Apunts de Machine Learning Image Compression

Sebastià Mijares i Verdú, GICI, UAB

Machine Learning Compression (MLC)

5 d'octubre de 2021

Notes on Machine Learning Compression

Machine Learning compression algorithms are generally based on a decorrelation-quatisation-entropy coding scheme, where decorrelation is done with an **Artificial Neural Network** (ANN). These networks are often **auteoncoders** or variants of such networks, and often use convolutional layers rather than conventional layers. We'll see more about them later on.

Artificial Neural Networks

Compression using autoencoders generally uses artificial neural networks, but this is not a closed case. There is a whole bunch of types of neural networks, in particular those used for autoencoders, depending on the operations their nodes and connections. Here we'll make note of the different types involved.

ANNs are in general composed by **nodes** (neurons), which have an input and a single output. These generally perform a *weighted sum* of all the inputs (the weights being parameters), add a *bias* (a constant, again a parameter overall), and pass the output through an *activation function* (generally non-linear, not a parameter) which may confine the output to a specific interval, or prevent the values from going very far.

These nodes may then produce inputs for nodes in the next **layer**. Neurons in ANNs are generally arranged into layers that operate sequentially. We have an *input layer* at the start, an *output layer* at the end, and a number (0 or more) of *hidden layers*. These layers may be fully connected to the ones before or after, and there may be **pooling**, where a group of neurons in one layer connect to a single neuron in the next layer, thereby reducing the number of neurons in that layer.

Networks that form an acyclic graph are called **feedforward (FF)** networks. Those with at least one cycle or loop are called **recurrent** networks.

Universal approximation theorems

We can approach these theorems in two ways: we can consider either a three-layer network with an arbitrarily large number of nodes in the hidden layer, or by considering a fixed number of nodes arranged in an arbitrary number of layers.

Definition: *A function* $\sigma : \mathbb{R} \to \mathbb{R}$ *is* ***sigmoidal*** *if it's continuous and:*

$$\sigma(t) \to \begin{cases} 1 & t \to +\infty \\ 0 & t \to -\infty \end{cases}$$

Theorem (Cybenko[1]): *Let* $\sigma : \mathbb{R} \to \mathbb{R}$ *be a sigmoidal function, and let* $f : \mathbb{R}^n \to \mathbb{R}^m$ *be a continuous function (our target function). Then, for every* $\epsilon > 0$ *and every compact subset* $K \subset \mathbb{R}^n$, *there exist affine functions* $F_1 : \mathbb{R}^n \to \mathbb{R}^D$ *and* $F_2 : \mathbb{R}^D \to \mathbb{R}^m$ *such that, defining* $f_\epsilon = F_2 \circ \sigma \circ F_1$*:*

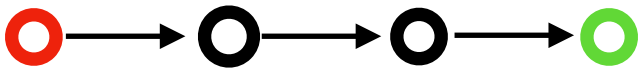
$$sup_{x \in K}\{\, |f(x) - f_\epsilon(x)| \,\} < \epsilon$$

The result regarding arbitrary layers has a more complex statement, and is less interesting given that a neural network's complexity increases exponentially with the number of layers, but polynomially over the number of nodes.

In short, we can approximate an arbitrary function using a three-layer setup for our ANN, given a large enough number of nodes. In the case of autoencoders, this translates into a five-layer overall setup, as we'd be approximating two different arbitrary functions (the encoder and the decoder)[2]. The **parameters** for this ANN will be the coefficients that define those affine functions: the entries of the matrix (linear part, often called the *weights*), and the entries of the summing vector (affine part, known as the *bias*).

Backpropagation

Backpropagation is the recursive algorithm by which we can compute the gradient of the cost function with respect to the parameters (weights and bias) of the ANN. The gradient of the cost function is key for the training of the ANN we'll explore later on. The mani idea behind backpropagation is the *chain rule* from calculus, which is recursively applied from the output layer (ground case) towards the input layer.



We'll start with a basic example, with two hidden layers and one node per layer:

---

[1] Cybenko, G. (1989) "Approximation by superpositions of a sigmoidal function", *Mathematics of Control, Signals, and Systems*, 2(4), 303–314.

[2] Needs further clarification.

We'll call our nodes values $a^{(i)} : i \in \{0,1,2,3\}$, the weights $w^{(i)} : i \in \{1,2,3\}$, and the biases $b^{(i)} : i \in \{1,2,3\}$. We have that for each node:

$$a^{(i)} = \sigma(w^{(i)}a^{(i-1)} + b^{(i)}) = \sigma(z^{(i)})$$

Where $z^{(i)} : i \in \{1,2,3\}$ is the affine component. Let $C : \mathbb{R} \to \mathbb{R}$ be our cost function we wish to minimise. Then, at the last layer we have that:

$$\frac{\partial C}{\partial w^{(3)}} = \left(\frac{\partial z^{(3)}}{\partial w^{(3)}}\right)\left(\frac{\partial a^{(3)}}{\partial z^{(3)}}\right)\left(\frac{\partial C}{\partial a^{(3)}}\right) = \left(a^{(2)}\right)\left(\sigma'(z^{(3)})\right)\left(\frac{\partial C}{\partial a^{(3)}}\right)$$

$$\frac{\partial C}{\partial b^{(3)}} = \left(\frac{\partial z^{(3)}}{\partial b^{(3)}}\right)\left(\frac{\partial a^{(3)}}{\partial z^{(3)}}\right)\left(\frac{\partial C}{\partial a^{(3)}}\right) = (1)\left(\sigma'(z^{(3)})\right)\left(\frac{\partial C}{\partial a^{(3)}}\right)$$

$$\frac{\partial C}{\partial a^{(2)}} = \left(\frac{\partial z^{(3)}}{\partial a^{(2)}}\right)\left(\frac{\partial a^{(3)}}{\partial z^{(3)}}\right)\left(\frac{\partial C}{\partial a^{(3)}}\right) = \left(w^{(3)}\right)\left(\sigma'(z^{(3)})\right)\left(\frac{\partial C}{\partial a^{(3)}}\right)$$

With the derivative values from the last layer, we can compute the derivative values of the previous (mainly with $\frac{\partial C}{\partial a^{(2)}}$):

$$\frac{\partial C}{\partial w^{(2)}} = \left(\frac{\partial z^{(2)}}{\partial w^{(2)}}\right)\left(\frac{\partial a^{(2)}}{\partial z^{(2)}}\right)\left(\frac{\partial C}{\partial a^{(2)}}\right) = \left(a^{(1)}\right)\left(\sigma'(z^{(2)})\right)\left(\frac{\partial C}{\partial a^{(2)}}\right)$$

$$\frac{\partial C}{\partial b^{(2)}} = \left(\frac{\partial z^{(3)}}{\partial b^{(2)}}\right)\left(\frac{\partial a^{(2)}}{\partial z^{(2)}}\right)\left(\frac{\partial C}{\partial a^{(2)}}\right) = \left(\sigma'(z^{(2)})\right)\left(\frac{\partial C}{\partial a^{(2)}}\right)$$

$$\frac{\partial C}{\partial a^{(1)}} = \left(\frac{\partial z^{(2)}}{\partial a^{(1)}}\right)\left(\frac{\partial a^{(2)}}{\partial z^{(2)}}\right)\left(\frac{\partial C}{\partial a^{(2)}}\right) = \left(w^{(2)}\right)\left(\sigma'(z^{(2)})\right)\left(\frac{\partial C}{\partial a^{(2)}}\right)$$

With this, we can compute $\frac{\partial C}{\partial w^{(i)}}$ and $\frac{\partial C}{\partial b^{(i)}}$, derivatives with respect to our parameters, from a given input $a^{(0)}$ and its output $a^{(3)}$, at any particular current value of the

parameters. Of course, we must also know the cost function itself explicitly. For example, in an autoencoder setup, the cost function could be the squared difference $C(a^{(L)}, a^{(0)}) = (a^{(0)} - a^{(L)})^2$, so $\frac{\partial C}{\partial a^{(L)}} = 2(a^{(L)} - a^{(0)})$ would be the partial derivatives that yield the final layer's gradient.

These formulas work analogously when we have an arbitrary number of layers and nodes in each layer. Let us have L layers, with n_L nodes in each layer. Let $a_i^{(k)}$ be the *i*-th node of the *k*-th layer, and let $W^{(k)} = (w_{i,j}^{(k)}) \in Mat_{n_k \times n_{k-1}}(\mathbb{R})$ be the weights, so that:

$$a^{(k+1)} = \sigma\left(W^{(k)} a^{(k)} + b^{(k)}\right) = \sigma\left(z^{(k)}\right)$$

Our derivatives will be:

$$\frac{\partial a_i^{(k)}}{\partial z_j^{(k)}} = \begin{cases} 0 & i \neq j \\ \sigma'(z_i^{(k)}) & i = j \end{cases}$$

$$\frac{\partial z_l^{(k)}}{\partial w_{i,j}^{(k)}} = \begin{cases} 0 & l \neq j \\ a_i^{(k-1)} & l = j \end{cases}$$

$$\frac{\partial z_i^{(k)}}{\partial b_j^{(k)}} = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}$$

$$\frac{\partial z_i^{(k)}}{\partial a_j^{(k-1)}} = w_{j,i}^{(k)}$$

And with these, recursion can be computed as in the one node per layer case, finding the gradient ∇C with respect to the parameters at a given point (value of the parameters) for each training example.

Training

The training of ANNs can be simply understood as a practical minimisation problem: we have a cost function we wish to minimise, which depends on a series of para-

meters (the weights and biases). The problem is that the number of parameters (dimensions) is probably very large, especially once images are involved, and the cost function is generally not known explicitly, at least not right away.

Consider our autoencoders, for example. Their cost function is that the input should be as similar as possible to the output. We can write that principle as $C(x) = \|x - \varphi(x)\|^2$. This is an example of **unsupervised learning**, as we can compute the cost function for any input. In other instances, the value of the cost function cannot be written explicitly: whether an image contains a cat or not, for instance, is a tag that has to be attached to the input image by a human manually. We call this **supervised learning**.

In either case, we study the cost function, not in terms of the input values, but in terms of our parameters: the ANN is the input of our cost function we actually care about! However, computing the gradient of the cost function with respect to the ANNs parameters is much much harder, especially since we can only know the value of C and ∇C at specific points given by the ANNs input. Therefore, we have to approximate ∇C at a current value of the parameters, with respect to one (or rather a number of) inputs to the ANN, using backpropagation, for example.

As in other minimisation problems, the negative gradient $-\nabla C$ indicates the local direction of descent. We then have to take a step in that direction. Steps are generally of a fixed length or fixed proportion of the gradient. This is referred to as the **learning rate**: a smaller learning rate may improve local performance and find a lower minimum, but it takes longer to get there, especially from the start.

The basic traditional algorithm is **stochastic gradient descent (SDG)**, which is essentially making steps in the direction and module of the gradient multiplied by a fixed learning rate, $-\alpha \nabla C$, where the gradient is calculated as a mean over a batch of samples. Several variations and implementations of this basic approach have been proposed, notably Adam[3], which implements an adaptive gradient algorithm with different learning rates per parameter, which is adapted based on the first and second moment (mean and mean of the squares) of the gradients in a batch.

---

[3] Diederik P. Kingma, Jimmy Ba (2015). "Adam: a method for stochastic optimisation", ICLR 2015.

Convolutional Neural Networks (CNNs)

The traditional mathematical operator of convolution is an operation applied to two functions, using an integral transform, but this is not what the "convolutional" here stands for. The **convolution** is an operation between two *d*-dimensional arrays of different sizes, whose output is a single real number. Let $I \in Mat_{N\times M}(\mathbb{R})$ and $K \in Mat_{n\times m}(\mathbb{R})$ be two 2-dimensional arrays (for simplicity), with $n < N$, $m < M$. Let $(i, j) \in [N] \times [M]$ be a pair of coordinates in I. The **convolution** of I at the position (i, j) using **kernel** K is defined as:

$$C_{i,j,K}(I) = \frac{\sum_{x=1}^{n}\sum_{y=1}^{m} K_{x,y} I_{i-\lfloor \frac{x}{2} \rfloor, j-\lfloor \frac{y}{2} \rfloor}}{S(K)}$$

Where $S(K)$ is equal to the sum of the entries of K or 1 if that sum is 0.

Essentially, it's taking the submatrix of I centered at (i, j) of size $n \times m$, computing the dot product with K, and dividing by the sum of the entries of K (when possible).

This operation is clearly badly defined for **pixels on edges** (or close enough to edges). We have several alternatives for this:

- Wrapping the image, so it's a flat torus.
- Ignoring edge pixels and only apply convolution to those for which the operation is well defined.
- Filling the "missing" outside pixels with copies of the edge pixels.
- Missing pixels are all equal to 0.

The last option is by far the most popular. The definition we provided explicitly is for 2-dimensional arrays, but you can clearly see that it would be defined the same way for a **larger number of dimensions**. In the case of images, 2 and 3 dimensions are the most interesting cases, counting either an image as a 2D array of integers, or as a 3D array with two spatial coordinates, and one for colour components.

Given an array I (an image) and a kernel K, the **convolution of the image by the kernel** is usually defined as the application of the convolution operation to each pixel in

the image, resulting in another array of the same dimension. This can be done in many ways, so the size of the resulting array may vary to any size smaller or equal to the original. This reduction in size may be achieved in many ways, such as:

- Ignoring edge pixels or a band of edge pixels.
- Not applying convolution to every pixel in I, but instead taking steps longer than 1 pixel.

