Universitat Politècnica de Catalunya
Facultat d'Informàtica de Barcelona
Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona
Facultat de Matemàtiques i Estadística

Degree in Data Science and Engineering

Bachelor's Degree Thesis

Deep Colorization networks for image compression

Guillem Borràs Sans

Supervised by Javier Ruiz Hidalgo

30th June, 2021

Abstract

Considering the billions of pictures stored world wide and the rising of streaming services, image compression is, nowadays, a very important field. In this TFG we propose a lossless framework to compress images that takes advantage of the color redundancy when the luminance information is known. We extract the image chrominance information via the LAB transform. Using an autoencoder approach we reduce the chrominance information dimensionality obtaining a color feature vector. We encode, apart from the feature color vector, a decolorized image which only needs the extracted color information to perfectly reconstruct the original image. The decolorized image is less expensive to encode than the original image thanks to the energy reduction. This decolorized image can be encoded using any lossless classical approach.

The framework proposed can outperform PNG by a 5% improvement in compression rates across different image sizes when compressing the decolorized image with PNG.

Abstract

Avui en dia, la compressió d'imatge és un camp important tenint en compte que s'emmagatzemen milers de milions d'imatges arreu del món i l'auge dels serveis en *streaming*. En aquest TFG, proposem un algorisme de compressió sense pèrdues que aprofita la redundància de color un cop la informació de il·luminació és sabuda. Obtenim la crominància de les imatges a codificar mitjançant la transformació de color LAB. Usant un *autoencoder*, reduïm la dimensionalitat d'aquesta crominància obtenint un vector que representa la informació de color de la imatge. Codifiquem, a part d'aquest propi vector, la imatge original descolorida la qual únicament necessita la informació del vector de color per reconstruir perfectament la imatge original. La imatge descolorida serà menys costosa de codificar ja que tindrà menys energia. A més, la podrem codificar amb qualsevol estàndar de compressió sense pèrdues.

Hem aconseguit millorar els ràtios de compressió de l'estàndar PNG per a diferents tamanys d'imatge i quan comprimíem la imatge descolorida amb PNG.

Abstract

Hoy en día, la compresión de imagen es un campo importante teniendo en cuenta que se almacenan miles de millones de imágenes en todo el mundo y el auge de los servicios de *streaming*. En este TFG, proponemos un algoritmo de compresión sin pérdidas que aprovecha la redundancia de color una vez la información de la iluminación es sabida. Obtenemos la crominancia de las imágenes a codificar mediante la transformada LAB. Usando un *autoencoder*, reducimos la dimensionalidad de esta crominancia obteniendo un vector que representa la información de color de la imagen. Codificamos, a parte de este propio vector, la imagen original descolorida la cual solo necesitará la información del vector de color para reconstruir perfectamente la imagen original. La imagen descolorida será menos costosa de codificar ya que tendrá menos energía. Además, la podremos codificar con cualquier estándar de codificación sin pérdidas.

Hemos conseguido mejorar los ratios de compresión del estándar PNG en un 5% con diferentes tamaños de imagen y cuando comprimíamos la imagen descolorida usando PNG.

Contents

1. Introduction

Nowadays, image compression is an important field that plays a huge part in our daily use of technology. The billions of pictures in social media or the millions of hours of video prepared to be watched on stream need to be stored in the most efficient possible way in order to not saturate our disks capacity.

However, the most important task of image compression and the one that is pooling the most advances is image transmission via a certain channel, and most specifically via the Internet.

Suppose we have a limited bandwidth, which is an scenario that currently happens quite often. If we wanted to transmit images to a certain rate we would need them to weight as minimum as possible in order to meet the rate requirements. Also, at a set rate, the better we compress our images, the more quality we can have in our transmitted images. Most of the time, the frame rate is set, for example in Europe the frame rate in cinema is 24 frames per second. The image quality is the huge factor that demands our images and video to be as light as possible. No one wants to watch a movie at a 144p resolution!

In this TFG we will create an image compression algorithm that tries to minimize the bitrate of compressed images. The main characteristic behind the compression that we will try to exploit is image color redundancy. We came up with this colorization approach because, in a few courses, we have seen examples of automatic colorization of black and white images. So we wondered: how much information does the illuminance of images have?

For a human is quite easy to color a black and white image: sky is blue, grass is green and so on. But sometimes we, humans, even need some sort of color information: an apple can be red or green. So by only having little information about color of a black and white image we could successfully color most of the objects that appear in it. Could a machine learn to colorize images with the luminance component and little color information?

If so, we could compress RGB images by only compressing classically the luminance and transmitting through the channel just the least amount of color information necessary to color the black and white pictures. The decoder, with the luminance part of the image plus the color information, would be able to perfectly reconstruct the images. This color feature extraction will be performed by an autoencoder which will be explained in *section 2.1*.

1.1 Objectives

The principal objectives we aim to achieve are:

- Create a lossless compression framework that exploits color redundancy and can improve lossless standards like PNG.
- Analyze the framework's behaviour with different image sizes.
- Provide its strengths and weaknesses.
- Study and analyze the compression of all the elements needed to be sent trough the channel.

1.2 Personal Motivation

I got involved in the image compression field during my Data Science Bachelor's degree in the Image Processing course which later led me to pick Audiovisual Coding Content as an elective course. In this course, I was really amazed by the genius tricks and algorithms that are used in image compressing and also the different approaches that an engineer could take depending on the rate-distortion requirements. The interest in this field and the opportunity to also learn the state of art made me decide to finally focus my TFG around image compression.

The idea of using Deep Learning in my TFG was something that I wanted to do in the very beginning even before knowing that I would work on image compression. I learned a lot about Deep Learning in many courses and I was stunned by the many applications it had and how well these Neural Networks solved many problems: it was really natural to merge these image encoding problem with some Deep Learning techniques. Also, I thought that developing a Deep Learning project would be very beneficial for gaining some experience on the topic.

2. Problem Definition

We decided to create a lossless compression framework, that given an RGB image returned a compressed image with the necessary information to perfectly be decoded. The reason why we decided to go for a lossless compression framework is to avoid subjectivity in determining which parameters are better than others when some information is lost. It is true that there exist rate distortion metrics but sometimes larger distortions do not imply better visual perception. Moreover, evaluating the algorithm in different rates would take extra work and also propose an schema that would accept different rates. This last part is something we will not entirely discard. We will try to propose a framework that at some point has mechanisms to lower the rate at the exchange of losing the minimum information.

We will assume that the channel will not present losses so no further error correction techniques will be needed nor take into account.

Once a framework is set and an algorithm is defined, the goal will be obviously to minimize the image weight as much as possible by tuning the different parameters that different parts of the pipeline will accept.

Consequently, let us define intuitively what will our framework structure and the main parts be as well as different parameters and implementation decisions that this parts will arise. Also, we will discuss how we would want the output of every part to be modeled to ultimately achieve the lowest image weight.

2.1 Color Feature Vector extraction

This part of the scheme will take as an input an RGB image and will output a feature vector with the maximum color information as we can in order to reconstruct the whole image.

The decoder will receive this color feature extraction and will reconstruct the image with these features, which represent its color information. The decoder will also need the luminance part, so the luminance will

be required to be transmitted to the channel too. We then want this feature vector to be also as light as possible because, as we said, this information will be needed to be transmitted through the channel and therefore compressed somehow.

As we reduce the size of the feature vector to make it weight less bytes, it is fair to assume that the decoder will not be able to perfectly reconstruct the original image from the feature vector: some error will be generated between the ground truth (original image) and the reconstructed image in the decoder's part. If we want to build a lossless algorithm, this error will also have to be sent to the decoder who will add it to the reconstruction of the image, using the color information, to obtain the exact same original image.

We find a clear tradeoff between the amount of information of this color feature vector (more information more bytes needed to encode it) and the energy of the error produced in the reconstruction: if less information is stored in the vector, more energy the error image will have and more bytes will be invested in encoding this error. The size of this vector is a key factor when it comes to the information encoding so we will need to carefully decide it, taking into account the above tradeoff.

The approach that we will implement in this project to extract this color features is through an autoencoder scheme (*Figure 1*). One could think that an autoencoder is composed by two different networks. The A-encoder that compresses the input until we reach a bottleneck and the A-decoder that decompresses the bottleneck tensor to reconstruct the ground truth input which in our case is the color information.

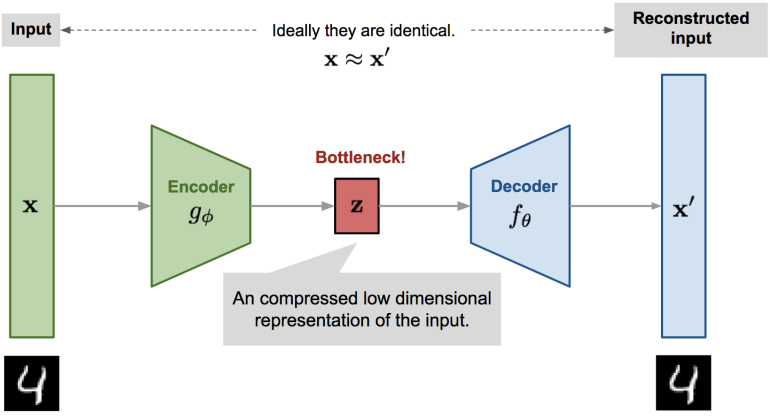

Figure 1: Autoencoder scheme. Figure credit to [1].

In the encoding phase, the A-encoder will take the color information of an image as an input; this colorization information can be obtained via a domain transform from the RGB space to a Luminance-Chrominance space for example. Then, it will reduce its dimensionality hoping that through this reduction the most relevant image color information will be kept. This color information compressed in the bottleneck will be send to the decoder. The encoder will also need to use the A-decoder output to reconstruct the image and compute the error produced that will later send to the decoder.

