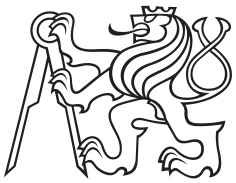
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Radioelectronics

Machine Learning Techniques for High Performance Image Compression

Metody strojového učení pro efektivní kompresi obrazu

Rudolf Studený

Supervisor: Ing. Fliegel Karel Ph.D.
January 2021

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Studený Rudolf** Personal ID number: **434707**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Radioelectronics**

Study program: **Electronics and Communications**

Specialisation: **Audiovisual and Signal Processing**

II. Master's thesis details

Master's thesis title in English:

Machine Learning Techniques for High Performance Image Compression

Master's thesis title in Czech:

Metody strojového učení pro efektivní kompresi obrazu

Guidelines:

Give an overview of recent methods for image compression using machine learning techniques (ML). Focus on the latest applications of deep learning (DL) methods. For the selected problem and test data, analyze the performance of the machine learning based methods compared to classical approaches.

Bibliography / sources:

[1] Chen, Z., He, T., Learning based Facial Image Compression with semantic fidelity metric, Neurocomputing, 338, 2019.
[2] Minnen, D., Ballé, J., Toderici, G., Joint autoregressive and hierarchical priors for learned image compression, Advances in Neural Information Processing Systems, 2018.

Name and workplace of master's thesis supervisor:

Ing. Karel Fliegel, Ph.D., Department of Radioelectronics, FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **14.02.2020** Deadline for master's thesis submission: **05.01.2021**

Assignment valid until: **30.09.2021**

Ing. Karel Fliegel, Ph.D.	doc. Ing. Josef Dobeš, CSc.	prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

. _____
Date of assignment receipt

_____
Student's signature

Acknowledgements

At this place I would like to give a big thank you to supervisor of this master thesis, Karel Fliegel. His calm nature and expertise in field results in very valuable points and advice, during origin of this paper.

Declaration

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

I hereby declare that I have written this master thesis independently and quoted all the sources of information used in accordance with methodological instructions on ethical principles for writing an academic thesis. Moreover, I state that this thesis has neither been submitted nor accepted for any other degree.

Abstract

This paper aims to create a guideline in growing field of neural networks, and their application in learning based system for effective compression methods. In first chapter looking to create theoretical foundation, as to understand the terms that we will further work with - such as machine learning, (deep) neural networks, deep learning etc. Further we then discuss application of these methods in field of image coding as well as discussing criteria for classification of such systems. Second chapter, the practical part then discovers various implementation of learning based codecs (e.g. using TensorFlow platform) and their performance, comparing with conventional methods (JPEG, JPEG2000). The paper is wrapped up by discussing the results and future possibilities for research.

Keywords: machine learning, image processing, compression methods, neural networks

Supervisor: Ing. Fliegel Karel Ph.D. Praha, Technická 2, B3-556

Abstrakt

Tato práce si klade za cíl vytvořit orientační návod stále rostoucím oborem strojového učení, neuronových sítí a jejich aplikace v systémech efektivních kompresních metod založených na učení. První kapitola tvoří teoretický základ, napomáhající k pochopení termínů se kterými se budeme nadále setkávat - t.j. strojové učení, (hluboké) neuronové sítě, hluboké učení atd. Dále budou diskutovány aplikace těchto metod právě v poli kódování obrazu, jakožto i návrh a diskuze kritérií pro klasifikaci těchto systémů. Následující kapitola, praktická část se věnuje různým implementacím kodeků, systémech založených na učení (např. s použitím platformy TensorFlow) a jejich efektivita v porovnání s konvenčními kompresními metodami (JPEG, JPEG2000). Práce je uzavřená diskuzí výsledků, návrhů zlepšení a možnostmi navazujícího výzkumu.

Klíčová slova: strojové učení, zpracování obrazu, kompresní metody, neuronové sítě

Překlad názvu: Metody strojového učení pro efektivní kompresi obrazu

Contents

Figures

Chapter 1

Introduction

With gradually increasing requirements for effective representation and compression of audiovisual content, for purpose of storing and sharing, we are always looking for new ways and methods to push the boundary of target quality and compression ratio achievable. With help of learning based algorithms, in-depth research of neural networks and development of efficient algorithms to train them - this appears to be a promising field to be applied in new compression systems; this is also supported by JPEG Committee forming a group, focused on exploring learning based image coding, so called JPEG-AI, which is aiming to develop an end-to-end compression system with potential for a standard to be defined. In this paper we would like to discuss state of the art compression methods, their design, architecture, as well as their performance in comparison with conventional systems and methods that are being commonly used on daily basis. Further a classification criteria would also be discussed.

Chapter 2

Theoretical part

Before delving deep into the topic at hand, let us briefly mention some important terms that might often be intertwined or substituted - such as *Artificial Intelligence, Neural Networks, Deep Learning* and more. For better understanding, let us refer to Venn diagram 2.1 suggested by [18], that is depicting the possible mutual connections of these terms, that we will further be working with.

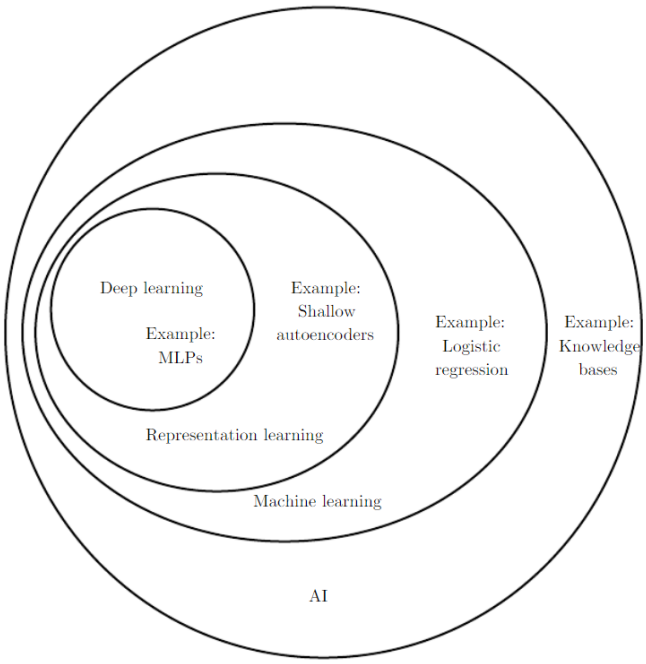


Figure 2.1: A Venn diagram showing the relations of fields like AI, machine learning, deep learning etc. Adapted from [18]

As is apparent from diagram above, the most superior field that we define and is related to this thesis (although way too broad) is **Artificial Intelligence** (AI). One of possible definitions define the field as the study of "intelligent agents", meaning any device that perceives its environment and takes actions that maximize its chance of successfully achieving its goals [39].

Formally, **machine learning** (ML) is a sub-field of artificial intelligence. However, in recent years, some organizations have begun using the terms artificial intelligence and machine learning interchangeably (hence the ambiguity of the terms). By ML we usually refer to a program or system that builds (trains) a predictive model from input data. The system uses the learned model to make useful predictions from new (never-before-seen) data drawn from the same distribution as the one used to train the model. It also can refer to the field of study concerned with these programs or systems [30]. We will discuss the topic in further detail in upcoming section 2.1.

Last two terms that we would like to mention in this section, which are closely linked is **Artificial Neural Networks** and **Deep Learning**. Former term refers to the biologically-inspired programming paradigm which enables a computer to learn from observational data. Latter term then describes a powerful set of techniques for learning in neural networks. Neural networks and deep learning currently provide the best solutions to many problems in image recognition, speech recognition, and natural language processing [37], and therefore being fundamental concepts for the aims of this thesis. These will be deeply mentioned in section 2.2 as well.

2.1 Machine learning

As our understanding of computers continues to mature, it seems inevitable that machine learning will play an increasingly central role in computer science and computer technology. [34] Let us describe some general ideas behind the term **Machine learning** (abbr. ML) and why is it perspective discipline in these mentioned fields.

2.1.1 General idea

With growing complexity of tasks that we deal with in modern computer science, programming, we naturally wish to use optimal approaches to reach the best possible efficiency. By using Machine learning methods, we gain time saving capabilities to process larger amount of data than ever before. Furthermore with traditional programming, our program might be limited or rather tailor-made to specific task that is defined apriori, without further adaptability to possible changes or exceptions that might come up in future. Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves. [56] This is to some extent analogous to human way of learning - we change our behaviour and approach to different tasks based on experience gained through our life.

To sum this up, let's use definition from [34]:

Definition 2.1.1 (Machine Learning)**.** A computer program is said to learn from experience E with respect to some class of tasks T and performance

measure P, if its performance at tasks in T, as measured by P, improves with experience E.

As was mentioned before, this might not be the only possible definition, but for our purpose of better understanding and it works well enough.

Let us propose one example from practice. The most significant early milestone was A. L. Samuel's study using the game of checkers. Samuel devised detailed procedures both of "rote-learning" and "learning by generalization". When coupled with efficient methods of look ahead and search, these procedures enabled the computer to raise itself by prolonged practice from the status of a beginner to that of a tournament player. Hence there now exists a checkers program which can learn through experience of checkers to play better checkers. [31] In general, to have a well-defined learning problem, we must identity these three features: *the class of tasks, the measure of performance to be improved, and the source of experience.* To demonstrate this on our checkers learning problem:

- Task T: playing checkers
- Performance measure P: percent of games won against opponents
- Training experience E: playing practice games against itself

We can specify many learning problems in this fashion, such as learning to recognize handwritten words, or learning to drive a robotic automobile autonomously. [34]

The final design of our checkers learning system can be naturally described by four distinct program modules that represent the central components in many learning systems. These four modules, summarized in 2.2, are as follows:

- The **Performance System** is the module that must solve the given performance task, in this case playing checkers, by using the learned target function(s). In our case, the strategy used by the Performance System to select its next move at each step is determined by the learned $\hat{V}$ evaluation function [1].
- The **Critic** takes as input the history or trace of the game and produces as output a set of training examples of the target function. As shown in the diagram, each training example in this case corresponds to some game state in the trace, along with an estimate V_{train} of the target function value for this example.
- The **Generalizer** takes as input the training examples and produces an output hypothesis that is its estimate of the target function. It generalizes from the specific training examples, hypothesizing a general function that covers these examples and other cases beyond the training examples.

[1] V is one particular target function among many that produces optimal play.