These are the main ideas behind **Convolutional Neural Networks** (CNNs): taking an array as input, and convolving it by a series of kernels (also known as *filters*). The change between one layer and the next is defined by a single kernel, with variable coefficients, that is applied to that entire layer. The parameters of these ANNs are, therefore, the kernels' coefficients. As in traditional ANNs, an activation function is used, generally the **ReLU** function:

$$ReLU(x) = \begin{cases} x & \forall x \geq 0 \\ 0 & \forall x < 0 \end{cases}$$

Training is equivallent for CNNs as in other traditional neural networks. Here, our parameters are the kernels' entries, and we will compute $-\nabla C$ with respect to those parameters. To find such gradient, we can use **backpropagation**, though the formulas for this backpropagation are slightly different. For simplicity, suppose all the layers are square 2D arrays. Let σ be the activation function, let N_l be the number of rows of each layer, and, for simplicity, suppose our kernels are all 3x3, so $N_{l+1} = N_l - 2$ (we're applying the kernel only to those pixels for which the full convolution is well defined). Let $A_l = (a_{i,j}^{(l)})$ be the layers' nodes, and let $K_l = (k_{i,j}^{(l)})$ be the kernels. Then:

$$a_{i,j}^{(l)} = \sigma(z_{i,j}) = \sigma\left(\frac{\sum_{x,y=1}^{3} k_{x,y}^{(l)} a_{i+x-1,j+y-1}^{(l-1)}}{S(K_l)}\right)$$

Therefore, for our backpropagation, we have the formulas:

$$\frac{\partial C}{\partial k^{(l)}_{x,y}} = \sum_{i,j=1}^{N_l} \left(\frac{\partial C}{\partial a^{(l)}_{i,j}}\right)\left(\frac{\partial a^{(l)}_{i,j}}{\partial z^{(l)}_{i,j}}\right)\left(\frac{\partial z^{(l)}_{i,j}}{\partial k^{(l)}_{x,y}}\right) = \sum_{i,j=1}^{N_l} \left(\frac{\partial C}{\partial a^{(l)}_{i,j}}\right)\sigma'(z^{(l)}_{i,j})\left(\frac{a^{(l-1)}_{i+x-1,j+y-1}S(K_l) - \sum_{x,y=1}^{3} k^{(l)}_{x,y}a^{(l-1)}_{i+x-1,j+y-1}}{S(K_l)^2}\right)$$

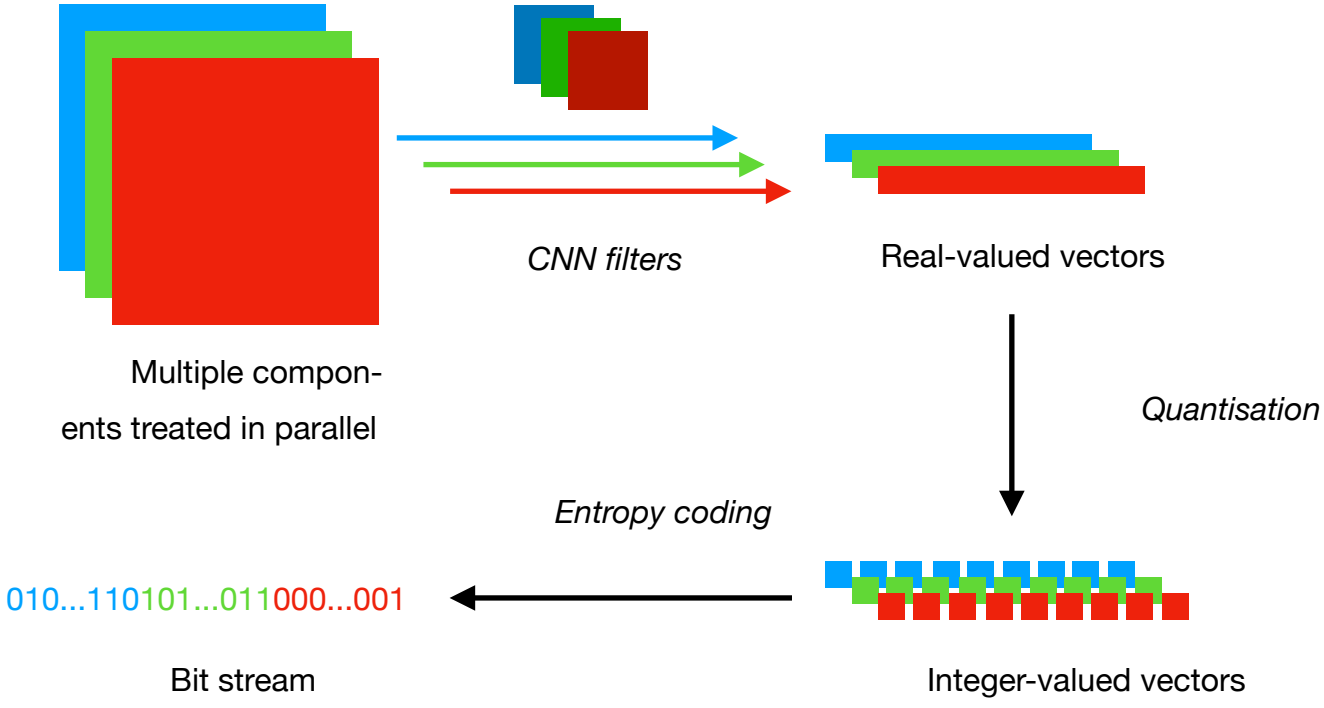
$$\frac{\partial C}{\partial a^{(l-1)}_{p,q}} = \sum_{i,j=1}^{N_l} \left(\frac{\partial C}{\partial a^{(l)}_{i,j}}\right)\left(\frac{\partial a^{(l)}_{i,j}}{\partial z^{(l)}_{i,j}}\right)\left(\frac{\partial z^{(l)}_{i,j}}{\partial a^{(l-1)}_{p,q}}\right) = \sum_{x,y=1}^{3} \sum_{p+1-x,q+1-y=1,\dots,N_l} \left(\frac{\partial C}{\partial a^{(l)}_{p+1-x,q+1-y}}\right)\sigma'(z^{(l)}_{i,j})\left(\frac{k^{(l)}_{x,y}}{S(K_l)}\right)$$

Note the complicated summing indices in the second function are only to ensure good definition of the terms.

Non-linear Transform Coding (NTC)

Architecture

Non-linear Transform Coding (NTC) is a family of methods that use CNNs for compressing images. They rely on a multistage architecture, following the traditional structure of **decorrelation**, **quantisation**, and **entropy coding**[4]. The following diagram illustrates the overall architecture of NTC.



First, the image, of arbitrary size and number of components, is fed into a CNN. This CNN outputs an array of real numbers (a vector), which is then quantised, and en-

[4] J. Ballé *et al*, (2020) "Nonlinear Transform Coding", *Google Research*.

tropy coded. Note this vector will be of *variable* length[5], as well as the entropy codes applied to it.

The CNN is generally an **autoencoder**, or some variant of an autoencoder (variational, with hyperprior, etc.), with an encoder and a decoder parts, which are trained together (see the following section) but operate independently of each other.

Different authors use different structures for the CNN. Some treat each component separately with different filters[6]. Others[7] studied other architectures with a varying number of components per layer.

Decorrelation

As described earlier, decorrelation in NTC is done using an autoencoder CNN. This autoencoder will output an array (a vector) of real values that will be our decorrelated data. There is no partitioning of the image: the output data is integrated for the particular image, and has global effects.

The autoencoder has two goals to achieve: first is to deconstruct and reconstruct a vector (image) as accurately as possible using the smallest possible latent representation (or more realistically, using the given smaller dimensionality of the latent representation). Second, it has to produce a codebook of quantised vectors in the latent space for that decompression, which in turn should have low entropy, so that when applying lossless coding to the encoded representation we can compress the furthest. In short, **quantisation is not generally incorporated in the autoencoder's training**, thus decorrelation and quantisation take place in the same system.

To achieve these two objectives, low distortion and low entropy, the target function the autoencoder is trained to attain combines them with a tradeoff parameter. Let S be our source of vectors (images), x. Let $\varphi(x)$ be the encoder function, $\psi(x)$ be the de-

---

[5] Convolution applies the same filter to any size of the array, with its output being of variable size as well.

[6] J. Ballé, V. Laparra, E. Simoncelli, (2017) "End-to-end optimised image compression", *ICLR Conference*.

[7] N. Johnston, *et al.*, (2019) "Computationally efficient neural image compression", *Google Research*.

coder function, and $Q(x)$ be the quantiser function of our autoencoder. We have the autoencoder performs:

$$x \mapsto \psi(Q(\varphi(x)))$$

Let $Pr(\cdot)$ denote probability, and let $H_r(S)$ be the Shannon entropy of a source. The autoencoder's target function will be[8]:

$$L(\varphi, \psi, Q) = \mathbb{E}_{x \in S}[-log(Pr(Q(\varphi(x))) + \lambda d(x, \psi(Q(\varphi(x)))]$$

$$L(\varphi, \psi, Q) = H_r(Q(\varphi(x))) + \lambda \mathbb{E}_{x \in S}[d(x, \psi(Q(\varphi(x)))]$$

Where λ is the tradeoff parameter, the entropy of a vector is and d is some distance function between the vectors. Note we take the logarithm, as it best approximates the optimal lossless encoding average word length. Therefore, we'll attempt to minimise this weighted combination of average word length and distortion for a given parameter λ.

Quantisation

Non-linear Transform Coding generally uses quantisation by **rounding to integers** in the latent representation. In other words, the encoder maps the input vector $x \in \mathbb{R}^n$ to the latent space $\varphi(x) \in \mathbb{R}^m : m < n$, which is then quantised to the nearest integer $[\varphi(x)] \in \mathbb{Z}^m$ and then the decoder part of the autoencoder maps it back to a *real* vector.

This is not necessarily the setting for the network's training[9]. There, soft quantisation (such as mapping by the identity or some more sophisticated function) may be applied or uniform noise, in order to make the target function, $L(\varphi, \psi, Q)$, differentiable. This is key, as that is necessary to compute the gradient and thus the minimisation path of the network.

---

[8] L. Theis *et al.*, (2017) "Lossy image compression with compressive autoencoders". *Twitter Research*.

[9] J. Ballé *et al*, (2020) "Nonlinear Transform Coding", *Google Research*.

Training

As described in the decorrelation section, the target function of the autoencoder is $L(\varphi, \psi, Q) = H_r(Q(\varphi(x))) + \lambda \mathbb{E}_{x \in S}[d(x, \psi(Q(\varphi(x))))]$. Since quantisation is not differentiable, for training the function used is the following continuous relaxation:

$$L(\varphi, \psi) = H_r(\varphi(x)) + \lambda \mathbb{E}_{x \in S}[d(x, \psi(\varphi(x))]$$

That is, replacing quantisation with the identity. With this, we can compute the gradient $-\nabla L$, and apply stochastic gradient descent (SDG) for training. To approximate the expected value, we can use the mean of a batch of training samples.

Autoencoders (AE)

These are ANNs whose input and output layers are identical, and the objective function to be minimised is their difference (in other words, they aim to create a copy of the input), but the central hidden layer, called the **latent representation** layer, is smaller that the input and output. In short, autoencoders create a bottleneck, and the network has to pass the information as similar to the original as possible using fewer bits in the process.

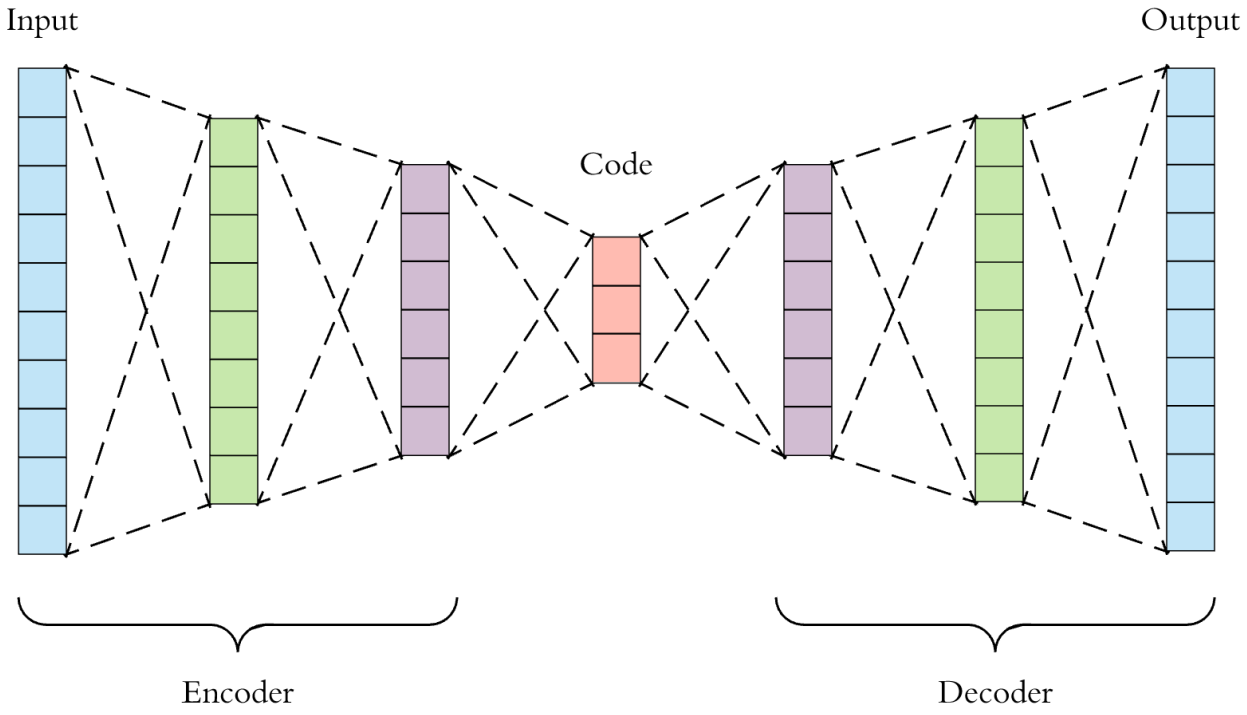


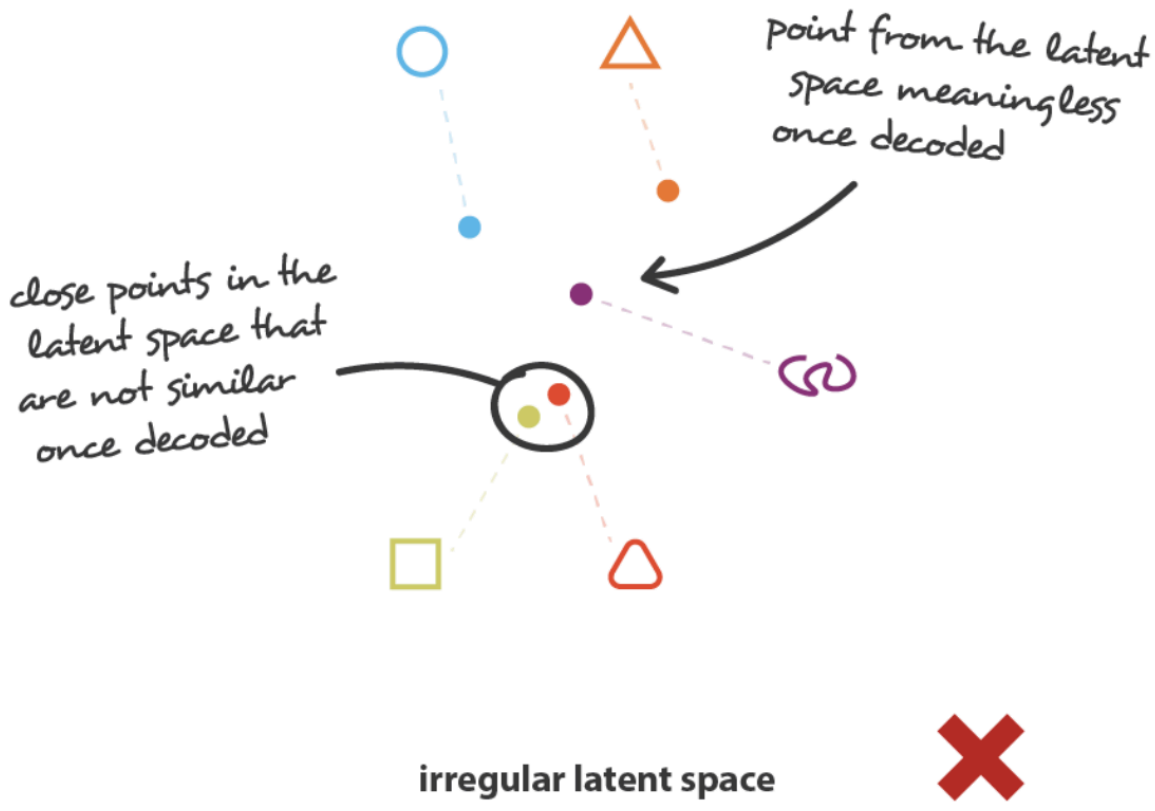
Figure 1: General structure of an autoencoder RNN.

These networks produce an encoder and a decoder (the two halves of them), and may perform *decorrelation* and *quantisation* simultaneously. The output code may then be further treated using some entropy coding algorithm (say, Huffman or arithmetic coding).

The cost function includes the distortion of the output with respect to the input, $d(x, \hat{x})$, and may also include some component regarding the latent representation (such as its entropy after quantisation).

Problems

Autoencoders in their basic form have a problem with the continuity of the latent representation. Latent vectors that are *close* (with respect to euclidean distance) may be decoded into completly different outputs, or even meaningless outputs. This is referred to as an **irregular latent space**, and is particularly troublesome in our setup after we apply quantisation.



These issues require a regularisation of the latent space, which is often achieved by introducing **variational autoencoders**. Another less popular approach is simply learning a model (see next subsection).

