Tampere University

Jukka Ilari Ahonen

IMAGE CODING FOR MACHINES

Deep learning based post-processing filters

Master's Thesis
Faculty of Information Technology and Communication Sciences
Examiner: Prof. Esa Rahtu
November 2021

ABSTRACT

Jukka Ilari Ahonen: Image coding for machines
Master's Thesis
Tampere University
Master's Programme in Information Technology
November 2021

Machine vision tasks such as object detection and instance segmentation are becoming more and more popular these days due to the quickly increasing performance of deep neural networks. Consequently, more and more multimedia content such as images will presumably be consumed by machines in the years to come. Since the images coded with state-of-the-art traditional codecs such as Versatile Video Coding (VVC) are designed to maximize the subjective quality perceived by humans, they may not be optimal for machine consumption. Hence, to address this, new codecs designed solely for machines are needed.

There are roughly three different directions on machine-oriented compression methods explored in the literature. First type of methods are based on adapting the existing traditional codecs, for example, by changing their parameters, while the second type of methods are based fully on End-to-End (E2E) learned neural networks. The third type of methods are hybrids, which combine the traditional codecs with learned approaches. The hybrid methods are usually performance-wise superior to the solely traditional based methods, while they also have intriguing properties, which E2E based methods might lack. These include certain benefits such as real-time decoding, hardware implementation availability and interoperability.

In this regard, this thesis introduces a hybrid system to train post-processing filters that aim to enhance the performance of the VVC reconstructed images on different machine tasks. One of these filters called Task Specific Enhancement (TSE) filter achieves 45% and 49% Bjøntegaard Delta Rate (BD-rate) gains over plain VVC on instance segmentation and object detection tasks, validated on a subset of Open Images validation dataset with Mask R-CNN and Faster R-CNN based models, respectively. Moreover, another filter called Task Agnostic Enhancement (TAE) filter also achieves over 40% BD-rate gain when validated similarly. It also generalizes well, preserving a high performance even when the validation dataset and the model are changed.

Keywords: compression, image compression, image coding for machines, learned compression, neural networks, autoencoder, VVC

TIIVISTELMÄ

Jukka Ilari Ahonen: Image coding for machines
Diplomityö
Tampereen yliopisto
Tietotekniikan DI-ohjelma
Marraskuu 2021

Koneiden välinen kommunikointi ja erityisesti niiden käyttökohteet, kuten kohteen tunnistaminen ja segmentointi (engl. object detection ja instance segmentation) ovat jatkuvassa kasvussa ja niiden voidaan sanoa olevan nykyään jo melko arkipäiväistä. Tämä johtaa suurella todennäköisyydellä siihen, että tulevaisuudessa suurin osa multimedialiikenteestä on koneiden välistä. Koneiden väliselle dataliikenteelle tarvitaan siis täysin omanlaisensa kompressiokoodekit, sillä jo olemassa olevat koodekit on tehty ihmisiä varten, eivätkä ne sen takia ole optimaalisimmillaan konekäytössä.

Kirjallisuudessa esitetyt tekniikat voidaan jakaa karkeasti kolmeen eri ryhmään. Ensimmäisessä ryhmässä ovat metodit, jotka perustuvat jo olemassa olevien koodekkien, kuten Versatile Video Coding (VVC) muokkaamiseen koneille sopivammiksi esimerkiksi parametreja muuttamalla (engl. traditional based methods). Toiseen ryhmään kuuluvat täysin neuroverkkopohjaiset koodekit (engl. End-to-End learned methods) ja kolmannessa ryhmässä ovat ns. hybridijärjestelmät (engl. hybrid methods), joissa perinteiseen koodekkiin yhdistetään neuroverkkopohjaisia tekniikoita. Erityisesti hybridijärjestelmien etuna on niiden nopeus yhdistettynä korkeaan suorituskykyyn sekä mahdollisuus hyödyntää jo olemassa olevia laitteistotason toteutuksia.

Tässä diplomityössä esitellään hybridijärjestelmä, joka perustuu Versatile Video Codingilla pakattujen kuvien parantamiseen autoenkooderipohjaisella (engl. autoencoder) ratkaisulla. Versatile Video Codingiin verrattuna, hybridijärjestelmällä valmistettu Task-Specific Enhancement (TSE) -suodatin parantaa Open Images tietoaineistossa (engl. dataset) kohteen tunnistamistehtäviä 45 prosenttia Bjøntegaard Delta -asteikolla (BD-rate) ja kohteen segmentointitehtäviä 49 prosenttia samalla asteikolla mitattuna. Task-Agnostic Enhancement (TAE) -suodatin saavuttaa keskimäärin yli 40 prosentin Bjøntegaard Delta -asteikon parannuksen VVC-koodekkiin nähden, kun se evaluoidaan samalla tavalla. TAE-suodatin säilyttää korkean suorituskykynsä, vaikka evaluoinnissa käytetty tietojoukko tai neuroverkko vaihdettaisiin täysin erilaisiksi.

Avainsanat: kompressio, pakkaus, neuroverkot, koneoppiminen, koodekki

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Today, most of the data moved around the Internet consist of images and videos. Because of the fast development in domains of artificial intelligence and machine vision in general, most of this data will probably be consumed by machines in a couple of years. Cisco Annual Internet Report [1] gives an estimate that by the year 2023, half of the internet traffic will be between machines. In this regard, there is a demand for better compression codecs, which can handle the increasing bandwidth. Moreover, the existing traditional state-of-the-art codecs for video coding such as the High Efficiency Video Coding (HEVC) [2] and the Versatile Video Coding (VVC) [3] are developed for human consumption. This means that they emphasize the visual quality of the decoded data and thus the performance may not be the best for machine vision tasks. Codec for machine consumption is currently a highly researched area. For example, JPEG-AI, which is a subgroup of Joint Photographic Experts Group (JPEG) [4] has started a standardization process for Image Compression technologies for Machines (ICM). Similarly, Video Coding for Machines (VCM), which is a Ad-hoc group of Moving Picture Experts Group (MPEG) [5] has done the same, but for video compression technologies.

The main contribution of this thesis is to introduce a hybrid system, which is based on learned post-processing filters. Some of these results are also published on a research paper [6]. This will be done in the implementation part of the thesis. Note that the terms post-processing filter, post filter and enhancement filter mean the same thing in the scope of this thesis and will be referred interchangeably for now on. However, in order to understand the subjects discussed, this thesis starts by introducing the basics of compression in chapter 2. Some methods are discussed from the domains of lossless and lossy compression with heavy weight on the image compression point of view. More importantly, a state-of-the-art traditional codec called Versatile Video Coding (VVC) is introduced, since it is the codec used to encode and decode the images that the post-processing filters trained in the implementation part are actually filtering. Moreover, the VVC also works as a benchmark for the filters. Next, some basics about the deep neural networks (DNN) and their sub-types such as convolutional neural networks (CNN) and autoencoders (AE) are discussed in chapter 3. This is due to the fact that all the trained enhancement filters have a structure of a convolutional autoencoder. After this, image compression for machines is further discussed in chapter 4. It includes a introduction to machine tasks

such as object detection and instance segmentation, along with some example neural networks built specifically for these tasks and also used in the implementation part in chapter 5. Also, some common machine task metrics are introduced. Finally, three of the most common types of methods/systems in ICM are introduced on a high level along with existing technologies based on each method. This concludes the theory part. The implementation part in chapter 5 introduces a hybrid image compression system for machines. With this system, three types of enhancement filters are trained: Baseline Fidelity Enhancement (BFE) filter, Task Specific Enhancement (TSE) filter and Task Agnostic Enhancement (TAE) filter. These enhancement or post-processing filters are used to filter the VVC reconstructed images in order to boost the performance on machine tasks. The performance of all the filters are evaluated on object detection and instance segmentation tasks and compared to the plain VVC. Finally, conclusion is given and further work discussed in chapter 6.

Essentially, this thesis aims to answer for the need of a new image compression codec aimed for machines by introducing a hybrid compression codec, which utilizes deep learning based post-processing filters. Here are listed the most important questions that arise considering the post-processing filters:

Task performance and generalization – How well the post-processing filters enhance the machine task accuracy of the images reconstructed by the state-of-the-art traditional codec VVC? Furthermore, what affects to the generalization of the post-processing filters for different task networks?

Filtering speed – What is the filtering speed of the filters? This is important, since the filtering happens completely on the decoder side. Thus, the system needs to be relatively fast.

Visual impact – What is the visual impact of filtering images with a post-processing filter? Most importantly, are those filtered images still acceptable for human consumption? If there is no big degradation in the visual quality, the resulting images could be consumed also by humans.

2. TRADITIONAL COMPRESSION

Before talking about the image compression for machines, one must first know the basic concepts of traditional compression. "Traditional" in this context means that no learned methods are used and the compression is aimed for human consumption. This chapter goes through very briefly about what is meant by compression, the concept of information entropy, lossless compression and lossy compression. Also, a state-of-the-art video compression codec called Versatile Video Coding (VVC) is introduced on a high level, since it will also be used as the benchmark codec on the implementation part of the thesis. Moreover, the images being post filtered by the implemented enhancement filters will be first encoded and decoded by the VVC. Other compression codecs, albeit important, are not discussed thoroughly, other than briefly mentioned.

Compression in the information theory means that the data is encoded by a fewer amount of bits than it has in the original format. This is especially important considering video and image data, because they already consume the most of the internet bandwidth and the trend is predicted to only increase in the future [1].

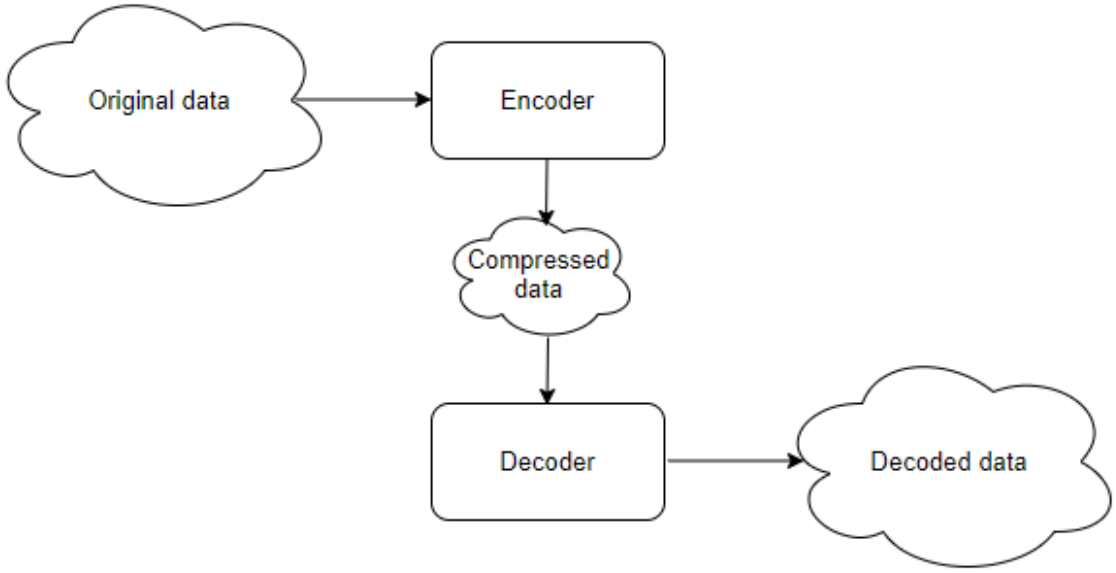


Figure 2.1. *Basic idea of compression*

The basic idea of compression can be seen from Figure 2.1, where the original data, which can be e.g. an image, is first compressed by an encoder into a bitstream. This

bitstream or compressed data takes up a smaller space than it did in the original format. After this, a decoder decompresses the bitstream back into the original form. A system, which uses this kind of encoder decoder structure is called a codec. In image compression, a metric often used to describe the amount of compression, is called Bits Per Pixel (BPP). It measures the amount of bits on the compressed bitstream needed to code each pixel of the original image.

Information entropy, first introduced by Claude Shannon in [7] for discrete probabilities is defined as

$$H = -\sum P_i \log P_i, \tag{2.1}$$

where the P_i denotes the outcome of some discrete event and the base of the logarithm determines the output unit, e.g. base 2 yielding bits and base e yielding nats. Thus, the information entropy describes how predictable the data is. Moreover, if considering the encoded data as a string of symbols, the formula can be used to estimate the average number of units needed to code the data losslessly, based on the frequency of the symbols in the string. This type of coding, which uses the entropy information along with other statistics of the data to code it is called entropy coding.

2.1 Lossless compression

In lossless compression, no information is lost when encoding and decoding the data. This means that with optimal form of lossless compression the bits used to encode a symbol should approach the number of bits pointed by the entropy formula (2.1). Techniques where the number of bits to code a different symbols varies are called variable-length coding. Maybe the most popular techniques of variable-length coding/entropy coding being Huffmann coding and arithmetic coding [8], [9]. There exists also many lossless codecs for images such as: PNG, GIF and JPEG-LS, see [10, pp. 301–310, 371–390] for further information. As one might guess, the lossless compression is used in the areas where the data being losslessly stored is very important. These include e.g. medical imaging, text compression and remote sensing. [10, p. 207] However, this thesis is more interested in lossy compression, which is the method often used when compressing images for machines.

2.2 Lossy compression

Lossy compression on the other hand loses information on the compression, meaning the decompressed data isn't exactly the original anymore. Therefore, it is used in applications where the loss of the information is not so critical, such as storing common images, videos or sound. [10, pp. 432, 440] The main reason lossy compression is used, is that it achieves much higher compression ratio than its lossless counterpart. Compression ratio

in this context means the ratio of the size of the uncompressed data and compressed data. One of the most common methods in the lossy compression is called transform coding, where the data is first transformed into some other representation, where it is then quantized and further processed to achieve more efficient compression with lower amount of loss as in the original domain. Quantization in this context means that the sample values are mapped from one range to another with less precision. Popular transform methods include a discrete cosine transform (DCT) [11] and a discrete wavelet transform (DWT) [12]. DCT, being more popular option of the two, is used for example in JPEG, AVC, HEVC and VVC [2], [3], [13], while the most famous DWT-based codec is JPEG-2000 [14]. Furthermore, all of these codecs also utilize a form of lossless entropy encoding on their pipeline, which is generally very common in a lossy compression pipeline. Example of this can be seen from Fig. 2.2, where after some transformation and quantization, the uncompressed image x_{gt} is losslessly entropy encoded into its final form of bitstream and the decoding part is just the inverse of the encoding procedures to get the reconstructed image x.

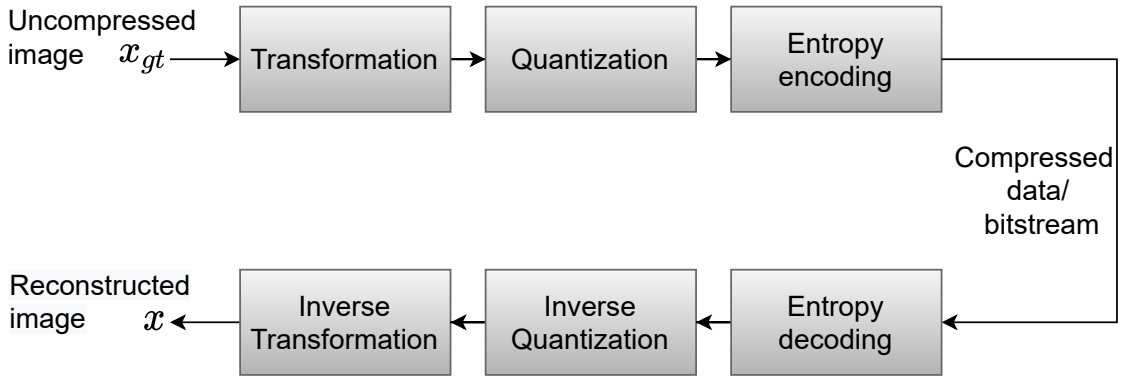


Figure 2.2. *Common lossy compression pipeline*

2.2.1 Versatile Video Coding

At the moment when this thesis is written, Versatile Video Coding (VVC) [3] also known as H.266 is the state-of-the art traditional video codec. VVC has both lossless and lossy functionalities, but in the scope of this thesis, it is used only as a lossy image codec. VVC is the successor of the High Efficiency Video Coding (HEVC) [2] aka. H.265, reducing the bitrate over it roughly 50 %. HEVC, on the other hand, is a successor of a well known video codec Advanced Video Coding (AVC) [13] aka. H.264, between those two there is also a 50 % bitrate improvement. Furthermore, VVC is also able to handle more complex formats than AVC or HEVC, such as 16K video. The only disadvantage VVC has over its predecessors is the coding time, since it is a more complex codec. Next, a rather high level overview is given of the VVC.

