UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
E scola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

Semantic Segmentation in the Latent Representation

Master Thesis submitted to the Faculty of the
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya
by

Artur Díaz Juan

In partial fulfillment
of the requirements for the master in
Advanced Telecommunication Technologies

Supervised by: Michela Testolina
Changsheng Gao
Prof. Dr. Touradj Ebrahimi
Prof. Dr. Philippe Salembier

Lausanne, September 2022

Multimedia Signal Processing Group
EPFL

Abstract

This project proposes a method to merge image compression and semantic segmentation, in a single stage, for the foreground/background segmentation approach. This binary segmentation is based on the case of person segmentation, whereby the foreground corresponds to each person in the image and the background is everything else.
The proposed method is analysed and compared with an end-to-end compression model followed by a semantic segmentation stage, as well as with the results obtained from uncompressed image segmentation.
The CompressAI cheng2020-anchor model and the Pyramid Scene Parsing Network (PSP-Net), implemented through the Semseg repository, have been used to develop this approach.
The results obtained by the proposed merging method outperform those obtained by the end-to-end compression model followed by the semantic segmentation stage for low bitrates.

Contents

List of Figures

List of Tables

1 Introduction

Deep-learning-based image compression algorithms have demonstrated excellent performance over the last few years, in terms of compression efficiency and perceived visual quality.
These types of algorithms are based on the autoencoder approach [6]. An autoencoder is a special type of neural network that is trained to reproduce its input into its output. It is made up of two distinct blocks, the encoder, in which the input image is mapped to a compact representation in latent space, and the decoder, in which this latent representation is reconstructed as similar as possible to the original image.
Computer vision methods are often applied to images reconstructed in the pixel domain. Some recent research has presented the possibility of applying it to the latent space, before decoding, reducing the computational cost and achieving the same performance in accuracy [7].

The main objective of this project is to obtain a method capable of using a semantic segmentation network directly in the latent domain, obtained from the compression network, and compare it with the case of reconstruction in the pixel domain.
Specifically, the project focuses on the foreground/background segmentation approach, a case of binary semantic segmentation. As the focus will be on people segmentation, the foreground label will be used for people and everything else will be treated as background. The idea is to segment people from their scenes using the latent code obtained from the original image encoding.

The different tasks defined for the project to achieve the main objective are as follows:

1. Study the state of the art of deep-learning-based image compression followed by identification of the best network to use during the project.
2. Study the state of the art of deep-learning-based image semantic segmentation followed by identification of the best network to use during the project.
3. Collect a suitable dataset for experiments.
4. Propose and implement a solution by merging the image encoder with the semantic segmentation network and training it. The use case of foreground/background segmentation should be addressed.
5. Evaluate the above and report the results.

This report is divided into different sections. First, a review of the state of the art is given in Section 2. Section 3 explains the dataset used. Next, the evaluation metrics used for image quality and semantic segmentation are presented in Section 4. The following one presents the compression and semantic segmentation networks used in this project. Section 6 introduces the approaches proposed during the development of the project, followed by the experiments and results obtained for the presented approaches in Section 7. Finally, the conclusions of this study and future work are discussed.

2 Related work

2.1 Compression methods

Deep learning-based compression methods have been used in recent years to outperform traditional methods, such as JPEG2000 [8] or BPG [9], in terms of perceptual metrics [10]. These methods are based on encoding an input image into a feature map (latent representation), which is quantified into a set of symbols through a compressed bitstream and then decoded to reconstruct the input image.

In [10], the authors summarise a set of end-to-end learned image compression methods, highlighting the main contributions of each new method and providing a detailed comparative analysis of them. Furthermore, they show a benchmark evaluation of the presented methods through comparative results in both PSNR [11] and MS-SSIM [12]. As each method is sorted according to the date of publication, it is easier to identify the most recent contributions to the field. Some of the papers reviewed with the most important contributions to the development of these deep learning compression methods are [13], [14] and [4].

In [13], the authors present an end-to-end trainable model for image compression based on [15] that uses nonlinear transform coding and describe a framework for optimising end-to-end rate-distortion performance.
A new contribution of [13] is the introduction of a hyperprior on the local scale parameters of the latent representation. This leads to a more powerful variational image compression model. The authors extend the model presented in [15], based on a fully factored prior, with a hyperprior that captures how spatially neighbouring elements of the latent representation change together in their scales and is added as side information. The models presented in both [13] and [15] make use of Generalised Divisive Normalisation (GDN) layers, a type of non-linearity that implements local normalisation and has been shown to be more efficient at removing statistical dependencies in image data.

Following previous work, [14] extends the Gaussian Scale Mixture (GSM) entropy model in [13] to a Gaussian Mixture Model (GMM), inspired by generative models, and adds an autoregressive component. This generates a mean and a scale parameter conditioned on the hyperprior. It is shown that the GMM-based entropy model outperforms the GSM-based model in terms of rate-distortion performance. The improvement of the entropy model leads to a requirement for more context information. Therefore, the authors introduce an autoregressive component that does not cause a potential rate penalty, as the predictions are based only on causal context. Thus, the hyperprior can learn to preserve the information needed to reduce uncertainty in the autoregressive model, while avoiding data that can be accurately predicted from the context.

Focusing on [4], the authors propose to use discretised Gaussian Mixture Likelihoods to parameterise the latent code distributions, achieving a more accurate and flexible entropy model. In addition, they incorporate attention modules to the network architecture in order to improve performance and achieve high coding efficiency. Parameterised distributions cannot represent the use of context, side information and extra bits, which might be limited by the fixed shape of the single Gaussian distribution. For this reason, the authors consider a more flexible parameterised model to achieve arbitrary likelihoods. The visual quality of the models presented in the article outperforms existing methods in terms of MS-SSIM.

One of the most recent research in this field [16], proposes an improvement of the last paper. The authors realise that the strongest correlations do not always appear in neighbouring channels,

so they propose to split the latents into several groups depending on the channel index and to code the groups one by one. In this way, the currently encoded group can leverage the context information of the previous ones to efficiently capture the correlation between channels.
This method provides better performance than the latest video coding standard VVC [17] when measured with PSNR distortion, as well as more visually pleasing reconstructed results when trained with MS-SSIM.

2.2 Semantic segmentation methods

When talking about the semantic segmentation task, most of the recent research is based on two different networks, DeepLab and PSPNet [5].

On the one hand, DeepLab has proposed many contributions to the state of the art in recent years, DeepLabv1 [18], v2 [19] and v3 [20].
The most recent release, DeepLabv3, significantly improves on previous versions and achieves comparable performance with other state-of-the-art models in the PASCAL VOC 2012 semantic image segmentation benchmark. The model proposes an atrous convolution with upsampled filters to extract dense feature maps and capture context in multiple ranges, in the framework of both cascaded modules and spatial pyramid pooling.

On the other hand, another well-known method for semantic segmentation is the Pyramid Scene Parsing Network (PSPNet) [5], which performs spatial pooling at several grid scales and provides excellent results in many semantic segmentation benchmarks. The authors exploit the capability of global context information by aggregating context based on different regions, providing additional contextual information.

Following previous work, other literature, such as [21], explores the impact of global context information. Which captures the semantic context of scenes and selectively highlights class-dependent feature maps. Moreover, they discuss how the proposed module improves the feature representation of relatively shallow network features. Experimental results show that this contribution improves network feature representations at an early stage using a global context, which is difficult to learn for a standard network architecture consisting only of convolutional layers, non-linearities and downsamplings.

Closing the chapter on semantic segmentation, a new architecture called ShelfNet [22] presents a structure with many encoder-decoder blocks with skip connections at each spatial level, leading to accurate fast semantic segmentation results. Compared to non-real-time methods, such as PSPNet, it achieves four times faster inference speed while maintaining similar accuracy.

2.3 Computer vision tasks in the compressed domain

The baseline for this project is the paper *Towards Image Understanding from Deep Compression without Decoding* [7]. The authors present two different computer vision tasks from compressed image representation, which are image classification and semantic segmentation. For the compression network, they use the image compression autoencoder described in [23] and a modification of ResNet [24] and DeepLabv2 [19] for the classification and segmentation tasks. The results presented are optimal in terms of compressed domain performance against image reconstruction. Following the good results obtained in the semantic segmentation task within the compressed domain, this work wants to reproduce it to the human segmentation task presented in the introduction.

3 Dataset

A human segmentation dataset has been used in order to train the foreground/background segmentation model. The dataset selected for this task is the Person Dataset of Supervisely which contains many images of people in different scenarios.

3.1 Supervisely tool

Supervisely [25] is a powerful platform for computer vision development, where individual researchers and large teams can annotate and experiment with datasets and neural networks.
It allows to:

- Label images, videos, 3D point clouds, volumetric slices and other data in the best labelling tool.
- Manage and track annotation workflow at scale with teams, workspaces, roles and labelling jobs.
- Train and apply neural networks to custom data.
- Explore data and automate common tasks with integrated Python notebooks and scripts.

3.1.1 Supervisely Person Dataset

The Supervisely Person dataset is made freely available to academic and non-academic entities for non-commercial purposes such as academic research, teaching, scientific publications or personal experimentation. The full version of the tagged person dataset contains 5711 images with annotated person instances. For this project, a filtered version of the original dataset is used in which images with low quality labels have been removed. Thus, the dataset used consists of a total of 2667 images, divided into 2533 for the training set, 80 for the validation set and 53 for the test set. This partitioning has been done empirically to divide tests that do not share the same types of images in order to make them as independent as possible and to maximise learning from the models. Each RGB image in the dataset has a corresponding binary mask of the profile of the person(s).

