architecture as seen in Figure 3 demonstrates how E_1 will encode the video input X_1 using the H.264 encoder. The key frames of the H.264 encoder are represented as x_1. Traditionally, the H.264 decoder would then use x_1 and motion information to interpolate between frames thereby compressing the video sequence. However, in the approach proposed by Kim *et al.*, the non-key frame images are decoded by H.264 and fed through the pipeline in Figure 4. This pipeline features a down sampler which spatially reduces the dimensions of the image, a soft edge detector

Generator Architecture		
Layer Type (K= Kernel Size, S=Stride, P=Padding)	Input Shape (channel, height , width)	Output Shape (channel, height , width)
Encoder		
Reflection Padding (3, 3)	3, 512, 1024	3, 518, 1030
Conv 2D, Instance Norm, ReLU (K=7x7 S=1, P=0)	3, 518, 1030	60, 512, 1024
Conv 2D, Instance Norm, ReLU (K=3x3 S=2, P=1)	60, 512, 1024	120, 256, 512
Conv 2D, Instance Norm, ReLU (K=3x3 S=2, P=1)	120, 256, 512	240, 128, 256
Conv 2D, Instance Norm, ReLU (K=3x3 S=2, P=1)	240, 128, 256	480, 64, 128
Conv 2D, Instance Norm, ReLU (K=3x3 S=2, P=1)	480, 64, 128	960, 32, 64
Reflection Padding (1, 1)	960, 32, 64	960, 34, 66
Conv 2D, Instance Norm, ReLU (K=3x3 S=1, P=0)	960, 34, 66	C, 32, 64 (bottleneck)
Quantizer	C, 32, 64	C, 32, 64
Decoder		
Conv 2D, Instance Norm, ReLU (K=3x3 S=2, P=0)	C, 32, 64	960, 32, 64
Residual Block (K=3x3, S=1, P=1) x 9	960, 32, 64	960, 32, 64
Conv Transpose 2D, Instance Norm, ReLU (K=3x3, S=2, P=1)	960, 32, 64	480, 64, 128
Conv Transpose 2D, Instance Norm, ReLU (K=3x3, S=2, P=1)	480, 64, 128	240, 128, 256
Conv Transpose 2D, Instance Norm, ReLU (K=3x3, S=2, P=1)	240, 128, 256	120, 256, 512
Conv Transpose 2D, Instance Norm, ReLU (K=3x3, S=2, P=1)	120, 256, 512	60, 512, 1024
Reflection Padding (3, 3)	60, 512, 1024	60, 518, 1030
Conv 2D, Instance Norm, ReLU (K=3x3 S=1, P=0)	60, 518, 1030	3, 512, 1024

Table 1: Generator architecture by Agustsson *et al.* The residual blocks are the same as those specified by He *et al* with specific parameters specified [10].

which is the Canny Edge detector, and a novel lossless compression technique for this data. In effect, the image is spatially reduced, converted into a sparse edge map, and losslessly compressed. [13]

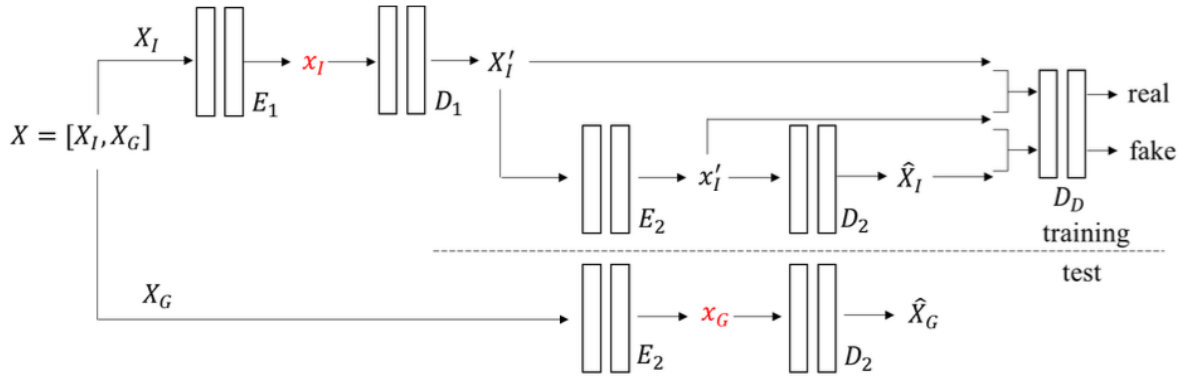

Figure 3: High level architecture proposed by Kim *et al.*

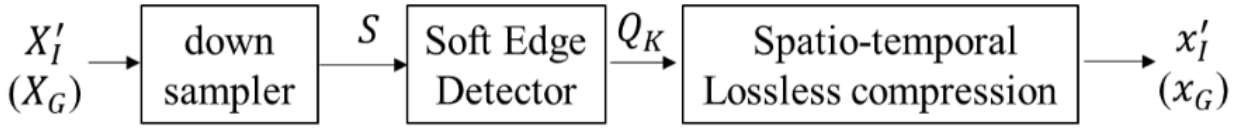

Figure 4: Compressed edge map pipeline proposed by Kim *et al.*

Figure 5 demonstrates how the sparse edge map is then used to reproduce inter-key frame images by guiding the conditional GAN (cGAN).

One of the main takeaways from this paper is the demonstration that Canny Edge Detector (or other edge detection methods) are still relevant and can provide sparse, but useful, maps to guide conditional GANs. This is useful because we can take advantage of the sparsity of edge maps to get high compression rates.

2.3 An End-to-End Compression Framework Based on Convolutional Neural Networks

This paper by Jiang *et al.* propose essentially an autoencoder, with a twist. The encoder Jiang *et al.* employ is a shallow, 3-layer CNN as seen in Figure 6. The encoder downsamples

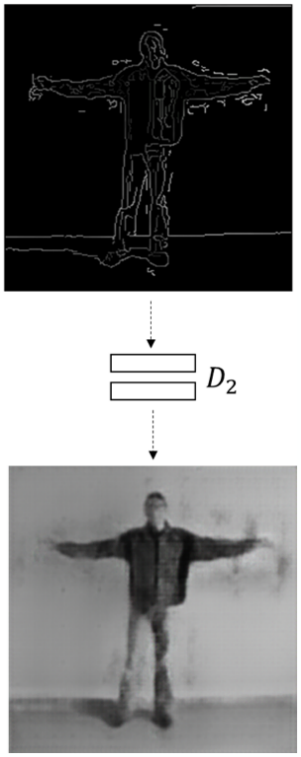

Figure 5: Down sampled edge map input to cGAN outputs recovered $\hat{X}$.

the spatial dimensions of the input image by a factor of 2 which is followed by encoding the compressed image using JPEG, JPEG2000, or BPG [11].

The compressed image is then fed into the decoder which is a deeper, 20-layer CNN. This CNN predicts the residual data that was lost during compression. This residual output is then combined with the input image which is upsampled using bicubic interpolation. The result is the recovered image.

One key takeaway from this paper is that it uses traditional image codecs (JPEG, JPEG2000, BPG) to encode the compressed representation of the image. Another is that the encoder network is actually very shallow, which should help preserve structure in the image as the encoder is fairly limited in its ability to manipulate the image.

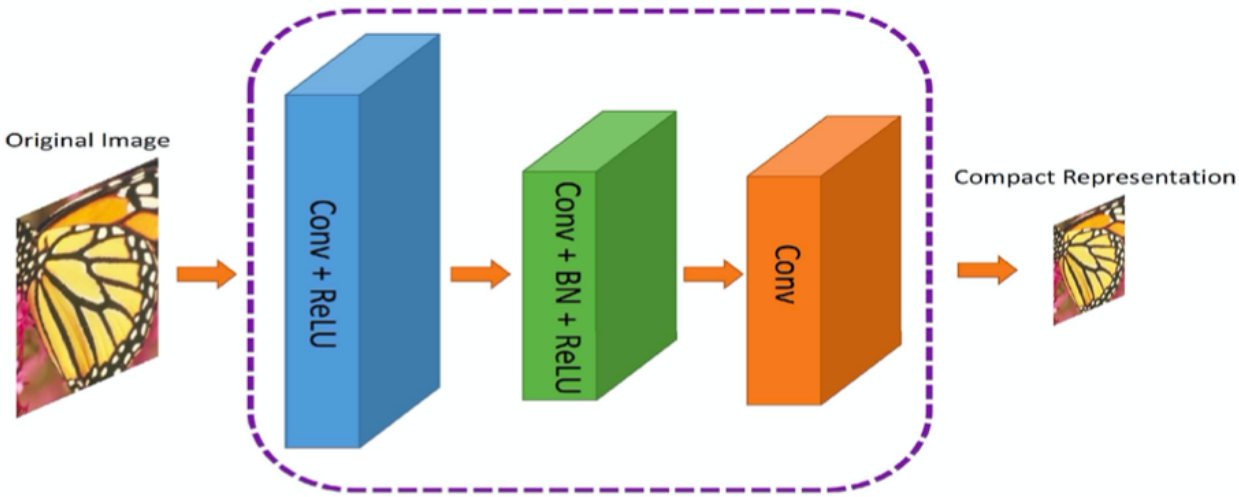

Figure 6: Down sampled edge map input to cGAN outputs recovered $\hat{X}$.