- The **Experiment Generator** takes as input the current hypothesis (currently learned function) and outputs a new problem (i.e., initial board state) for the Performance System to explore. Its role is to pick new practice problems that will maximize the learning rate of the overall system.

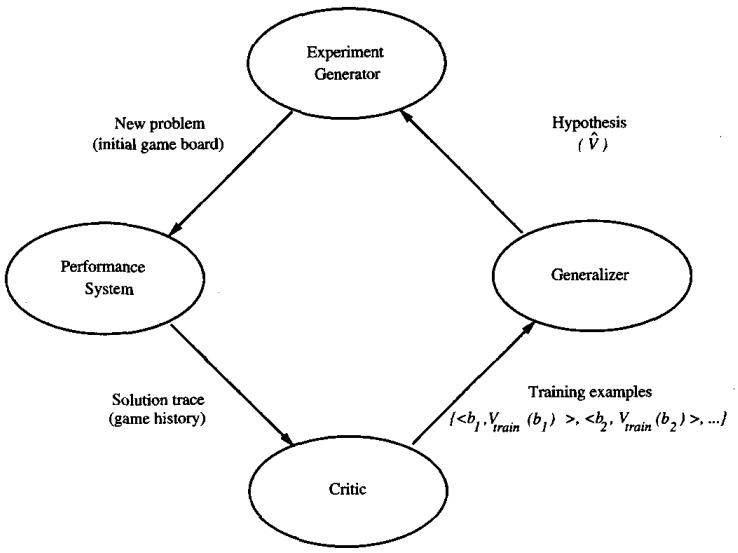


Figure 2.2: Design of checkers learning program. Adapted from [34]

2.1.2 Problem classification in Machine learning

In section 2.1.1 we discussed general ideas behind the the ML - as well as some basics terms - with help of an example of simple checkers "program". Let us take a step back now, and generalize the problems (as in types of tasks we deal with) that we define in modern ML methods.

Supervised vs. Unsupervised Learning

Often, we talk about ML as having two paradigms - *supervised* and *unsupervised learning*. However, it is more accurate to describe ML problems as falling along a spectrum of supervision between supervised and unsupervised learning. Nevertheless, for the sake of simplicity, we will be describing these two extremes.

In **supervised machine learning**, you feed the features and their corresponding labels into an algorithm in a process called training. During training, the algorithm gradually determines the relationship between features and their corresponding labels. This relationship is called the model. Often times in machine learning, the model is very complex. Further it finds patterns between data and labels that can be expressed mathematically as functions. Given an input feature, you are telling the system what the expected output

label is, thus you are supervising the training. The ML system will learn patterns on this labeled data. In the future, the ML system will use these patterns to make predictions on data that it did not see during training [8]. Such example of ML can be seen on figure 2.3 on the left. Below we will present some of the most representative classifiers [40]:

- *Perceptron* and *Logistic Regression (LR)* are probably the simplest linear classifiers. For both models, the model (i.e., weights and bias) is basically a simple linear transformation.
- *Artificial Neural Networks (ANN)* is a general extension of the aforementioned linear classifiers. Compared with Perceptron or LR which linearly project input data to the output, ANN has an additional "hidden layer" (with a non-linear activation function), which enables ANN to model non-linearity. We will go deeper into this type of classifier in section 2.2.
- *Decision Tree (DT)* [41] and *Random Forest (RF)* [6] are two tree-structure based non-linear classifiers. Based on certain attribute-splitting criteria (e.g., Information Gain or Gini Impurity), DT can analyse the most informative attributes sequentially (i.e., splitting) until the final decision can be made.
- *Support Vector Machine (SVM)* [9] is another popular supervised learning method, also called large margin classifier, as it aims at finding a hyperplane that is capable of separating the data points (belonging to different classes) with the largest margin. For non-linearly separable data sets, various kernels (e.g., RBF (Radial Basis Function)) can be applied into the SVM framework with good generalization ability.
- In comparison to SVM, we have *K-Nearest Neighbour (KNN)* [10], which does not require a training process (also referred to as lazy learning), is another powerful non-linear classifier. The classification is performed by distance calculation (between query and all the training examples), distance ranking, and majority voting among the (K) nearest neighbours.

In **unsupervised learning**, the goal is to identify meaningful patterns in the data (refer to the figure 2.3, right). To accomplish this, the machine must learn from an unlabeled data set. In other words, the model has no hints how to categorize each piece of data and must infer its own rules for doing so [8]. For example, the clustering algorithm can be used to find the potential patterns of some unlabelled data and the obtained results can be used for future analysis. [40]

K-Means [19] and *Principal Component Analysis (PCA)* [45] are the two most popular unsupervised learning algorithms. K-means aims to find K group patterns from data by iteratively assigning each sample to different clusters based on the distance between the sample and the centroid of each cluster. PCA is normally used for dimensionality reduction, which can de-correlate the raw features before selecting the most informative ones.

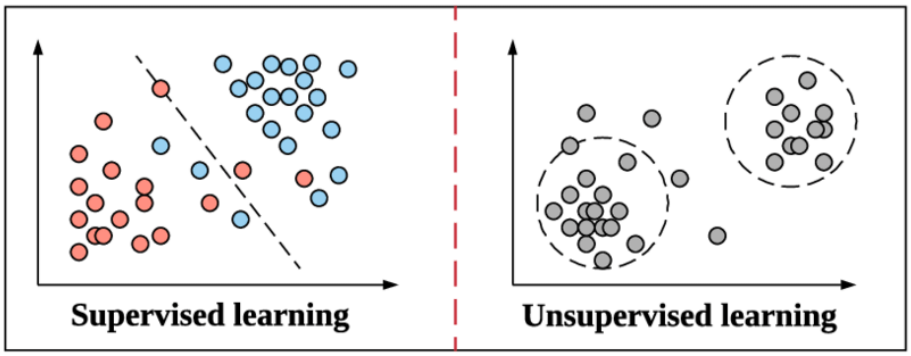


Figure 2.3: Examples of Supervised Learning (Linear Regression) and Unsupervised Learning (Clustering). Adapted from [40]

Reinforcement Learning

Another quite specific approach that is also worth to mention is **reinforcement learning**. The term by itself is somewhat ambiguous, in that it refers to simultaneously a problem, a class of solution methods that work well on the class of problems, and the field that studies these problems and their solution methods.

Reinforcement learning problems involve learning what to do—how to map situations to actions—so as to maximize a numerical reward signal. In an essential way they are closed-loop problems because the learning system's actions influence its later inputs. Moreover, the learner is not told which actions to take, as in many forms of machine learning, but instead must discover which actions yield the most reward by trying them out [46]. The lack of a data requirement makes RL a tempting approach. However, designing a good reward function is difficult, and RL models are less stable and predictable than supervised approaches [8].

To put this simply reinforcement learning focuses on "trial and error" style of training, by discovering best possible ways to reach the predefined goal of our task, while receiving positive or negative feedback, to determine how well is the learning agent doing.

2.2 Artificial Neural network

The Artificial Neural Networks (ANN) or more often simply **Neural Networks** are computing systems vaguely inspired by the biological neural networks [7]. Before we begin with the technical description, it would be useful to start with brief description of the biology of neural networks as we know it from living organisms.

2.2.1 Biological background of Neural Networks

The entire information processing system, i.e. the vertebrate nervous system, consists of the central nervous system and the peripheral nervous system, which is only a first and simple subdivision. In reality, such a rigid subdivision does not make sense, but here it is helpful to outline the information processing in a body.

The *peripheral nervous system (PNS)* comprises the nerves that are situated outside of the brain or the spinal cord. These nerves form a branched and very dense network throughout the whole body.

We would naturally like to focus on brain, as the main center of information processing. To keep things simple, we will divide brain into 4 main parts, as seen on 2.4.

The *cerebrum* (telencephalon) is one of the areas of the brain that changed most during evolution. Along an axis, running from the lateral face to the back of the head, this area is divided into two hemispheres, which are organized in a folded structure.

These cerebral hemispheres are connected by one strong nerve cord ("bar") and several small ones. A large number of **neurons** are located in the *cerebral cortex* (cortex) which is approx. 2-4 cm thick and divided into different *cortical fields*, each having a specific task to fulfill. *Primary cortical fields* are responsible for processing qualitative information, such as the management of different perceptions (e.g. the *visual cortex* is responsible for the management of vision). *Association cortical fields*, however, perform more abstract association and thinking processes; they also contain our memory. [27]

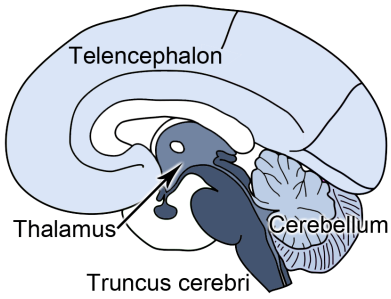


Figure 2.4: Illustration of the brain. Adapted from [27]

To close out this section, we will focus on fundamental description of said neuron, which we can consider as a basic processing unit the brain. A neuron is nothing more than a switch with information input and output. The switch will be activated if there are enough stimuli of other neurons hitting the information input. Then, at the information output, a pulse is sent to, for example, other neurons. To put it simply:

- *Synapses* weight the individual parts of information
- *Dendrites* then collect all parts of this information

- In the *soma* the weighted information is accumulated
- And the axon transfers outgoing pulses

All of the parts mentioned above can be found depicted in figure

2.2.2 Artificial neuron

Now let us put the knowledge gained in previous section 2.2.1, into context of basic concepts of ANNs, their structure and design.

The fundamental unit the forms our neural network is once again neuron, be it artificial one, called **perceptron**. To be more precise, perceptron is a simple artificial neuron whose *activation function* consists of taking the total net input and outputting 1 if this is above a threshold T, and 0 otherwise[57]. We will describe what is activation function further in this chapter. Perceptrons were developed in the 1950s and 1960s by the scientist Frank Rosenblatt, inspired by earlier work by Warren McCulloch and Walter Pitts. Today, it's more common to use other models of artificial neurons, and the main neuron model used is one called the *sigmoid neuron* [37].