Figure 1: Examples from the Supervisely Person Dataset in [1].

The dataset contains images in different conditions, such as indoor and outdoor images, images with a diversity of light conditions, close-up and long-distance views, portraits, and images with

multiple people together and separated, and also includes some grey level images in addition to the colour images. This varied information adds value to the dataset as it can be useful for training models for different scenario variants. All images in the dataset have a resolution of between 377x800 and 6200x800, and vary only in width, while the height is the same for each image. The most common images in the dataset are 1200x800. All images are in Portable Network Graphics (PNG) format [26].
Some examples of images from the dataset are shown in Figure 1.

4 Assessment metrics

To evaluate the performance of the methods used during the project, different metrics are taken into account. On the one hand, objective image quality metrics such as PSNR and MS-SSIM would indicate the performance of the compression methods.
On the other hand, evaluation metrics for semantic segmentation such as Intersection over Union and Pixel Accuracy would provide information about the performance at the image segmentation stage.

4.1 Objective image quality assessment

The objective of image quality assessment is to obtain quality measures that can be used to assess the performance of image processing systems.
This subsection describes the two metrics used in the project: PSNR and MS-SSIM.

4.1.1 PSNR

The Peak Signal to Noise Ratio (PSNR) (1) is a rate between the maximum achievable value (power) of a signal and the power of the distorting noise that affects the quality of its representation [11]. It is expressed in terms of a logarithmic decibel scale due to its wide dynamic range. The PSNR value approaches infinity as the MSE (2) approaches zero, this proves that a higher PSNR value provides higher image quality. In contrast, a small PSNR value implies high numerical differences between the compared images.
The mathematical expression for the PSNR is:

$$PSNR(x,y) = 10 * log_{10}(\frac{255^2}{MSE(x,y)}) \tag{1}$$

where

$$MSE(x,y) = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}(x_{ij} - y_{ij})^2 \tag{2}$$

Considering x as the reference image and y the reconstructed one, both of size $M \times N$.

The main problem with this metric is that it is only based on a numerical comparison and does not take into account any level of the biological factors of the human visual system, as does the structural similarity index (MS-SSIM).

4.1.2 MS-SSIM

The structural similarity image quality paradigm is based on the assumption that the human visual system is highly adapted for extracting structural information from the scene, and therefore a measure of structural similarity can provide a good approximation to perceived image quality [12].

Using SSIM [27] as a basis, MS-SSIM extends the technique by making multiple SSIM image evaluations at different image scales, with a total of M as seen in (3). MS-SSIM is performed at multiple scales through a multi-step downsampling process, similar to the multiscale processing of the primitive visual system. With MS-SSIM, the scale of the images becomes less important than in SSIM, up to the point that images that have been downsampled or upsampled can still be compared objectively. A high score expresses better image quality.

$$MS\text{-}SSIM(x,y) = l_M(x,y) \prod_{j=1}^{M} c_j(x,y) s_j(x,y) \tag{3}$$

where

$$l(x,y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{4}$$

$$c(x,y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{5}$$

$$s(x,y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} \tag{6}$$

where μ_x and μ_y are the sample means of x and y respectively, σ_x^2 and σ_y^2 are the sample variances of x and y respectively, and $\sigma_x\sigma_y$ is the sample correlation coefficient between x and y. The small constants C_1, C_2, C_3 are used to avoid instability when the denominator might approach zero. Equation (4) defines the function for the luminance comparison of the signals, (5) the contrast comparison of the signals, and (6) the structure comparison of the signals.

4.2 Semantic segmentation evaluation

To assess the quality of the semantic segmentation on the selected dataset, both IoU and pixel accuracy are calculated. These two metrics are described below.

4.2.1 IoU

Intersection Over Union (IoU) (7), also known as Jaccard's index, is one of the most popular metrics for the evaluation of semantic segmentation.
This metric measures the intersection between the real and predicted mask (pixels in common), divided by the union of both (total pixels). It basically quantifies the percentage of overlap between the target mask and the predicted output. It is described by the following formula:

$$IoU = \frac{|A \cap B|}{|A \cup B|} \tag{7}$$

The IoU score is calculated for each class separately and then averaged across all classes (k) to obtain an overall mean IoU (mIoU) (8) score of the semantic segmentation prediction.

$$mIoU = \frac{1}{k} \sum_{i=1}^{k} \frac{|A \cap B|_i}{|A \cup B|_i} \tag{8}$$

4.2.2 Pixel accuracy

Overall pixel accuracy (9) evaluates the percentage of pixels in the images correctly classified for each class [28].
This metric evaluates the percentage of pixels in the images classified correctly for each class. This metric calculates the ratio between the number of pixels classified correctly and the total number of pixels in the image as:

$$allAcc = \frac{\sum_{j=1}^{k} n_{jj}}{\sum_{j=1}^{k} t_j} \tag{9}$$

where n_{jj} is the total number of pixels both classified and labeled as class j. It corresponds, otherwise, to the total number of true positives for class j. t_j is the total number of pixels labelled as class j.

Overall pixel accuracy can sometimes give erroneous results due to unbalanced classes, as some of them may have a small representation within the image and the measurement will be biased, obtaining good results due to the effect of over-represented classes.

Since there are multiple classes present in the semantic segmentation, two in the case of this project, and to give equal weight to each image not depending on its contribution to the total number of pixels, the mean pixel precision (10) represents the average class precision as:

$$mAcc = \frac{1}{k}\sum_{j=1}^{k}\frac{n_{jj}}{t_j} \tag{10}$$

Being overall pixel accuracy (allAcc) and mean pixel accuracy (mAcc) intuitive and interpretable metrics.

5 Assessment models

5.1 Compression model

To implement an end-to-end compression model, it is necessary to apply a framework compatible with this type of model. Most of the models presented in the related work chapter are available for Python on the open-source platforms Tensorflow [29] and Pytorch [30]. For this project, the implementation is done in Pytorch. Thus, a well-known library for this framework that implements most of the compression algorithms seen is the CompressAI [2].

5.1.1 CompressAI

As presented by the authors in [2], CompressAI is a platform that provides customised operations, layers, models and tools to investigate, develop and evaluate end-to-end image and video compression codecs. In particular, CompressAI includes pre-trained models and evaluation tools to compare learned methods with traditional codecs. Multiple state-of-the-art end-to-end compression models learned and trained from scratch have been re-implemented in PyTorch. This library currently implements models for still image compression, which fits well with the development of this project.

The list of available pre-trained models is as follows. Only models optimised for the mean square error (2) calculated on the RGB channels are provided. These models have been pre-trained on the Vimeo90K dataset [31].

- *bmshj2018-factorized*: factorized prior model from [13].
- *bmshj2018-hyperprior*: scale hyperprior model from [13].
- *mbt2018-mean*: scale hyperprior with non-zero-mean Gaussian conditionals from [14].
- *mbt2018*: joint autoregressive hierarchical priors model from [14].
- *cheng2020-anchor*: anchor model variant from [4].
- *cheng2020-attn*: self-attention model variant from [4].

The pre-trained weights of the *cheng2020-attn* model are not available at the moment, so this model cannot be considered an option for the project.
The quality parameter of the architectures models the target bit rate for compression. All models set a scale from 1 to 8, except *cheng2020-anchor* which is only trained on the first six indices. According to the information provided by CompressAI, the library implementations achieve the same results in terms of PSNR and MS-SSIM as those presented in the original paper.

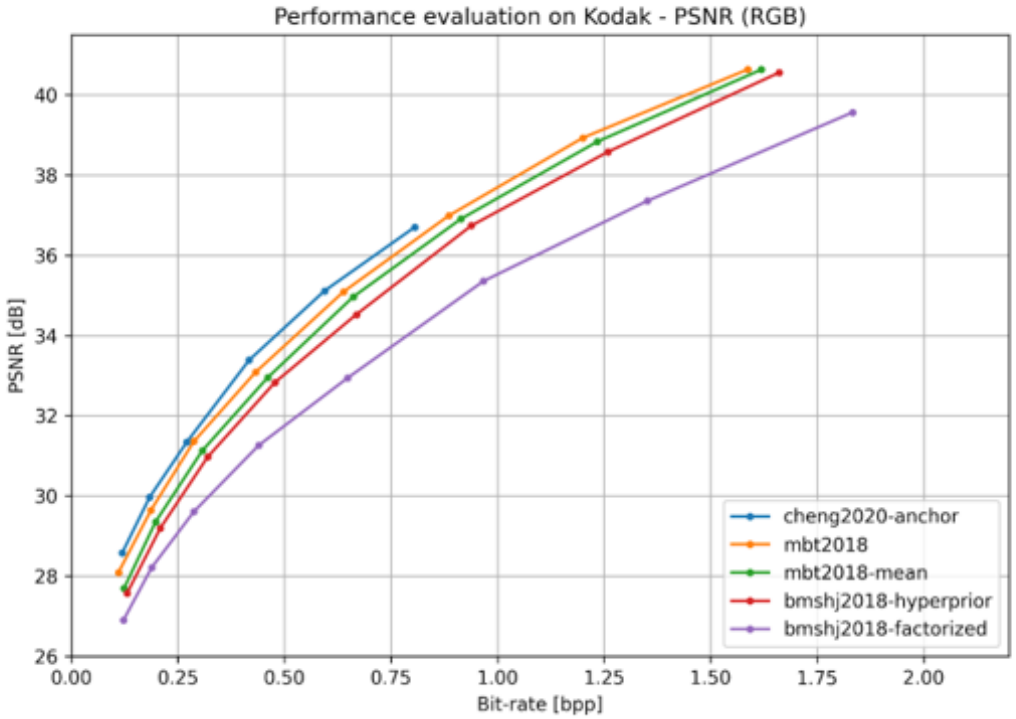

Figure 2: PSNR evaluation on Kodak dataset extracted from CompressAI [2]

.

