Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Predictive Image Compression using Autoencoders

Henrique Costa Jung

Dissertação apresentada como requisito parcial para
conclusão do Mestrado em Informática

Orientador
Prof. Dr. Bruno L. Macchiavello Espinoza

Brasília
2021

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Predictive Image Compression using Autoencoders

Henrique Costa Jung

Dissertação apresentada como requisito parcial para
conclusão do Mestrado em Informática

Prof. Dr. Bruno L. Macchiavello Espinoza (Orientador)
CIC/UnB

Prof. Dr. Teófilo E. de Campos
CIC/UnB

Prof. Dr. Eduardo A. B. da Silva
UFRJ

Prof.a Dr.a Genaína Nunes Rodrigues
Coordenadora do Programa de Pós-graduação em Informática

Brasília, 27 de janeiro de 2021

Dedicatória

Esse trabalho é dedicado aos meus pais, Luiz e Eusângela, por terem apoiado todas as minhas escolhas (mesmo as ruins que não deveriam ter apoiado).

Agradecimentos

Gostaria de agradecer minha família, grande demais para ser listada, mas em especial as minhas sobrinhas e meu sobrinho: Clara, Samuel e Alice, que me atrapalharam tantas vezes.

Gostaria de agradecer aos amigos, tanto os de longa data (Daniel, Igor, João Elias, etc) e os de curta data (Dennis, Jefferson, Valeska, Juninho, Jeremias) pelas alegrias compartilhadas no caminho.

Agradeço os professores, Ian, Laís, Ana, Pedro, Lary e Paçoca, por me ajudarem a aprender coisas realmente úteis.

Agradeço também ao Nilson, parceiro de sofrimentos, aos professores mais próximos, Bruno, Eduardo, Mintsu, e Téo, e aos colaboradores, Renam, Pedro, e Vanessa.

Por fim, agradeço à minha Jussara querida, que conheci no final dessa jornada.

Resumo

Este trabalho discute maneiras de integrar técnicas de Predição Intra, um conceito clássico de algoritmos para compressão de imagem, com os novos codecs que vem surgindo baseados no uso de Redes Neurais.

No Capitulo 1, é introduzido o problema, e é discutido como o texto vai ser estruturado. Primeiro apresentamos algumas justificativas para o estudo de novos codecs, e descrevemos como codecs baseados em redes neurais tem obtidos bom resultados, e que a ideia de predição intra tem sido pouco explorada por eles. Depois disso, fazemos um detalhamento dos objetivos específicos do texto, e por fim descrevemos a organização dos demais capítulos.

O segundo capítulo, que trata dos Fundamentos, busca cobrir a maioria dos conceitos usados no restante do texto. Para isso, ele é dividido em 4 sessões. Na preimeira sessão, são apresentados conceitos básicos de Teoria da Informação: Entropia, Distância de Kullback-Leibler, e Codificadores Aritméticos. A segunda sessão trata dos conceitos de compressão de imagem. Primeiro, são discutidas as etapas de um codec de imagens genérico: DPCM, Transformada, Quantização, Codificação e as respectivas eetapas inversas. Depois disso, é dado um detalhe maior ao funcionamento das técnicas de DPCM, divididas em Predição Intra e Predição Inter. Ainda na sessão de Compressão de Imagens, discutimos também a Optimização Taxa-Distorção (RDO), e como nem todos os codecs de fato fazem esse tipo de otimização. Por fim, discutimos as duas métricas que usamos para avaliar nossos codecs, PSNR e MS-SSIM.

A próxima sessão do capítulo de Fundamento trata de Redes Neurais Artificiais. Discutimos como elas são treinadas, as funções de ativação usadas, e as camadas que de Redes Neurais que usamos em nossos experimentos. Por fim, temos uma sessão sobre Autoencoders, que são um tipo específico de rede neural usado em problemas de compressão de imagens. Nessa sessão discutimos também as diferentes formas de quantizar as represetações que são obtidas dos Autoencoders, e que se tornam de fato nossa representação binária para as imagens.

O terceiro capítulo trata da Revisão da Literatura, e começa com uma sessão sobre Autoencoders usados para Inpainting. Inpainting é um problema clássico de processamento

de imagens, que geralmente surge durante edições de imagens, onde se busca preencher um pedaço de uma imagem que não esteja mais presente com algo que possua algum sentido. A ideia de usar Redes Neurais para fazer Inpainting surge com [Pathak et al., 2016], e é posteriormente aprimorada por [Iizuka et al., 2017] e [Yu et al., 2018]. Para o nosso trabalho, contudo, o maior foco é em [Minnen et al., 2017], que teve a ideia de usar redes para inpainting no lugar de métodos de predição intra tradicionais.

A segunda sessão do Capítulo 3 trata sobre Autoencoders usados para Compressão, e é dividida entre Autoencoders Recursivos e Iterativos e os baseados em Modelos Variacionais. Autoencoders Iterativos são descritos em [Toderici et al., 2016], que inicia a ideia de montar um codec com base em Redes Neurais com uma ideia simples de colocar várias camadas em sequência, buscando reconstruir na saída da rede a imagem de entrada, mas com uma camada binarizadora no meio. Como esses resultados não são bons, ele conclui que é melhor encadear várias dessas redes, o que seria o "codec iterativo". Para melhorar esse resultado, ao invés de encadear várias redes, ele resolve usar apenas uma única rede, mas que possui camadas recorrentes LSTM, que possuem memória. Esse é o chamado codec recursivo. Em seguida, detalhamos mais dois artigos, [Toderici et al., 2017] e [Johnston et al., 2018], que elaboram um pouco mais essas ideias, incorporando mais módulos para melhorar os resultados, e passando as imagens mais vezes pela rede ("priming") antes de obter o resultado da codificação. Por fim, discutimos novamente o artigo [Minnen et al., 2017], que usa uma rede de inpainting para realizar predição intra e detalhando que o codec principal usado é um codec recursivo. Este último artigo serve de base para o artigo [Jung et al., 2020], que detalha nossos experimentos com codecs recursivos.

A subsessão seguinte trata de Autoencoders para compressão baseados em Modelos Variacionais. Modelos Variacionais, como definidos por [Kingma and Welling, 2014], são modelos de Autoencoders em que se supõe que a distribuição dos dados que estão sendo analisados possui na verdade uma distribuição latente mais simples, e que pode ser aproximada por uma distribuição normal de média zero e variância unitária. Quando criamos um modelo variacional, queremos descobrir descobrir uma transformada da distribuição que estamos lidando para a distribuição normal, e uma transformada inversa que leve da distribuição normal de volta para a que estamos lidando. [Ballé et al., 2016] nota que existe uma analogia entre modelos variacionais e otimização taxa-distorção, e cria um modelo para compressão de imagens para fazer isso. Ele minimiza conjuntamente a entropia da distribuição latente, já criando um modelo para o codificador aritmético, e a distorção da imagem reconstruída. Um dos motivos para isso dar tão certo é o uso de camadas GDN, e suas respectivas inversas, IGDN. Camadas GDN tem o efeito de gaussianizar o sinal de entrada, e aproximá-lo de uma transformada unitária.

Seguindo nessa linha, [Ballé et al., 2018] adiciona uma segunda modelagem de hiperparametros sobre o código, pois ele observa que há ainda uma grande diferença espacial nos códigos obtidos, correlacionada com a imagem. Por fim, discutimos também [Theis et al., 2017], que tem ideias similares, mas cujo ponto mais interessante é o uso de uma multiplicação por escalares durante a quantização. Esse procedimento permite obter vários pontos na curva de RDO a partir de um único treinamento.

No Capítulo 4, que trata da Metodologia usada, buscamos detalhar os experimentos que fizemos. Primeiro, descrevemos dois modelos de Autoencoders para Inpainting que usamos. O primeiro modelo possui a mesma arquitetura que o modelo usado por [Minnen et al., 2017], e possui como entrada um patch de tamanho 64×64, em que a parte inferior direita é ocultada, e como saída esperada gerar essa mesma parte inferior direita que foi ocultada na entrada. O segundo modelo possui uma arquitetura similar, mas com a diferença que é adicionado um patch adicional no canto superior direito, e os patchs de entrada são rearranjados no formato $4 \times 32 \times 32$. Por causa dessa mudança, as primeiras camadas do modelo são ajustadas.

Na segunda sessão do Capítulo 4, descrevemos os experimentos feitos em [Jung et al., 2020], lidando com Autoencoders para Compressão Recursivos. Para analisar os efeitos da predição intra, usamos 3 modelos. O primeiro é baseado na arquitetura base de [Toderici et al., 2017], e não possui modelos de predição intra. O segundo modelo usa a mesma arquitetura de base, mas realiza predição intra usando o primeiro Autoencoder para Inpainting descrito na sessão anterior, e codifica apenas resíduos. O terceiro modelo realiza predições intra usando os dois Autoencoders para Inpainting descritos na sessão anterior, além de previsões intra do HEVC. Ele escolhe a melhor predição, e codifica ela. Devido à arquitetura desses modelos, devemos ressaltar que eles trabalham com patches de tamanho fixo 32×32.

Por fim, na terceira sessão, descrevemos os codecs baseados em modelos variacionais que usamos. Como no caso dos codecs recursivos, usamos três modelos. O primeiro deles não faz uso de predição intra, e possui uma arquitetura muita semelhante àquela usada por [Ballé et al., 2016], com um pequeno ajuste no tamanho dos filtros. O segundo modelo faz uso de predição intra, e possui na entrada o nosso primeiro modelo de Autoencoder para Inpainting. Uma limitação desse segundo modelo é que ele é limitado a usar um patch size de tamanho 32×32, por causa do modelo de predição intra usado. Buscando contornar essa limitação, nosso terceiro modelo também possui um modelo de predição intra, mas este é completamente convolucional, e treinado junto com o codec principal. Por causa disso, o terceiro modelo, assim como o primeiro, não possui limitações para o patch size usado.

No Capítulo 5, discutimos os resultados dos experimentos propostos no capítulo 4.

Primeiro, descrevemos como construímos nosso dataset de treinamento, e como é nosso dataset de testes. Em seguida, descrevemos os resultados do nosso Autoencoder para Inpainting, comparando sua performance sozinho contra os modos intra tradicionais do HEVC. Em seguida, descrevemos os resultados dos nossos codecs recursivos. Começamos primeiro fazendo uma análsie da performance do nosso codec base sem predição intra ao longo do treinamento, e chegamos à conclusão que podemos treinar esse tipo de modelo por cerca de 300,000 iterações, pois a partir desse ponto já começa a surgir uma saturação das capacidades do modelo. Em seguida, comparamos as curvas de PSNR e MS-SSIM para os 3 modelos recursivos. A conclusão que chegamos é que o modelo com apenas um modo intra possui resultados piores no geral, mas que o modelo com multiplos modos possui resultados razoáveis, especialmente se considerarmos as taxas mais baixas. Isso acontece porque no modelo de um único modo, caso a predição gerada seja ruim, o modelo acaba se esforçando para consertar ela. Para melhorar os resultados dos nossos codecs recursivos, resolvemos usar um algoritmo de alocação de bits, em que alocamos bits baseados em limiares de PSNR, mas com um parâmetro adicional que decide se vale a pena continuar gastando bits com um dado patch da imagem. Usando esse algoritmo de alocação, temos uma melhora expressiva dos resultados dos codecs recursivos. Na sessão seguinte, analiamos os resultados dos nossos codecs variacionais. Primeiro, fazemos um teste de duração do treinamento, e observamos que começa a haver uma saturação do codec em cerca de um milhão de iterações. Com esse resultado, fazemos uma segunda análise, comparando a relação do patch size de treinamento com o patch size usado na fase de testes. Essa análise nos diz que, exceto por patch sizes muito pequenos, não há diferença significaiva em usar patch sizes diferentes durante treinamento e teste.

Um fator inusitado, que temos que ressaltar, é que os pontos indexados com lambda mais baixo, 0.0001, para os codecs variacionais com predição intra que usa um modelo pré-treinado, e aquele com a predição intra treinada junto com o modelo, mas usando patches de tamanho 64, possuem valores muito piores que o esperado. Seria esperado que eles tivessem a menor bpp, do modelo, mas na verdade a bpp deles é bastante alta. Nossa hipótese para esse resultado é que nessas taxas tão baixas, onde em muitos casos nem mesmo a cor é codificada, o modelo de predição intra, que é treinado com imagens naturais, não consegue funcionar mais, e por isso acontece um colapso do modelo, resultando em taxas elevadas. Esses pontos, onde o codec não funciona, são ignorados nos demais resultados, já que não são úteis na prática.

Comparando os resultados dos codecs, propriamente ditos, o que notamos é que o codec que usa um modelo para predição intra pré-treinado tem resultados muito piores que o codec variacional sem predição intra. Já o modelo em que a predição intra é treinada junto com o codec possui resultados semelhantes ao codec sem predição intra, mas com

resultados um pouco melhores para baixas taxas.

Por fim, comparamos os melhores resultados dos nossos codecs recursivos e variacionais entre si, e contra os codecs JPEG e JPEG2000. O que podemos ver é que os codecs recursivos, usando o algoritmo de alocação de bits, possuem resultados melhores que os do JPEG tradicional, porém inferiores aos do JPEG2000. Os codecs variacionais, contudo, conseguem obter resultados melhores do que o JPEG 2000. Em baixas taxas, particularmente, o codec variacional com o modelo treinado em conjunto obtém bons resultados.

O último capítulo trata das conclusões e possíveis continuações do trabalho. Com base nos nossos resultados, vemos que existem evidências de que o uso de predição intra pode melhorar o resultado de codecs de imagens usando redes neurais, principalmente para taxas baixas. Esses resultados, contudo precisam ser mais bem explorados. Na sessão de trabalhos futuros, sugerimos modificar os nossos codecs baseados em modelos variacionais para introduzir a modelagem por hiperprior usada por [Ballé et al., 2018], ou possivelmente extender ainda mais usando um modelo Autoregressivo sobre os códigos. Outro ponto que seria interessante seria fazer um treinamento por etapas para lidar com o problema de *moving target* dos codecs de predição intra, já que eles são treinados com imagens naturais, mas durante a fase de testes são usados com imagens distorcidas. Isso inclusive poderia resolver os pontos do codec em que ele para de funcionar. Da parte dos Codecs Recursivos, o maior problema a ser resolvido é deixar o codec mais rápido, já que atualmente a predição intra deve ser feita sequencialmente. Uma possibilidade seria ignorar efeitos de drifting, e gerar as predições intra usando a imagem original. Um outro ponto que pode melhorar esses codecs seria adaptar sua arquitetura para uma completamente convolucional, e que permita usar patches de tamanho variável.

Palavras-chave: Compressão, Redes Neurais, IA, Imagens

Abstract

This work discusses if using intra prediction can improve the results of codecs that are based on Neural Networks (also called learned image codecs. It begins with a review of the basic principles of both image compression and neural networks, which is followed with a review of the state-of-the-art neural networks used for compression, which are called autoencoders. In Chapter 4, we describe our methodology, describing the models we tested, with or without intra prediction. In Chapter 5, we detail the results of our experiments, and in Chapter 6 we present our conclusions that intra prediction can improve the results of learned image codecs, especially at lower rates, but that these results need to be further studied.

Keywords: Compression, Neural Networks, AI, Images

List of Symbols

x	a scalar value.
X	a monochromatic image, or a 2D matrix.
$\mathbf{X}$	a tensor, representing 3D or N-D matrices.
$f(x)$	a function f that takes scalar values as input.
$p_X(x)$	a probability density function p_X associated with a random variable X.
$X \sim p_X(x)$	the $\sim$ symbol indicates that the random variable X is sampled from $p_X(x)$.
$\mathcal{N}(\mu, \sigma)$	a Gaussian distribution with mean μ and variance σ.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As Internet use becomes more and more popular, so does the transmission of image and videos. The popularization of smart phones has vastly increased the amount of photos and videos being produced, stored and transmitted, and companies like Netflix, Youtube and Twitch (to name a few), work by transmitting copious amounts of video, either pre-recorded or via live streaming.

The Cisco Annual Internet Report [Cisco, 2020] predicts that by 2023 there will be an average yearly growth of 27% connection speed for mobile users, and 25% of Wi-Fi connection speed. This increase in speed is paired with an increase in consumption. For instance, the number of 4k TVs is increasing on average 27% every year, which is correlated with a demand for 4k video.

Therefore, there is an urgent need to improve upon compression techniques to reduce storage and transmission costs. One possible solution is the use of neural networks, which have revolutionized the fields of Computer Vision and Natural Language Processing, and therefore seems a natural candidate to improve image and video compression.

1.1 Main Objective

The main objective of this work is to see if the use of Intra predictions can improve learned image codecs, that is, codecs based on the use of Neural Network. Intra prediction techniques are vastly used by traditional video codecs, such as H.264 [Wiegand et al., 2003] and HEVC [Sullivan et al., 2012], and the increase in complexity of intra prediction from H.264 to HEVC shows that it makes a difference in these codecs.

We are not yet sure if Neural Networks will fully replace traditional codecs, as those are generally developed with built-in brute force strategies to find the best solution, yet we feel that their performance can be improved by relying on these techniques.

1.2 Specific Objectives

Our specific goals in this work are to compare the performance of several different codecs that use neural networks. Specifically:

- Train and evaluate Neural Networks that generate intra productions.
- Train recursive codecs, based on the ideas started by [Toderici et al., 2016], comparing the use or not of Intra prediction models.
- Train variational codecs, that follow the line started by [Ballé et al., 2016], comparing the use or not of Intra prediction models.

1.3 Text Organization

In this text, we aim to discuss some aspects of how compression and neural networks work together. In the second chapter, we discuss the main ideas behind compression and neural networks. On the third chapter, we discuss some of the recent literature in the area we have judged relevant to this work.

In Chapter Four, we detail the methodology we use to test if intra prediction can improve neural networks. In Chapter Five, we present the results of our experiments, and then in the sixth chapter we sum up our conclusions, as well as suggest some future improvements to our work.

Chapter 2

Fundaments

In this chapter we introduce the main concepts of the two large areas involved in this work: image and video compression, and neural networks. Our goal is to introduce a few of the terms and concepts used through all the text, in varying degrees of detail.

We begin with a few concepts from information theory, and then we give a panorama of techniques used by traditional image and video codecs, with a focus on intra prediction techniques, and how we evaluate the performance of such codecs.

We then follow up with the basic concepts of neural networks, describing how they are trained and the layers most commonly used. Afterwards, we present a brief description of Autoencoders and how they relate to our work.

2.1 Information Theory

In this section we define some of the concepts from information theory that we use in our text.

2.1.1 Entropy

Entropy is a measure of uncertainty of a random variable, or a sequence of random variables [Cover, 1999]. For a random variable X sampled from a distribution $p_X(x) = \Pr(X = x)$, we define the entropy of X, $H(X)$, as:

$$H(X) = -\sum_x p_X(x) \log_2 p_X(x) \tag{2.1}$$

The entropy of a random variable is also the expected length in bits of an optimal code for that random variable [Cover, 1999].

2.1.2 Kullback-Leibler distance

The Kullback-Leibler distance (also referred as KL distance) measures the similarity between two probability distributions, $p_X(\cdot)$ and $q_X(\cdot)$. It is defined as:

$$\mathrm{KL}(p_X, q_X) = \sum_x p_X(x) \log \frac{p_X(x)}{q_X(x)} \tag{2.2}$$

If we attempt to construct an optimal code for a random variable X using a distribution q_X when it actually follows the distribution p_X, we will be able to construct a code using $\mathrm{H}(X)+\mathrm{KL}(p_X,q_X)$ bits.

In our case, we interpret q_X as our model for the distribution of the data, and p_X as the real distribution. Therefore, the smaller the KL distance, the closer our model of the data is to the real data, and the better the codes generated.

2.1.3 Arithmetic Coder and Range Coder

One way to make an optimal code, or at least close to it, is to use an arithmetic code [Rissanen and Langdon, 1979]. Supposing that a sequence of symbols is sampled from some probability distribution, then any sequence of symbols has a unique probability associated with it. An arithmetic encoder, instead of encoding the sequence of symbols, encodes a range of values that contains this probability. For a finite precision, this probability range is uniquely decodable, and therefore can transmit a given message.

For the first symbol, the arithmetic coder encodes keeps the bottom and upper limits of the cumulative distribution function of the symbol. For the second symbol, it adjusts these two values based on the c.d.f of the second symbol, and keeps these values. It keeps doing this for every symbol in the sequence. When the two values start reaching precision limits for floating points, the encoder re-scales the values by multiplying them by a constant. When there are no more possible sequences for a given range, the code can be sent.

One variation of Arithmetic Coders is the Range Coder [Martín, 1979], that is mostly equivalent, but instead of using floating point probabilities, uses integer ranges. Performance differences between Arithmetic and Range coders are more related to implementation details than to the algorithms themselves.

2.2 Image Compression

In this section we describe the principles of image compression. We describe a generic image codec, followed by a more detailed description of DPCM techniques, and then the

metrics used to evaluate the performance of our codecs.

2.2.1 Traditional Image Compression

Image and video compression is a computer problem with the goal of obtaining as small as possible representation for a given image. It can be divided in *lossless* or *lossy* compression. Lossless compression means that all the information in the original image is preserved when the image is compressed, while in lossy compression some of the information is discarded. By choosing which information to keep, and which to discard, lossy compression makes trade-offs between generating smaller files and introducing distortions in the images. This work focuses on lossy image compression.

In most cases, compression is done by removing redundancies in the data. For images and video, they are generally divided into Spatial, Temporal, and Statistical redundancy. Spatial Redundancy means that neighboring pixels in an image are similar to one another (that is, pixels representing the same object generally have similar values). Temporal redundancy means that frames in a video are very similar to one another (most objects in a given frame are probably present in the previous and next frames). These two redundancies are dealt with by using DPCM techniques: Intra and Inter prediction. Statistical redundancy, which is a bit different, deals directly with the data ready to be encoded, and with a modelling of the raw data, and is handled by Entropy Coders.

In traditional codecs like JPEG [Wallace, 1992], images are encoded using a block based, raster scheme encoding, which means that each image is first split into several blocks, which are them processed in a left to right, up to down order, as illustrated by figure 2.2. Video codecs such as H.264 [Wiegand et al., 2003] and HEVC [Sullivan et al., 2012] also work by splitting larger blocks into smaller blocks, also using this same order. There is also the possibility of processing the blocks in more elaborated orderings, but those are seldom used.

To compress each of the image blocks, as schematized on Figure 2.1, we first do a residual encoding via DPCM, where we estimate the values of the pixels inside the block based on the previously encoded blocks, and them encode only the difference between the predicted block and the original one, which is called the residual. Generally, residuals have smaller entropy than images, and therefore they can be compressed at a lower rate [Sayood, 2017].