Very similarly as with biological neuron, a perceptron takes several binary inputs, $x_1, x_2, ..., x_n$ and produces a single binary output:

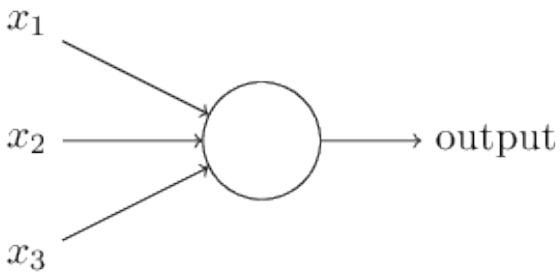


Figure 2.5: Simple illustration of perceptron. Adapted from [37]

Example above on figure 2.5 shows perceptron with three inputs x_1, x_2, x_3, which is in this case completely arbitrary, and in general perceptron can have fewer or more inputs. Rosenblatt proposed a simple rule to compute the output. He introduced weights, $w_1, w_2, ..., w_n$ real numbers expressing the importance of the respective inputs to the output. The neuron's output, 0 or 1, is determined by whether the weighted sum $\sum_j w_j x_j$ is less than or greater than some threshold value, arithmetically put:

$$\text{output} = \begin{cases} 0, & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1, & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases} \tag{2.1}$$

That's the basic mathematical model, although perceptron isn't a complete model of human decision-making. Connection between the neurons carries the information that is to be processed. Naturally our next question should be - how is the incoming information processed? For our neuron j, [27] defines *propagation function* converting vector inputs to scalar network inputs:

Definition 2.2.1 (Propagation function and network input). Let $I = \{i_1, i_2, ..., i_n\}$ be the set of neurons, such that $\forall z \in \{1, ..., n\} : \exists w_{i_z,j}$. Then the network input of j, called net_j , is calculated by the propagation function f_{prop} as follows:

$$\text{net}_j = f_{\text{prop}}(o_{i_1}, ..., o_{i_n}, w_{i_1,j}, ..., w_{i_n,j}) \tag{2.2}$$

Here the *weighted sum* is very popular: The multiplication of the output of each neuron i by $w_{i,j}$, and the summation of the results:

$$\text{net}_j = \sum_{i \in I} (o_i \cdot w_{i,j}) \tag{2.3}$$

At every moment, every neuron in network happens to be in some state, it's "active" so to speak. This state further defines its reaction on incoming information, and we refer to it as activation state or more often just shortly *activation*. [27] provides following general definition:

Definition 2.2.2 (Activation state / activation). Let j be a neuron. The activation state a_j is explicitly assigned to j, indicates the extent of the neuron's activity and results from the activation function.

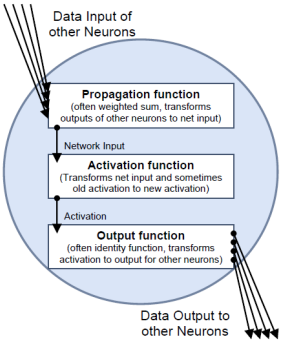


Figure 2.6: Information processed in neuron. Adapted from [27]

So the current activation state of neuron is based on the activation state as well as the input, incoming information. This relation is described by *activation function* [27]:

Definition 2.2.3 (Activation function). Let j be a neuron. The activation function is defined as

$$a_j(t) = f_{\text{act}}(\text{net}_j(t), a_j(t-1), \Theta_j). \tag{2.4}$$

It transforms the network input net_j, as well as the previous activation state $a_j(t-1)$ into a new activation state $a_j(t)$, with the threshold value playing an important role.

Unlike the other variables within the neural network the activation function is often defined globally for all neurons or at least for a set of neurons and only the threshold values are different for each neuron. The threshold value can also be changed during time, e.g. by learning procedure. [27] As for the most common activation functions: the most simple one is the *binary threshold function*, also well known as Heaviside function (refer to figure 2.7) - value changes from one to another, once overcoming given threshold; otherwise it stays constant. Other very popular functions are *Fermi function* (also logistic function), possibly expanded by a temperature agent T

$$\frac{1}{1 + \text{e}^{\frac{-x}{T}}} \tag{2.5}$$

which gives us the possibility to change the gradient of the function and approximate the Heaviside function. Another function would be *hyperbolic tangent*. Both lastly mentioned, in contrast to Heaviside function, are differentiable (fig. 2.7). Incidentally, there exist activation functions which are not explicitly defined but depend on the input according to a random distribution - *stochastic activation function*. [27]

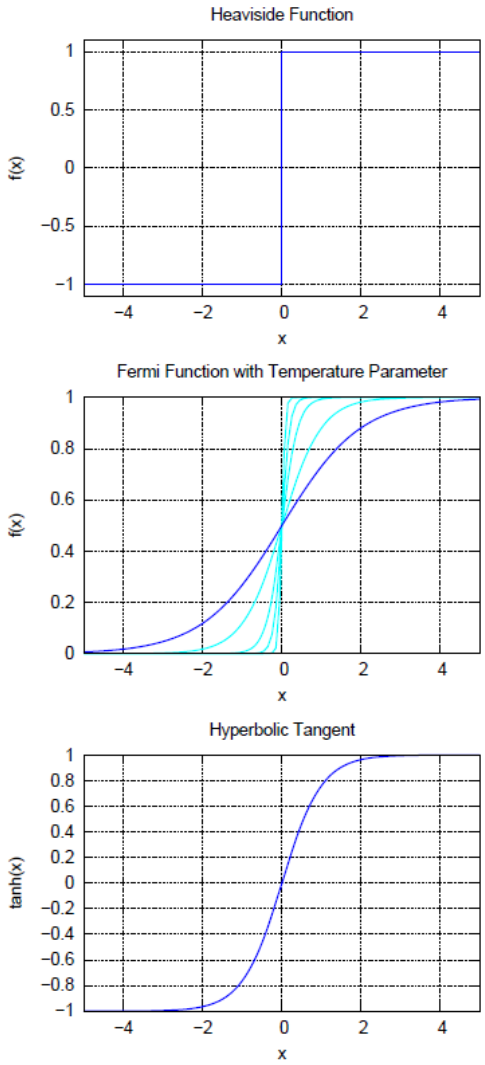


Figure 2.7: Example of various activation functions. Adapted from [37]

Lastly we define *output function* of a neuron j calculates the values which are transferred to the other neurons connected to j:

Definition 2.2.4 (Output function)**.** Let the j be a neuron. The output function

$$f_{\text{out}}(a_j) = o_j \tag{2.6}$$

calculates the output value o_j of the neuron j from it's activation state a_j.

Generally, the output function is defined globally, too. Often this function is the identity, i.e. the activation a_j is directly output:

$$f_{\text{out}}(a_j) = a_j, \text{ so } o_j = a_j \tag{2.7}$$

Let us illustrate an example, how a perceptron can weigh up different kinds of evidence in order to make decisions - this is plausible with a complex network of perceptrons that could make quite subtle decisions:

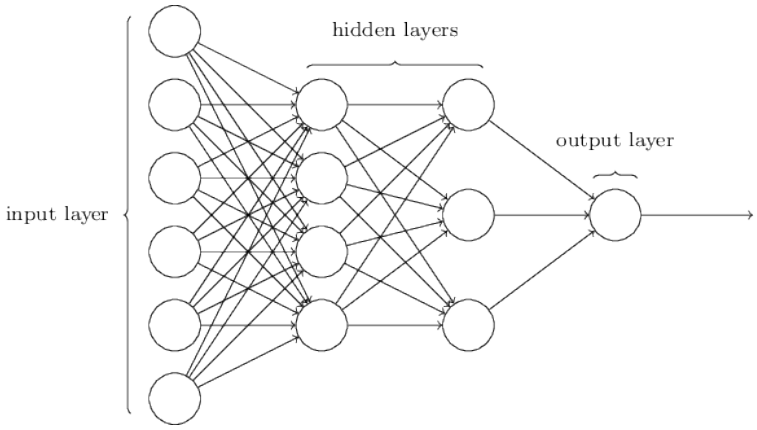


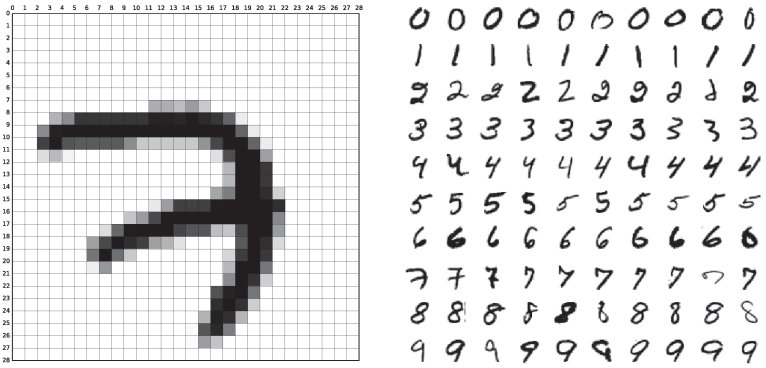
Figure 2.8: Illustration of multilayered neural network. Adapted from [37]

The leftmost layer in this network is called the input layer, and the neurons within the layer are called input neurons. The rightmost or output layer contains the output neurons, or, as in this case, a single output neuron. The middle layer is called a hidden layer, since the neurons in this layer are neither inputs nor outputs [37]. Therefore an example in figure 2.8 is four-layer network, with two hidden layers.

While the design of the input and output layers of a neural network is often straightforward, there can be quite an art to the design of the hidden layers. In particular, it's not possible to sum up the design process for the hidden layers with a few simple rules of thumb. Instead, neural networks researchers have developed many design heuristics for the hidden layers, which help people get the behaviour they want out of their nets. For example, such heuristics can be used to help determine how to trade off the number of hidden layers against the time required to train the network. [37]

Let us propose a simple example how to interpret the structure of simple neural network for recognition of handwritten numbers. Taking the **MNIST data set of handwritten digits** [28] - a single d igit would be represented by field of pixels (28x28). Using simple black and white representation, therefore just binary one, where 0 means white (blank space) and 1, black (written part of the number). This easily fit our input (leftmost) layer of neurons (perceptrons), specifically $28 \cdot 28 = 784$ of them; one for each pixel in image on the input. (Refer to the figure 2.9)

Output layer then can consist of 10 neurons, each representing digits from 0 to 9, and their values in boolean logic - 0 (false) or 1 (true).



(a) MNIST sample belonging to the digit '7'. (b) 100 samples from the MNIST training set.