Fig. 2.3 shows the main parts of the VVC encoder and decoder. The basic principle of the

VVC on the encoder side is exactly similar as was described on the lossy compression pipeline in Fig. 2.2. It contains transformation with DCT, quantization and entropy encoding. However, the input to the transform is not the actual image but rather the residual, which is the difference between the predicted and uncompressed frame. In the beginning, the input image is divided into blocks, which do not overlap. These blocks are then divided to even smaller blocks, which are called Coding Units (CU). The shape of these coding units are determined by the content, meaning that smaller blocks are chosen for more detailed parts of the image, whereas larger blocks are assigned to the non-detailed parts. These hierarchical partitions are called partition maps and they are predicted from the earlier processed data. If inter-prediction / motion-compensated prediction is used, the partitions are predicted from frames that have already been coded, taking advantage of the temporal redundancies. The other option is to use intra-prediction, meaning that the partitions are predicted only from the parts of the current frame that have already been encoded. There exist an "all-intra" configuration, which causes the encoder to code all the images independently. Since the tests in this thesis are later on conducted for images rather than video, the all-intra configuration must be chosen for the VVC to act as an image codec. At the decoder side, after the quantization of the residual coefficients, they will be inverse quantized and inverse transformed to access the reconstructed residual. This reconstructed residual can then be summed together with either the intra or inter predicted block to get the reconstructed block. The reconstructed frame or block is yet filtered with in-loop filters to generate the final reconstruction. The encoder also uses this reconstructed output to estimate the motion between the next input frame, and this motion data is then sent back to the decoder in order to do the inter-prediction. [3], [15]

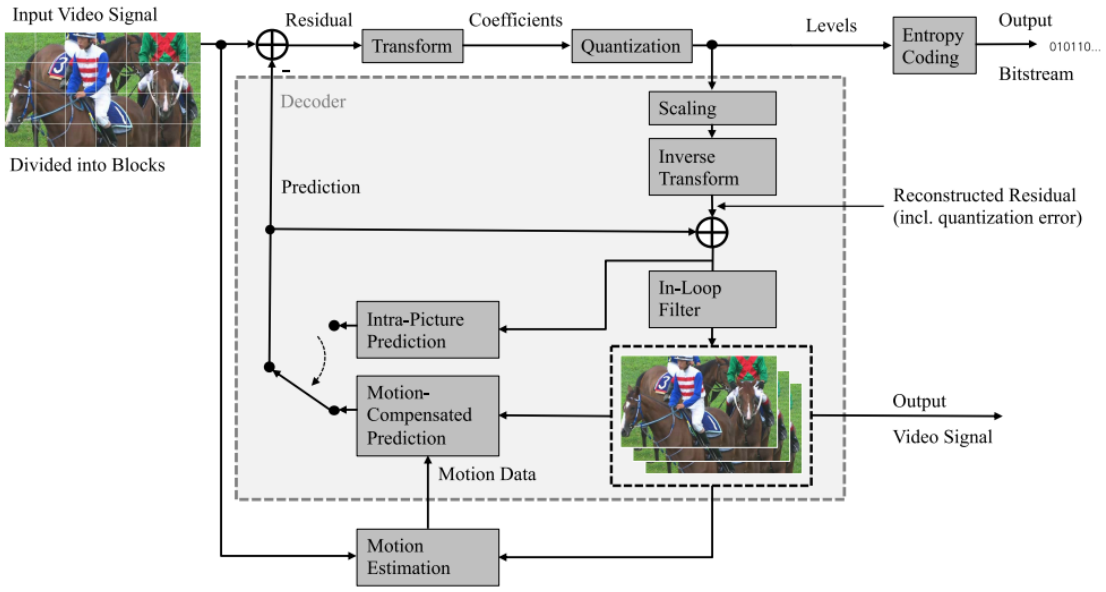


Figure 2.3. *VVC structure on a high level [15]*

3. DEEP NEURAL NETWORKS

Alongside the traditional compression, a deep neural network (DNN) is an important concept in order to understand the image compression for machines better. This is because both the machine tasks and the compression methods of ICM often utilize DNNs. Thus, this chapter introduces the basic principles behind a DNN. Their subtypes known as convolutional neural networks (CNN) and autoencoders (AE) will also be studied further, since the enhancement filters trained in implementation part are all based on a convolutional autoencoder.

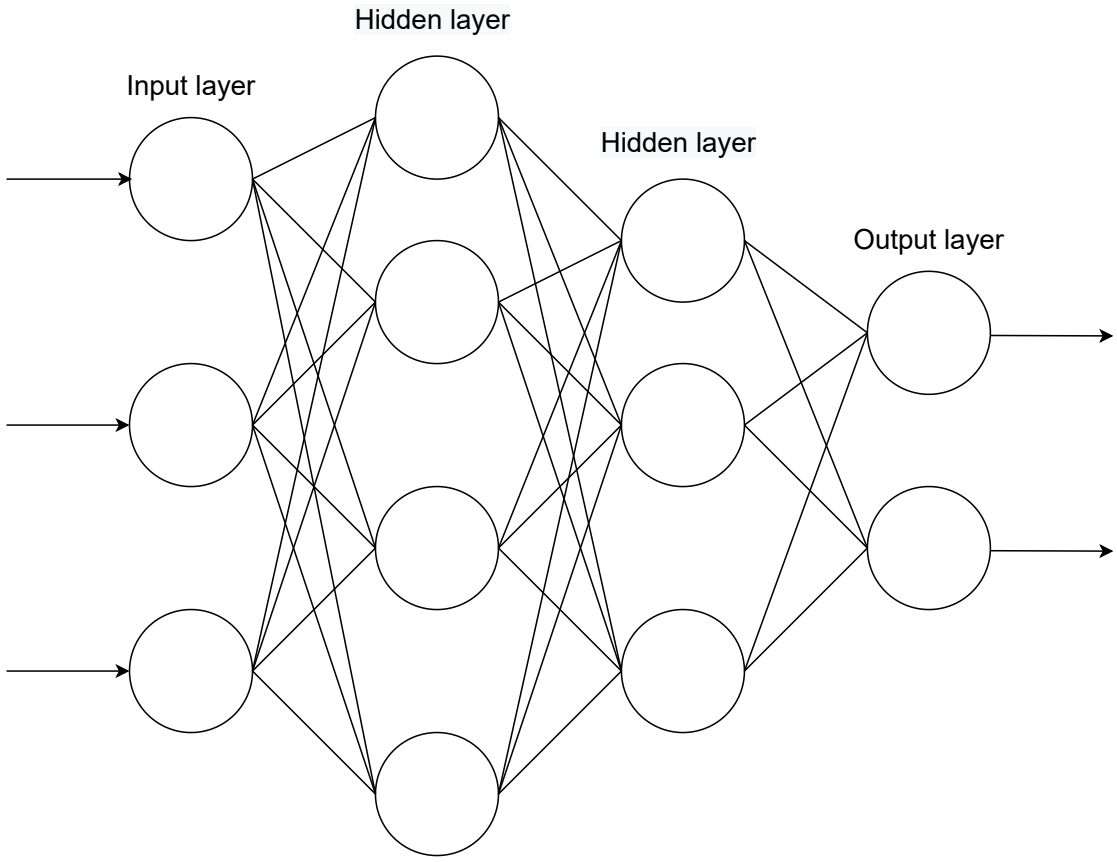


Figure 3.1. *Example of a feedforward fully connected neural network, circles denote neurons [16]*

A neural network is simply a statistical model, which tries to find some relationship between its input and output data. Basic neural networks learn to adaptively adjust their parameters by learning from the examples given to them. As seen from the Figure 3.1, a

feedforward neural network has three basic types of layers, which all consists of one or more base units called neurons. First type of layer is called an input layer, which takes the input data e.g. images as their input. Neurons in the input layer are connected to the neurons in the next layer, which is called a hidden layer. Hidden layer consists of neurons, which are "hidden" inside the network, these can be again connected to a next hidden layer or to an output layer. Output layer is the last layer of the network and it yields the wanted output of the network. The terms deep neural network and feedforward neural network mean that there are several hidden layers in the NN and the information flows to one direction, respectively. In addition to feedforward NN, there exists recurrent neural networks (RNN), where the information can flow to both directions. However, since the RNN-like structures are not utilized in this thesis, they are not discussed further.

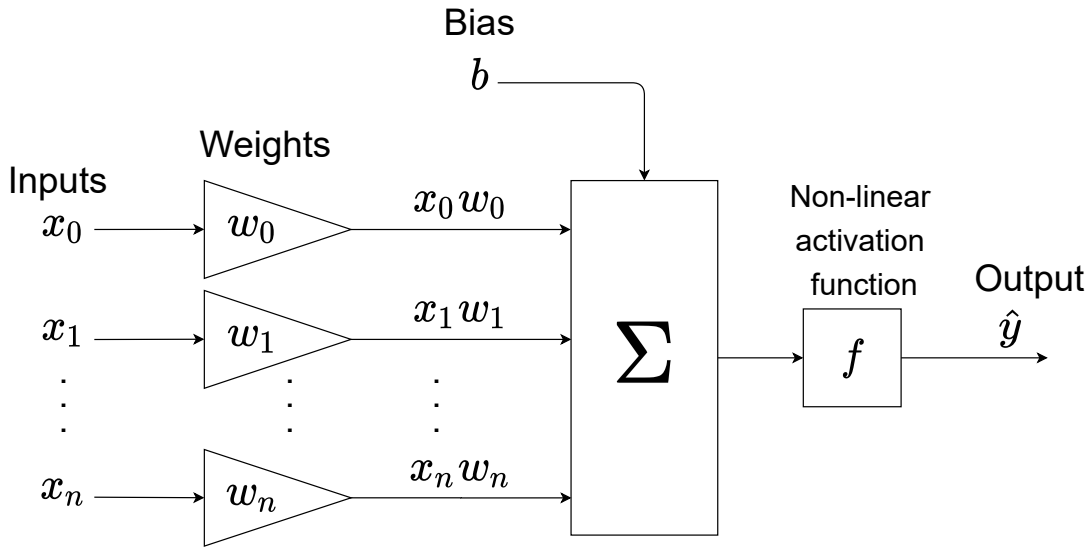


Figure 3.2. *Neuron of a feedforward neural network. Triangle denotes multiplication and Sigma denotes summation.*

Every neuron has the following structure as seen in the Figure 3.2. The inputs $x_0 ... x_n$ can be either the actual input to the network, or a lateral representation of the data, meaning that the input comes from other neuron and is in different form that in the beginning. All the neurons connections to other neurons have also weights $w_0 ... w_n$, which tell how strong the connections are. All the inputs are multiplied by their corresponding weights and summed together with the bias term b. Together the weights and biases are the trainable parameters of the network. Finally, the result of the summation is fed to an activation function, which is some non-linear function, this makes also the output non-linear. One of the most used activation function is called Rectified Linear Unit (ReLU) introduced in [17], which is defined as:

$$f(x) = max(0, x) = \begin{cases} x_i, & \text{if } x_i \geq 0 \\ 0, & \text{if } x_i < 0 \end{cases}, \tag{3.1}$$

where x is the input to the function. The reason for its popularity is the fact that it reduces the vanishing gradients problem, while also having a great performance. There exists multiple variants of the ReLU, see [18] for a comparison. One of these functions is called Parametric Rectified Linear Unit (PReLU) first introduced in [19]. According to the authors, it is superior to ReLU with minimal added computational cost. PReLU is defined as:

$$f(x) = max(0, x) = \begin{cases} x_i, & \text{if } x_i \geq 0 \\ a_i x_i, & \text{if } x_i < 0 \end{cases}, \tag{3.2}$$

where the a_i is a learnable parameter for the negative slope. That is in fact the only difference compared to ReLU as can be seen also from the Figure 3.3

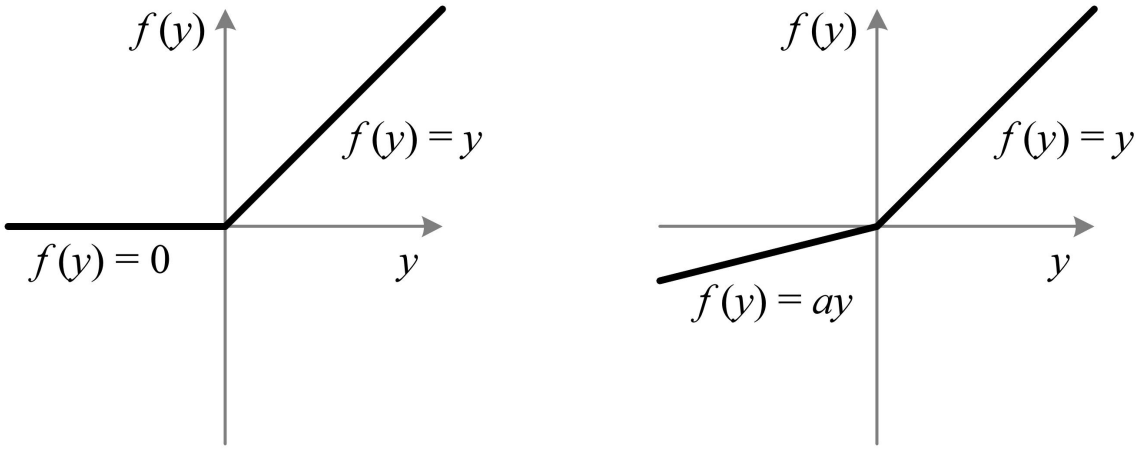


Figure 3.3. *Comparison of ReLu and PReLU [19]*

Once the input data has gone through all neurons and is at the end of the network, a *forward pass* has been executed. After this, a *backward pass* begins. It starts by feeding the network outputs $\hat{y}_i$ to a *cost function*, which compares the outputs to a desired ground truths y_i, which are known. When some known data is used to train the network, the process is called *supervised learning*. On the other hand, when there is no ground truth, the learning is called *unsupervised*, meaning that the system must find the correlations completely by itself. Neural network-wise this thesis focuses on supervised learning for now on. One of the most common cost function is Mean Squared Error, which is defined as:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2, \tag{3.3}$$

where n, $\hat{y}_i$ and y_i denote the number of outputs, outputs of the network and ground truths, respectively. After the cost function has given the difference of the predicted data and the ground truth data, all the weights and biases are updated one by one accordingly, in order to minimize the cost function. The algorithms, which do this minimization are called optimizers. Most common optimizers are gradient based, meaning the gradients

of the loss w.r.t the parameters are used for updating the parameters, i.e. weights and biases. Mathematically the process can be expressed as:

$$p \leftarrow p - \alpha \nabla_p f(p), \tag{3.4}$$

where p are the parameters to be optimized, α is a learning rate (LR), which determines the amount of change at one update and $\nabla_p f(p)$ is the gradients of the loss function w.r.t to the parameters. The learning rate should be chosen such that the jumps towards the optimum of the gradient are not too big or too small. With too large LR, the gradient might always jump over the optimum thus never reaching it. On the other hand, with too small LR the jumps might be so small that the outcome becomes similar or a too local optimum is reached. Thus, the LR must often be tested empirically to find a one that works with the used optimizer, data, network, loss-function etc. It might also be beneficial to use a learning rate scheduler, which changes the LR over time according to some rule, e.g. reducing the LR after a certain amount of iterations over the entire dataset. For more in depth explanation of the mathematics behind the gradient based methods and the back propagation, see chapter 6. from [20], where the whole process is explained also with matrices. This is useful since most common inputs to any NN are tensors, which are multidimensional matrices. Probably the most commonly used gradient based optimizer today is called an Adam optimizer [21], which uses the estimates of the first and second moments of the gradients seen in Eq. 3.4 to calculate an adaptive learning rate for the parameters. Fig. 3.4 shows a comparison of different optimizers, representing the training cost against iterations over entire dataset, also known as epochs . The dataset in that particular experiment is the Modified National Institute of Standards and Technology (MNIST) dataset [22], which contains $60,000$ hand written digits as the training images along with the ground truth annotations. These type of free to use datasets such as MNIST are also a very common sight in the domain of machine learning. The actual model in the experiment is a regular feedforward NN containing two hidden layers with $1,000$ neurons each, followed by ReLU activations. Adam seems to be clearly the fastest one minimizing the training cost in this comparison. While optimizer performance is always specific for the network and task at hand, it can be safely said that currently Adam is a good choice for a generally high performing optimizer [21].

In addition to training the DNNs, their performance must be evaluated with a validation set, which contains unseen data for the model to predict on. This process can be called as evaluation or validation interchangeably. If no evaluation is done, but the trained model is used to predict unseen data, that process is called inference. As stated before, the neural network tries to find some relationship between its input and output data. However, this trained relationship should be general enough to make the model perform well also on the validation data. If the training dataset is a too small representation of the problem, training for too long might cause the model to overfit into the training data, causing the

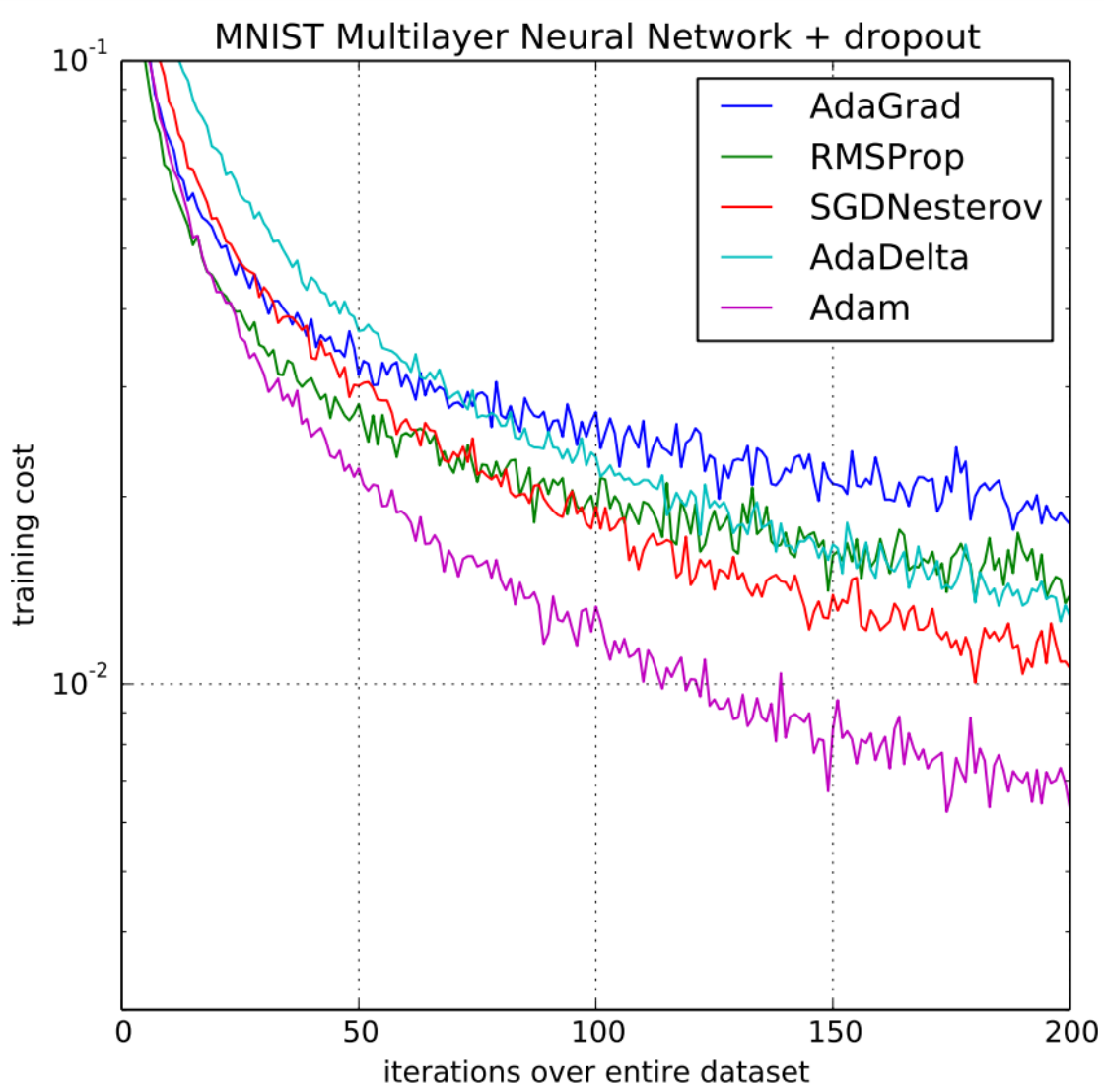


Figure 3.4. *Comparison of the learning speed of Adam optimizer with other optimizers [21]*

performance to drop when predicting on the validation data. Overfitting is generally known to be one of the biggest problems in machine learning and in order to resolve this, many preventative methods have been generated. For example, dropout introduced in [23] simply drops out randomly some percentage of the units between the layers. Regularization such as L1 or L2 adds an extra term to the used loss function to penalize the higher parameter values, essentially making the model more simple and thus more general. Data augmentation on the other hand focuses on slightly modifying the input data in order to generate new artificial training data. In the case of an image data, the augmentation can be for example slightly cropping, flipping or stretching the image [24]. Even with using these preventative methods, the model might still overfit. Thus, it is important to validate the model often enough to find the best fit as is represented in Fig. 3.5. On the other hand, if the training is stopped too early, the model is in the regime of underfitting, which is also suboptimal.