During the decoding part, the A-decoder will take the feature vector received as the input of the A-decoder: the bottleneck layer. After that, it will just forward the network from the bottleneck layer and will

be able to reproduce the exact same information that the encoder got as the A-encoder output because they will share the exact same values in the bottleneck layer. In other words, the image reconstructed by the decoder using the A-decoder's output will be the same as which the encoder used when computing the error. Because both images will be the same, the decoder will simply need to add the error (received from the channel) to his image reconstruction so that the original image information will perfectly be reconstructed.

The encoder will need to have at its end the A-encoder to generate the bottleneck and the A-decoder to generate the error. The decoder will need to have the A-decoder to generate the reconstructed color information.

We will evaluate the performance of compacting this color information by the capacity of the A-decoder to recreate the original input from the reconstructed color information and the luminance information. The more the reconstructed color information reassembles the original one, the better this reconstruction will be. The performance can also be evaluated by the complementary of having a good reconstruction: the less energy the generated error image will have, the better we will have compressed the color information in the feature vector.

There are many parameters that can affect to the minimization of the difference between the input and output images. Among them there is the network architecture and the common hyperparameters that one have to set when training a neural network: batch size, learning rate, etc. We will have to choose a loss function, which will take a huge roll in how the error will be modeled. Also, a very impactful decision to take is the size of the bottleneck of the autoencoder because, as we already said, the smaller it is the more energy the error will have and more expensive to compress.

2.2 Luminance compression

We should think how transmitting the luminance information through the channel should be defined because the decoder needs it to reconstruct the original image. Our own system can not be reused with single channel images so we will need to use classical ways of compressing it. Due to our lossless constraint, we will need to use an encoding algorithm that is also lossless. We will work with the PNG standard which is lossless and it is commonly used to compress single channel images . This standard and its characteristics will be explained in *section 3.1.1*.

2.3 Error Compression

The error compression has to be sent through the channel and will likely be one of the most critical things in the compression rate of our algorithm. It is key to ensure that our error meets the properties and strengths of the compression algorithm we decide to choose. We have not defined nor modeled our error yet but, as we want to correct a reconstructed image, it is fair to assume that the error will be shaped as several 2-D vectors, if it is represented as an error channel image in some color space. We could even have it represented as a 1-D vector in some implementations.

In case we have a 1-D vector, common entropy compression techniques like arithmetic coding or Huffman encoding could be a good approach.

One approach to compress an error image, would be to straight out flatten it and also use entropy techniques. However, depending on the nature of this reconstructed image and how we extract its error, we might be able to exploit the spatial dependency. Several classical image compression algorithms work well with 2-D spatial dependant images. Among the most common ones, we find JPEG and PNG. We have to immediately discard JPEG since it is a lossy compression technique and we can not allow losing information in the error transmission. If we did so, we would not be able to fully reconstruct the image because the encoder's color feature vector would not be the same at the decoders end: different error values. PNG, on the other hand, is a lossless algorithm that could work if the error characterization suited its strengths; strengths that will be discussed in *section 3.1.1*.

Whatever the case is, all the above techniques work the best when the entropy is low but also when the input satisfies some characteristics:

- Small dynamic range: tables used to map pixel values to bits representation (think of Huffman encoding or values to probabilities in the arithmetic coding case) would have less entries generating less overhead resulting in less bytes to be transmitted.

- Low frequencies: having low frequencies thus plain areas with similar or exact same values would allow us to explore vector quantization. If we were already exploiting vector quantization, we would be able to construct longer vectors, compared to high frequency images, while keeping the same average code length. Having low frequencies is somewhat related to having low entropy.

- Low energy: the more energy an image has, the more bits per value we will need to represent its values. If an image maximum was 1024 we would need 10 bits. Instead, if and image maximum was 256 we would only need 8. Nowadays, most algorithms avoid this issue by normalizing the signal by a maximum or by differentiating it so the variation is what is actually encoded. In both cases, once in a while the energy/power of the signal has to be transmitted so decoder is able to know which is the normalization factor or what is the first value which we started differentiating. The larger the energy/power the more bits invested in transmitting it.

Trying to minimize all the above all together will indirectly lower the error entropy. So, a good way of trying to generate our error will be by trying to minimize this three characteristics.

2.4 Color Feature Vector Compression

As we said, we need to transmit through the channel the feature color vector extracted by the autoencoder. We could simply send the whole array of numbers; however some compression techniques could be applied. We could, for example, quantize the vector. In this case, we will not lose information because the quantization error induced by changing the bottleneck, thus changing the output, will be represented in the error image. The error image energy will increase and, therefore, it will make it more expensive to compress. In conclusion, a tradeoff arises: quantization of the bottleneck vs error image compression.

In order to quantize the vector, a small dynamic range would provide, for the same number of bins, less quantization error we make since each value is closer to its representative. However, by restricting

the dynamic range of the vector's values, we reduce the potential information that this vector can encode so we need to be careful when we reducing this range. Also, even when quantizing the vector, further techniques can be applied to compress it even more. Among these possible techniques we may find arithmetic coding or, if we set the vector as an image, we could also classically compress it in PNG or JPEG.

We should keep in mind that if any change is performed to the vector during its compression that makes it not fully recoverable in the decoder's end, the encoder should compute his error with the changed vector to ensure that both encoder and decoder will have the same reconstruction at their ends to later apply the error. Otherwise, the decoder would reconstruct the color information differently and the reconstructed image would not be the same. Thus applying the error correction would not result in the original image.

2.5 Problem summary

If the error and color information compression is less than the compression of the image itself, our algorithm would be useful and will prove that we have done a good job in our compressing task. All the ideas and tradeoffs discussed in the above subsections will be explored later in this memory when we explain our proposed solution.

In *Figure 2 and Figure 3*, we can see how we intuitively want to build our framework.

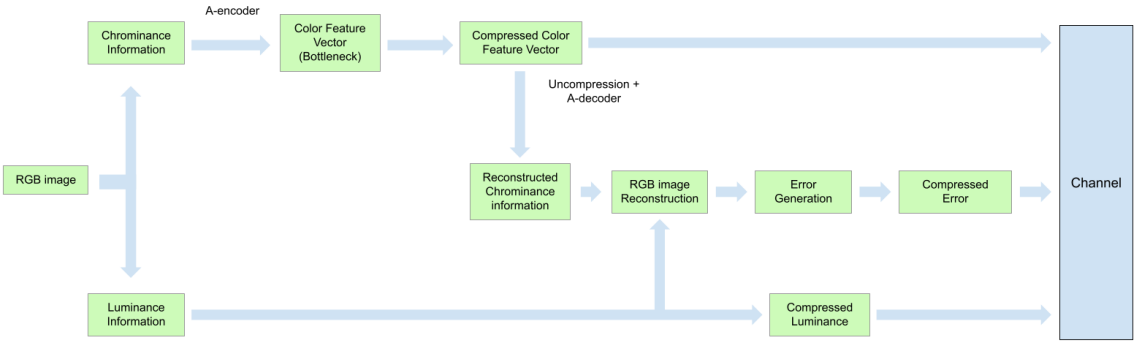

Figure 2: Framework's intuitive encoder part.

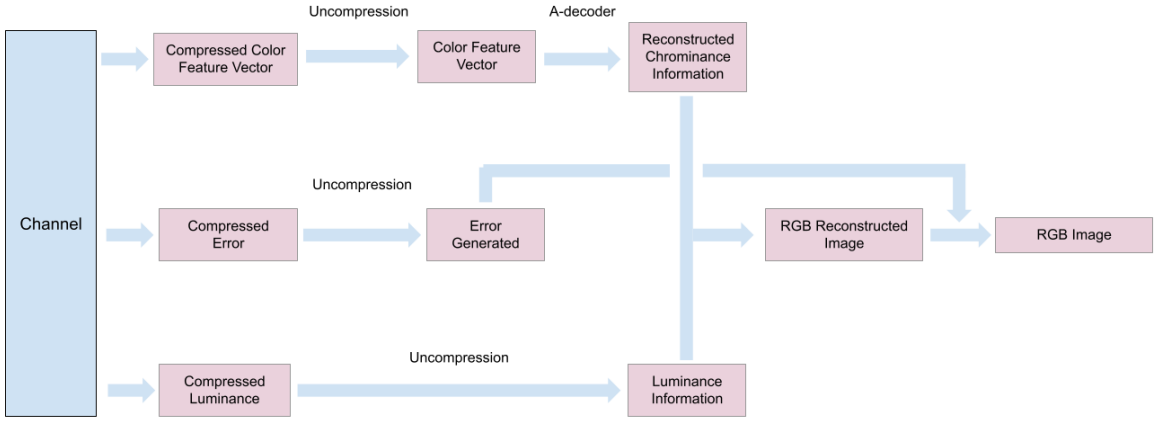

Figure 3: Framework's intuitive decoder part.

3. Related Work

3.1 Fundamentals

3.1.1 PNG

[1] PNG compression method will be quite important in the development of this TFG. We will use it to encode the luminance and, as we will later see, we will use it to compress the image error too. This standard is a lossless compression process used for image encoding that works in two stages: filtering and bit reduction.

The filtering of the image is used to exploit the spatial correlation of images. It aims to encode the differences between pixels rather than their absolute value. Presumptively, the differences between near pixels will be around 0. This filtering will reduce the energy and the dynamic range of the image so it will be easier to compress.