Figure 2.9: Figure (a) shows closeup digital representation of single digit (seven); figure (b) then shows 100 samples from MNIST data set. Adapted from [3]

2.2.3 Learning and efficiency

Since we are aiming to create an efficient model suitable to given task, we'd like is an algorithm which lets us find *weights* and *biases* so that the output from the network approximates $y(x)$ for all training inputs x. To quantify how well we're achieving this goal we define a **cost function** [2]:

$$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2 \tag{2.8}$$

Here, w denotes the collection of all weights in the network, b all the biases, n is the total number of training inputs, a is the vector of outputs from the network when x is input, and the sum is over all training inputs, x. This is actually nothing else than *mean squared error (MSE)* [3]. This means that objectively the goal of our training algorithm would be to keep the cost function as low as possible, $C(w,b) \approx 0$. The lower the MSE value, the closer the output value is to the correct result.

The next milestone on our way is - how do we find this minimum of cost function? This is reached by using an algorithm called *gradient descent*. Suppose we are trying to minimize cost function $C(v)$. This could be any real-valued function with arbitrary amount of values: let's say C is a function of m variables, $v_1, v_2, ..., v_m$. Then change ΔC in C produced by a small change $\Delta v = (\Delta v_1, ..., \Delta v_m)^T$ is

$$\Delta C \approx \nabla C \cdot \Delta v, \tag{2.9}$$

where gradient ∇C is the vector

$$\nabla C \equiv (\frac{\partial C}{\partial v_1}, ..., \frac{\partial C}{\partial v_2})^T. \tag{2.10}$$

We will further also define

[2] Sometimes referred to as a loss or objective function.
[3] Or quadratic cost function.

$$\Delta v = -\eta \nabla C, \tag{2.11}$$

where η is a small, positive parameter (known as the learning rate). Then equation 2.9 tells us that $\Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta \|\nabla C\|^2$. Because $\|\nabla C\|^2 \geq 0$, this guarantees that $\Delta C \leq 0$, i.e., C will always decrease, if we change v according to the prescription in 2.11.

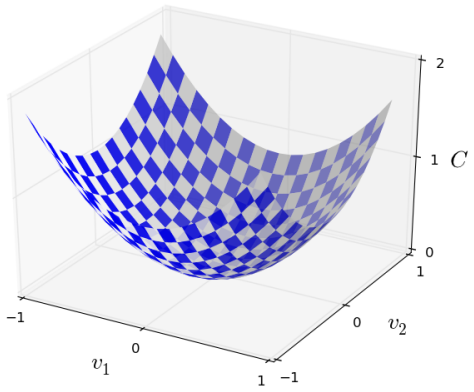


Figure 2.10: 3-D plot of two variable cost function. Adapted from [37]

To put this into a better perspective, let's present a case, where $C(v)$ being function of two variables v_1, v_2 and as we know, our goal is to find a global minimum of said function $C(v)$. By one glance at 3-D plot 2.10 of presented function $C(v)$, we can immediately recognize there exist ideal combination of variables v_1, v_2, the would give us the desired result [37].

The way the gradient descent algorithm works is - by making small changes (defined by step η) and evaluating change Δv using equation 2.11 - to repeatedly compute the gradient ∇C, and then to move in the opposite direction, "falling down" the slope of the valley. This concept is then upscaled to arbitrary amount of values, which takes us back to 2.10.

2.2.4 Deep Learning

It is worth mention what do we understand by the term **deep learning** and deep neural networks (DNNs) since this term is being more and more common while talking about modern learning-based methods.

Since idea of ANNs, that we started to described in previous sections, isn't anything new, it's no surprise that the whole field developed during the years. More often then not, we can see neural networks that doesn't have just a few stages, let's say single hidden layer - that is something we call *shallow neural network* models. We often look for more complex solutions to describe our task; extracting smaller, more detailed (but also more abstract) features.

Lately we often achieve this by stacking hidden layers, leading to so called *deep neural networks* (DNNs).

Unsurprisingly this also brings a question of how to (effectively) train such NNs. Learning (or sometimes called credit assignment) is about finding weights (as described in section 2.2.3) that make the NN exhibit desired behavior, such as driving a car for instance. Depending on the problem and how the neurons are connected, such behavior may require long causal chains of computational stages, where each stage transforms (of ten in a non-linear way) the aggregate activation of the network. Deep Learning is about accurately assigning credit across many such stages. BP-based training of deep NNs with many layers, however, had been found to be difficult in practice by the late 1980s and had become an explicit research subject by the early 1990s. DL became practically feasible to some extent through the help of Unsupervised Learning (section 2.1.2), further in he 1990s and 2000s also saw brought many improvements of purely supervised DL and lastly Deep NNs also have become relevant for the more general field of Reinforcement Learning (2.1.2). [43]

2.2.5 Topology of ANN

This section is going to focus on common topologies of neural networks and their names, often abbreviated. The fact that abbreviations are so commonly spread in this field, makes it rather hard and overwhelming for inexperienced person to orientate himself. Following chapter will often refer to very useful sheet by Fjodor van Veen [53], providing well sorted information and visual representation of topic at hand. Another note - list that will follow further below won't (and almost can't be) complete, but should provide reader with basic idea of structure and design of NNs, especially those applicable for image processing (compression).

Feed Forward Neural Networks

Feed forward neural networks (FF or FFNN) and perceptrons represent the simplest examples of neural networks. We already described perceptrons in more details under section 2.2.2, but very briefly: it is the fundamental unit of neural networks, defined by weighted inputs from other neurons, activation function and its output.

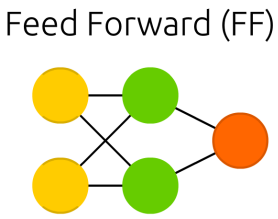


Figure 2.11: Feed forward neural network. Adapted from [53]

Putting neurons into layers, we create network. Each layer containing input, hidden or output cells in parallel. FF network, as the name suggests, is a one-directional network - feeding the information from front and it propagates to the end, towards output cells on the right. All nodes are fully connected, meaning that every node in given

layer has connections to all nodes in the next one. The applications of FFNNs are somewhat limited and they are often paired with other types of networks for more complex tasks. By itself, they can be used as model for classification.

The decision-making of the network lies in the hidden layer in the middle. As was mentioned in 2.2.2, every neuron has weights, which represents the decision-making capabilities, once the information is being propagated through them. These weights are set by process called *training*, during which we feed the network the data on the input and pair them with expected data on the output (so called supervised learning, 2.1.2). We can go back, to the input layer, layer by layer comparing layer input with layer error. The error being propagated is often some variation of the difference between the input and the output (like MSE or just the linear difference) [53]. Each layer, no matter how deep it is, contributes to next layer error; therefore, we can adjust weights by value we can get by multiplying previous layer output by output error multiplied by current layer output.[47] This process is called **Backpropagation**.

Amongst some variations to general FF network we put: **Radial Basis Network** (RBF), which is FF network that uses *radial basis function* [4] as an activation function.2.2.3.

Another example is **Deep feed forward network** (DFF) pioneering the way for deep learning in early 90s. General idea is the same as for FFNs, adding more hidden layers (therefore *deep*). Although stacking hidden layers led to exponential growth of training time causing them to be impractical, with forthcoming more efficient approaches in early 00s, they now form a core of modern ML systems.[47][48]

Recurrent Neural Networks

Recurrent Neural Networks (RNN) brings new type of nodes, cells - recurrent cells. Our goal is to represent time implicitly by its effects on processing rather than explicitly (as in a spatial representation) [14]. This idea first described by Jordan, M.I. in 1986 [22] allows us to involve the use of recurrent links in order to provide networks with a dynamic memory.

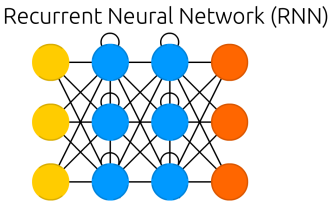


Figure 2.12: Recurrent neural network. Adapted from [53]

Neurons are fed information not just from the previous layer but also from themselves from the previous pass. This also means that order in which data are fed to the network matter - feeding it "dog" then "cat" may yield different results compared to opposite order. One big problem with RNNs is the vanishing (or exploding) gradient problem where, depending on the activation functions used, information rapidly gets lost over time, just

[4] real-valued function φ whose value depends only on the distance between the input and some fixed point, either the origin, or some other fixed point **c** [12]

like very deep FFNNs lose information in depth. To put it in perspective: if the weight reaches a value of 0 (vanishing) or 1 000 000 (exploding), the previous state of the cell won't be very informative. In general, recurrent networks are a good choice for advancing or completing information, such as autocompletion. [53]

Variation of RNNs called **Long / short term memory** (LSTM) aims to deal with mentioned problem with exploding/vanishing gradient. This is achieved by an efficient, gradient-based algorithm for an architecture enforcing constant (thus neither exploding nor vanishing) error flow through internal states of special units - *memory cells*. [20] These cells are made composed of so called *gates*, which define whether to feed the information forward (and how much of it) or erase/forget it. [53] They are used in fields such as speech and writing recognition.

Gated recurrent units (GRU) then are very much similar to LSTMs by design. They are wired slightly differently, with different gating - introducing *update* and *reset* gates. GRUs are slightly less expressive, but a bit faster; commonly used in speech (or sound in general) synthesis.

To all networks mention above in this section also exist a bidirectional variation, called **Bidirectional recurrent neural networks, bidirectional long / short term memory networks and bidirectional gated recurrent units** (BiRNN, BiLSTM and BiGRU respectively). As for the topology itself, they appear to be all them same. The difference is in fact that they can be trained without the limitation of using input information just up to a preset future frame. This is accomplished by training it simultaneously in positive and negative time direction. This trains the network to fill in gaps instead of advancing information, e.g. instead of expanding an image on the edge, it could fill a hole in the middle of an image. [44][53]

Autoencoders

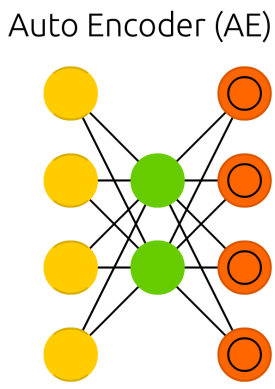


Figure 2.13: Node map of an autoencoder network. Adapted from [53]

Autoencoders (AE) are considerably similar by their design to FFNNs, applied a bit differently. Their aim to encode (compress) the information on the input. Structurally they resemble the hourglass-like shape (as can be seen on 2.13), and are symmetrical around the (hidden) layer in the middle. This smallest layer (or multiple layers potentially) are the place with the highest grade of compression of the information, and it is so called *bottleneck* or *chokepoint* of the network. Everything up to the middle layer is called the encoding part, everything after the decoding and the middle the code. [53] Bourlard (2000) shows that the nonlinearities of these hidden units are useless, and that the optimal parameter values can be derived directly

by purely linear techniques relying on singular value decomposition and low rank matrix approximation, similar in spirit to the well-known Karhunen-Loève transform (KLT). [5] Autoencoders are normally used for classification, clustering and feature compression.[48]

Denoising autoencoders (DAE) is variation of AE networks, with a little spin. Vincent at al. (2008) is investigation specific criterion of AEs called *robustness to partial destruction of the input* (basically ability to restore data with presence of noise on the input). The motivation behind is such: a good representation is expected to capture stable structures in the form of dependencies and regularities characteristic of the (unknown) distribution of its observed input. For high dimensional redundant input (such as images) at least, such structures are likely to depend on evidence gathered from a combination of many input dimensions. They should thus be recoverable from partial observation only. [54]

In simple terms this means that by training with the presence of noise, the model is encouraged to focus on more broad features in the data (e.g. image), rather than details, which are often shifting, changing. This leads to higher robustness to noise.