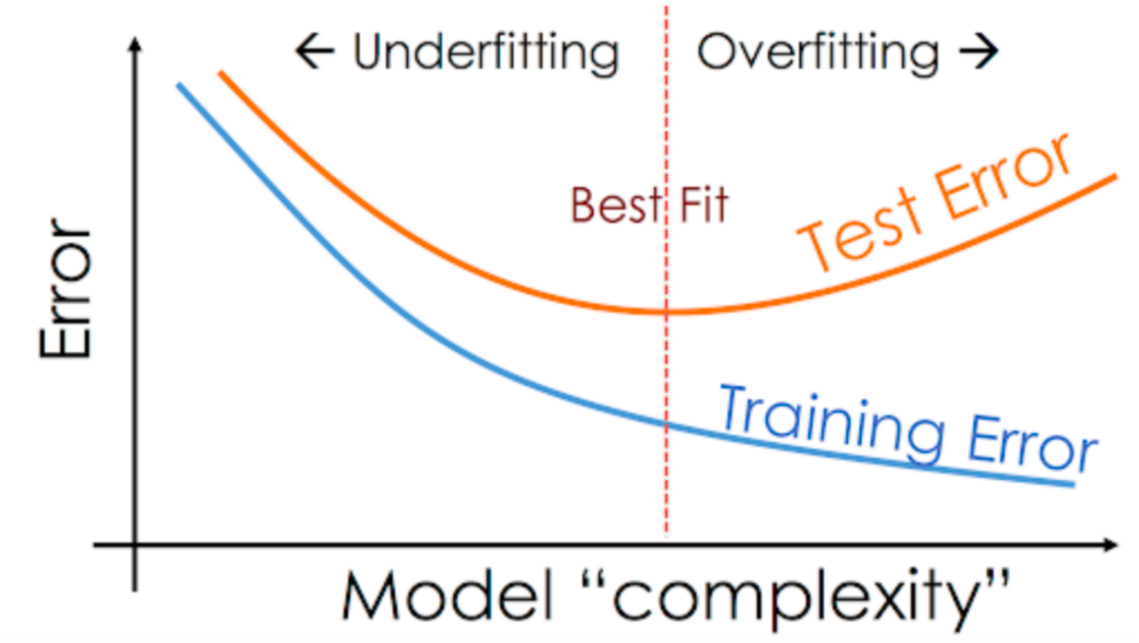


Figure 3.5. *Underfitting and overfitting*

3.1 Convolutional neural networks

For the past years Convolutional Neural Networks popularized by LeCun & al in [25] have been a state-of-the-art method in all types of machine vision tasks [26]–[30]. The popularity rises from the fact that the CNNs tend to have better performance than normal NN, while maintaining a light structure.

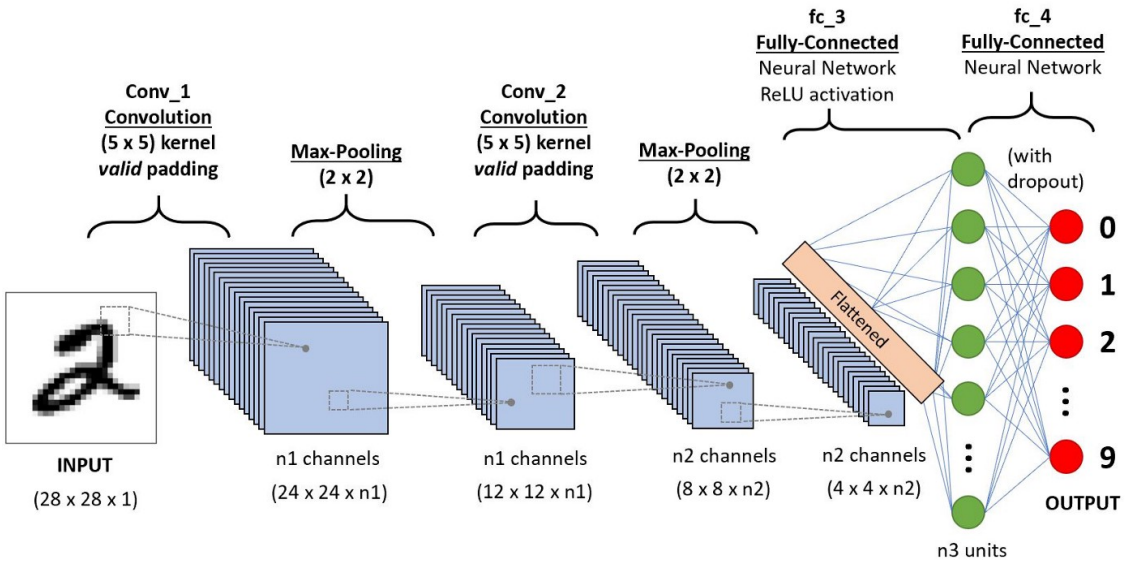


Figure 3.6. *Example structure of a CNN*

The basic principle of a CNN is the same of a NN, but now the neurons are actually computational kernels, which are small e.g. 5x5 squares as in Fig. 3.6. These kernels

convolve over the image producing feature maps, which are again fed to the next convolutional layer. The amount of kernel shift in pixels is determined by a stride, which is usually a relatively small number such as 1 or 2. After each convolutional layer, there is usually an activation function such as ReLu, followed by a spatial sub-sampling such as max-pooling. The spatial sub-sampling aims to improve the learning such that the network can learn different sized spatial features. Because the kernels share their parameters, compared to a fully connected network, the resulting structure is much lighter. Although, fully connected layers are still often used as the last layers of the CNN to produce the wanted output from the features extracted by the convolutional layers.

3.2 Autoencoder

Autoencoder is yet another subtype of a DNN, which first compresses the input into a latent representation and then decompresses it back to its original dimensions. Thus, it can be represented as having an encoder-decoder structure as seen from Fig. 3.7. The compression is done by the encoder $E_{\Phi}(x)$, which is a neural network on its own, and decompression by decoder $D_{\theta}(z)$, which does the opposite actions of the encoder. The encoder and decoder try to find their optimal respective parameters ϕ and θ in order to optimize the relation between input x and output $\hat{x}$. This relation is often measured as either similarity or difference, with e.g. using some error function such as the MSE described on Eq. 3.3. The smaller the dimensions of encoded lateral representation are on the bottleneck, the more information is lost. Thus, the autoencoder has to learn such weights that represent the most important parts of the data. Therefore, also being a powerful tool for dimensionality reduction.

Fig. 3.7 also shows example of lateral and residual connections. Residual connections also known as skip connections or identity shortcuts introduced in [31] are connections, which skip one or multiple network layers and give the same representation of the lateral data to the further layer in order to prevent the gradient from vanishing or exploding. The prevention of the extreme values of the gradient is especially important in very deep architectures. Convolutional autoencoder with residual connections is proven to beat the performance of a similar AE without residual connections in [32], underlining the importance of the residual connections for learning abstract representations by image reconstruction. Lateral connections are similar to the residual connections, but they are connected from the lateral layer in the encoder to the corresponding lateral layer in the decoder. By giving this encoded lateral representation available to the decoder, it aims to help the decoder to decode a representation more similar to the given lateral representation. Generally, convolutional autoencoders with residual and skip connections are a very popular network of choice for tasks which have something to do with images. E.g. in [33] a deep autoencoder with residual connections is part of the system used for pavement-defect segmentation.

More importantly, when looking back at the Figure 2.1 and seeing the similarities of autoencoder and basic idea of compression, one can understand why autoencoders are a very popular concept especially in the domain of image compression [34]–[38].

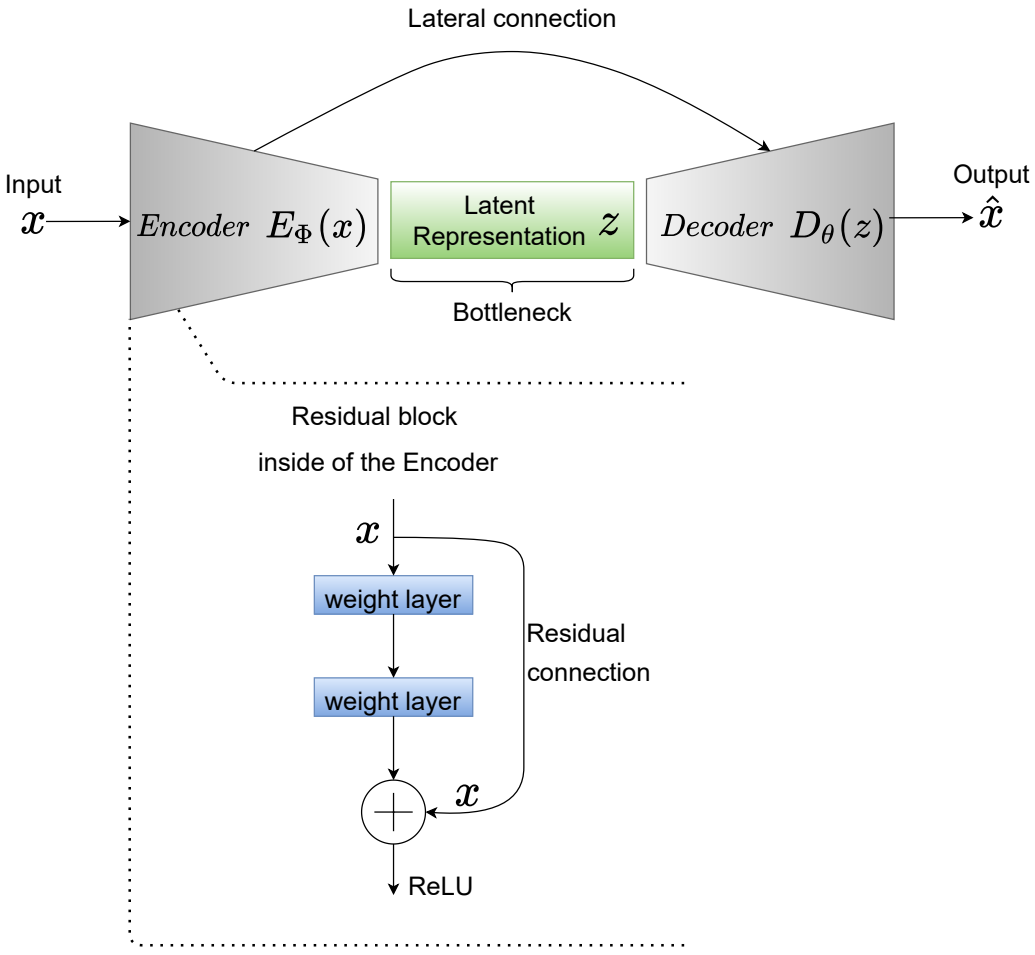


Figure 3.7. *Structure of an autoencoder including lateral and residual connections*

4. IMAGE COMPRESSION FOR MACHINE CONSUMPTION

When talking about image compression for machines, the most important thing is to maximize the performance of the images on the machine task, while keeping the compressed size minimal. Since the codecs for human consumption e.g. VVC considers only the visual quality for humans, it is clear that there must be more effective ways to code the data for machines.

First, this chapter briefly explains two very common machine tasks; instance segmentation and object detection. After this, example neural networks designed for both of these tasks are introduced. The introduced networks are also related to the training and validation of the post-processing filters implemented later on. Next, some common machine task metrics are explained. Finally, existing methods are discussed, especially the ones that are related to the method described in this thesis.

4.1 Object detection and instance segmentation with neural networks

In object detection, the system is trying to detect the object instance by drawing a bounding box over the detected instance. There can be multiple instances and they can possibly overlap each other. See the purple rectangle in Fig. 4.1, which represents a bounding box. These boxes, or rather their corner points are usually presented as some type of coordinates e.g. pixel coordinates with respect to the image.

In instance segmentation, the predictions are on a pixel-level. The systems tries to define segmentation masks, telling which pixels are part of the instance. Fig. 4.1 also shows the purple colored area, which is an example of a segmentation mask. These segmentation masks can be for example binary encoded, 1 representing the pixel is part of the instance and 0 that it is not.

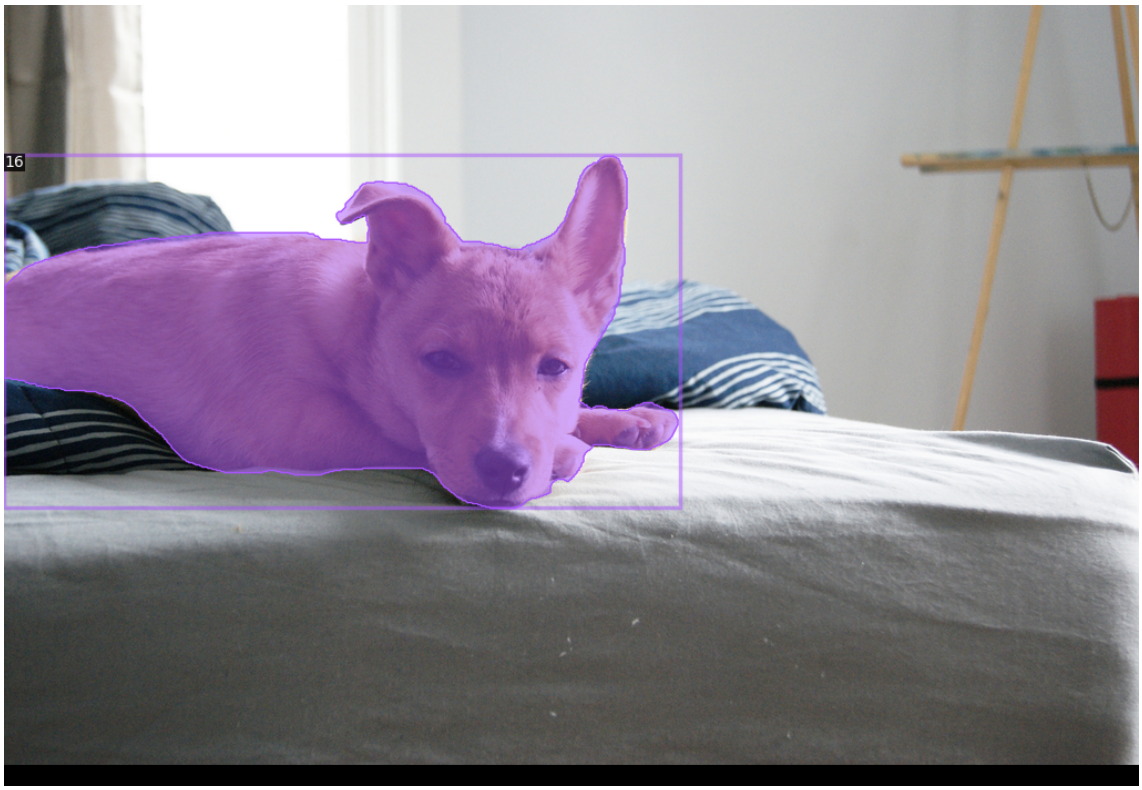


Figure 4.1. *Example of a bounding box and a segmentation mask*

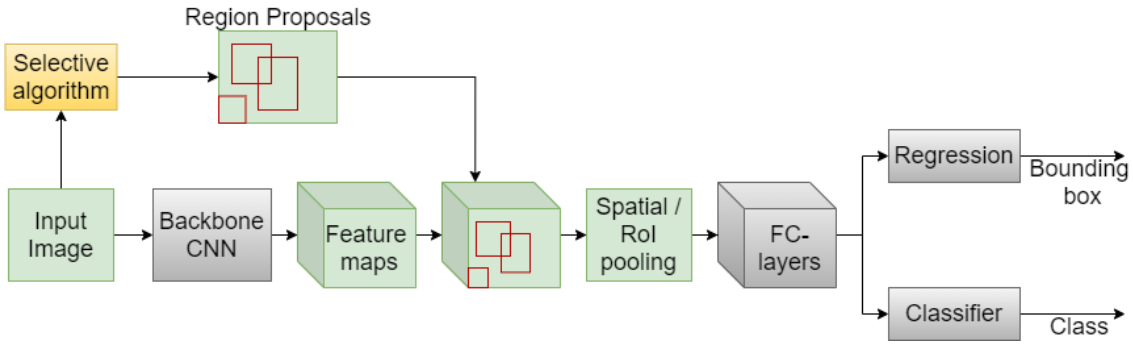
4.1.1 Region Based Convolutional Neural Networks