In the official publication of CompressAI [2], the authors compare all the implemented models and other traditional ones. Figure 2 is an example of the PSNR performance of the recent pre-trained models as a function of the bit rate achieved in the compression for the Kodak dataset. Other results with different datasets and comparisons with traditional models are also presented throughout the article. The *cheng2020-anchor* model only appears for low bit rates for the reason explained above. Despite this fact, it is the model that achieves the best performance for lower bit rates.

5.1.2 Model comparison

In order to verify the results presented in [2] and to select the appropriate model for the image compression stage, images extracted from the test set of the JPEGAI repository [3] have been analysed in terms of PSNR and MS-SSIM. The images of the set used to evaluate the compression methods are shown in Figure 3.

Figure 3: JPEGAI test set [3].

To analyse the performance of each compression method implemented, each image of the JPEGAI test set is passed through each model, presented in 5.1.1, applying compression at different bit rates. After having reconstructed the input image for each model, both PSNR and MS-SSIM are computed for each model, this step is repeated one by one through each image in the set. Finally, the results obtained for each image at similar bit rates are averaged to obtain the graph described in Figure 4 and to obtain the results that will serve as a reference to select the optimal model to be used in the project.

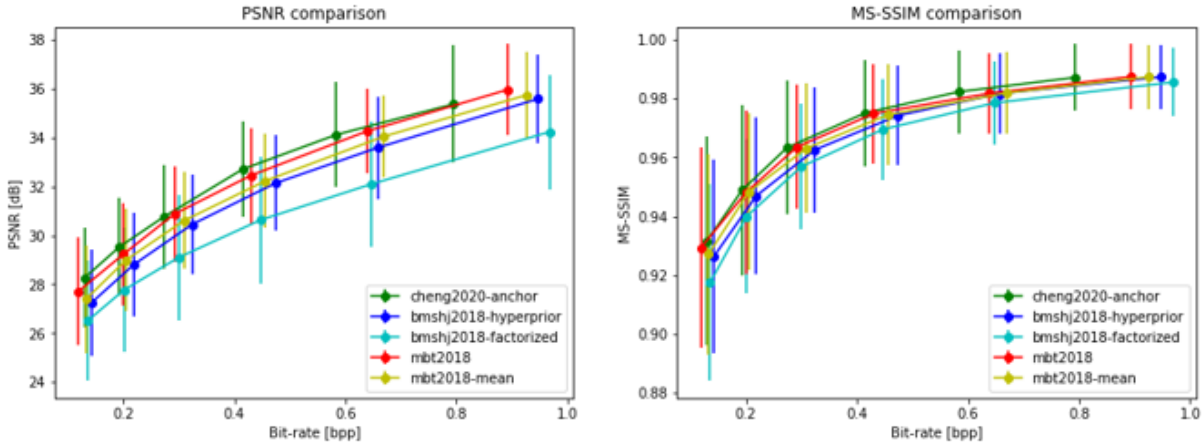

Figure 4: Left: PSNR comparison for the JPEGAI test set. Right: MS-SSIM comparison for the JPEGAI test set.

In the bit rate domain, for both PSNR and MS-SSIM metrics, *cheng-2020* (green line) performs better in both, achieving better values at lower bit rates compared to the other models in a similar bit rate range.

Therefore, this study on the JPEGAI test dataset [3] corroborates the results published by the authors of CompressAI, where for low bit rates *cheng-2020* performs slightly better, as shown in Figure 2 for CompressAI and in Figure 4 left in our experiment, both representations for the PSNR scenario. These results are then ratified, in addition to the fact that the model also performs better in terms of visual perception, which we affirm through the MS-SSIM results in the graph in Figure 4 right.
The vertical lines at each point in both charts in 4 show the standard deviation within the set of images averaged to obtain the results.

5.1.3 Selected model

The chosen model is the anchor model variant from *Learned Image Compression with Discretized Gaussian Mixture Likelihoods and Attention Modules* [4].
This model presents discretized Gaussian Mixture Likelihoods to parameterize the distributions of latent codes, a more accurate and flexible entropy model.

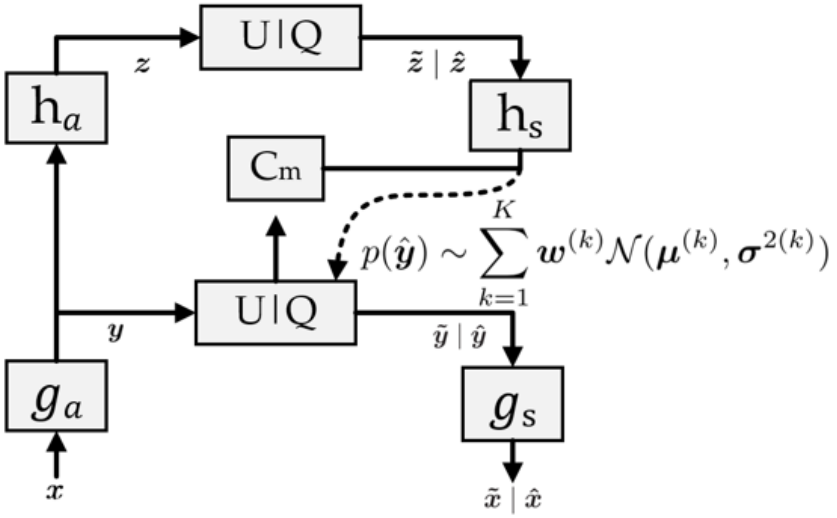

Figure 5: Proposed anchor model from Cheng et al. in [4].

From Figure 5, the image coding and decoding process can be formulated by:

$$y = g_a(x; \phi) \tag{11}$$

$$\hat{y} = Q(y) \tag{12}$$

$$\hat{x} = g_s(\hat{y}; \Theta) \tag{13}$$

where x, $\hat{x}$, y and $\hat{y}$ are raw images, reconstructed images (13), latent representation before quantization (11) (continuous) and compressed codes (12) respectively. ϕ and Θ are optimized parameters of analysis and synthesis transforms. $U|Q$ is the quantization and entropy coding to generate the bitstream.

$$z = h_a(y; \phi_y) \tag{14}$$

$$\hat{z} = Q(z) \tag{15}$$

$$p_{\hat{y}|\hat{z}}(\hat{y}|\hat{z}) \leftarrow h_s(\hat{z}; \Theta_h) \tag{16}$$

h_a and h_s denote the analysis (14) and synthesis (15) transforms in the auxiliary autoencoder, where ϕ_h and Θ_h are optimized parameters. $p_{\hat{y}|\hat{z}}(\hat{y}|\hat{z})$ are estimated distributions conditioned on $\hat{z}$ (16).

During training, quantization is approximated by a uniform noise to generate noisy codes $\tilde{y}$. During inference, $U|Q$ represents the real round-based quantization to generate $\hat{y}$ and followed entropy coders to generate the bitstream.
It is proposed a Gaussian Mixture Model using discretised Gaussian mixture likelihoods:

$$p_{\hat{y}|\hat{z}}(\hat{y}|\hat{z}) \sim \sum_{k=1}^{K} w^k N(\mu^k, \sigma^{(2k)}) \tag{17}$$

Each mixture in (17) is characterized by a Gaussian distribution with 3 parameters: weights $w_i^{(k)}$, means $\mu_i^{(k)}$ and variances $\sigma_i^{2(k)}$ for each element $\hat{y}_i$. C_m is the cumulative function of each mixture.
The Gaussian mixture model can achieve better performance because it selects K's most probable values and assigns a small scale (i.e. high likelihoods) to them.

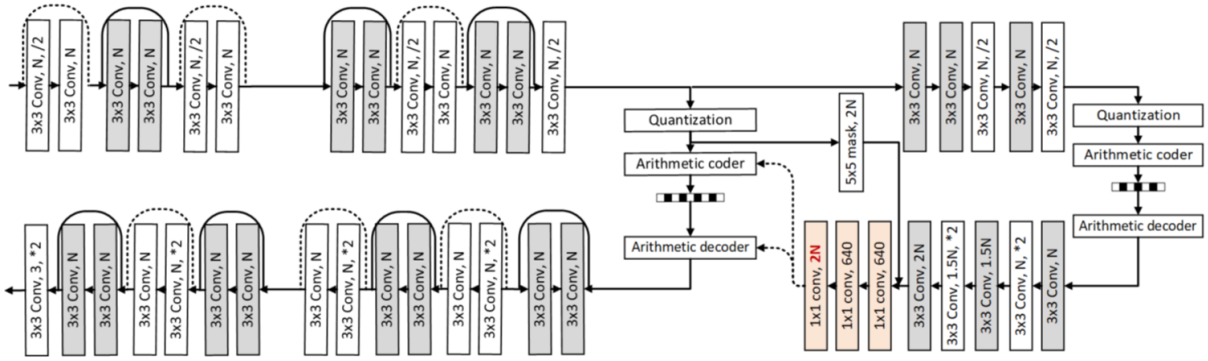

Figure 6: Architecture of the anchor model from [4].

The network architecture of the anchor model represented in Figure 6 uses four 3×3 kernels with residual connection for each downsampling operation and use subpixel convolution in synthesis transform. The number of filters N is equal to 128. There are used residual blocks to increase large receptive field and improve the rate-distortion performance. Decoder side uses subpixel convolution instead of transposed convolution as upsampling units to keep more details. N denotes the number of channels and represents the model capacity.