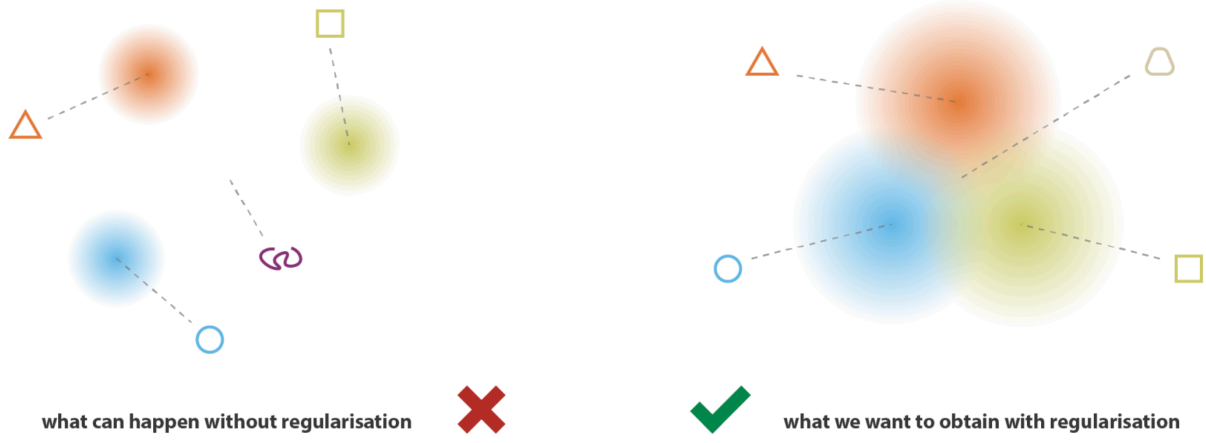
Probability model learning

A simpler alternative to variational autoencoders (VAEs) is learning a probability model[10]. In this approach, the autoencoder's loss function has two parts, as earlier: rate (logarithm), and distortion (mean squared error or other metric). As earlier, the input is mapped through a bottleneck (latent representation) and then reconstructed out of it. However, in this system, we have a model probability distribution we aim to fit our latent representation into (of our choice).

Unlike in VAEs, the AE isn't learning to fit the latent representation into a distribution. Instead, the model learns to deconstruct and reconstruct like a standard AE, but the rate term is learned from the probability distribution, so the probabilities used are those the symbols would have if they came from said model. Further, the model's parameters (such as standard deviation) may be adjusted empirically during training.

Variational autoencoders (VAE)

Variational autoencoders are an improvement upon basic autoencoders that introduce the idea of encoding a probability distribution. This is done in order to achieve some latent space regularisation.



[10] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár (2017) - "Lossy image compression with compressive autoencoders", ICLR 2017.

Instead of approximating a *feature space* in our latent space, a VAE will approximate a paramterised probability distribution, such as a multivariate Gaussian distribution (or any other deemed appropriate). For the sake of explanation, we'll focus only on the Gaussian distribution[11], but the choice of one distribution or another is entirely up to the developer.

A multivariate Gaussian is defined by two sets of parameters: a mean vector, μ, and a covariance matrix Σ. We may further assume that the different dimensions of our Gaussian are independent, so Σ is a diagonal matrix, and we may simplify it to a standard deviation vector: σ.

Our encoder will now not just map an input to a latent representation, $x \mapsto z$, but also to a mean and standard deviation vectors, $x \mapsto z \mapsto (\mu, \sigma)$. These define our underlying probability distribution. Then (during training) we sample a random latent vector from that distribution, and map it through the decoder, $z \mapsto \hat{x}$. This clearly ensures a regularisation of the latent space, as we have both spatial continuity and we fill all space with some meaningful output.

We have two issues to address now:

1. The sampling layer before the decoder blocks backpropagation.

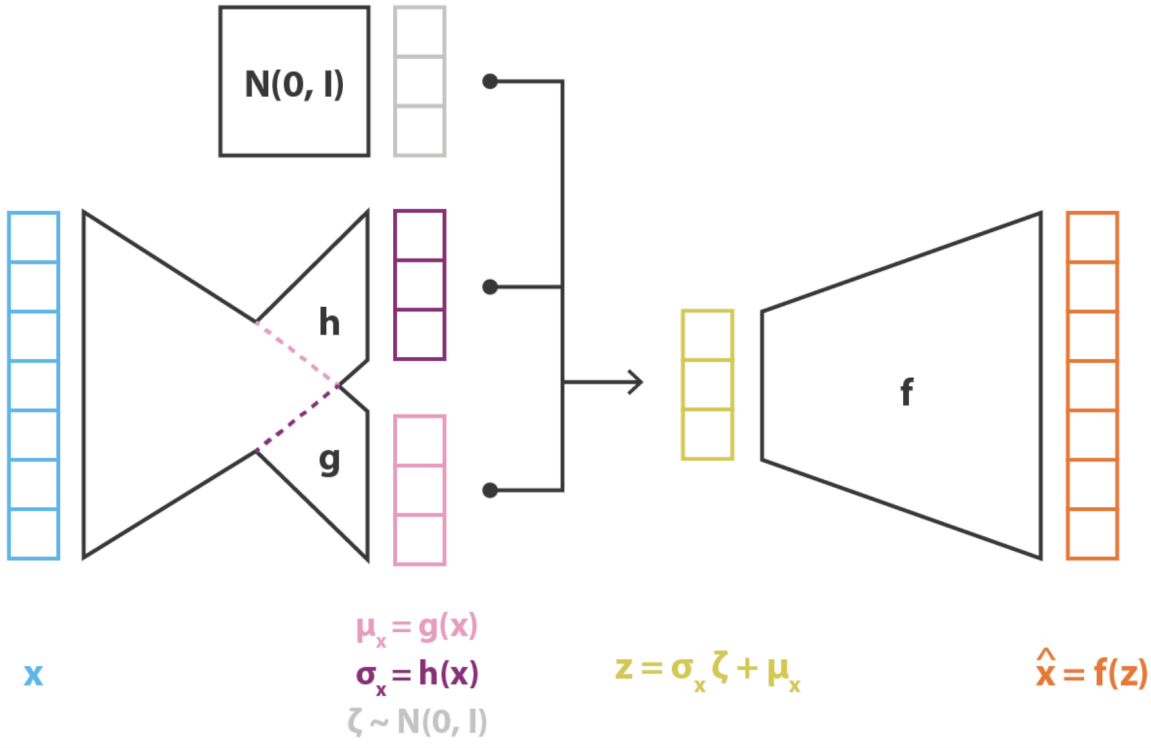
2. Given a random sampled z in the latent space, what is the corresponding input x that maps to it, so we can compute $d(x, \hat{x})$?

For 1. we can use a **reparametrization** of our architecture. For 2., we can use Bayesian inference.

Reparametrization

To overcome the block to backpropagation that our sampling layer imposes before the decoder, we can change the parameters upon which the sampling is done, so we randomly select from a fixed distribution. In our case, using a Gaussian, we may

[11] https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73



pick an $\epsilon \sim N(0,1)$[12] from a normal (0,1) distribution, and write $z = \mu + \sigma\epsilon \sim N(\mu, \sigma)$, which is distributed as our intended Gaussian. Although this is a stochastic parameter, we may treat it as a constant in backpropagation, thus we can continue to differentiate past this layer.

Bayesian inference of the input

This setup change modifies how we introduce distortion in our cost function. Instead of considering the encoder and decoder as parametric functions, we may see them as a parametric probability distributions: and **encoder distribution** $p(z \mid x)$, which is given by the probability of obtaining some latent sepresentation from a given input, and a **decoder distribution** $p(x \mid z)$, the probability of reconstructing an input out of a given latent representation.

Using Bayes' rule, we get:

$$p(z \mid x) = \frac{p(x \mid z)p(z)}{p(x)}$$

[12] This (0,1) normal distribution has the same dimensionality as our distribution, and the product of the two vectors will be component by component. In effect, we may see this as picking a different epsilon for each component.

Where we keep in mind that $p(z)$ is the prior distribution we fixed beforehand (Gaussian (0,1) distribution). The $p(x \mid z)$ distribution is also a Gaussian, given by our parametrised mean and standard deviation, and some deterministic function (the decoder, ψ), so $p(x \mid z) \equiv N(\psi(z), cI_n) : c > 0$. However, in general we don't know the decoder function (it's what we're computing), and $p(x)$ is also intractable, so we need a more sophisticated approach.

We may use **variational inference**, and approximate $p(z \mid x)$ with a Gaussian distribution whose mean and standard deviation vectors are given by two functions (the ones previously discussed in our encoder). To compute how far $p(z \mid x)$ is from a normal $N(\mu_x, \sigma_x)$ distribution, with our deduced mean and variance, we use the **Kullback-Leibler (KL) Divergence**, an operator on probability distributions, which we can calculate from our input and outputs of the overall VAE.

Overall, if the VAE is given by the encoder parametric function $\phi(x) = (\mu_x, \sigma_x)$ and the decoder parametric function $\psi(z)$, our **cost function** will be:

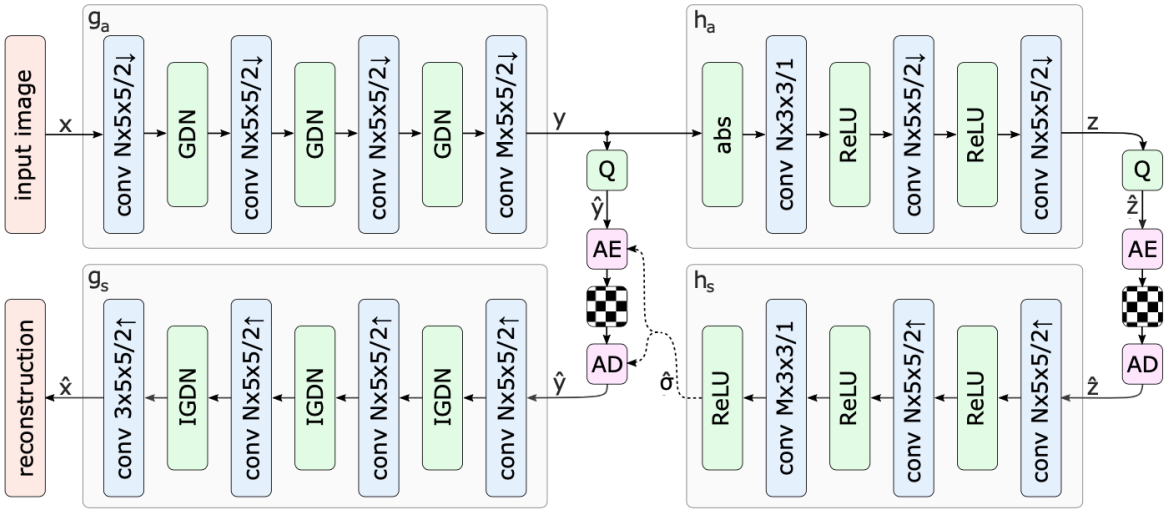
$$\mathcal{L}(\phi, \psi, x) = KL\left(N(\mu_x, \sigma_x), N(0,1)\right) + \lambda \mathbb{E}_{z \sim N(\mu_x, \sigma_x)}\left(\|x, \psi(z)\|^2\right)$$

Where λ is a constant parameter resulting from the choice of c, the standard deviation in $p(x \mid z) \equiv N(\psi(z), cI_n) : c > 0$.

Hyperprior

Ballé *et al.* introduced the idea of a **hyperprior** to improve the VAE compression performance[13]. This is an additional ANN that provides side information from the latent representation to encapsulate spatial differences. For example, pixels *close together* tend to have *similar* values, thus local standard deviation may differ from the overall standard deviation in the entropy model. The hyperprior then deduces the standard deviation correction for the specific values, used then for the distribution of the latent

[13] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston (2018). "Variational image compression with scale hyperprior", Conference Paper, ICLR 2018.



Example of VAE with hyperprior (J. Ballé, 2018).

space[14]. In this approach, som authors have found the Laplacian distribution to outperform the Gaussian distribution.

Some authors[15] use simpler architectures for the hyperprior, such as a simple variance estimator to save on complexity, epecially in running. Other authors[16] have tried expanding the complexity of the hyperprior, by adding other systems such as an autoregressive context model (essentially, a learned stochastic affine combination of the previous *k* samples). The context model achieved a slight improvement on the performance of the compressor (in terms of bit rate and distortion), but at a very large computational cost. Katto *et al.* showed that the performance of the compressor doesn't increase indefinitely with an ever more complex hyperprior, and that a "small" hyperprior will achieve better results than a "larger" one[17].

---

[14] L. Zhou, C. Cai, Y. Gao, S. Su, J. Wu (2018). "Variational autoencoder for low bit-rate image compression", IEEE.

[15] Vinicius Alves de Oliveira, Marie Chabert, Thomas Oberlin, Charly Poulliat, Mickael Bruno, Christophe Latry, Mikael Carlavan, Simon Henrot, Frederic Falzon, and Roberto Camarero (2021). "Reduced-Complexity End-to-End Variational Autoencoder for on Board Satellite Image Compression", MDPI.

[16] D. Minnen, J. Ballé, and G. D. Toderici (2018). "Joint autoregressive and hierarchical priors for learned image compression", 2nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

[17] Z. Cheng, H. Sun, M. Takeguchi, J. Katto (2019). "Deep Residual Learning for Image Compression", CVPR 2019.

The hyperprior approach works by updating the probability distribution under which the latent vectors are encoded in real time. It's important to remember here that the arithmetic coder doesn't measure the current probabilities of the sent symbols; it instead assumes the probabilities from a predefined source (the prior, generally Gaussian or Laplacian) and works out the code from there. Therefore, when the hyperprior sends an improved version of the standard deviation, for example, adapted to the specific sent image, those codeword probabilities are improved, and thus compression more closely approaches the optimal.

Generative Adversarial Netowrks (GAN)

Some authors have approached the machine learning image compression problem using GANs. Some, have introduced a discriminator in the VAE compression architecture, together with other features like a hyperprior, channel-spatial attention blocks[18], or an importance map[19], to improve the reconstructed image's quality. This adversarial network therefore trains the VAE as the generator part (in particular, the decoder), and

(a) Proposed method without adversarial mechanism, 0.169bpp

(b) Proposed method, 0.174bpp

adds a binary discriminator with pairs of real or fake input + output. Results of this approach can yield very slightly improved details[20] at the cost of slight loss of compres-

[18] Jiayu Yang, Chunhui Yang, Yi Ma, Shiyi Liu, Ronggang Wang (2020). "Learned Low Bit-Rate Image Compression With Adversarial Mechanism", CVF.

[19] Lirong Wu, Kejie Huang, Haibin Shen (2020). "A GAN-based Tunable Image Compression System", CVF.

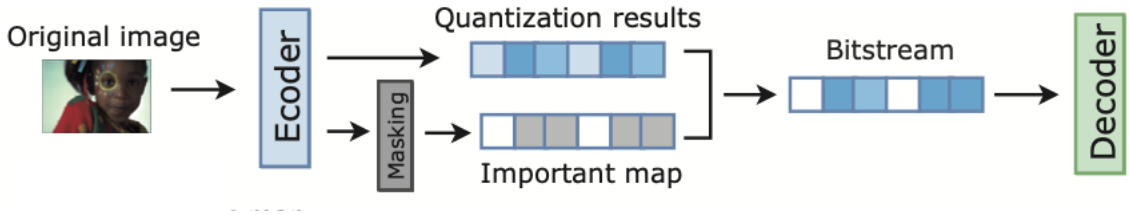
[20] Jiayu Yang, Chunhui Yang, Yi Ma, Shiyi Liu, Ronggang Wang (2020). "Learned Low Bit-Rate Image Compression With Adversarial Mechanism", CVF.

sion rate, however some authors claim their results slightly outperform in quality other lossy compressor paradigms with better compression ratios[21].