Each pixel is filtered regarding the difference between itself and to the left, the pixel above, and the pixel above-left. How this pixels are combined to compute the difference is optimized line by line. Sometimes we only want to take into account the above value (vertical correlation), the left value (horizontal correlation) or an average of them plus the above-left one (vertical and horizontal correlation). The offset needed to reconstruct the filtered image should be compressed too.

After that the bit reduction stage starts.The filtered image is compressed using the Deflate [3] data compression approach. This method divides the image by blocks and for each block builds a Huffman tree not only for pixel values existing in the block, but also for these values repetition: how many repeated

[1] This section has been summarized from [2]

values we have in a row. In other words, we are mixing a Huffman value through value codification with a Huffman run-length encoding of the block's values. Consequently, we are interested in offering to the Deflate algorithm constant and homogeneous images thus the filtering. On top of the Deflate algorithm, a duplicate string elimination is also done to check if we have repetitions and redundancy in the bit string obtained after the Huffman encoding.

So, from we have seen, we can intuitively say that PNG will work well with low entropy images that have low frequencies (when filtering we will have lots of near 0 values), small dynamic range (Huffman trees will take advantage of nonexistant values) and have low energy (less expensive to encode the offset to reconstruct the filtered image).

3.1.2 LAB Colorspace

The LAB colorspace is a three dimensional space that represents a color in a dimension (L) representing the color's luminance and two other dimensions (A and B) that represent the chrominance of the color (see *Figure 4*. More concretely, A represents the greenness (negative values) or the redness of the color (positive values) and B represent the blueness (negative values) or the yellowness (positive values) of the color. Chrominance values close to 0 represent gray colors, in other words, colorless pixels. The L component takes values in the $[0, 100]$ interval whereas the A and B components take unbounded values. However, in order to match the range of RGB values that 8 bits (1 byte) can represent these two components are bounded from -128 to 127.

This color space, unlike RGB, has a color representation and progression more along the lines of the human eye perception. This implies that the straight space transformation from RGB to LAB and viceversa is not linear.

3.2 Deep Learning Approaches to Image Compression

In [4], they tried to exploit the overfitting power of multi layer perceptrons (MLP) to compress images. They overfit an MLP which tries to predict for every coordinate *(x,y)* their pixel values. The encoder is the one that overfits the MLP and transmit its weights through the channel. The decoder then forwards the MLP for every possible *(x,y)* combination and reconstructs the original image with the output got. This is a lossy approach because, unless the network perfectly overfits the input image, the reconstruction will not be perfect. They have good rates with small images but struggle with bigger ones. We think that this approach is very interesting and quite original.

In [5], they present a lossy algorithm which use a very interesting idea. They present an importance map feature extractor. This importance map identifies image regions that are more important than others. This information will be considered when performing the bit allocation: more bits will be allocated to important regions causing a better rate/distortion performance. We will not make use of this technique since we work on a lossless framework and we will not have to worry about bit allocations and bit distortions. However, in the paper, they present an autoencoder-based framework similar to the one we propose. First, an A-encoder extracts features, they are quantized and then compressed. The compressed features are sent through the channel and received by the decoder. The decoder uncompresses the features and forwards the A-decoder with the extracted features to finally get a reconstruction of the original image. In their

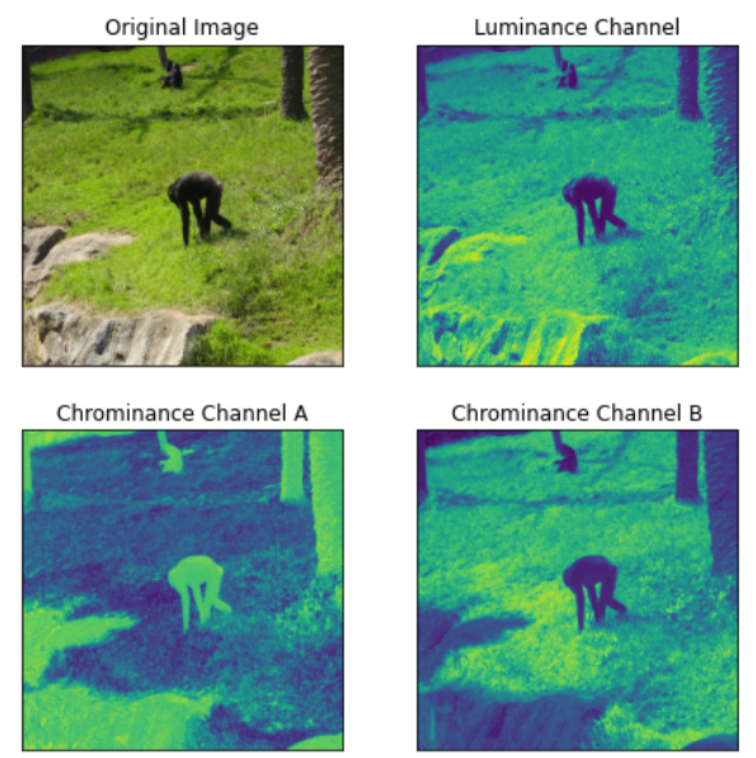

Figure 4: Example of a LAB decomposition.

case, they do not need to compress the error because they do not have the lossless constraint.

The authors of [6] present a lossless framework a bit different from ours. They use a deep learning approach to reconstruct a pixel from its neighbourhood whereas we try to reconstruct the image from a vector that encodes features of the whole image. They later encode the prediction error using a reference codec. We could say that they use a recurrent approach which is quite common nowadays in the image reconstruction field [7]. However, we will decide to use an autoencoder approach because there are lots of different versions and types of recurrent networks which all require different architectures and implementations (LSTM, Transformers, etc).

4. Proposed solution

So considering the structure and parts that our framework aims to have, in this section we explain and reason the solutions found and their advantages and disadvantages. The proposed solution have some changes with respect to our initial main scheme presented in *Figure 2 and 3*. The proposed solution scheme is shown at the end of this section in *Figure 8 and 9*.

4.1 Color Feature Vector Extraction

As we said in the previous sections, we will try to encode color information with the bottleneck extracted from autoencoder. The autoencoder will take as an input the original image's color information and will try to output a reconstruction which tries to reassemble as much as possible to the input under the criterion of a loss function. In the autoencoder, the dimension of the input image is reduced until reaching the bottleneck which we will use as the color feature vector that will be sent through the channel. Then, the decoder will be the one in charge of reconstructing the image from the color information of the bottleneck, the error and the luminance. The weights of this autoencoder will have to be trained under some criterion in orther to make the neural network learn how to extract this color information.

4.1.1 Autoencoder Inspiration

The structure of our autoencoder has been inspired from [8]. In [8], they aim to color images from the luminance channel by transforming the RGB image to the LAB space. Their color information is the two color LAB channels and their luminance is also the luminance channel from the LAB space. So they take as an input a $H \times W$ tensor (luminance channel) and the autoencoder outputs a $2 \times H \times W$ tensor which try to imitate the original image two color LAB channels. They downsize the original image to reach the bottleneck dimension by a factor of 8. We also should note that, in their case, the bottleneck consist of 512 2-D channels. In other words, it has a shape $512 \times \frac{H}{8} \times \frac{W}{8}$.

They use some global hints which are shown to the bottleneck just one time. These hints have color information that is merged with the features extracted from the black and white image. By doing so, it will be easier for the A-decoder to recover the color channels because it will have some extra knowledge about the desired output.

The loss function they use is the Huber Loss defined as:

$$L(y, f(x)) = \begin{cases} \frac{1}{2}[y - f(x)]^2 & \text{for } |y - f(x)| \leq \delta, \\ \delta\,(|y - f(x)| - \delta/2) & \text{otherwise.} \end{cases} \tag{1}$$

4.1.2 Autoencoder's Input and Output

From [8] we will take the main architecture of the autoencoder. That is, the composition of their main convolutional layers, as well as the approximate number of them. Their objective was to start from a luminance image and then output the 2 color channels. In our case, we want to output the 2 color LAB channels too but we do not have the restriction to only use the luminance channel because at the beginning we have all three LAB channels of the image. Because our objective is to have as much color information as possible in the bottleneck, we have to provide an input that maximizes this reconstruction. What is better than having as an input the same exact channels that you want to reconstruct? If we input the same two LAB color channels that we want to reconstruct we will make sure that the bottleneck has the maximum information about the output as possible. So we will replace the luminance channel input they use in [8] for both color channels.

The autoencoders' output will have values between -128 and 127 (LAB color components range). However, if we just tell the network to predict values in this range we may lack performance in the loss function minimization. That is because the neural network will have to learn to output the values in this wide range. It might get stressed during the learning process because it will have to learn in the first stages numbers so split across images. In order to ease this task for our autoencoder, we will multiply, as they do in [8], by a constant of 110 so that the predicted values will approximately fall between the range $[-1, 1]$.

4.1.3 Luminance Importance

We do not have to completely ignore the luminance channel. The decoder will also have it at his end so it can be used and merged somehow in the decoding part when trying to reconstruct both color channels from the bottleneck. We believe that the luminance channel can really help in extracting the Color feature vector from the autoencoder making reconstruction easier and therefore decreasing the reconstruction error energy. Let us elaborate why and how luminance information should be used.

Let us suppose that in the bottleneck we have the information "there is something blue in the top corner of the image". When reconstructing the image with only the bottleneck information, the A-decoder will not know exactly what shape has this blue color unless the bottleneck is big enough to hold this information. But as we have seen, we ideally want the bottleneck to be as small as possible. So, if the A-decoder can not guess from the color extraction where this blue shape exactly is, the information will be encoded in the error image which will lead to a poorer compression rate. However, if we take advantage of the fact that the decoder will already have the luminance channel and therefore some information about the contour, the A-encoder will not have to encode contour information into the bottleneck. In this case, less information will be needed to be kept in the bottleneck which will allows us to use a smaller one. Remember that larger bottlenecks mean more bytes needed to send them through the channel.