5.2 Semantic segmentation model

As mentioned in 2.2, the most common networks for the semantic segmentation case are those of DeepLab and PSPNet. All of them are implemented in Tensorflow, but only PSPNet is available for Pytorch in open source, which is necessary to make the necessary modifications to finally integrate the compression stage into the semantic segmentation network. Therefore, PSPNet is the network selected to be used in the project through the Semseg repository, which are explained as follows.

5.2.1 Pyramid Scene Parsing Network

Pyramid Scene Parsing Network (PSPNet) [5], through the pyramid pooling module, exploits the capability of global context information by different region-based context aggregation. In addition to traditional dilated Fully Convolutional Network (FCN), it extended the pixel-level feature to the designed global pyramid pooling module. The local and global clues together make the final prediction more reliable.

PSPNet is designed to improve performance for open-vocabulary object and stuff identification in complex scene parsing.

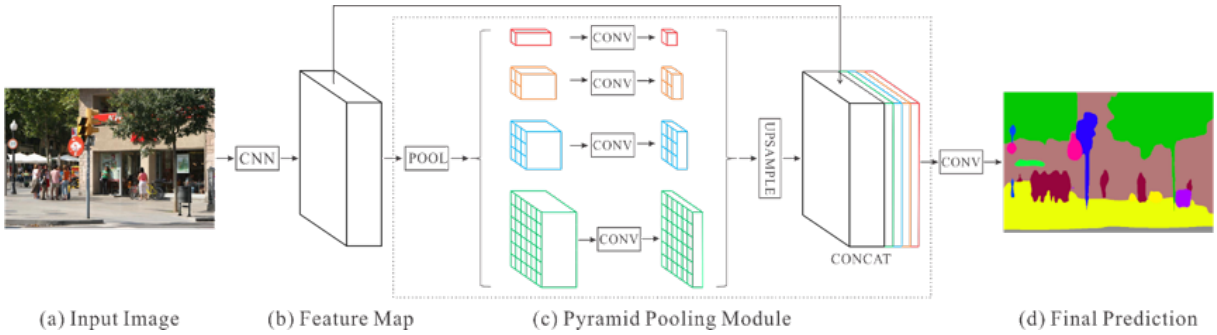

Figure 7: PSPNet architecture from [5].

A hierarchical global prior is proposed in [5] to further reduce context information loss between sub-regions, containing information with different scales and varying among sub-regions. The pyramid pooling module fuses features under four different pyramid scales. The coarsest level highlighted in red in Figure 7 is global pooling to generate a single bin output. The following pyramid level separates the feature map into different sub-regions and forms pooled representation for different locations. The output of different levels in the pyramid pooling module contains the feature map with different sizes. To keep the weight of global feature, it is used a 1x1 convolution layer after each pyramid level to reduce the dimension of context representation to $\frac{1}{N}$ of the original one if the level size of pyramid is N. Then it is directly upsampled the low-dimension feature maps to get the same size feature as the original feature map via bilinear interpolation. Finally, different levels of features are concatenated as the final pyramid pooling global feature.

A pre-trained ResNet is used as a backbone to generate the input feature map whose size is $\frac{1}{8}$ of the input image. Then, the pyramid pooling module is used to gather context information. The four pyramid levels are fused as the global prior and concatenated to the original feature map. Finally, a fully connected layer is added to generate the final prediction output.

As increasing the depth of the network could lead to difficulties in optimization, two different losses are used for the training step. An auxiliary loss will generate initial results by supervision and the final one will learn the residue. The auxiliary loss helps optimize the learning process, while the master branch loss takes the most responsibility. Finally, the auxiliary loss is added to the final one by a weighted factor.

Figure 7 depicts all the above steps, where (a) is the input RGB image for the complete semantic segmentation architecture, (b) is the above mentioned ResNet backbone that produces the feature map from the input image, (c) is the complete pyramidal pooling module containing all the convolution layers at different levels, the upsampling of them and the final concatenation prior to the final fully connected layer in (d) to produce the output prediction.

5.2.2 Semseg repository

The Semseg repository [32] is a PyTorch implementation for semantic segmentation/scene parsing. The code is easy to use for training and testing on various datasets. The codebase mainly uses ResNet50/101/152 as a backbone and can be easily adapted to other basic classification structures. Implemented networks including PSPNet and PSANet, which ranked 1st place in ImageNet Scene Parsing Challenge 2016 @ECCV16, LSUN Semantic Segmentation Challenge

2017 @CVPR17 and WAD Drivable Area Segmentation Challenge 2018 @CVPR18. Sample experimented datasets are ADE20K, PASCAL VOC 2012 and Cityscapes.

The original repository is adapted to run the PSPNet for the Supervisely Person Dataset required for the project. In addition to being able to adapt the parameters in terms of the necessary hardware performance, as it is designed to train with eight GeForce RTX 2080 Ti.

6 Proposed methods

After having analysed the options for the compression and semantic segmentation stages, a couple of approaches are proposed to compare how well people are finally segmented in the presented dataset after compressing the original images.
For this comparison, a baseline is defined as the original anchor, which will correspond to the results obtained by the semantic segmentation network in the pixel domain from the original images of the uncompressed Supervisely Person Dataset.
For the first approach, it is proposed to separately perform the compression and segmentation stages. First, image compression and reconstruction is performed for different bitrates, after which the obtained reconstructed images will serve as input for the semantic segmentation network. Thus, it will be an end-to-end compression plus segmentation which will be referred to as a decoded anchor.
The second approach consists in integrating the image compression stage into the semantic segmentation network. In this case, the model input will be the original image, and the compression stage is performed within the backbone of the Pyramidal Scene Parsing Network. Therefore, the semantic segmentation network will learn directly from the latent domain instead of having to reconstruct the encoded image at a prior stage. This second approach is called latent code approach.

6.1 Baseline

The baseline model is trained directly from the Supervisely Person Dataset images, taking into account only the semantic segmentation step without having compression losses in the images.
For this training, all training images of the dataset are resized to 768x768 due to the requirements of the *Semseg* implementation, and the model is trained for 200 epochs with a batch size of 4, using a single GPU. The remaining configuration is the standard one used in the *Semseg* repository.

From this baseline model, the original anchor is obtained using the uncompressed images in the inference step. This will allow obtaining the metrics of the original images and will be useful to later analyse how the information lost in the compression stage affects the semantic segmentation prediction.

6.2 Decoded anchor

This first approach consists of applying, in the inference step, an end-to-end image compression stage followed by the semantic segmentation stage, applied to the model trained for the baseline.
The pre-trained CompressAI *cheng-2020* model is used. The original images are resized to 768x768 as required by the model and to recreate the same training conditions.
The compression bit rate is defined by the quality factor of the model, which ranges from 1 (lowest bit rate, highest compression) to 6 (highest bit rate, lowest compression). Applying this quality factor, each image in the dataset is encoded and decoded following the diagram in 5.1.3, $\hat{x}$ is obtained corresponding to the reconstructed image for each quality case.
With all the reconstructed images for the six compression levels, the semantic segmentation stage is performed through the Pytorch implementation of the Pyramid Scene Parsing Network (*Semseg*).
With the original structure presented in 5.2.1, the reconstructed image is the input for the pre-trained ResNet-50 network used as a backbone to obtain the feature map, as seen in Figure 5, that will act as input for the Pyramid Pooling module. After the appropriate concatenation and

final layers, the segmented image map is obtained for every input image at different compression levels.

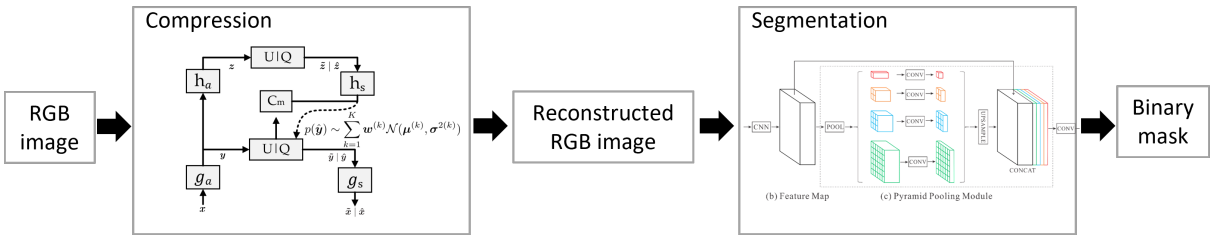

Figure 8: Diagram of the decoded anchor approach in which compression and semantic segmentation are performed in different stages. The diagrams from Compression and Segmentation are obtained from [4] and [5] respectively.

6.3 Latent code approach

The aim for this second approach is to integrate the compression stage within the semantic segmentation network, avoiding the decoding step in the compression model. To implement this architecture, a new model has to be trained for each compression level (1-6) with corresponding modifications in the network architecture.

Following the diagram presented in 5.1.3, the compression network will only be applied until the quantified latent $\hat{y}$ is obtained, the rest of the structure is ignored as it corresponds to the decoding step. The use of the continuous, prior quantization, latent y to integrate it directly into the semantic segmentation network has been discarded as models trained with it perform worse than models trained with $\hat{y}$. This is because the CompressAI pre-trained compression network has been trained taking into account the losses of $\hat{y}$. Figure 9 illustrates the explained modifications to the original scheme. The red cross is striking out the decoding step, then the corresponding output is added to the segmentation stage.