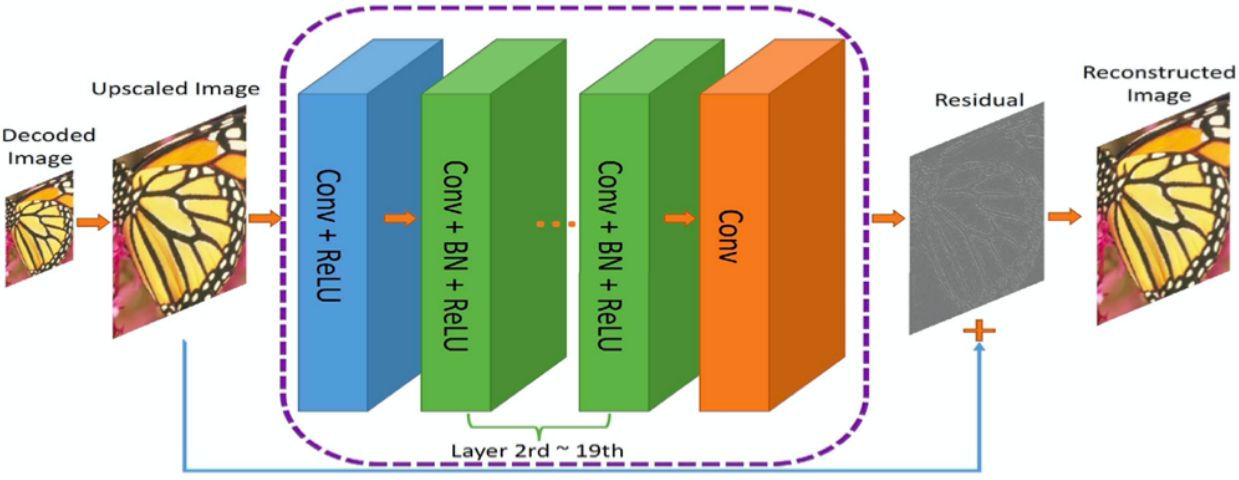

Figure 7: Down sampled edge map input to cGAN outputs recovered $\hat{X}$.

2.4 Generative Compression

This paper by Santurkar *et al.* first introduced the concept of generative compression [16]. Naturally, this paper is very similar to previously described works. However, the one thing this paper does that others exclude is use a distortion loss based on the MSE difference of x and $\hat{x}$ in the 4^{th} layer of AlexNet. This is combined with the MSE loss as another term in the distortion loss function as seen in Equation 2.5.

$$L(x, \hat{x}) = \lambda_1 \|x - \hat{x}\|_2 + \lambda_2 \left\|\mathrm{conv}_4(x) - \mathrm{conv}_4(\hat{x})\right\|_2 \tag{2.5}$$

Other works have used this approach, calling this loss feature loss or style loss. This loss function has been made famous by neural style transfer, where the MSE of the input and target are calculated through the activations of pretrained layers of VGG net. [8]

CHAPTER 3 GAN-BASED IMAGE COMPRESSION

3.1 Objective

The development of generative adversarial networks (GANs) by Goodfellow *et al* introduced a new paradigm of deep learning[9]. This approach introduced the idea that one model can attempt to recreate some data and that another model could evaluate whether or not that data was generated or real. This idea can be leveraged in training a model for image compression as the desired goal is to produce a model which is capable of generating images that seem real. This section explores two approaches to image and video compression using GANs.

3.2 Methodology

3.2.1 Full Image Based Compression

The experiment conducted in this section approaches the image compression problem by training a model to compress an entire image into a reduced representation, and then reconstructing the image from the compressed representation. In order to train a model that is capable of performing quality image recovery, a GAN-based approach is taken to guide a generator toward producing higher quality images.

Architecture The architecture for this experiment builds upon 'GANs for Extreme Learned Image Compression' by Agustson *et al*. The architecture for the encoder-decoder can be viewed in Table 2. The architecture of the discriminator can be seen in Table 3. The overview of the entire architecture can be seen in Figure 8. While the encoder-decoder remains the same as in the work of Aguston *et al*, the discriminator and loss functions have been modified. The discriminator includes the use of the spectral normalization to improve the behavior of the gradient. Spectral normalization is a technique to regularize the weights so that when the gradient is calculated, that values are no larger than a constant value. This ensures that the exploding gradient problem does not occur and largely eliminates stability issues while training our GAN [14]. Additionally, the size of the bot-

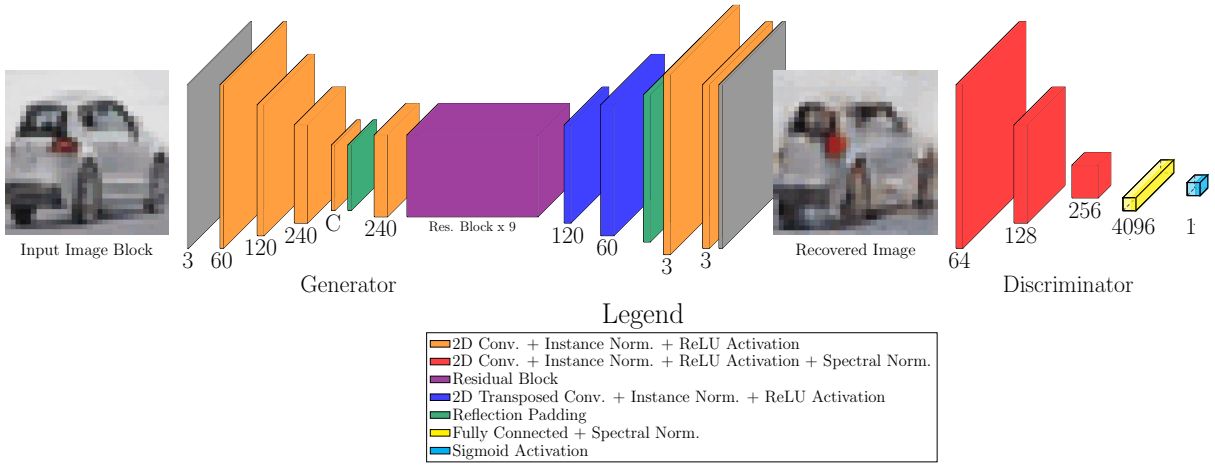

Figure 8: Architecture of image based image compression experiment. C represents the variable number of feature maps at the bottleneck layer.

tleneck layer C for this experiment was set to 2. As a result the compression ratio was 31.3469 as calculated in Equation 3.1.

$$Compression\,Ratio: \frac{3 \times 32 \times 32}{2 \times 7 \times 7} = 31.3469 \tag{3.1}$$

Loss The use of the Wasserstein GAN (WGAN) loss has been adopted for training the generator and discriminator. Because of some unique properties of the WGAN loss, the authors call the discriminator a critic instead. The idea behind WGAN loss is to not constrain the output of the critic to the range of 0.0-1.0. Traditionally, the output of a critic represents the probability that the input data is real or fake. However, WGAN does not use a sigmoid activation to compress the output into the range of a probability. Instead, the output represents the realness of the image as a real number. The larger the number the more real the critic perceives the image to be. As a result, we have two objective functions to maximize. Equation 3.2 gives us the objective function we want to maximize for the critic. The term $x^{(i)}$ is the real image, $G(x^{(i)})$ is the compressed and decompressed image, i is the index in the batch, and m is the batch size. Effectively, we want to maximize the ability of the discriminator to recognize real data samples and minimize its likelihood of being fooled by generated samples. For the generator, the objective function in Equation 3.3

Generator Architecture		
Layer Type (K= Kernel Size, S=Stride, P=Padding)	Input Shape (channel, height , width)	Output Shape (channel, height , width)
Encoder		
Conv 2D, Instance Norm, ReLU (K=7x7 S=1, P=0)	3, 32, 32	60, 26, 26
Conv 2D, Instance Norm, ReLU (K=3x3 S=2, P=1)	60, 26, 26	120, 13, 13
Conv 2D, Instance Norm, ReLU (K=3x3 S=2, P=1)	120, 13, 13	240, 7, 7
Conv 2D, Instance Norm, ReLU (K=3x3 S=1, P=1)	240, 7, 7	C, 7, 7
Quantizer	C, 7, 7	C, 7, 7
Decoder		
Reflection Pad (P=3)	C, 7, 7	C, 10, 10
Conv 2D, Instance Norm, ReLU (K=1x1 S=1, P=0)	C, 10, 10	240, 10, 10
Residual Block (K=3x3, S=1, P=1) x 9	240, 10, 10	240, 10, 10
Conv Transpose 2D, Instance Norm, ReLU (K=3x3, S=2, P=1)	240, 10, 10	120, 19, 19
Conv Transpose 2D, Instance Norm, ReLU (K=3x3, S=2, P=1)	120, 19, 19	60, 37, 37
Reflection Pad (P=3)	60, 37, 37	60, 40, 40
Conv 2D, Instance Norm, ReLU (K=7x7 S=1, P=0)	60, 40, 40	60, 34, 34
Conv 2D (K=3x3 S=2, P=1)	60, 34, 34	3, 32 32
Conv 2D (K=3x3 S=1, P=0)	3, 32 32	3, 32 32

Table 2: Generator architecture inspired by Agustsson *et al.* The residual blocks are the same as those specified by He *et al* with specific parameters specified [10].