Next, we pass the block through some transform, such as the Discrete Cosines Transform (DCT) [Ahmed et al., 1974], or in the case of the JPEG 2000
[Rabbani and Joshi, 2002], a Wavelet Transform [Daubechies, 1990], that map the image values into other domains. These transforms are chosen for image compression because they provide some ordering to the image values that can be used during quantization.

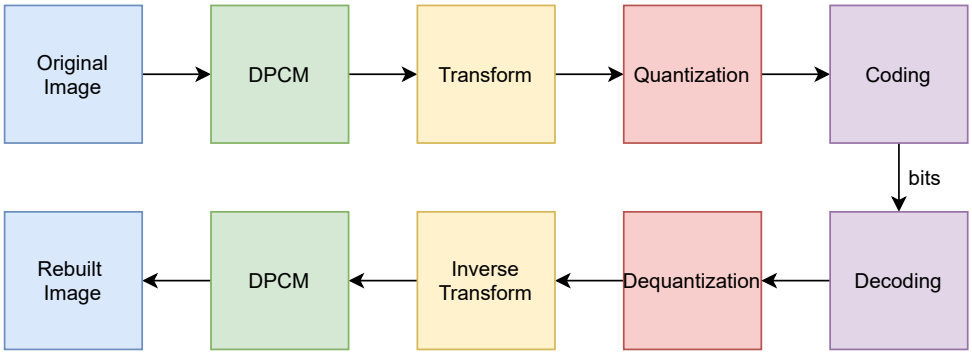

Figure 2.1: Block Diagram of a generic Image Codec

Defining which of the components can be ignored is defined by the next step in the process, quantization. For JPEG, quantization is implemented as a element-wise division of the outputs of the transform by some table, followed by the floor rounding of the values. Increasing quantization means that the numbers on the quantization table will be bigger, and therefore more numbers will be rounded to zero after division by the quantization parameters. This step is where the *lossy* compression occurs, as numbers rounded down to zero can't be recovered during decompression, and is non-reversible. Other codecs have more elaborate algorithms, but work on the same principle.

Finally, these numbers are encoded using some form of entropy coding. JPEG uses Huffman Coding, H.264 uses either CAVLC or CABAC [Marpe et al., 2003], and HEVC uses CABAC exclusively. These encoding schemes use a model for how the data they are compressing is distributed, which can be a pre-computed (and defined as a part of the codec standard) or computed for each individual image, either after an initial pass on the data (in which case the encoder must send the model to the decoder) or calculated while the data is encoded/decoded (in which case both encoder and decoder construct the same model in parallel). Generally, a better modeling of the data improves compression rates, but there are trade-offs in constructing and sending the model, and the complexity of the calculations. The binary output of the entropy coder is then stored in a file or transmitted.

Decompressing follows the reverse process: we first decode the binary streams, and them multiply them using the same quantization table used during compression, and them use the inverse transform, and use the already decoded blocks to generate the prediction to add to the residue and obtain the decompressed image.

2.2.2 DPCM

Differential Pulse Coded Modulation is a signal transmission technique that sends only the difference between the current signal and the previous one, saving on transmission costs, as

on average residual signals have smaller entropy than the original signal [Sayood, 2017]. Nowadays, the abbreviation DPCM is used as a portmanteau term for many different coding techniques based on encoding a signal using the information of previously encoded signals.

On video compression, DPCM appears mainly in two situations: intra frame prediction, and inter frame prediction. Intra frame prediction treats each frame as an independent image, and uses the information on already decoded parts of the image to predict values of the remaining image, and modern image codecs use some of its techniques. Inter frame prediction, on the other hand, uses multiple video frames, and can only be used on video contexts. Inter frame essentially deals with encoding the movement from one video frame to another, and warping a frame into another.

Intra Frame Prediction

Intra frame prediction relates to compressing a video frame as a stand-alone frame, or as an independent image. It uses information of the frame that is already available to the decoder to generate predictions of the remainder of the image.

For H.264 and HEVC intra frame prediction, blocks already decoded (in raster scan order, see figure 2.2) are used to estimate the values of the next block, using the idea that adjacent blocks are more likely to be similar. Using a function of the pixels in the neighboring blocks, a mode of prediction is generated. The encoder tests all available modes and chooses the best prediction, taking into account not only the magnitude of the residuals, but also the estimated number of bits needed to transmit that mode.

As an example, HEVC uses the line of pixels above and to the left of the block as the context for the prediction. On figure 2.3 we illustrate the context pixels, along with some edge cases where there aren't enough pixels to form a full context and some pixels are extrapolated as mirrored versions. The first mode to be predicted is the DC mode, where the predicted block is filled with the average value of the context pixels. There are two other simple modes, Vertical and Horizontal. On Vertical mode, the pixels from the above context line are repeated on all lines of the prediction, and on Horizontal mode the pixels on the left column are repeated on all columns of the prediction. Some of these modes are shown on figure 2.4.

The Horizontal and Vertical modes are two of the 33 Angular modes of the HEVC standard. The other modes are similar, but use diagonal lines and interpolation of the pixel values.

Finally, there is the Planar mode, which in HEVC is an average of the Horizontal and Vertical modes, but on H.264 is a more elaborated gradient of the context pixels.

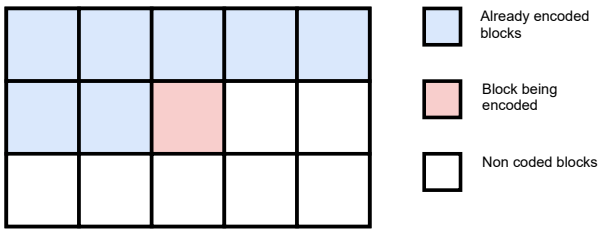

Figure 2.2: Rastering Scheme

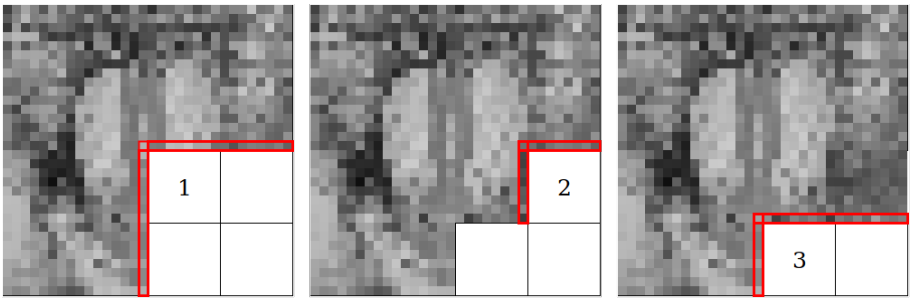

Figure 2.3: Examples of different contexts (in red) for 3 blocks being processed sequentially

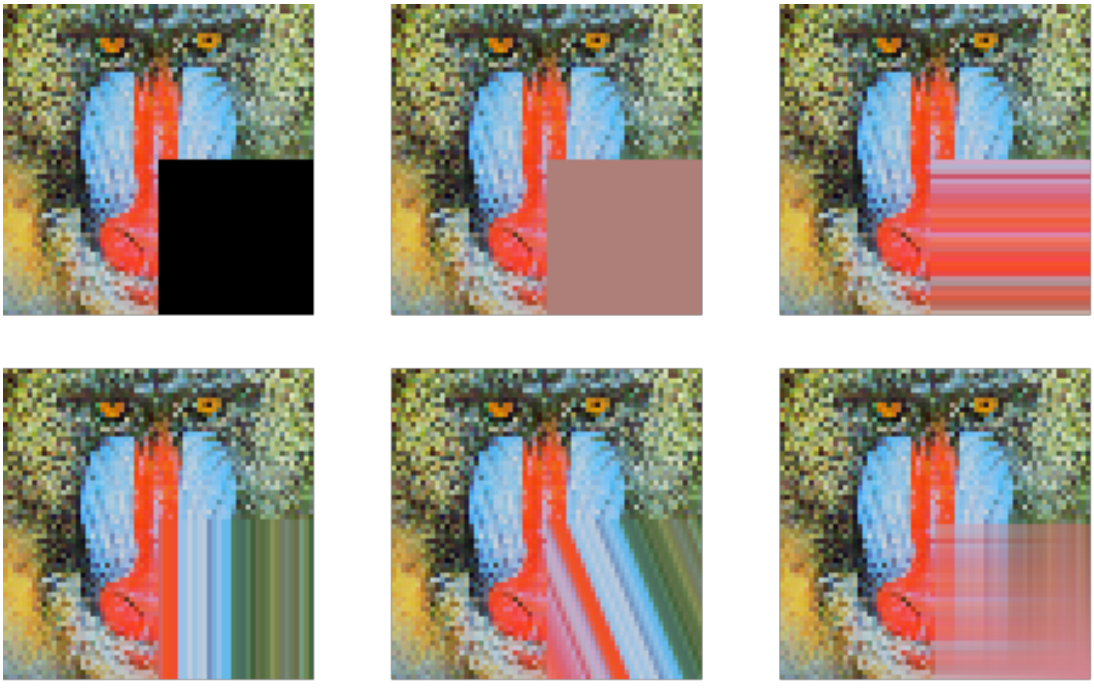

Figure 2.4: HEVC Intra Prediction examples - from left to right, top to bottom: block showing what should be predicted, DC prediction, horizontal prediction, vertical prediction, angular prediction, and planar prediction.

Inter Frame Prediction

Another kind of video prediction is Inter frame prediction, that uses the other video frames for compressing a frame. Generally, it is classified into P type and B type. P frames are compressed using only previous frames, while B type uses both frames before and after in the sequence. This is possible because Intra frames can be decoded independently, and therefore a future Intra frame can be made available during the encoding/decoding process.

To encode an Inter frame, the video is split into blocks, and the encoder finds the most similar block in a list of other frames. Then, the apparent displacement of the block, called motion vector, is calculated. Both coordinates of the most similar block and its motion vector are encoded in the bitstream. The mapping of best blocks using the motion vectors is called warping, and the predicted frame is sometimes called warped frame, or motion predicted frame. In some cases, the residual between the motion predicted frame and the reference frame is also encoded, if it will improve the results.

2.2.3 Rate-Distortion Optimization - RDO

Rate-Distortion Optimization, commonly abbreviated as RDO, is a fundamental concept in image and video compression. Essentially, it synthesizes a trade-off between rate and distortion, that is, as decreasing the number of bits used to compress an image generally causes a decrease in image quality (or an increase in distortion). On the other hand, allowing the encoder a larger bit budget generally produces images of better quality.

Looking at Equation 2.3 below, we see that for a given rate $R(X)$ to encode the image X, and the distortion $D(X, Y)$ between the distorted image Y and the original image X, we can use a parameter λ to index this trade-off. We define a value J as the sum between these two terms.

$$J = R(X) + \lambda \cdot D(X, Y) \tag{2.3}$$

Each codec has different behaviours towards these equations. JPEG, for example, defines QP, a number varying from zero to 100, as an approximation for this value of λ. In theory, setting a higher QP will generate images with higher rates and smaller distortions. However, in practice there are many QP that produce sub-optimal solutions. This is illustrated on Figure 2.5. More refined codecs may check whether or not the points are placed in the convex hull of the RDO curve, and allow only these points to be generated.

Another example is the choice of which arithmetic coder is used in H.264. CABAC and CAVLC both are losslessly encoding the data they receive, but at higher rates for

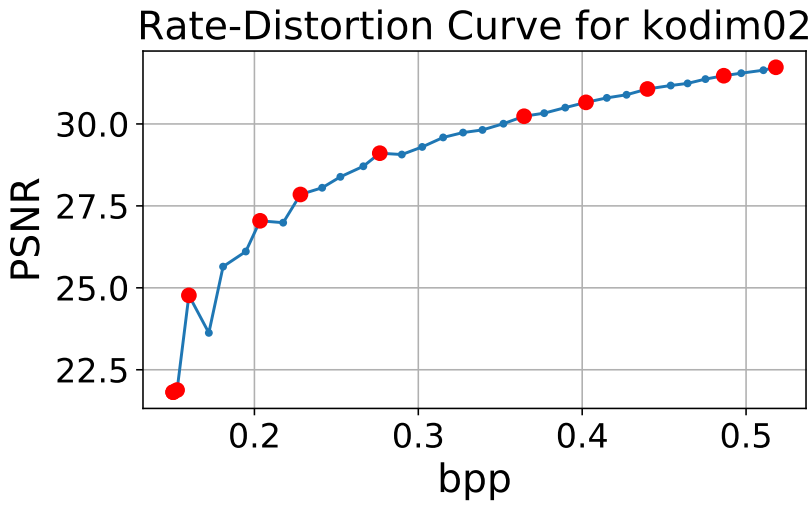

Figure 2.5: RD curve for a JPEG encoded image. The red points belong to the convex hull, and are optimal points for the codec. The remaining points are sub-optimal, and would not be chosen by a codec that enforces RDO.

the CAVLC than CABAC. Here, using CABAC will lower the value of J, which would be better, but actually the trade-off is being replaced by time and complexity, meaning that the curve can't be the only reference in practical situations.

For some of the learned image codecs we will see later, the common practice is to train the codec by first defining the lambda parameter, and optimize simultaneously both the rate and the distortion, but keeping this relationship between them.

2.2.4 Distortion Metrics

On this section we discuss the main metrics used to evaluate performance of codecs, PSNR and MS-SIM. These metrics are called objective metrics, as they are mathematical functions, instead of being based on the opinion of humans, such as the result obtained by a Mean Opinion Score - MOS [ITU, 1996] Generally, the performance of a codec is evaluated by compressing an image at different rates, and comparing the distortion of the images at each rate with the original (reference) image.

By plotting the rate and distortion, we can have a rough judgement on whether one codec is better than another, as points higher and to the left indicate better performance.

PSNR

The simplest and most common metric to evaluate the distortion of an image after being compressed is the Peak Signal-Noise Ratio, or PSNR. Essentially, the PSNR is the

logarithm of the Mean Squared Error calculated between the reference image and the compressed version, adjusted by the level of quantization used.

We can calculate the Mean Squared Error for monocromatic images x and y using the following equation:

$$\mathrm{MSE}(X,Y) = \frac{1}{mn}\sum_{i=0}^{m}\sum_{j=0}^{n}\Big(X(i,j) - Y(i,j)\Big)^2 \tag{2.4}$$

With the MSE, we can calculate the PSNR using the following formula, where MAX_I represents the maximum value a pixel can assume (255 for an 8 bit image, 1023 for an 10 bit one).

$$\mathrm{PSNR} = 10 \cdot \log_{10}\left(\frac{\mathrm{MAX}_I^2}{\mathrm{MSE}}\right) \tag{2.5}$$

$$= 20 \cdot \log_{10}\mathrm{MAX}_I - 10 \cdot \log_{10}\mathrm{MSE} \tag{2.6}$$

SSIM and MS-SSIM

SSIM[Wang et al., 2004] is a metric calculated over the *luma* channel of an image pair that aims to measure distortion in a manner more similar to human observers than PSNR. It is a product of three factors: luminance, contrast and structure.

For the reference and compressed images, we extract patches x and y, and calculate the so called luminance $l(X,Y)$, contrast $c(X,Y)$ and structure $s(X,Y)$ using the following formulas:

$$l(X,Y) = \frac{2\mu_x\mu_y + c_1}{\mu_x^2 + \mu_y^2 + c_1} \tag{2.7}$$

$$c(X,Y) = \frac{2\sigma_x\sigma_y + c_2}{\sigma_x^2 + \sigma_y^2 + c_2} \tag{2.8}$$

$$s(X,Y) = \frac{\sigma_{xy} + \frac{c_2}{2}}{\sigma_x\sigma_y + \frac{c_2}{2}} \tag{2.9}$$

where $c_1 = (0.01 \cdot L)^2$ and $c_2 = (0.03 \cdot L)^2$, (L is the dynamic range of the image), μ_x is the mean value of image x, σ_x the variance of x, and σ_{xy} the covariance between images x and y:

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{2.10}$$

$$\sigma_x = \left(\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)^2\right)^{\frac{1}{2}} \tag{2.11}$$

$$\sigma_{xy} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y) \tag{2.12}$$

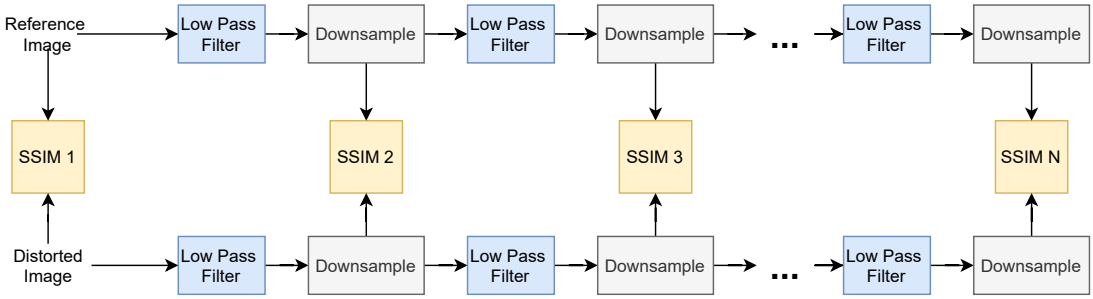

Figure 2.6: Multi-Scale SSIM

Multiplying each of the terms, we have:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{2.13}$$

Generally, this metric is calculated for every pixel in the image, using a window of size 11, and them the results are averaged (one possible use is also to view a map of the SSIM results).

One improvement on the SSIM results is the use of MS-SSIM[Wang et al., 2003], or Multi-Scale Structural Similarity Index. The main idea is that codecs produce different forms of distortions in different scales, so we should measure SSIM at different scales to obtain a better result.

To do this, SSIM is calculated for the image pair, which passes through a low pass filter and is downsampled. This new downsampled version of the image pair is a new scale at which SSIM is again calculated, and so on for a given number of scales. This is schematized at figure 2.6.

The product of the SSIM at each scale is then calculated as the MS-SSIM.

2.3 Neural Networks

On this section we describe the fundamental concepts of Neural Networks. We first describe how a Neural Network is trained, detail some of the activation functions and layers used in the models we discuss, and explain what are Autoencoders, that are a specific kind of Neural Network we use, along with some problems related to quantization and neural networks.

2.3.1 Definition and Training Process

Neural Networks[1] are a set of chained functions ($f(g(h(\dots))))$ that are used to approximate an arbitrary function f^* [Goodfellow et al., 2016]. They are so named because they were originally inspired by how a brain works, with each neuron receiving several inputs from other neurons, and then triggering a response, that is forwarded to other neurons. Traditionally, Neural Networks are made by several simple functions, sometimes called neurons, that are arranged into layers. Each neuron receives as input the outputs of the neurons in the previous layer, processes them, and forward its own outputs to the next layer, subsequently until the final layer, which produces the output of the Neural Network.

Each layer, or, more precisely, each neuron in each layer, has a number of adjustable parameters, also called weights. Because of this, it is also common to represent Neural Networks as parametric functions $f(\theta, \cdot)$, where θ represents all the weights in all the layers. To adjust these weights, we train the model using a set of (x, y) pairs, called the training set, where $y = f^*(x)$, i.e., the mapping of the function we wish the Neural Network to learn. For each input y, the Neural Network outputs a value $\hat{y} = f(\theta, x)$, called an inference or prediction, and the process of generating the output and the intermediate values is called the *forward pass*.

The difference between the correct values and the predictions made by the Neural Network is called Error or Loss[2], and is evaluated by some metric. For example, on pixel synthesis problems, the metric can be the Mean Squared Error (MSE) between each of the pixels, while in classification problems the network commonly outputs a probability assigning the input to each of the classes, with the loss function being the KL divergence between this prediction and the correct class.

If the metric used to measure the loss is differentiable, we can then calculate the gradient of the Loss function in relation to the weights, and then adjust the weights on the direction of this gradient. Calculating the correct derivative of each weight in relation to the difference between the correct output and the prediction is called *back-propagation*, and adjusting the weights based on the gradient is called *Gradient Descent*. The following equation indicates how the weights θ of a network trained with a loss function L are adjusted. Note that gradients are modulated by a parameter α, called the learning rate, so that we have more control over the weight adjustment.

$$\theta \leftarrow \theta - \alpha \nabla \mathrm{L}\Big(f^*(x), f(\theta, x)\Big) \tag{2.14}$$

[1] we also implicitly refer to Neural Networks as models through all the text

[2] here we use these term as the same, but some Machine Learning publications ascribe specific meanings to them

2.3.2 Activation Functions

One of the most important parts of a Neural Network is the activation functions it uses. These functions introduce non-linearities in the models, and allow the Neural Networks to learn non-linear functions. Sometimes they are classified as independent layers, and sometimes as part of the previous layer.

We begin with the Hyperbolic Tangent activation function, defined in equation 2.15. As one can see by the plot 2.7, it has an S shape, and because of that it belongs to a family of functions called Sigmoids. For large input values this function saturates at 1, and for large negative numbers it saturates at -1. Near zero, the function behaves very much like a linear function.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.15}$$

$$\frac{d}{dx}\tanh(x) = 1 - \tanh^2(x) \tag{2.16}$$

A special case of the hyperbolic tangent is the logistic function, defined on equation 2.17 below. The logistic function outputs values on the [0,1] range, and it also has a smaller saturation region, as we can see in figure 2.7.

$$\text{logistic}(x) = \frac{1}{2} + \tanh\left(\frac{x}{2}\right) \tag{2.17}$$

Commonly, the logistic function is represented by the letter σ, and is calculated using the following equation:

$$\text{logistic}(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.18}$$

One remarkable property is that the derivative of the logistic function can be defined in terms of itself, simplifying the backpropagation step:

$$\frac{d}{dx}\sigma(x) = \sigma(x)(1 - \sigma(x)) \tag{2.19}$$

Another common activation function is the Rectified Linear Unit, or ReLU, defined as $\text{ReLU}(x) = \max(0, x)$. That is, for values larger than zero, the ReLU is the identity, and for values smaller than zero it outputs zero. To calculate its derivative, we use an analytical extension for the discontinuity at zero:

$$\text{ReLU}(x) = \begin{cases} x & \text{for } x \geq 0, \\ 0, & \text{for } x < 0 \end{cases} \tag{2.20}$$

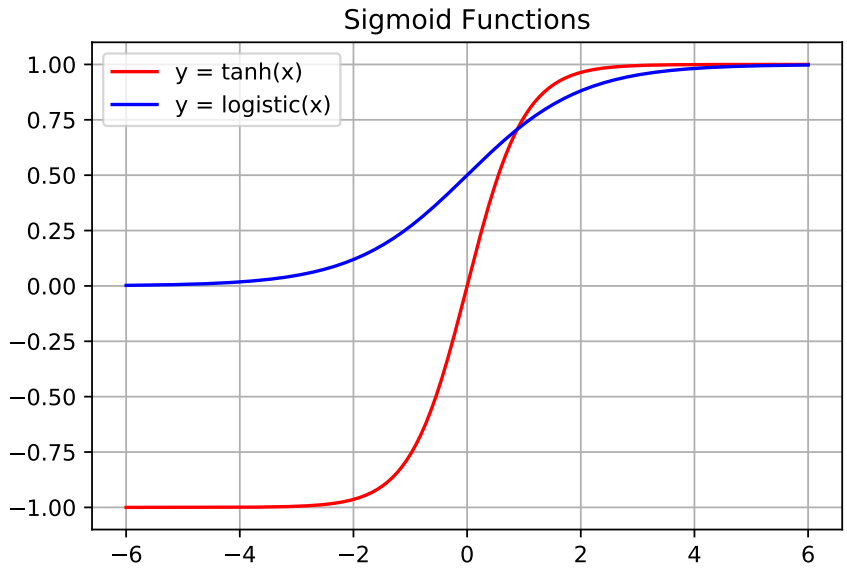

Figure 2.7: Sigmoid functions in the [-6, 6] range. In red, we have the Hyperbolic Tangent function, and in blue the Logistic function. As we can see, the Hyperbolic Tangent saturates faster than the Logistic function.

$$\frac{d}{dx}\text{ReLU}(x) = \begin{cases} 1 & \text{for } x > 0, \\ 0, & \text{for } x \leq 0 \end{cases} \tag{2.21}$$

As ReLUs are computationally simpler than sigmoids, both during the forward pass and while calculating the gradients, their use allows faster training of neural networks [Krizhevsky et al., 2012], and in many cases obtaining better results.