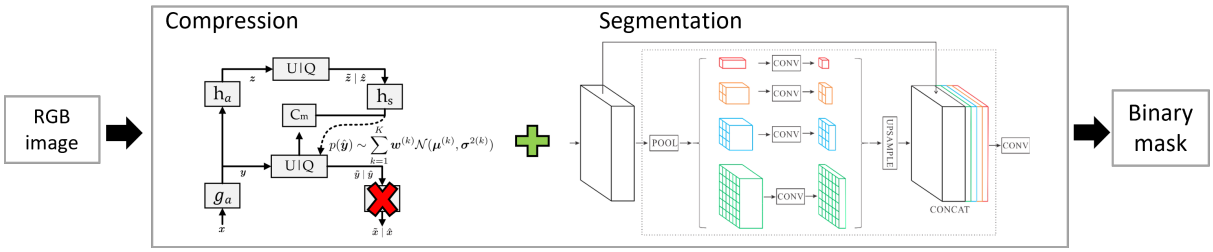

Figure 9: Diagram of the latent code approach in which compression and semantic segmentation are performed in the same stage. The diagrams from Compression and Segmentation are obtained from [4] and [5] respectively.

With the above modifications, the encoding and quantization steps of the compression network will replace the backbone CNN block, as shown in Figure 10, which consisted of a sequence of convolution (*conv*), batch normalisation (*bn*) and reLu activation layers with a final max pooling (*maxpool*). This block highlighted in red is replaced by the green one consisting of the encoding ($g_a(x)$) and quantization (*quantize*) steps of the compression network. Therefore, the quantized

latent code $\hat{y}$ will be the input for the next layer of the backbone structure to produce the feature map for the input of the pyramid pooling module. In this way, the compression stage is integrated into the semantic segmentation network architecture.

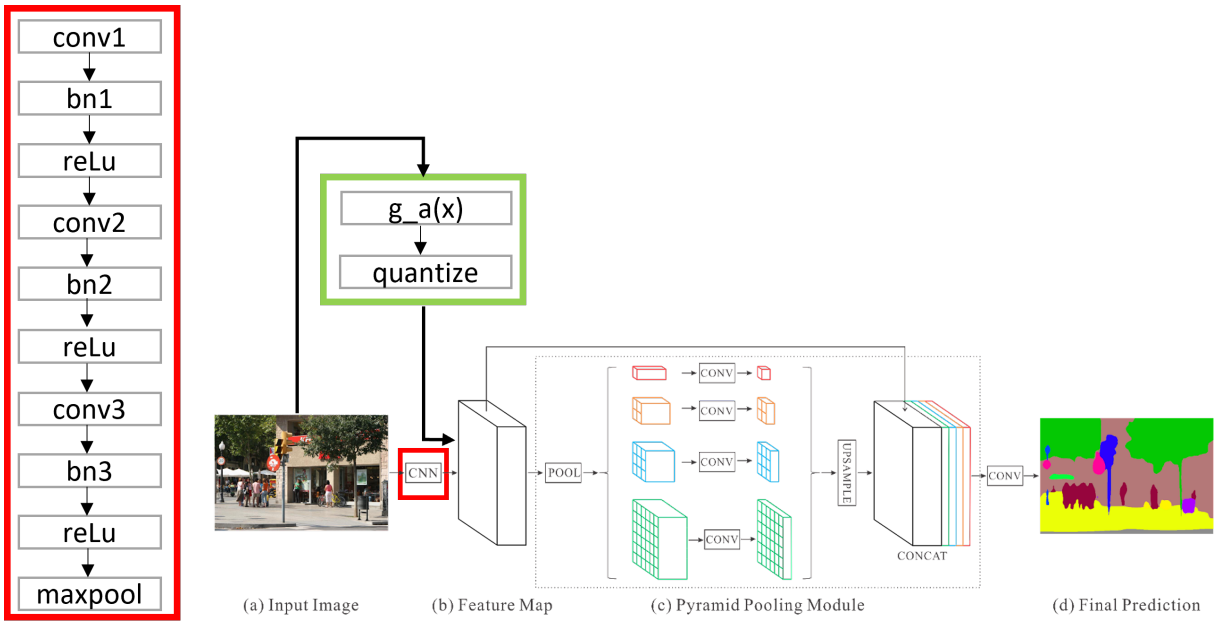

Figure 10: Diagram of the proposed architecture for the latent code approach. Green block replaces red one.

The proposed architecture is then implemented for each level of compression, both for the training and inference stages, unlike the decoded anchor, where modifications were only applied for inference.
Thus, six different models are trained, each one for each quality factor (1-6), under the same conditions as those explained for the baseline model.

It should be noted that the latent dimension depends on the quality factor as defined by the compression network model. Hence, for the first three quality factors (1-3), the latent dimension is 128, while for levels 4-6 the latent depth is 192. This would affect the behaviour of the overall approach, since rather than depending on the losses in the pixel domain, this approach depends on the latent domain.
Trying to reduce the latent dimension of models whose latent is 192, to 128, does not improve the final predictions as it does not learn in the same way. So there will be three models that work in the latent dimension of 128 and three that work in the latent dimension of 192.
Since the input size corresponding to the RGB image is $w \times h \times 3$, the size for the latent code will be $\frac{w}{16} \times \frac{h}{16} \times 128$ for the first three models (the same as for the output of the original CNN block) and $\frac{w}{16} \times \frac{h}{16} \times 192$ for the last three models.
The following layers of the backbone are in charge of processing the different input dimensions to finally extract a map of size $\frac{w}{8} \times \frac{h}{8} \times 2048$ which will be the input for the pyramid pooling module.

By doing the inference for each trained model as a function of the compression level, the prediction for different bitrates is obtained and can be compared with the maps obtained for the decoded anchor.

7 Experiments

7.1 Baseline model

The reference model is built on the original *Semseg* repository with the uncompressed images of the dataset. From this model the reconstruction approach is implemented.
This subsection shows how this model is trained and some examples of segmentation to illustrate possible limitations in prediction that will later influence approaches where compression is used.

7.1.1 Training

The training stage is defined over 200 epochs, being a total of 126600 steps. The input images are resized to 768×768 to match the input requirements of the network and to take advantage of the GPU power. It is used the *izar* cluster from the *SCITAS* server of the EPFL to realize all the experiments.
Some of the hyperparameters defined for the semantic segmentation network include:

- batch size of 4 for training and 1 for validation.
- learning rate of 10^{-2} with a weight decay of 10^{-4} per epoch.
- momentum of 0.9.

The network is configured to apply data augmentation during the training step by cropping, rotating, flipping and blurring the input images.

Figure 11 shows the training evaluation at each step. The blue line represents the overall accuracy of the predictions, the green line indicates the average accuracy of all classes, in this case foreground and background, and the red line the average intersection over the union.
The overall accuracy will give the score by treating the image as a function of the prediction of each of its pixels. Then, this metric depends on the number of pixels in each class, and will make a highly represented class contribute more to the overall score. In the case of mean accuracy, this behaviour is reduced because the score of each class is calculated independently and then averaged by the number of classes (two in this case), so the amount of pixels per class will not determine the mean accuracy score.

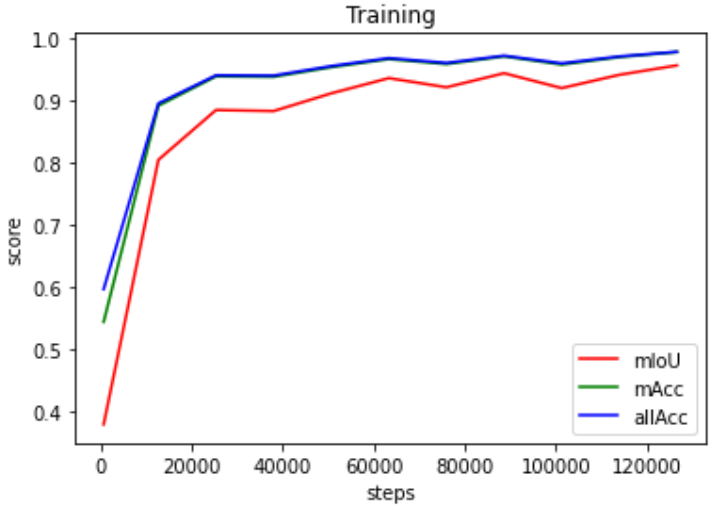

Figure 11: mIoU, mAcc and allAcc throughout training steps for the training set.

It is noteworthy that the blue line overlaps with the green line, so that both get similar scores. In addition, the peaks in the lines are due to the computational limits of the model as it has to be trained in stages of 50 epochs, so there is a transient between the different trainings which means that it does not obtain a completely smooth behaviour. The three metrics stabilise at scores between 0.95 and 0.98, learning almost perfectly.

To check there is no overfitting in the training, a validation set is used to evaluate the performance on a different set of images for each epoch.