Discriminator Architecture		
Layer Type (K= Kernel Size, S=Stride, P=Padding)	Input Shape (channels, height, width)	Output Shape (channels, height, width)
Conv 2D, Leaky ReLU, Spectral Norm (K=3x3, S=2, P=1)	3, 32, 32	64, 16, 16
Conv 2D, Leaky ReLU, Batch Norm 2D, Spectral Norm (K=3x3, S=2, P=1)	64, 16, 16	128, 8, 8
Conv 2D, Leaky ReLU, Batch Norm 2D, Spectral Norm (K=3x3, S=2, P=1)	128, 8, 8	256, 4, 4
Fully Connected Layer, Sigmoid, Spectral Norm	245, 4, 4 => 4096	1

Table 3: Discriminator architecture inspired by Aguston *et al* with Spectral Normalization improvement [14].

demonstrates that we want the generator to maximize its ability to fool the discriminator.

$$\max_{D} \mathcal{L}_{Critic} = \frac{1}{m}\sum_{i=1}^{m} D\left(x^{(i)}\right) - \frac{1}{m}\sum_{i=1}^{m} D\left(G\left(x^{(i)}\right)\right) \tag{3.2}$$

$$\max_{G} \mathcal{L}_{Generator} = \frac{1}{m}\sum_{i=1}^{m} D\left(G\left(x^{(i)}\right)\right) \tag{3.3}$$

In order to ensure that the model converges by using this loss function, after every network update iteration, the weights of the generator network are clipped into the range of $[-0.01, 0.01]$. More about the mathematics behind why can be found in the original WGAN paper [5]. However, the main idea is that because we are optimizing a function that is non-differentiable everywhere, we can instead artificially limit the maximum value of the derivative to some constant such that we produce a derivative/gradient that is effectively continuous. In addition to the WGAN loss, the generator also uses Mean Squared Error (MSE) or L_2-Norm, as defined in Equation 3.5, to calculate distortion loss. The two loss functions are combined to form Equation 3.4.

$$\max_{G} \mathcal{L}_{Generator\,+\,MSE} = \mathcal{L}_{Generator} - \mathcal{L}_{MSE} \tag{3.4}$$

$$\mathcal{L}_{MSE} = MSE(x, \hat{x}) = \frac{1}{n}\sum_{i=1}^{n} \left(x_i - \hat{x}_i\right)^2 , n = \#ofpixels \tag{3.5}$$

3.3 Image Block Based Compression

In this section a block based approach is taken to compress images as well as video frames. Taking inspiration from JPEG's block based compression approach, this approach is designed to reduce the computational complexity and size of the neural network model by learning a subset of the video frame. The block based approach will allow the model to learn within a smaller domain. In this case, how to compress 3x32x32 blocks into a compressed representation for robust encoding and decoding. By reducing the size of the

learning space, the hope is that the model will learn image blocks with high fidelity so that the model can then be applied iteratively to compress an image or video frame.

Architecture Figure 9 outlines the architecture of the model used in this section. The GAN architecture includes generator and discriminator submodules for learning. The generator includes an encoder and decoder used to compress and recover video frame blocks. The encoder consists of three blocks of downsampling each consisting of 2D convolution, followed by 2D instance normalization, followed by the ReLU activation. The instance normalization layer is similar to a batch normalization layer, with the difference that the statistics are calculated on a per image basis, as opposed to on a per batch basis. The 2D convolution has a stride of 2, padding of 1, and a kernel size of 4x4. This reduces the video frame block's spatial dimensions by a factor of 2 for each layer. As a result, the compressed representation has a spatial dimension of 4x4. Variable compression is possible by varying the number of feature maps at the bottleneck layer. In this experiment, $C\epsilon\{4, 8, 16\}$, represents the number of feature maps at the compressed layer. This allows for variable compression at an architectural level. By varying the C parameter, compression ratios of 48, 24, and 12 can be achieved.

The decoder submodule is a mirror image of the encoder module. It consists of 3 blocks of upsampling each consisting of 2D transposed convolution, instance normalization, and ReLU activation. This recovers a video frame block from the compressed representation back into its original 3x32x32 shape.

The discriminator is designed to guide the generator's training process to produce more realistic recovered video frame blocks. The discriminator is effectively a classifier that determines whether or not the video frame block it is looking at is from the ground truth or from the output of the generator. The generator consists of three downsampling blocks consisting of 2D convolution, batch normalization, and leaky ReLU activation. The output is then flattened into a single vector and fed into a fully connected layer which outputs a single value, which is then fed into a sigmoid activation for real/fake classification.

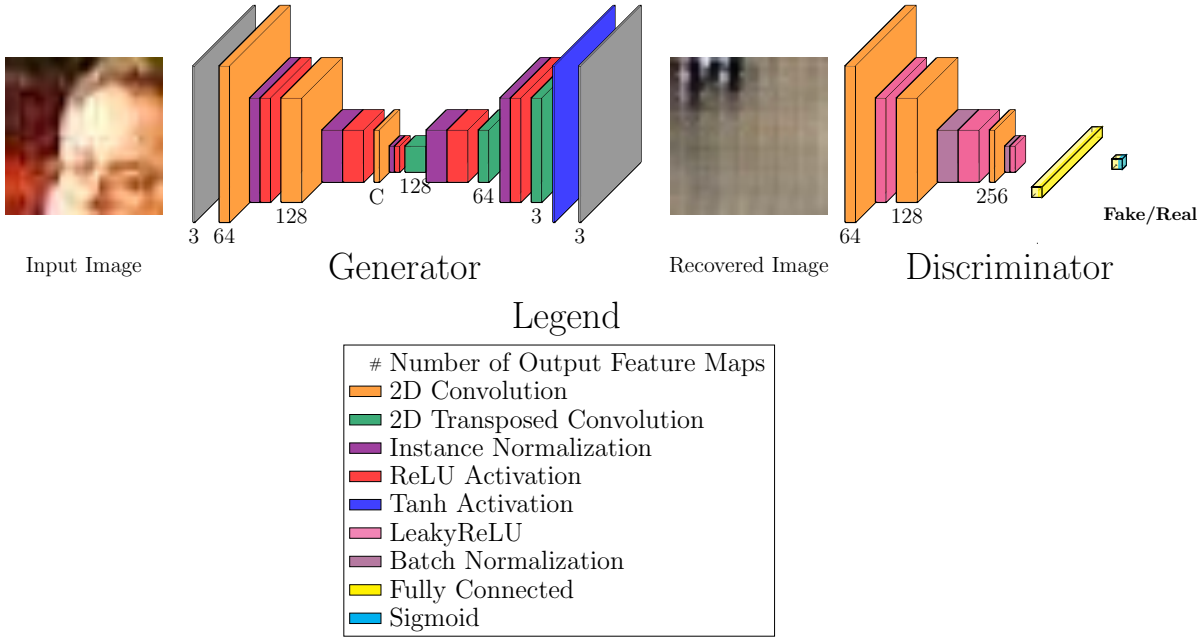

Figure 9: Architecture of block based image compression experiment. C represents the variable number of feature maps at the bottleneck layer. The recovered image does not resemble the input image, and as a result is a major shortcoming of this GAN based approach.