Frequency split

Another approach to improved quality in lossy compression is to split the input pixels into high and low frequency parts, where the high frequency elements are encoded at a higher bit rate (higher quality) and low frequency regions at a lower bit rate (lower quality). On such attempt also used GANs in the training[22], but first filtered the image through a filter (they called it *masking*) which indicated the "important" regions of the latent representation to be enhanced in the final bitstream.

In this case (Wu, Huang, and Shen), the masking is done by an additional CNN that is trained to enhance some regions of the image to improve the quality results.



Architecture of the L.Wu, K. Huang, and H. Shen compressor

Another approach is to split the image by channels rather than by spatial regions, treating the two inputs as two semi-independent contents[23]. Here, the high-frequency part takes a portion of the channels in full size, and the low-frequency part takes the rest of the channels with a stride of 2 (half the spatial resolution). They then compress these using a VAE with a hyperprior, which in turn outputs two values (one for the high-frequency distribution, and one for the low-frequency distribution).

---

[21] Fabian Mentzer, George Toderici, Michael Tschannen, Eirikur Agustsson (2020). "High-Fidelity Generative Image Compression", Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

[22] Lirong Wu, Kejie Huang, Haibin Shen (2020). "A GAN-based Tunable Image Compression System", CVF.

[23] Mohammad Akbari, Jie Liang, Jingning Han, Chengjie Tu (2020). "Generalized octave convolutions for learned multi-frequency image compression", SFU, Canada.

Subsampling in YUV

Most practical state-of-the-art compressors for images and video (such as HEVC and VVC) use the **YUV scale** rather than the RGB representation, which in turn is usually **subsampled**: rather than taking 3 bands of equal dimensions, $n \times n$, we take the luminescence band as $n \times n$, and the chrominance components (U and V) as $\frac{n}{2} \times \frac{n}{2}$, thus reducing significantly the number of samples to encode right off the bat. This is due to the fact that human vision perceives a lot less strongly changes in chrominance than in luminance, hence the resulting images look almost the same.

Following this principle, authors Egilmez *et al.* designed and trained networks using the YUV scale rather than the RGB scale[24]. Since two of the components are of a different size than the first one, their networks required an architecture adaptation. Several of these were tested, finally proposing one whose performance was on par with the HEVC and VVF standards on both video and images coding.

Lossless compression

Lossless compression using machine learning is usually approached using the traditional predictor transform and then an entropy coder. There are several proposed models in this regard producing promising results.

Predictors have tested several different conventional architectures, such as Long Short-Term Memory (LSTM) layers for NNCP[25] and CMIX[26], or more nuance systems like DZip[27], with supporter models. These state-of-the-art general purpose compressors don't learn to predict the next value, but rather probability models for the coder.

CMIX, the current state-of-the-art NN-based general- purpose lossless compressor, uses several thousand context models followed by an LSTM byte level mixer

---

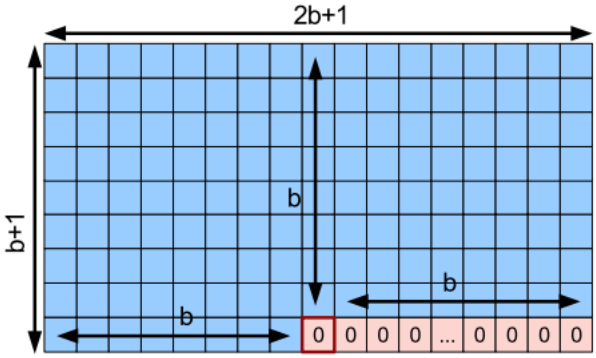
[24] Hilmi E. Egilmez, Ankitesh K. Singh, Muhammed Coban, Marta Karczewicz Yinhao Zhu, Yang Yang, Amir Said, Taco S. Cohen (2021). "Transform Network Architectures for Deep Learning based End-to-End Image/Video Coding in Subsampled Color Spaces", CoRR (arXiv).

[25] https://bellard.org/nncp/

[26] https://www.byronknoll.com/cmix.html

[27] Mohit Goyal, Kedar Tatwawadi, Shubham Chandak and Idoia Ochoa (2021). "DZip: improved general-purpose lossless compression based on novel neural network modeling", DCC 2021.

(to combine predictions) and a bit level NN-based context mixer. The context models and the mixers are then trained through backpropaga- tion adaptively while encoding the input data. CMIX is specialized for text and executable data. NNCP is also an LSTM-based compressor which adaptively compresses the input sequence while simultaneously updating the weights of the RNN. NNCP uses seven stacked LSTM layers which incorporate feature normalisation layers, further adding to the overall runtime.[28]



Prediction neighbourhood used by Schiopu and Munteanu. Generally, the parameter is set to *b=15* in their experiments.

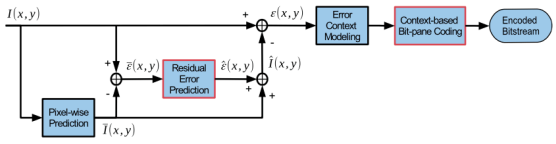


Fig. 1. The hybrid lossless coding framework. The red rectangles mark the modules improved in [9].

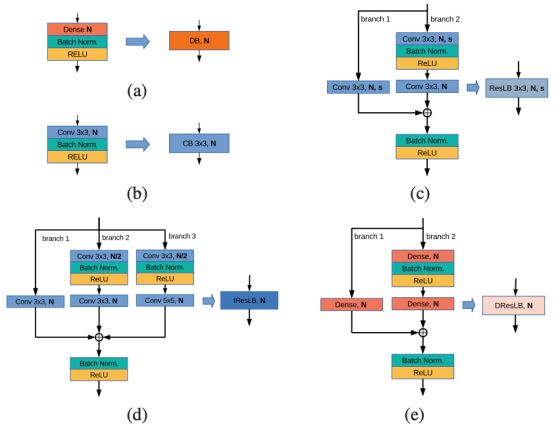


Fig. 2. Layer structure of the blocks from [9]: **(a)** Dense Block (DB); **(b)** Convolution Block (CB); **(c)** Residual Learning based Block (ResLB); **(d)** Inception and ResLB (IResLB). Proposed layer structure: **(e)** Dense ResLB (DResLB).

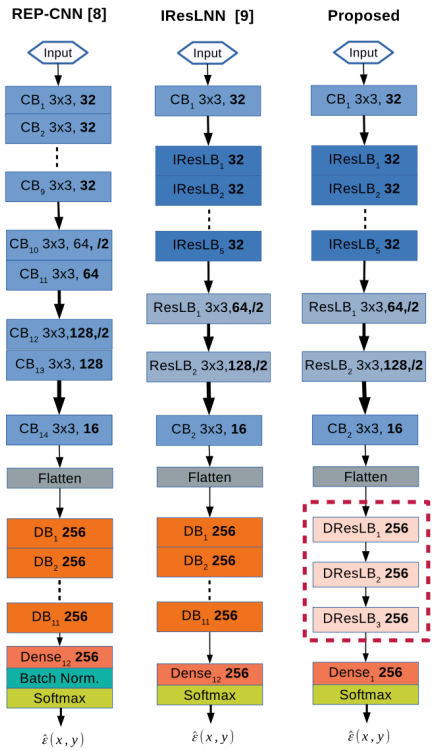


Fig. 3. Comparison between the neural network design of REP-NN [8], IRESLNN [9], and *Proposed* architecture.

Predictor architectures by Schiopu and Munteanu.

[28] Mohit Goyal, Kedar Tatwawadi, Shubham Chandak and Idoia Ochoa (2021). "DZip: improved general-purpose lossless compression based on novel neural network modeling", DCC 2021.

Other predictors are more conventional in nature. Schiopu and Munteanu have proposed several variants of predictor-based compressors[29] [30] [31] using convolutional neural networks. They introduce a modified system of prediction, where both the pixel value and and error are predicted, and incorporate a deep-learning CNN predictor, with a large prediction environment. Later iterations of these predictor models introduced residual blocks for both convolutional layers and dense layers, whcih improved compression results slightly at the cost of increased computational complexity.

Another completely different and novel approach is to use a fast and powerful lossy compressor, such as BPG, as a predictor, training the overall architecture to learn the lossy reconstruction's distribution and the parameters for lossy construction (if any)[32]. The compressor then stores both the lossy compressed image and the residuals, encoding both separately with their own systems.

This method is flexible to incorporate new lossy compression algorithms in its architecture, and achieves state-of-the-art lossless compression results[33]. However, predictor approaches such as the ones described earlier have provided much better results with respect to traditional compressors (FLIF)[34].

Architecture used by Mentzer, Van Gool, and Tschannen for amlossy compressor as predictor.

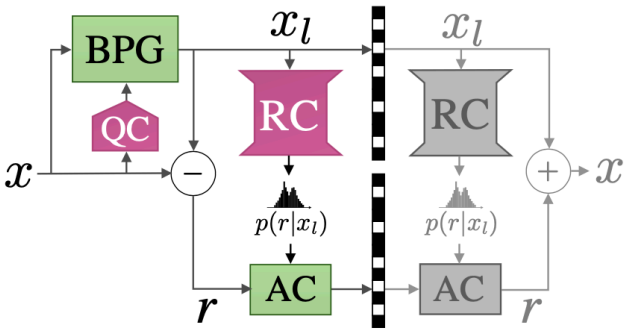


Figure 1. Overview of the proposed learned lossless compression approach. To encode an input image x, we feed it into the Q-Classifier (QC) CNN to obtain an appropriate quantization parameter Q, which is used to compress x with BPG. The resulting lossy reconstruction x_l is fed into the Residual Compressor (RC) CNN, which predicts the probability distribution of the residual, $p(r|x_l)$, conditionally on x_l. An arithmetic coder (AC) encodes the residual r to a bitstream, given $p(r|x_l)$. In gray we visualize how to reconstruct x from the bistream. Learned components are shown in *violet*.

[29] Ionut Schiopu and Adrian Munteanu (2020). chine learning techniques for lossless image cc

[30] Ionut Schiopu, Yu Liu and Adrian Munteanu (2018). "CNN-based Prediction for Lossless Coding of Photographic Images", PCS.

[31] Ionut Schiopu and Adrian Munteanu (2019). "Deep-Learning-Based Lossless Image Coding", IEEE.

[32] Fabian Mentzer, Luc Van Gool, Michael Tschannen (2020). "Learning Better Lossless Compression Using Lossy Compression", IEEE.

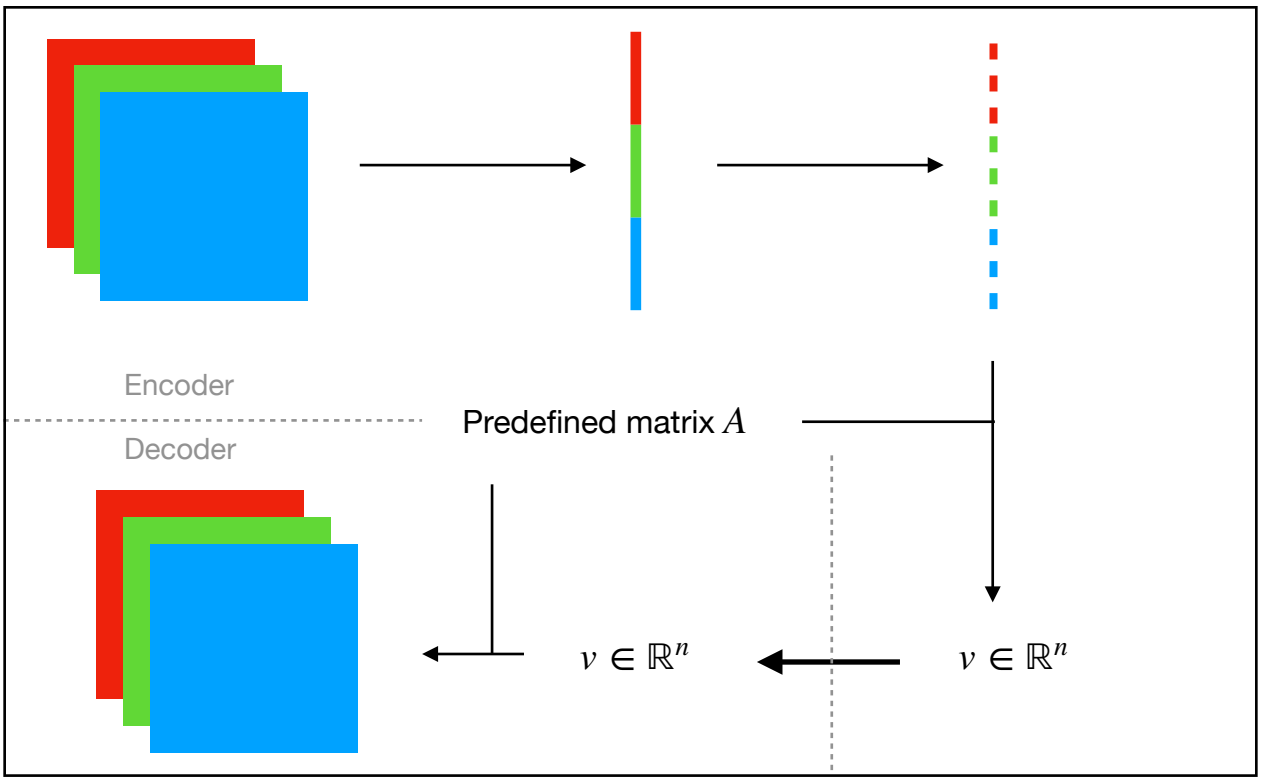
[33] Fabian Mentzer, Luc Van Gool, Michael Tschannen (2020). "Learning Better Lossless Compression Using Lossy Compression", IEEE.

[34] Ionut Schiopu and Adrian Munteanu (2020). "A study of prediction methods based on machine learning techniques for lossless image coding", ICP.