Sparse autoencoders (SAE) is in a way opposite to standard AEs. The idea is that instead of compressing the information in the middle (hidden) layer containing less nodes than input/output layer, we expand the information through bigger amount of nodes, as seen in 2.14. These types of networks can be used to extract many small features from a data set. To prevent getting identity network each time, during training we feed back the input plus a *sparsity driver*, that can take the form of a threshold filter, where only a certain error is passed back and trained, the other error will be "irrelevant" for that pass and set to zero. [53][42]

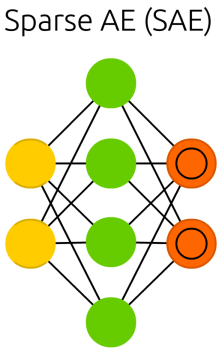


Figure 2.14: Sparse autoencoder map. Adapted from [53]

Variational Autoencoders

The **Variational Autoencoders** (VAE) can be viewed as two coupled, but independently parameterized models: the encoder or recognition model, and the decoder or generative model. These two models support each other. The recognition model delivers to the generative model an approximation to its posterior over latent random variables, which it needs to update its parameters inside an iteration of "expectation maximization" learning. Reversely, the generative model is a scaffolding of sorts for the recognition model to learn meaningful representations of the data, including possibly class-labels. The recognition model is the approximate inverse of the generative model according to Bayes rule.

The VAE is inspired by the Helmholtz Machine (Dayan et al., 1995[11]) which was perhaps the first model that employed a recognition model. How-

ever, its wake-sleep algorithm[5] was inefficient and didn't optimize a single objective. The VAE learning rules instead follow from a single approximation to the *maximum likelihood objective*. [24]

Convolutional Neural Networks

Convolutional neural networks (CNN or deep convolutional neural networks, DCNN) first described by LeCun et al. (1998) [29] are conceptually bit different. They are often used for image processing, recognition (or audio), where other networks such as FFNNs could bring big disadvantages.

Firstly, data (images) on the input are typically large, let's say several hundred pixels (variables). Fully-connected first layer with 100 hidden cells would then already contain several tens of thousands weights; increasing capacity of system, as well as memory requirements and would require large training set. Although with latest development in CNNs Valueva et al. (2020) [52] proposed convolutional neural network architecture in which the neural network is divided into hardware and software parts to further increase performance and reduce the cost of implementation resources. Secondly, images (or time-frequency representation of audio signal, speech) have a strong 2D local structure: variables that are spatially or temporally nearby are highly correlated. Therefore these correlations are advantageous for extracting and combining *local features*. CNNs force the extraction of local features by restricting the receptive fields of hidden units to be local. [29]

Simply put we start with *scanning layer* with a smaller dimension in comparison to input, that is moving (scanning) over the input image, feeding these data into convolutional layers. These convolutional layers also tend to shrink as they become deeper, mostly by easily divisible factors of the input (e.g. if we have layer of 20x20, it's likely that would be followed by 10, and 5). Besides these convolutional layers, they also often feature pooling layers. Pooling is a way to filter out details: a commonly found pooling technique is max pooling, where we take say 2 by 2 pixels and pass on the pixel with the most amount of red. [53]

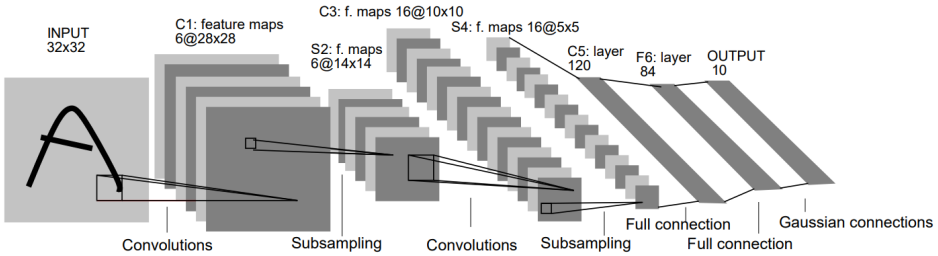


Figure 2.15: Architecture of LeNet-5, a CNN, here for digits recongnition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical. Adapted from [29]

[5] an unsupervised learning algorithm most common with Helmholtz machines[35]

Generative Adversarial Networks

Framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game. In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to $frac12$ everywhere. In the case where G and D are defined by multilayer perceptrons, the entire system can be trained with *backpropagation*. [17]

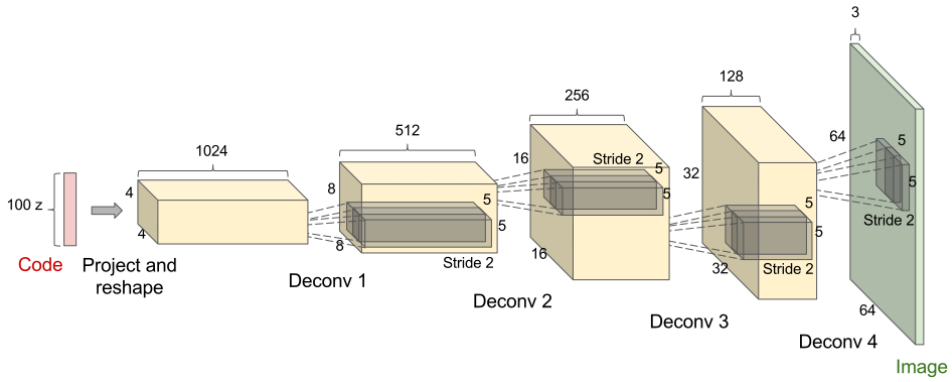


Figure 2.16: Image generated by deconvolutional layers. Adapted from [23]

In simple terms: GAN is actually two networks designed to compete against each other. Discriminative network D determines whether the input produced by the generative network G is good enough (difference from reference data); through backpropagation G adjust its weights until maximal quality solution.

2.3 Applications in image compression

In recent years learning based methods and especially deep learning approaches have achieved a great success in many computer vision tasks, and are gradually used in **image compression**.

With increasing quality and fidelity of visual content, the raw volume of such data drastically increase. This content is then stored and/or shared; accordingly, our bandwidth requirements while consuming/sharing content, makes *effective* compression methods huge priority.

In general we distinguish between two main groups of methods - **Lossless** and **lossy**. Lossless refers to compression methods, where data before and after compression are identical, without losing any information, quality of our content. On the contrary, lossy methods refers to such methods, that have impact on the output data and we lose some of the initial information. While working with audiovisual content we further differentiate between

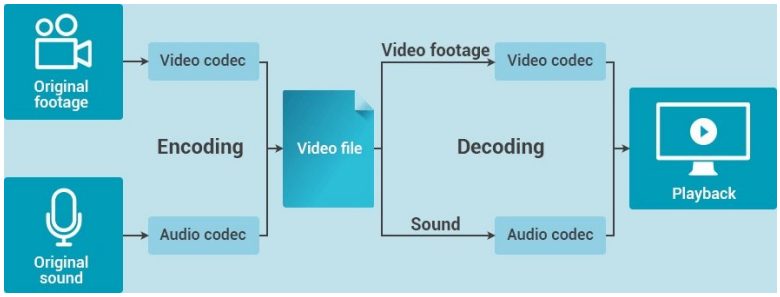


Figure 2.17: Motivation for data encryption/decryption - storing or further sharing. Adapted from [49]

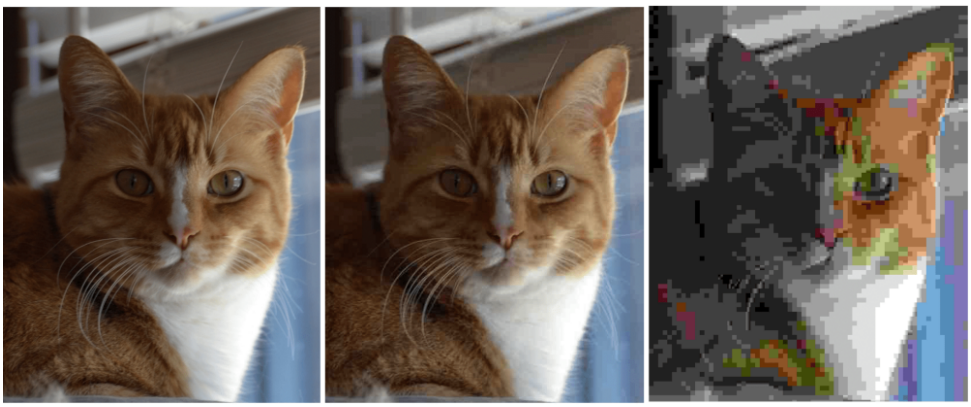


Figure 2.18: Example of JPEG compression. (Left) raw, uncompressed image, (middle) medium level compression, perceptually lossless, (right) highly compressed image. Adapted from [38]

perceptually lossless (with limited or non impact on subjective quality) and perceptually lossy.

Our main goal while designing compression methods (and it's implementation - encoder and decoder) is generally to minimize the final size of our content, while maximizing the subjective quality as close as possible to quality of raw uncompressed data. Natural images tend to contain a high number of redundant information; neighbouring pixels tend to exhibit a correlation [25]. Most image compression algorithms tend to take advantage of this correlation to reduce the size of images stored. [16]

Many models and methods aims to enhance conventional compression systems or in some cases substitute them completely by introducing whole end-to-end image compression systems. This brings its pros mostly from performance point of view - very efficient systems, achieving high compression ratios, high subjective quality etc.