4.1.4 Merging Luminance and Chrominance

We will take advantage of the scheme used in [8] for global hints for introducing the luminance information to the architecture. So in the end, the A-encoder will have as input two color channels and will output a bottleneck. On the other hand, the A-decoder will have as input the bottleneck and the luminance channel. The A-decoder will have to output the reconstruction of the two color channels. So in this decoding phase one question arises: Which merging technique should we use for combining both luminance channel and color feature?

We propose two of them: concatenating tensors or using some sort of attention mechanism [9]. The performance and discard of these techniques will be discussed when we present the results of the autoencoder modeling (*Section 5*).

4.1.5 Loss Function

In [8] they present an interesting discussion about what loss function should be used in this kind of colorization task. L_2 loss, in a colorization framework, may lead to dull and desaturated images because this loss function punishes the network for providing wrong guesses to saturated colors. L_1 loss is more prone to

predict larger values in the exchange of not being more punishable when this predictions are wrong. Higher values in error images will be present but the network will be bold to predict saturated colors. Often, this predictions will be correct and, therefore, the error in high saturated colors will be lower.

Another way to enhance the model to make bolder decisions would be to use a classification loss where we a class for all possible 256 values is assigned($[-128, 127$ range). In this case the model is equally punished when it misses by either one value or by a hundred. However, we will not deeply explore this kind of loss function because the color channels in our lab space take continuous values making the creation of a set number of classes undoable. Moreover, we still want the error to be close to the predicted value even thought the prediction is not perfect. We can take advantage of a low dynamic range where the values close to 0 are more present.

We could also consider the Huber loss (1) which [8] ended up deciding sticking to. However, as it accepts an additional hyperparameter to explore, we will, at first, discard it. We want to evaluate hyperparameters more directly related to the image compression topic even though the choice of the loss function is quite important.

So before deciding which will be our go to loss function, keep in mind that we want homogeneous, low energy and low frequency error images. Considering the above, we are fine with using the L_2 loss. Comparing it with the L_1 loss, when using the L_2, we will have more frequent error but it will be less energetic. The prediction will tend to be closer to 0 than to the other side of the colors spectrum. This happens due to the conservativeness of the L_2. Having more controlled and compacted dynamic range will also help when trying to model the error.

4.1.6 Bottleneck Shape

One of the other aspects we should analyze is what shape our bottleneck will have. If we use the same architecture as [8], we realize that the bottleneck will be a tensor of size $512 \times \frac{H}{8} \times \frac{W}{8}$ which is too large! For example if we picked a two color channel input of size $2 \times 512 \times 512$ which consists in 524288 pixels, we would have that our bottleneck would have more values than the input image. The bottleneck (yet to be decided how to be compressed) would weight more than the original image. Therefore, we have two factors to play with: the factor of image downsizing, which will affect to the bottleneck size of both H and W dimensions, and also the number of channels. From now on, during all the experiments we will set the image downsizing to a factor of 32, so we will end up with $Channels \times \frac{H}{32} \times \frac{W}{32}$. The downsampling will be performed by max pooling layers with a factor of 2 and the upsampling will be done with Transposed Convolutional Layers [10].

The main parameter that we will focus on when exploring the tradeoff of bottleneck size vs error energy will be the number of channels. For the sake of the problem comprehension and baseline construction, we will explore excessive large bottlenecks with 512 channels down to 1 or 2 channels.

4.1.7 Skip Connections

In [8] they also explore the idea of skip connections between A-encoder and A-decoder. Skip connections are a good technique to avoid gradient vanishing and are commonly used in deep learning frameworks [11]. Unfortunately, due to our problem defintion, we can not use this technique between A-encoder and A-decoder because we would need to send this information through the channel causing a decrease in our compression rates. However, we can make use of this technique inside the A-encoder or A-decoder themselves. Specifically, we will try to introduce some residual layers in the A-decoder where we will merge somehow the latest convolutional blocks with the bottleneck layer. In the results section we will discuss whether this technique implementation is useful or not.

All in all, our network will have the scheme showed in *Figure 5*.

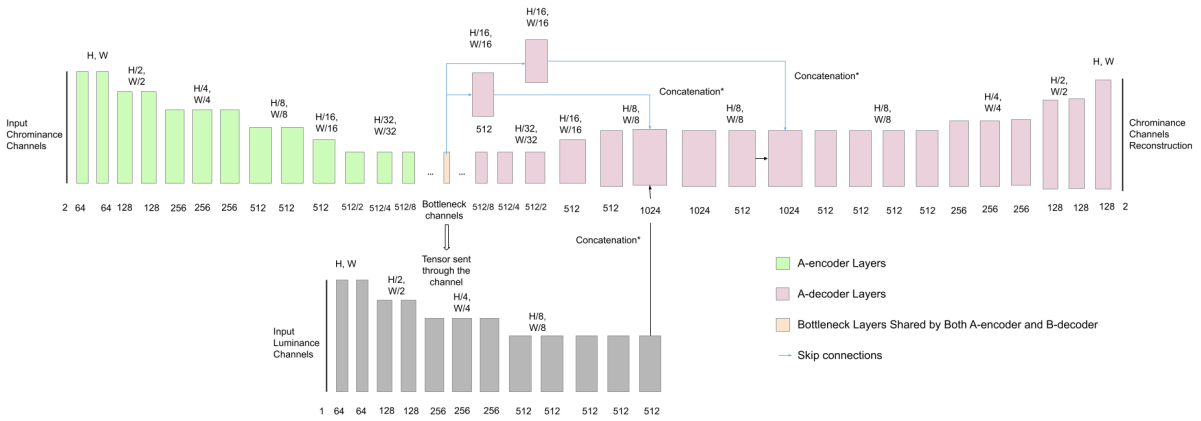

Figure 5: Proposed Autoencoder. Below each block of layers we have the number of output channels. Their number of input channels are the output channels of the previous layer. We used max pool layers to upsample images and transposed convolution to downsample tensors. After each block we use a RELU activation function followed by a batch normalization. *Concatenation can be substituted by any tensor merging technique.

4.2 Color Feature Vector Compression

As we have already said, the color feature vector will correspond to the bottleneck of the autoencoder. This bottleneck will be a 3-D tensor with shape $Channels \times \frac{H}{32} \times \frac{W}{32}$. We can actually think of this 3-D tensor as *Channels* 2-D tensors since these 2-D tensors will have some sort of spatial relation. Having that said, we think we could attack its compression in three different ways: inter-channel compression where we explore redundancies through different channels; intra-channel compression (compress *Channels* 2-D tensors) where we take advantage of spatial redundancy or straight out vector compression where we compress the bottleneck as a flat vector.

- Inter-channel compression: we believe we will not be able to exploit much in here. That is because if the autoencoder tries to encode as much information as possible each channel should store independent information. If some redundancies across channels did exist, it would mean that we could

reduce the number of channels without a major decrease in the color error. So we will not explore this kind of redundancy.

• Intra-channel compressions: in here we believe some redundancies might exist although, if the redundancy is important it would mean that we could downsize these 2-D tensors. We think that we can have some redundancy because, for example, if a color is predominant in an image, its influence in the intrachannel values might also be predominant. Even though, we should say that we can not predict how the neural network will encode the color information and this redundancy might not exist. Anyways, if it existed, we could exploit it by compressing the 2-D tensors as regular images. PNG could be a good way of doing so but also could JPEG introducing loss to the bottleneck transmission. This information loss would then be reflected in the error image so we are not braking the lossless constrain. We should keep in mind that if H and W of the input image are small, these 2-D images are going to be really small too and in some cases compression techniques' overhead might discard compressing these 2-D tensors: larger H and W might give us better chances of exploiting the 2-D compression.

• Flat vector compression: we could flatten the whole 3-D tensor into a 1-D tensor. Then, applying arithmetic coding for example, would in theory compress it. In here, small bottlenecks with few *Channels* and small input image's H and W could also lead to have overhead in this entropic approach since probability tables could be more expensive than the vector itself. If that was the case, we could consider binarizing the vector's values thus extending each value to 8 1's or 0's. Then the table would only have two entries and the vector would be longer so the overhead would not be as prevalent. However, the probability of 0 or 1 in binarizing random values is approximately a half so this approach could not work since entropy would be maximum.

Another common technique to reduce the number of bytes needed to encode this vector could be quantization. In our network we will work with floating points which occupy 4 bytes of memory. This means that every value in our bottleneck will occupy 4 bytes. However, we could analyze how the quantization of this 4 bytes would impact the autoencoder's performance and the error generation. The bigger loss of information in the quantization of this vector, the more energy the output error will have since the values on the decoder's A-decoder bottleneck will not the ones that the encoder's A-encoder found as optimal to represent the color information. We will reduce the number of bytes used in encoding the bottleneck's values.

Another important aspect in the quantization is to try to set a controlled dynamic range as small as possible. This way, the error in quantization will be less impactful. However, reducing the bottleneck's dynamic range can lead to a poorer color information extraction since we will have less margin to encode this information. This tradeoff will also be studied in the results section. The approach we will use is to try to reduce the dynamic range using activation functions just before the bottleneck layer. More specifically, we will try with the tanh function which bounds values between $[-1, 1]$ and the sigmoid function which bounds values between $[0, 1]$.

4.3 Error and Luminance Compression

We could characterize our error and build our own compressing algorithm from the study of this characterization. However, it would be hard and time consuming so we will recycle an already existing compression technique which we think it will work well in our framework.