Compressed sensing

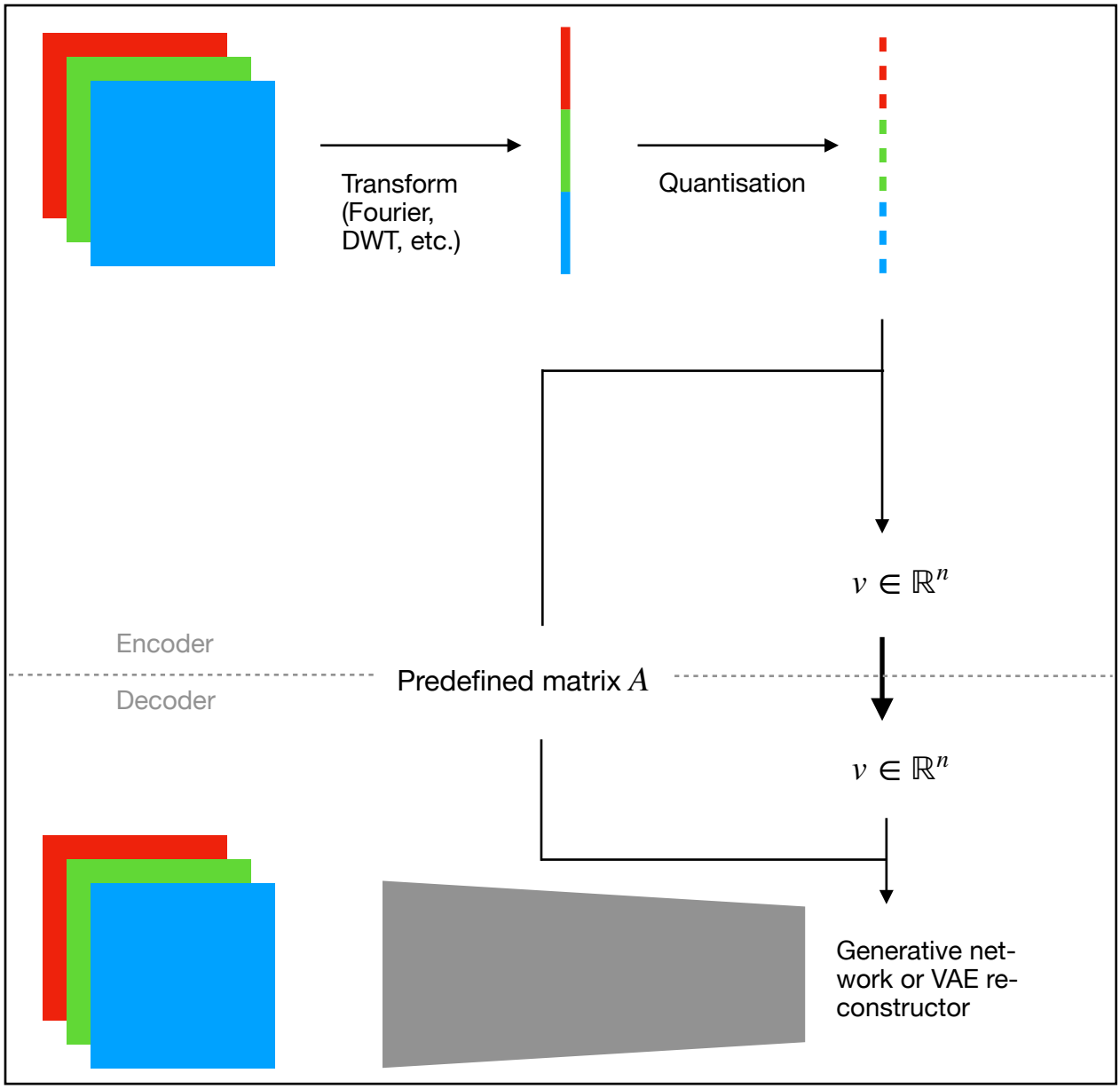
Compressed sensing is a signal compression technique that shifts the computation complexity from the encoder to the decoder[35]. **Compressed Sensing (CS)** is based on the collection of a sparse set of measurements, and then using those to reconstruct the full original signal almost perfectly. This is a fundamental change from the traditional approach to compression, where the full set of measurements is taken, and then it is compressed into a much smaller number of entries, which in turn are then reconstructed by the receiver.

The overall problem is generally modelled in terms of a linear system of equations with more unknowns than equations. Such a system generally has infinite solutions, but additional knowledge about the source is incorporated into the restrictions such as the maximum frequency of the signal, or the sparsity and incoherence of the measurements. In all, very little of the computational complexity remains in the side of the encoder, which at best has only sent the few raw measurements taken, and at worse has only had to filter the full image into a sparse sample that is then sent. All the system's complexity is now on the receiver, on whom it is to reconstruct the input signal.



[35] Yaman Dua, Vinod Kumar, Ravi Shankar Singh (2020). "Comprehensive review of hyperspectral image compression algorithms", Society of Photo-Optical Instrumentation Engineers (SPIE).

Although the original statement of the theoretical framework is in terms of continuous signals, it can also be applied to a discrete signal, such as an image (or hyperspectral image), which is why we can regard it as a compression paradigm.



Put formally, suppose we wish to recover a vector $x_0 \in \mathbb{R}^m$ (e.g. a digital signal or image) from incomplete and contaminated observations $y = Ax_0 + e$; A is an n by m matrix with far fewer rows than columns ($n \ll m$) and e is an error term. Our question is: is it possible to recover x_0 accurately based on the data y?

To recover x_0, Candès, Romberg, and Tao consider the solution x^* to the l_1-regularisation problem:

$$min\|x\|_{l_1} \text{ subject to } \|Ax - y\|_{l_1} \leq \epsilon$$

Where ϵ is the size of the error term e. They show that if A obeys a uniform uncertainty principle (with unit-normed columns) and if the **vector x_0 is sufficiently sparse**, then the solution is within the noise level $\|x^* - x_0\|_{l_2} \leq C\epsilon$.[36]

Following the theoretical layout developed by Candès and Tao, the compressed sensing architecture is based on collecting (or just sending) some of the data, combined in some adequate way (matrix A) so that the *uniform uncertainty* conditions are satisfied and we can recover the original vector.

However, observe that the original vector had a *support boundary*: it's mostly zeros! This can simply be the result of **lossy compression**: as in transform-based coders, we remove all coefficients smaller than some threshold, and then encode the remaining ones. There are two central problems the different techniques try to address: **turning the image into a sparse vector**, and **reconstructing the encoded vector afterwards**.

Since the theoretical foundation for CS is rather strict in its findings, and neural networks are computationally expensive, these are implemented on the decoder side. The central problem in decoding a CS measurement is the computational time to reconstruct the image from the original measurement (the minimisation problem). Authors like Bora *et al.*[37] and Wu *et al.*[38] use generative models like GANs or VAEs that learn the measurements distribution (for a fixed matrix A), and that aim to reconstruct from them the original image.

The most notable work in this regard is that by Ali Mousavi and Richard Baraniuk, and that of Kuldeep Kulkarni *et al.*, developing compressed sensing systems involving

---

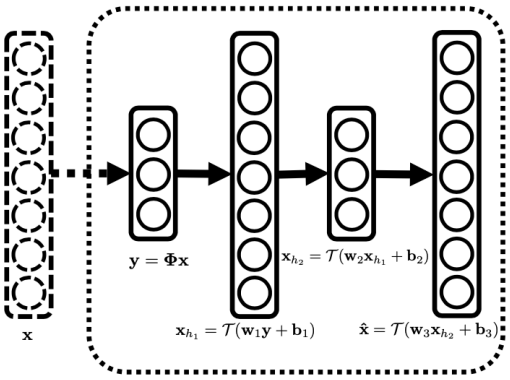
[36] E. Candès, J. Romberg, and T. Tao (February 2005). "Stable signal recovery from incomplete measurements".

[37] Bora, A., Jalal, A., Price, E., and Dimakis, A. G. (2017). "Compressed sensing using generative models".

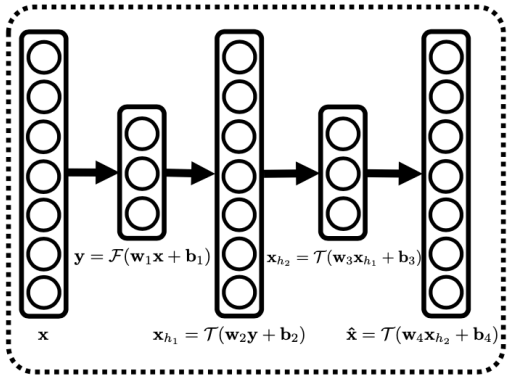
[38] Yan Wu, Mihaela Rosca, and Timothy Lillicrap (2019). "Deep Compressed Sensing", Proceedings of the 36 th International Conference on Machine Learning, Long Beach, California, PMLR 97.

neural networks. In their first submission, Mousavi, Baraniuk, and Patel[39] introduced two models for fully-connected (i.e. conventional) neural networks reconstructing compressed sensed images. In the first model, the image was passed through a classic linear measurement matrix, and the network learned reconstruction from the compressed measurements. In the second model, the input is the raw image itself, made into a vector, thus is a more conventional autoencoder.

This paper was followed by the work of Kuldeep Kulkarni *et al.*[40], whom introduced CNNs to compressed sensing. Their approach was to split the input image into 33x33 RGB blocks, from which compressed measurement vectors were calculated. These vectors were fed to a convolutional neural network that reconstructed the original 33x33 block. All of the blocks would then be put together into the original image ans passed through a denoising post-processing to improve reconstruction quality and eliminate block artefacts.



First model by Mousavi, Baraniuk, and Patel, where the input is the compressed measurements vector.



Second model by Mousavi, Baraniuk, and Patel, where the input is the full measurements vector.

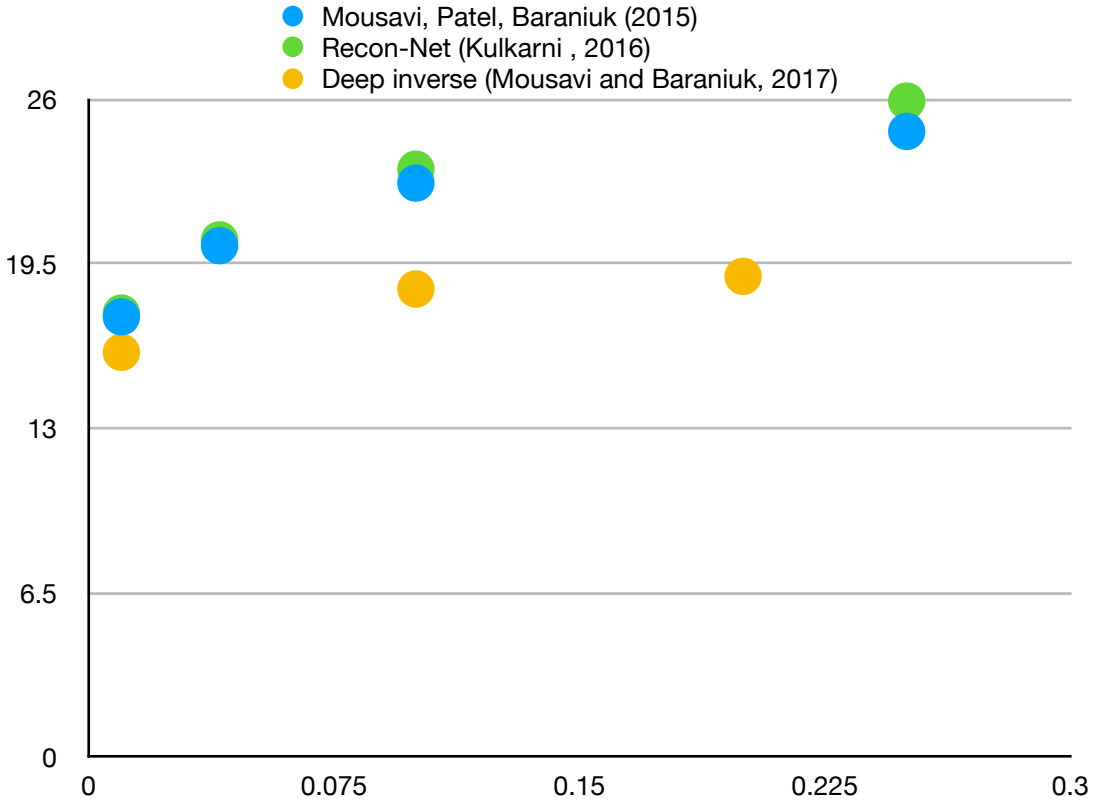
Finally, the 2017 Mousavi and Baraniuk paper[41] uses a CNN to reconstruct images from the compressed measurements vector and the measurements matrix, without splitting it into blocks.

---

[39] Ali Mousavi, Ankit B. Patel, and Richard G. Baraniuk (2015). "A deep learning approach to structured signal recovery".

[40] Kuldeep Kulkarni, Suhas Lohit, Pavan Turaga, Ronan Kerviche, Amit Ashok (2016). "Reconnet Non-iterative reconstruction of images from compressively sensed measurements".

[41] Ali Mousavi and Richard Baraniuk (2017). "Learning to invert: Signal recovery via deep convolutional networks".



These implementations aim to reduce the time of reconstruction of regular CS algorithms while keeping their rate-distortion results, perhaps even by improving on the human perception of said distortion.

From the results of these papers we may deduce that the best approach is to split the image into blocks, which are compressed.measured and fed through a neural network rather than taking the full image as input. However, more work has to be done in this regard.

Later models have improved the performance (reconstruction quality and runtime) of those earlier ones and introduced new features such as **scalability**: models which are capable of using a single learned system to reconstruct an image at different rates[42]. Shi *et al.* achieved this scalable model by introducing a hierarchy in the CS measurements (after the matrix product). Considering the measurements in that order, they could chose a finite pre-determined number of compression rates (say, 0.5, 0.25, 0.1, and 0.05 the size of the original image). On the decoder side, a larger number of measurements was then reconstructed using additional layers on the same base mod-

[42] W. Shi, F. Jiang, S. Liu, D. Zhao (2019). "Scalable Convolutional Neural Network for Image Compressed Sensing", IEEE International Conference on Computer Vision and Pattern Recognition (CVPR).

el, rather than a completely independent model altogether, thus achieving easier training.

The Shi *et al.*[43] paper also implemented a **linear-learned** model: CS measurements were made by a learned matrix rather than a randomly generated one. Their implementation used convolutions, although the resulting operations were identical to those of a matrix: having a fixed size for the input images, $n \times n$, they used m kernels (filters) of size $n \times n$, where m is the number of CS measurements to keep[44].

Table 4: Comparison of residual networks and upsampling operations on Kodak, optimized by MS-SSIM with $\lambda = 5$.

Method	PSNR	MS-SSIM	Rate
Hyperprior-9	26.266	0.9591	0.1690
ResNet-3×3(3)	26.378	0.9605	0.1704
ResNet-3×3(4)-TConv	26.457	0.9611	0.1693
ResNet-3×3(4)-SubPixel	26.498	0.9622	0.1700

Table 5: The effect of wide bottleneck on Kodak dataset.

Method	PSNR	MS-SSIM	Rate
ResNet-3x3(4)-Bottleneck128	26.498	0.9622	0.1700
ResNet-3x3(4)-Bottleneck192	26.317	0.9619	0.1667

Table 6: Rate control on CLIC validation dataset [27].

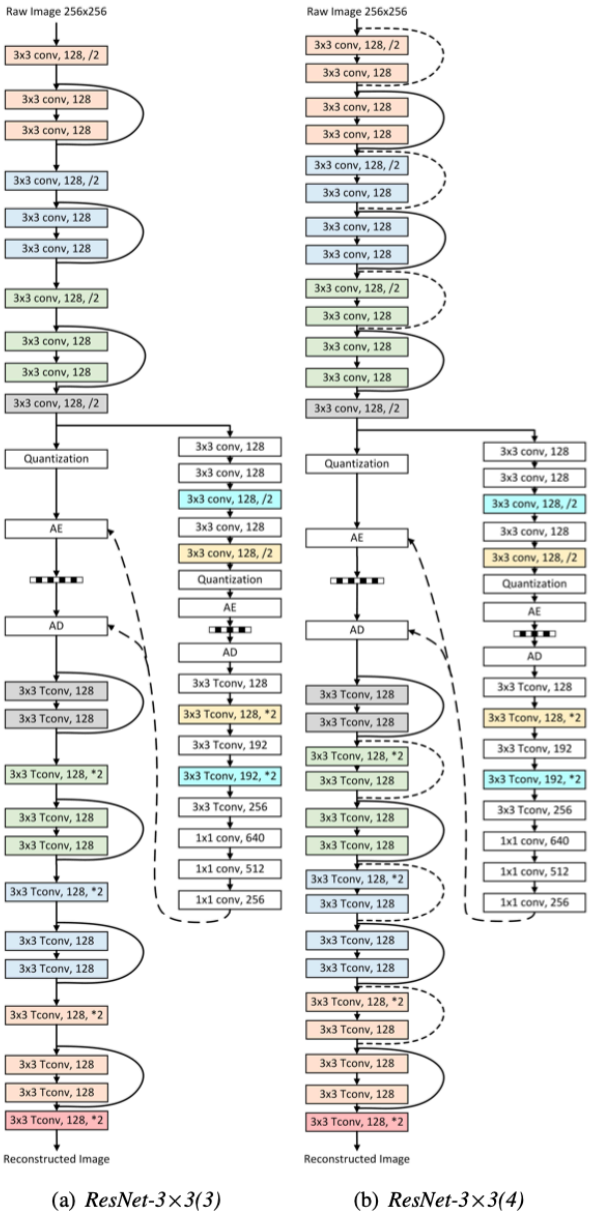
Method	λ	PSNR	MS-SSIM	Rate
ResNet-3x3(4)-Bottleneck192	5	29.708	0.9697	0.1369
ResNet-3x3(4)-Bottleneck192	10	30.710	0.9765	0.1816

Compression performance of different architectures by Katto *et al.*

Table 9: The model complexity of different architectures.