Loss The objective function used to train the GAN can be seen in Equation 3.9. It is a modified version of the standard GAN objective function that includes a term for the perceptual quality of the generator's output.

$$\begin{aligned} BCE(\hat{x}, \hat{y}) = mean(L), L = \{l_1, \ldots, l_N\}^\top, \\ l_n = y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n) \quad \forall x_n \, \epsilon \, \hat{x}, \; y_n \, \epsilon \, \hat{y} \end{aligned} \tag{3.6}$$

$$\mathcal{L}_{Discriminator} = BCE(D(G(x)), 0) + BCE(D(x), 1) \tag{3.7}$$

$$\mathcal{L}_{Generator} = \lambda_{MSE} \times MSE(x, G(x)) + \lambda_{GAN} \times BCE(D(G(x)), 1) \tag{3.8}$$

$$\min_G \max_D \mathcal{L}(D, G) = \mathcal{L}_{Generator} + \mathcal{L}_{Discriminator} \tag{3.9}$$

Hyperparameters As mentioned previously, the generator supports variable compression by changing the number of latent feature maps C in the compressed representation. In this experiment $C \epsilon \{4, 8, 16\}$ and thus supports a compression ratio of 48, 24, and 12, respectively. The term λ_{MSE} in Equation 3.8 controls the extent to which the perceptual loss has an impact on the overall loss of the generator. In this experiment, the term $\lambda_{MSE} \, \epsilon \, \{0.1, 1.0, 10\}$. Similarly, the term λ_{GAN} controls the extent to which the discriminator contributes to the loss of the generator and similarly $\lambda_{GAN} \, \epsilon \, \{0.1, 1.0, 10\}$. Additionally, the GAN can be trained in three different modes: GAN only, autoencoder head start, and autoencoder only. The GAN only mode consists of the experiment as described. The autoencoder head start mode jump starts the generator by training it as an autoencoder for a fixed number of epochs (in this case 50) before introducing the discriminator and GAN loss term. Finally, the autoencoder only mode trains the generator only and completely ignores the discriminator and discriminator loss terms.

3.4 Experiments

3.4.1 Full Image Based Compression

Training Details Both the generator and discriminator are optimized using the Adam gradient descent optimizer available in Pytorch. The generator's optimizer has a learning rate of $1e-4$ and the discriminator's optimizer has a learning rate of $3e-4$. The experiment runs for a total of 200 epochs, where the first 50 epochs only the MSE error is used. As a result, the first 50 iterations of training is equivalent to training an autoencoder. The batch size is 16. Network weights are initialized using Xavier/Glorot initialization. The training set consists of 50,000 images and the testing set consists of 10,000 images.

Results Table 4 demonstrates using the discriminator in addition to the generator provides a small increase in PSNR and SSIM metrics. These are metrics for perceptual similarity of two images, one that is the ground truth x, and the other that has been compressed and recovered in some fashion is $\hat{x}$. The equation for PSNR can be found in Equation 3.10. The term MAX_I^2 refers to the maximum value in the image. In the case of Pytorch tensors, the maximum value is 1.0, so the value is replaced with 1. The equation for SSIM can be found in Equation 3.11. SSIM calculates the statistical similarity between two $N \times N$ image subblocks. As a result, the mean μ_x and μ_y is calculated in addition to the variance σ_x^2 and σ_y^2 for comparison of image x versus y. The terms C_1 and C_2 are small values to prevent divison by zero. Perceptually speaking, the GAN results seem sharper across the board. However, in some cases the image becomes more distorted than in the case of the autoencoder only method.

$$PSNR = 10 \times log_{10}\left(\frac{MAX_I^2}{MSE}\right) \Rightarrow 10 \times log_{10}\left(\frac{1}{MSE}\right) \tag{3.10}$$

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1) + (2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{3.11}$$

Figure 10 shows the resulting images from a GAN-based model versus an autoencoder.

Autoencoder vs. GAN Results PSNR/SSIM		
	PSNR	SSIM
Autoencoder	11.0976	0.4457
WGAN+SN	11.3479	0.4610

Table 4: The WGAN + Spectral Normalization method has higher perceptual quality metrics than solely training by using an autoencoder.

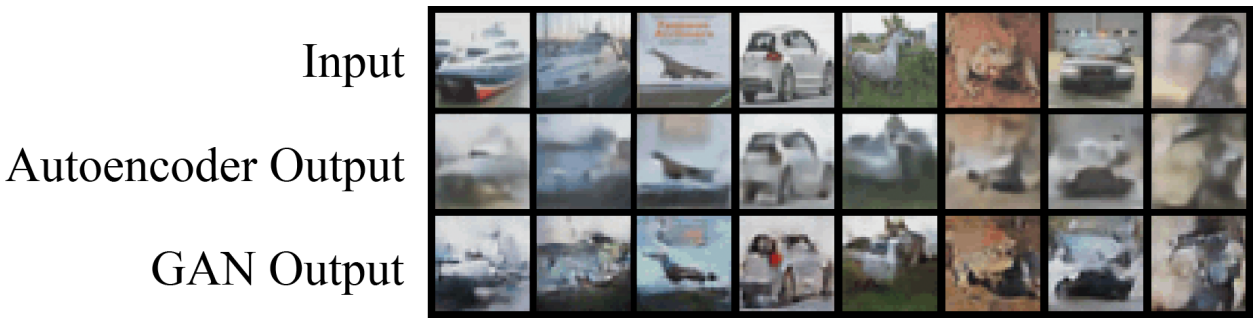

Figure 10: The GAN-based approach has generally sharper results, however, the GAN images are also more subject to various forms of warping. For example, the rightmost photo of an emu is certainly sharper in the GAN image, however, it is also more distorted.

Table 5 demonstrates the time required to encode and decode a $3 \times 32 \times 32$image.

Image Encoding/Decoding Time (on Nvidia 1060 GTX 6GB)	
	Time (in ms)
Encoding + Decode	0.917 ms/image

Table 5: Performance of encoding/decoding an image.

3.4.2 Image Block Based Compression

Training Method To train the GAN, the standard GAN training approach was taken. This involves training the generator and discriminator in alternation to ensure that one does not exceed the capabilities of the other by having a chance to learn for a longer period of time. As described in Equation 3.7 and Equation 3.8, the $\mathcal{L}_{Discriminator}$ and $\mathcal{L}_{Generator}$ terms are dependent on the adversarial objective. The generator is attempting to minimize the perceptual difference and ability of the discriminator to discern the difference between the ground truth x and the images it produces $\hat{x} = G(x)$. On the other hand, the discriminator is attempting to maximize its ability to discern the difference between generated images

$\hat{x} = G(x)$ and real images x.

Dataset The GAN was trained on a subset of the Youtube User Generated Content (UGC) dataset [18]. The subset consisted of video frames from any video that was of the resolution 640x480 pixels. This subset would then be randomly sampled for 3x32x32 video frame blocks.

Implementation Both the generator and discriminator utilized the default Adam optimizer available in Pytorch to perform gradient descent. Additionally, the Pytorch Lightning framework was utilized to automate various menial tasks and hyperparameter searching. Each experiment model was trained on an Nvidia Quadro RTX 6000 for 200 epochs.

Results Unfortunately, the results are lackluster and the GAN succumbs to various problems that GANs are notorious for. For example, in Figure 11 the GAN is unable to recover images that are more complex than a single color, a form of training collapse in which only the simplest translation has been learned. Additionally, the two rightmost images in the GAN output in Figure 11 are similar despite having different inputs. This indicates that the GAN has suffered from mode collapse in which the GAN can only produce the same image over and over despite varying inputs. This is because the GAN has found that the best way to optimize the objective function is to use a single recovered video frame block that fools the discriminator the most.

Failure Analysis Further analysis indicates that the discriminator is able to learn that the generator's outputs are fake almost instantly. For example, Figure 14 is a Tensorboard log of the F1 score (Equation 3.14) of the discriminator. In effect, this metric tells us how accurately the discriminator is able to tell the difference between real and fake images. In this case, we can observe that after the autoencoder head start has expired and the discriminator is introduced, the discriminator is capable of telling the difference between real and fake at very high rates nearing 100%. This means that the generator will not be receiving useful feedback from the generator because no matter what it does it will fail to fool to the discriminator, and as a result the generator has lost the adversarial

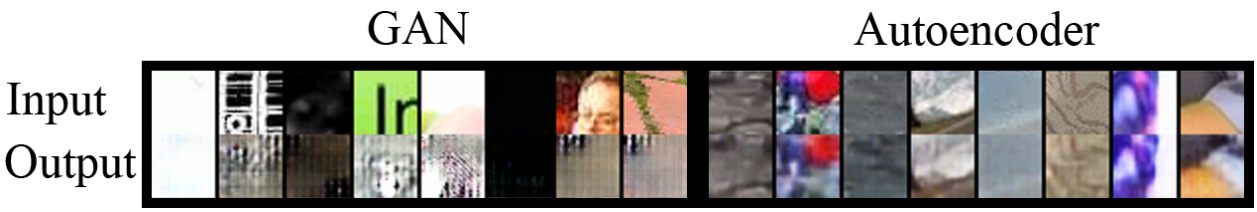

Figure 11: The GAN based approach is only able to learn very simple representations, such as the all white or all black video frame blocks. Otherwise, the GAN based approach exhibits symptoms of mode collapse and training collapse, whereas the autoencoder based approach recovers a blurry version of the image. This model's PSNR has a value of 8.144 dB, which is among the highest in the GAN based approach.The hyperparameters are $\lambda_{MSE} = 10$, $\lambda_{GAN} = 1$, autoencoder head start is used, and $C = 16$, so the compression ratio is 12.