There are also some different variations of ReLUs. Leaky ReLUs, defined on equation 2.22, multiply negative values by some parameter α, that can be either fixed or learned. Their advantage over ReLUs is that they always have non-zero gradients, which speeds up training, and avoid some specific situations where a ReLU neuron stops learning and always outputs zero, being called "dead".

$$\text{leakyReLU}(\alpha, x) = \begin{cases} x & \text{for } x \geq 0, \\ \alpha \cdot x, & \text{for } x < 0 \end{cases} \tag{2.22}$$

Another variation of notice are ELUs, or Exponential Linear Units [Clevert et al., 2016], that are calculated using equations 2.23 and 2.24. They have good results, but they are more computationally expensive as well.

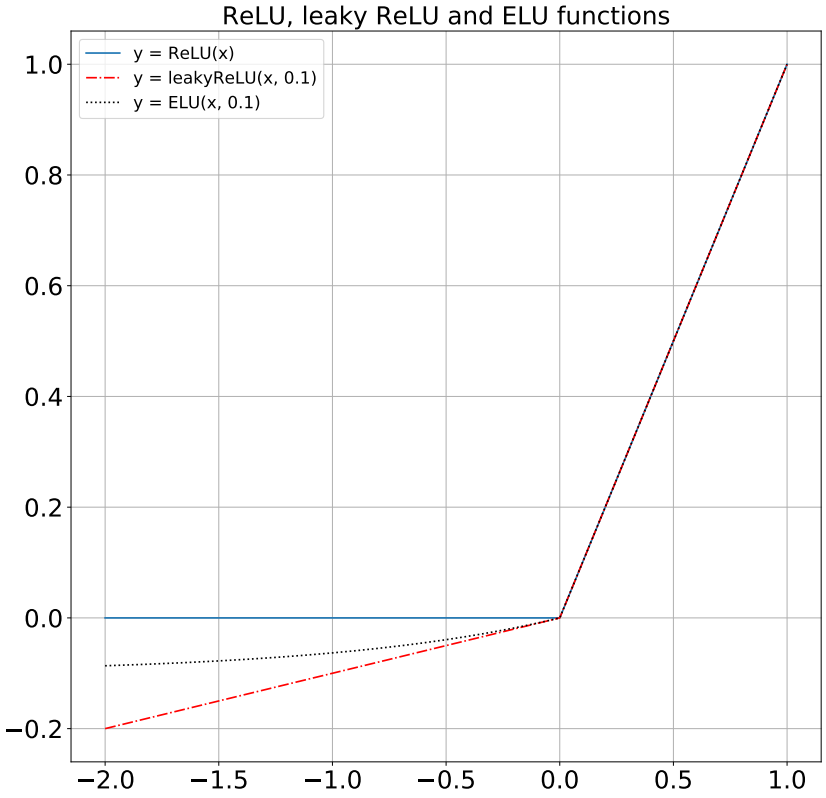

Figure 2.8: Comparison between ReLU, ELU and leaky ReLU. Both ELU and ReLU are shown with their alpha parameter set to 0.1.

$$\mathrm{ELU}(\alpha, x) = \begin{cases} x & \text{for } x \geq 0, \\ \alpha(e^x - 1), & \text{for } x < 0 \end{cases} \tag{2.23}$$

$$\frac{d}{dx}\mathrm{ELU}(\alpha, x) = \begin{cases} 1 & \text{for } x > 0, \\ \mathrm{ELU}(x) + \alpha, & \text{for } x \leq 0 \end{cases} \tag{2.24}$$

2.3.3 Dense Layers

The most traditional kind of layer in Neural Networks are the so called Dense layers, also known as Fully Connected layers. This kind of layer receives as input a one dimensional vector $\mathbf{x}$, and outputs a single scalar value y. This is done by calculating the dot product

of the layer weights $\mathbf{w}$ and adding a bias term b:

$$y = \mathbf{w} \cdot \mathbf{x} + b = \sum_i w_i x_i + b \tag{2.25}$$

In situations where we need to calculate multiple outputs for the same input, we perform a matrix multiplication:

$$\mathbf{y} = \mathbf{W} \cdot \mathbf{x} + \mathbf{b} \tag{2.26}$$

In traditional machine learning texts, a single Dense layer followed by an activation function is called a Perceptron, or Single Layer Perceptron, and models using several Perceptrons in sequence is called a Multi Layer Perceptron.

Finally, we must note that if the input is an image of dimension $m \times n$, we must first flatten the image into a vector of shape $mn \times 1$. Therefore, Dense layers end up having $\mathcal{O}(n^2)$ complexity and are expensive for image processing.

2.3.4 Convolutional Layers

Convolutional layers are Neural Network layers that perform convolutions on the inputs received from the other layers in the model. We must note that in many papers and implementations names like cross-correlation, inverse convolution, deconvolution and transposed convolution are mixed up. Neural Networks that use Convolutional layers are sometimes called Convolutional Neural Networks, and abbreviated as CNNs.

Convolutions by definition excel at finding and matching repetitive patterns, which are used in filtering operations, as long as the pattern is proportional to the kernel size. When dealing with images, we use 2D convolutions, that are better at extracting useful information from the images than Dense layers. On classification problems, the first layers of a CNN learn to detect lines, borders and color changes, and posterior layers learn to combine these primary objects into circles, edges and more complex shapes, and later into objects such as windows and eyes.

Each convolutional layer has K weights of shape $\kappa_1 \times \kappa_2 \times F$, where F is the number of feature maps of the previous layer, K is the number of outputs of the layer, and $\kappa_1 \times \kappa_2$ is the kernel size of the convolution operation. Their advantage in relation to Dense layers is that for images of size $n \times n$, the convolutional layer will require only $O(\kappa_1 \cdot \kappa_2 \cdot n)$ (with $\kappa_1 \cdot \kappa_2$ being the kernel size) operations, instead of $O(n^2)$ for Dense layers. As n is generally in the hundreds and thousands of pixels, and κ_1 and κ_2 are generally smaller than 10, we can see that this is an enormous reduction in complexity.

Padding and Strides

When working with convolutions, the resulting image will have a smaller dimension than the input image, unless we pad the image. Padding means that the image will be given additional pixels at the border, and depending on the situation these pixels are commonly defined as either a repetition of the last pixel, a mirror of the border pixels, or zeroes.

Padding can also be divided as *valid* or *same* (some authors prefer to define the convolution itself with these names, with an implicit padding). For valid padding, the convolution is only calculated on the points where there is a total overlap between the image and the convolutional kernel (that is, there is not in fact any padding being performed), and on same padding the number of pixels is calculated to preserve the image.

For example, applying valid padding to an image of size $W_1 \times H_1 \times C_1$ with a convolutional filter of size $K \times L$ will produce an image of size $W_2 \times H_2$, where H_2 and W_2 are calculated s follows.

$$W_2 = W_1 - K + 1 \tag{2.27}$$

$$H_2 = H_1 - L + 1 \tag{2.28}$$

If we wanted to preserve the image size, we would apply a padding of $K + 1$ to the image.

Another way to control the size of the feature maps is by defining the stride of the convolution. Normally, we calculate the convolution on every pixel in the image, but we can also do it instead at every "stride" pixels. This has the effect of dividing the output size by the stride we used. For example, if the output size of a convolutional layer would be of 128×128 using the default stride of 1, by using a stride of 2 we would get an output of size 64×64.

One point to mention is the usefulness of reducing the image size. Heuristically, CNNs perform better at a smaller resolution with more feature maps than at a larger resolution with fewer feature maps. As we in general are constrained by RAM and GPU, reducing the size of an image and allocating most convolutions to smaller resolutions give better results.

2.3.5 Depthwise, Pointwise and Separable Convolution

Another kind of convolution we use is the Separable Convolution. It is so called because it resembles matrix decomposition into separable filters, but here the main idea is to replace regular 2D convolution by a Depthwise Convolution, followed by a Pointwise Convolution.

This kind of layer is more efficient at capturing information present at different scales [Chollet, 2017], and offers a better performance than a Dense layer that is commonly used as a bottleneck in many autoencoder structures.

Depthwise Convolution

A Depthwise Convolution is very similar to a convolutional layer, but instead of having kernels as deep as the number of feature maps in the previous layer, we have kernels with depth equal to 1.

For example, suppose the we have 32 feature maps in the previous layer and want to output 128 feature maps to the next layer, using kernels of size 2×2. On a traditional convolution, we would have 128 kernels of size $2 \times 2 \times 32$. On a Depthwise convolution, we would have 128 kernels with size $2 \times 2 \times 1$. One important point here is that the number of output channels must be a multiple of the number of input channels.

Pointwise Convolution

The second part of the Separable Convolution is the Pointwise Convolution, also known as 1×1 convolution. This kind of convolution employs kernels of size $1 \times 1 \times C_i$, where C_i means the number of channels in the previous layer. Because of the kernel size 1, this layer preserves the size of the input layer.

Separable Convolution

In Deep Learning frameworks, the filters parameter of the Separable Convolution indicates the number of filters used by the Pointwise convolution, and the Depth Multiplier parameter the number of filters used by the Depthwise convolution (as a multiple of input feature maps).

2.3.6 Recurrent Layers

When dealing with sequential data, convolutional layers may not be enough to produce good results for the Neural Network, so we use Recurrent layers. These layers are very commonly used when processing text, although they have some uses in when dealing with images and arithmetic encoder contexts.

Theoretically, when using sequential data, we have another dimension, time, to deal with, and convolutional layers would be limited by their "time kernel" to find an answer, and are very sensitive to the ordering of the sequence (requiring different kernels for different orderings). Recurrent layers solve these problems by storing information in a

state variable, similar to a state machine, and therefore they can deal with arbitrarily long sequences dealing only with the state and current sequence term.

LSTM Layers

The most commonly used recurrent layers are LSTM layers
[Hochreiter and Schmidhuber, 1997], an abbreviation of Long Short Term Memory. There are many variations of LSTM layers, such as GRUs [Cho et al., 2014], but here we will describe the model proposed by [Graves, 2013], where the LSTM layer decides at each time step whether the memory should me used, kept, replaced or erased.

For a given instant t, the LSTM layer has three input variables: the current sequence term x_t, the cell state c_{t-1} (i.e., the network memory), and the output of the previous term h_{t-1}. At each time step the layer will produce an output h_t and update the cell state c_t. To do this, the layer calculates some auxiliary variables:

The first one, i_t, is called the input gate, which has four sets of weights: W_{xi}, W_{hi}, e W_{ci}, corresponding to the three inputs, and a bias term b_i. The network multiplies each of the weights by the inputs, adds the bias, and them passes them through a logistic function, as shown in equation 2.29.

$$i_t = \sigma(W_{xi} \cdot x_t + W_{hi} \cdot h_{t-1} + W_{ci} \cdot c_{t-1} + b_i) \tag{2.29}$$

The second one is called the forget gate, f_t. It is calculated like the input gate, with internal weights corresponding to the three input variables, and a bias, as described by equation 2.30.

$$f_t = \sigma(W_{xf} \cdot x_t + W_{hf} \cdot h_{t-1} + W_{cf} \cdot c_{t-1} + b_f) \tag{2.30}$$

With these two gates, we can calculate the new value for the cell memory c_t. The forget gate f_t calculates how much the old value (c_{t-1}) will influence the new value, and the input gate calculates the influence of the input variables x_t and h_{t-1}, after an hyperbolic tangent activation. This is shown on equation 2.31.

$$c_t = f_t c_{t-1} + i_t \tanh(W_{xc} x_t + W_{hc} h_{t-1} + b_c) \tag{2.31}$$

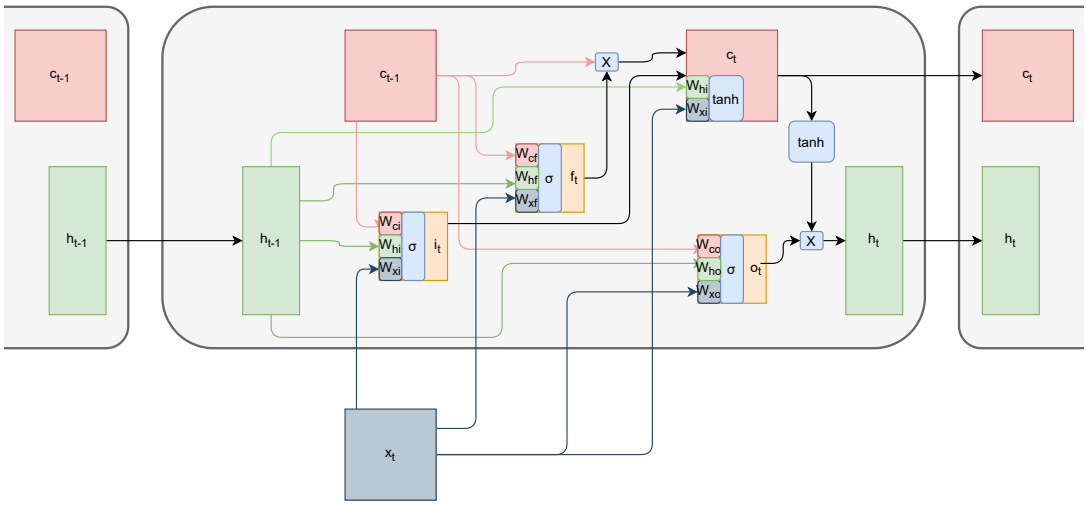

Figure 2.9: Simplified LSTM layer diagram, showing how the internal variables interact with each other through time.

To calculate the current output h_t, we will need an additional auxiliary variable, called the output gate o_t. Again, it is calculated like i_t and f_t, with weights relative to the inputs and a bias term.

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_{t-1} + b_o) \tag{2.32}$$

With the value of o_t, we modulate c_t to obtain the layer output h_t.

$$h_t = o_t \tanh(c_t) \tag{2.33}$$

In Figure 2.9, we illustrate the relationship between these variables.

2.3.7 Depth-to-Space Layers

One last kind of layer that we must mention are Depth-to-Space layers, also known as Pixel Shuffle layers and as Subpixel convolutions [Shi et al., 2016]. These layers make a reordering of the values of convolutions, reducing the number of feature maps and increasing the spatial dimension. For example, suppose an input of size $4 \times 4 \times 512$, the output of a Depth-to-Space layer would have size $8 \times 8 \times 128$, where we doubled the spatial dimension and reduce the number of channels by four. This reordering also involves a reshuffling of the pixels, mixing channels and spatial dimensions on the output.

According to [Shi et al., 2016], using regular convolutions followed by Depth-to-Space layers gives better results than Transposed convolutions, and we see it used in some of the models we discuss.

2.4 Autoencoders

An autoencoder is a neural network that aims to find a smaller representation for its input data [Goodfellow et al., 2016], generally by having as output the same input they received. They are trained under constraints that make them learn useful representations of the data, instead of only learning an identity function.

They are commonly represented as two networks in sequence, the Encoder and the Decoder, that are always trained together, but in many applications used separately. The Encoder receives the input data x and generates a representation r, and the Decoder takes the data representation r and outputs a reconstruction $\hat{x}$ of the data. We can represent this in the following equation:

$$r = \text{Enc}(x, \theta) \tag{2.34}$$

$$\hat{x} = \text{Dec}(r, \phi) \tag{2.35}$$

One way of making the autoencoder generate useful representations is by using *Denoising Autoencoders*: during training, the input is corrupted by random noise, and the expected output is the original input without corruption. To solve this, the autoencoder has to learn about the probabilistic distribution from which the inputs are sampled, and therefore learning this distribution becomes indirectly the training goal.

An even more common approach is what we call *Compressive Autoencoders*, where there is only a size limitation: the Encoder layers become smaller and smaller, until it reaches its smaller size when it passes through the bottleneck layer. The Decoder then restores the representation to the original size. This fitting of the data through the bottleneck makes the network learn an efficient representation of the data. If a compressive autoencoder is trained without any non-linearity, it would must likely learn a transform equivalent to the Karhunen-Loève Transform [Goodfellow et al., 2016]. When using non-linearities like a ReLU, the autoencoder can learn a transform pair with an even better performance than the Karhunen-Loève Transform.

On our case, we focus on two different kinds of autoencoders: inpainting autoencoders and Coding Autoencoders. Inpainting autoencoders are generally trained using an image with some missing portion, called context, and with the target being this missing portion. The idea here is that the representation generated by the encoder will generalize the

context information, and be able to generate a prediction following the same patterns. This can be seen as if the autoencoder is modelling the distribution from which the context is sampled, and them sampling the target from the same distribution.

Coding Autoencoders focus on a different kind of problem, where we need to transmit/store the information from the bottleneck layer. To do this, it needs to be in a binary form. If we consider a floating point with 32 bits, in theory it is able to transmit 2^{32} more information, so the mapping of the bottleneck data into binary values during training must be well considered.

One other point is that all operations performed during training must be differentiable. A standard binarization, as defined on equation 2.36, has zero derivatives at all points, except for the origin, where it is undefined. Therefore, no gradient would be able to pass through a layer like this, and the network would not be able to learn.

$$b(x) = \begin{cases} 1, & x \geq 0 \\ -1, & x < 0 \end{cases} \tag{2.36}$$

One way of going around this issue is to use a stochastic binarizer [Raiko et al., 2015] [Toderici et al., 2016]. Here, given an x belonging to the interval $[0, 1]$, a ϵ is sampled from the following distribution:

$$\epsilon \sim \begin{cases} 1 - x, & \text{with probability } \frac{1+x}{2} \\ -1 - x, & \text{with probability } \frac{1-x}{2} \end{cases} \tag{2.37}$$

The binarized value of x, $b(x)$, is then defined as the sum of x and the noise ϵ, $b(x) = x + \epsilon$, and assumes only the values 1 and -1. To calculate the gradient, we take the derivative from the expected output of the binarization:

$$\frac{d}{dx} b(x) = \frac{d}{dx} \mathrm{E}[b(x)] = \frac{d}{dx} x = 1 \tag{2.38}$$

When this form of binarization is used, the signal is already binarized during training, but with a chance of flipping the bit. This complements the idea that neural networks are resistant to noise, and it acts with a regularizing effect similar to dropout layers.

Another solution to the binarization problem, introduced by [Ballé et al., 2016], is the use of additive uniform noise to simulate binarization during training, which was a common strategy when modeling systems that use binarization [Gray and Neuhoff, 1998]. For quantization intervals with a bin size of 1, we sample a noise $\Delta x \sim \mathcal{U}[-0.5, 0.5]$, and define $b(x)$ as:

$$b(x) = x + \Delta x \tag{2.39}$$

In this case, the addition of uniform noise makes an analytic continuation of the sampling operation, and allows the autoencoder to work with points from the whole interval.

Finally, [Theis et al., 2017] suggests to simply round the inputs, and pass the gradients unchanged, using the same Expectation argument we described before. In this case:

$$b(x) = \text{round}(x) \tag{2.40}$$

$$\frac{d}{dx}b(x) = \frac{d}{dx}\text{E}[\text{round}(b(x))] = 1. \tag{2.41}$$

Chapter 3

Literature Review

In this chapter we discuss some recent results that are related to our work. First, we describe the recent results on Inpainting that use autoencoders, which in turn are used as a inspiration for intra prediction autoencoders. Then, we turn our attention to two different types of autoencoders focused on compression: iterative and recursive autoencoders, and autoencoders based on variational models.

3.1 Inpainting Autoencoders

Inpainting algorithms, such as [Barnes et al., 2009] and [Darabi et al., 2012], are algorithms designed to fill a missing region in a image. This missing region may be caused by small defects that appear over time, when dealing with image restoration, but generally they appear during an image editing process (for example, when removing an unwanted object from a photograph).

Autoencoders generally make an internal model of the images they are fed, so to fill a missing portion from these images, an autoencoder would only need to sample a missing patch from this model, and fill the image with it. Based on this idea, [Pathak et al., 2016] proposed the first CNN architecture to inpaint images. Their idea was to get intact images, remove certain regions from them, and them pass to the network these images with missing regions as inputs, and the extracted regions as the target outputs.

From the several points they raise on their article, the most important one is about the network loss function. According to them, training the network using only MSE leads to blurry images, and therefore it is very important to also train the network using an adversarial model, that helps in generating sharper and more realistic images.

Following these ideas, the first big contribution comes from [Iizuka et al., 2017], that uses not one, but two adversarial networks. Their justification is that one adversarial network is responsible for local consistency, that deals with border artifacts and color

distortions, and the other adversarial network deals with global consistency, dealing with the semantic consistency of the filled region with the rest of the image. To do this, the global adversary has a larger input region than the local adversary. One other innovation from [Iizuka et al., 2017] is the use of Separable Convolutions in inpainting problems.

After that, [Yu et al., 2018] proposes two other changes. The first one is to separate the network in two halfs, with the first half being trained using a SAE loss, and the second half, that receives the first half as input, is also trained using the two adversarial networks. The second change is the introduction of Attention Layers, that create a probabilistic model of the directions the missing pixels come from the original image, improving the results, as there is more pixel translation than pixel generation.