We are compelled to use a lossless approach to compress the error; otherwise we would not be able to fully recover the original image. So one initial take would be PNG compressing both error channels individually since we have seen that our error properties suit the PNG strengths. Unfortunately, this approach will not work since the A and B components in the LAB space are continuous in the $[-128, 127]$ interval. We would need floating points to represent this values which occupy 4 bytes whereas an RGB image is represented with 256 discrete values which only take 1 byte to be represented: compressing the error represented in floating points would be so much more expensive than simply compressing the original image with the same PNG approach. So we need to find a way of compressing this error image with at most one byte per value.

One may think that quantizing the error image in 256 because it is the same quantization error that standards make nowadays (1 byte per value) and the LAB space can take 256 different integer numbers. Nonetheless, this quantization can cause some other issues. In one hand, we have that we would be committing quantization error which arguably could not be taken into account when breaking the lossless constraint because, at some point, we will need to quantize the error. On the other hand, we also find that we will be quantizing in the LAB space which is not linear with respect to the RGB space. So, quantizing with and error of $[-0.5, +0.5]$ could imply having more than a $[-0.5, +0.5]$ loss in the RGB space, which is a constraint that most standards have as a maximum.

The solution we propose is to transform the two chrominance error images from the LAB space to the RGB space and the submit through the channel of this RGB image. We already have the A and B channels of the LAB but we can not do the transformation without the L component. So we will decide to jointly transform the A and B errors with the luminance of the image to the RGB space. Implicitly, we are also solving the problem of compressing the luminance channel.

The decoder, by transforming this RGB image back to the LAB space, will have the luminance component that it will use to forward the A-decoder together with the bottleneck, and also the error image in the A and B channels that will be used to correct the A-decoder's prediction. Later, with the A and B corrected channels, and with the luminance part, the decoder can transform again from LAB to RGB to fully reconstruct the original image.

The intuition behind this solution and why we think it will work is because when substituting the original color channels by the error in them as the A and B LAB channels, we are decolorizing the image in all the RGB components because chrominance values close to 0 represent grey pixels. We are lowering the energy, the entropy and the range of values of the original RGB picture which, theoretically as we already have discussed, will be less expensive to compress than encoding the original image itself. In *Figure 6*, we can observe how using the error as the A and B channels decolorizes the image. In *Figure 7*, we can see how the error image has a lower single pixel entropy than the original chrominance values.

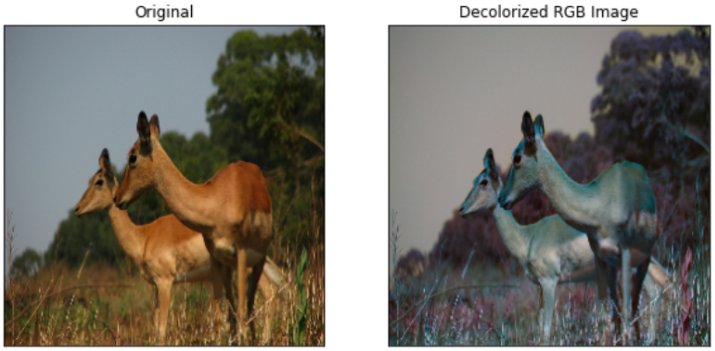

Figure 6: Original image vs image with prediction error in the chrominance channels.

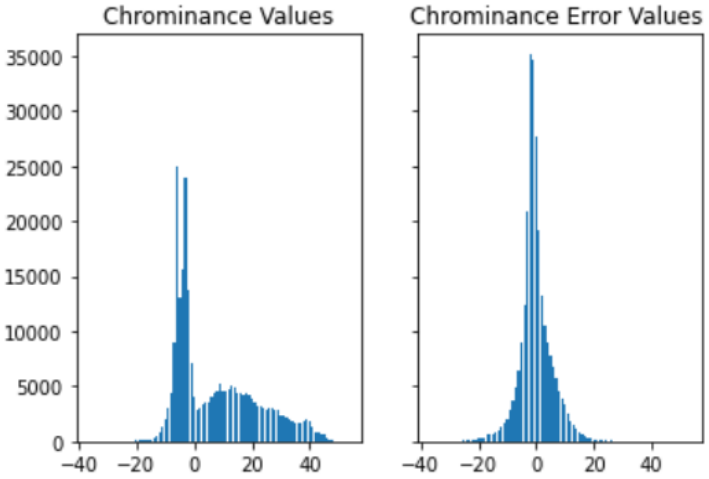

Figure 7: Histogram of the chrominance values of Figure 6 original image and histogram of the prediction error.

This solution solves the quantization issue in the LAB space and quantizes in 256 integers in the RGB space. This level of quantization in the RGB space is very common across many compression standards. However, this approach also carries out some other issues. We have that A and B components in the LAB space can only take values from -128 to 127 which means that if we made errors larger than this values (maximum of -256 or $+256$]) we would encounter some issues with the LAB to RGB transformation because it is not defined in these out of the range values. Anyways, we will assume that this will not happen and, if it did, it would hardly ever happen. Presumably, the autoencoder will not make these large errors due to the conservativeness of the L_2 loss function.

After explaining how we want to solve the different parts of our framework, we can revisit our framework's first draft shown in *Figure 2* and 3. In *Figure 8* and *Figure 9* we put more detail on how the encoder and decoder should actually proceed regarding our proposed solution to the initial intuited framework.

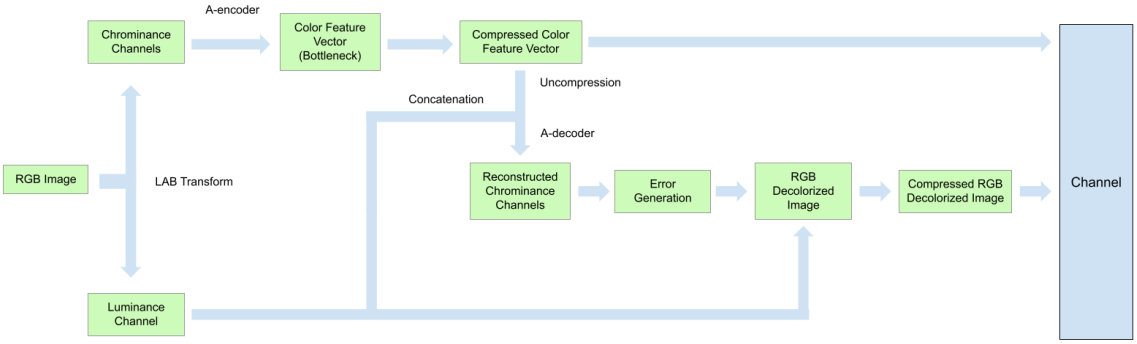

Figure 8: Proposed solution encoder part.

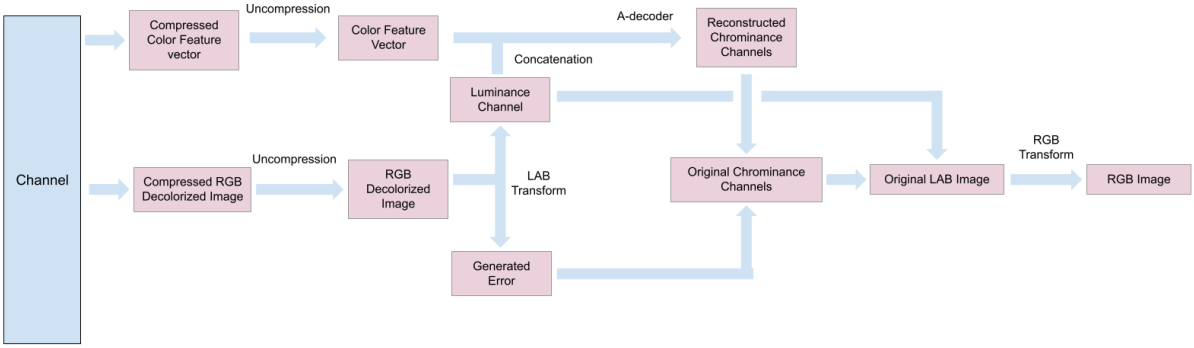

Figure 9: Proposed solution decoder part.

5. Experimental Results

We will present results in a way that will recreate chronologically what we were observing from our test and experiments. This way, the reader will comprehend and understand the reasons behind every decision and parameter choice.

5.1 Dataset

To train and validate our proposed algoritm we will use a subset of images of the places365 standard dataset. The set we used consisted in 374985 training images and 36421 validation images. All of them where encoded in the RGB colorspace and had a size of 256×256 pixels. For the final testing we used a subset of 20121 images from Imagenet of same images sizes as validation. The RGB images will take values across their three channels from 0 to 256.

5.2 Autoencoder Optimization

We will evaluate the performance of the autoencoder itself with the minimization of the L_2 loss function because we believe that, indirectly, minimizes the entropy of the error image generated. Obviously, this L_2 loss will be affected by the different choices we take when setting all the parameters. But the most important one will be the bottleneck's number of channels. So we initially wanted to test how the L_2 loss behaved when we set a large bottleneck and a small bottleneck.

The network architecture we used for this first experiment and the one that we will use is the one described in *Figure 5*. However, we will not be using skip connections yet. We initially joined the luminance features with the chrominance by concatenating the tensors. The following training parameters will be permanently set (unless we say the opposite):

- Learning rate: 0.0005
- Batch size: 32
- Image downsampling: by a factor of 32
- Number of epochs: 124
- No initial trained weights

The initial results with respect to the bottleneck size can be seen in *Table 1*.

Table 1: Validation MSE comparison across different bottelneck sizes.