contest. Furthermore, we can see that the instant that the discriminator is introduced and the autoencoder head start expires, the PSNR and SSIM visual quality measures drop dramatically.

$$Precision = \frac{True\,Positive}{True\,Positive + False\,Positive} \tag{3.12}$$

$$Recall = \frac{True\,Positive}{Positive} \tag{3.13}$$

$$F_1 score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{3.14}$$

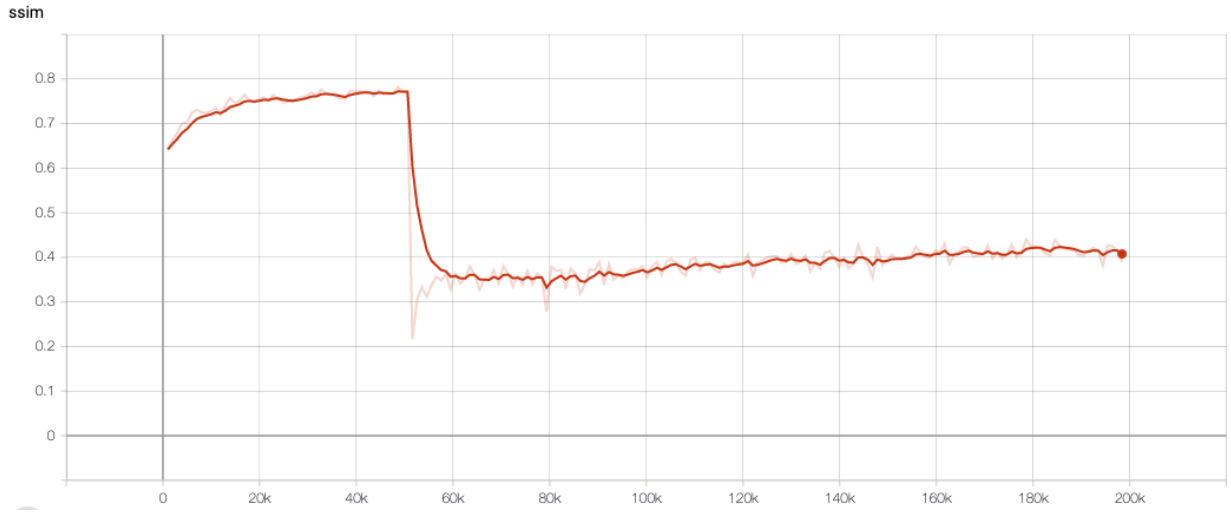

Figure 12: Almost immediately after the discriminator is introduced, the visual quality plummets dramatically.

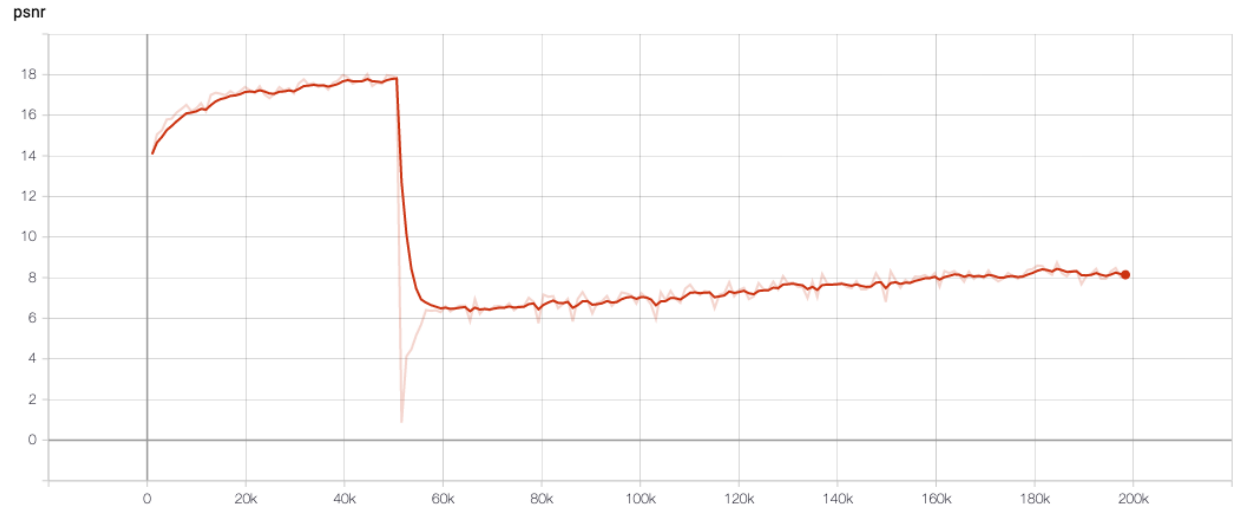

Figure 13: Almost immediately after the discriminator is introduced, the visual quality plummets dramatically.

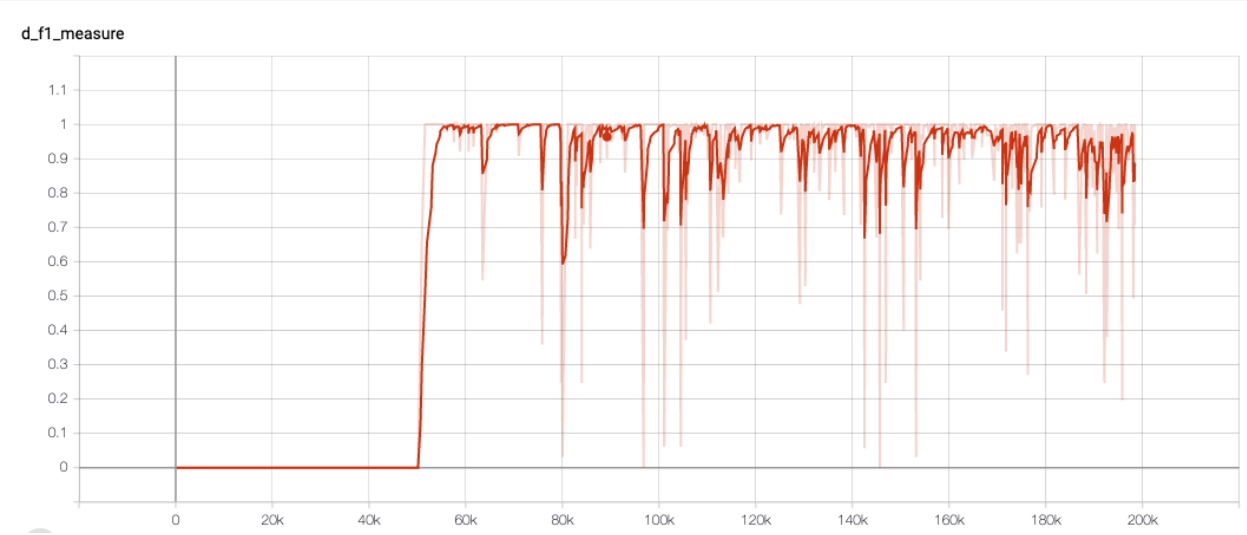

Figure 14: Almost immediately after the discriminator is introduced, the discriminator is able to distinguish between real and fake images at rates near 100% accuracy, thus providing no useful feedback to the generator.

CHAPTER 4 IMAGE CODING FOR MACHINE CLASSIFICATION

4.1 Objective

A growing consumer of video and image data stem from so called machine-to-machine applications [6]. This paradigm includes applications such as video surveillance systems and self-driving car cameras where humans are not viewing the images or videos. Nonetheless, these applications require vision of some sort. However, these applications typically need images to produce bounding boxes, segmentation maps, regions of interest, class predictions, or other insights derived from the image itself. However, these applications do not necessarily need to view images the same way humans do. As a result, encoding and decoding entire images and videos turns out to be a very costly and wasteful approach to machine-to-machine applications. Instead, images can be partially processed, and then transmitted, as opposed to transmitted then processed. This section explores various deep learning models for representing images in the most compressed manner possible specifically for the purpose of image classification.

4.2 Methodology

4.2.1 Concatenated Autoencoder and Classifier

An autoencoder and classifier are jointly trained with the dual purpose of image reconstruction as well as image classification. The term concatenated refers to training an autoencoder and image classifier by combining the two networks and training it as one end-to-end network.

Architecture Figure 15 illustrates the architecture of the concatenated autoencoder and classifier. This autoencoder has 3 downsampling 2D convolutional layers which reduces the original image of dimensions 3x224x224 down to Cx28x28. In effect, the spatial dimensions are reduced by a factor of 8, and by varying the number of feature maps C in the channel dimension, varying compression rates can be achieved. The compression ratio can be calculated via Equation 4.1.