Finally, [Minnen et al., 2017], which is the main inspiration of our work, was the first to propose the use of inpainting networks as intra prediction networks. Here, they train an inpainting network using as input three blocks obtained in raster scan ordering, and that has as target predict a lower right block (shown on figure 3.1). This prediction is used along with the block of the original image to calculate a residual, which is then passed to a compression autoencoder. One improvement we have made over this, described in [Jung et al., 2020] and in Section 4.2.3, is to introduce another intra prediction model. This other model uses an additional block, the top right one, and rearranges the inputs to avoid passing a large black region to the network.

Figure 3.1: Example of input (left) and target (right) of the Intra prediction network used by [Minnen et al., 2017]. The input is the causal context obtained during the compression of the image, as it is a block based codec with raster scan ordering.

3.2 Compression Autoencoders

In this section we describe some recent autoencoder architectures used for compressing image and video. We divide them between iterative and recursive, that follow the line defined by [Toderici et al., 2016], and those based on variational approaches, following the line defined by [Ballé et al., 2016]

3.2.1 Iterative and Recursive Autoencoders for Compression

The first progress on compressing an image using autoencoders comes from [Toderici et al., 2016], that shows one can do a progressive compression of an image (that is, as more bits are used, there is an improvement of quality). They experiment on two main architectures, one iterative and another recursive, schematized on Figure 3.2.

The iterative architecture deals with several networks connected in sequence, where the first network tries to encode the input image (actually a thumbnail of size $32 \times 32 \times 3$), and the following networks try to encode the residuals between the sum of the previous networks outputs and the original images. For a given image I, we define the output of the first network r_1 as $r_1 = \text{Dec}_1(\text{Enc}_1(I))$. Then, each subsequent network will be defined by $r_t = r_{t-1} - \text{Dec}_t(\text{Enc}_t(r_{t-1}))$, and the final reconstruction will be $\hat{I} = \sum_{i=1}^{n} r_n$.

In the recursive architecture, we have only one network, that uses modified LSTM layers and treats the residuals as terms in the sequence being processed by the LSTM layers. One detail here is that although each iteration produces residuals, the loss is always calculated between the sum of the terms of the sequence ($r_t = I - \sum_{i=0}^{t-1} r_i$) and the original image.

There are a few interesting results to point here. The first one is that using convolutional layers generates better results than using Dense layers, which is not unexpected when dealing with images. The second one is to question whether or not the iterative network would perform better using a single network for each iteration ($\text{Enc}_1 = \cdots = \text{Enc}_n$, $\text{Dec}_1 = \cdots = \text{Dec}_n$), or using several different networks. Their conclusion is that using several networks is better, as each level of the residuals has different statistical properties. Finally, they conclude that using a recursive approach is better than an iterative one, and that both their approaches are better than JPEG and JPEG2000 when evaluating through MS-SSIM, even considering they were not trained by optimizing this metric.

[Toderici et al., 2017] extends [Toderici et al., 2016] to work with high resolution images, and not only thumbnails, and obtains many important results:

- To better explore the redundancies in the binary codes generated by the network, it uses a modified version of pixelRNN [van den Oord et al., 2016]. Originally, PixelRNN is a network trained to generate pixels from an image in a raster scan ordering, one at a time, conditioned on past pixels of the network. Here, a CNN called BinaryRNN uses LSTM layers to estimate the probability of the bits in the sequence, and acts as an entropy coder, improving the rate of the model.
- They test if it is better as each iteration to output only the current residual that was given as input, or if at each iteration the network should always output the

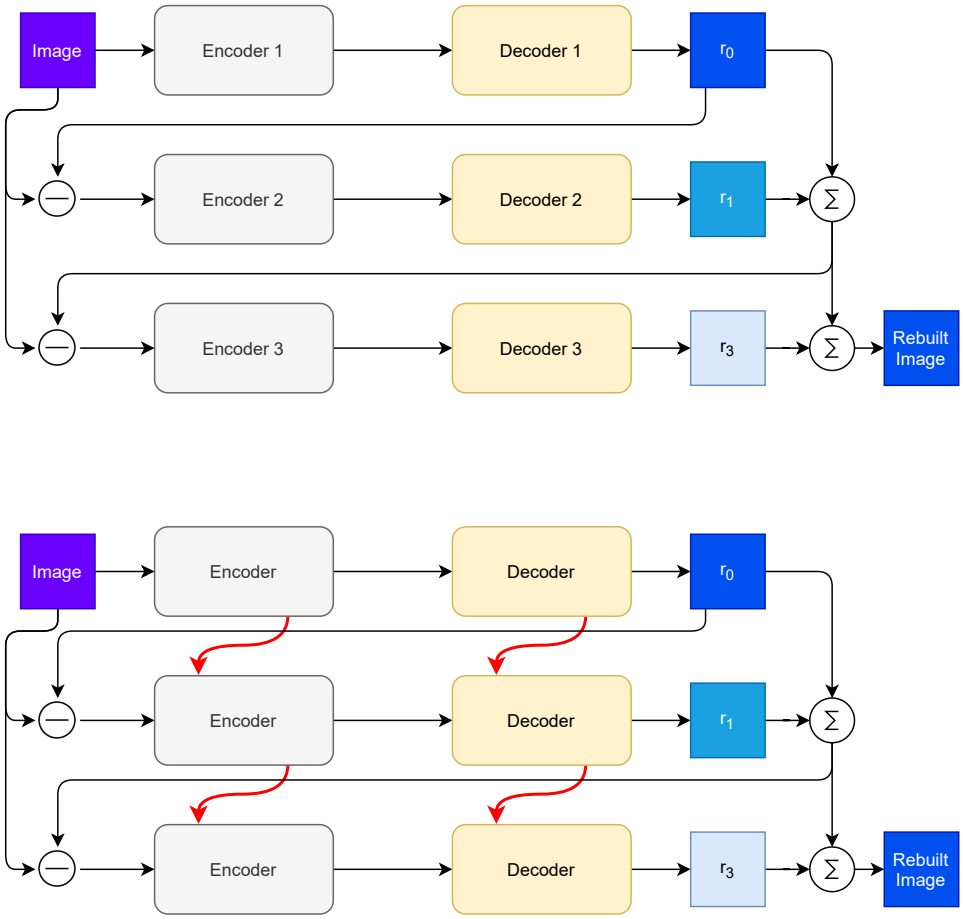

Figure 3.2: Iterative and Recursive Autoencoder Codecs, shown here with 3 levels. On top we show an Iterative Codec, where for each iteration a different network, with different parameters is used. On the bottom, we show a Recursive Codec, where the same network is used in all iterations. The arrows in red are used to symbolize the LSTM layers transmitting information across iterations, but bypassing the bitstream.

current reconstruction of the image (that is, the sum of all previous outputs). The conclusion is that attempting to output the entire image at each iteration offers better results than outputting only the residual.

- They compare the use of a number of different architectures of recurrent layers: LSTM [Hochreiter and Schmidhuber, 1997] [Graves, 2013], associative LSTM [Danihelka et al., 2016], and GRU [Cho et al., 2014]. Their conclusion is that there isn't a clearly better recurrent layer, as each layer performs better for different metrics.
- They also tested whether or not it is useful to scale the residuals by a gain g_t, calculated by a CNN, before passing them to the network. The justification for this is that the residuals at different levels have different ranges of values. This may improve the results depending on the Recurrent layer used and whether or not it is only rebuilding the residual, but is not always useful.
- Finally, they conclude that a network trained on a high entropy dataset performs better than a network trained on a low entropy one.

[Johnston et al., 2018] introduces the concept of *priming*, where the internal states of the recurrent layers are stimulated before coding. While encoding, they pass the original patch several times through the encoder, but the generated codes are discarded. Doing this makes the h_t variables of the GRU layers reach a stage more suitable for coding. After doing this k times, the image is encoded just like in [Toderici et al., 2017]. The decoder is similar, but the decoder attempts to decode the binary signal several times, with its reconstruction being discarded. One extension to this method is to do this at each iteration, which they call *Diffusion*. Doing this outperforms Webp and BPG, but it is extremely heavy computationally.

Another strategy they introduce is combining a L_1 loss with a perceptual loss. To do this, they split the training patch into 8×8 blocks, calculate one minus the SSIM of these blocks, and sum these results. This sum is called S, and the moving average of S is called $\hat{S}$. They them multiply the L_1 loss by $S/\hat{S}$. Practically, this ends up prioritizing hard to encoder images during training, as images that are already well encoded end up having a small L_1 loss.

$$L(x, y) = w(x, y)L_1(x, y) \tag{3.1}$$

$$w(x, y) = \frac{S(x, y)}{\hat{S}} \tag{3.2}$$

One detail, however, is that they trick their framework by defining the weighting w as constant, as this loss is not in fact differentiable. Their weight update equation becomes:

$$\theta' = \theta - \alpha w \nabla_\theta ||Dec_\theta(Enc_\theta(x)) - x|| \tag{3.3}$$

Finally, when coding images, they employ what they call SABR (Spatially Variable Bit Rate), by setting a distortion threshold for each patch, and passing each patch through the network several times until that distortion is reached. The information of how many times each patch is encoded is passed as a side information. [Covell et al., 2017], it also uses masks during training that turn to zero some of the binary representations. This skews the distribution of the binary values toward zero, and makes the entropy coder more efficient (here, they use a LZ77 [Ziv and Lempel, 1977]. According to the authors, training a network this way also makes the patch quality more uniform, which improves human perception score.

Finally, as we described in the previous section, [Minnen et al., 2017] introduces the use of a predictive network as a first step in coding. This improve mostly first level results. Additionally, they also use a simple adaptive bit-rate algorithm, that uses a larger bit-rate according to how a hard to encode a patch is, by setting thresholds a patch need to reach. If a patch has reached the threshold, it finishes being encoded. Otherwise, it keeps being encoded.

3.2.2 Compression Autoencoders based on Variational Models

In this section, we detail a second class of models used for coding images, which are inspired by Variational Autoencoders, or VAE [Kingma and Welling, 2014], that are autoencoder models which implement a variational hypothesis on the data, modelling the distribution of the data as if it is being sampled from a latent variable. That is, it is assumed that for every point x sampled from the distribution of the data, there is also a unique point y to some latent distribution, forming a bijection.

In a practical case, VAEs are modelled assuming the latent variable follows a normal distribution with zero mean and unitary variance. To do this, networks are trained to minimize two different restrictions: the first is a distortion loss, that measures the distortion between the original x and the reconstructed $\hat{x}$ data, which is in most cases either a Mean Absolute Error or Mean Squared Error loss, although [Ballé et al., 2016] generalizes this formulation. The second training restriction, i.e. that the latent representation, $y = \text{Enc}(x)$ can be interpreted as if y is sampled from a normalized Gaussian distribution ($p_y = \mathcal{N}(0, 1)$), is obtained by minimizing the Kullback-Leibler Divergence [Cover, 1999] between the distribution p_y and the normal distribution.

Although there are many possible uses for Variational Autoencoders, mainly focused on data generation, data compression is the one we are interested here. The first notable attempt at this is from [Ballé et al., 2016], that realizes there is analogy between the Rate-Distortion Optimization problem and the definition of a VAE. They train an autoencoder to minimize the entropy of the codes generated by the model and the distortion, weighted by a parameter λ, which controls the model position in the RDO curve (one model is trained for each lambda). This is shown in the next equation, which shows the loss they use minimizes simultaneously both distortion and rate.

$$\mathrm{L}(\phi,\theta) = \mathbb{E}_{\mathbf{x},\Delta\mathbf{y}}\Bigg[-\sum_i \log_2 p_{\tilde{y}_i}\left(\mathrm{Enc}(\mathbf{x};\phi)+\Delta\mathbf{y}\right) + \\ +\lambda\ \mathrm{MSE}(\mathbf{x}, \mathrm{Dec}\left(\mathrm{Enc}(\mathbf{x};\phi)+\Delta\mathbf{y}\right);\theta)\Bigg] \tag{3.4}$$

The parameter Δy in equation 3.4 is sampled from a random uniform noise $U \sim [-\frac{1}{2};\frac{1}{2}]$, and simulates quantization noise during training. We note that [Ballé et al., 2016] does not work with binary values properly, but instead quantizes its produced values to the nearest integer, and then binarizes these values and passes them through a range coder.

One of the reasons this works so well is that the model of [Ballé et al., 2016] uses as non-linearities the so called Generalized Divisive Normalization (GDN) layers (first defined in [Ballé et al., 2016] , that are layers that favour Gaussianization of the data, which is a process of transforming, as best as possible, a given probability distribution into a normal/Gaussian distribution, as well as obtaining at the same time the inverse transform pair that maps from the Gaussian distribution to the original distribution [Chen and Gopinath, 2001]. GDNs have received this name because they were originally presented as a generic formula that can represent several different schemes for Gaussianization, but the formulation that ends up being used is simpler.

In the equation bellow, we present the equation for the GDN layers as used by [Ballé et al., 2016]. u_i represents i-th feature map that is being calculated, and w_i represents the i-th feature map of the previous layer. The learnable parameters are β (one for each feature map) and γ (one for each combination of feature maps). As we can see, to obtain the normalized response, the GDN layer divides each pixel by a weighted summation of pixels at the same coordinates in other feature maps.

$$u_i(m,n) = \frac{w_i(m,n)}{\beta_i + \sum_j \gamma_{ij}(w_j(m,n))^2} \tag{3.5}$$

The IGDN equation, presented below, is very similar, but instead of division, there is a multiplication.

$$u_i(m,n) = w_i(m,n) \cdot (\beta_i + \sum_j \gamma_{i,j}(w_j(m,n))^2)^{1/2} \tag{3.6}$$

The great advantage of these layers is that Gaussianization was a very computer expensive process [Laparra et al., 2009], but by stacking several simpler transforms, it greatly reduced the computational cost and improved performance. One possible use of such a transform is image generation (which is, in fact, an example case of data synthesis): given a transform pair, we can sample from a Gaussian distribution, and use the inverse transform to generate pairs as if they were sampled from the original transform.

Following that work, [Ballé et al., 2018] notices that the latent distribution itself could be better modeled, as there is a correlation between the latent data and the model. To do this, they start modeling the latent using a Gaussian Mixture, and them start to estimate the variance parameters of this Mixture. This prior on the prior of the data is called hyperprior, and is passed together in the bitstream as side information.

[Theis et al., 2017] also uses as basis the work of [Ballé et al., 2016]. However, there are many interesting differences. First, the model uses more traditional CNN techniques: instead of GDN's, it uses leaky ReLU's [Xu et al., 2015] as non-linearities. It also uses residual blocks [He et al., 2016], and instead of using transposed convolutions it first performs dimension preserving convolutions, followed by Depth-to-Space layers [Shi et al., 2016]. To improve training, the authors implement a so called incremental training, where the latent is masked during training, and as quality thresholds are reached, the mask allows more and more of the latent terms to be used by the network. Modelling of the latent is done by Gaussian mixtures, an idea that would be followed by [Ballé et al., 2018]. As for quantization, the strategy used is to round coefficients during the forward pass, but to ignore this rounding during the backward pass, passing gradients unchanged.

The main innovation in this paper is the idea of using fine-tuned scales to generate multiple points on the Rate-Distortion curve. Essentially, before quantizing the values of the representation, each parameter is multiplied by by a scaling factor, that possibly changes the mapping of the values during quantization by an arithmetic encoder, in turn changing the rate and quality of the compression. On the decoder side, coefficients are then divided by the same scales of the encoder. This process is analogous to changes on quantization bin size, as later proposed (but not strictly mentioned) by [Choi et al., 2019] . To obtain the best value for each scale, the authors train the scaling parameters as fine-tunings of an already trained model, and to generate even more points they also

interpolate scales. Although in theory this could be used to generate points all over the curve from a single trained model, it is noted that this approach is suboptimal, and ideally there should be a balance between fully trained models and scales.

3.3 Conclusions

As a final point to this chapter, we want to remark a couple things. First, that the task of inpainting is a completely separate area of research, with a small overlap with our focus of coding, and therefore that our review here can only be superficial.

Secondly, we need to point out that most of the literature has followed the line of the Variational Models listed here, instead of the Recursive models. Recursive models, as we shall see, do not give as good results as the variational models, are slower, and are more computationally expensive. Much of the research in this kind of models, has been left behind to pursue Variational Models.

This change is reflected in the next chapters, where we discuss our methodology and the results of our experiments. Sections 4.2 and 5.3 cover our research using recursive models, and published in [Jung et al., 2020], but we have also performed experiments using VAE based codecs in a search to improve our results.

Chapter 4

Proposed Methodology

The main idea of our proposal is to see whether or not ideas that work on traditional image codecs will improve the performance of learned image codecs. Although much work has been done in the opposite direction, where neural networks are used to improve traditional codecs (for example: [Li et al., 2020] [Sun et al., 2020], we followed the line of [Minnen et al., 2017] that introduces intra prediction on codecs based on neural networks.

Traditionally, codecs such as H.264 [Wiegand et al., 2003] and
HEVC [Sullivan et al., 2012] use multiple modes for intra prediction, as there is a possibility of generating better predictions and therefore improving compression performance. Following the idea that more modes are better, we proposed the idea of a multi-mode learned image codec [Jung et al., 2020], which we describe in the following sections, and is based on iterative autoencoders (section 3.2.1). Additionally, we also compare it to a similar codec, but based on Variational Autoencoders (section 3.2.2).

4.1 Autoencoders for Intra Prediction

We begin by detailing our autoencoder models for generating intra predictions. The first one follows the same architecture as described in [Minnen et al., 2017]. This network is based on the inpainting ideas of [Pathak et al., 2016], and frames intra prediction as an inpainting problem. To do this, the network takes as input the neighbouring blocks above and to the left (figure 4.1), already encoded during raster scan ordering, and generates a prediction for the patch currently being encoded.

This network is a sequential model, with a simple topology where the output of a layer is the input of the next layer, and most of its layers are traditional Convolution and Transposed Convolution layers, followed by ReLU activations. However, the middle layer is a Separable Convolution layer, which, as described in section 3.2.1, consists of a Depthwise Convolution (itself a special convolution that takes into account only one

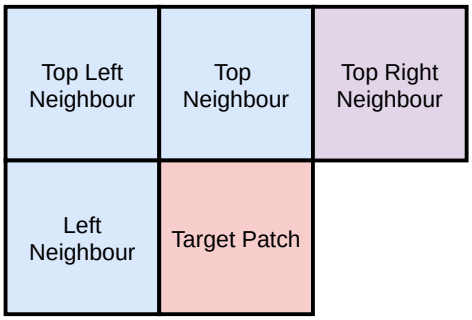

Figure 4.1: Block disposition during raster-scan ordering. On traditional schemes, only the neighbouring blocks to the left and to the top (in light blue) are used to generate intra predictions. We use the top right neighbour in our second intra prediction autoencoder as it contains additional information that is already available at the decoder.

feature map from the previous layer), followed by a Pointwise Convolution, and then a reshape layer. This network is detailed in figure 4.2 and table 4.1.

Table 4.1: Architecture of the Intra Prediction Network

Layer	**Output Shape**	**Kernel Size**	**Strides**	**Padding**	**Activation**	**Parameters**
Input Layer	64x64x3	-	-	-	-	-
Conv2D	32x32x64	4	2	Same	ReLU	3,136
Conv2D	16x16x128	4	2	Same	ReLU	131,200
Conv2D	8x8x256	4	2	Same	ReLU	524,544
Conv2D	4x4x512	4	2	Same	ReLU	2,097,664
DepthwiseConv	1x1x8192	4	-	Valid	ReLU	139,264
Reshape	4x4x512	-	-	-	-	-
Conv2D	4x4x512	4	2	Valid	ReLU	262,656
Conv2DTranspose	8x8x256	4	2	Same	ReLU	2,097,408
Conv2DTranspose	16x16x128	4	2	Same	ReLU	524,416
Conv2DTranspose	32x32x64	4	2	Same	ReLU	131,136
Conv2D	32x32x3	1	1	Same	ReLU	195
Total Parameters						5,911,619

Our second intra prediction autoencoder is very similar to this one, with only a few differences. First, we realised that including the top right neighbour, which is also available during raster scan ordering, could help in some situations. Then, we decided to concatenate the blocks, instead of simply passing a larger image with a blacked out part where the target should be (figure 3.1). This helps because deeper networks perform better than broader networks, and prevents several of the network activation being zero in the first few layers. Now, instead of passing an input of shape $(64, 64, 3)$, our input has shape $(4, 32, 32, 3)$. The changes in the architecture are the introduction of 3D convolutions in the first two layers, to deal with this change on the input shape.

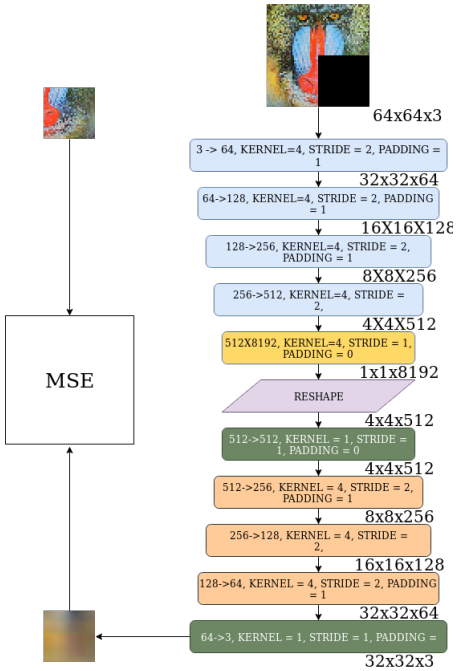

Figure 4.2: Schematics for the Intra Prediction Autoencoder developed by [Minnen et al., 2017] and also used in our Recursive Codecs.

4.2 Recursive Autoencoder Codecs for Compression

In this section, we describe three different codec models to do our analysis. The first one is a codec without intra prediction, the second one has one mode of intra prediction, using the first intra autoencoder from the previous section, and the third one has multiple modes of intra prediction.