Method	Para	FLOPs	Relative
Baseline-3	997379	4.25×10^{9}	0.36
Baseline-5	2582531	1.18×10^{10}	1.00
Baseline-9	8130563	3.82×10^{10}	3.24
HyperPrior-3	4055107	4.78×10^{9}	0.40
HyperPrior-5	5640259	1.23×10^{10}	1.04
HyperPrior-9	11188291	3.88×10^{10}	3.28
ResNet-3×3(3)	5716355	1.75×10^{10}	1.48
ResNet-3×3(4)	6684931	2.43×10^{10}	2.06
ResNet-3×3(4)-SubPixel	8172172	2.50×10^{10}	2.12
ResNet-3×3(4)-SubPixel-Bottleneck192	11627916	2.56×10^{10}	2.17

Complexity of different architectures by Katto *et al*.



(a) *ResNet-3×3(3)* (b) *ResNet-3×3(4)*

Deep network architectures used by Katto *et al*.

[44] Their images were greyscale. In case of a higher number of layers, you'd adjust m to fit the output number of values.

Complexity and performance

A natural tradeoff is that of network complexity and performance improvements. Complexity, however, isn't a simple parameter that is increased only in one dimension. For example, we can increase complexity by increasing the kernel size (or number of elements in each layer) or the number of layers, and these may have different impacts in the complexity-performance tradeoff.

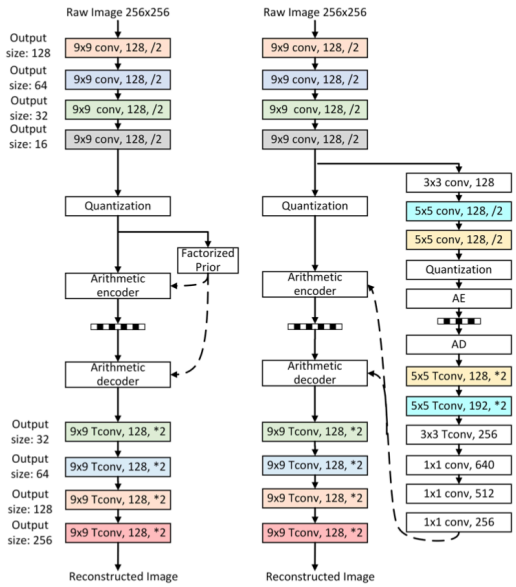


Table 1: The effect of kernel size on *Baseline* on Kodak, optimized by MSE with $\lambda = 0.015$.

Method	PSNR	MS-SSIM	Rate
Baseline-3	32.160	0.9742	0.671
Baseline-5	32.859	0.9766	0.641
Baseline-9	32.911	0.9776	0.633

Table 2: The effect of kernel size on *HyperPrior* on Kodak, optimized by MSE with $\lambda = 0.015$.

Method	PSNR	MS-SSIM	Rate
HyperPrior-3	32.488	0.9742	0.543
HyperPrior-5	32.976	0.9757	0.518
HyperPrior-9	33.005	0.9765	0.512

Table 3: The effect of kernel size in the auxiliary autoencoder on Kodak, optimized by MS-SSIM with $\lambda = 5$.

Method	PSNR	MS-SSIM	Rate
HyperPrior-9-Aux-5	26.266	0.9591	0.169
HyperPrior-9-Aux-9	26.236	0.9590	0.171

Kernel size - performance analysis by Cheng, Sun, Takeguchi, and Katto

Some experimental results[45] show that in VAE architectures an increase in the kernel size in the main AE significantly improves compression performance, but not in the hyperprior, where smaller complexity is far sufficient. Katto *et al.* also show in their paper than the increase in complexity by incrementing the number of layers (depth) is far more moderate than the increase of kernel size, and far more effective in improving performance at the cost of complexity. They also show how the inclusion of a hyperprior makes a small contribution to overall complexity and makes a great impact in compression results.

[45] Z. Cheng, H. Sun, M. Takeguchi, J. Katto (2019). "Deep Residual Learning for Image Compression", CVPR 2019.

Remote sensing data compression

Remote sensing data compression raises two crucial constraints with respect to ordinary image compression: **hyperspectral** data, and **low encoding power**.

Hyperspectral images, unlike RGB images, may have an arbitrary number of components, many more than the 3 traditional ones, which may be very unrelated to each other (again, unlike RGB). This increased amount of information, is amplified by the lack of computing power on board, and the tight bandwidth of the transmission channel.

Not all remote sensing applications are tolerant to compression loss, as often the usage of that data needs it to be completely reliable. In applications where lossy compression is acceptable, and again, unlike in RGB images, human perception of the distortion is no longer a factor. Of course, this factor depends on the specific application and mission, but more refined distortion measures than MSE do not depend on human perception.

These specific challenges mean two things for Machine Learning implementations for remote sensing data compression:

- Any ANNs installed onboard must be very simple, as computational cost there is critical.
- Distortion metrics for lossy approaches used, other than MSE, must be adapted to the specific mission and/or application.

Local statistical properties

Remote sensing images capture large spatial areas into each sample (for example AVIRIS captures $20m \times 20m$ in each sample). This may generate strong local irregularities in our data: the contents of areas of that size can make adjacent pixels very different from one another. Since convolutional neural networks process images by local transforms (the kernels), local variability is an important factor that may hinder our network's capacity to learn.

Our experiments[46] have shown that images with low local variance and/or range are those where our models perform best, and even those which our models learn the

[46] MLC2021111201

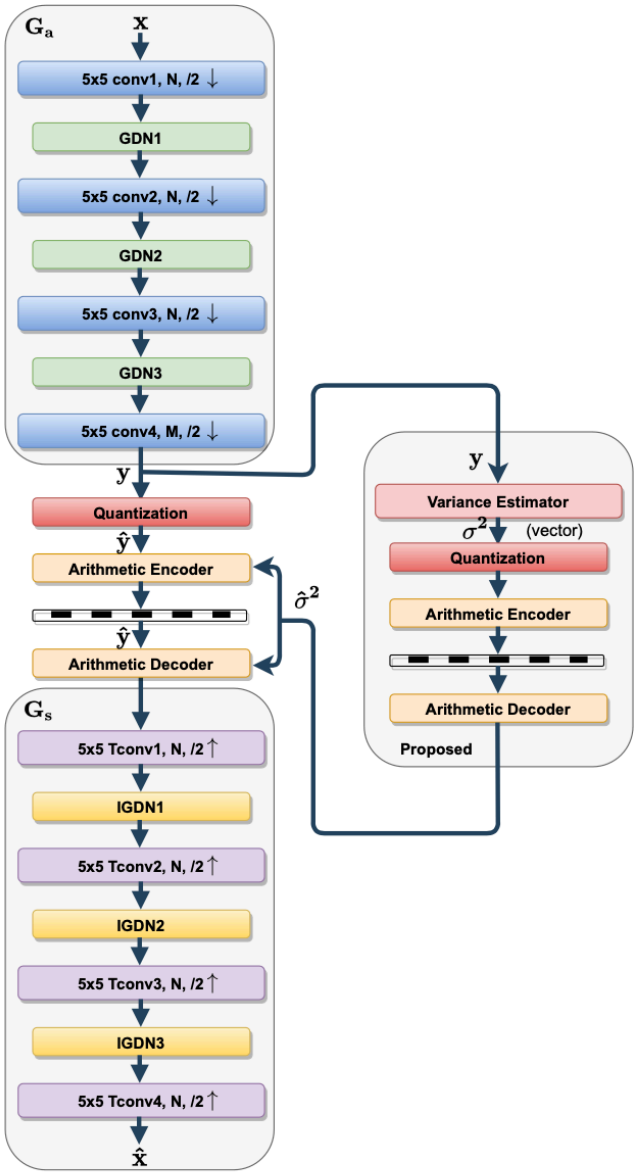


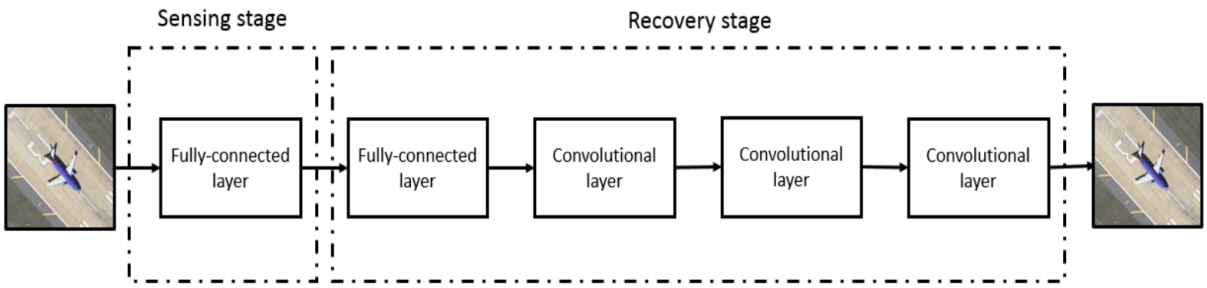
Figure 6. Proposed architecture after entropy model simplification: main autoencoder (**left column**) and simplified auxiliary autoencoder (**right column**).

Simplified autoencoder by Alves de Oliveira *et al*.

most effectively. Conversely, within the same images, bands with higher local variance are bands where our models perform worse, particularly in comparison to benchmark codecs such as JPEG 2000.

Transform coding

Using an autoencoder for hyperspectral images is one of the possible approaches. Here the challenge is to drastically simplify the complexity of the network for



Architecture by Mirrashid and Beheshti.

it to work on board. Alves de Oliveira *et al.*[47] developed a simplified architecture based on those by Ballé *et al.* which performed similarly to larger architectures while being up to 73% simpler (in FLOPs). This simplified architecture outperformed the CCSDS-122 standard and JPEG2000 in the rate-distortion tradeoff, but is also more complex than both.

Compressed sensing

Following prior work on CS using neural networks, Mirrashid and Beheshti[48] proposed a network that can be described as an *unbalanced autoencoder*. This network has an encoder part formed by just one learned layer, which is fully-connected and treats the input image as a single vector, and a decoder part that has one fully-connected layer and several convolutional ones. No clear results regarding the rate-distortion tradeoff are published by these authors.

Tensor decomposition

Let $\mathcal{X} \in \mathbb{R}^{N_1 \times \ldots \times N_d}$ be a d-dimensional tensor. The **Tucker Decomposition (NTD)** of the tensor will be something like $\mathcal{X} = \mathcal{G} \times_1 A_1 \times_2 \ldots \times_d A_d$, where we have the **core tensor** $\mathcal{G} \in \mathbb{R}^{M_1 \times \ldots \times M_d} : M_i < N_i$, and factors $A_i \in \mathbb{R}^{M_i \times N_i}$. The point of this decomposition is to transform $\mathcal{X}$ into $\mathcal{G}$ of much smaller dimension (hence, *compress*

[47] Vinicius Alves de Oliveira, Marie Chabert, Thomas Oberlin, Charly Poulliat, Mickael Bruno, Christophe Latry, Mikael Carlavan, Simon Henrot, Frederic Falzon, and Roberto Camarero (2021). "Reduced-Complexity End-to-End Variational Autoencoder for on Board Satellite Image Compression", MDPI.

[48] Alireza Mirrashid and Ali Asghar Beheshti (2018). "Compressed Remote Sensing by using Deep Learning", 9th International Symposium on Telecommunications.

$\mathcal{X}$) using a **fixed set of factors** $A_i \in \mathbb{R}^{M_i \times N_i}$ and minimising error. This process can therefore be seen as a form of dimensionality reduction.

In practical terms for hyperspectral images, this decomposition is done in frequency space (i.e. after a transform like DCT or KLT). The conventional algorithms that iteratively derive a core tensor $\mathcal{G}$ minimising the error $\|\mathcal{X} - \hat{\mathcal{X}}\|^2$, where $\hat{\mathcal{X}} = \mathcal{G} \times_1 A_1 \times_2 \ldots \times_d A_d$, are computationally expensive.

Authors Jin Li and Zilong Liu[49] developed a CNN autoencoder that reduces the dimensionality of the input image (tensor) down to a smaller tensor which is then transformed by the DCT and finally compressed and encoded using NTD. They tested this method on an unspecified dataset of images with, apparently 32 bands[50].

Generative Networks

Generative Neural Networks (GNN) learn to reconstruct data (such as images) from small latent representations; essentially, VAE without the encoder. In compression, these require that the input data be encoded into a latent representation that has a regular distribution, such as some form of normalisation. In VAE, the encoder learns such a distribution simultaneously. In a GNN, other techniques are used, as the point is that there is no encoder network at play.

Deng *et al.*[51] used a **GNN as the representation of the image**: a large hyperspectral image would be partitioned into spatial patches of fixed size, numbered $1, \ldots, k$. These indices could then be embedded into a normally distributed vector, so each index would map to a "region" in that vector space. Given one such index and its corresponding patch, the index would be transformed into said randomised vector, and the network would learn (via training) to reconstruct that patch from the vector, using an l_p norm between the original patch and the reconstructed patch as target function.

---

[49] Jin Li and Zilong Liu (2019). "Multispectral Transforms Using Convolution Neural Networks for Remote Sensing Multispectral Image Compression", MDPI.

[50] The authors do not specify the source of their dataset, nor the number of bands on each image, only described as "We tested our CNN-based NTD method on 50 multispectral images that consist of a variety of buildings, cities, and mountains.". The number of bands is inferred from another diagram in the paper representing the network "when the number of bands is 32".

[51] Chubo Deng, Yi Cen, and Lifu Zhang (2020). "Learning-Based Hyperspectral Imagery Compression through Generative Neural Networks", MDPI Remote Sensing.

Importantly, **the network itself would be the compressed version of the image**. In other words, training is the compression process. Then the image (or a patch of it) would be recovered by giving the decoder an index, transform it into one of those embedder vectors, and feeding those into the network. For further compression and to give some choice on the rate-distortion tradeoff for the user, the resulting network could be pruned up to a certain user-selected threshold, reducing its performance but making the network lighter.

Onboard power limitations

As said multiple times, the main constraint in onboard compression applications is the availability of computational resources. As a benchmark, one of the most widely used CPUs in current spacecraft is the RAD750[52], made by BAE Systems, which is in missions such as Landsat 8[53], WISE, or Martian rovers. It has a core clock of **110 to 200 MHz** and can process at 266 MIPS[54] or more.

Of course, runtime is also bound by the number of **images needed per minute**. In the case of Landsat 8, for example, capture rate has been of, at most, 0,49 images per minute (725 scenes per day)[55].

Noise removal

There are several **types** of noise in digital images. **Salt and pepper noise** (sparse light and dark disturbances) pixels in the image are very different in color or intensity from their surrounding pixels; the defining characteristic is that the value of a noisy pixel bears no relation to the color of surrounding pixels. Generally this type of noise will only affect a small number of image pixels. When viewed, the image contains dark and white dots, hence the term salt and pepper noise. Typical sources include flecks of dust inside the camera and overheated or faulty CCD elements.

In Gaussian noise, each pixel in the image will be changed from its original value by a (usually) small amount. A histogram, a plot of the amount of distortion of a pixel

---

[52] https://en.wikipedia.org/wiki/RAD750

[53] https://earth.esa.int/web/eoportal/satellite-missions/content/-/article/landsat-8-ldcm

[54] Millions of Instructions Per Second

[55] https://earth.esa.int/web/eoportal/satellite-missions/content/-/article/landsat-8-ldcm

value against the frequency with which it occurs, shows a normal distribution of noise. While other distributions are possible, the Gaussian (normal) distribution is usually a good model, due to the central limit theorem that says that the sum of different noises tends to approach a Gaussian distribution.