On the other hand we have to also consider the cons such as: preparing big enough data set, containing enough examples representative of the information, features we aim to handle (or rather, to be handled by our network); training time needed for model preparation - this heavily corresponds with previously mentioned size of training data, as well as the user's hardware capabilities or

complexity (depth) of the network we are training.

2.3.1 Examples from practice

This section should represent some real implementations that have been proposed, and that would further be part of a performance comparison in practical part of this paper.

Paper of Minnen et al. (2018) [32] introduces they learning-based compression method; they network basically consist of two sub-networks, each having a different purpose. The first is the core *autoencoder*, which learns a quantized latent representation of images (Encoder and Decoder blocks). The second sub-network is responsible for learning a probabilistic model over quantized latents used for entropy coding. It combines the Context Model, an autoregressive model over latents, with the hyper-network (Hyper Encoder and Hyper Decoder blocks), which learns to represent information useful for correcting the context-based predictions. The data from these two sources is combined by the Entropy Parameters network, which generates the mean and scale parameters for a conditional Gaussian entropy model.

Toderici et al. (2016) proposed an architecture (fig. 2.19)with RNN-based encoder and decoder, a binarizer, and a neural network for entropy coding. Their goal was to provide a neural network which is competitive across compression rates on images of arbitrary size. Also most of autoencoders of the time were defined by fixed compression rate, based on the size of bottleneck layer in the middle. [50] extends this idea by supporting variable rate compression while maintaining high compression rates beyond thumbnail-sized images.

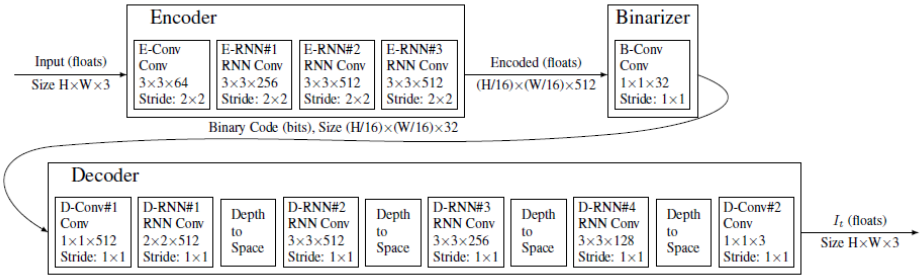


Figure 2.19: Single iteration of shared RNN architecture from [50]

While the network weights are shared between iterations, the states in the recurrent components are propagated to the next iteration. Therefore residuals are encoded and decoded in different contexts in different iterations. Proposed network incorporates LSTM units, Associative LSTMs [6] (experimentally proved to be effective only in decoder) and GRU - computation block that passes residual information around the block in order to speed up convergence.

[6] Associative LSTM extends LSTM using holographic representation

2.3.2 Quality assessment - metrics

The most widely used full-reference[7] image quality and distortion assessment algorithms are peak signal-to-noise ratio (PSNR) and mean squared error (MSE), which do not correlate well with perceived quality. That is a main reason for developement of method called **Multi-scale structural similarity index measure** (abbr. MS-SSIM), that will be further used as main metric of quality.

Structural similarity provides an alternative and complementary approach to the problem of image quality assessment.[59][58] The perceivability of image details depends the sampling density of the image signal, the distance from the image plane to the observer, and the perceptual capability of the observer's visual system. A older single-scale method may be appropriate only for specific settings and is surpassed by multi-scale version that is a convenient way to incorporate image details at different resolutions. [55]

As was mentioned before, more simple metrics such as **MSE** or **PSNR** might not as descriptive for the subjective quality assessment, nevertheless they still might be useful as indicators, e.g. during training our network. So for the image I ($m \times n$ pixels) and its reconstruction K, we define:

$$\mathrm{MSE} = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2 \tag{2.12}$$

$$\mathrm{PSNR} = 10 \cdot \log(\frac{\mathrm{MAX}_I^2}{\mathrm{MSE}}) \tag{2.13}$$

2.3.3 Data sets

To precisely define the objective quality of compression method at hand, we should also use suitable testing data, broad enough, various so we can define what and where are the strengths and weaknesses.

From my research, the most common data set used by many papers is still **Kodak Lossless True Color Image Suite**[26], which contains 24 images in so called "full colour" (24 bits per pixel, 8 bits per colour layer) contained in lossless format PNG. Each image is of size 768 × 512 pixels (both dimensions divisible by 32, commonly used factor in compression systems). This set was released by the Kodak Corporation for unrestricted research usage in 1999.

Although possibly prevalent, it's hardly the only data set, so let us also propose other options.

- **Tecnick** [2] containing data set of 100 images, size 1200 × 1200.
- **MCL-JCI** [21] has 50 images of 1920 × 1080 with 8-bit depth that can be classified into ten semantic categories such as people, animals, plants, buildings, water or lake, sky, bridge, transportation vehicles (boats or cars) and indoor.

[7] based on an initial uncompressed or distortion-free image as reference

- **LIVE in the Wild** [15] with 1162 distorted images from different mobile devices (therefore different sizes) Images of faces, people, animals, close-up shots, wide-angle shots, nature scenes, man-made objects, images with distinct foreground/background configurations and without any object of interest.

Many others could be listed, the choice heavily depends on the expected task, what would be the subject of our study.

2.4 Classification proposal

This paper also aims to propose a taxonomy, how to classify and orient yourself in vest field of neural networks, based on various criteria. This would also help us with even field on which we could compare various networks and their implementations. We have to consider that we are discussing very broad field and different perspectives should be pointed out. Nelson and Rogers (1992) [36] points out, that there might be different criteria to chose as *best* per se, for the same task. Therefore the final criteria should be selected by the researcher or application specialist depending on the results desired. Their paper also proposes division of criteria into 3 main groups - topology, training and performance.

2.4.1 Topology

Let us start with topological point of you; where we are considering the architecture, design of the network itself. In section 2.2.5 we've been describing different types of network topologies [13] simply defines category *Neural network type*, i.e. top level description of the network. Our aim is to create efficient compression method - do we use AE network? Or some variaton, such as VAE? Or possibly we could train model through combination of networks, e.g. GANs? This is also connected to what type of units/neurons/nodes do we define in our network and their activation functions (2.2.3). Also, what is the extent of our network in means of end-to-end image compression system - NN focused on encoding residual part, the entropy coder or combination of both?

If we were to dissect this idea into smaller problems, we should discuss the complexity of the network. **How many (hidden) layers** should our NN - shallow or deep NN? Considering solely the amount of layers, while assuming that same performance regardless of this amount, we would prefer *fewest layer amount possible*.

How many nodes do we have per layer? Generally speaking the more nodes a network has the greater the tendency to memorize the training set and the less tendency to properly generalize. The network with the *fewest nodes*, irrespective of how they are arranged or connected could be selected by this criterion. [36]

And lastly, what is the **amount of connections between the nodes** in layer. This affects networks ability to solve given (or possibly its inability to do so) while using too few interconnections. In opposite case the network tends to produce noise in the system. So we are looking for optimal solution with *fewest possible interconnections*.

2.4.2 Training

Training, or learning is unmistakably integral part of neural networks, that has to be considered, while comparing with other systems. Also, because the training phase is usually the one which requires the most time, this is a good area to make comparisons according to [36].

First criterion that comes to mind could be the **time efficiency** - how fast can the NN be trained, converging to the optimal solution. If we would discuss the fact, that learning is iterative process, and we are feeding the training data to the network in cycles, converging to expected solution; that aim should be to do so in *least epoch/iterations as possible*.

Further we can also talk about the actual real **clock time**, that would be consumed by training on given hardware. With growing complexity and depth of NNs, the required hardware capabilities are to be consider. All of this is in the end defined by the time spent, which we expect to be *as short as possible*.

Training also involves the **requirements for data set**, that would be descriptive enough for our task. This means, that we need to feed the network data of some variety, in cases of general tasks, e.g. while training NN for image compression, feeding the network only portraits of people, we couldn't expect good results while compressing (and mainly reconstructing) the image of scenery in nature. The features extracted might not be general enough for both, therefore model would have more specific application. Summing up the idea, our criterion would be *smallest possible data set*, that would fit the expected results.

Nelson and Rogers [36] describe a **computational complexity** of operations during one epoch of learning algorithm, providing example of so called Hammering and Stretching Algorithm. This method uses just one epoch to train, however requires the construction and inversion of a very large matrix, which might be detrimental during training on bigger data sets. This implies our desire of *least possible computational complexity* - the amount of operations needed.

2.4.3 Performance

The performance is logical context to be discussed as our main aim while designing learning-based solutions to our task should be based on efficiency. We already discussed time efficiency, complexity and now quality of final result, produced by our network. **How accurate** is the representation, in our case of this paper: the reconstructed image once decompressed. This

might be different based on the input data and their relation to the training data set. What we mean by this, is the possibility that we are might try to process (compress) content similar to that provided during training, or vice versa; this might (or might not!) impact the resulting quality, e.g. in means of MSE or MS-SSIM in our case. Which naturally leads us to another criterion, which is the ability of network to **generalize** from the training data set. As a result, some researches include not only training data set, but also evaluation data set for purpose of accurate quality assessment (which in our case is represented by Kodak image set [26]).

We can also describe the **dependency on initial setup of weights** in our NN. Since a lot of networks start training phase with random weights, it leads to the different starting point of finding the optimal solution to our task, therefore best possible network. This implies that there's possibility of training getting stuck on a local minima, instead of being able to find the most optimal global minimum, and most effective network.

Dependency on data set order fed to NN it also to be considered. There are basically two ways to train a neural network. The first is to update the weights after calculating the error for each exemplar. The second is to accumulate the weight updates as each exemplar in the epoch is presented. At the end of the epoch (presentation of each training exemplar one time), the weight updates are made. For epoch training, the order of the presentation of exemplars will make no difference. Exemplar training can result in different results, depending on the order of the exemplar presentation. [36] Optima case would therefore consist of complete independency on order, in which the data are fed to network during training.