4.2.1 Baseline Recursive Autoencoder Codec

Our baseline codec is based on the architecture used by [Toderici et al., 2017], with its layers detailed in table 4.2 and schematized on figure 4.3. This is a recursive autoencoder that receives as input an image patch of size 32×32, and tries to reconstruct it on its outputs. On the first pass through the network, the input patch x is reconstructed as r_0. On the second pass, and on following iterations, the network receives as input the difference between the sum of all previous outputs and the original image x, which we call residue. Therefore, with the exception of the first pass, this is a residual encoding

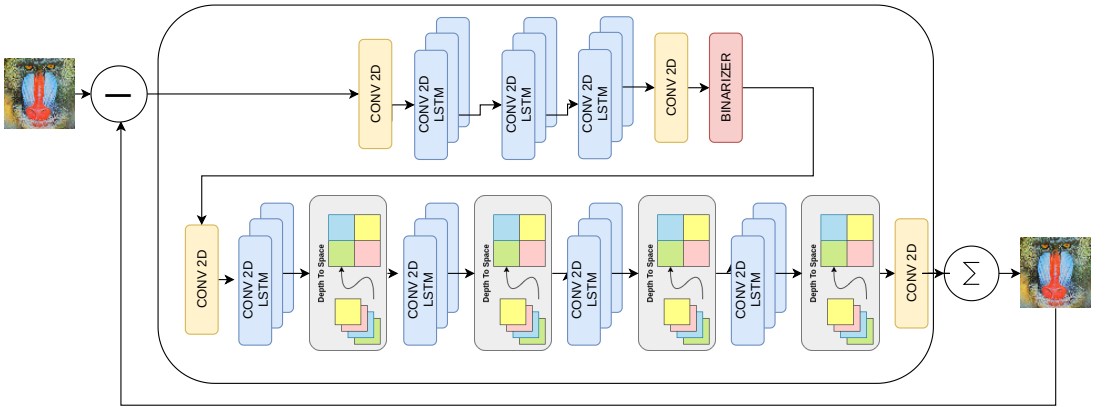

Figure 4.3: Baseline Recursive Codec diagram

network. The equations representing this network are:

$$r_0 = \text{Enc}\Big(\text{Dec}(x)\Big) \tag{4.1}$$

$$r_n = \text{Enc}\Big(\text{Dec}(x - \sum_{i=0}^{n-1} r_i)\Big) \tag{4.2}$$

To work well, this network has LSTM layers, that have an internal state and are able to preserve some of the information from the previous residues that were passed through the network (one detail is that these layers are custom LSTM layers, as the next term in the sequence is not available, as is not generally the case when training with sequential data). The encoder consists of a 2D convolution, followed by three 2D Convolutional LSTM layers, and then a Binarizer layer. This binarizing layer samples a random noise ϵ based on the value x (equation 4.3) and then adds this value, mapping all values to either 1 or -1.

$$\epsilon \sim \begin{cases} 1 - x, & \text{with probability } \frac{1+x}{2} \\ -x - 1, & \text{with probability } \frac{1-x}{2} \end{cases} \tag{4.3}$$

The decoder architecture is similar, but it also has Depth-to-Space layers between its 2D LSTM convolution layers, so that it can increase the shape of the patch being compressed back to its original size.

4.2.2 Single-mode Autoencoder Codec

One improvement on the baseline codec, introduced by [Minnen et al., 2017], is to use intra prediction. Here, this is done by calculating a prediction based on the neighbouring patches, calculating the residual between the block we want to code, and then compressing this residual.

Table 4.2: Architecture of our Baseline Codec

	Layer	Output Shape	Kernel/ Hidden Kernel Size	Strides	Padding	Activation	Parameters
Encoder	Input Layer	32x32x3			-		
	Conv2D	16x16x64	3/ -	2	Same	Linear	1,728
	Conv2DLSTM	8x8x256	3/1	2	Same	Sigmoid	851,968
	Conv2DLSTM	4x4x512	3/1	2	Same	Sigmoid	5,767,168
	Conv2DLSTM	2x2x512	3/1	2	Same	Sigmoid	10,485,760
	Conv2D	2x2x32	1/-	1	Valid	Tanh	16,384
	Binarizer	2x2x32			-		
Decoder	Conv2D	2x2x512	1/-	1	Same	Linear	16,384
	Conv2DLSTM	2x2x512	3/1	1	Same	Sigmoid	10,485,760
	DepthToSpace	4x4x128			-		
	Conv2DLSTM	4x4x512	3/1	1	Same	Sigmoid	3,407,872
	DepthToSpace	8x8x128			-		
	Conv2DLSTM	8x8x256	3/1	2	Same	Sigmoid	1,441,792
	DepthToSpace	16x16x64			-		
	Conv2DLSTM	16x16x128	3/1	2	Same	Sigmoid	360,448
	DepthToSpace	32x32x32			-		
	Conv2D	32x32x3	1/-	1	Same	Tanh	96
Total Parameters							32,835,360

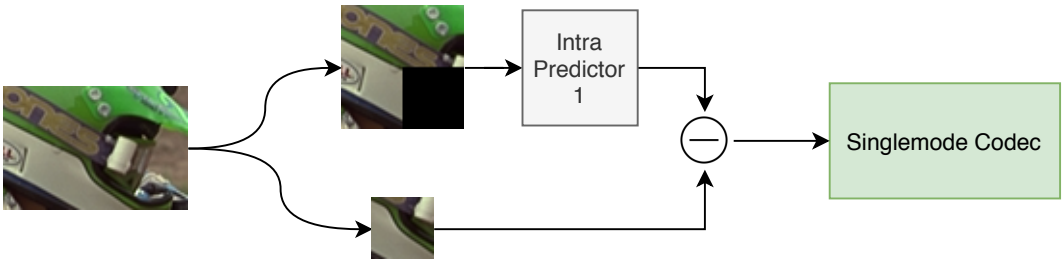

Figure 4.4: Codec with only one intra prediction mode

This model is represented on figure 4.4. The intra predictor we used is described on 4.1. This has the advantage that all levels of the network are dealing with residuals.

One point to note here is that, as there is only one mode, it is implicit, and does not need to be sent into the bitstream.

4.2.3 Multi-mode Autoencoder Codec

Finally, we present a codec that has multiple intra prediction modes. To do this, when extracting the image patches, we generate three kinds of predictions:

The first one is the same from our Single-mode Codec, described on the previous section. The second one is similar, based on generating intra predictions using an autoencoder, but uses a different input, as described on section 4.1. Using two autoencoders for intra prediction is justifiable as the differences in their architectures make them comple-

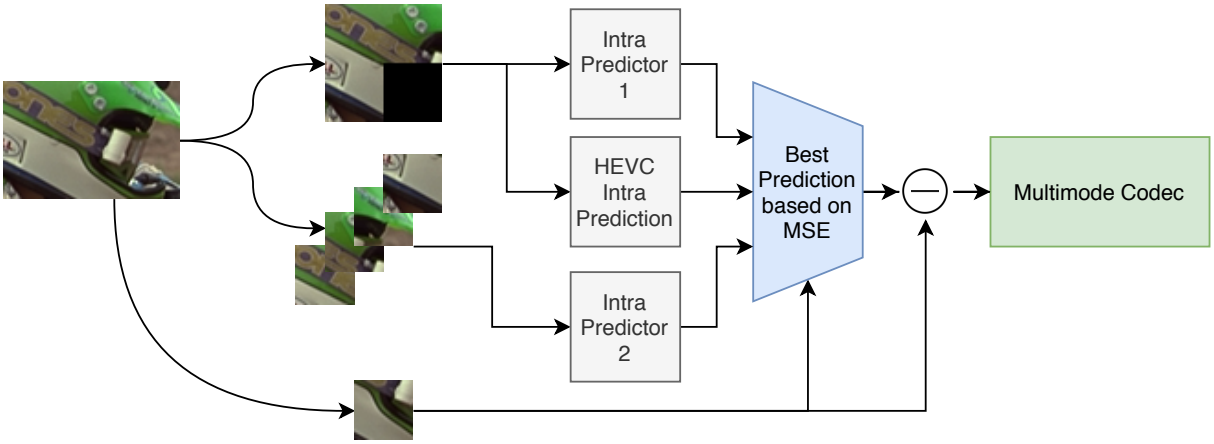

Figure 4.5: Multi-mode Codec during training

ment each other in some situations, although they do indeed produce very similar results in several cases.

The third kind of prediction we use is HEVC intra prediction modes [Lainema et al., 2012]. The idea behind using HEVC predictions is that in many situations, the HEVC directional predictions are more than enough, specially in high resolution images, where each patch is simpler.

During training, the best prediction is chosen based on the MSE between the prediction and the original patch. After training, during test phase, we encode all predictions, and pick the best one by calculating the MSE between the output patch and the original input.

4.3 Variational Based Encoder

In this section, we detail our experiments of encoding residues using as baseline the architecture proposed by [Ballé et al., 2016]. This change from [Toderici et al., 2017] in the main codec architecture is based on two important points. First, encoding and decoding using the baseline architecture of [Toderici et al., 2017] is a slow process, as each patch needs to be encoded and decoded for each possible iteration level. Although parallelism allows this to be computed in a somewhat feasible time, our algorithm for intra prediction is inherently sequential, which makes coding and decoding large images unmanageable. Another important point is that our recursive codecs are progressive, and although it has uses in many applications, it also has the drawback that encoding high quality images takes longer, as all lower quality points need to be encoded first.

Secondly, [Ballé et al., 2016] has better Rate-Distortion performance than [Minnen et al., 2017], which has caused a shift on the literature in the field, as exempli-

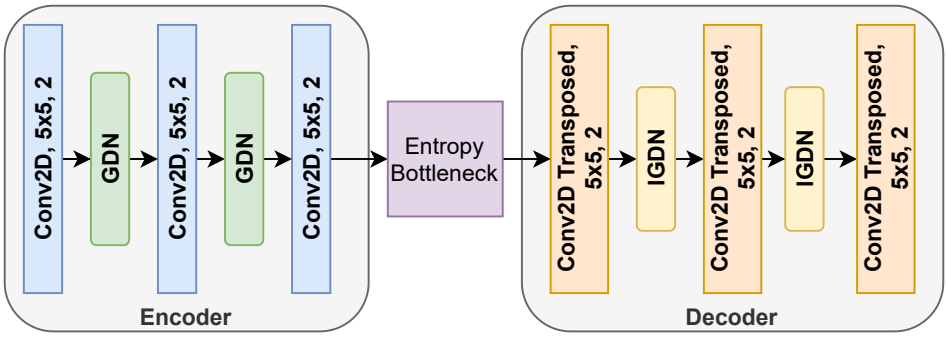

Figure 4.6: Architecture of our Variational Based Codec

fied by [Choi et al., 2019], [Lee et al., 2019], [Zhou et al., 2019] and [Akbari et al., 2020]. Therefore, implementing our ideas on this framework is a next logical step.

In the following sections, we describe the framework of [Ballé et al., 2016], and the changes we have had to make to perform intra prediction in this framework.

4.3.1 Baseline VAE Codec

The baseline codec we use here is based on the codec developed by [Ballé et al., 2016]. The encoder consists of three 2D convolutions, the first layer has a kernel of size 9×9, and the other two have kernels of size 5×5. Every layer has same padding and strides of 2, with the first two followed by GDN layers, and the last convolutional layer without any activation. The decoder is a flipped version of the encoder, with two transposed 2D convolutions of kernel size 5×5 and a transposed 2D convolution with kernel size 9×9. The first two layers have inverse GDN layers as activations, and the last layer has no activation function.

The output of the last layer of the encoder passes through an Entropy Bottleneck, which during training emulates quantization by adding uniform noise, and after the training quantizes the bitstream. The Entropy Bottleneck is also responsible for modelling the distribution of the data, and it passes these models to a range coder during the test phase.

One important point is this codec is that it is Fully Convolutional, and therefore it has no restrictions on the patch size used during either training or testing (other than memory limits on the computer being used). This is a great advantaged over the Recursive Codecs we have discussed, that were limited by their architecture to only deal with patches of size 32×32.

For this codec, we have to train one model for each point on the RD curve, instead of having one model that fits all points.

Table 4.3: Architecture of the baseline VAE Codec

	Layer	Kernel Size	Strides	Padding	Parameters
Encoder	Conv2D	5x5	2	Same	31,104
	GDN	-	-	-	16,384
	Conv2D	5x5	2	Same	409,600
	GDN	-	-	-	16,384
	Conv2D	5x5	2	Same	409,600
Entropy Bottleneck					
Decoder	Conv2DTranspose	5x5	2	Same	409,600
	IGDN	-	-	-	16,384
	Conv2DTranspose	5x5	2	Same	409,600
	IGDN	-	-	-	16,384
	Conv2DTranspose	5x5	2	Same	31,104
Total Parameters					1,766,144

4.3.2 VAE Codec with Intra Prediction

As the VAE codec has only convolutional layers, it generally works by compressing the entire image in one step, and is therefore not patch based. However, switching to a patch based approach is less RAM intensive when compressing 4k and UHD images.

If we want to use the same Intra Prediction autoencoder we used in our single-mode codec, we are restricted to using a patch size of 32×32, as the Separable Convolutions used in our intra prediction are restricted to this size. We have therefore performed two experiments. The first one with the same intra prediction autoencoder from our single-mode codec, and the second one with an intra predictor using only convolutional layers, trained end-to-end.

VAE Codec with pre-trained Intra Prediction Autoencoder

This codec uses a structure similar to our single-mode codec, where we generate an intra prediction based on the neighbouring patches and using our intra prediction autoencoder. This intra prediction is then subtracted from the patch we are dealing with, generating a residual, which is then compressed using our baseline VAE codec. We then again add our prediction to the decompressed residual, generating our final result.

This codec is schematized in figure 4.7 . As we described in the previous section, this codec is limited to a patch size of 32.

VAE Codec with embedded Intra Prediction

Our final codec to be considered is our VAE Codec with embedded Intra Prediction. To take advantage of the variable patch size approach, we have decided on also training a

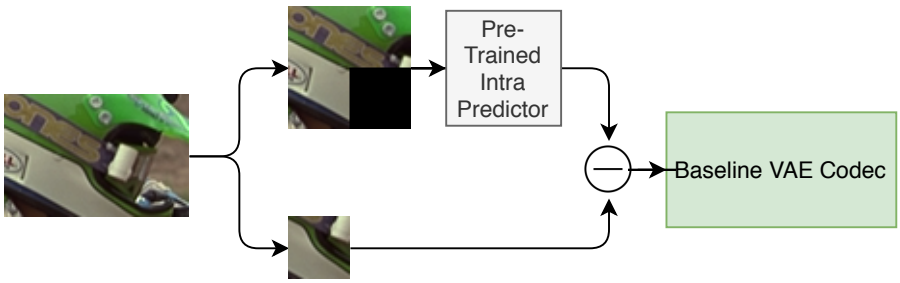

Figure 4.7: VAE codec with pre-trained Intra Prediction Network. The Intra Prediction Network used here is the same as used by our Single-mode Codec and originally designed by [Minnen et al., 2017].

codec with an embedded intra prediction. This intra prediction network is trained end-to-end with the baseline VAE codec, and is composed of three convolutional layers with leaky ReLU activations, followed by three transposed convolution layers followed by leaky ReLU activations. This module is described on table 4.4.

This intra prediction network uses as input the same context we used in our second intra prediction autoencoder, but here the four context patches are stacked in the channel dimension. Therefore it has input patches of shape $(Batchsize, Patchsize, Patchsize, 12)$, and outputs a prediction of shape $(Batchsize, Patchsize, Patchsize, 3)$, from which we calculate the residual and pass it to our baseline VAE codec. In figure 4.8, we have a diagram showing the structure of this codec.

Table 4.4: Architecture of the embedded Intra Prediction Network

Layer	**Kernel Size**	**Strides**	**Padding**	**Activation**	**Parameters**
Conv2D	5x5	2	Same	LeakyReLU	38,400
Conv2D	5x5	2	Same	LeakyReLU	409,600
Conv2D	5x5	2	Same	LeakyReLU	409,600
Conv2DTranspose	5x5	2	Same	LeakyReLU	409,600
Conv2DTranspose	5x5	2	Same	LeakyReLU	409,600
Conv2DTranspose	5x5	2	Same	LeakyReLU	9600
Total Parameters					1,686,400

4.4 Conclusions

This chapter has presented the overall structure of the codecs we have decided to compare. In the next chapter, we first describe the results of our Inpainting Autoencoders against HEVC modes, without any compression involved.

We then describe how we have trained our codecs, how extensive the training was, and for the VAE codecs the effects of different patch sizes. Afterwards, we compare the

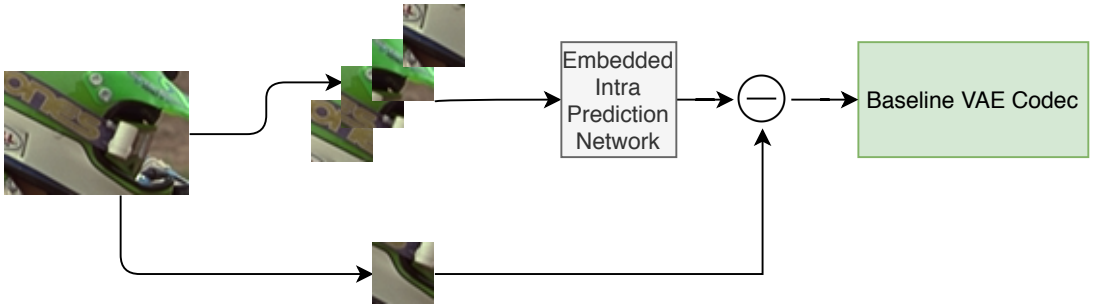

Figure 4.8: VAE codec with embedded Intra Prediction Network. Note that the Context Patches are stacked in the Channels dimension, and that the intra model is trained together with the rest of the codec.

PSNR and MS-SSIM curves for each of the codecs, and at the end the results of these codecs against JPEG and JPEG 2000 image codecs.

Chapter 5

Results

In this chapter we detail the experiments we have performed to evaluate our codecs. We start by describing the datasets we used, and then we discuss the results of our models. First, we detail our intra encoders as standalone models, and then discuss bot recursive and variational codecs.

5.1 Datasets

To perform our experiments, we have used two kinds of image datasets, the training dataset and the test dataset. All of our models were trained using the same dataset, and all results evaluated using the same test dataset.

To build a large training dataset, we have downloaded the CLIC, DIV2K [Agustsson and Timofte, 2017], Flickr2K [Lim et al., 2017], Ultra-Eye [Nemoto et al., 2014] and MCL-JCI [Jin et al., 2016] datasets. The CLIC dataset is divided into images taken by professional photographers and images taken by mobile phone users. We have ignored the distinction between *professional* and *mobile* images, and taken all CLIC training images. The DIV2k dataset was originally designed for Super Resolution problems, and has 800 training images. The Flickr2K dataset has 2650 images with 2k resolution taken from Flickr. The Ultra-Eye was originally a dataset used for eye-tracking information, but we have used only the raw images, consisting of 41 4K images. The MCL-JCI dataset has 50 raw images with 2K resolution.

From each image in these datasets, we have extracted random cropped patches of size 256×256, and saved them in a different folder. This helps reduce CPU use, as otherwise we would need to decode an entire 2K image to extract a single small patch.

One common practice in computer vision problems is the use of data augmentation, where images are flipped, rotated, and slight color variations are applied to the dataset

to increase its size. In our case, this does not seem necessary, as our dataset was large enough, and none of our networks presented overfitting problems.

All of our tests were conducted using the Kodak dataset. This dataset is composed of 24 images of size 768×512, and is very commonly used by the image processing community.

5.2 Intra Prediction Autoencoders

The autoencoders for intra prediction, described in section 4.1, can be evaluated both by judging subjectively the images being generated, and by measuring their performance against more traditional intra prediction methods.

To give a subjective evaluation, we can take a look at the images in Figure 5.1. This image shows two example cases of the Inpainting Autoencoder used for Intra Prediction, showing both the causal context, the ground truth, and the prediction generated by the model.

As we can see, the model does generate sensible predictions in all cases, considering the context. These predictions are in general better than HEVC predictions, as they are in general non-linear, as shown by the flower on the left of Figure 5.1. The iamge on the right, however, generates a plausible prediction, but the image has new objects that were unpredictable given the context. Although the prediction is bad, no HEVC prediction would be able to predict these objects, and therefore it is still competitive.

Figure 5.1: Examples of predictions by the Intra Prediction Autoencoder. Inside each sub-image, left is the input, top right the ground truth image, and bottom right the generated prediction. On the both right and left images, we see the Autoencoder is making sensible predictions, but the prediction on the right, although plausible, does not match the actual image, as it has objects that could not be predicted by the causal context.

To evaluate the results in a more objective manner, one thing we can compare how many times more the AI mode would be chosen instead of the HEVC modes. Here, the choice is made on the MSE between the generated prediction and the original image. We did not take rate into account, as we are judging the intra prediction by itself, but that can make a difference in a complete codec.

In Figure 5.2, we compare, for each image in the Kodak dataset, whether or not the residuals generated by the Intra autoencoder would be chosen. As we can see, except for one image, it is chosen at least 30% of the time, and for kodim06 and kodim22 it is chosen over 60% of the time.

In Figure 5.3, we see a distribution of the modes chosen. In blue, we see how the HEVC modes (0 to 34) would be selected if they were the only available option. In red, we see how the modes would be chosen if we included the intra predictor as an option (mode 35). As we can see, DC and Planar modes are the most commonly chosen modes, and there are two peaks for Horizontal and Vertical modes. When the AI mode is introduced, almost 50% of the time it is chosen, most of the time replacing DC and Planar modes.

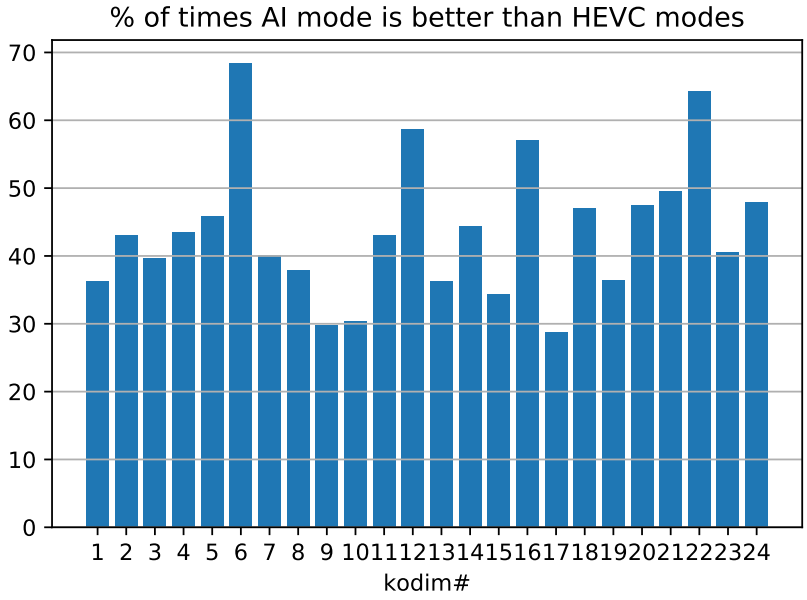

Figure 5.2: Percentage of the results where predictions generated by the Intra Autoencoder have lower MSE than the ones generated by the best HEVC mode. Each number on the bottom corresponds to the index of an image from the Kodak dataset.