Region Based Convolutional Neural Networks (R-CNN) are a family of CNNs used for both object detection and instance segmentation tasks. In fact, specific versions called Faster R-CNN [27] and Mask R-CNN [28] are used in this thesis to perform object detection and instance segmentation, respectively. Thus, they are introduced more thoroughly. Both of the said networks are based on a Fast R-CNN [26], which is an object detection network. The Fast R-CNN classifies and detects objects from different regions of interest (RoI) from an image. The RoIs are selected by a selective algorithm introduced in [39]. Fig 4.2a represents a rough structure of the Fast R-CNN network. The training loss of this network can be stated as:

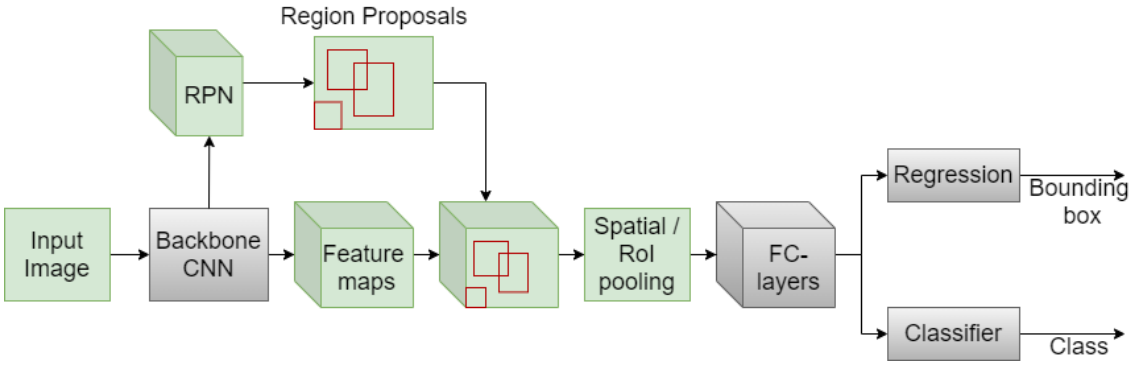
$$\mathcal{L}_{task} = \mathcal{L}_{cls} + \mathcal{L}_{reg}, \tag{4.1}$$

where $\mathcal{L}_{cls}$ is the object classification loss and $\mathcal{L}_{reg}$ is the bounding-box regression loss, calculated against the predictions and ground-truths of the class labels and bounding boxes, respectively.

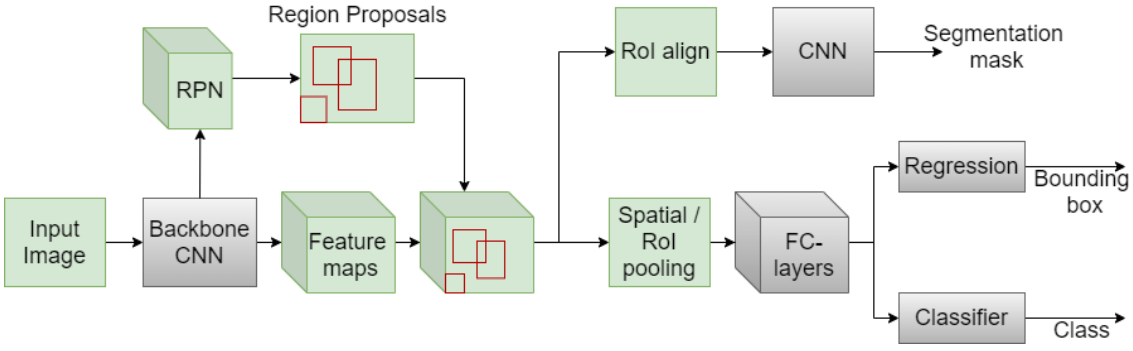
The Faster R-CNN, is very similar to the Fast R-CNN, the biggest difference is that now the Region Proposals are generated by a Region Proposal Network (RPN) instead of the selective algorithm. The RPN has access to both the original images and the features as seen in Fig. 4.2b, which increases the proposal generation speed significantly, hence



(a) *Rough structure of Fast R-CNN based on [26]*



(b) *Rough structure of Faster R-CNN based on [27]*



(c) *Rough structure of Mask R-CNN based on [28]*

Figure 4.2. *R-CNN network structures*

the name Faster R-CNN. The RPN outputs two components. First, region proposal regression, which is basically the difference between the proposed region and ground truth region. Second, the objectness, which tells that are the proposed regions more likely part of an object or a background. Thus, the training loss can be formulated as:

$$\mathcal{L}_{task} = \mathcal{L}_{cls} + \mathcal{L}_{reg} + \mathcal{L}_{preg} + \mathcal{L}_{obj}, \tag{4.2}$$

where $\mathcal{L}_{preg}$ and $\mathcal{L}_{obj}$ are the region proposal regression loss and objectness loss, respectively. With these modifications, the Faster R-CNN is both faster and more accurate than the Fast R-CNN.

Mask R-CNN is in turn an extension of the Faster R-CNN. Now, the network is extended to predict also the segmentation mask in addition to the bounding box. The segmentation mask prediction doesn't affect to the rest of the pipeline as seen in Fig. 4.2c. Training loss of the Mask R-CNN is defined as:

$$\mathcal{L}_{task} = \mathcal{L}_{cls} + \mathcal{L}_{reg} + \mathcal{L}_{preg} + \mathcal{L}_{obj} + \mathcal{L}_{mask}, \tag{4.3}$$

where $\mathcal{L}_{mask}$ denotes the segmentation mask loss, which is calculated against the ground truth segmentation masks. This training loss is especially important for one of the enhancement filter types introduced afterwards.

4.1.2 YOLOv5

This section briefly introduces the You Only Look Once v5 (YOLOv5) [30] neural network for object detection. This time the introduction is on a bit higher level than with the R-CNNs and this is because the YOLOv5 is later on used only on validating the post filters rather than training any of them. Moreover, the essential part here is to emphasize the completely different structures of the YOLOv5 and the R-CNNs, to which the validation models used on the implementation part of the thesis are based on. This aspect will be further clarified on the later sections.

YOLOv5 consists from 3 base parts, which are the Backbone, Neck and Head as represented in 4.3. The Backbone is a CNN itself, whose job is to extract the features as different types of lateral representations from the input images. The Backbone of YOLOv5 is called CSPDarknet, which combines the Cross Stage Partial network (CSP) [40] with the Darknet [41]. This Combined structure has both solid performance while it is also fast on inference stage. This is mainly because of the CSP, which ensures the gradient information is not repeated when it is changed, but rather incorporated on the feature maps, decreasing the computational cost by 20 % [40]. Next, there is the Neck, which is a Path Aggregation Network (PANet) [42] in YOLOv5. PANet boosts the information flow and combines the features given by the Backbone into a form, which the Head can

more easily predict on. Finally, the Head of the network, which is called the Yolo Layer on YOLOv5, simply just consists of three different convolutional layers, which each generate different sized feature maps to output the detection results of different sized objects.

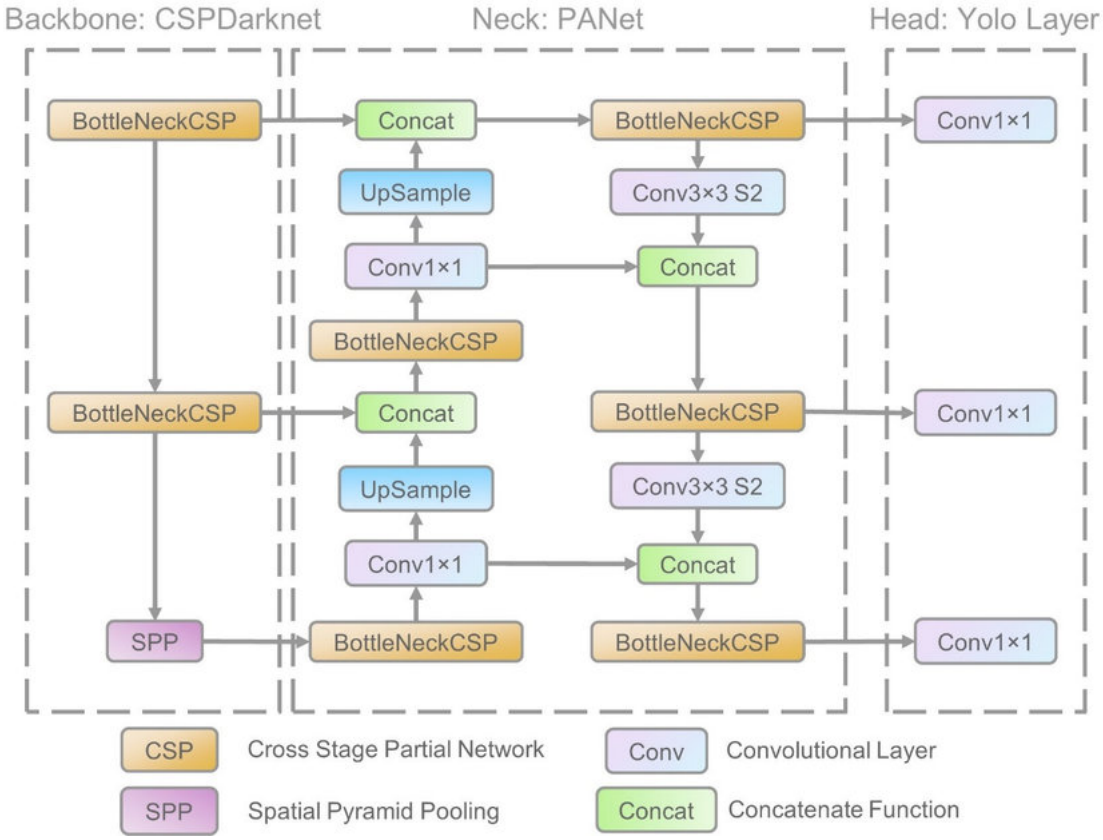


Figure 4.3. *Architecture of YOLOv5 [43]*

As was shown, the structures of YOLOv5 and R-CNN are completely different, biggest difference being that the YOLOv5 does not have an explicit RPN as in Faster R-CNN and Mask R-CNN. This is a noteworthy fact when evaluating the generalization of implemented enhancement filters later on.

4.2 Common machine task metrics

One of the most important metrics to measure a machine task performance is mean average precision (mAP), particularly for the tasks such as object detection and instance segmentation. In order to understand how this evaluation metric works, let us first define the terms *true positive*, *false positive*, *true negative* and *false negative*. True positive means that the system predicts something to be true and it is true also in reality. In false positive case, system predicts false, but in reality it is true. This logic goes on similarly to true negative and false negative cases. To determine what is true and what is false in object detection or instance segmentation, lets define a term *Intersection over Union*

(IoU) as:

$$IoU = \frac{Area\ of\ Overlap}{Area\ of\ Union}, \tag{4.4}$$

where $Area\ of\ Overlap$ is the overlap of the predicted and ground-truth bounding box in object detection task, and similarly, the overlap of segmentation masks in instance segmentation task. Correspondingly, $Area\ of\ the\ Union$ denotes the union of the prediction and the ground-truth. Thus, the values of IoU are always between [0, 1]. Now, if the IoU limit is set to e.g. 0.4, all those predictions, which would have the IoU over 0.4, would be classified as true. Similarly, those below 0.4 would be classified as false. This knowledge about the IoU is crucial, when defining the next terms which are *recall* and *precision*.

Recall is defined as follows:

$$Recall = \frac{tp}{tp + fp}, \tag{4.5}$$

where tp denotes true positive and fp false positive. Thus, recall tells how well the positive predictions were found amongst all possibilities.

Then, the precision is defined as:

$$Precision = \frac{tp}{tp + fn}, \tag{4.6}$$

where tp denotes true positive and fn false negative. Precision, on the other hand, tells how accurate the predictions are. [44]

Next, to define mAP, *average precision* (AP) is first defined. Easiest way to understand AP is to look at an example. Figure 4.4 shows a precision-recall curve for person class in detection task, where the points have been plotted on different confidence thresholds of the model, from high to low. Also, the IoU used is 0.5, usually stated as IoU @0.5. As expected, the high probability threshold causes the model to be very precise in the expense of recall and vice versa. The AP can be defined as the area under the precision-recall curve, so it too gets the values between [0, 1]. Mathematically it can be stated as

$$AP = \int_0^1 p(r)dr, \tag{4.7}$$

where $p(r)$ is the precision-recall curve. [46] When there exist no actual function for the curve, which is the usual discrete case, the AP can be simply measured by interpolating the measured points and measuring the area under curve.

Finally, the mAP can be calculated by taking the mean over AP of all classes:

$$mAP = \frac{\sum_0^n AP}{n}, \tag{4.8}$$

where n is the number of classes. Usually when measuring mAP, one might be interested

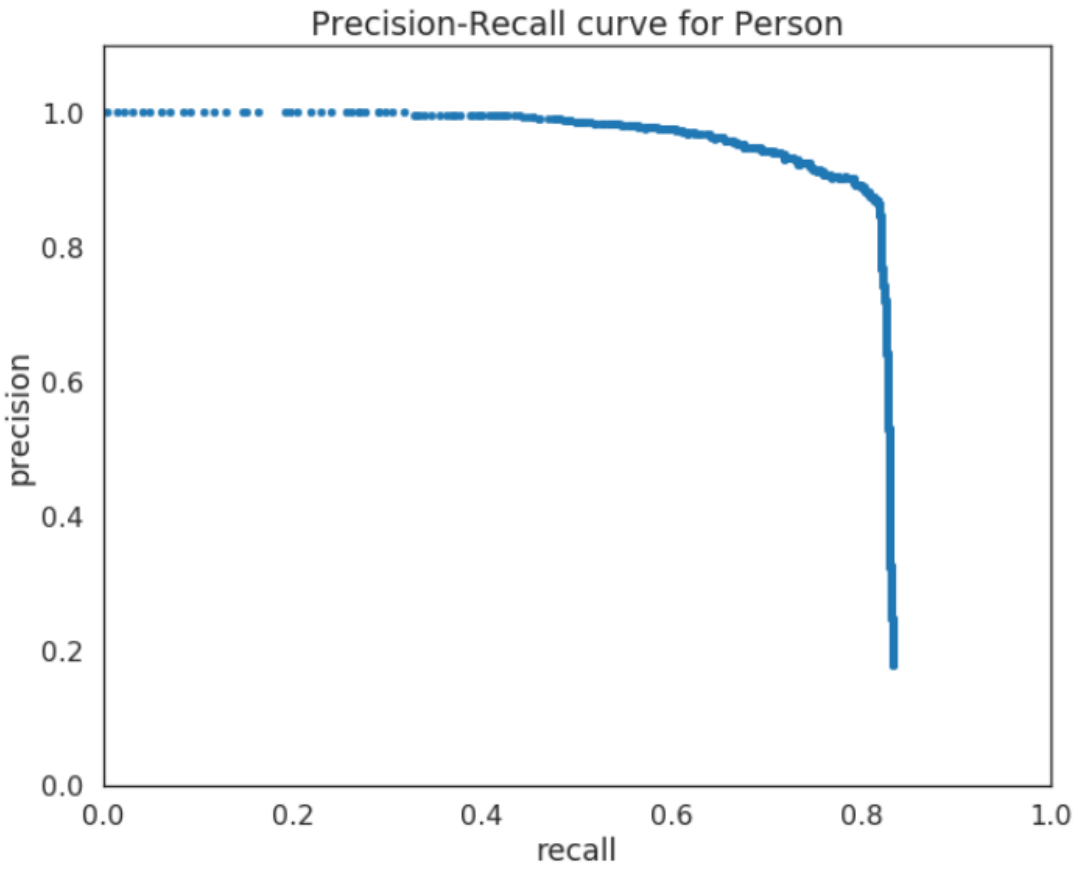


Figure 4.4. *Precision-recall curve for person class [45]*

on mAPs measured on different IoU:s, for example in the Common Objects in Context (COCO) dataset evaluation they are measured between [0.5, 1], with steps of 0.05 [47].

Other important metric apart from mAP is called Bjøntegaard Delta (BD) Rate [48]. It is a metric often used for comparing different compression methods against each other. For example, let there be an image, which is VVC encoded with quantization parameters (QPs) 42, 37, 32 and 27. Now, each of these encoded images have different BPP and mAP on a certain machine task, yielding 4 data points on mAP/BPP plot. Let there be also the same image compressed with some arbitrary method to the same BPP levels, resulting in 4 points on mAP/BPP plot which have the same BPPs as before but different mAPs. Now, an integral of the curve must be found for both of these data point sets. The BD-rate gain is actually the difference of these found integrals divided by the integration interval. I.e. the area between the plotted curves. BD-rate is also the metric used later on to compare the performance of the enhancement filters to the plain VVC compressed images. In addition to using mAP/BPP plots, BD-rate can also be calculated from PSNR/BPP plots. The Peak Signal-to-Noise Ratio (PSNR) is a metric often used to measure the reconstruction quality of some lossy codec. PSNR can be defined as:

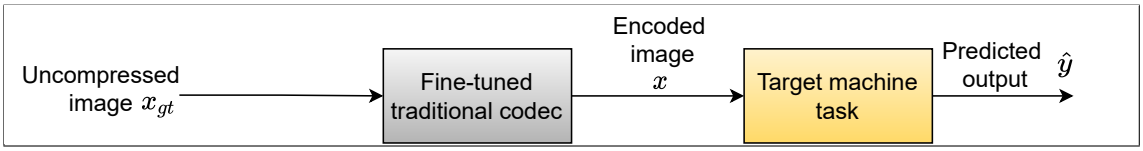
$$PSNR = 10 * \log_{10}\left(\frac{L^2}{MSE(x_1, x_2)}\right), \tag{4.9}$$

where L denotes the dynamic range of the pixel values and MSE denotes the mean squared error between the original image x_1 and reconstructed image x_2. [49] While the PSNR is not necessarily a machine task metric, it is still interesting to see how much the images reconstructed by ICM codecs differ from the original images visually.