Specifically for image processing networks, we might focus on the final **quality range** of our system. In some cases, the trained model might not be able to achieve best possible quality in relation to target compression rate for example. Let's describe it on specific example - Agustsson et al. (2018) [1] presented GAN compression system, with main focus on low quality images, i.e. giving best possible quality outcome at extremely high compression rate. This implies that you might not get satisfying results for higher quality compression in comparison to other systems (be it ML-based, or conventional). [13]

Somewhat similar to quality range, we might discuss **flexibility of bitrate control**. Our network might be designed in a way, that doesn't allow variability and works with fixed bitrate model(s). On the contrary [51][50] both presents *single* model, which can specify target bitrate by so called lambda parameter. Once again situational criterion, that might have a trade-off of being flexible at the cost of achievable quality. [13]

Lastly **coding unit size** gives defines the accessibility to spatial context. With bigger size of coding unit, NN give us better results; on the other hand, we must consider the representation of input data (image resolution) - since architecture is usually tailored for a specific coding unit size, which means that images that have a resolution that is not multiple of this size need to be adapted.

Chapter 3

Practical part

Practical part should focus on comparison of implementations of various ANN systems, for image compression. How the score amongst each other and against such common compression methods such as JPEG and JPEG2000. Further we would like to summarize proposed classification, or rather set of criteria to be considered for classification of learning based systems.

As was described in section 2.3.2, main metric for objective quality assessment was chosen MS-SSIM, implemented in MATLAB:

```
score = multissim(I,Iref)
```

where `I` represents the tested image (compressed one) and `Iref` referential one (raw, uncompressed). Further we will also use PSNR for comparison how purely objective quality metric (PSNR) corresponds with MS-SSIM.

```
peaksnr = psnr(A,ref)
```

Johannes Ballé et al. (2018 and further) created a repository and GitHub containing (amongst others) models from proposals of Minnen [33][32] as well as their own [4], using Python and package Tensorflow developed by Google. I've been using platform called Anaconda, containing distributions of Python and R programming languages and many other useful packages for data science (e.g. SciPy, NumPy, Pandas and many more).

```
conda create --name ENV_NAME python=3.6 cudatoolkit=10.0 cudnn
conda activate ENV_NAME
pip install tensorflow-gpu==1.15 tensorflow-compression==1.3
```

Once environment was prepared and initialized, we gained access to pretrained models from papers mention before. To compress the image, we what have define the file, expected in PNG format (convenient for our use of Kodak testing set [26]), and model we would like to use, such as `mbt2018-mean-msssim-[1-8]`, last digit defining the target quality of compression (1 for lowest, 8 for highest) and optimised for MS-SSIM metric. Optionally we could provide the expected output file name.

```
python tfci.py compress <model> <PNG file>
```

Compressed image would be contained in file with `.tfci` extension. Now to get the decompressed, reconstructed image:

```
python tfci.py decompress <TFCI file>
```

from which we would receive the reconstructed image with target quality we defined before compression. Thus providing us with data for our quality measurement and comparison.

In case of models by Toderici et al. [50], variable bitrate control is present, and we are able to define through lambda parameter to specify what is or targeted bitrate (bpp).

```
python encoder.py --input_image=/your/image/here.png
--output_codes=output_codes.npz --iteration=15
--model=/path/to/model/residual_gru.pb
```

Mentioned lambda parameter is here represented by value `-iteration` within range of 0 - 15; where 0 corresponds to a target of 1/8 bpp and every increment results in an additional 1/8 bpp. Same as before, supplying `.npz` file we are able to decompress the data again:

```
python decoder.py --input_codes=codes.npz
--output_directory=/tmp/decoded/ --model=residual_gru.pb
```

Figure 3.1: Comparison of (left) conventional compression method JPEG2000 (right) ML-based compression system [50] with identical target compress ratio of 192 : 1 (0.125 bpp)

Once we have all the reconstructed images, we could evaluated the output quality based on the metrics defined before (MS-SSIM and PSNR). Note that all result values are averaged over constant quality parameters of given model (constant QP in case JPEG/JPEG2000 and constant lambda value for ML-based methods).

Conventional methods JPEG and JPEG 2000 used for compression in this work were also implemented through MATLAB environment, that is through `imwrite()` method, where we are able to specify target compression method as well as the quality expected; this is shown in code further below, specifically for JPEG compression, where all 24 images from Kodak data set

are compressed with target quality defined by parameter `q` and immediately evaluated, filling out matrix with results.

```
for i=1:24
ref_file = %referential, uncompressed file
A = imread(ref_file);

for q=1:100
    test_file = %output file name after compression
    imwrite(A, test_file, 'jpg', 'quality', q); %
    B=imread(test_file);

    S=imfinfo(test_file);
    w=getfield(S, 'Width');
    h=getfield(S, 'Height');
    size=getfield(S, 'FileSize')*8;


    ms=0;
    for k=1:3
        ms=ms+multissim(B(:,:,k),A(:,:,k));
    end
    MS_SSIM(q,i)=ms/3; %MS-SSIM evalutation, avarage for each layer
    peaksnr(q,i)=psnr(B,A); %PSNR evaluation
    bpp(q,i)=size/(h*w); %target bpp calculation
end
```

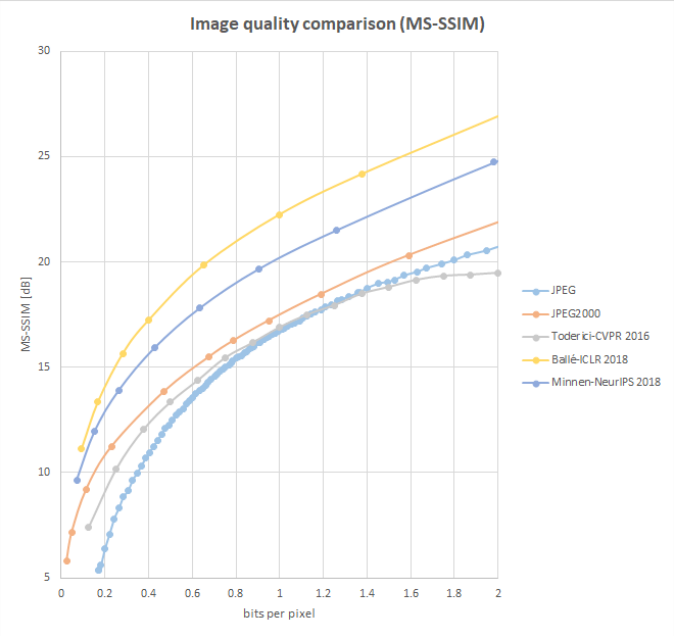


Figure 3.2: Compression method comparison, MS-SSIM metric

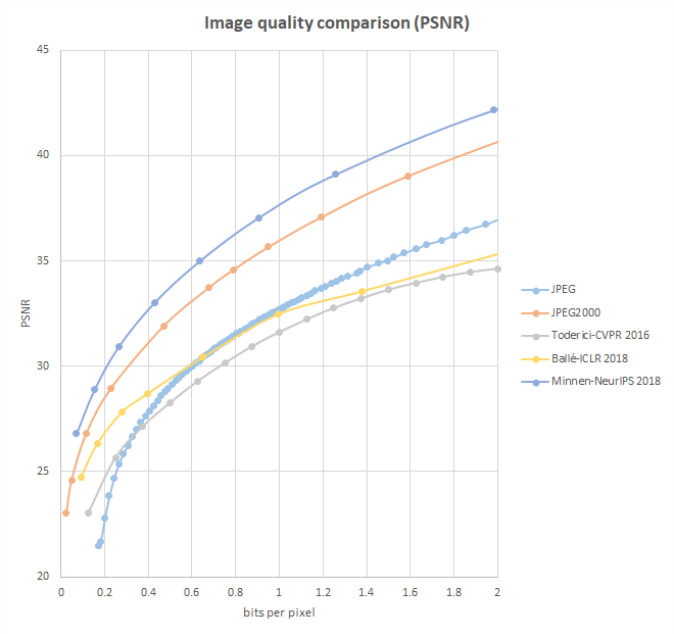


Figure 3.3: Compression method comparison, PSNR metric

Chapter 4

Conclusion

ML based compression methods prove to have their use in practice. As could be evidently seen from comparison results in practical part 3, it is very promising field, with ability to already qualitatively exceed the conventional methods of image processing. As is shown in 3.2, main quality metric chosen for comparison (MS-SSIM) gives us very good quality performance for basically most of the target quality range (up to 2 bpp, although suggests continuing in similar manner). As to the results given by simple PSNR metric, they show comparable results; though in case of JPEG2000 in overcomes most of the methods, scoring second best. This indicates discussed strength of MS-SSIM metric, that better correlates with a subjective quality assessment, that we can experimentally do with naked eye.

This is strictly speaking from performance point of view. The conventional methods are also characterized by their easy of use, after the years of research and development. They are flexible as to the format of the input, quality range provided and implementations in most of the modern systems. Same can't be told for ML-based systems; as for now at least. NNs for image compression are often tailor-made, constricted by definition in advance. Another thing to consider is possibility of distribution, and hardware requirements on the consumer/user side. Last but not least, the learning/training of new models is trivial matter still, and appears to be quite time-consuming, without proper high-end hardware (GPU).

As for the classification (taxonomy) of ML-based system, there's plethora of criteria to look for, extensively described in 2.4. Also many of these criteria are connected and interlaced, sometimes forcing us to make trade-offs, e.g. efficiency vs. flexibility etc.. Also not every criterion is applicable to all problem areas; this implies a lot of decision and criteria application has to be done in advance with fore knowledge. This implies necessity for deeper understanding of

This paper tries to navigate the ever-growing field ANNs and their application in image compression; that is specifically in comparison to conventional compression methods. Practical part there is focuses on this fact is holding possibilities for further improvement, and would be worth of more extensive research. Specifically considering the platforms used for NN implementations - this paper solely focused on methods based on Python language and on Ten-

sorFlow platform. Other (still Python-based) platform are available (such as) and would be worth of deeper research, such as Theano, PyTorch, OpenCV or Keras; they might provide valuable data to be compared with those provided in this paper. Same could be applied for conventional methods - this paper is covering JPEG and JPEG2000, but some other methods - developed more recently - could be presented in future. From my research this could include formats such as BPG (presented by Bellard in 2014) or WebP (by Google), which might prove to be more efficient, since they are designed to be more efficient, and to surpass formats like JPEG.