In either case, the noise at different pixels can be either correlated or uncorrelated; in many cases, noise values at different pixels are modeled as being independent and identically distributed, and hence uncorrelated.

The main **tradeoff** in removing noise is in the sacrificing of detail in an effort to remove noise. Denoising an image will inevitably blur some finer details, thus noise removal has its limitations.

Luminance-Chroma noise removal

In the Luminance-Chroma space, most of the detail is encapsulated in the luminance (brightness) component, thus denoising in this component comes with a greater risk of losing details. Further, in natural images, noise in the chroma components is seen as far less acceptable than in the luminance component. Therefore most noise removal applications in digital cameras only apply to these components.

Linear smoothing filters

One method to remove noise is by **convolving** the original image with a mask that represents a low-pass filter or smoothing operation. For example, the Gaussian mask comprises elements determined by a Gaussian function. This convolution brings the value of each pixel into closer harmony with the values of its neighbors. In general, a smoothing filter sets each pixel to the average value, or a weighted average, of itself and its nearby neighbors; the Gaussian filter is just one possible set of weights.

Smoothing filters tend to **blur** an image, because pixel intensity values that are significantly higher or lower than the surrounding neighborhood would "smear" across the area. Because of this blurring, linear filters are seldom used in practice for noise reduction; they are, however, often used as the basis for nonlinear noise reduction filters.

Anisotropic diffusion

Another method for removing noise is to **evolve the image** under a smoothing partial differential equation similar to the heat equation, which is called **anisotropic dif-**

fusion. With a spatially constant diffusion coefficient, this is equivalent to the heat equation or linear Gaussian filtering, but with a diffusion coefficient designed to detect edges, the noise can be removed without blurring the edges of the image.

Non-local means

Another approach for removing noise is based on **non-local averaging** of all the pixels in an image. In particular, the amount of weighting for a pixel is based on the degree of similarity between a small patch centered on that pixel and the small patch centered on the pixel being de-noised.

Non-linear filters

A **median filter** is an example of a non-linear filter and, if properly designed, is very good at preserving image detail. To run a median filter:

1. Consider each pixel in the image
2. Sort the neighbouring pixels into order based upon their intensities
3. Replace the original value of the pixel with the median value from the list

A median filter is a rank-selection (RS) filter, a particularly harsh member of the family of **rank-conditioned rank-selection (RCRS) filters**;[56] a much milder member of that family, for example, is one that selects the **closest of the neighboring values** when a pixel's value is external in its neighborhood, and leaves it unchanged otherwise, is sometimes preferred, especially in photographic applications.

Median and other RCRS filters are good at removing salt and pepper noise from an image, and also cause relatively little blurring of edges, and hence are often used in computer vision applications.

---

[56] Liu, Puyin; Li, Hongxing (2004). Fuzzy Neural Network Theory and Application. Intelligent Robots and Computer Vision Xiii: Algorithms and Computer Vision. 2353. World Scientific. pp. 303–325.

Wavelet transform

The main aim of an image denoising algorithm of this type is to achieve both noise reduction[57] and feature preservation[58] using the **wavelet filter banks**[59]. In this context, wavelet-based methods are of particular interest. In the wavelet domain, the noise is uniformly spread throughout coefficients while most of the image information is concentrated in a few large ones[60]. Therefore, the first wavelet-based denoising methods were based on **thresholding** of detail subbands coefficients[61]. However, most of the wavelet thresholding methods suffer from the drawback that the chosen threshold may not match the specific distribution of signal and noise components at different scales and orientations.

To address these disadvantages, **non-linear estimators** based on Bayesian theory have been developed. In the Bayesian framework, it has been recognized that a successful denoising algorithm can achieve both noise reduction and feature preservation if it employs an accurate statistical description of the signal and noise components[62].

Deep learning

Various deep learning approaches have been proposed to solve noise reduction and such image restoration tasks. **Deep Image Prior**, for example, is one such tech-

---

[57] Chervyakov, N. I.; Lyakhov, P. A.; Nagornov, N. N. (2018). "Quantization Noise of Multilevel Discrete Wavelet Transform Filters in Image Processing". Optoelectronics, Instrumentation and Data Processing. 54 (6): 608–616.

[58] Craciun, G.; Jiang, Ming; Thompson, D.; Machiraju, R. (March 2005). "Spatial domain wavelet design for feature preservation in computational data sets". IEEE Transactions on Visualization and Computer Graphics. 11 (2): 149–159.

[59] Gajitzki, Paul; Isar, Dorina; Simu, Călin (November 2018). "Wavelets Based Filter Banks for Real Time Spectrum Analysis". 2018 International Symposium on Electronics and Telecommunications (ISETC): 1–4.

[60] Forouzanfar, M.; Abrishami-Moghaddam, H.; Ghadimi, S. (July 2008). "Locally adaptive multiscale Bayesian method for image denoising based on bivariate normal inverse Gaussian distributions". International Journal of Wavelets, Multiresolution and Information Processing. 6 (4): 653–664.

[61] Mallat, S. (1998). A Wavelet Tour of Signals Processing. London: Academic Press.

[62] Forouzanfar, M.; Abrishami-Moghaddam, H.; Ghadimi, S. (July 2008). "Locally adaptive multiscale Bayesian method for image denoising based on bivariate normal inverse Gaussian distributions". International Journal of Wavelets, Multiresolution and Information Processing. 6 (4): 653–664.

nique which makes use of convolutional neural network and is distinct in that it requires no prior training data[63].

Compression for computer vision

A novel field of research for Machine Learning Image Compression is **compression optimised for computer vision**. This works by essentially considering the codec and the learned task jointly, and optimising these systems (either separately or jointly) for a rate-success rate tradeoff rather than the usual rate-distortion tradeoff used for human perception of the images.

An example of one such application considered a now basic VAE architecture by Ballé *et al.*[64], removing the hyperprior, and incorporating an object-detection algorithm at the other end. The target function was then the parametrised tradeoff between the compression ratio of the image into the latent representation and the success rate of the task at the other end. The authors of this paper tested whether training only the task parameters, only the codec, or the joint model, finding that the best-performinng scenario was the jointly-trained system, somewhat closely followed by the task-only trained model, and finally far behind the codec-only trained system.[65]

Conditional Neural Networks

Conditional Neural Networks are a type of feed-forward neural network aimed at incorporating a conditioning parameter (such as a style to apply to the picture) on the network, as opposed to training individual networks for each value of the parameter.

This is achieved by a **normalisation of the activations** in the network as a function of the conditioning. More formally, the activations (outputs of a layer *after* passing through the activation function) are normalised and transformed by a function that depends on the conditioning parameter. For example, let $x \in \mathbb{R}^n$ be the activations, let μ, σ be the mean and standard deviation of x, and let s be our conditioning parameter

---

[63] Ulyanov, Dmitry; Vedaldi, Andrea; Lempitsky, Victor (30 November 2017). "Deep Image Prior"

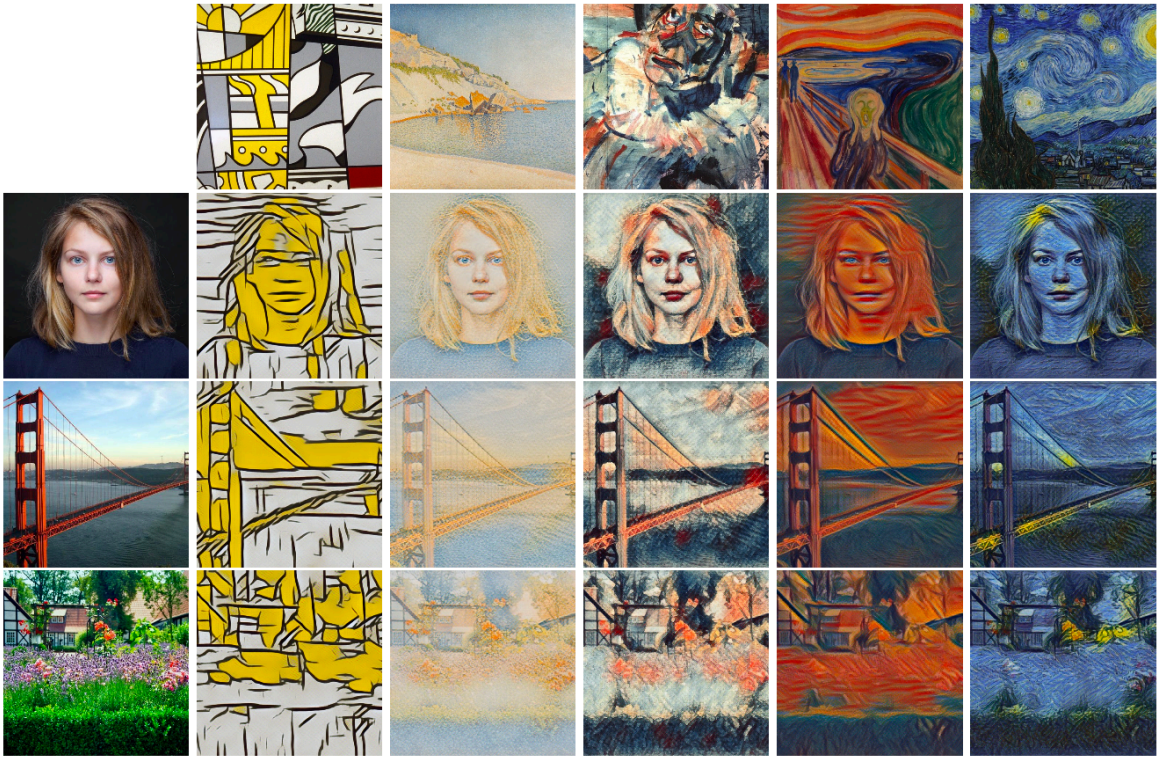
[64] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston (2018). "Variational image compression with scale hyperprior", Conference Paper, ICLR 2018.

[65] Lahiru D. Chamain, Fabien Racapé, Jean Bégaint, Akshay Pushparaja, and Simon Feltman (2021). "End-to-end optimized image compression for machines, a study", Data Compression Conference (DCC) 2021.

(an integer or pair of integers, for example). An affine conditioning, as presented by V. Dumoulin *et al.*[66] [67] would be:

$$z = \gamma_s \left(\frac{x - \mu}{\sigma} \right) + \beta_s$$

Conditioning in this manner was introduced by V. Dumoulin *et al.*[68] as a modification to a network that transformed images into styles of artists and specific paintings. Instead of training a network for each style, they introduced the style as a conditioning in the activations, achieving highly successful results in a single network. There, the conditioning parameter was the image in whose style the output needed to be tested, and the function that produced the parameters γ_s and β_s was a secondary ANN (actually, more than one such ANN as the parameters changed in each layer).



Pastiches by V. Dumoulin *et al*. in their ICLR 2017 paper.

---

66 Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur (2017). "A learned representation for artistic style", ICLR 2017.

67 Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville (2017). "FiLM: Visual Reasoning with a General Conditioning Layer", AAAI 2018.

68 Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur (2017). "A learned representation for artistic style", ICLR 2017.

These techniques have not been yet applied to a compression scenario.

Multispectral and hyperspectral images compression

Multispectral and hyperspectral images compression introduces new issues that usual images with 1 or 3 channels do not really have, namely **spectral correlation**. De-correlation for natural images (such JPEG, or the 2D-convolutional autoencoders we've mostly seen in other sections[69]) usually focuses on the spatial component, and correlation between bands is mostly neglected.

In multispectral or hyperspectral images correlation is generally far higher between channels (spectral correlation) than among neighbouring pixels (spatial correlation)[70]. Therefore, 2D convolutions that operate only within a single band at a time will mostly miss this correlation, achieving less competitive results. A few techniques have been published for ML multispectral image compression, and none for hyperspectral image compression.

---

[69] J. Ballé, V. Laparra, E. Simoncelli, (2017) "End-to-end optimised image compression", *ICLR Conference*.
Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár (2017) - "Lossy image compression with compressive autoencoders", ICLR 2017.
J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston (2018). "Variational image compression with scale hyperprior", Conference Paper, ICLR 2018.
J. Ballé *et al*, (2020) "Nonlinear Transform Coding", *Google Research*.
Vinicius Alves de Oliveira, Marie Chabert, Thomas Oberlin, Charly Poulliat, Mickael Bruno, Christophe Latry, Mikael Carlavan, Simon Henrot, Frederic Falzon, and Roberto Camarero (2021). "Reduced-Complexity End-to-End Variational Autoencoder for on Board Satellite Image Compression", MDPI.

[70] N. Amrani, J. Serra-Sagristà, M. Hernández-Cabronero and M. Marcellin (2016). “Regression Wavelet Analysis for Progressive-Lossy-to-Lossless Coding of Remote-Sensing Data,” In IEEE Data Compression Conference (DCC).
S. Alvarez-Cortes, J. Serra-Sagristà, J. Bartrina-Rapesta and M. Marcellin (2020). “Regression Wavelet Analysis for Near-lossless Remote Sensing Data Compression,” IEEE Trans. Geoscience and Remote Sensing, vol. 58, no. 2, pp. 790-798.

This type of images is the norm in **remote sensing data**. Most applications of ML remote sensing data compression use single-channel images[71], thus this area of research is still largely undeveloped.

Challenges

Any codec designed for compressing remote sensing data must tackle the following challenges this type of data presents:

1. Remote sensing data is usually collected on **low-power hardware**, most critical in the case of satellites, thus a codec must have as low complexity as possible.

2. Remote sensing data is **very diverse** in nature, with varying bit depth, image size, resolution, number of channels, and correlation conditions. Each of these factors interact differently with a codec, particularly a learned codec.

3. Each source of remote sensing data has different **usage requirements**: some may need to identify certain regions of the image or certain spectral sections, and loss may be more or less acceptable in different scenarios. A codec must either be flexible in this regard, or must be tailored for specific real applications.

Beyond these general challenges there are specific factors to deal with in the realm of Machine Learning image compression:

1. A network's learning capacity is fundamentally ruled by its **number of parameters**, and further by how these parameters are arranged (filter sizes, number of feature maps, biases, size of the latent representations, residual operations, etc.). The complexity of running and training said network is also fundamentally determined by that number of parameters. Since these images usually have many features and are potentially huge, there is a balance to be found in this setup.

2. Neural networks are generally more complex than equivalent conventional linear transforms (such as DCT and DWT) used in compression.

---

[71] Vinicius Alves de Oliveira, Marie Chabert, Thomas Oberlin, Charly Poulliat, Mickael Bruno, Christophe Latry, Mikael Carlavan, Simon Henrot, Frederic Falzon, and Roberto Camarero (2021). "Reduced-Complexity End-to-End Variational Autoencoder for on Board Satellite Image Compression", MDPI.
Alireza Mirrashid and Ali Asghar Beheshti (2018). "Compressed Remote Sensing by using Deep Learning", 9th International Symposium on Telecommunications.
Jin Li and Zilong Liu (2020). "Efficient compression algorithm using learning networks for remote sensing images", Applied Soft Computing Journal.

Spectral and spatial decorrelation

In general hyperspectral images are more highly correlated spectrally than spatially. This is especially true if the number of bands is high and resolution low. Despite the fact that we may understand hyperspectral images as 3D tensors, we have one dimension -the spectral dimension- behaving differently from the other two -the spatial ones- and a learned transform must reflect that at a fundamental level.

Correlation spectrally is not only usually stronger than spatially, but is also of a different nature than correlation within bands, that is, spatially. In both experiments of our own[72] and published articles [73] on, for example, AVIRIS data, we see this correlation is not remotely continuous on the band number (that is bands that are *close* hare *more* correlated than bands that are *far away*), instead displaying clearly defined clusters with high *internal* correlation and lower *external* correlation. This reality may be exploited in different ways when designing an architecture for a learned codec.