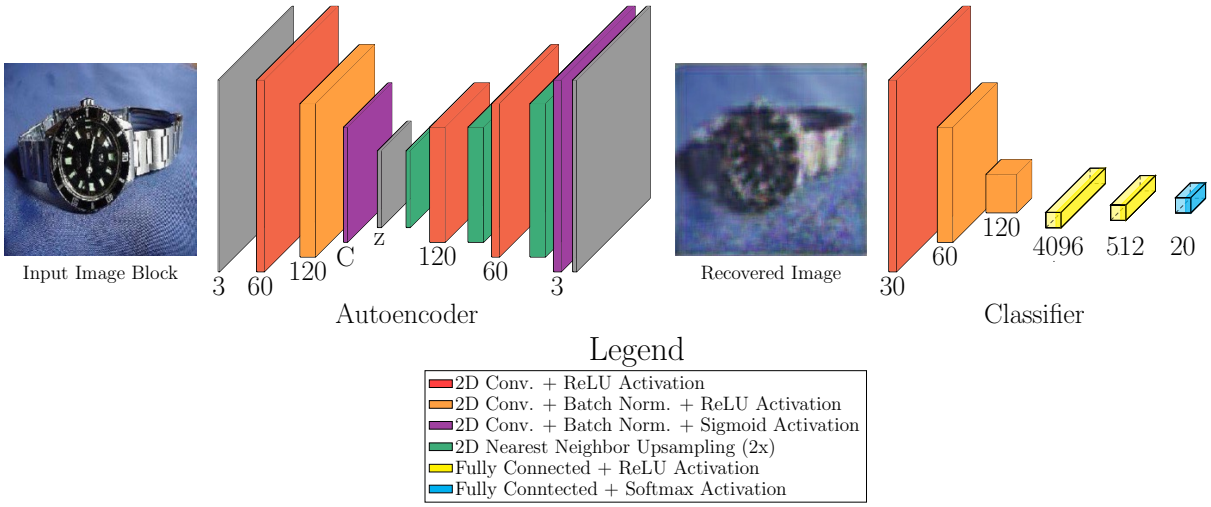

Figure 15: The architecture of the proposed architecture for both the concatenated and separated models.

$$Compression\ Ratio = \frac{224^2 \times 3}{28^2 \times C} \tag{4.1}$$

Once the image has been converted to its compressed representation z, this vector is then fed into the decoder to produce a recovered image $\hat{x}$. This is accomplished through the use of 3 layers of 2D nearest neighbor upsampling and 2D convolution. Odena *et al* find that using a 2D nearest neighbor upsampling followed by a 2D convolutional layer reduces the checkerboard artifacts that result from using the 2D transposed convolution [15]. Once $\hat{x}$ has been generated, it is fed into the classifier for classification to produce the predicted label $\hat{y}$.

Loss The concatenated experiment model in Figure 15 was trained by feeding an input image x into the autoencoder and then directly feeding the recovered image $\hat{x}$ into the classifier. The result of feeding the reconstructed image $\hat{x}$ into the classifier is a predicted class $\hat{y}$, which is then compared with the ground truth label y. A loss is then calculated based on the quality of the image reconstruction and the classification accuracy according to Equation 4.2. This is similar to the previous GAN-based experiment, where a reconstruction loss and classification loss is used to train the network. However, the main difference

in this approach is that the network is not trained as a generator and discriminator. Instead, it is trained end-to-end in such a way that instead of being an adversarial approach, the two networks are working together to achieve two separate goals.

$$\mathcal{L}_{Autoencoder/Classifier} = \lambda_{MSE} \times MSE(x, \hat{x}) + BCE(\hat{y}, y) \tag{4.2}$$

4.2.2 Separated Autoencoder and Classifier

In this set of experiments, an autoencoder is trained first to compress and reconstruct images. Then, using the pretrained, frozen weights of the autoencoder the classifier is trained.

Architecture The separated model uses the same architecture as the concatenated model as in Figure 15.

Loss Because the separated autoencoder and classifier are trained separately, this approach can be thought of as training two independent networks. As a result the loss function for the autoencoder is simply the $MSE(x, \hat{x})$ and the loss for the classifier is $BCE(y, \hat{y})$.

Quantization and Compression Both the concatenated and separated models can further compress the latent vector z that the encoder submodule of the autoencoder produces. This latent vector z, can be further compressed by quantizing and then using an entropy coding algorithm in addition to the reduced spatial dimensions of the original image. In this experiment, the output layer of the encoder submodule features a sigmoid activation. This means that the latent vector z is in the domain 0.0-1.0. To compress this vector, the latent vector is scaled by 255 so that the vector is now in the domain of 0-255. Then, the vector is rounded to the nearest integer. A Huffman code is then applied to encode the quantized vector down to the vector's entropy. In practice, the reduced, quantized, and compressed vector would then be saved to disk or sent over the network. However, in this case, the vector is instead recovered by decoding the compressed representation back

into the quantized vector. This quantized vector is then fed into the decoder submodule of the autoencoder to recover the image $\hat{x}$. In order to ensure that the decoder is not "surprised" by the quantized latent vector $q(z)$ at test time, the decoder submodule is trained with noise from the uniform distribution $U(-0.5, 0.5)$ to simulate the rounding that occurs during quantization.

4.2.3 Hydra-net Based Autoencoder and Classifier

Inspired by Duan *et al* and Andrej Karpathy's talk at Pytorch Devcon 2019 about Tesla's self driving approach, this set of experiments implements an architecture dubbed hydra-net for having a single shared backbone and multiple heads that perform various tasks [6] [12]. In this case, there are two heads that implement image reconstruction and image classification and a single backbone feature extractor.

Architecture The architecture used can be seen in Figure 16. In addition to the the use of a hydra-net style approach, this model also forwards the feature maps of the encoder/feature extractor to the classifier. However, in order to do so, these feature maps need to be downsampled further so the additional convolutional layers outside of the classifier downsample the encoder's feature maps. Then these feature maps and the latent vector z are concatenated channel-wise as an input to the classifier. Empirically, this has resulted in about an additional 2-3% gain in testing classification accuracy versus not including the downsampled feature maps of the encoder. This set of experiments is also capable of quantization and compression as described previously.

Loss This model is trained using the loss function in Equation 4.3, which is the average of the BCE loss of the classifier head and the MSE loss of the image reconstruction head. The term $\hat{y}$ is the predicted class label from the classifier, which is generated by feeding the latent feature vector z into the classifier. The term $\hat{x}$ is the output of the decoder head.

$$\mathcal{L}_{Hydra} = \frac{BCE(\hat{y}, y) + MSE(x\,, \hat{x})}{2} \tag{4.3}$$