Number of channels	Validation MSE	Values in bottleneck (encoded as floats)
512	19.0	32768
16	32.4	1024
8	36.4	512

We can see that the values obtained are along the lines of the expected. The more information we let through the bottleneck the less error we make in reconstructing the input image. However, we believe that 512 channels in the bottleneck are not viable from a compression standpoint since the bottleneck will weight too much with respect to the mean PNG weight of the Validation data set (9614 bytes). Nevertheless, 8 and 16 are an affordable number of channels: we can see in *Table 1* that the number of values in the bottleneck is less than the average PNG weight of the validation set (9614 bytes). This bottleneck has yet to be compressed and we still have to check if the gaining in the error compression images plus the weight of the bottleneck is higher than the weight of the PNG compression of orignal images.

Once we check that we have successfully trained our autoencoder and that the first modeling steps have reasonable behaviour we will try to improve this Validation L_2 loss.

Our first idea to modify the network is to change how the luminance information and the chrominance information interact which each other. We tried to compare, besides concatenating both tensors, using an attention mechanism [9] between them both.

Table 2: Validation MSE comparison across with different merging mechanisms.

Merging mechanisms	Number of channels	Validation MSE	Values in bottleneck (encoded as floats)
Concatenating	512	19.0	32768
Attention	512	20.5	32768
Concatenating	8	36.4	512
Attention	8	38.8	512

From *Table 2* we can see that Concatenating and using Attention have similar L_2 minimization behaviour with concatenating tensors being slightly ahead in both large bottlenecks and small bottlenecks. These results will makes us decide to use concatenation from now on since it seems to be performing a bit better regarding the L_2 loss and also, because its simplicity with respect to the attention mechanisms.

5.3 First Models Compression Gains

Once we have established that, we will concatenate the luminance features with the chrominance ones. Let us check what happens when we use skip connections in the A-decoder.

Table 3: Validation MSE comparison with addition or removal of skip connections.

Skip Connections	Number of channels	Validation MSE	Values in bottleneck (encoded as floats)
NO	512	19.0	32768
YES	512	17.56	32768
NO	8	36.4	512
YES	8	31.4	512

From *Table 3* we can see that the addition of skip connections has an impact on the Validation MSE. Specifically, it has a major improvement when having small bottlenecks, the domain we will most probably work around. We will keep using skip connections from now on.

Now that we are taking some important decisions about the autoencoder's architecture, we will start to analyze their generated error and how good the error plus the luminance compresses with respect to the original images. We will present MSE values for the models as well as the entropy comparison between the original validation images's chrominance channels and the single pixel entropy of the error generated. Original images' average chrominance single pixel entropy is 4.8. We believe that the gain in compressing rates will be correlated with MSE and ultimately with the single pixel error entropy. However, we should keep in mind that PNG standard is using joint entropy blocks; our single pixel entropy will not lower bound the compression rates but, as we said, will be a good indicator of the potential performance of

PNG's compression. In *Table 4* we present all this metrics for both skip connection models already analyzed.

Table 4: Compression information of two skip connection models.

Model	Validation MSE	Average error entropy	Average error + L compression (bytes)	Gain wrt raw PNG* (bytes)
Skip w/ 512 ch	17.56	3.42	8598	1016
Skip w/ 8 ch	31.4	3.55	8854	760

* This gain does not take into account bottleneck's encoding needed bytes.

Several conclusions can be drawn from *Table 4*. Low error entropy seems correlated with the average error compression. However, the differences between both entropies seem smaller considering the differences between MSE values and the differences between gains. The error. As we present this metrics for several other models, we will check whether there is a correlation between all these metrics.

5.4 Bottleneck's compression

5.4.1 Lowering Dynamic Range

The sooner we take into account bottleneck's compression and provide real gains, the sooner we will start working towards encoding it. The first idea that comes to mind and has already been discussed is quantizing it. So our first step to achieve this quantization will be to try to reduce the dynamic range of the bottleneck by using activation functions just before the bottleneck layer.

Table 5: Compression information using activation functions.

Model	Validation MSE	Average error entropy	Average error + L compression (bytes)	Gain wrt raw PNG* (bytes)
No activation w/ 8 ch	33.2	3.55	8854	760
Tanh w/ 8 ch	41.7	3.60	8920	695
Sigmoid w/ 8 ch	37.35	3.80	8854	594

* This gain does not take into account bottleneck's encoding needed bytes.

We can see in *Table 5* that the the bigger restiction of the dynamic range of our bottleneck, the poorer error plus luminance compression we have. This result is reasonable since the autoencoder has a smaller margin to encode the optimal bottleneck values to reconstruct the input channels. We can also check that the gain/loss in the error + luminance compression correlates well with the single pixel entropy of the error. On the other hand, the MSE does not entirely behave as we would expect since we assumed that the MSE minimization was indirectly correlated to the error entropy minimization. Using the tanh gives us lower entropy than the sigmoid activation function but tanh has a higher MSE. Surely, MSE minimization will have an impact to entropy minimization and later better compression gains but it seems that is not that linear.

5.4.2 256 levels Uniform Quantization

The compression gains still remain high enough to encourage us to keep experimenting with this dynamic range bounding. We will proceed to quantize the bottleneck uniformly with 256 different values. That is, one byte per bottleneck value. For the model with no activation functions we set our dynamic range to be $[-max(abs(bottleneck)), +max(abs(bottleneck))]$. We would need an extra 1 or 2 bytes to encode this maximum value which we will ignore in our metrics computation. As sigmoid and tanh have set dynamic ranges, no extra maximum values will be required to be transmitted. During the autoencoder's training, we will not quantize the bottleneck. The bottleneck will only be quantized when doing model inference. In *Table 6* we can see the results of this quantization.

Table 6: Compression information with 256 uniform quantization.

Model	Validation MSE	Average error entropy	Gain w/o bn encoding	Gain wrt raw PNG (bytes)
No activation w/ 8 ch	33.2	3.70	698	186
Tanh w/ 8 ch	52.4	4.0	695	183
Sigmoid w/ 8 ch	52.9	3.88	591	81

As we expected, looking at the 'Gain w/o bn encoding' column in the *Table 6* the compression gain for the model that has no activation has dropped quite a bit when the quantization error is large because the dynamic range is unbounded. On the other hand, the compression gains when we use some activation function stayed almost the same. We could justify this behaviour by saying that, as the quantization error is lower, the model does not get affected by the bottleneck changes. However, if we compare the activation function's MSE and entropy error between *Table 5* and *Table 6* we see an important variation in both metrics when compression gains remained the same. This phenomenon is quite strange and is hard for us to find the reason why this is happening apart from saying that both MSE and single pixel entropy are not direct measures that perfectly correlate with the performance of the PNG compressor.

From the experiments from *Table 6*, we can also conclude that we will rather use autoencoders modeled with activation functions before the bottleneck because they allow us to encode bottlenecks with 1 byte per value without a major decrease in performance. Nevertheless, this bottelneck compression, for this bottleneck size, is insufficient as we can see in the last column in *Table 6*. We will have to either lower the bottleneck channels or keep compressing the bottleneck. For now, let us explore what happens if we reduce the number of bottleneck channels while still quantize the bottleneck. We can see the results in *Table 7*.

5.4.3 Bottleneck's Size Decrease

In *Table 7* there are some interesting results. First of all we can see that sigmoids with 4 and 2 channels outperform the sigmoid model with 8 channels in terms of luminance plus error compression. This behaviour is quite unexpected.

We also conclude, now that we have presented more results, that MSE seems to have little to no

Table 7: Compression information with 256 uniform quantization and different bottleneck's size.

Model	Validation MSE	Average error entropy	Gain w/o bn encoding	Gain wrt raw PNG (bytes)
Tanh w/ 4 ch	42.5	3.7	747	491
Sigmoid w/ 4 ch	52.9	3.9	693	437
Tanh w/ 2 ch	52.4	3.9	634	506
Sigmoid w/ 2 ch	39	4.0	611	483
Tanh w/ 1 ch	70.1	4.1	490	424
Sigmoid w/ 1 ch	66.8	4.0	468	404

correlation to the compression rates we are getting. On the other hand, we could say that error entropy is a bit more correlated to the compression of the luminance plus the error image. This make us believe that perhaps, we should train our autoencoder to minimize the overall error entropy rather than the L_2 loss. Nevertheless, the entropy of an image is not differenciable since we are binning the image in order to calculate the probabilities. Also, the error follows a distribution characterized by the difference of a ground truth image minus an image generated minimizing some loss. We lack intuition to think what loss could directly minimize this error entropy. During our experiments we have done some testing with changing the loss to check whether the error entropy and error gains improved or not. We tested the L_1 loss, the L_3 loss and the Huber loss (1) with several δ. Unfortunately, we struggled to find any improvement so we will not walk through the results.

Anyways, the most important take from *Table 7* is that we can clearly see the tradeoff between the number of bottleneck channels (expensive to compress) with better compression rates and less bottleneck channels (cheaper to compress) with poorer compression rates. The tradeoff seems to be maximized when we have 2 channels in the bottleneck so the following analysis will be centered around using tanh and sigmoid activation layers and using 2 channel-bottlenecks.

5.4.4 Other Compression Techniques

In section 4.3 we discussed about several compression approaches. We highlighted the intra channel compression (PNG or JPEG compressing independently every channel) or straight flattening the bottleneck and try and arithmetic coding aproach. However, as our bottleneck has 64 values ($2 \times 8 \times 8$), the overhead in these techniques will be higher than the vector itself. Compressing the bottleneck later on with them will be not worth it unless we heavily increase the size of this bottleneck.