Appendix A

List of electronic attachments

- evaluation.m: evaluation MATLAB script for JPEG and JPEG2000 compression, with resulting MS-SSIM, PSNR, bpp
- Ballé/tfci.py: python script implementing various compression methods, downloading related models (availability changes during time); by Ballé et al.
- Toderici/decoder.py: short python script for decoding image file by Toderici et al. (2016)
- Toderici/encoder.py: short python script for encoding image file by Toderici et al. (2016)

Bibliography

[1] Eirikur Agustsson et al. *Generative Adversarial Networks for Extreme Learned Image Compression*. 2019. arXiv: 1804.02958 [cs.CV].

[2] N. Asuni and A. Giachetti. "TESTIMAGES: A large-scale archive for testing visual devices and basic image processing algorithms (SAMPLING 1200 RGB set)". In: *STAG: Smart Tools and Apps for Graphics*. 2014. URL: https://sourceforge.net/projects/testimages/files/OLD/OLD_SAMPLING/testimages.zip.

[3] Alejandro Baldominos, Yago Saez, and Pedro Isasi. "A Survey of Handwritten Character Recognition with MNIST and EMNIST". In: *Applied Sciences* 9.15 (Aug. 2019), p. 3169. ISSN: 2076-3417. DOI: 10.3390/app9153169. URL: http://dx.doi.org/10.3390/app9153169.

[4] Johannes Ballé et al. *Variational image compression with a scale hyperprior*. 2018. arXiv: 1802.01436 [eess.IV].

[5] H. Bourlard and Y. Kamp. "Auto-association by multilayer perceptrons and singular value decomposition". In: *Biological Cybernetics* 59 (2004), pp. 291–294.

[6] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.

[7] Yung-Yao Chen et al. "Design and Implementation of Cloud Analytics-Assisted Smart Power Meters Considering Advanced Artificial Intelligence as Edge Analytics in Demand-Side Management for Smart Homes". In: *Sensors* 19 (May 2019), p. 2047. DOI: 10.3390/s19092047.

[8] *Common ML Problems | Introduction to Machine Learning Problem Framing*. URL: https://developers.google.com/machine-learning/problem-framing/cases.

[9] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine learning* 20.3 (1995), pp. 273–297.

[10] Thomas Cover and Peter Hart. "Nearest neighbor pattern classification". In: *IEEE transactions on information theory* 13.1 (1967), pp. 21–27.

[11] Peter Dayan et al. "The Helmholtz Machine". In: *Neural Comput.* 7.5 (Sept. 1995), pp. 889–904. ISSN: 0899-7667. DOI: 10.1162/neco.1995.7.5.889. URL: https://doi.org/10.1162/neco.1995.7.5.889.

[12] DeepAI. *Radial Basis Functions*. May 2019. URL: https://deepai.org/machine-learning-glossary-and-terms/radial-basis-function.

[13] Source Diego et al. "Coding of Still Pictures". In: (Sept. 2001).

[14] Jeffrey L. Elman. "Finding structure in time". In: *COGNITIVE SCIENCE* 14.2 (1990), pp. 179–211.

[15] D. Ghadiyaram and A. C. Bovik. "Massive Online Crowdsourced Study of Subjective and Objective Picture Quality". In: *IEEE Transactions on Image Processing* 25.1 (2016), pp. 372–387. DOI: 10.1109/TIP.2015.2500021.

[16] Arcadi Gonzalez. *IMAGE COMPRESSION USING MACHINE LEARNING TECHNIQUES*. Oct. 2015. DOI: 10.13140/RG.2.1.1140.7121.

[17] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[19] John A Hartigan and Manchek A Wong. "Algorithm AS 136: A k-means clustering algorithm". In: *Journal of the royal statistical society. series c (applied statistics)* 28.1 (1979), pp. 100–108.

[20] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[21] Lina Jin et al. "Statistical Study on Perceived JPEG Image Quality via MCL-JCI Dataset Construction and Analysis". In: *Electronic Imaging* 2016 (Feb. 2016), pp. 1–9. DOI: 10.2352/ISSN.2470-1173.2016.13.IQSP-222.

[22] M I Jordan. "Serial order: a parallel distributed processing approach. Technical report, June 1985-March 1986". In: (May 1986). URL: https://www.osti.gov/biblio/6910294.

[23] Andrej Karpathy. *Generative Models*. Sept. 2020. URL: https://openai.com/blog/generative-models/.

[24] Diederik P. Kingma and Max Welling. "An Introduction to Variational Autoencoders". In: *Foundations and Trends® in Machine Learning* 12.4 (2019), pp. 307–392. ISSN: 1935-8245. DOI: 10.1561/2200000056. URL: http://dx.doi.org/10.1561/2200000056.

[25] W. Kinsner. "Compression and Its Metrics for Multimedia". In: *Proceedings of the 1st IEEE International Conference on Cognitive Informatics*. ICCI '02. USA: IEEE Computer Society, 2002, pp. 107–121. ISBN: 0769517242.

[26] Eastman Kodak. *Kodak Lossless True Color Image Suite (PhotoCD PCD0992)*. URL: http://r0k.us/graphics/kodak.

[27] David Kriesel. *A Brief Introduction to Neural Networks*. 2007. URL: http://www.dkriesel.com.

[28] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. *THE MNIST DATABASE*. URL: http://yann.lecun.com/exdb/mnist/.

[29] Yann Lecun et al. "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE* 86 (Dec. 1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[30] *Machine Learning Glossary | Google Developers*. URL: https://developers.google.com/machine-learning/glossary.

[31] Donald Michie. ""Memo" functions and machine learning". In: *Nature* 218.5136 (1968), pp. 19–22.

[32] David Minnen, Johannes Ballé, and George Toderici. *Joint Autoregressive and Hierarchical Priors for Learned Image Compression*. 2018. arXiv: 1809.02736 [cs.CV].

[33] David Minnen et al. *Image-Dependent Local Entropy Models for Learned Image Compression*. 2018. arXiv: 1805.12295 [cs.CV].

[34] Tom M. Mitchell. *Machine Learning*. New York: McGraw-Hill, 1997. ISBN: 978-0-07-042807-2.

[35] Radford M. Neal and Peter Dayan. "Factor Analysis Using Delta-Rule Wake-Sleep Learning". In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1781–1803. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1781. URL: https://doi.org/10.1162/neco.1997.9.8.1781.

[36] D. E. Nelson and S. K. Rogers. "A taxonomy of neural network optimality". In: *Proceedings of the IEEE 1992 National Aerospace and Electronics Conference@m_NAECON 1992*. 1992, 894–899 vol.3.

[37] M.A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. URL: http://neuralnetworksanddeeplearning.com/.

[38] Jo Plumridge. *Are You Losing Too Much of Your Photograph to Compression?* July 2020. URL: https://www.lifewire.com/the-effect-of-compression-on-photographs-493726.

[39] David Poole, Alan Mackworth, and Randy Goebel. "Computational Intelligence". In: (1998).

[40] Bin Qian et al. *Orchestrating the Development Lifecycle of Machine Learning-Based IoT Applications: A Taxonomy and Survey*. 2019. arXiv: 1910.05433 [cs.DC].

[41] J. Ross Quinlan. "Induction of decision trees". In: *Machine learning* 1.1 (1986), pp. 81–106.

[42] Marc'Aurelio Ranzato et al. "Efficient Learning of Sparse Representations with an Energy-Based Model". In: Jan. 2006.

[43] Jürgen Schmidhuber. "Deep learning in neural networks: An overview". In: *Neural Networks* 61 (Jan. 2015), pp. 85–117. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2014.09.003. URL: http://dx.doi.org/10.1016/j.neunet.2014.09.003.

[44] Mike Schuster and Kuldip Paliwal. "Bidirectional recurrent neural networks". In: *Signal Processing, IEEE Transactions on* 45 (Dec. 1997), pp. 2673–2681. DOI: 10.1109/78.650093.

[45] Jonathon Shlens. "A tutorial on principal component analysis". In: *arXiv preprint arXiv:1404.1100* (2014).

[46] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*. Vol. 135. MIT press Cambridge, 1998.

[47] Andrew Tch. *DIY AI: An old school matrix NN*. July 2017. URL: https://towardsdatascience.com/diy-ai-an-old-school-matrix-nn-401a00021a55.

[48] Andrew Tch. *The mostly complete chart of Neural Networks, explained*. Aug. 2017. URL: https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464.

[49] Movavi Blog Team. *What is a Video Codec?: A Useful Lesson from Movavi*. Mar. 2020. URL: https://movavi.io/codec-2/.

[50] George Toderici et al. *Full Resolution Image Compression with Recurrent Neural Networks*. 2016. arXiv: 1608.05148 [cs.CV].

[51] George Toderici et al. *Variable Rate Image Compression with Recurrent Neural Networks*. 2015. arXiv: 1511.06085 [cs.CV].

[52] M.V. Valueva et al. "Application of the residue number system to reduce hardware costs of the convolutional neural network implementation". In: *Mathematics and Computers in Simulation* 177 (2020), pp. 232–243. ISSN: 0378-4754. DOI: https://doi.org/10.1016/j.matcom.2020.04.031. URL: http://www.sciencedirect.com/science/article/pii/S0378475420301580.

[53] Fjodor van Veen. *The Neural Network Zoo*. Apr. 2019. URL: https://www.asimovinstitute.org/neural-network-zoo/.

[54] Pascal Vincent et al. "Extracting and Composing Robust Features with Denoising Autoencoders". In: ICML '08. Helsinki, Finland: Association for Computing Machinery, 2008, pp. 1096–1103. ISBN: 9781605582054. DOI: 10.1145/1390156.1390294. URL: https://doi.org/10.1145/1390156.1390294.

[55] Z. Wang, E. P. Simoncelli, and A. C. Bovik. "Multiscale structural similarity for image quality assessment". In: *The Thrity-Seventh Asilomar Conference on Signals, Systems Computers, 2003*. Vol. 2. 2003, 1398–1402 Vol.2. DOI: 10.1109/ACSSC.2003.1292216.

[56] *What is Machine Learning? A definition*. URL: https://expertsystem.com/machine-learning-definition/.

[57] Bill Wilson. URL: http://www.cse.unsw.edu.au/~billw/dictionaries/mldict.html.

[58] Zhou Wang and A. C. Bovik. "A universal image quality index". In: *IEEE Signal Processing Letters* 9.3 (2002), pp. 81–84. DOI: 10.1109/97.995823.

[59] Zhou Wang et al. “Image quality assessment: from error visibility to structural similarity”. In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612. DOI: 10.1109/TIP.2003.819861.