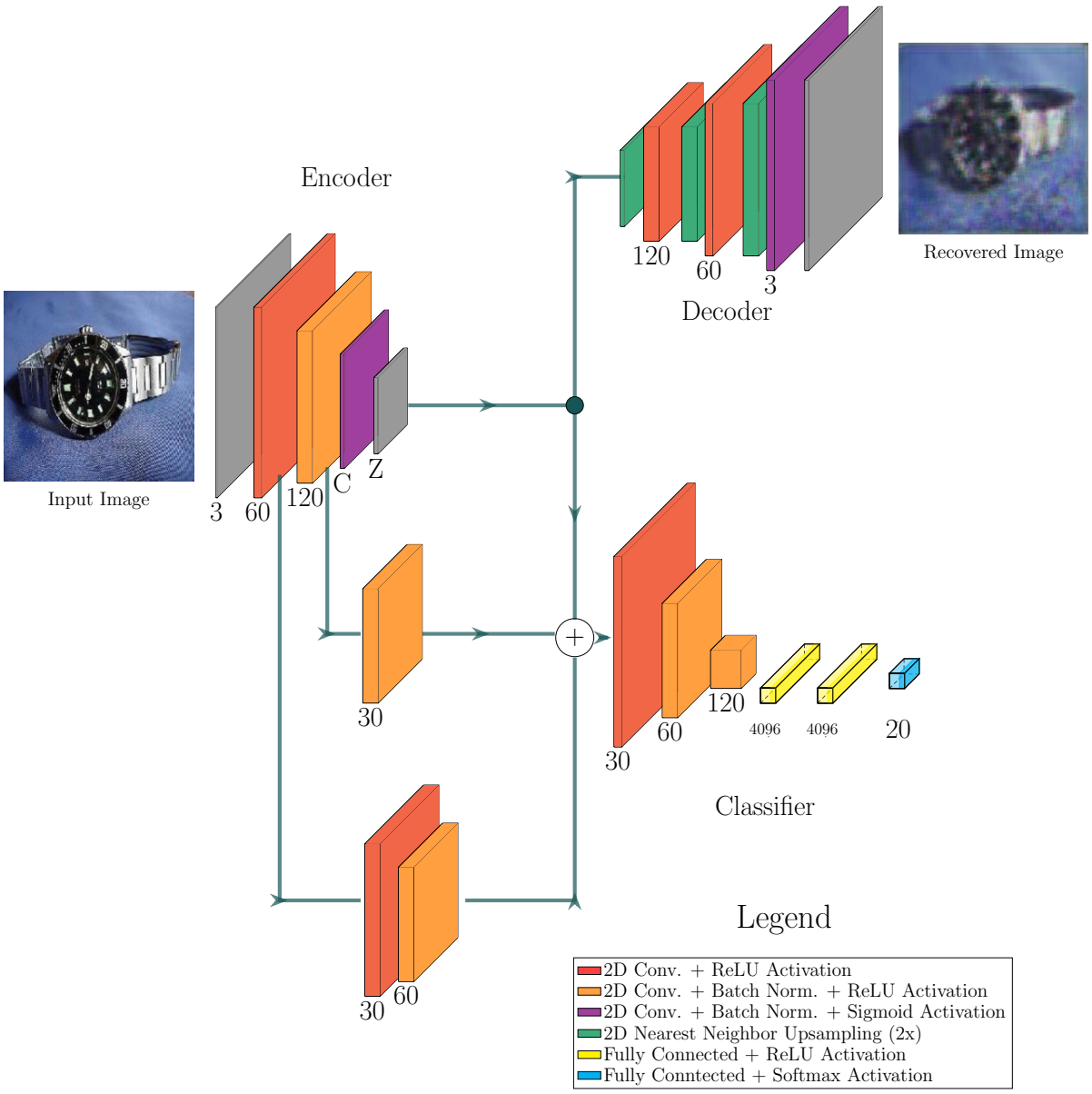

Figure 16: The architecture of the proposed hydra-net model.

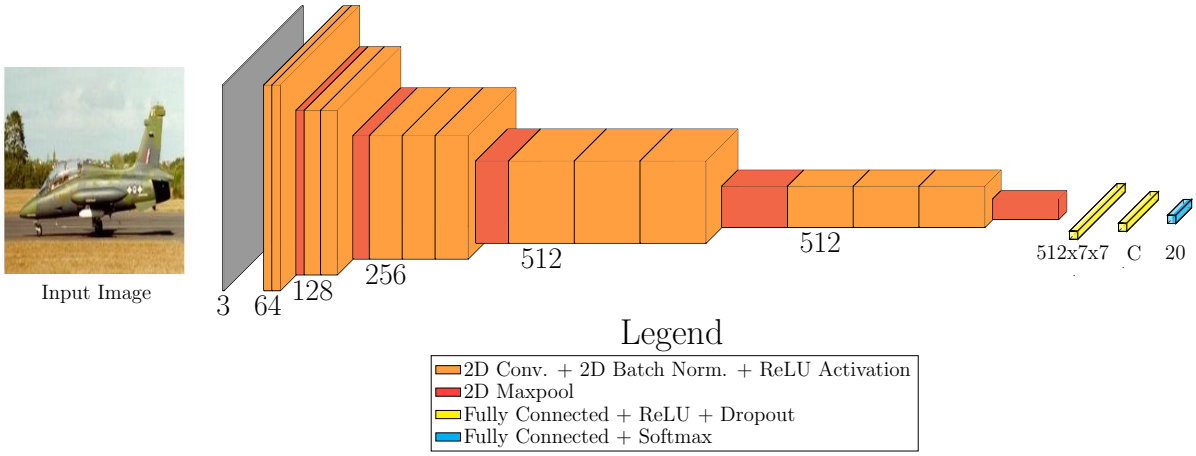

Figure 17: The architecture of VGG 16 with Batch Normalization.

4.2.4 VGG16 with Batch Normalization Baseline Classifier

VGG16 with Batch Normalization (VGG16BN) is selected as a baseline model to evaluate against other experiments. The classification accuracy of VGG net is compared to the classification accuracy of the previous experiments.

Architecture The architecture of VGG16 with Batch Normalization can be seen in Figure 17[17]. The rightmost value C represents a modified version of VGG net in which the fully connected layer can be adjusted to reduce the number of features, and in effect increase the compression ratio of the model. Typically, VGG net has an intermediate representation of a length 4096 vector, but by varying this vector size C, we can adjust the compressed representation of the data.

Loss VGG16BN is trained with the loss function in Equation 4.4. The network is solely trained on the accuracy of its classification predictions.

$$\mathcal{L}_{VGG} = BCE(y, \hat{y}) \tag{4.4}$$

Airplanes-101	Motorbikes-101	Faces-Easy-101	T-Shirt	Hammock
Billiards	Horse	Ladder	Bathtub	Binoculars
Gorilla	People	Mushroom	Grapes	Watch-101
Mattress	Leopards-101	Light-house	Mussels	Soccer Ball

Table 6: The 20 most populous classes selected from the Caltech 256 dataset used to construct an evenly distributed dataset.

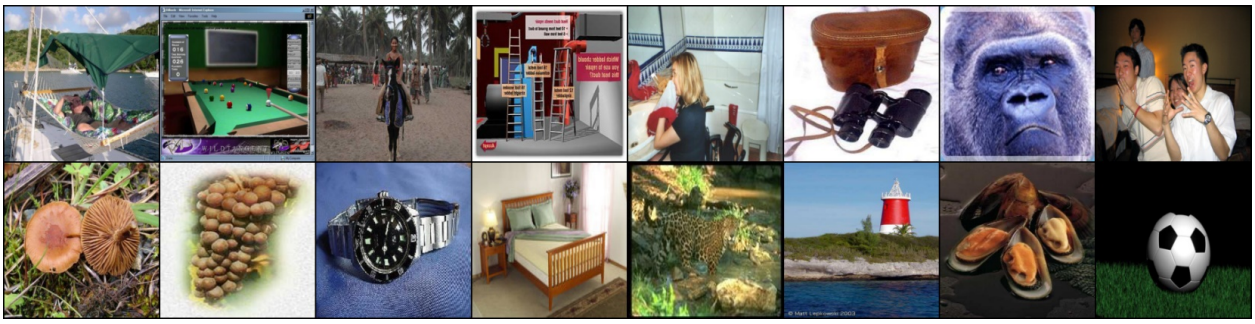

Figure 18: A sample of images from the training set.

Pretrained and Training from Scratch Because VGG net was used in ImageNet Large Scale Visual Recognition Competition (ILSVRC), it has pretrained weights that can be useful for boostrapping other models. In this case, we can use the weights from the pretrained network and apply it to the classification problem we have at hand. Additionally, by initializing the weights of VGG net using Xavier/Glorot initialization, we can train the network from scratch and evaluate its performance without pretrained weights.

4.3 Experiments

4.3.1 Dataset

The dataset used for this set of experiments was a subset of the Caltech 256 dataset. The 20 categories with the most images were selected, and then a random sample of 80 images were taken from these 20 categories resulting in a total of 1600 images total. The list of categories that were used are listed in Table 6. Figure 18 includes a sample of some of the images from this dataset. All images were resized to 3x224x224 regardless of previous aspect ratio or dimensions.

4.4 Training Details

For all networks in this experiment, the Adam optimizer available in Pytorch with default parameters is used for gradient descent optimization.

Concatenated Autoencoder and Classifier The concatenated model is trained on 960 training images, using 240 images for validation and early stopping. Early stopping monitors the validation accuracy of the classifier and stops once the accuracy stops increasing for more than 5 epochs. After training, 400 images are used for calculating the test accuracy of the model.

Separated Autoencoder and Classifier For the separated model, the autoencoder is first trained using only the MSE loss. The autoencoder is trained on 400 images (20 from each class) for 20 epochs. Then, the autoencoder's weights are frozen and the classifier is trained by feeding the image x through the autoencoder to get $\hat{x}$. Then, $\hat{x}$ is fed into the classifier and the BCE loss is calculated between the predicted and actual label. The classifier uses 640 training images, and 160 validation images for early stopping to stop training if the validation accuracy does not improve for more than 5 epochs. Finally, the classifier is tested on 400 images to calculate the accuracy of the model.

Hydra-net Based Autoencoder and Classifier The hydra-net based model is trained on 960 training images, and 240 validation images with early stopping. Then, 400 testing images calculate the test accuracy of the classifier.

VGG16 with Batch Normalization Baseline Classifier The VGG16BN model is trained with 960 training images, and 240 validation images with early stopping. Then, 400 images calculate the test accuracy of the classifier.