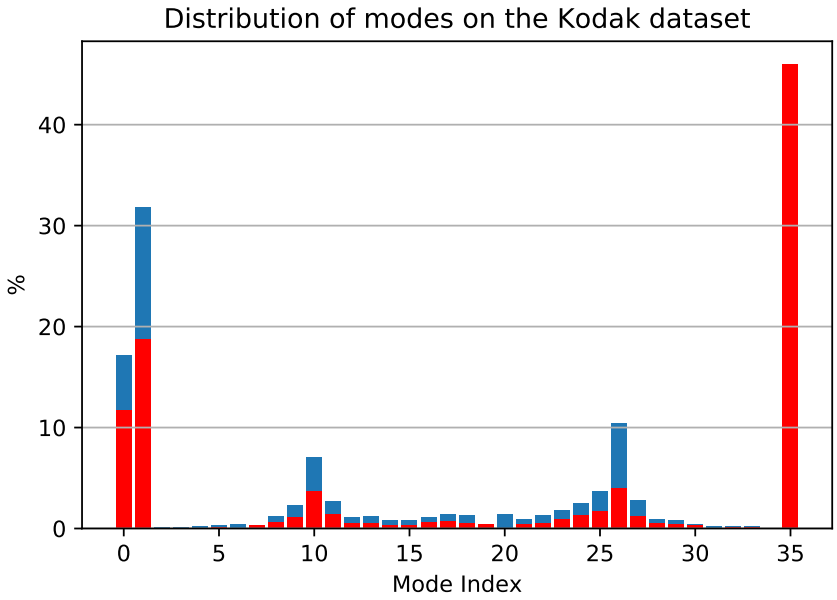

Figure 5.3: Distribution of Intra prediction modes in the Kodak dataset. In blue, we show how the modes are distributed when only HEVC modes are available. Notably, Planar and DC modes at index 0 and 1 account for nearly 50% of the modes chosen, with the rest being distributed along the 35 angular modes, with little peaks for Horizontal and Vertival modes. In red, we show how the distribution shifts as we introduce the Autoencoder Mode. This mode is represented by index 35, and we see a reduction in values from all other modes, but specially DC and Planar, which now are used about 30% of the time.

5.3 Recursive codecs

In this section, we describe the recursive codecs used. We first detail the baseline codec, and then our two recursive codecs that use intra prediction: the single-mode codec and the multi-mode codec. We then compare the use of a bit allocation algorithm, and the use of Range Coders to improve the performance of the codecs.

5.3.1 Baseline Recursive Codec

The first step in our codecs is defining how much they need to be trained. To do this, we compare the performance of our baseline codec along training. We train each of these codecs using a batch of size 64, and as we can see from Figure 5.4, there is a saturation in the performance of our baseline codec as we train it for over 300 thousand iterations.

This codec, as described in section 4.2.1, is based on the codec proposed by [Toderici et al., 2017]. It is a recursive codec that uses convolutional LSTM layers, that attempts to rebuild the patch it was given as input. Each pass of the patch through the network is called an iteration, and seem by the LSTM layers as one term in the sequence they are dealing with.

For our project, we trained the codec to deal with 10 levels of iterations. This was based on practical reasons, as increasing the number of iterations would require the use of a larger GPU. Also, it leaves our codec in a low to mid range considering the bits-per-pixel (bpp) rate. At higher rates, codecs such as JPEG2000 start to become nearly lossless codecs, and therefore the comparison is not so interesting.

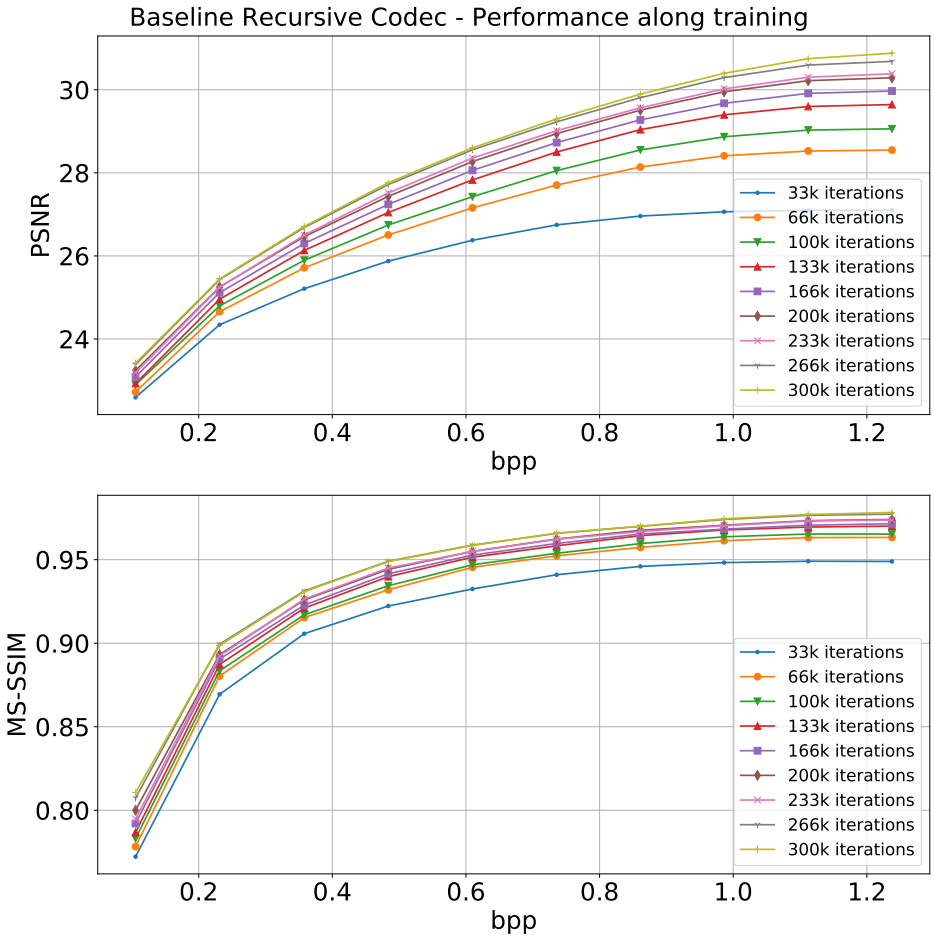

Figure 5.4: Performance of our baseline recursive codec along training, averaged over the results from the Kodak dataset. As we can see, the distance between the curves for higher rates gets smaller as we reach 300,000 iterations, showing that training the model for more iterations will not improve the results. For lower rates, saturation occurs earlier.

Figure 5.5: Example where DPCM generates a bad prediction: the context for the prediction is very different from the patch that needs to be encoded, so using intra prediction actually degrades the performance for the codec. This image was encoded by the Single-mode Codec, and our hypothesis is that patches similar to this one cause degrade the performance of the codec for all images.

5.3.2 Single-mode codec

The first variation in our codec is the introduction of an intra prediction mode. This mode is generated by another network, which was pre-trained and described in Section 5.2.

During training, we extract patches of size 64×64, and divide them into a patch of size 32×32, and a patch of size 64, with a masked bottom right portion, and pass this to the intra prediction autoencoder that generates a prediction.

This prediction is then subtracted from the bottom right of the input patch, making our residual. This residual is then passed to a codec with the same architecture as our baseline codec, and them the output of the codec is then added to the prediction again, and the loss is calculated.

One important point to note is that this codec is trained to encode residuals, and although it has the same architecture as the baseline codec, it has different weights.

Ideally, as residuals have a simpler distribution than images, we would expect this codec to have a better performance. However, as this is a single-mode codec, there are a number of situations where the predictions generated by the network are very different from the actual patch. The plots on Figure 5.6 show that this codec under performs the baseline at all rates.

One example is illustrated in Figure 5.5. Here, the prediction, although consistent with the neighbouring patch, is very different from the patch to be coded. Because of this, the codec actually has to correct the prediction, instead of the prediction helping the codec.

5.3.3 Multi-mode codec

The third variation on our baseline codec is our multi-mode codec, that has several intra prediction modes to choose from. First, we have two intra prediction generated by au-

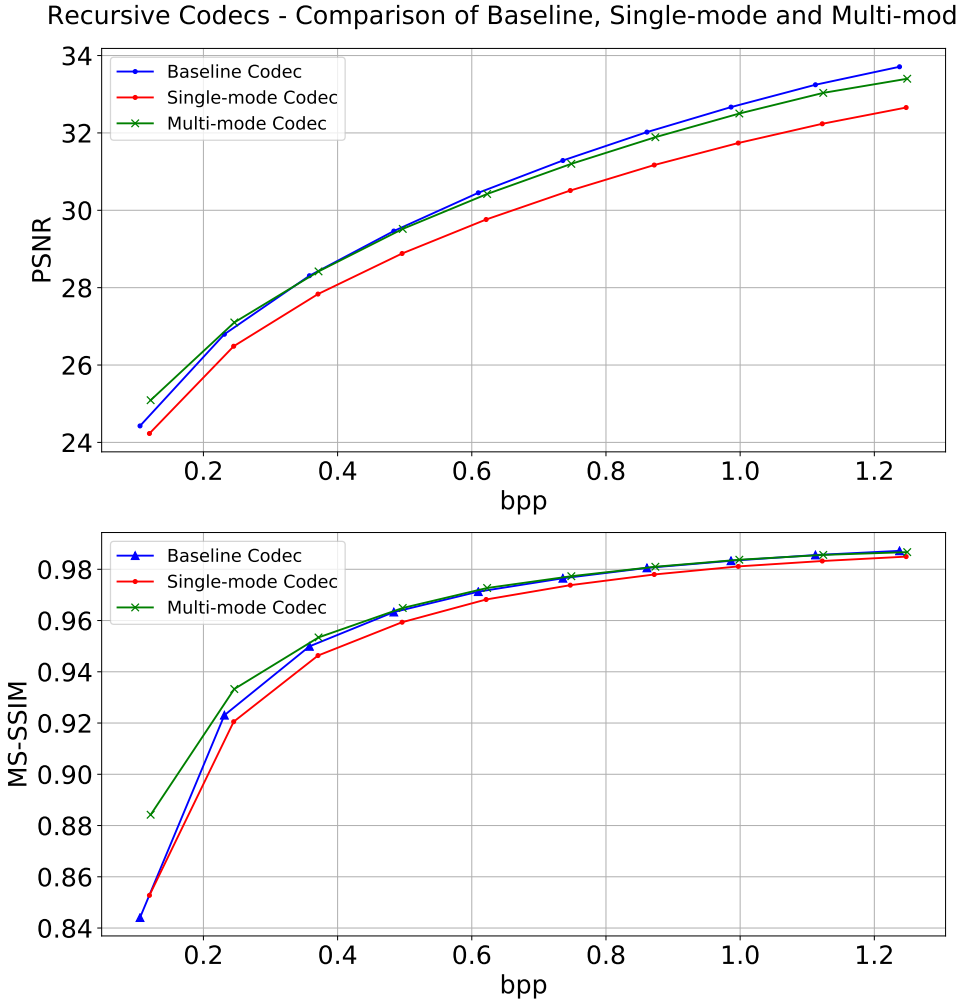

Figure 5.6: Comparison between Baseline, Single-mode and Multi-mode Recursive codecs, averaging the results obtained on the Kodak dataset. As we can see, the Single-mode Codec has the worst performance, and there is some overlap between the performance of the Baseline and Multi-mode Codec.

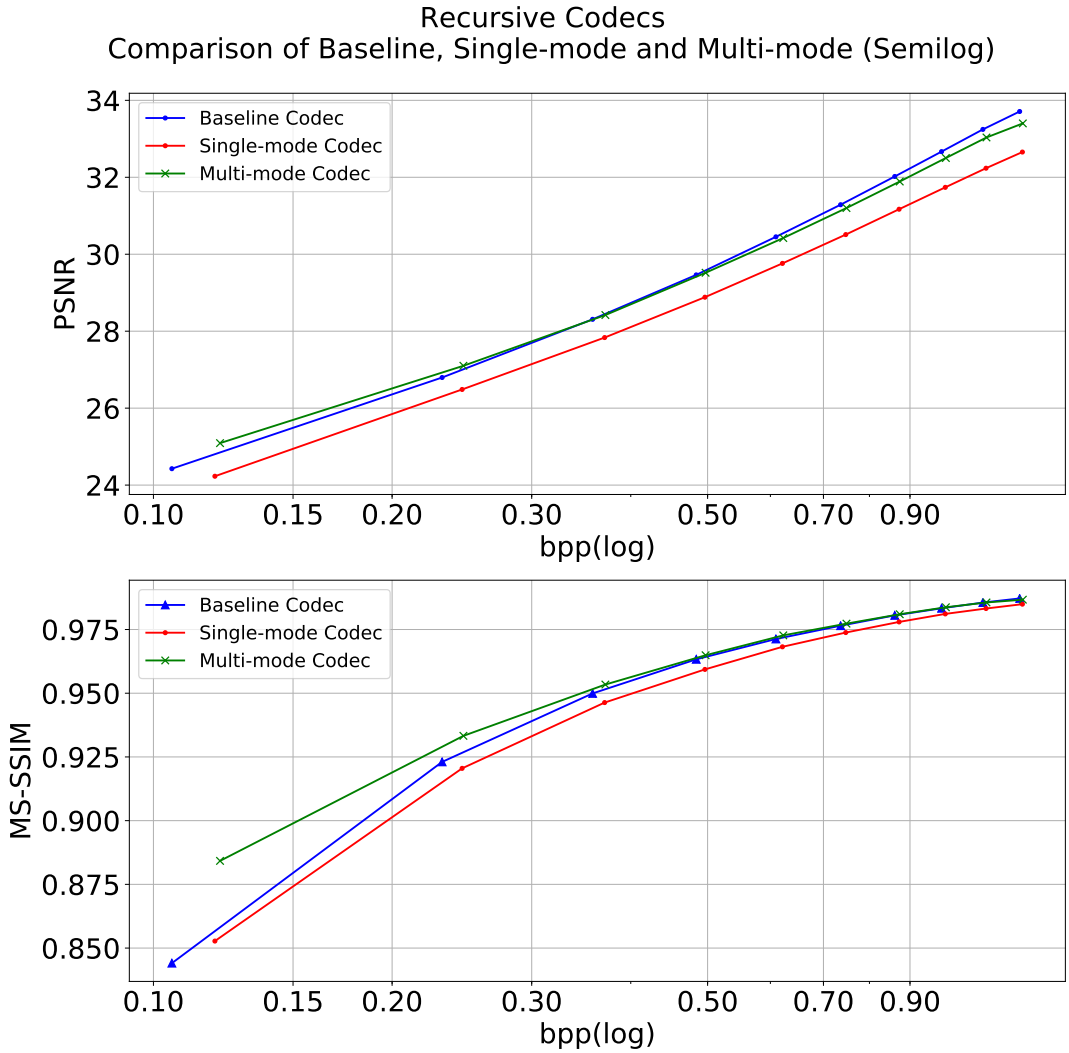

Figure 5.7: Comparison between Baseline, Single-mode and Multi-mode Recursive codecs, averaging the results obtained on the Kodak dataset. The semilog plot is used to show how the Multi-mode Codec performs better at low rates.

toencoders, one identical to the one used by our single-mode codec, and another with a similar architecture, but using an additional neighbouring block.

Additionally, we also use HEVC intra prediction modes, which were re-implemented in CUDA, in order to be generated during training. Finally, there is also a no-prediction mode, where no prediction is made, and the network acts like our baseline codec.

During training, this codec encodes the prediction that generates the residual with the least energy, but during tests, we encode all modes, and then pick the best mode as our final result.

As we can see from the plots in 5.6, if we compare the PSNR, the multi-mode codec has better performance at lower rates, but loses at higher rates. However, if we compare MS-SSIM, we see it is much better than the baseline at lower rates, and has similar performance at higher rates.

Another interesting analysis is to compare when each mode is chosen. As we can see in Figure 5.8, in simpler regions, the HEVC modes are chosen more frequently. Also, as we increase the general image quality, the number of times the HEVC modes are chosen increases as well. One possible explanation for this is that HEVC predictions are more sensit-ive to noise than the AE predictions, as they have smaller contexts, and therefore at lower rates the AE predictions are better.

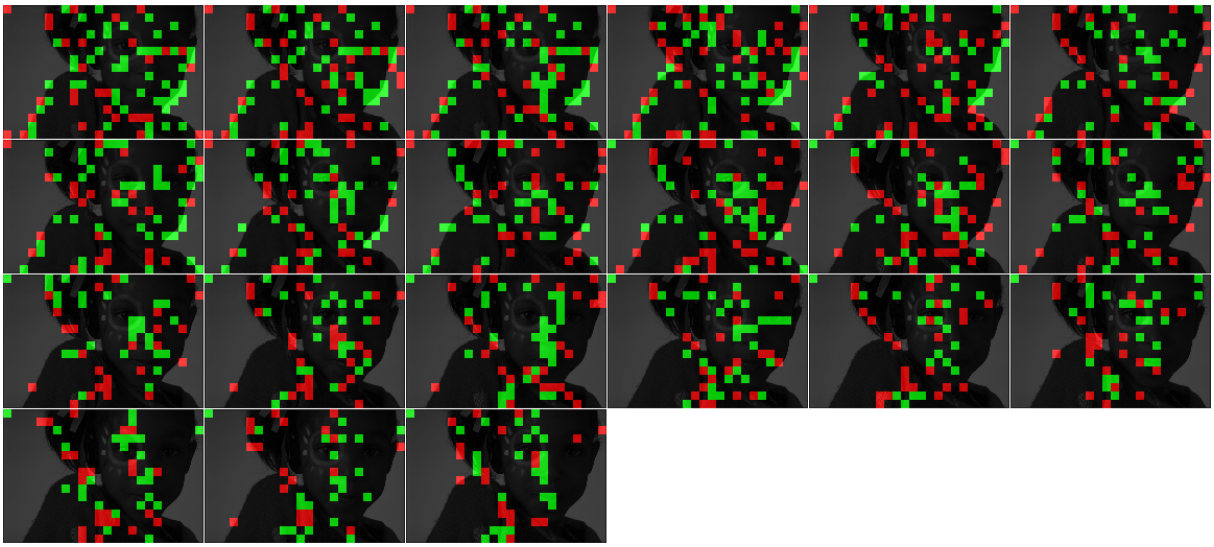

Figure 5.8: Distribution of the modes as we improve image quality. In red, the intra prediction mode inspired by [Minnen et al., 2017], in green our proposed mode with extended context, and in gray one of the HEVC prediction modes.

5.3.4 Using a bit-allocation algorithm

In the previous section, our recursive codecs are used so that all blocks are encoded for the same number of iterations. This approach is not so efficient, as there are many patches which are easy to encode, and require few iterations to be encoded at a high quality, while other patches are more complicated, and even using several iterations of the codec will not generate good results.

To deal with this, we use a bit allocation algorithm that encodes some of the patches using more iterations than others [Jung et al., 2020]. We define a set number of MSE thresholds, and attempt to encode a patch at the needed threshold. If the patch can't reach the quality level, we check to see whether or not it is within a δ ratio of the threshold.

If it is within this ratio, it means that the patch can still improve, and so we encode the patch again. If it is beneath this ratio, it means that the patch is probably saturating at its maximum quality, and so we stop encoding this patch.

We used this algorithm in our baseline codec and in our multi-mode codec. We heuristically set the MSE thresholds as [818, 650, 516, 410, 325, 258, 205, 163, 129, 103, 81, 65, 51, 41, 32, 25, 20, 16, 10, 6, 3], and the δ parameter that defines whether or not the patch is saturating as 0.7.

If we look at the results, we see that the baseline codec with bit-allocation has better PSNR results at higher rates, but at lower rates the Multi-mode codec with bit-allocation has better results. MS-SSIM wise, the Baseline Codec with bit-allocation has the worst performance of all, and that the Multi-mode codec with bit-allocation has almost the same performance as the Multi-mode codec without bit-allocation.

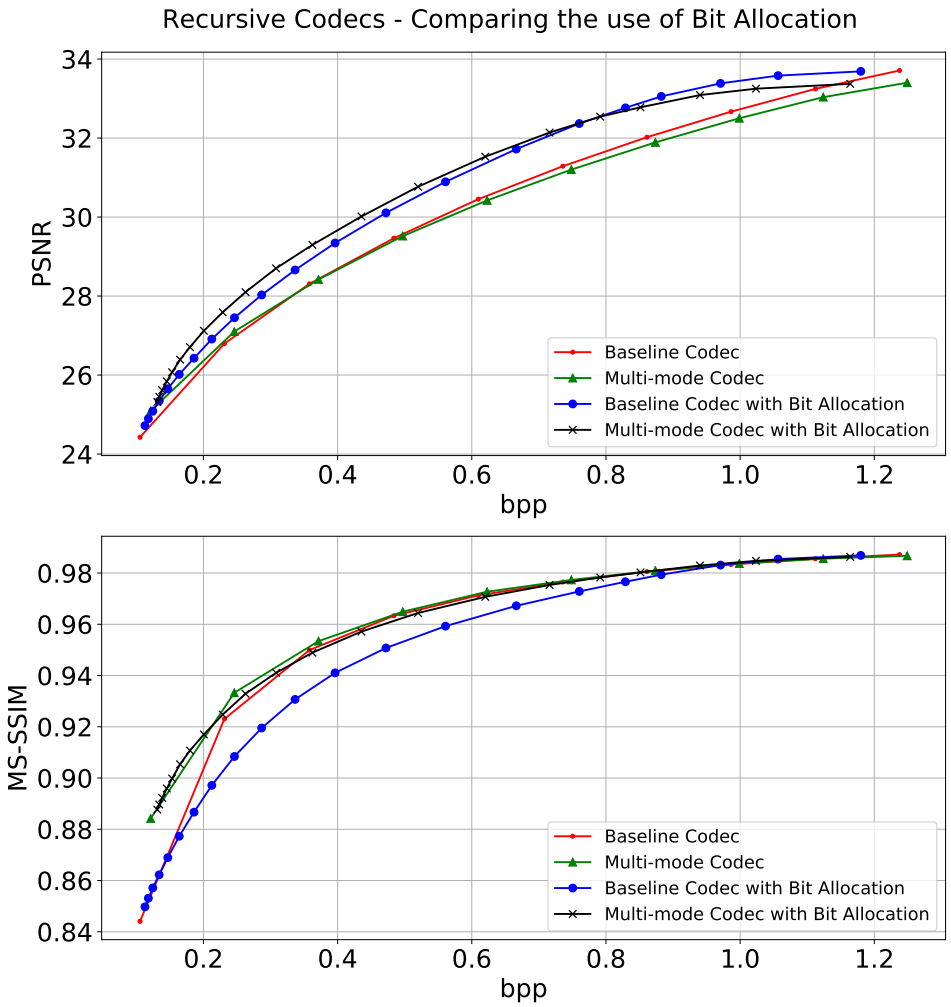

Figure 5.9: Comparison between Baseline and Multi-mode codec and their Bit-allocation counterparts. We can see that using Bit-allocation greatly improves PSNR results, but as it is focused on PSNR thresholds, there is an impact on MS-SSIM for the Baseline Codec. The MS-SSIM results for the Multi-mode codec with Bit-allocation do not have this effect.