Once we have found the best number of channels per bottleneck, let us see how we can further compress it. One way of doing so, would be improving our uniform quantization. We could analyze the bottlenck values distribution and rearrange the 256 levels to fit its statistics. We will not pursuit this idea because uniform quantization seems to perform good enough and we have decided to focus on testing some other ideas like decreasing the number of levels; that is, lowering the 1 byte per value bound.

But before jumping into these experiments, in order to decrease the amount of tests, we will try to find which is our best model among the two selected. To do so, we will check how well the models generalize

when we input images larger than the training size.

5.4.5 Image Size Increase

In this subsection, we will test whether our framework works well when compressing images larger than the training ones ($3 \times 256 \times 256$). We will upsample our validation images up to ($3 \times 512 \times 512$) and compare their average PNG compression to our average compression rate. At first, one could think that both ours and PNG compressor would multiply by 4 the rates obtained with the $3 \times 256 \times 256$ images since images are 4 times bigger. However, we will expect a better performance from our framework since the autoencoder should be able to make less mistakes when predicting these images. That is because bigger images have more spatial redundancies which are more exploitable. The regular PNG compressor should take advantage of these redundancies as well, and their rates should be lower than 4×9614. Indeed, the average compression rate of PNG on upsampled images is 21957 bytes. This upsampled validation images have, on average, a chrominance single pixel entropy of 4.8.

We will test all of the above with our two selected models. Both have 2 channels in the bottleneck so, if we keep quantizing it up to 1 byte per value, we have that the bottlenecks will weight $2 \times 16 \times 16 = 512$ bytes.

Table 8: Compression information with ($3 \times 512 \times 512$) image size.

Model	Validation MSE	Average error entropy	Gain w/o bn encoding	Gain wrt raw PNG (bytes)
Tanh w/ 2 ch	24.4	3.6	1824	1312
Sigmoid w/ 2 ch	249.5	5.1	315	-197

In *Table 8*, we can see that the sigmoid model can not generalize very well whereas the tanh model keeps up with the size increase. So we decide that the tanh model with 2 bottleneck channels will be our best model and the one we will do further analysis.

When testing with images of size $3 \times 256 \times 256$ we obtained an average gain of $2.1e-2$ bits per pixel. Now, with images of size $3 \times 512 \times 512$, by looking at *Table 8*, we have an average gain of $1.3e-2$. The small decrease in this metric comes from the improvement of the PNG compressor: 2.28 times higher average bytes with a 4 times image increase; with respect to our compressor: 3.77 times higher. On a side note, we can observe that the MSE has halved probably is due to the color redundancy increase of the image upsampling.

When we increase images to size $3 \times 1024 \times 1024$, the PNG compressor has an average performance of 55233 bytes per validation image which is less than 16 times the 9614 bytes needed for $3 \times 256 \times 256$ input images, concretely it is only 5.7 times higher. This upsampled images have, on average, a chrominance single pixel entropy of 4.8. When our model deals with this image size, with 1 byte per value in the bottleneck, we have that the bottleneck will weight $2 \times 32 \times 32 = 2048$ bytes.

As we can see in *Table 9*, all the patterns found with $3 \times 512 \times 512$ images repeat when we upsample images up to size $3 \times 1024 \times 1024$. We have an average gain of $7.56e-3$ bits per pixel which is again smaller compared to the metric evaluated on $3 \times 256 \times 256$ images. This is also happening because of the

Table 9: Compression information with $(3 \times 1024 \times 1024)$ image size.

Model	Validation MSE	Average error entropy	Gain w/o bn encoding	Gain wrt raw PNG (bytes)
Tanh w/ 2 ch	15.0	3.6	5020	2972

better performance of the PNG compressor.

5.4.6 Different Levels Uniform Quantization

With all these experiments we have selected tanh model with 2 bottleneck channels as our best model because of its good generalization with respect to different input images. Now, let us check what happens when we reduce the number of levels when quantizing the bottleneck.

Table 10: Compression information with different quantization levels on the tanh model with 2 bottleneck channels and $3 \times 256 \times 256$ input image size.

Levels	Validation MSE	Average error entropy	Bottleneck bytes	Gain wrt raw PNG (bytes)
256	52.4	3.9	128	506
128	52.3	3.9	112	530
64	53	4.0	96	504
32	58.1	4.1	80	278
16	86.8	4.5	64	-302
8	235,2	5.0	48	-1058

Table 11: Compression information with different quantization levels on the tanh model with 2 bottleneck channels and $3 \times 512 \times 512$ input image size.

Levels	Validation MSE	Average error entropy	Bottleneck bytes	Gain wrt raw PNG (bytes)
256	24.4	3.9	512	1312
128	24.3	3.9	448	1389
64	25.1	4.0	384	1262
32	30.7	4.1	320	444
16	61.3	4.5	246	-1263
8	223.3	5.0	192	-3007

We can see, in *Tables 10 and 11* that the MSE error goes up as we quantize more and more: the autoencoder struggles when bottleneck values are vastly changed. Also, the error entropy increases as we worsen the model's performance. Regarding the compression gain, we find that it tends to lower as we quantize the bottleneck except when we use 128 levels. The bottleneck quantized with 128 levels outperforms the bottleneck quantized with 256 levels. Our best model then will turn out to be the tanh model with 2 bottleneck channels quantized uniformly at 128 levels. We present also the performance of

this model when we input images of size $3 \times 1024 \times 1024$ in *Table 12*. The bottleneck size in this case is 1792 bytes.

Table 12: Compression information with $(3 \times 1024 \times 1024)$ image size.

Model	Validation MSE	Average error entropy	Gain w/o bn encoding	Gain wrt raw PNG (bytes)
Tanh w/ 2 ch and 128 quantization levels	26.1	3.7	6150	4358

5.5 Error Encoding Problematic

In this subsection, we will analyze whether our compression error, in the validations set, exceeds the $[-128, 127]$ bound that we imposed. This bound comes from our error compression algorithm which substitutes the A and B chrominance channels for their reconstruction error. This A and B channels have this particular $[-128, 127]$ dynamic range.

If our generated error exceeded this values, we would be breaking our lossless constraint. If it also happened regularly, all the results and metrics presented in this report would be severely affected since the function we used to transform from LAB to RGB clipped the exceeded values.

We checked if that was the case with our validation set of image size $3 \times 256 \times 256$. It happens to 326 (0.9%) validation images. Specifically, in those images it affected, on average, 1051 (1.6%) pixels. 0.9% is not a meaningless amount of images but, as it only affects their 1.6% of pixels, we could say that this issue is not a regular pattern in our image compression framework. We are losing a very small information proportion of the overall pixels. In fact, the MSE error per pixel when comparing all our reconstructions (99.1% are perfect and 0.9% are not) with original images is *4.5e-3*.

We visually looked when this errors were made. We observed that they only occurred when the original image had high original chrominance values and our network predicted values from the other side of the color spectrum. In other words, we had to predict intense tons of colors and our autoencoder completely missed those. Moreover, this problematic error values are located in compact regions which contain objects with intense color tones.

We must say that, as clipping sets the values of a problematic region to the same clipped value, it will be easy for the PNG to compress these constant values thanks to the low entropy of this clipped problematic region. However, we believe that this clipping did not affect much to our given metrics. This is because the regions surpassing the $[-128, 127]$ bound are more or less constant as well because they share the same color tonality. They have low dynamic range, they have low frequencies and, after the PNG's filtering, they will have low energy. Nevertheless, it is fair to assume that our the metrics would be slightly worse. But, even if we had a a 25% decrease in compression rate in these affected images, as it happens only to a 0.9% of them, the compression gain would go from 530 to 510.

5.5.1 Possible Solutions

If nothing was changed in our framework and we sent the clipped image, we would be breaking the lossless constraint. The images sent would be the same except for color tones intensive which, occasionally, would have they energy lowered. The mean squared difference between the original image and the lossy one would be *4.5e-3*.

We propose an alternative solution to keep our algorithm lossless. For the chrominance values to clip, we will not transmit their error. Instead, we will transmit the original image's chrominance values. This means that we will have to send a mask indicating which pixels have been encoded using their error prediction and which have been encoded using their actual chrominance values. The mask would only needed to be sent when this problematic arose (0.9 % of the cases).

This mask will be expensive but not as expensive as one could initially think. Keep in mind that, it would only take a one channel image of 256×256 with values either 1 or 0 indicating whether we have a pixel encoded with its error or not. Moreover, the mask entropy would be really low since we have a Bernoulli of $\{p, 1-p\}$ where p is the probability that a pixel error has surpassed the $[-128, 127]$ bound. In our case, $p = 0.016$ and the entropy is 0.01. The entropy H of this Bernoulli distribution is $8.1e-2$. So we would expect to need $256 \times 256 \times H$ (5439.5 bits/ 679 bytes) to encode the mask if we used an entropic compression algorithm. We also believe that the mask entropy would be much lower since there exist a high spatial redundancy between the pixel positions were we exceed the bound.

Another approach that would encode these pixel positions is using bounding boxes. When we identify a region where we needed to encode their actual values, we would encode all the actual chrominance values of the box that contained the given region. For encoding a bounding box we only need 4 bytes (x, y, H, W) so it would be much cheaper to send it instead of the whole mask. However, we would be increasing the error image energy: low error predicted pixels that have been included to the box will have encoded their actual value.

In fact, we are increasing the image error energy and its frequencies in both approaches. Our error image is centered around 0 and we are introducing values that might be way above 0 considering that they probably belong to intensive tones. If so, we would expect a major decrease in our error compressing rate.