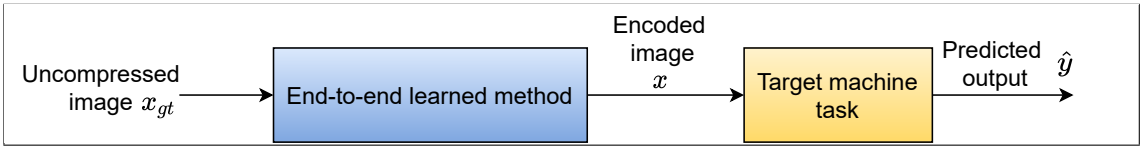
4.3 Existing methods on image compression for machines

Image and video compression for machines has been researched widely in recent years. At the time of writing this thesis, JPEG-AI, a subgroup of Joint Photographic Experts Group (JPEG) [4] has started a standardization process for image compression technologies for machines, while MPEG's VCM group [5] is investigating possibilities for standardizing a video codec for machine consumption.

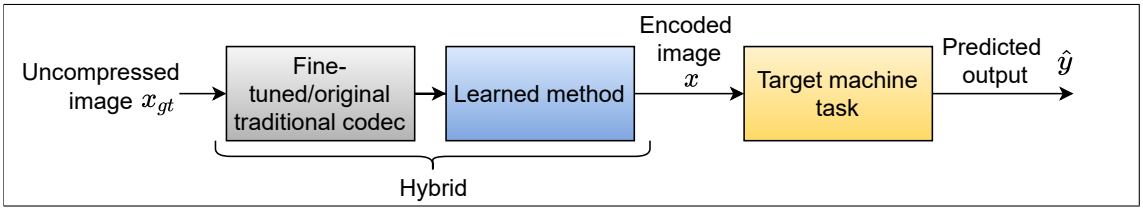
In general, three different approaches have been studied for machine-oriented image compression in the literature. First type of methods such as [50] and [51] are based on adapting traditional codecs parameters in order to achieve better task performance on target machines, see Fig. 4.5a. However, these types of methods might not be the most optimal for machine consumption, since the traditional codecs they are based on are made solely for human consumption.



(a) *Method based on fine-tuning the traditional codec*



(b) *E2E method*



(c) *Example of a hybrid method*

Figure 4.5. *Different methdods for image coding for machines on a high level*

Second type of methods, for example [37] and [38] are End-to-End (E2E) learned, meaning that they are fully based on E2E learned neural networks, see Fig. 4.5b. It is possible

that in the future the E2E learned methods will be the most optimal solution for image or video codec for machine consumption. This is due to the fact, that on the contrary of the traditional based methods, they do not have any components designed for human consumption, if designed properly.

Third and final type of methods are called hybrid methods, which use traditional codecs in addition to the learned methods. The traditional codec might be either fine-tuned or unmodified. An example of this is given in Fig. 4.5c. However, the original codec and the learned methods are not necessarily in the order described in the Figure and the line between them might not even be that clear. When compared to the E2E, hybrid methods have pros and cons. Maybe the main drawback of the hybrid method is that it still contains the traditional codec designed for human consumption (or finetuned version for machine consumption), which ultimately makes the hybrid system slightly suboptimal compared to a E2E system. Although, the effect is probably a lot smaller than with the methods solely based on traditional codecs. On the other hand, the biggest benefit of the hybrid systems over E2E systems currently is the speed. Since the traditional codecs such as VVC are so well optimized, and the learned part of the hybrid systems tend to be lighter than their E2E learned counterpart, this results in a relatively fast system. Also, as the encoder can even be left completely untouched, this increases the interoperability as well as allows to exploit the existing hardware implementations. Interesting hybrid systems are described in [52]–[54], which all use a CNN based post-processing filters to process the VVC decoded data. However, all of these methods are actually aimed to images for human consumption. Generally, it seems that this type of post filtering for machine consumption is not yet such explored topic and this is also the fact that motivated the implementation of this type of system.

5. IMPLEMENTATION OF A HYBRID IMAGE COMPRESSION SYSTEM FOR MACHINES

This section introduces a hybrid system, which utilizes deep learning based post-processing filters for image coding for machines. Some of these results are also published in a research paper [6].

First, a general overview of the system and its goal is given on a higher level. Second, the system components are introduced more thoroughly one by one. Third, three different training strategies are introduced to train three different types of enhancement filters. Also, a technique where luma partition maps are included into the filter inputs is introduced. After this, the test setup is described in detail. And finally, the results of the tests for all the three trained filters are shown and analysed.

5.1 System overview

The goal of the system is to train a post-processing filter, which enhances an image encoded and decoded by a traditional codec to further increase its performance on machine tasks, such as instance segmentation and object detection. Thus, the system can be seen as a hybrid method for ICM as described earlier on Fig. 4.5c. Since the VVC is considered as the "anchor", i.e. a baseline to which the filter performance is compared to, a sufficient goal performance-wise would be to clearly outperform this baseline. Speed-wise, it would be beneficial to have a codec, which can handle real-time applications. The system structure can be seen from Fig. 5.1. The pipeline starts by VVC encoding and decoding the original images x_{gt}, which gives the VVC decoded images x.

After this, the VVC decoded images x are fed to the post-processing filter giving the filtered output $\hat{x}$. The post-processing filter itself is a CNN based autoencoder with residual and lateral connections. The post filter will be introduced more thoroughly on section 5.1.2.

After the post-processing filter, the filtered images $\hat{x}$ are fed to the task network, which gives the predicted output $\hat{y}$. This also concludes the inference stage, which has been exactly similar to the training stage until this point. In training stage, the original image x_{gt} and its task ground truth y_{gt} are used to calculate different kind of losses, which are all

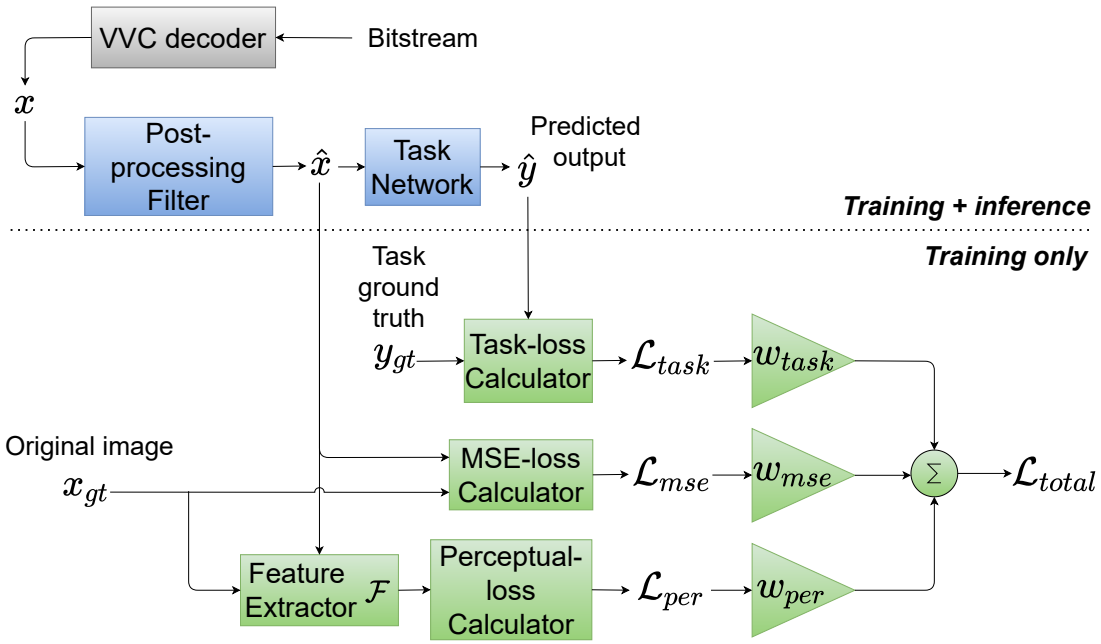


Figure 5.1. *The system pipeline. Triangles denote gain, i.e. multiplication by the corresponding weight inside of them. Sigma denotes summation. Components on green are used on training stage only, whereas blue components are used on both training and inference stages. [6]*

weighted and summed together to finally update the weights of the post-processing filter.

Now that there is a general overview about how the system works, the components and their connections to each other are introduced more thoroughly in the following sections.

5.1.1 VVC encoding/decoding of the images

First of all, before filtering any images, they are encoded and decoded with VVC codec explained in section 2.2.1. MPEG evaluation framework for VCM [49] specifies the exact technologies used in generating the VVC encoded data, which the pipeline follows.

Fig. 5.2 represents the anchor creation pipeline, i.e. how the images are processed from prior to the encoding until after to the decoding in order to generate the baseline results. First, the images are converted into a lossless PNG format, at this point they are still considered as the original image x_{gt}, which corresponds to the x_{gt} seen in Fig. 5.1. After this, the images are downscaled if needed. However, in these experiments, the accuracy of both plain VVC images and post filtered images were measured without downscaling. If downscaling were to be used, the scales would be 75%, 50% and 25% of the normal scale. Next, the images are converted to YUV color space using the FFmpeg software release $4.2.2$ [55]. After the images are in YUV format, they are encoded by the reference software VTM-8.2 [56] with All Intra configuration according to JVET common test conditions [57]. The QPs chosen for the VVC encoding were $\{22, 27, 32, 37, 42, 47, 52\}$,

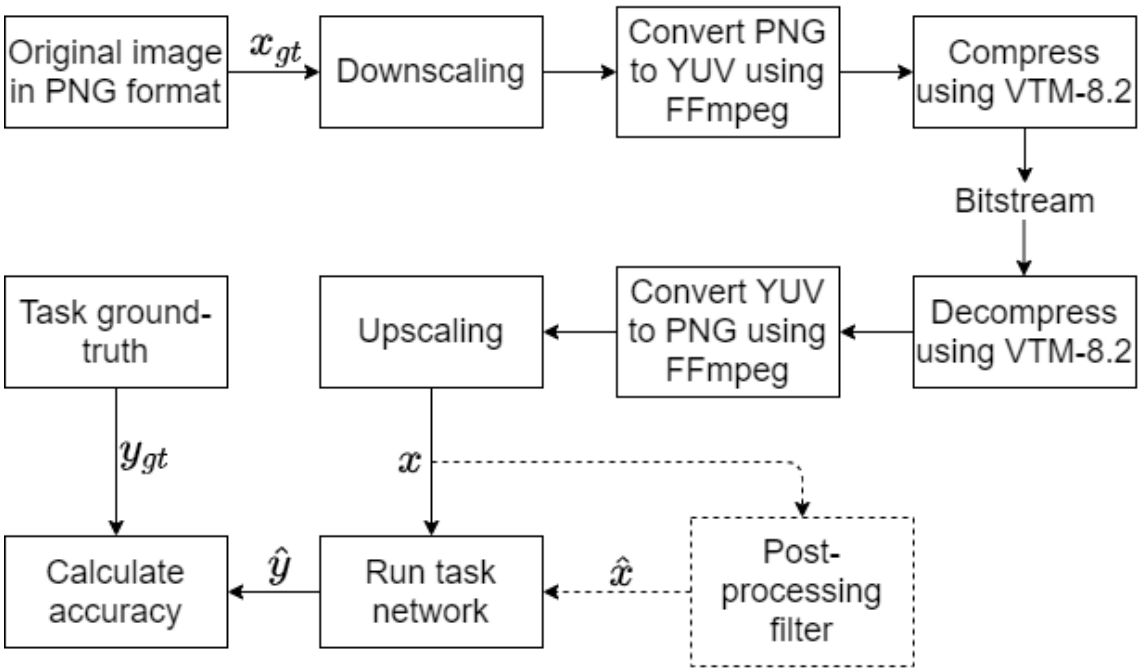


Figure 5.2. *Anchor creation pipeline, dashed lines represent where the post filtering will happen and are not part of the anchor creation. [49]*

which determine the amount of compression and therefore the quality of the decoded images, higher QP meaning more compression. The BPPs of the decoded images are measured from the bitstream at this point. Next step is the decompression, which follows the same setup as the compression. Similarly, the YUV to PNG conversion is the exact opposite of PNG to YUV conversion. After the last conversion stage, the images would have been upscaled back to the original size if downscaling was used. The resulting VVC decoded image x corresponds to the x in the system pipeline in Fig. 5.1. Finally, the VVC decoded image x can be fed to some task network, which predicts an output $\hat{y}$ and then the mAP can be calculated using the task ground truth y_{gt} to produce the VVC anchor results. Alternatively, before giving the VVC decoded image x to the task network, it can be post filtered to become $\hat{x}$, which corresponds to the $\hat{x}$ in Fig. 5.1. This should improve the mAP on the machine task, which this hybrid system is all about.

5.1.2 Post-processing filter

As mentioned, the whole goal of the system is to train a post-processing filter, which enhances the VVC reconstructed images such that they perform better on machine tasks. The structure of this filter can be seen from Fig. 5.3. As can be seen, it is a CNN based autoencoder, a concept explained in section 3.2. Other facts to note are that there are two lateral connections between the encoder and decoder and also that there are residual connections in the convolutional blocks. Both of these connections boost the performance significantly, since part of the goal is to try to reproduce the original images x_{gt} from VVC

compressed images x, which are fed to the post filter. Activation functions used are PReLU and ReLU, which were explained in chapter 3. This structure was chosen as the result of multiple experiments with different structures. The chosen structure was kept unmodified through every test that followed. The structure is also lightweight, having only $782,663$ trainable parameters.

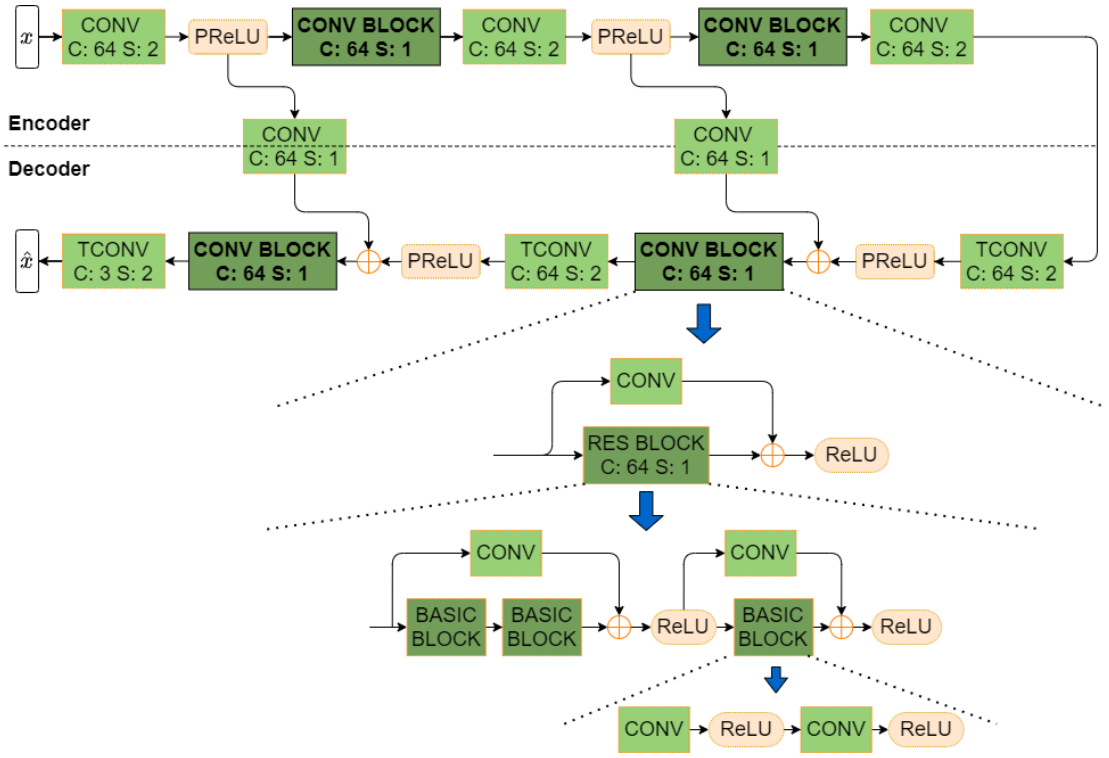


Figure 5.3. *Structure of the post-processing filter. x and $\hat{x}$ denote the VVC encoded image and output of the post-processing filter, respectively. TCONV denotes a transposed convolutional layer. C denotes the number of output channels and S the stride for all the convolutional blocks. Parameters of the children blocks are inherited from their parent blocks. [6], [38]*

5.1.3 Task network

In addition to the MSE-loss, the system has an option to include a task loss component to further increase the performance on machine tasks. For this, a task network is needed. Thus, a Mask R-CNN (see section 4.1.1) based network for instance segmentation was chosen as the task network. However, the setup is not dependent on this particular task network and would also work with different task networks. Task network weights are kept frozen throughout the experiments, since the goal is to only train the post-processing filter. The task loss is calculated between the predicted output $\hat{y}$ and the task ground truth y_{gt} as seen in Fig. 5.1. The Task loss is defined as the training loss of the Mask R-CNN network, see Eq. 4.3 from section 4.1.1.