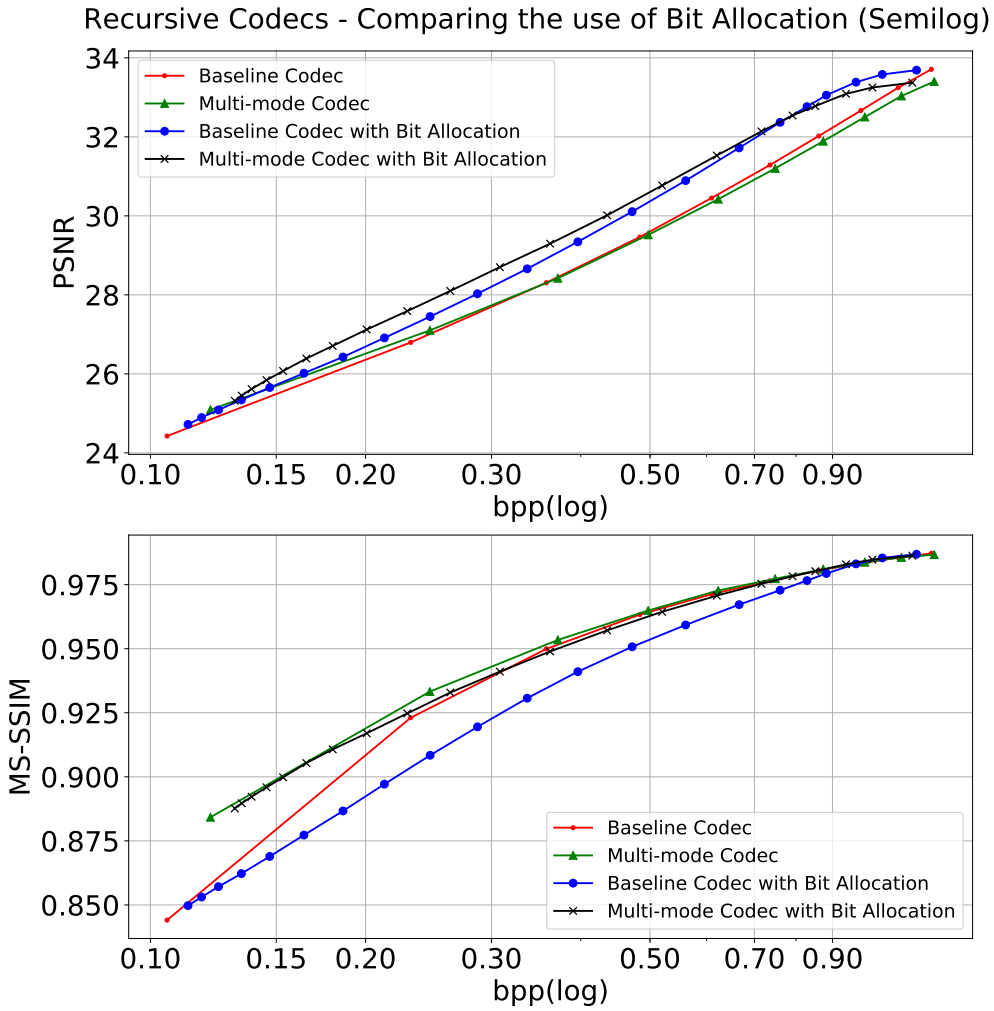

Figure 5.10: Comparison between Baseline and Multi-mode codec and their Bit-allocation counterparts. We use a semilog plot to emphasize the lower rate points.

5.4 Variational Based Codecs

In this section, we describe the codecs we developed based on Variational Autoencoders. We first describe the baseline VAE codec we used, and then the two intra prediction codecs we tested based on this approach, one using a pre-trained intra prediction network, and another with some layers attempting to do intra prediction being trained together with the codec.

5.4.1 Baseline VAE Codec

Our baseline codec is based on the codec described in [Ballé et al., 2016]. The main change in this codec is a difference in the number of strides made by the first convolutional layer, which was reduced to 2 from 4. This was necessary to allow us to deal with smaller patch sizes. We trained seven models, indexed by lambdas [0.1, 0.5, 0.01, 0.05, 0.001, 0.005, 0.0001].

The first analysis we did with this codec was to evaluate its performance along training. Like our recursive codec, we trained the codec for a large number of iterations, and evaluated if the codec was improving its performance, or if it had already saturated given the training constraints given. As we can see from Figure 5.11, at around 1 million iterations, there is not much improvement anymore, so we can safely train our codecs up to that point.

The second analysis we did was how the codec performed if we switched to a patch size approach. As this architecture is fully convolutional, it generates better results by compressing the whole image at once (as one giant patch). However, we wanted to see how it would perform if patches of size 32, 64, and 128 were used.

Therefore, we have trained models for each of these patch sizes, and also tested how the codecs trained for one patch size perform if they are used to compress images using a different patch size. Except for the lower rate points trained and tested using patch sizes of 32, there is not much difference between the performance of the codecs if we use different patch sizes at training and testing. On Figure 5.12 we show the average results obtained by compressing the images from the Kodak dataset.

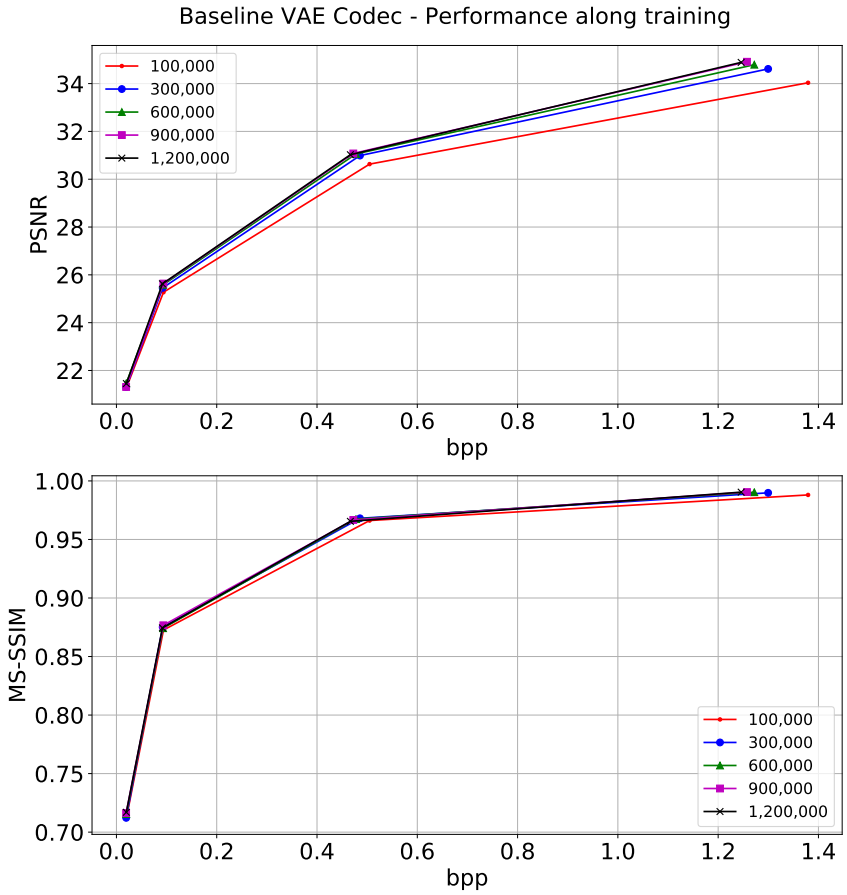

Figure 5.11: PSNR and MS-SSIM performance of the Baseline VAE codec along training

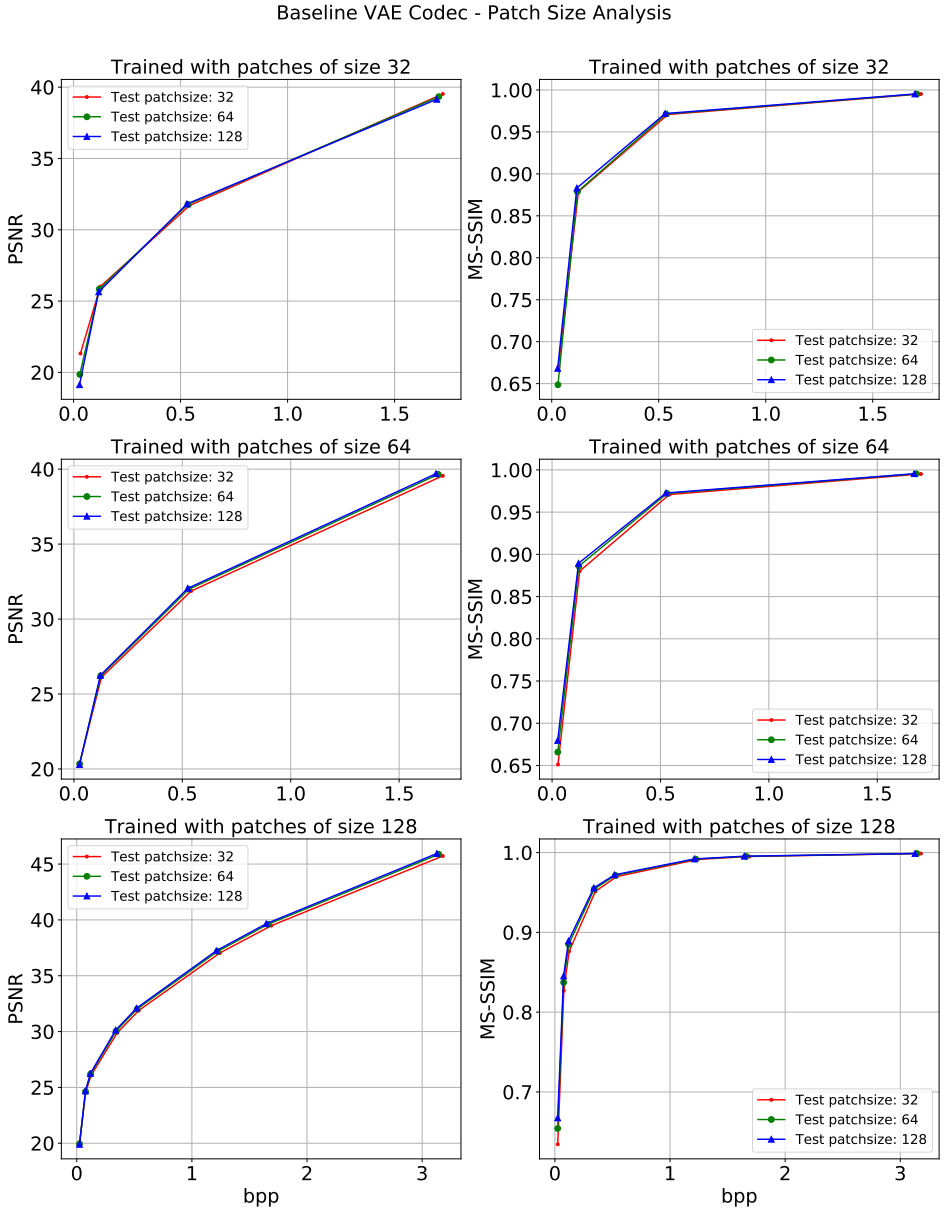

Figure 5.12: PSNR and MS-SSIM performance of the Baseline VAE codec as we change the training patch size and the testing patch size, mean results on Kodak dataset

5.4.2 VAE Codec with pre-trained Intra Prediction Autoencoder

To train our VAE codec with pre-trained autoencoder, we were limited to using patch sizes of size 32, as our pre-trained intra predictor model is not fully convolutional, and can only work with patches of this size. We have trained with contexts of size 64×64, so that the encoded patch size would be of size 32×32. In Figure 5.14, we show how the same patch is encoded through the different rates.

In the plots on Figure 5.13, we show the performance of this codec. As we can see, the point indexed by the lambda 0.0001, which would be expected to be the point with the lowest rate, has a very high rate. Our hypothesis is that the model is collapsing, and is unable to work at such low rates. During training, the intra prediction network receives uncompressed patches, but during the real use of the codec, the rate is so low that the generated images are of very poor quality, creating a snowball effect, as the next patches to be encoded don't receive a good prediction.

On the plots that compare all the codecs, we have chosen not to show this point, as it would not be used in a practical codec.

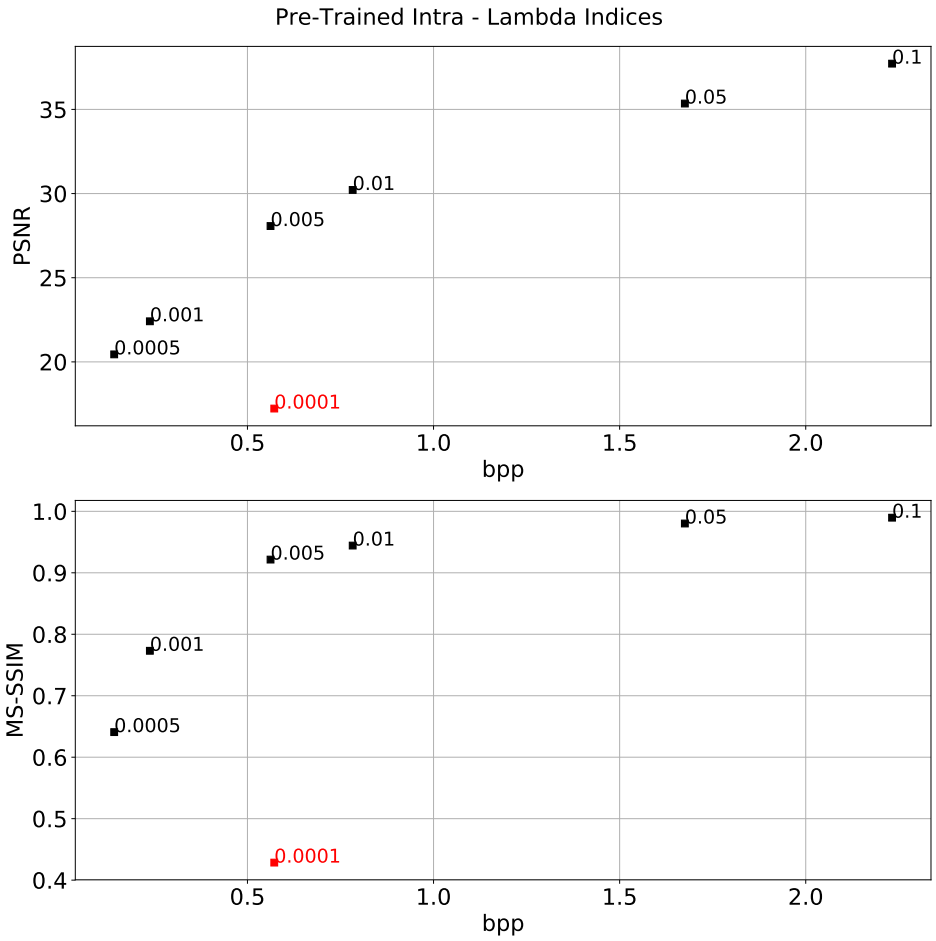

Figure 5.13: On this plot we show the mean PSNR and mean MS-SSIM performance on the Kodak dataset of the VAE Codec with Pre-Trained Intra Model. Next to each point, we show the lambda index used to train the model, and for which it was optimized. In red, we show the point for the lambda 0.0001, which was expected to be the leftmost point on the plot.

Figure 5.14: Examples of patches encoded by our VAE codec with pre-trained intra network

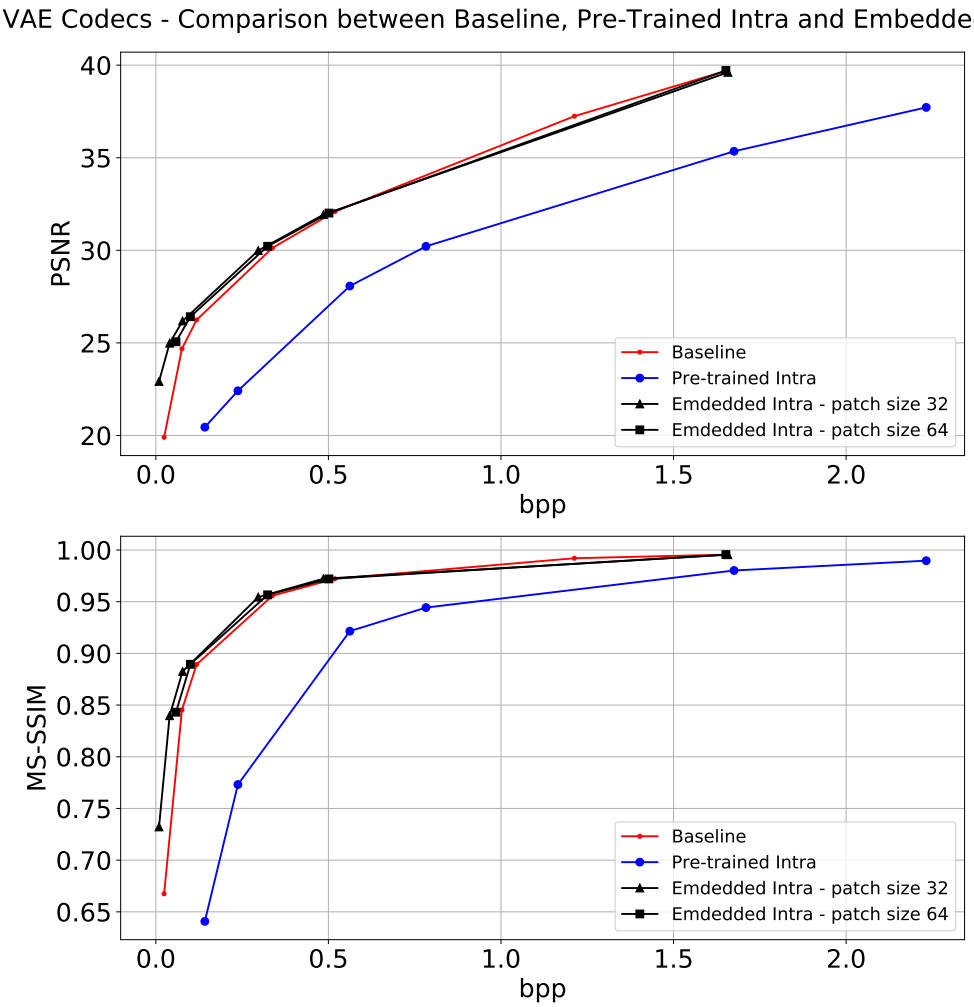

Figure 5.15: Mean PSNR and MS-SSIM performance on the Kodak dataset of our VAE codecs. As we can see, the VAE codec with Pre-Trained Intra has the worst results, but there is an overlap between the Baseline VAE codec and the Embedded Intra Codec. We remind the reader that the points indexed by the lambda 0.0001 for the intra models are not shown here, as they do not have a very good performance.

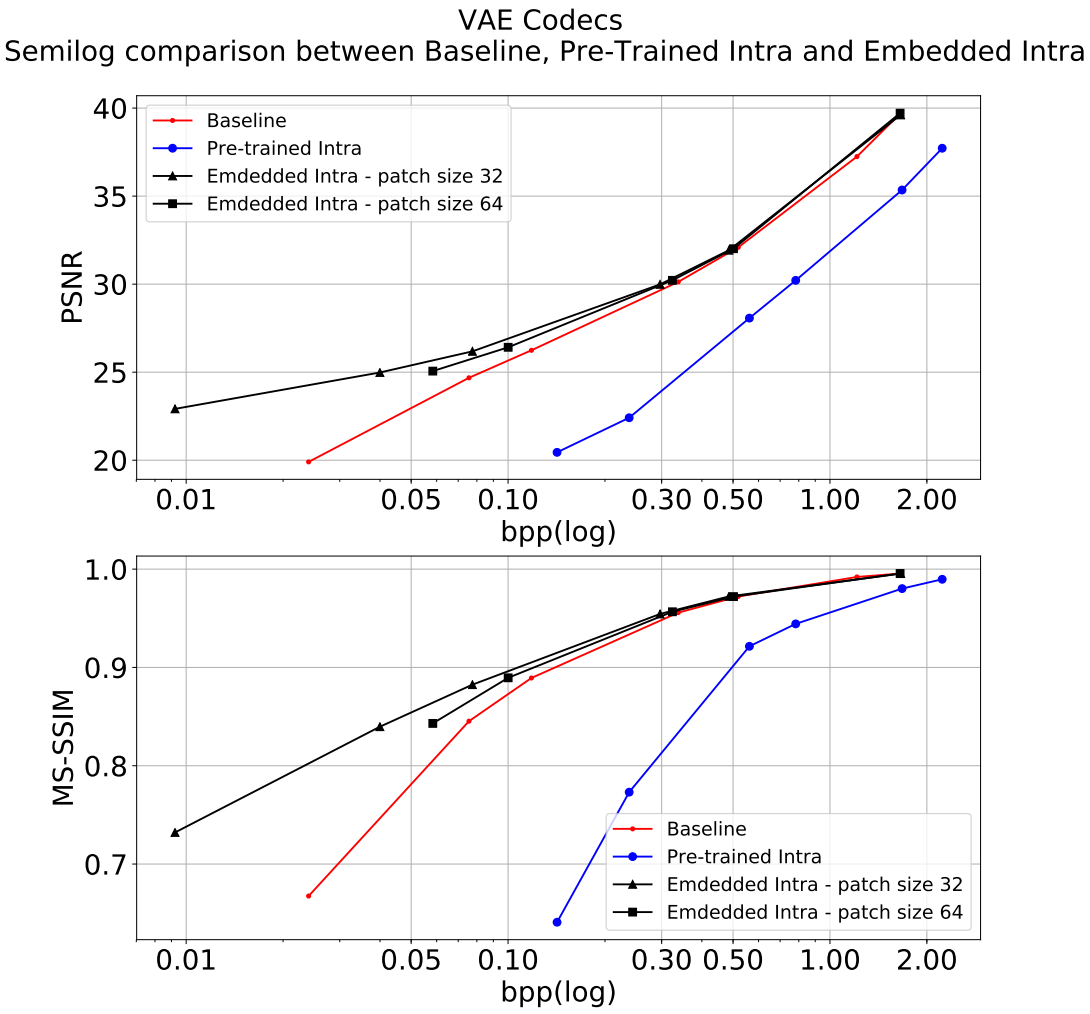

Figure 5.16: Mean PSNR and MS-SSIM performance on the Kodak dataset of our VAE codecs. Here we use a semilog plot to show that at very low rates, using an intra prediction gives better results than not using it. We remind the reader that the points indexed by the lambda 0.0001 for the intra models are not shown here, as they do not have a very good performance.