Dynamic range and normalisation

A common feature of hyperspectral images is their large dynamic range. Samples in these images typically are of 16-bit unsigned integers or other such data types, which is a key difference with natural images which store their samples using 8 bits.

The main issue with an expanded dynamic range is the random behaviour of the least significant bits of each sample. Consider, for example 16-bit unsigned integers. We have a most significant byte that encodes the value of the sample with an error of up to 128, and a least significant byte that encodes the remaining values between 0 and 255. That first significant byte has a more regular behaviour, whereas the least significant byte is practically random noise.

Experiments of our own[74] show that models trained to compress images quantised to 8 bits (considering only the most significant byte) perform as well or better than

---

[72] MLC2021100603

[73] N. Amrani, J. Serra-Sagristà, M. Hernández-Cabronero and M. Marcellin (2016). “Regression Wavelet Analysis for Progressive-Lossy-to-Lossless Coding of Remote-Sensing Data,” In IEEE Data Compression Conference (DCC).
S. Alvarez-Cortes, J. Serra-Sagristà, J. Bartrina-Rapesta and M. Marcellin (2020). “Regression Wavelet Analysis for Near-lossless Remote Sensing Data Compression,” IEEE Trans. Geoscience and Remote Sensing, vol. 58, no. 2, pp. 790-798.

[74] MLC2021111501

JPEG 2000, whereas models trained for the full 16-bit samples fall behind. This is likely due to the almost noisy quality of the details in that numerical range below 256.

On the other hand, there is the question of real dynamic range versus theoretical dynamic range. For example, it is not uncommon to store 12-bit data as 16-bit data, since it is more convenient to process bytes in full pieces, essentially wasting 25% of the storage space. The networks usually divide by the theoretical maximum at the start to only deal with values in the $[0,1]$ or $[-1,1]$ intervals, in order to avoid backpropagation issues. Our experiments[75] show that this is not necessarily an issue, and that the networks achieve equivalent results regardless of the normalisation's magnitude (assuming no loss in that normalisation process).

3D-convolutional networks

An approach to incorporate spectral decorrelation in the network's operations (convolutions) is to apply **3D convolutions**. These are analogous in operation to the 2D convolutions, but operate in one additional dimension. Just as applying k 2D filters transformed an image (with one or many bands) into a 3D tensor, **multiple 3D filters will yield a 4D tensor**.

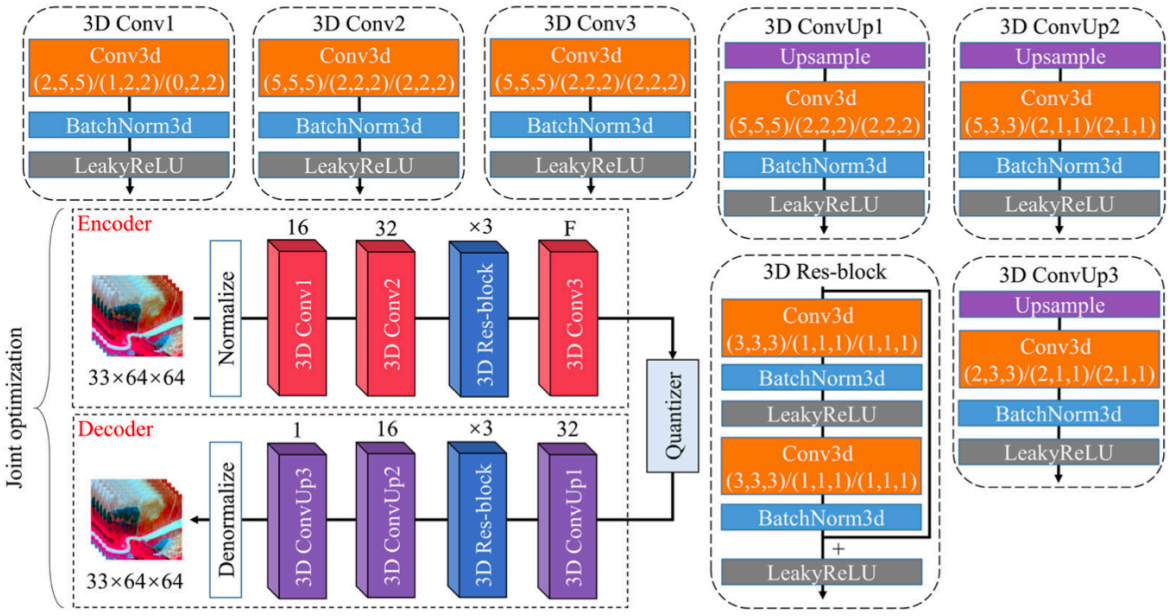


Fig. 3 Framework of 3D-CAE.

[75] MLC2021111501

Architecture by Chong, Chen, and Pan. Conv3d (k1, k2, k3)/(s1, s2, s3)/(p1, p2, p3) represents a 3D convolution with convolution kernel size (k1, k2, k3), stride (s1, s2, s3), and padding (p1, p2, p3). The labels above each block (16, 32, 3, F) represent the number of filters in each layer. The F value controls the number of bits per pixel in the model, $F \in [2,6]$.

Using 3D convolutional networks for multispectral data compression is a novel area of research, pioneered by Chong, Chen, and Pan[76]. These authors developed an autoencoder for multispectral natural images[77] with 33 channels (8-bit images). The autoencoder's architecture can be seen in the image.

This model's bottleneck was between 0,8% and 2,5% the size (number of entries) of the input image[78]. Their target function incorporated several different distortion metrics in a predefined linear combination:

$$L = 1000\text{SAM} + (45 - \text{PSNR}) + 1000(1 - \text{SSIM})$$

Where SAM is the Spectral Angle Mapper, PSNR is Peak Signal-to-Noise Ratio, and SSIM is Structural Similarity.

This codec was compared to the equivalent 2D networks, which it consistently outperformed in all metrics. The authors claim the codec also claim the network outperformed standard JPEG, though they only provide a single datapoint (as opposed to the whole rate-distortion curves). No comparison was made against JPEG2000 or other conventional standards.

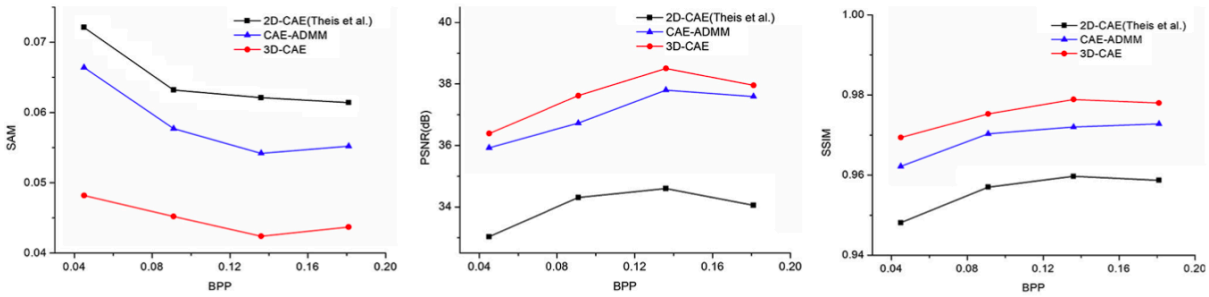


Fig. 4 Performance comparison of 2D-CAE, 3D-CAE, and CAE-ADMM at different BPPs (0.045, 0.091, 0.136, 0.182).

[76] Yanwen Chong, Linwei Chen, and Shaoming Pan (2021). "End-to-end joint spectral–spatial compression and reconstruction of hyperspectral images using a 3D convolutional autoencoder", Journal of Electronic Imaging.

[77] S. M. C. Nascimento, K. Amano, D. H. Foster (2015). "Spatial distributions of local illumination color in natural scenes", Vision Research.

[78] The bottleneck size and the values of F have been calculated from their published values of BPP (which is in fact bits per sample, not per spectral pixel).

Tensor decomposition

Let $\mathcal{X} \in \mathbb{R}^{N_1 \times \ldots \times N_d}$ be a d-dimensional tensor. The **Tucker Decomposition (NTD)** of the tensor will be something like $\mathcal{X} = \mathcal{G} \times_1 A_1 \times_2 \ldots \times_d A_d$, where we have the **core tensor** $\mathcal{G} \in \mathbb{R}^{M_1 \times \ldots \times M_d} : M_i < N_i$, and factors $A_i \in \mathbb{R}^{M_i \times N_i}$. The point of this decomposition is to transform $\mathcal{X}$ into $\mathcal{G}$ of much smaller dimension (hence, *compress* $\mathcal{X}$) using a **fixed set of factors** $A_i \in \mathbb{R}^{M_i \times N_i}$ and minimising error. This process can therefore be seen as a form of dimensionality reduction.

In practical terms for hyperspectral images, this decomposition is done in frequency space (i.e. after a transform like DCT or KLT). The conventional algorithms that iteratively derive a core tensor $\mathcal{G}$ minimising the error $\|\mathcal{X} - \hat{\mathcal{X}}\|^2$, where $\hat{\mathcal{X}} = \mathcal{G} \times_1 A_1 \times_2 \ldots \times_d A_d$, are computationally expensive.

Authors Jin Li and Zilong Liu[79] developed a CNN autoencoder that reduces the dimensionality of the input image (tensor) down to a smaller tensor which is then transformed by the DCT and finally compressed and encoded using NTD. They tested this method on an unspecified dataset of images with, apparently 32 bands[80].

Band-by-band methods

In September and October 2021 we developed the band-by-band methods of compression for multispectral and hyperspectral images, which as the name suggests, consisted in neural networks that compressed the images band-by-band. This was designed on two premises:

1. Conventional multi-band models such as those published by Ballé *et al.*[81] are expensive to train when the number of bands grows considerably, as their computational complexity grows linearly with the number of bands if we fix the number of

---

[79] Jin Li and Zilong Liu (2019). "Multispectral Transforms Using Convolution Neural Networks for Remote Sensing Multispectral Image Compression", MDPI.

[80] The authors do not specify the source of their dataset, nor the number of bands on each image, only described as "We tested our CNN-based NTD method on 50 multispectral images that consist of a variety of buildings, cities, and mountains.". The number of bands is inferred from another diagram in the paper representing the network "when the number of bands is 32".

[81] J. Ballé, V. Laparra, E. Simoncelli, (2017) "End-to-end optimised image compression", *ICLR Conference*.

filters, and quadratically if we expand the number of filters in proportion to the number of bands (which seems necessary from our experimental measurements[82]).

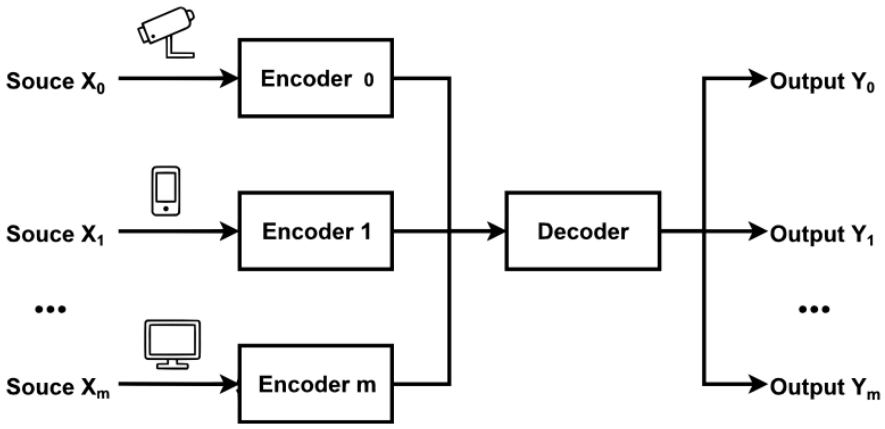
2. Similarity (correlation) between bands typically comes in clusters rather than as a continuous gradient, and models trained for an interval of bands can easily match the performance of models trained for one single band[83].

With that, the band-by-band methods fielded an array of models, each trained for a particular interval in the images' spectrum (for example, one model is trained to compress bands 1 to 50, the next one bands 51 to 134, etc.). These methods did not perform, therefore, any spectral transform, and thus could never be optimal for the full image. They were, however, less computationally expensive than such codecs.

This approach has found several flaws: the initial interval design for AVIRIS data was found to be redundant[84], as some intervals had very high correlation between them as well; models seem to get caught-up in intervals where they underperform and this makes performance on the overall image fall behind that of the standards[85]; and some parts of the spectrum are fundamentally noisy and thus not useful for our compression purposes.

Distributed Source Coding

Distributed Source Coding (DSC) is a coding paradigm consisting in a distributed encoding of the data (being from different sources or following some split in the input), and then decoding it jointly. This paradigm can be applied to data coming from different correlated sources, both as a coding system (lossless) and as a compression system (lossy). Such a system is easily scalable, and less computationally expensive than decorrelating across sources (in case these come together). One of the main advantages of DSC is that the computation complexity of the encoder is transferred to the decoder.



82 M

83 M

84 M

85 M

An example of an application of DCS could be hyperspectral data compression. In that case we'd consider each channel or band as an independent source. These are highly correlated, and it is clear why transforming each band independently is less computationally costly that applying a spatial and spectral transform. Further, it is also clear how this codec scales linearly (in the compressor end) with the number of bands, as opposed to quadratically when considering a spatial-spectral transform.

It is tempting to think that splitting raw data for different encoders compromises the compression quality, however, the Slepian-Wolf Theorem shows that lossless coding of two or more correlated data sources with separate encoders and a joint decoder can compress data as efficiently as the optimal coding using a joint encoder and decoder[86]. This was extended to lossy compression with Gaussian data sources by the Wyner-Ziv Theorem[87].

A first implementation of this paradigm in ML[88] achieved results that matched or surpassed those of the baseline model, where they encoded multiple sources of natural images independently and decoded them jointly (in the same decoder).

Questions

1. Do these types of compression require *large* computational power (for encoding)? Compare with JPEG, JPEG2000, or CCSDS standards. *A one-layer network will be as expensive (in terms of both memory and CPU) as a traditional transform-based compressor (like JPEG2000). Each additional layer adds on to that, not multiplicatively.*

2. How many layers do autoencoders normally have (as ANNs)? *It depends. You may adapt your ANN to have as many **convolutional** layers as possible.*

---

[86] David Slepian and Jack Wolf, “Noiseless coding of correlated information sources,” *IEEE Transactions on information Theory*, vol. 19, no. 4, pp. 471–480, 1973.
T Cover, “A proof of the data compression theorem of slepian and wolf for ergodic sources (corresp.),” *IEEE Transactions on Information Theory*, vol. 21, no. 2, pp. 226–228, 1975.

[87] Aaron Wyner and Jacob Ziv, “The rate-distortion function for source coding with side information at the decoder,” *IEEE Transactions on information Theory*, vol. 22, no. 1, pp. 1–10, 1976.

[88] Enmao Diao, Jie Ding, and Vahid Tarokh (2020). "DRASIC Distributed Recurrent Autoencoder for Scalable Image Compression", Data Compression Conference (DCC) 2020.

3. Is the input to the ANN (autoencoder) always the same size? Is the input the whole image, a row or column, or is it a block of fixed size as in JPEG? *The input is the whole image, to which you apply a fixed-size convolutional block pixel by pixel.*

4. How is the distortion parameter managed? Are there several values tested and one is selected? Or is an optimal of some kind computed? It's chosen. *You set your tradeoff at will.*

5. For the NTC ANN training, probabilities are computed somehow. How is that done? Is a general abstract distribution computed by a batch and then used to assess, after the fact, the performance (as in the target function) of said batch? Is it in some way computed live? What distribution do those correspond to?