5.1.4 Perceptual loss component

The perceptual loss calculation is added to the system in hope for a better generalization of the enhancement filter. It consists of feature extractor $\mathcal{F}$ and the perceptual loss calculator as seen in Fig. 5.1. The feature extractor is based on VGG-16 [29], which is a CNN for object detection and classification. The actual VGG-16 model used is pretrained on ImageNet dataset [58]. The perceptual loss of the system is defined as:

$$\mathcal{L}_{per} = \mathbf{MSE}(\mathcal{F}_2(\hat{x}), \mathcal{F}_2(x_{gt})) + \mathbf{MSE}(\mathcal{F}_4(\hat{x}), \mathcal{F}_4(x_{gt})), \tag{5.1}$$

where $\mathbf{MSE}$ denotes mean squared error operator. $\mathcal{F}_2(\hat{x})$ and $\mathcal{F}_4(\hat{x})$ denote the feature tensors extracted from the filtered image $\hat{x}$ after the second and fourth max-pooling layer of the VGG as visualized in Fig. 5.4. $\mathcal{F}_2(x_{gt})$ and $\mathcal{F}_4(x_{gt})$ denote the same for the original image x_{gt}.

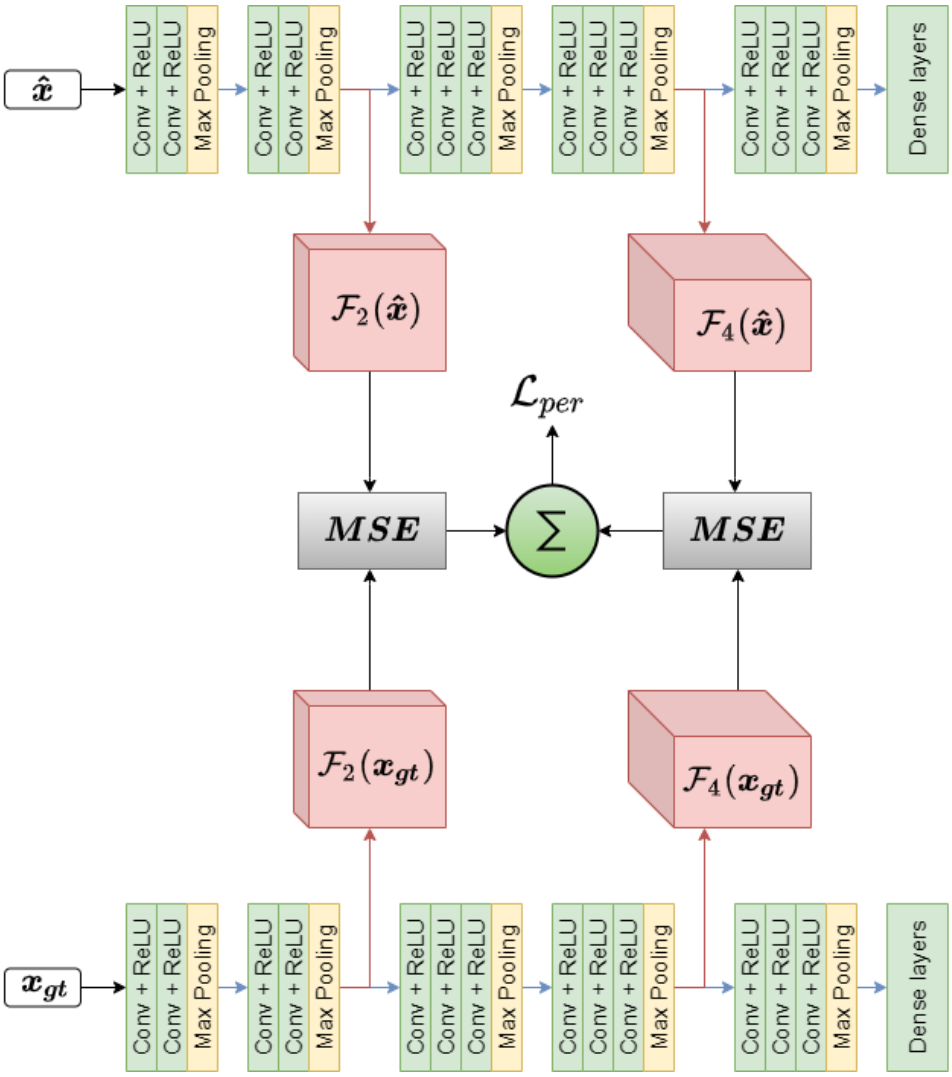


Figure 5.4. *Feature extraction and perceptual loss calculation [6], [38]. Sigma denotes summation.*

5.1.5 Enhancement filter types

Now that the needed sub-losses have been defined, the total loss can be stated as:

$$\mathcal{L}_{total} = w_{mse} \cdot \mathcal{L}_{mse} + w_{task} \cdot \mathcal{L}_{task} + w_{per} \cdot \mathcal{L}_{per}, \tag{5.2}$$

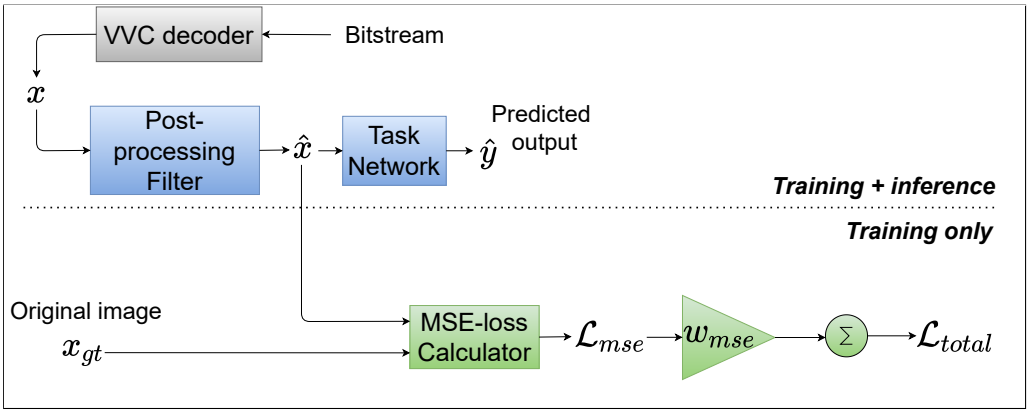
where $\mathcal{L}_{mse}$ denotes the mean squared error between the filtered image $\hat{x}$ and the original image x_{gt}. $\mathcal{L}_{task}$ denotes the task loss defined in Eq. (4.3) and $\mathcal{L}_{per}$ denotes the perceptual loss defined in (5.1). w_{mse}, w_{task} and w_{per} are the analogous weights of the losses, which stabilize the effect of each loss.

By changing each of the weights in Eq. 5.2, three different filter types are constructed. Therefore, the structures of these filters are similar, but only the training strategy is changed. First, by setting (w_{mse}, w_{task}, w_{per}) to (1, 0, 0), respectively, a filter called Baseline Fidelity Enhancement (BFE) is introduced. Thus, only MSE-loss is used to train the post-processing filter, which means that in the training stage described with green squares in Fig. 5.1, only the middle section is used. The word Baseline in BFE comes from the fact that the BFE filter works as a training starting point for the other filters, which will be introduced next.

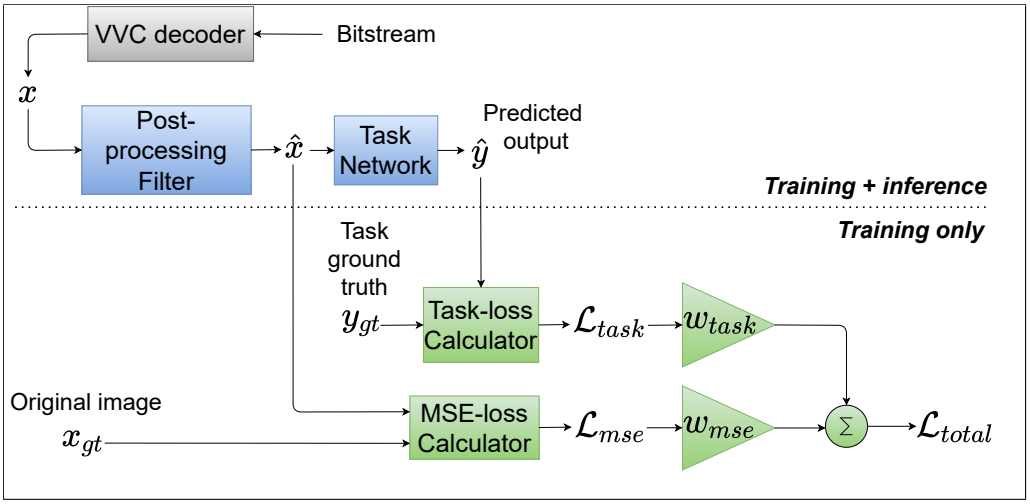
The second filter is called Task-Specific Enhancement (TSE) filter. As mentioned, the starting point of this filter type is always the pretrained BFE. This is simply due to the fact that it gives the best performance according to multiple empirical tests. The TSE filter is constructed by setting the weights (w_{mse}, w_{task}, w_{per}) in 5.2 to (1, 0.01, 0), respectively. This includes the Mask R-CNN training loss described in Eq. 4.3 as a task loss into the equation.

The third filter is introduced to make the system more task agnostic and still perform on a high level. This filter is called Task-Agnostic Enhancement (TAE), which is composed by setting the (w_{mse}, w_{task}, w_{per}) in 5.2 to (1, 0, 0.01), respectively. Now in addition to the MSE-loss, a perceptual loss is used instead of the task loss. The pretrained BFE is a starting point for also to this filter. To make the filter training setups more clear, in Fig. 5.5 all the three setups have been visualized similarly as the whole system was visualized in Fig. 5.1.

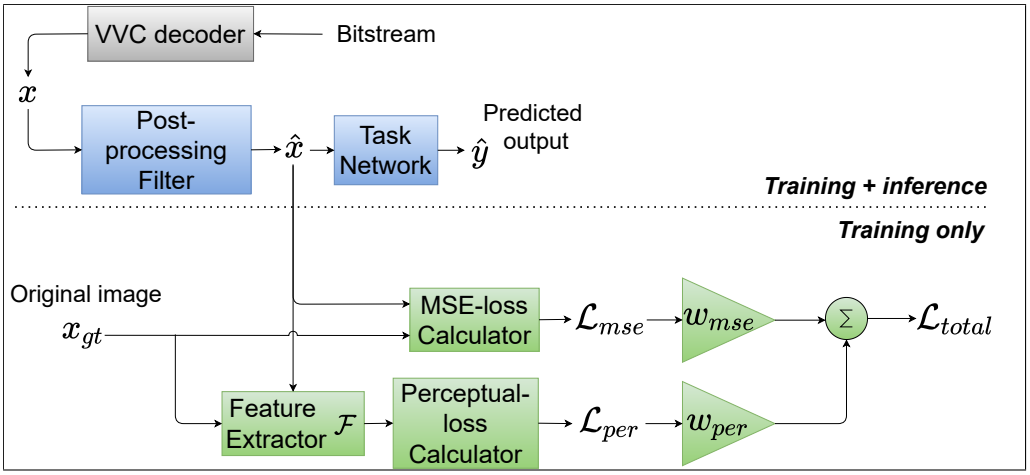
As can be seen, these weighting setups simplify the training stage described with green components quite a lot. One another type of a filter could naturally be a one, where all of the loss components are included. However, initial tests showed that the $\mathcal{L}_{task}$ and $\mathcal{L}_{per}$ seem to work better when used separately, thus this aspect is not further considered in this thesis.



(a) *Baseline Fidelity Enhancement (BFE)*



(b) *Task-Specific Enhancement (TSE)*



(c) *Task-Agnostic Enhancement (TAE)*

Figure 5.5. *Training pipelines of the three enhancement filters*

5.2 Luma partition maps on filter inputs

Luma partition maps can be said to contain information about the more detailed areas of the image. They are in fact the luma components of the partition maps described in

section 2.2.1. As can be seen in Fig. 5.6 the more detailed areas or blocks are encoded with values closer to 0 (darker in the image) and less detailed blocks as values closer to 1 (whiter in the image). A method is suggested, where these luma partition maps are concatenated to the filter input as an extra channel such that the RGB images shape of (3 x Height x Width) become (4 x Height x Width). This extra data could possibly help the filters to learn the most important details better, resulting in a boost on a task performance. Note that this method can be used with any filter type and can be considered as an extra complementary tool.

(a) *VVC encoded image at QP 42* ***(b)*** *VVC luma partition map at QP 42*

Figure 5.6. *Example of a luma partiton map from VVC encoded image (QP 42)*

5.3 Test setup

All the enhancement filters were trained with the same dataset, which is a subset of the Open Images dataset [59]. This subset was constructed such that $30,000$ images were randomly chosen from the training set of the Open Images dataset, which all had to contain a minimum of 1 instance of the following 5 classes: *bird*, *cat*, *dog*, *person* and *car*. This 5-class training dataset is denoted as $\mathcal{X}_t$. Each filter was trained with these images from $\mathcal{X}_t$, which were first VVC encoded with 7 different QPs, $\{22, 27, 32, 37, 42, 47, 52\}$ as mentioned in 5.1.1, resulting in a total of 7 different models for each of the three filters. All the models of the BFE filter were trained for 150 epochs until they converged, meaning that the performance wasn't getting noticeably any better. TSE and TAE filter models used the best performing BFE filter of each QP as their starting point and were both trained for 70 epochs. All the models were trained on Nvidia DGX1 system on a Tesla V100-SXM2 16GB GPU. The trainings for BFE, TSE and TAE took roughly 45min, 3h5min and 1h20min per epoch, respectively. Thus, there is a quite big difference on training times of these filters favoring the TAE and BFE over TSE.

For the evaluation, two different datasets were chosen. The first dataset was evaluated on both instance segmentation and object detection tasks and is denoted as $\mathcal{X}_1$. It consists of $2,352$ randomly selected images from validation set of the Open Images and has the

same classes as in $\mathcal{X}_t$.

The second evaluation dataset is a validation set of the COCO dataset which contains $5,000$ images from 80 different classes [60] and is denoted as $\mathcal{X}_2$. The evaluation is considered in two ways; 80 classes involved and just the 5 classes that correspond to the $\mathcal{X}_t$. This is to see how the filters trained with 5 classes generalize to the 80 class case. All the evaluation datasets are encoded and decoded similarly as the training set by VVC according to what was mentioned in section 5.1.1.

The evaluation procedure for different datasets also differs from each other. This is to further emphasize the validation diversity and to better asses the generalization. The $\mathcal{X}_1$ validation dataset is evaluated on both object detection and instance segmentation tasks. The validation models are similar to the task network used in training the TSE, more precisely *Mask_rcnn_X_101_32x8d_FPN_3x* for the instance segmentation task and *Faster_rcnn_X_101_32x8d_FPN_3x* for the object detection task. Both of the models are provided by Detectron2 [61], which is a Python framework for computer vision from Facebook AI. Evaluation metrics used were the mean Average Precision (mAP@0.5) according to [62] for task performance and Bits Per Pixels (BPP) as the metric for "bitrate". Those different bitrates are obtained by encoding the datasets with different QPs using the VTM-8.2 as stated in 5.1.1. The COCO evaluation dataset $\mathcal{X}_2$ on the other hand, is only evaluated on the object detection task. This is mostly because the used evaluation task network supports only detection. The task network used for the evaluation is YOLOv5s, which is the smallest version of the 4 available YOLOv5 models. Structure of YOLOv5 was explained in section 4.1.2. Again, it is a good network of choice for evaluating the generalization of the trained filters, as it has a completely different structure and is a lot smaller than the R-CNN networks used on validating the performance on $\mathcal{X}_1$. For the task performance on $\mathcal{X}_2$, the evaluation metric used was mAP@[$0.5 : 0.05 : 0.95$].

The performance of all the enhancement filters described in 5.1.5 were compared to the results which were only encoded and decoded with VVC. The metric chosen for this comparison was Bjøntegaard Delta Rate (BD-Rate) metric [48], explained in section 4.2. BD-rates with PSNRs were also extracted from the filtered results to see if the filters improve the visual quality of the images.

When constructing the setup, empirical tests showed that for every filter type, losses don't necessarily follow exactly the validation task accuracies, although they are a very good indicator what the accuracies are going to be. This caused learning rate schedulers to have only a little effect on the performance. Thus, considering the computational cost of the training and available computing resources, the learning rate of the system was fixed to $1e-4$, which seemed to produce the best results. The optimizer used with this learning rate for every filter was Adam, explained in chapter 3.

The whole system, besides the VVC encoding/decoding, was implemented with Python3

[63] programming language with the help of PyTorch [64] framework. Generally in the field of machine learning, Python combined with the PyTorch framework is a very common combination, especially when conducting some research prototyping.

5.4 Test results

5.4.1 Performance against VVC

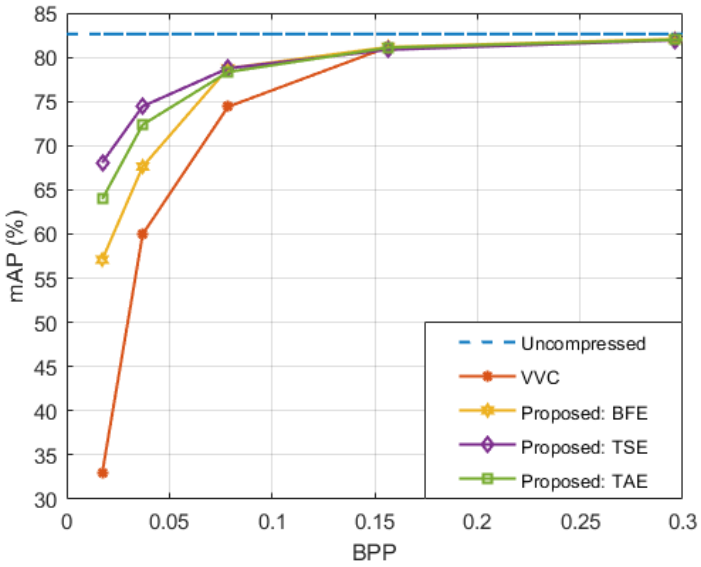
For clarification, luma partition map method described in 5.2 was not utilized in any of the results described on this subsection. Table 5.1 shows the BD-rate gains in context of mAP and BPP for the trained filters on different evaluation datasets and networks over plain VVC. The table shows that every filter improve the performance significantly. However, lets consider the BFE filter first.