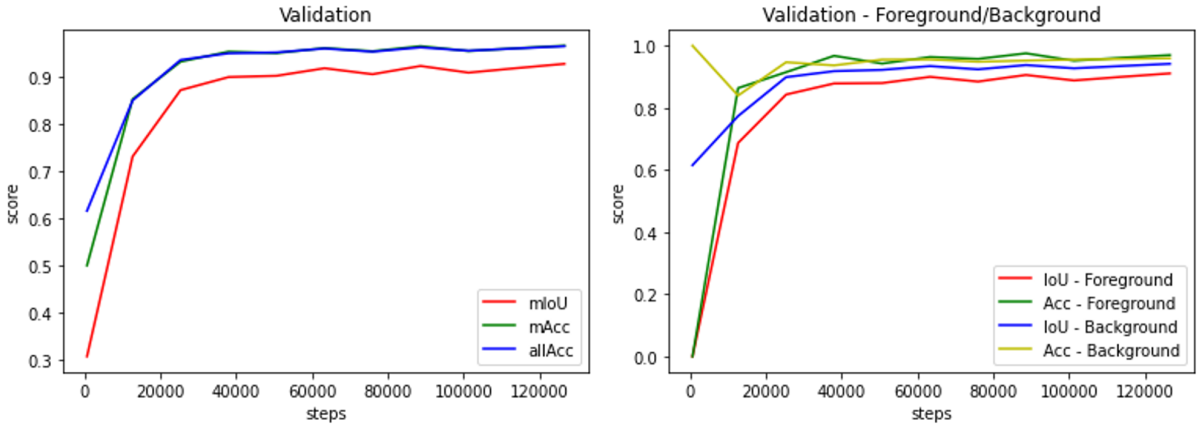

Figure 12: Left: mIoU, mAcc, allAcc throughout training steps for the validation set. Right: IoU and accuracy are separated on class level for the validation set.

Figure 12 shows the evolution of the score on the validation set for the global metrics in the left chart and the class-based metrics in the right one.
The curves in Figure 12 left have a similar behaviour to the training set, so the model is being trained correctly avoiding overfitting. The final score of the metrics is between 0.92 and 0.97, slightly lower than the training set, so it achieves the expected behaviour, as the validation set is only used to evaluate performance, not to learn.
Focusing on Figure 12 right, as expected, the background class scores significantly better than the foreground class, and it is curious to see that at the beginning the accuracy for the background is 1 and for the foreground is 0. This is because the model has not been trained yet and is predicting a whole black mask, that is the reason why the accuracy of the background is 1. When more steps are taken, all the metrics stabilise reaching final values above 0.9.

Therefore, having seen the evolution of the training, both for the training set and the validation set, it can be concluded that the model is training correctly and achieving reasonably high results to be tested with another independent set of images.

7.1.2 Inference

Good segmentation Some examples of good segmentation of the reference model are presented in the following pictures. These results are used as original anchor for the approaches presented in the results.

Figure 13: Left: RGB image. Center: Ground truth. Right: Predicted mask with an accuracy of 0.9758 and mIoU of 0.9523.

Figure 13 shows a good performance of multiple person segmentation. The network correctly predicts where each person is located and does not overlap with any of them. The only difference is at the border between children and rocks, but it is a very small difference between prediction and ground truth.

Figure 14: Left: RGB image. Center: Ground truth. Right: Predicted mask with an accuracy of 0.9965 and mIoU of 0.9930.

Figure 14 is an example of a close-up view, where the player is quite close to the camera and the background is blurred. In this case the person is accurately predicted with a high level of detail as can be seen in the hole located in the right arm.

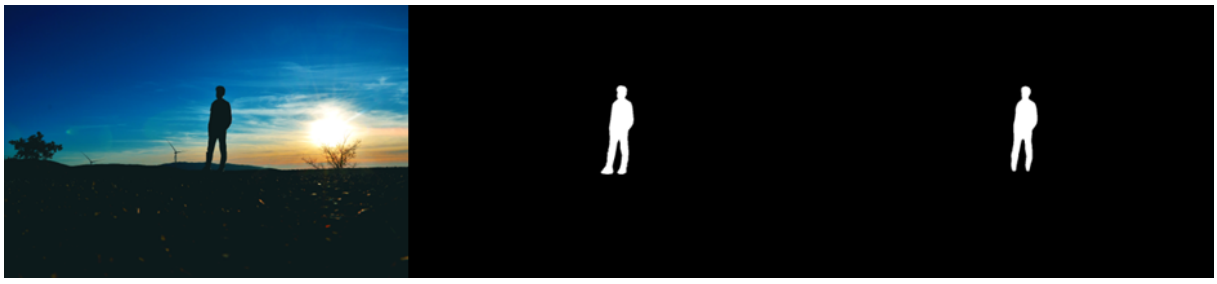

Figure 15: Left: RGB image. Center: Ground truth. Right: Predicted mask with an accuracy of 0.9981 and mIoU of 0.9963.

Finally, Figure 15 presents an example of a person in a distant view. It also has a handicap because the person is seen in black due to the light, so only a silhouette is visible. The prediction reconstructs this silhouette almost perfectly and avoids the rest.

Bad segmentation In contrast, the worst segmented images are presented as follows. All three examples have in common that the person(s) are well segmented but some noise is also introduced, which leads to a low accuracy compared to the predictions seen in the previous examples.

Figure 16: Left: RGB image. Center: Ground truth. Right: Predicted mask with an accuracy of 0.7614 and mIoU of 0.6144.

Figure 16 is an example of a close-up view, the child is correctly segmented, but the other elements in the first view, such as the leg of the bench, are also segmented as a person. Some parts belonging to the background of the image are also wrongly segmented.

Figure 17: Left: RGB image. Center: Ground truth. Right: Predicted mask with an accuracy of 0.7697 and mIoU of 0.6208.

Figure 17 is another example of a close-up view, but in this case the noise is produced by the leaves of the trees in the background of the girl. The person is again well segmented.

Figure 18: Left: RGB image. Center: Ground truth. Right: Predicted mask with an accuracy of 0.9097 and mIoU of 0.8444.

In the last example, Figure 18 shows two people side by side. The prediction is good for both bodies, but on the left side of the image there is noise produced by the floor of the room.

7.2 Bitrates

The test set for the evaluation of the models consists of 53 images, as explained in 3.1.1.
For both the reconstructed code approach and the latent code approach, the compression step is necessary at different levels. In the decoded anchor, the encoding and decoding stages are used to reconstruct the images and use them as input for the semantic segmentation network, while in the latent code approach, only the encoding stage is required as a prior step for the segmentation.
In the case of CompressAI models, the level of compression is defined by the quality factor. For the network used in this project, *cheng2020*, only six compression levels are available, where 1 refers to the maximum compression and 6 to the minimum one. Table 1 presents the average bits per pixel (bitrate) at each compression level and the standard deviation computed over the 53 images in the test set.

Quality	1	2	3	4	5	6
Mean bpp	0.085	0.125	0.179	0.272	0.384	0.523
Std bpp	0.054	0.080	0.113	0.184	0.252	0.324

Table 1: Mean and standard deviation of bits per pixel for each compression level in the test set.

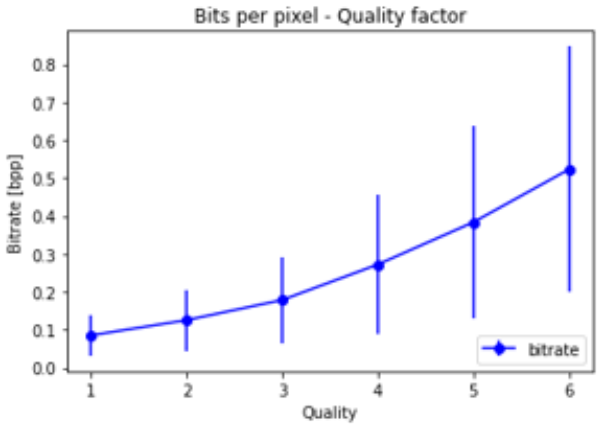

Figure 19: Chart of the average bits per pixel together with the standard deviation for each compression level (quality factor) in the test set.

Figure 19 shows graphically the information in Table 1. For each level of compression (quality factor), the bits per pixel required for each image increases, which makes sense since more information from the original image is retained for higher quality. In addition, the standard deviation also increases as the quality factor also increases. The standard deviation will depend on the type of images in the set. For instance, a textured image will need many more bits per pixel than a smoothed one, and this is maximised at the lowest compression level (quality 6), where as much information as possible should be preserved. Hence, for high compression, more information will be removed, so the standard deviation in the image set when calculating the average bits per pixel will be reduced compared to more permissive compression rates.

7.3 Decoded anchor

For the decoded anchor, it is used the model trained for the original images and evaluated with the decoded ones. Therefore, in this approach the reconstruction performance for different

compression levels is first evaluated for both PSNR and MS-SSIM metrics. In addition, some visual examples are shown to observe the effect of the compression rate.
After the quality assessment, the evaluation of the semantic segmentation is performed, and the results for the intersection over union and pixel accuracy metrics are presented for each of the quality factors.
Examples of the predicted masks are also shown.

7.3.1 Reconstructed images

The two metrics for objective quality assessment are presented in 4.1 and are used to obtain the reconstruction performance as a function of the quality level applied. PSNR will show an objective performance based on the reconstruction error and MS-SSIM will provide a score of the visual quality of the reconstruction.

PSNR The PSNR evaluated in the reconstructed test set follows an increasing path as more bits are used to compress the image. This can be clearly seen in Figures 20 left and 20 right. For the hardest compression level, the average of the set is 31.4 dB, while for the softest compression level the score rises to 37.8 dB. This behaviour is evident because by the time the compression is harder the output image has lost more information compared to the original image.