In the mask case, we could center the values to 0 by either substracting or adding 192 depending on which side of the bound we have exceeded. 192 is the mean value between 128 (by definition the exceeding values are above 128) and 256 (the maximum abolute error we can make). However, we would need an extra bit for every value within the mask to indicate whether we have subtracted or added 192. We would need three codewords: 0 for non-mask values, 10 for mask + 192 and 01 for mask -192 with respective probability of $\{1-p, \frac{p}{2}, \frac{p}{2}\}$. The entropy H of this probability distribution is 0.52. So the mask would have a cost of $256 \times 256 \times H$ (34734) bits (4341 bytes). Keep in mind that, as we commented earlier, that we believe that this value will be lower due to the mask spatial redundancy.

For the bounding box scenario, we could subtract the mean power from all the values contained in the box. This mean power could be encoded with one extra bit.

We think that in the case that just few regions exceeded the $[-128, 127]$ interval, the bounding box approach could outperform the mask one. Unfortunately, we did not have time to check all this corrections and we will present the test results on a subset of Imagenet without taking into account this error encoding issue and its possible corrections.

5.5.2 Effect on Bigger Images

As the image size increases, we can check in *Table 13* that this issue slowly vanishes. This means that the metrics extracted from larger image size have even less distortion. We believe that the problematic gets better with larger image size because, an image size increase introduces more spatial redundancy. The autoencoder will make less mistakes.

Table 13: Validation exceeding values information across image size.

Input Size	Images Affected	Average exceeding pixels in affected images	Reconstruction MSE wrt OG images
512	288 (0.8%)	1379 (0.5%)	2.1e-3
1024	305 (0.8%)	1195.2 (0.1%)	1.3e-3

5.6 Imagenet Testing

Once our best model has been found and analyzed, we will test how it works with a set of images from a different nature than the places365 standard. We will test the model in a subset of 20121 imagenet images of size $3 \times 256 \times 256$ which on average weight 11321 bytes when compressing with PNG and their average chrominance single pixel entropy is 4.9. When we upsample them to have $3 \times 512 \times 512$ size, they weight 11321 bytes and their average chrominance single pixel entropy is 4.9. When we upsample them to have $3 \times 1024 \times 1024$ size, they weight 87766 bytes and their average chrominance single pixel entropy is 4.9.

Table 14: Compression information with imagenet images of Tanh w/ 2 ch model and 128 quantization levels.

Input Size	Test MSE	Average error entropy	Gain w/o bn encoding	Gain wrt raw PNG (bytes)
256	57.3	4.2	600	488
512	38.0	3.9	2171	1723
1024	15.5	3.3	5015	3223

We can see in *Table 14* that in test we still keep positive compression rates. They are not as good as the ones obtained in validation because the autoencoder has not been trained in Imagenet images so it has not learned its specific characteristics. In *Table 15* we can observe that this error metrics are not really distortioned by the exceeding value issue: a more percentage of images are affected but very few pixels of them have exceeded the error bound. We conclude that our testing values are along the lines obtained in validation and no major unexpected issue has been found. Our testing was successful.

Table 15: Test exceeding values information across image size.

Input Size	Images Affected	Average exceeding pixels in affected images	Reconstruction MSE wrt OG images
256	330 (1.6%)	1060.1 (1.6%)	6.1e-3
512	404 (2.0%)	905.1 (0.3%)	3.9e-3
1024	450 (2.2%)	1402 (0.1%)	2.5e-3

6. Conclusions

In this project, we have built a lossless compression algorithm which tries to exploit images' color redundancy. To do so, we have used an autoencoder that encodes in its bottleneck the maximum color information as possible. A decoder could reconstruct the original image with this compressed color information and the image luminance. As this color information encoding does not fully contain all the original color information, the encoder also needs to send the color reconstruction error.

We have seen that exist a trade off between the amount of information compressed in this color encoding and the error energy needed to reconstruct the image. We have found that small bottlenecks are the ones that maximize the compression rates. The ratio we achieved is that for 512 pixels we only need 1 value in our bottleneck to encode their color information. However, this ratio does not include the luminance information, which we treated differently.

The method we have applied to encode the image error is quite interesting and can be further explored. It follows the intuition of decolorizing an image by extracting its maximum color information to then encode it. The decolorized image will have less energy and dynamic range so it should be less expensive to compress than the original image giving us a compression gain. We used the LAB color space chrominance components to extract the color information. The method has its strengths when the input image has very few colors, have large areas with the same tonality or the colors represented have not high tonalities. It struggles when we have really high tonalities or when we have lots of colors in small regions so that our bottleneck can not encode them all.

Our approach is quite flexible and can be implemented with different color transformations and different compression algorithms. Our approach could be implemented on top of any lossless compression algorithm. We benchmarked against the effectiveness of the PNG algorithm using also the PNG to compress our error. We could compare ourselves with better standards than PNG by also compressing our error image with the same standards. So, if an algorithm works better than PNG, it will improve the average compression rate of both original images and also the error image. Hopefully, the error images compression keeps having an edge over the original image compression.

We have showed that we can outperform the PNG standard by 5-7% across different image sizes. However, in a very few cases, our method breaks the lossless constraint since we can only encode error values from $[-128, 127]$ (dynamic range of the chrominance values of the LAB compression). We have analyzed the impact of this issue as well as proposed solutions to fix it.

Not everything is positive with respect to the PNG standard. By definition, our algorithm is slower than PNG because we have to do extra calculations to decolorize the image. After that, this image is PNG compressed so the total compression time would be color extraction + error generation + PNG compression. It basically sums up to autoencoder forward + PNG compression. The more complex our autoencoder is, the more time will be needed to compress images. Also, autoencoder weights occupy 513 MB in memory: the device used to compress images should have this amount of memory available. So if compression had to be done in real time, PNG would have a better performance than our approach.

Throughout this section we have implicitly revisited all of our objectives set in section 1.1. We believe we have successfully achieved all of them even though we would always desire higher bitrate gains.

6.1 Further Work

There are many things we can further explore and analyze. We think that one of the most important ones would be to compare our framework with different standards apart from PNG using them to encode our error image and to regularly encode original images. We believe that in many cases we should still be getting positive gains since the input image will always be decolorized. We think that the decolorization has properties that suit well many lossless compression algorithms.

Regarding the building of the autoencoder, there are a lot of hyperprameters that can be more exhaustively tuned. We have not studied at all parameters like batch size or learning rates. Also, we could change the network's architecture by increasing or decreasing the number of blocks or even changing the blocks themselves.

One of the aspects we struggled the most, was to find a clear correlation between the loss function picked (L_2) and the compression of the error generated by the output of the A-decoder and the ground truth. Even though we tried unsuccessfully other losses, we should keep working on solving this issue. Once we find a good loss function that clearly correlates with the error compression, we will be able to build a better autoencoder and therefore, we will obtain better error compression rates. We should also try to train our autoencoder with more images from different natures in order to help generalize the algorithm.

For the bottleneck compression, we have tested uniform quantization with different levels. We have already discussed that we could study the bottleneck values' statistics and implement a non uniform quantization. Apart from that, we discarded other compression techniques because our bottleneck had very few values. However, our model generalizes well with bigger images, thus bigger bottlenecks. We could do further research on compressing this bigger bottlenecks. Most probably, we will have some redundancies we could exploit.

Testing how using bounding boxes or masks to patch the exceeding error values would affect the system would be another field to explore. We think that we would find tradeoffs between this two techniques. Ultimately, if an image needed a mask or several bounding boxes to encode the error, we should use the technique that gave us lower compression weight.

Also, we could study how our proposed method could fit in a lossy constrain. It would be interesting to analyze the distortion impact when we do not add the reconstruction error. If it was too high we could

compress the error in a lossy manner and check how the rate/distortion of the error compression affects our global rate/distortion.

7. Acknowledgments

I would like to thank my supervisor Dr.Javier Ruiz for helping and guiding me in this project. At first, I was not sure about my TFG topic but after a short conversation with him, we rapidly came up to this coloration idea and plenty of its details. He has provided me tons of advice and improvements throughout the project and I am sure that I could not have got the results achieved without his feedback. Again, I am really thankful for his effort, help and support.

I would also like to thank Dr.Ferran Marquès who was my compression professor and the one who awoke my interest in multimedia encoding. He was the first person to whom I communicated my TFG ideas revolving around compression and the one who got me in touch with Javier.

Last but not least, I would like to thank my family for the support given in these last four years. Thanks to my friends and peers who have always been opened to listen my ideas and struggles during this project. Sometimes I was even the one who had to listen to their ideas and takes about my own TFG.

References

[1] Lilian Weng. From autoencoder to beta-vae, Aug 2018.

[2] Colt McAnlis. How png works, May 2016.

[3] L. Peter Deutsch. DEFLATE Compressed Data Format Specification version 1.3. RFC 1951, May 1996.

[4] Emilien Dupont, Adam Goliński, Milad Alizadeh, Yee Whye Teh, and Arnaud Doucet. Coin: Compression with implicit neural representations. *arXiv preprint arXiv:2103.03123*, 2021.

[5] Mu Li, Wangmeng Zuo, Shuhang Gu, Debin Zhao, and David Zhang. Learning convolutional networks for content-weighted image compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[6] Ionut Schiopu and Adrian Munteanu. Residual-error prediction based on deep learning for lossless image compression. *Electronics Letters*, 54(17):1032–1034, 2018.

[7] Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *CoRR*, abs/1601.06759, 2016.

[8] Richard Zhang, Jun-Yan Zhu, Phillip Isola, Xinyang Geng, Angela S. Lin, Tianhe Yu, and Alexei A. Efros. Real-time user-guided image colorization with learned deep priors. *CoRR*, abs/1705.02999, 2017.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[10] Mars Xiang. Convolutions: Transposed and deconvolution, Mar 2021.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.