4.4.1 Results

Figure 19 illustrates the classification performance of each model by varying the bits per pixel (bpp) for the compressed representation of an image. The objective is to have as low

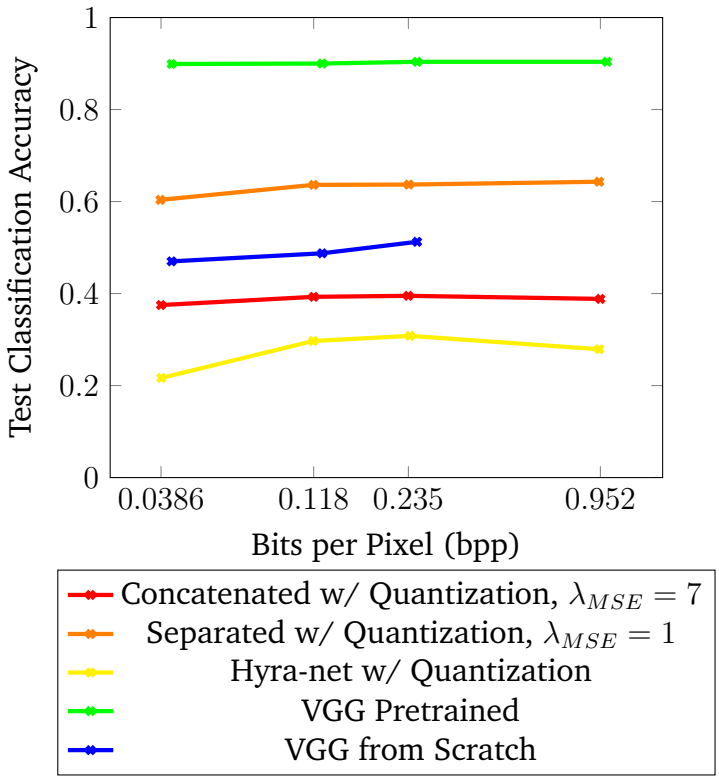

Figure 19: Bits per pixel versus the test classification accuracy for each model proposed for image coding for machine classification.

a bpp as possible while maintaining a high classification accuracy rate. Typically, the bits per pixel is 8 bits per pixel because an integer that represents the range 0-255 requires $log_2(2^8) \rightarrow 8$ bits. Equation 4.5 demonstrates how to calculate bpp The models that have been selected for display in this figure demonstrate the highest performing hyperparameters for each experiment type. Notably, the models that featured quantization and add uniformly distributed noise during training outperformed the models without quantization and noise. By adding noise, this forces the decoder to learn more robust representations so as not to trivially memorize how to reconstruct an image given a latent vector.

$$Bits\,per\,Pixel = \frac{total\,bits\,in\,compresseed\,representation}{number\,of\,pixels\,in\,image} \rightarrow \\ e.g.\,\frac{142020\,bits}{224 \times 224 \times 3\,pixels} \rightarrow 0.943bpp \tag{4.5}$$

Performance The results demonstrate that the highest performing model by far is pretrained VGG16BN. However, when VGG16BN is not pretrained, it is actually outperformed by the separated autoencoder and classifier approach. This clearly indicates the importance of unsupervised pretraining or transfer learning. Without the pretrained weights from training on ImageNet, VGG16BN cannot outperform the much simpler model used in the separated autoencoder classifier experiment. Furthermore, unlike what was hypothesized, the separated model outperforms the concatenated model. Because the separated model allows the classifier to focus solely on updating the its weights to optimize for classification accuracy, it seems that the separated model performs better. However, future work should focus on ensuring that the end-to-end approach is given more time and data for training so that the autoencoder and classifier can have a stronger learned connection. Finally, the hydra-net approach performs the worst. Something to note in the case of hydra-nets is that they are designed to amortize the computation of feature extraction for multiple tasks. Naturally, it makes sense that the amortization of feature extraction would reduce the capability of a model because the feature extractor is not specialized. Instead, the shared backbone must learn how to extract useful general features for many tasks, but because they are general, it reduces performance for each individual task.

Furthermore, Figure 20 demonstrates the ability for pretrained VGG16BN to achieve extreme compression rates with high classification accuracy far beyond the capabilities of the proposed concatenated, separated, and hydra-net models. This approach simply took a pretrained VGG net and remove and retrained the final fully connected layers of VGG net. VGG net is able to achieve a higher classification accuracy than any proposed method with an order of magnitude increased compression ratio. This suggests that for future experiments, utilizing VGG net as a feature extractor is an excellent baseline to start with.

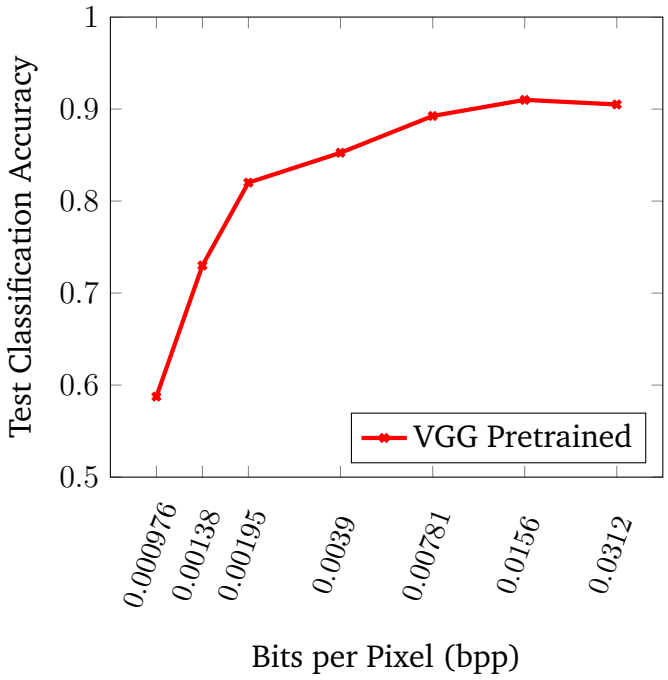

Figure 20: A bitrate of 0.00781 represents a compression ratio of Ĩ024x. Despite this massive compression, VGG net is able to achieve 89.25% test accuracy. This demonstrates the impressive ability of the VGG net feature extractor.

CHAPTER 5 CONCLUSION

The purpose of this research was to build upon current state of the art technologies in image/video compression and classification. In doing so, it has been demonstrated that GANs are capable of outperforming CNN based models for full image compression. In this case, the GAN-based SSIM outperformed the autoencoder-based approach by 3.43%. However, the experiments conducted indicate that full image based approaches are more promising than block based approaches. Additionally, it has been shown that VGG16 with Batch Normalization is capable of offering extremely efficient feature compression with compression ratios of 1000x with little to no difference in classification accuracy. In addition to these conclusions, I would like to offer some lessons learned and suggestions for improvement.

5.1 Suggestions for Improvement

Upon reflection, there are many different approaches that can be taken to build upon the current experimental results.

GAN-Based Image Compression

- Reuse the model from 'Full Image Based Compression' on the block based approach, and see if it yields better results.
- Move from CIFAR-10 to a dataset that offers higher resolution images.

Image Coding for Machine Classification

- The concatenated, separated, and hydra-net autoencoder classifiers were all very small compared to VGG net. Increasing the depth should allow these models to achieve higher accuracy rates.
- In addition to the number of features, the dataset size must be dramatically increased. 1600 images is insufficient for the benefits of deep learning models to fully take effect.
- As mentioned before, doing either unsupervised pretraining or transfer learning is

crucial for high performance as training a feature extractor from scratch makes the task at hand much more difficult.

- Seeing as noise improved all models, it would also be worthwhile to try adding dropout layers to introduce more noise to help with robust encoding.

REFERENCES

[1] Sandvine, 2018.

[2] Imagenet, May 2020.

[3] Jpeg, Jun 2020.

[4] E. Agustsson, M. Tschannen, F. Mentzer, R. Timofte, and L. V. Gool. Generative adversarial networks for extreme learned image compression, 2018.

[5] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan, 2017.

[6] L.-Y. Duan, J. Liu, W. Yang, T. Huang, and W. Gao. Video coding for machines: A paradigm of collaborative compression and intelligent analytics, 2020.

[7] EETimes. Emerging h.264 standard supports broadcast video encoding -, Jan 2003.

[8] L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style, 2015.

[9] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks, 2014.

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[11] F. Jiang, W. Tao, S. Liu, J. Ren, X. Guo, and D. Zhao. An end-to-end compression framework based on convolutional neural networks, 2017.

[12] A. Karpathy. Pytorch at tesla - andrej karpathy, tesla.

[13] S. Kim, J. S. Park, C. G. Bampis, J. Lee, M. K. Markey, A. G. Dimakis, and A. C. Bovik. Adversarial video compression guided by soft edge detection, 2018.

[14] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral normalization for generative adversarial networks, 2018.

[15] A. Odena, V. Dumoulin, and C. Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016.

[16] S. Santurkar, D. Budden, and N. Shavit. Generative compression, 2017.

[17] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[18] Y. Wang, S. Inguva, and B. Adsumilli. Youtube ugc dataset for video compression research. 2019.