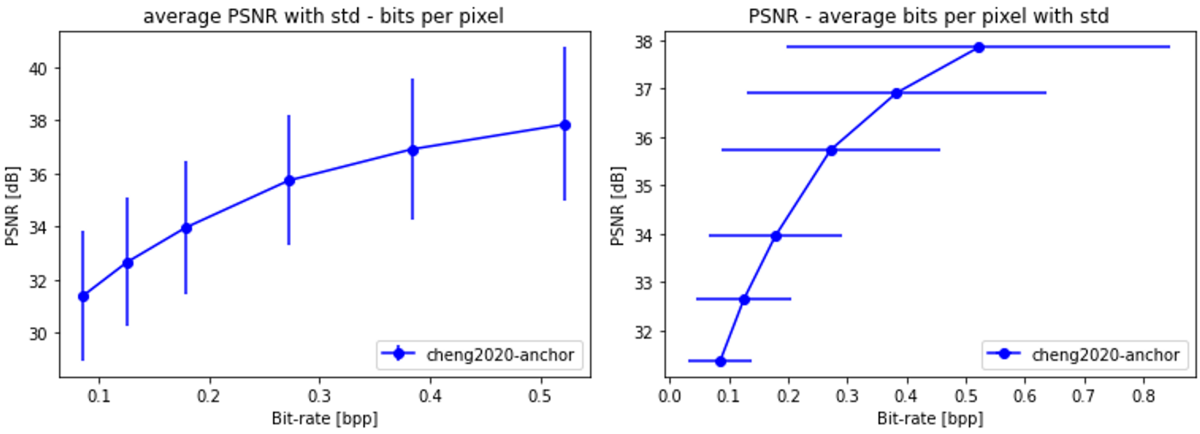

Figure 20: Left: mean and standard deviation of PSNR as a function of average bitrate for the test set. Right: mean of PSNR as a function of average bitrate together with the standard deviation for the test set.

Figure 20 left also shows the standard deviation across the test set at each compression level. Each sample has a similar deviation of around 2.5dB and does not depend on the number of bits per pixel. Therefore, the quality of the reconstructed image is higher as the compression is lower, having an almost constant deviation at each level knowing that this metric does not take into account the visual aspects.
Figure 20 right adds the standard deviation information for the average bits per pixel. It has the same behaviour as explained in Figure 19, where the standard deviation increases as the required bits per pixel do.
The average PSNR value is above 31dB for all bit rates. This gives us information that compression is achieving very good results even for hard compression rates.

MS-SSIM The MS-SSIM adds visual quality information to the result obtained, so the closer the score is to 1, the less information that the human visual system can recognise is lost from the original image. As in Figure 20, Figure 21 shows the metric in terms of average bits per pixel,

together with the standard deviation of the metric in 21 left and the standard deviation of the bit rate in 21 right.

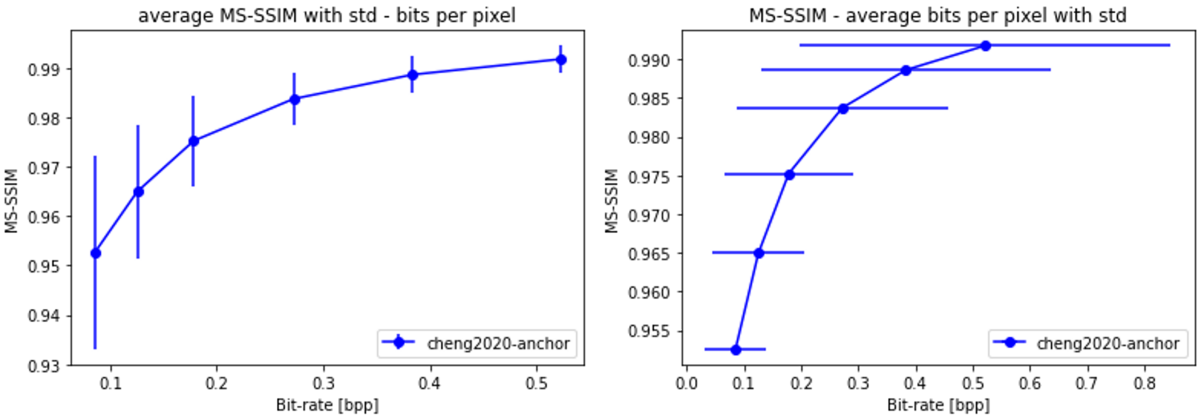

Figure 21: Left: mean and standard deviation of MS-SSIM as a function of average bitrate for the test set. Right: mean of MS-SSIM as a function of average bitrate together with the standard deviation for the test set.

The behaviour follows the expected, the more bits per pixel, the better the MS-SSIM score. Figure 21 left shows that all mean scores are above 0.95, performing well in terms of compression loss as a function of visual quality since they are close to 1. The standard deviation of the set varies depending on the level of compression applied. In the case of the hardest one, the standard deviation is higher than in the others. This makes similar sense to the theory explained in the case of Figure 19. This score will depend on the composition of the image. A textured image will have a greater loss when analysed in terms of visual saliency than a smooth image, so the standard deviation at an aggressive level will vary considerably. For low compression, less information will be lost in each case, so the deviation is smaller, as seen around 0.5 bpp.

Visual examples Analysing the metrics, it can be concluded that the compression method performs well for all bitrates, finding differences due to the factor applied in compression. Let us now analyse in terms of visual details depending on the quality level applied.

In this first image, Figure 22, it is shown how the lines of the hood emerge when the compression is less aggressive. For the extreme cases, in $q = 1$ some of the lines do not appear entirely, while in $q = 6$ all the lines of the original image are reconstructed in detail.

In the second image, Figure 23, the object to focus on is the pocket of the T-shirt. In the $q = 1$ image, it is impossible to recognise the existence of a pocket in this region, but as soon as the quality factor increases, the silhouette of the pocket emerges. In the $q = 6$ image (and some previous ones), a pocket with a button on it is easily distinguishable.

Figure 22: Left: Original RGB image with a red rectangle in the region to be cropped. Right: Reconstructed regions for the quality factor indicated in each subimage.

Figure 23: Left: Original RGB image with a red rectangle in the region to be cropped. Right: Reconstructed regions for the quality factor indicated in each subimage.

Figure 24: Left: Original RGB image with a red rectangle in the region to be cropped. Right: Reconstructed regions for the quality factor indicated in each subimage.

In the last example, Figure 24, the area studied is the person's face, which contains many skin blemishes. For the more aggressive compression rates, it looks like a smoothed face (examples in $q = 1$ or $q = 2$), but for the higher quality rates, $q = 6$ or $q = 5$, original skin blemishes reappear.

These three images can be seen quantitatively in Table 2, where the metrics for these specific images are shown.

	Quality	PSNR[dB]	MS-SSIM	Bitrate (bpp)
Figure 22	1	28.740	0.937	0.118
	2	30.420	0.960	0.191
	3	31.998	0.975	0.285
	4	34.014	0.985	0.445
	5	35.551	0.990	0.636
	6	36.706	0.993	0.866
Figure 23	1	32.987	0.974	0.056
	2	34.215	0.979	0.074
	3	35.280	0.983	0.099
	4	37.030	0.988	0.137
	5	38.093	0.990	0.185
	6	38.634	0.992	0.251
Figure 24	1	31.051	0.958	0.073
	2	32.497	0.972	0.115
	3	33.970	0.979	0.176
	4	36.111	0.988	0.280
	5	37.650	0.993	0.405
	6	38.768	0.995	0.548

Table 2: Mean and standard deviation of bits per pixel for each compression level in the test set.

Figures 22 and 24 have in common that the texture area is larger. This is reflected in the fact that for both images the required amount of bits per pixel at each compression level is higher than average, and also the metrics are lower for aggressive compression rates.
Figure 23 does not have as many textured areas as the others, so the metrics are higher and the required bits per pixel are quite low in comparison.
All three images perform quite well at quality 6, as can also be seen in the visual examples, where at this quality all textured problematic areas are very well reconstructed.

7.3.2 Assessment

After the compression stage, the obtained images are used as input for the semantic segmentation network that will generate a binary mask dividing the foreground of the image (people) into white and the background into black. This section analyses the behaviour of the foreground and background segmentation stage for the reconstructed images.

Intersection over Union The mean Intersection over Union (mIoU) gives information on how well the predicted mask for each class matches the target label. Ideally, these results should be 1 and would denote that the target mask is accurately predicted.
Figure 25 shows this metric taking into account the standard deviation of both classes (vertical red lines). The standard deviation is quite large because the background class tends to have a

significantly higher score than the foreground class and this causes when calculating the average of both classes the deviation is large. At some point the standard deviation line exceeds the original anchor score, this is because the anchor is the mean of both classes, in Figures 26 left and 26 right the original anchor of the corresponding label is provided.
The score depends on the number of bits per pixel used at each level (bitrate axis), and each red circle consists of the average of each compression level. The red line shows the mIoU for the reconstructed approach, while the black constant line refers to the anchor score produced by the predictions of the uncompressed images.

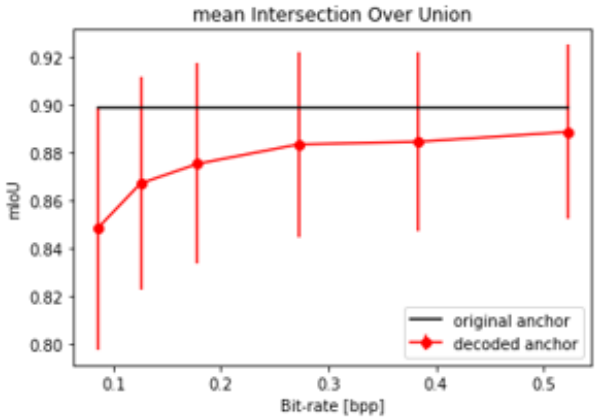

Figure 25: Mean and standard deviation of the IoU metric for the reconstructed test set.

The performance of the mIoU score is increasing as more bits per pixel are used. It starts at 0.85 for an average of 0.08 bits per pixel and tends to stabilise at 0.89 for the smoothest compression level. The anchor is 0.915, so with the reconstructed images, the model performs between 3% and 8% worse in terms of mIoU.