Table 5.1. *Comparison of average BD-Rates (%) of the different filters over VVC-only decoded images [6]*

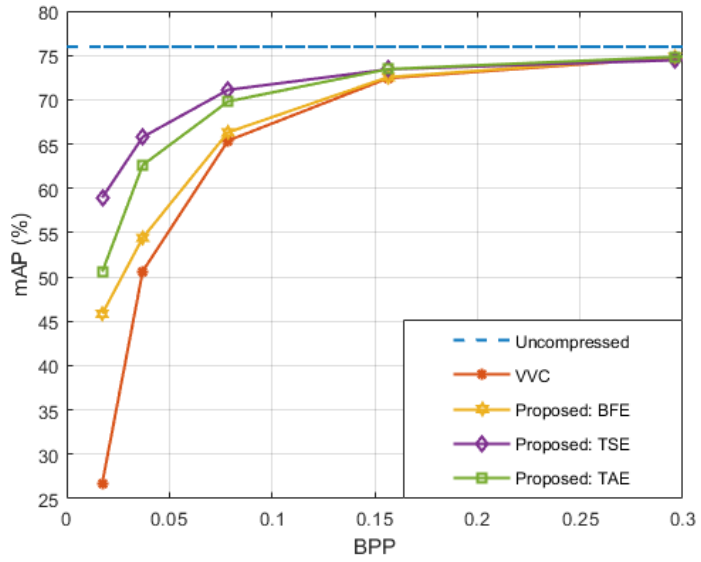
Task network	**BD-Rate (with mAP)**		
	BFE	***TSE***	***TAE***
Instance Segmentation on $\mathcal{X}_1$ (Mask R-CNN)	–30.72%	–45.79%	–40.30%
Object Detection on $\mathcal{X}_1$ (Faster R-CNN)	–13.91%	–49.39%	–40.57%
Object Detection on $\mathcal{X}_2$ with 5 classes (YOLOv5s)	–7.43%	–13.51%	–30.02%
Object Detection on $\mathcal{X}_2$ with 80 classes (YOLOv5s)	–6.65%	–5.14%	–24.85%

As expected, the BFE filter improves the performance on every test and has quite similar performance despite if there are 5 or 80 classes, since it doesn't use the task network information on its training stage. Interestingly, the gains on the COCO evaluation dataset $\mathcal{X}_2$ are not as good as the results on $\mathcal{X}_1$. One reason for this might be the size differences of the two datasets; $\mathcal{X}_1$ had maximum dimension size of 1024 pixels, which is the same size as the training images, while in $\mathcal{X}_2$ the max dimension size was 640 pixels.

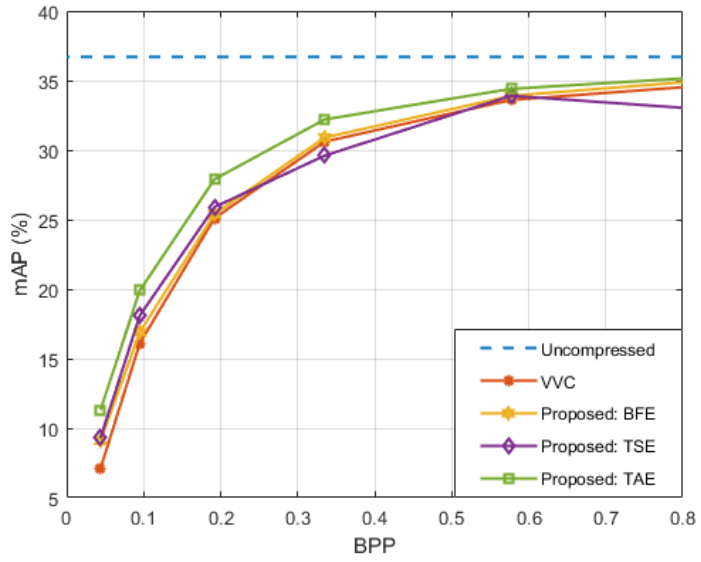
TSE filter on the other hand improved the detection and segmentation task the most for the $\mathcal{X}_1$, 45.79% and 49.39%, respectively. It seems that the detection task is actually better for the TSE, even though the training was done with instance segmentation task. The reason for the fact that the detection task performance also increases along with the segmentation is partly explained by the fact that the object detection is a sub-task of the instance segmentation when deriving the segmentation task loss, see Eq. (4.3). However, there is a huge drop on the performance, when the evaluation dataset and network are changed. This emphasizes the fact, that the TSE has to have an access to the task network it is going to be used on, otherwise the results are not so impressive. Similarly, there is a largest difference between considering results with 80 or 5 classes for this method.



(a) *Instance segmentation task with Mask R-CNN on* $\mathcal{X}_1$



(b) *Object detection task with Faster R-CNN on* $\mathcal{X}_1$



(c) *Object detection task with YOLOv5s on* $\mathcal{X}_2$ *- 80 classes*

Figure 5.7. *Rate-performance curves of different methods [6]*

The TAE filter seems to the best generalizing filter of the three according to the performance. It achieves a very solid, roughly 40% BD-rate gain on both instance segmentation and object detection tasks over plain VVC on $\mathcal{X}_1$. Even better, it achieves a 30.02% BD-rate gain on $\mathcal{X}_2$ with 5 classes and 24.85% with 80 classes. The fact that this filter is able to achieve such a gain on a task network it has never seen before and is trained using only images from 5 classes is impressive. In this regard, all the filters would most likely have even better performance on 80 classes, if those classes would be used to train the filters. This is true even for the BFE and TAE, since there would be more variety on the actual image data if images containing different classes would be introduced.

The Fig. 5.7 represents the rate-performance curves i.e. BPP–mAP of all the three filters in addition to the plain VCC and uncompressed results. From the Figure, it can be seen that the filters provide much more gain in the lower BPP area, especially for the R-CNN networks on $\mathcal{X}_1$. For the YOLOv5 on $\mathcal{X}_2$, this is not so drastic, but still noticeable. It seems that with higher BPP, the performance of the VVC encoded images are already near the uncompressed ones, thus the enhancement filters are not able to introduce much gain on the higher BPP end. Since the task performance on the high QPs is already very high, the enhancement on this high BPP end is considered as good to have and not necessary as important as on the low BPP end.

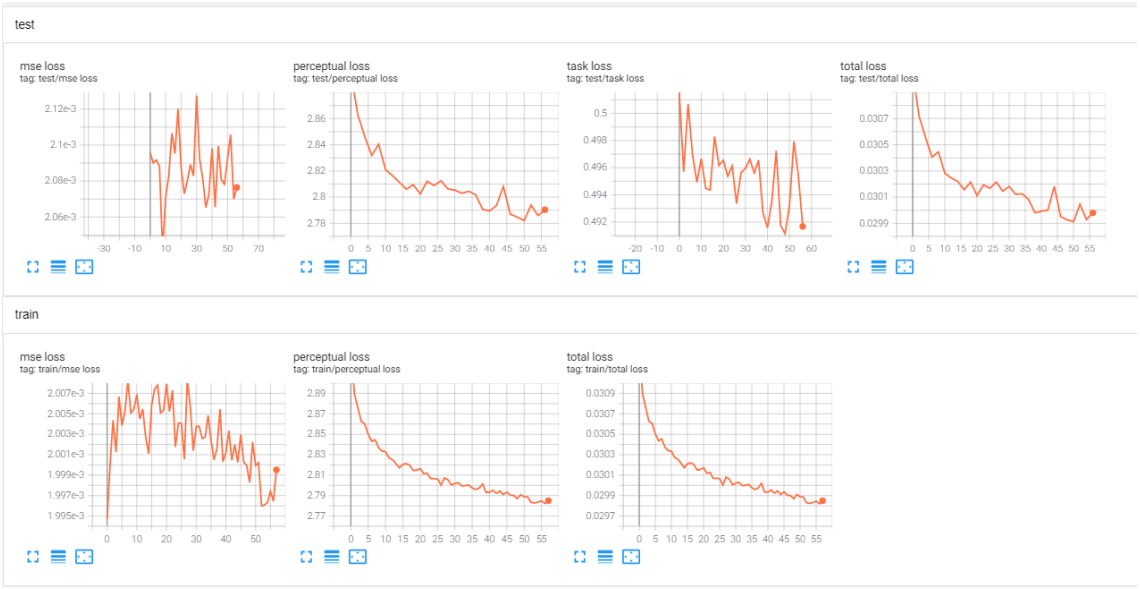
For every filter, the filtering speed on $\mathcal{X}_1$ validation set (max dimension of 1024 pixels) is approx. 13 ms/image, which is roughly 77 frame-per-second. Since the enhancement filters operate completely on the decoder side, this implies that all the enhancement filters would also be eligible for real-time applications. Moreover, this shows that the speed of all the enhancement filters are already on the level that they could be tested on video tasks.

5.4.2 Impact of luma partition maps

Fig. 5.8 illustrates the losses during training and validation on $\mathcal{X}_1$ for TAE filter, trained with and without luma partition maps. As seen, the losses are generally quite similar. The reason for the fact that only losses are shown for the luma partition map method for one QP is that the resulting mAP gains were not so impressive. Moreover, it gave small gains only on QPs 47 and 52. With lower QPs the results would actually get worse. This is probably due to the fact that with QPs 47 and 52 the images are so bad quality that this type of extra information can help the filters learn the details better. For the better quality images, the filter cannot really understand what to do with the extra data. Multiple tests were made to fine-tune this method by tweaking the hyperparameters such as the loss weighting in Eq. 5.2. It was also noted that it is important performance-wise to initialize the network weights for the luma partition map data channel close to 0, especially if a pretrained model is used as the starting point of the training. But, since the results were not getting significantly better, and the extra data added more complexity to the filter,

this method was put on hold. Further investigations would be needed to find a way to represent the luma partition maps, or some other extra data to the filter in a way that there would be significant benefits.

(a) *TAE losses at QP 47 without luma partition maps*



(b) *TAE losses at QP 47 with luma partition maps included*

Figure 5.8. *Example of TAE losses at QP 47 with and without luma partition maps*

5.4.3 Visual impact

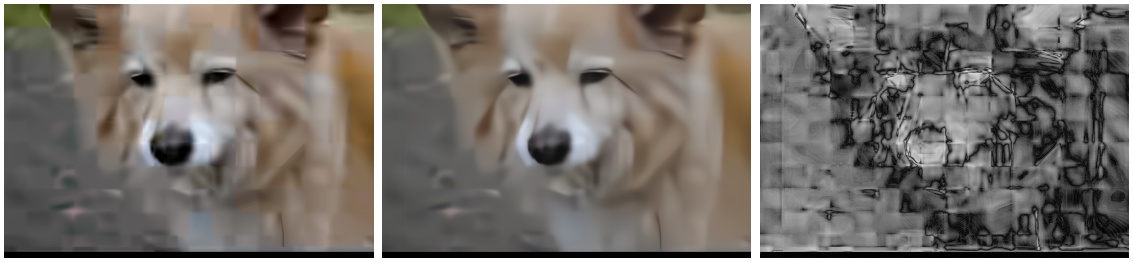
The table 5.2 shows the BD-rate percentage with PSNR along with the PSNR gain in dB for all the three enhancement filters. As both metrics suggest, there is quite a big improvement on the reconstructed image quality for all the filters. Most of the gain in the table metrics comes probably from the correction of the distortion caused by two color

conversions stages from RGB to YUV and vice versa, mentioned in section 5.1.1. This requires further study of the color conversion stages to be sure of the root cause.

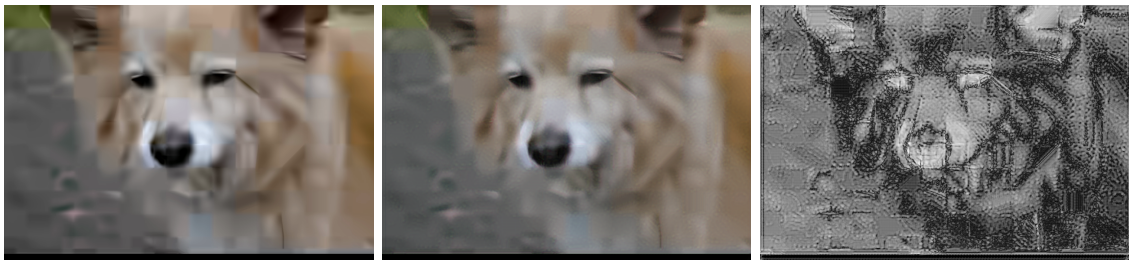
Table 5.2. *Average BD-Rate (%) and PSNR [dB] gains of the three enhancement filters over the plain VVC in RGB color space for $\mathcal{X}_1$ [6]*

Filter	*BD-Rate (with PSNR)*	*PSNR gain [dB]*
BFE	–84.48%	+5.55
TSE	–78.65%	+4.26
TAE	–81.52%	+5.35

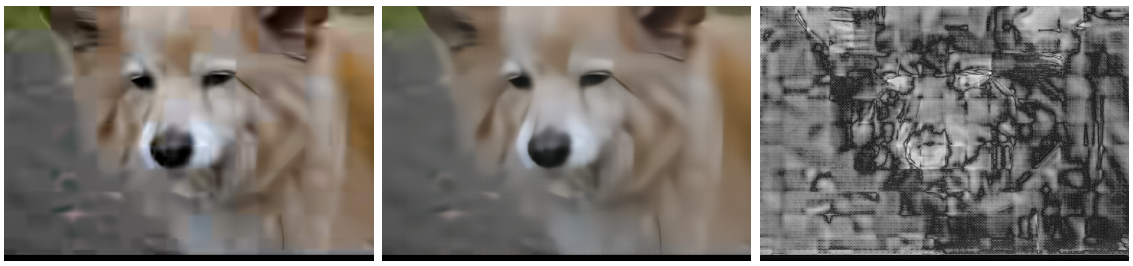
Fig. 5.9 illustrates examples from the QP 52 for the plain VVC and all the filters followed by corresponding difference image of the filtered and unfiltered image. QP 52 is the best QP to show visual impact, since the visual effect of filtering tends to be higher with higher QPs. Plain VVC encoded images also have a blocky structure, which is visually present on high QPs.



(a) *VVC* ***(b)*** *VVC + BFE* ***(c)*** *log difference*



(d) *VVC* ***(e)*** *VVC + TSE* ***(f)*** *log difference*



(g) *VVC* ***(h)*** *VVC + TAE* ***(i)*** *log difference*

Figure 5.9. *Example of each filter and their difference images compared to plain VVC at QP 52 on $\mathcal{X}_1$*

The BFE has efficiently smoothed the blockiness caused by the VVC, and the highest

difference area is the face and ears of the dog. TSE filter, on the other hand, has applied the filtering even more to the area of interest (head of the dog), as seen in the residual image and it hasn't reduced the blockiness as much as the BFE. Finally, the TAE filter has produced image quite similar to the BFE. Also, if looked closely, it can be seen that there is this dim grid-like pattern over the TAE-filtered image. Visually the filtered images seem to follow the same relation as the PSNR gains in Table 5.2.

5.4.4 Performance improvement visualization

The performance improvement on the instance segmentation and object detection is further illustrated by a visualization in Fig. 5.10, where examples from two different images drawn from $\mathcal{X}_1$ are shown. The top images are from QP 42 and the bottom ones from QP 47. From left to right, the images along with the predictions visualized on top of them are from: original image, plain VVC encoded, BFE-filtered, TAE-filtered and TSE-filtered, respectively.

First, considering the images from QP 42, there are two ground truth annotations on the example image. The first one is a bird that can be seen in the middle of the image and the second one is a bird on the top left corner of the image. There is also a zoom-in image included from the other bird. Now, considering the original uncompressed image, both of the birds are found from the image, along with a couple of false positive bird predictions close to the top left bird. For the VVC compressed image, the validation model cannot find anything from the top left corner, but rather finds false detections from the top right corner. It finds the bigger bird, but also detects its shadow, which is considered as a false positive detection. BFE filtering can be seen to fix the top right corner false positives from the VVC encoded image, but the bird on the top left corner still cannot be found. The bigger bird is found along with its shadow. Next, from the TAE filtered image, it can be seen that both of the birds are found without any false positives. In fact, in this example, the TAE filtered QP 42 VVC encoded image performs better in object detection and instance segmentation tasks than the original uncompressed one. Finally, the TSE filter can be seen to produce similar predictions than the original image containing the false positive predictions next to the other bird.

Next, considering the images on bottom rows from QP 42. These images have a ground truth annotations on two dogs, which can clearly be seen on the center of the images. This time all of the methods result in the model finding both of the dogs, but from plain VVC image the model also finds a person and a bird from the same spot where the other dog is. All of the enhancement filters remove these false positive predictions and provide significantly better segmentation masks than the plain VVC.