Figure 5.17: Examples of patches encoded by our VAE codec with embedded intra network - patch size of 32

5.4.3 VAE Codec with embedded Intra Prediction

As happened with other models, our VAE Codec with embedded Intra Prediction was trained for seven lambda values. We were not restricted to a fixed patch size during training because of the network architecture, but as we wanted to use the same dataset used in all previous experiments, which was created with patches of size 256×256, we were restricted in the size of the patches used during training.

Therefore, we have trained only two different versions of the codec, one using patches of size 32, and other using patches of size 64 (considering the context, the patches used were of size 64×96 and 128×192). In the end, we were left with fourteen trained models.

Much like what happened with the lowest rate point from our VAE codec with pre-trained intra, the lowest rate point of the codec trained with patches of size 64 also has bad performance (this point is excluded from the plot). The problem here is that the codec starts by generating bad predictions, and as the patch size is large, it is unable to move back into a range where the codec can work well. The predictions for patch size 32 start bad, but as there are more patches, with time the codec can restructure the image back into a working range.

Figure 5.18: Examples of patches encoded by our VAE codec with embedded intra network - patch size of 64

5.5 Comparing Recursive and Variational Codecs

In this section we compare both the Recursive and Variational approach. On Figure 5.19, we compare the best results from our approaches. For the recursive codecs, we compare the bit allocation versions of the baseline and the multi-mode codec, and for the VAE codecs we compare the results of the baseline VAE codec and the codec with an embedded intra prediction. Additionally, we compare the curves for both JPEG and JPEG 2000 codecs.

For the recursive codecs, we can see that they are better than the JPEG codec, but they are worse than the JPEG 2000 codec considering PSNR. For MS-SSIM, the Multi-mode codec with bit allocation has results on par with JPEG 2000 and the VAE codec with embedded intra prediction.

Considering the VAE codecs, the baseline codec and the embedded intra codec perform well across the entire range, with better performance than the JPEG 2000 codec. The exception here is the very low range, under 0.1 bpp. The baseline VAE codec has very bad performance, but the Embedded Intra codec has the best results.

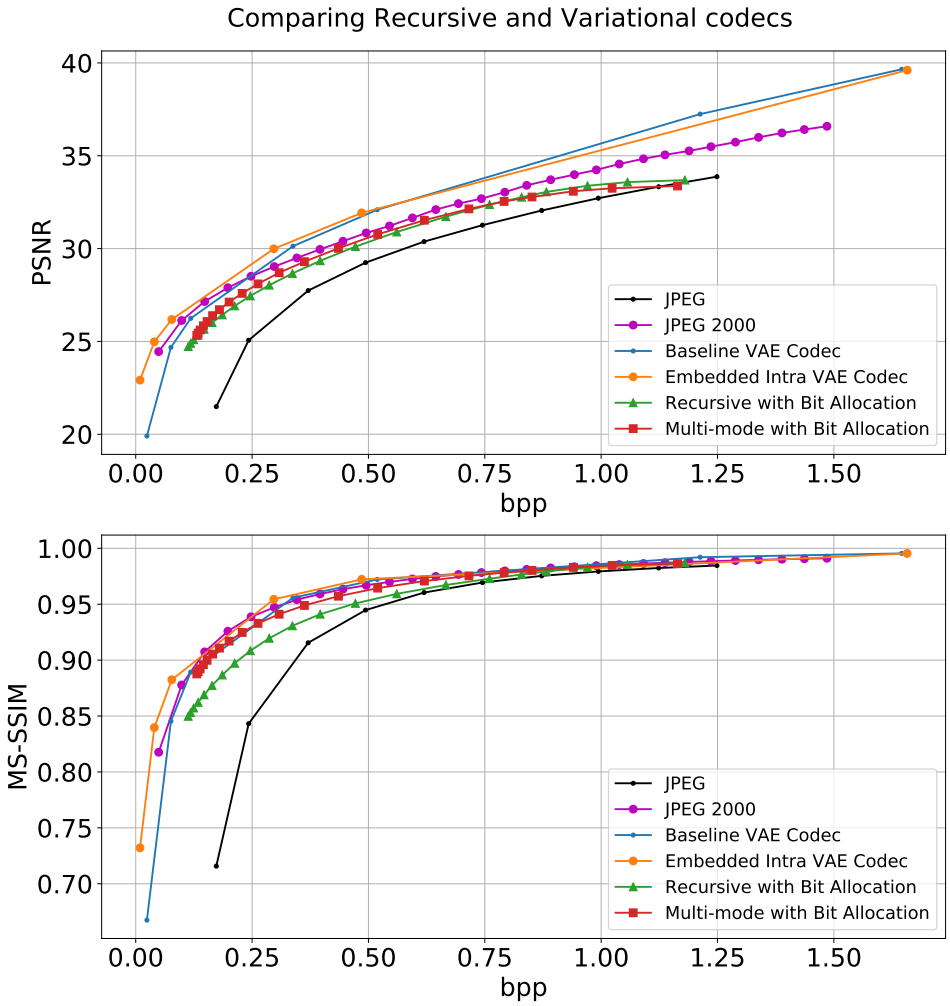

Figure 5.19: Comparing the different codecs, using average results on Kodak dataset. PSNR is taken considering luma values

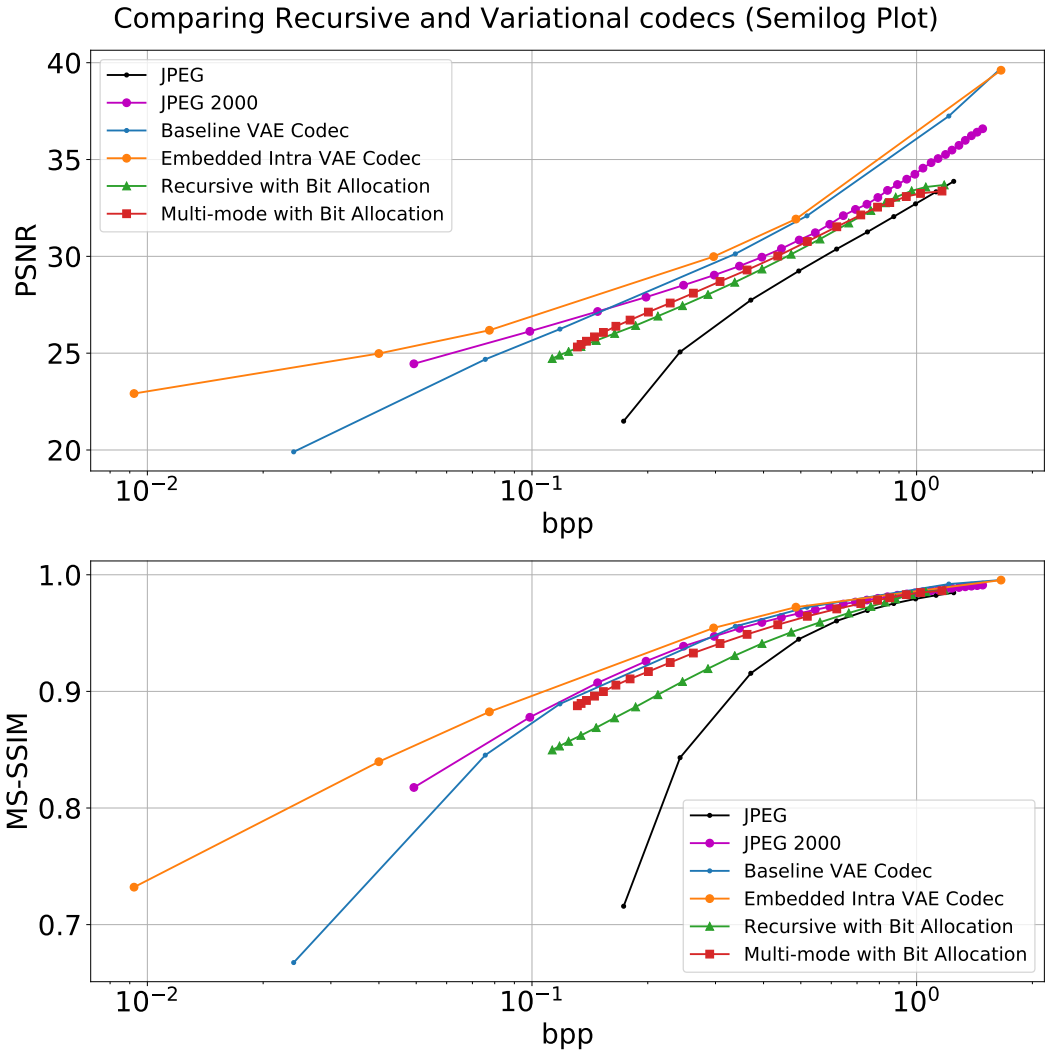

Figure 5.20: Comparing the different codecs, using average results on Kodak dataset.

Chapter 6

Conclusions

On this chapter, we first present our conclusions taken from the results of our experiments, and then present some possible improvements to them.

6.1 Conclusions regarding Intra prediction and Neural Networks

Looking at the results we have obtained, we can see that using intra prediction is a viable option for learned image codecs. Although it requires switching to a patch based approach, we can see there is some improvement in the results obtained, specially if we consider the MS-SSIM curves. Our results indicate that this could be a viable option, but needs to be further explored.

However, if we look at the recursive codecs, using intra prediction makes them too slow to be of any use, and it is hard to defend the cost-benefit of using it. Now, for the VAE codecs, using intra prediction does not cause any significant change in the time taken, and so it remains useful. And, as we have mentioned, encoding a series of smaller patches requires less powerful computers, which needs to be taken into account if we want phone users to use one of these codecs in the future.

6.2 Future Works

The first steps in extending our experiments would be modify our VAE codecs to introduce the hyperprior modellings taken from [Ballé et al., 2018], or possibly extend that even further to use Auto-regressive modelling on the codes. These models give much better results, but adapting them to patch size approaches is harder than adapting the models

we used. However, as these models are leading the way in learned image coding, this seems worth the effort.

Another problem to be explored is the moving target problem for the low-rate intra prediction codecs. One possible solution would be to first train the codec using raw images, and them train the codec again using the images already compressed by the codec. This feedback approach during training is not simple, but perhaps could generate more realistic points on the curve.

Considering our recursive codecs, the main concern is finding ways to speed up the coding process. Currently, intra prediction approaches are very slow, and are not viable as they are right now. One could possibly speed up encoding by generating intra predictions based on the original image, and ignore drifting effects, but even this would not solve how to speed up decoding.

Another simpler improvement to our recursive codec would be to change its structure to deal with different patch sizes as input. As we know from H.264 an HEVC codecs, using different patch sizes on easier and harder patches can improve performance.

Bibliography

[Agustsson and Timofte, 2017] Agustsson, E. and Timofte, R. (2017). Ntire 2017 challenge on single image super-resolution: Dataset and study. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 44

[Ahmed et al., 1974] Ahmed, N., Natarajan, T., and Rao, K. R. (1974). Discrete cosine transform. *IEEE transactions on Computers*, 100(1):90–93. 5

[Akbari et al., 2020] Akbari, M., Liang, J., Han, J., and Tu, C. (2020). Generalized octave convolutions for learned multi-frequency image compression. *CoRR*, abs/2002.10032. 40

[Ballé et al., 2016] Ballé, J., Laparra, V., and Simoncelli, E. P. (2016). Density modeling of images using a generalized normalization transformation. In Bengio, Y. and LeCun, Y., editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. 31

[Ballé et al., 2016] Ballé, J., Laparra, V., and Simoncelli, E. P. (2016). End-to-end optimization of nonlinear transform codes for perceptual quality. In *2016 Picture Coding Symposium (PCS)*, pages 1–5. IEEE. 26, 30

[Ballé et al., 2016] Ballé, J., Laparra, V., and Simoncelli, E. P. (2016). End-to-end optimized image compression. *CoRR*, abs/1611.01704. vii, viii, 2, 23, 31, 32, 39, 40, 57

[Ballé et al., 2018] Ballé, J., Minnen, D., Singh, S., Hwang, S. J., and Johnston, N. (2018). Variational image compression with a scale hyperprior. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. viii, x, 32, 70

[Barnes et al., 2009] Barnes, C., Shechtman, E., Finkelstein, A., and Goldman, D. B. (2009). Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics (ToG)*, 28(3):24. 25

[Chen and Gopinath, 2001] Chen, S. S. and Gopinath, R. A. (2001). Gaussianization. In *Advances in neural information processing systems*, pages 423–429. 31

[Cho et al., 2014] Cho, K., van Merrienboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078. 20, 29

[Choi et al., 2019] Choi, Y., El-Khamy, M., and Lee, J. (2019). Variable rate deep image compression with a conditional autoencoder. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3146–3154. 32, 40

[Chollet, 2017] Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258. 19

[Cisco, 2020] Cisco (2020). Cisco Annual Internet Report (2018–2023) White Paper. `https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html`. [Online]. 1

[Clevert et al., 2016] Clevert, D., Unterthiner, T., and Hochreiter, S. (2016). Fast and accurate deep network learning by exponential linear units (elus). In Bengio, Y. and LeCun, Y., editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. 15

[Covell et al., 2017] Covell, M., Johnston, N., Minnen, D., Hwang, S. J., Shor, J., Singh, S., Vincent, D., and Toderici, G. (2017). Target-quality image compression with recurrent, convolutional neural networks. *CoRR*, abs/1705.06687. 30

[Cover, 1999] Cover, T. M. (1999). *Elements of information theory*. John Wiley & Sons. 3, 30

[Danihelka et al., 2016] Danihelka, I., Wayne, G., Uria, B., Kalchbrenner, N., and Graves, A. (2016). Associative long short-term memory. In Balcan, M. and Weinberger, K. Q., editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1986–1994. JMLR.org. 29

[Darabi et al., 2012] Darabi, S., Shechtman, E., Barnes, C., Goldman, D. B., and Sen, P. (2012). Image melding: combining inconsistent images using patch-based synthesis. *ACM Transactions on Graphics (ToG)*, 31(4):1–10. 25

[Daubechies, 1990] Daubechies, I. (1990). The wavelet transform, time-frequency localization and signal analysis. *IEEE transactions on information theory*, 36(5):961–1005. 5

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`. 13, 22

[Graves, 2013] Graves, A. (2013). Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850. 20, 29

[Gray and Neuhoff, 1998] Gray, R. M. and Neuhoff, D. L. (1998). Quantization. *IEEE transactions on information theory*, 44(6):2325–2383. 23

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778. 32

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780. 20, 29

[Iizuka et al., 2017] Iizuka, S., Simo-Serra, E., and Ishikawa, H. (2017). Globally and locally consistent image completion. *ACM Transactions on Graphics (ToG)*, 36(4):1–14. vii, 25, 26

[ITU, 1996] ITU (1996). P.800 : Methods for subjective determination of transmission quality. 10

[Jin et al., 2016] Jin, L., Lin, J. Y., Hu, S., Wang, H., Wang, P., Katsavounidis, I., Aaron, A., and Kuo, C.-C. J. (2016). Statistical study on perceived jpeg image quality via mcl-jci dataset construction and analysis. *Electronic Imaging*, 2016(13):1–9. 44

[Johnston et al., 2018] Johnston, N., Vincent, D., Minnen, D., Covell, M., Singh, S., Chinen, T., Jin Hwang, S., Shor, J., and Toderici, G. (2018). Improved lossy image compression with priming and spatially adaptive bit rates for recurrent networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4385–4393. vii, 29

[Jung et al., 2020] Jung, H. C., Guerin Jr, N. D., Ramos, R. S., Macchiavello, B., Peixoto, E., Hung, E. M., Campos, T., Silva, R. C., Testoni, V., and Freitas, P. G. (2020). Multi-mode intra prediction for Learning-based Image Compression. In *International Conference on Image Processing (ICIP)*. vii, viii, 26, 33, 34, 54

[Kingma and Welling, 2014] Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. In Bengio, Y. and LeCun, Y., editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. vii, 30

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105. 15

[Lainema et al., 2012] Lainema, J., Bossen, F., Han, W.-J., Min, J., and Ugur, K. (2012). Intra coding of the hevc standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1792–1801. 39

[Laparra et al., 2009] Laparra, V., Camps-Valls, G., and Malo, J. (2009). Pca gaussianization for image processing. In *2009 16th IEEE International Conference on Image Processing (ICIP)*, pages 3985–3988. IEEE. 32

[Lee et al., 2019] Lee, J., Cho, S., and Kim, M. (2019). An end-to-end joint learning scheme of image compression and quality enhancement with improved entropy minimization. 40

[Li et al., 2020] Li, X., Sun, S., Zhang, Z., and Chen, Z. (2020). Multi-scale grouped dense network for vvc intra coding. In *3rd Challenge on Learned Image Compression*. 34

[Lim et al., 2017] Lim, B., Son, S., Kim, H., Nah, S., and Lee, K. M. (2017). Enhanced deep residual networks for single image super-resolution. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 44

[Marpe et al., 2003] Marpe, D., Schwarz, H., and Wiegand, T. (2003). Context-based adaptive binary arithmetic coding in the h.264/avc video compression standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):620–636. 6

[Martín, 1979] Martín, G. (1979). Range encoding: an algorithm for removing redundancy from a digitised message. In *Video and Data Recording Conference, Southampton, 1979*, pages 24–27. 4

[Minnen et al., 2017] Minnen, D., Toderici, G., Covell, M., Chinen, T., Johnston, N., Shor, J., Hwang, S. J., Vincent, D., and Singh, S. (2017). Spatially adaptive image compression using a tiled deep network. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 2796–2800. IEEE. vii, viii, xv, xvi, xvii, 26, 30, 34, 36, 37, 39, 42, 53

[Nemoto et al., 2014] Nemoto, H., Hanhart, P., Korshunov, P., and Ebrahimi, T. (2014). Ultra-eye: Uhd and hd images eye tracking dataset. In *2014 Sixth International Workshop on Quality of Multimedia Experience (QoMEX)*, pages 39–40. IEEE. 44

[Pathak et al., 2016] Pathak, D., Krahenbuhl, P., Donahue, J., Darrell, T., and Efros, A. A. (2016). Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2536–2544. vii, 25, 34

[Rabbani and Joshi, 2002] Rabbani, M. and Joshi, R. (2002). An overview of the jpeg 2000 still image compression standard. *Signal processing: Image communication*, 17(1):3–48. 5

[Raiko et al., 2015] Raiko, T., Berglund, M., Alain, G., and Dinh, L. (2015). Techniques for learning binary stochastic feedforward neural networks. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 23

[Rissanen and Langdon, 1979] Rissanen, J. and Langdon, G. G. (1979). Arithmetic coding. *IBM Journal of research and development*, 23(2):149–162. 4

[Sayood, 2017] Sayood, K. (2017). *Introduction to data compression*. Morgan Kaufmann. 5, 7

[Shi et al., 2016] Shi, W., Caballero, J., Huszár, F., Totz, J., Aitken, A. P., Bishop, R., Rueckert, D., and Wang, Z. (2016). Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1874–1883. 21, 22, 32

[Sullivan et al., 2012] Sullivan, G. J., Ohm, J.-R., Han, W.-J., Wiegand, T., et al. (2012). Overview of the high efficiency video coding(hevc) standard. *IEEE Transactions on circuits and systems for video technology*, 22(12):1649–1668. 1, 5, 34

[Sun et al., 2020] Sun, H., Liu, C., Katto, J., and Fan, Y. (2020). An image compression framework with learning-based filter. In *3rd Challenge on Learned Image Compression*. 34

[Theis et al., 2017] Theis, L., Shi, W., Cunningham, A., and Huszár, F. (2017). Lossy image compression with compressive autoencoders. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. viii, 24, 32

[Toderici et al., 2016] Toderici, G., O'Malley, S. M., Hwang, S. J., Vincent, D., Minnen, D., Baluja, S., Covell, M., and Sukthankar, R. (2016). Variable rate image compression with recurrent neural networks. In Bengio, Y. and LeCun, Y., editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. vii, 2, 23, 26, 27

[Toderici et al., 2017] Toderici, G., Vincent, D., Johnston, N., Jin Hwang, S., Minnen, D., Shor, J., and Covell, M. (2017). Full resolution image compression with recurrent neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5306–5314. vii, viii, 27, 29, 36, 39, 48

[van den Oord et al., 2016] van den Oord, A., Kalchbrenner, N., and Kavukcuoglu, K. (2016). Pixel recurrent neural networks. In Balcan, M. and Weinberger, K. Q., editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1747–1756. JMLR.org. 27

[Wallace, 1992] Wallace, G. K. (1992). The jpeg still picture compression standard. *IEEE transactions on consumer electronics*, 38(1):xviii–xxxiv. 5

[Wang et al., 2004] Wang, Z., Bovik, A. C., Sheikh, H. R., Simoncelli, E. P., et al. (2004). Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612. 11

[Wang et al., 2003] Wang, Z., Simoncelli, E. P., and Bovik, A. C. (2003). Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, volume 2, pages 1398–1402. Ieee. 12

[Wiegand et al., 2003] Wiegand, T., Sullivan, G. J., Bjontegaard, G., and Luthra, A. (2003). Overview of the h. 264/avc video coding standard. *IEEE Transactions on circuits and systems for video technology*, 13(7):560–576. 1, 5, 34

[Xu et al., 2015] Xu, B., Wang, N., Chen, T., and Li, M. (2015). Empirical evaluation of rectified activations in convolutional network. *CoRR*, abs/1505.00853. 32

[Yu et al., 2018] Yu, J., Lin, Z., Yang, J., Shen, X., Lu, X., and Huang, T. S. (2018). Generative image inpainting with contextual attention. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5505–5514. vii, 26

[Zhou et al., 2019] Zhou, L., Sun, Z., Wu, X., and Wu, J. (2019). End-to-end optimized image compression with attention mechanism. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 40

[Ziv and Lempel, 1977] Ziv, J. and Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE Transactions on information theory*, 23(3):337–343. 30