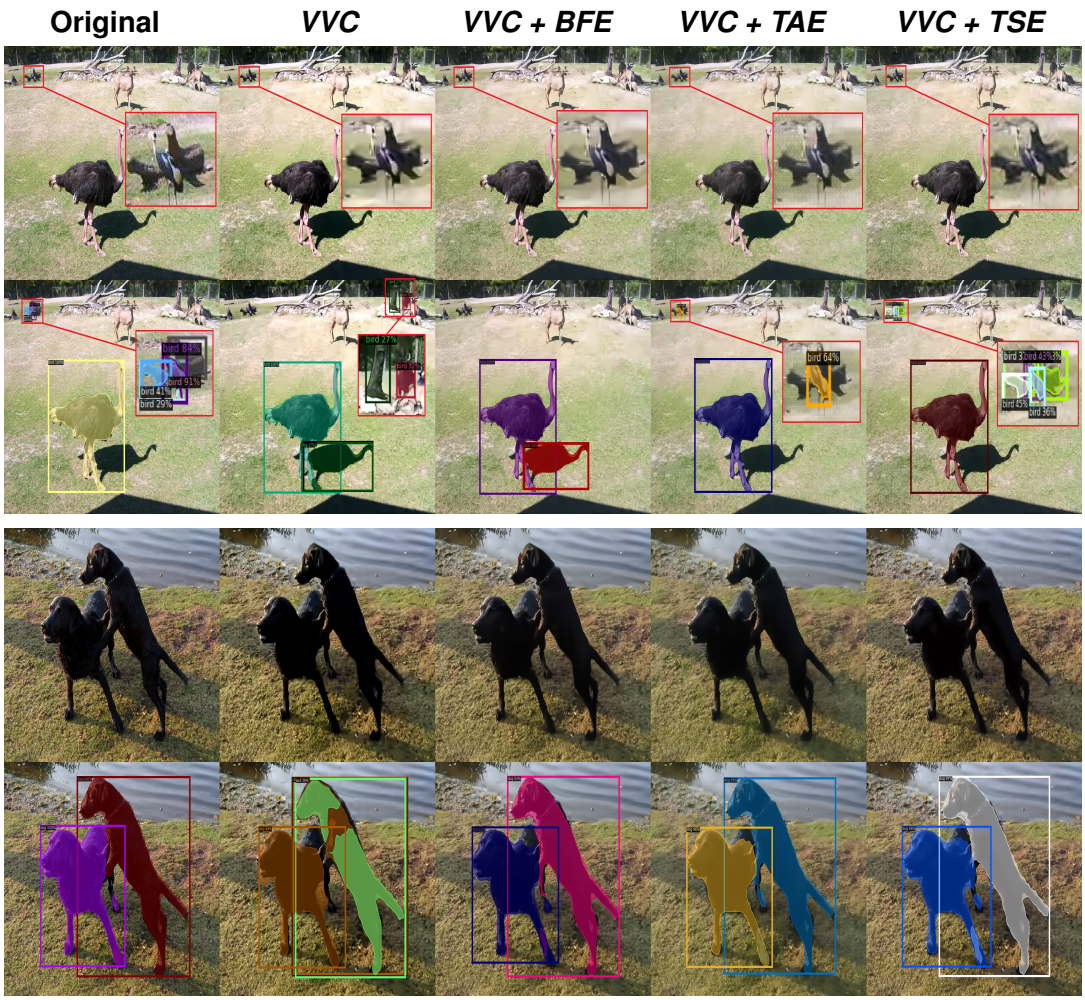


Figure 5.10. *Examples of machine task performance improvements on object detection and instance segmentation at QP 42 (top images) and QP 47 (bottom images) [6]*

6. CONCLUSION AND FURTHER WORK

One goal of this thesis was to introduce the basics of image compression for machines to the reader, along with the complementary topics surrounding the subject, in order to understand the implemented system better. This was addressed by the theory part, which include the chapters 2, 3 and 4. Second, and more importantly, it was wanted to know how well a hybrid system utilizing deep learning based post-processing filters will enhance the machine task performance of images reconstructed by the state-of-the-art traditional codec VVC. This was addressed in chapter 5 by proposing a hybrid system, which was used to train three different types of enhancement filters: Baseline Fidelity Enhancement (BFE), Task Specific Enhancement (TSE) and Task Agnostic enhancement (TAE). Here are listed the most important insights gained from the filters:

Task performance and generalization – All of the introduced filters outperformed the baseline (plain VVC) significantly. On the subset of the Open Images validation dataset referred as $\mathcal{X}_1$, the TSE filter achieved a 45.79% and 49.39% average BD-rate (mAP) gain over plain VVC on instance segmentation and object detection tasks, respectively. However, the TSE needs to be trained with a similar network as the validation task network is, in order to produce high gains over plain VVC. When TSE was evaluated with a completely different type of task network on a different validation set denoted as $\mathcal{X}_2$, it wasn't able to generalize anymore and achieved only 5.14% BD-rate (mAP) gain over VVC on object detection task with 80 classes. The BD-rate gains for the TAE filter were also over 40% for both segmentation and detection tasks on $\mathcal{X}_1$. Moreover, the TAE filter was able to generalize very well on the different validation dataset $\mathcal{X}_2$, achieving an average BD-rate (mAP) gain of 24.85% on object detection task with 80 classes. The reason for this is that the TAE doesn't use any task network in the training stage, making it task-agnostic. While on average the BFE filter on itself achieves the lowest BD-rate gains of the three, the BFE is crucial for the other filters to achieve the high performance, as it works as their training starting point. To sum it up, it is important to train a post-processing filter without using the target task network on the training phase, if the resulting filter is desired be task agnostic, such as the proposed TAE. On the other hand, if it is known that the target task network will always be the same, the information of this network can be used on the training phase. This results in a high performing but task specific filter, such as the TSE.

Filtering speed – Trained filters have only $782,663$ parameters resulting into filtering

speed of 77 frame-per-second on Nvidia Tesla V100 GPU for images that have maximum dimension of 1024 pixels. Since the filtering happens completely on the decoder side, this indicates that all the filters would be eligible for real-time applications.

Visual impact – All the filters achieve a significant PSNR gain [dB] over the plain VVC. More precisely, $+5.55$, $+4.26$ and $+5.35$ for BFE, TSE and TAE, respectively. However, most of these PSNR gains probably come from the correction of the distortions caused by the color conversion stages in the VVC pipeline. Thus, this aspect needs a further investigation to determine the root cause. Although, there certainly is also some subjective gain on the visual quality, meaning that the filtered images are suitable also for human consumption.

Overall, as seen from the experimental results, the proposed post-processing filters seem like an excellent solution for ICM and will definitely be a worthwhile topic for future studies. The most obvious aspect to improve, common to all NN based systems, would be to fine-tune all the hyperparameters and train the filters even further, in order to increase the performance. However, an important thing to note is that the training of a single QP took about a week on Nvidia Tesla V100 GPU. Due to this, the hyperparameter tuning with a similar dataset as used in this thesis will be computationally quite costly. In this regard, it was also stated that a LR scheduler would most likely result in a slightly better performance than the fixed LR of $1e{-}4$. Although, according to empirical tests, the reason to not use LR scheduler was that the extra epochs needed to train the filters yielded only a small performance improvement.

Finally, the most important aspect is that even though the filters were used for image data, they could easily be adapted to video data. When considering post filtering, VVC encoded video data is different than image data in a sense that it consists of frames which have some base QP and each frame is encoded with a QP that differs from this base value by some integer Δ. Thus, some modifications would be most likely needed in order to achieve similar results as with image data, but in the simplest solution the filters could be exactly the same as with the image case. There are lots of possibilities on how to further develop the system on this regard and it is definitely the next subject of research for the author of this thesis.

REFERENCES

[1] *Cisco annual internet report (2018–2023) white paper*, Accessed: Aug. 2021. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html (visited on 09/25/2021).

[2] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (hevc) standard", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, 2012. DOI: 10.1109/TCSVT.2012.2221191.

[3] B. Bross, Y.-K. Wang, Y. Ye, S. Liu, J. Chen, G. J. Sullivan, and J.-R. Ohm, "Overview of the versatile video coding (vvc) standard and its applications", *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 1–1, 2021. DOI: 10.1109/TCSVT.2021.3101953.

[4] J. Ascenso, "JPEG AI use cases and requirements", in *ISO/IEC JTC1/SC29/WG1 M90014*, Jan. 2021.

[5] "Call for evidence for video coding for machines", in *ISO/IEC JTC 1/SC29/WG 2, m55065*, Oct. 2020.

[6] J. I. Ahonen, R. Ghaznavi-Youvalari, N. Le, H. Zhang, F. Cricri, H. R. Tavakoli, M. M. Hannuksela, and E. Rahtu, "Learned enhancement filters for image coding for machines", in *23rd IEEE International Symposium on Multimedia*, IEEE, 2021.

[7] C. Shannon, "A mathematical theory of communication", *Bell System Technical Journal*, vol. 27, no. 4, pp. 623–656, 1948. DOI: 10.1002/j.1538-7305.1948.tb00917.x.

[8] D. A. Huffman, "A method for the construction of minimum-redundancy codes", *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952. DOI: 10.1109/JRPROC.1952.273898.

[9] J. Rissanen, "Generalized kraft inequality and arithmetic coding", *IBM J. Res. Dev.*, vol. 20, pp. 198–203, 1976.

[10] K. Sayood, *Introduction to data compression*, 4th ed. Waltham, MA: Morgan Kaufmann, 2012, ISBN: 9780124157965.

[11] N. Ahmed, T. Natarajan, and K. Rao, “Discrete cosine transform”, *IEEE Transactions on Computers*, vol. 23, pp. 90–93, 1974.

[12] I. Daubechies, “Orthonormal bases of compactly supported wavelets”, *Communications on Pure and Applied, Mathematics*, vol. 41, no. 7, pp. 909–996, 1988. DOI: 10.1002/cpa.3160410705.

[13] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra, “Overview of the h.264/avc video coding standard”, *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, 2003. DOI: 10.1109/TCSVT.2003.815165.

[14] M. Marcellin, M. Gormish, A. Bilgin, and M. Boliek, “Overview of jpeg-2000”, *Data Compression Conference Proceedings*, May 2000.

[15] B. Bross, J. Chen, J.-R. Ohm, G. J. Sullivan, and Y.-K. Wang, “Developments in international video coding standardization after avc, with an overview of versatile video coding (vvc)”, *Proceedings of the IEEE*, vol. 109, no. 9, pp. 1463–1493, 2021. DOI: 10.1109/JPROC.2020.3043399.

[16] J. Ahonen, *Robustness analysis of high-confidence image perception by deep neural networks*, 2020. [Online]. Available: https://trepo.tuni.fi/handle/10024/119585 (visited on 09/25/2021).

[17] V. Nair and G. Hinton, “Rectified linear units improve restricted boltzmann machines”, presented at the Proceedings of ICML, vol. 27, Jun. 16, 2010, pp. 807–814.

[18] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, “Activation functions: Comparison of trends in practice and research for deep learning”, *INCCST*, pp. 124–133, Dec. 2020.

[19] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”, in *2015 IEEE International Conference on Computer Vision (ICCV)*, vol. 2015, IEEE, 2015, pp. 1026–1034, ISBN: 1467383910.

[20] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[21] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization”, eng, in *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.

[22] L. Deng, “The mnist database of handwritten digit images for machine learning research”, *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[23] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting”, eng, *Journal of machine learning research*, vol. 15, pp. 1929–1958, 2014, ISSN: 1532-4435.

[24] *Tensorflow data augmentation*. [Online]. Available: https://www.tensorflow.org/tutorials/images/data_augmentation (visited on 09/25/2021).

[25] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition”, *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. DOI: 10.1109/5.726791.

[26] R. Girshick, “Fast R-CNN”, in *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015, pp. 1440–1448. DOI: 10.1109/ICCV.2015.169.

[27] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards real-time object detection with region proposal networks”, in *Advances in Neural Information Processing Systems 28*, Curran Associates, Inc., 2015, pp. 91–99. [Online]. Available: http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf (visited on 09/25/2021).

[28] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask R-CNN”, in *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017, pp. 2980–2988. DOI: 10.1109/ICCV.2017.322.

[29] S. Liu and W. Deng, “Very deep convolutional neural network based image classification using small training sample size”, in *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, Nov. 2015, pp. 730–734. DOI: 10.1109/ACPR.2015.7486599.

[30] *YOLOv5*. [Online]. Available: https://pytorch.org/hub/ultralytics_yolov5/ (visited on 09/25/2021).

[31] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition”, in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, ISSN: 1063-6919, Jun. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[32] L. Dong, Y. Gan, X. Mao, Y. Yang, and C. Shen, “Learning deep representations using convolutional auto-encoders with symmetric skip connections”, in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Apr. 2018, pp. 3006–3010. DOI: 10.1109/ICASSP.2018.8462085.

[33] R. Augustauskas and A. Lipnickas, “Improved pixel-level pavement-defect segmentation using a deep autoencoder”, eng, *Sensors (Basel, Switzerland)*, vol. 20, no. 9, pp. 2557–, 2020, ISSN: 1424-8220.

[34] L. Theis, W. Shi, A. Cunningham, and F. Huszár, “Lossy image compression with compressive autoencoders”, eng, in *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, 2017.

[35] D. Minnen, J. Ballé, and G. D. Toderici, *Joint autoregressive and hierarchical priors for learned image compression*, 2018.

[36] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston, “Variational image compression with a scale hyperprior”, eng, 2018.

[37] N. Le, H. Zhang, F. Cricri, R. Ghaznavi-Youvalari, and E. Rahtu, “Image coding for machines: An end-to-end learned approach”, in *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021, pp. 1590–1594. DOI: 10.1109/ICASSP39728.2021.9414465.

[38] N. Le, H. Zhang, F. Cricri, R. Ghaznavi-Youvalari, H. R. Tavakoli, and E. Rahtu, “Learned image coding for machines: A content-adaptive approach”, in *2021 IEEE International Conference on Multimedia and Expo (ICME)*, 2021, pp. 1–6. DOI: 10.1109/ICME51207.2021.9428224.

[39] J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders, “Selective search for object recognition”, eng, *International journal of computer vision*, vol. 104, no. 2, pp. 154–171, 2013, ISSN: 0920-5691.

[40] C.-Y. Wang, H.-Y. Mark Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh, and I.-H. Yeh, “Cspnet: A new backbone that can enhance learning capability of cnn”, eng, in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, vol. 2020-, 2020, pp. 1571–1580, ISBN: 1728193605.

[41] *Darknet*. [Online]. Available: https://github.com/pjreddie/darknet (visited on 09/25/2021).

[42] K. Wang, J. H. Liew, Y. Zou, D. Zhou, and J. Feng, “Panet: Few-shot image semantic segmentation with prototype alignment”, eng, in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, vol. 2019-, IEEE, 2019, pp. 9196–9205, ISBN: 9781728148038.

[43] R. Xu, H. Lin, K. Lu, L. Cao, and Y. Liu, “A forest fire detection system based on ensemble learning”, *Forests*, vol. 12, p. 217, Feb. 2021. DOI: 10.3390/f12020217.

[44] S. Yohanandan. (Jul. 10, 2020). “Map (mean average precision) might confuse you!”, [Online]. Available: https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2 (visited on 09/25/2021).

[45] T. Arlen. (Jan. 3, 2020). “Understanding the map evaluation metric for object detection”, [Online]. Available: https://medium.com/@timothycarlen/understanding-the-map-evaluation-metric-for-object-detection-a07fe6962cf3 (visited on 09/25/2021).

[46] J. Hui. (Mar. 7, 2018). “Map-mean-average-precision-for-object-detection”, [Online]. Available: https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173 (visited on 09/25/2021).

[47] (2017). “Coco evaluation metrics”, [Online]. Available: https://cocodataset.org/#detection-eval (visited on 09/25/2021).

[48] G. Bjøntegaard, “Calculation of average PSNR differences between RD-curves”, *ITU-T Video Coding Experts Group (VCEG)*, Apr. 2001. (visited on 09/25/2021).

[49] M. Rafie, Y. Zhang, and S. Liu, “Evaluation framework for video coding for machines”, *ISO/IEC JTC 1/SC 29/WG 2, MPEG Technical requirements, Document: N104*, Jul. 2021.

[50] K. Fischer, F. Brand, C. Herglotz, and A. Kaup, “Video coding for machines with feature-based rate-distortion optimization”, *IEEE 22nd International Workshop on Multimedia Signal Processing*, p. 6, Sep. 2020.

[51] B. Brummer and C. de Vleeschouwer, “Adapting JPEG XS gains and priorities to tasks and contents”, in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Seattle, WA, USA, Jun. 2020, pp. 629–633, ISBN: 978-1-72819-360-1. DOI: 10.1109/CVPRW50498.2020.00090. (visited on 09/25/2021).

[52] Z. Wang, R.-L. Liao, and Y. Ye, “Joint learned and traditional video compression for p frame”, in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Seattle, WA, USA, Jun. 2020, pp. 560–564, ISBN: 978-1-72819-360-1. DOI: 10.1109/CVPRW50498.2020.00075. (visited on 09/25/2021).

[53] F. Zhang, C. Feng, and D. R. Bull, “Enhancing vvc through cnn-based post-processing”, in *2020 IEEE International Conference on Multimedia and Expo (ICME)*, 2020, pp. 1–6. DOI: 10.1109/ICME46284.2020.9102912.

[54] M. Wang, S. Wan, H. Gong, Y. Yu, and Y. Liu, “An integrated cnn-based post processing filter for intra frame in versatile video coding”, in *2019 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 2019, pp. 1573–1577. DOI: 10.1109/APSIPAASC47483.2019.9023240.

[55] *FFmpeg software*. [Online]. Available: https://github.com/FFmpeg/FFmpeg (visited on 09/25/2021).

[56] *Versatile video coding (VVC) reference software VTM-8.2*. [Online]. Available: https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM (visited on 09/25/2021).

[57] F. Brossen, J. Boyce, K. Suehring, X. Li, and V. Seregin, "JVET common test conditions and software reference configurations for sdr video", *Joint Video Experts Team (JVET), Document: JVET-N1010*, Mar. 2019.

[58] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet large scale visual recognition challenge", *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015. DOI: 10.1007/s11263-015-0816-y.

[59] *Open Images V6*. [Online]. Available: https://storage.googleapis.com/openimages/web/index.html (visited on 09/25/2021).

[60] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft COCO: Common objects in context", *arXiv:1405.0312 [cs]*, Feb. 20, 2015. arXiv: 1405.0312. (visited on 09/25/2021).

[61] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, *Detectron2*, https://github.com/facebookresearch/detectron2, 2019.

[62] *OpenImages evaluation*. [Online]. Available: https://storage.googleapis.com/openimages/web/evaluation.html (visited on 09/25/2021).

[63] *Python3*. [Online]. Available: https://www.python.org/ (visited on 09/25/2021).

[64] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library", in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.