Figures 26 left and 26 right split Figure 25 according to class. Both charts follow the same pattern seen in the average case. For more bits per pixel, the better the IoU score. The main difference between both classes is the value of the score.
Figure 26 left shows the Intersection over Union (IoU) for the foreground class. Thus, the results of this graph give information about how well people are segmented. Objectively it is more interesting than the background as the main goal is to achieve the best possible segmentation for humans.

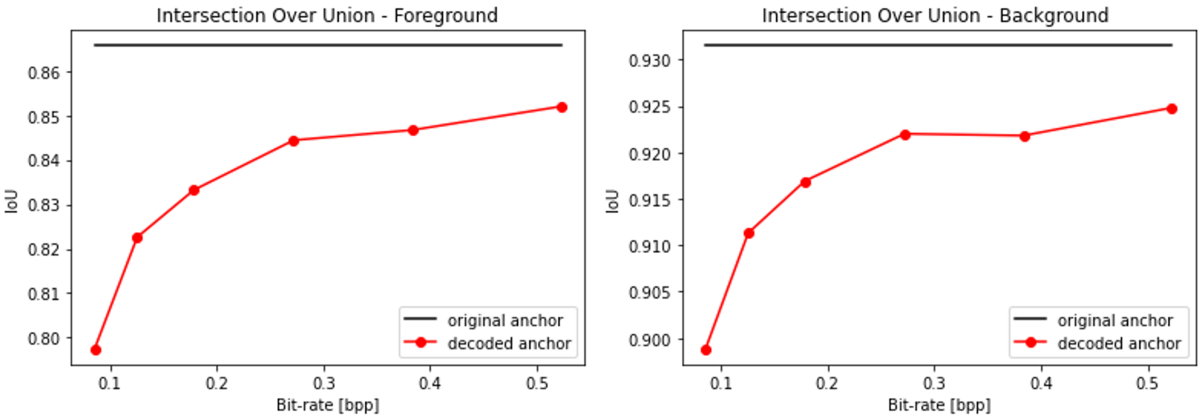

Figure 26: IoU for the test set in the case of, Left: Foreground class; Right: Background class.

The performance of the foreground class starts at almost 0.8 and goes up to 0.85 for the highest quality label. Compared to the anchor at 0.887, the performance decreases by 4% to 10%, slightly worse than average.

Figure 26 right, which shows the IoU performance for the background class, performs between 2% and 5% worse than the anchor. It is therefore more accurate than the foreground label as the difference with respect to the anchor is halved.

The explanation for the phenomenon of the background class getting a higher score and a lower reduction with respect to the anchor is that for a binary human segmentation, the background is the predominant label in the image. This leads to a good background score in images where people are far away, for example, because the percentage of the foreground class is quite low compared to the background. In the case of predicting a completely black mask (background only) for such an image, the IoU will be high for the background because the predicted and target masks will match everywhere in the image except for the small area where the people are located, thus scoring high. In contrast, the foreground class will have a score of 0 in this case because no pixels will overlap.

Pixel accuracy For the pixel accuracy metric, the focus is on the mean of both classes (mAcc) as well as the pixel accuracy of each of the labels, instead of computing the overall accuracy (allAcc) of the prediction. This is done in order to avoid erroneous results caused by unbalanced cases and to be able to observe the contribution of each class to the average.
Figure 27 shows the performance of the mAcc metric as a function of the bitrate. The standard deviation caused by both classes and the anchor score is also introduced.
In this case, the standard deviation is not as large as in the case of the mIoU, because the scores of both classes are closer. Of note is the point near 0.3 bpp where the deviation is somewhat zero because the values for both background and foreground accuracy are quite close.

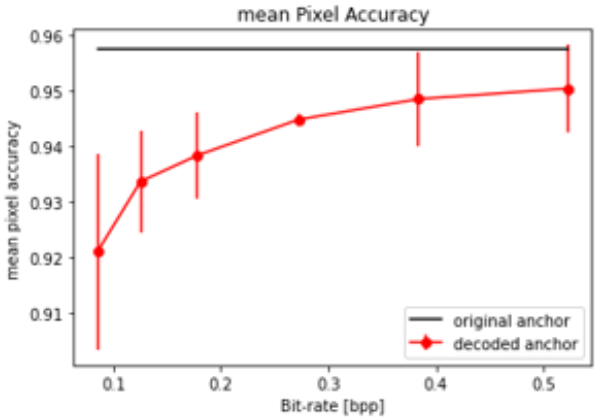

Figure 27: Mean and standard deviation of pixel accuracy for the reconstructed test set.

As seen for the mIoU metric, now the mAcc curve over different compression levels also increases as the compression becomes softer. This behaviour is due to the contribution of the foreground case, Figure 28 left, which tends to improve accuracy as more bits per pixel are used. For the background case, Figure 28 right, it does not tend to increase and instead fluctuates over a small range of values.

This ascending behaviour in Figure 28 left shows that the higher the number of bits per pixel, the more accurately the person is predicted, as the images used for the test are not very unbalanced.

Therefore, the higher the bitrate, the better the person segmentation. This can be seen in the examples in the prediction examples below.
The values start at 0.9 and reach 0.96 for the sixth quality level. Compared to the anchor, it performs between 0.7% and 6% worse.

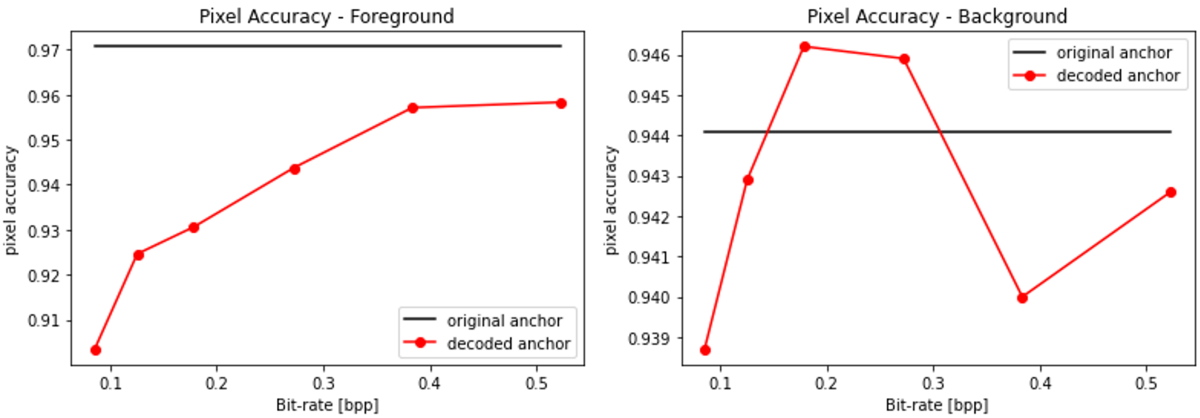

Figure 28: Pixel accuracy for the test set in the case of, Left: Foreground class; Right: Background class.

In the case of the background, the accuracy varies between 0.939 and 0.946, which means that the maximum and minimum scenario score fluctuates by less than 1%, so this behaviour can be defined as constant and not dependent on compression.
As the background is typically the predominant label in an image, it would get high scores for every compression level and therefore does not have an increasing behaviour. Fluctuations are caused by variations in the foreground, since the more background pixels are predicted, the higher the accuracy of the background, regardless of whether the predicted background pixels actually belong to the foreground.

7.3.3 Inference

Below are some examples of the predicted masks for each quality level and their comparison with the ground-truth.

Figure 29: First row: RGB image and ground-truth. Second row: predicted masks with quality factor 1-6 from left to right.

Figure 30: First row: RGB image and ground-truth. Second row: predicted masks with quality factor 1-6 from left to right.

Figure 31: First row: RGB image and ground-truth. Second row: predicted masks with quality factor 1-6 from left to right.

Figure 32: First row: RGB image and ground-truth. Second row: predicted masks with quality factor 1-6 from left to right.

In these examples it can be seen how the decoded anchor predicts. It follows the same patterns as the examples seen in Baseline, because the model is trained on the original images. So, the inference for the reconstructed images is similar to the uncompressed ones.
Focusing on Figures 29 and 30, the segmentation is quite accurate for both a multiple and an

individual person example. Note that in Figure 29, noise appears for $q = 1$ which disappears as the quality factor increases. This tells us that the more bits per pixel used in the compression step, the better the inference.
The other two examples, Figures 31 and 32, are examples of bad segmentation, as is the case with the original images. The silhouette of the person is always well predicted, but some noise appears for all compression levels. Note again that the noise reduces somewhat as the quality factor increases. Therefore, it is clear that the performance of the decoded anchor is fully dependent on the bitrate.

7.4 Latent code approach

In this second approach, the compression stage is integrated into the semantic segmentation network, so there is no need to decode the image before segmenting it. Now the semantic segmentation network learns directly from the latent code.
For this approach, it is necessary to train six different models, one for each compression level, to measure the segmentation performance at different bit rates.

7.4.1 Training

The six models, from q=1 to q=6, are trained under the same conditions as presented in 7.1.1 but following the architecture of 6.2.
Figure 33 shows the performance of the metrics (mIoU, mAcc and allAcc) throughout all training steps, converging to scores above 0.85. Compared to the training baseline, the scores on the training set are slightly lower. This is to be expected as the current architecture presents more complexity by mixing two different tasks, compression and segmentation, in a joint training process.
Note that the scores obtained for the models with quality factor 1, 2 and 3 are slightly higher than those with quality factor 4, 5 and 6. This is due to the difference in the latent dimension, which in the first case is 128 and in the second case 192.

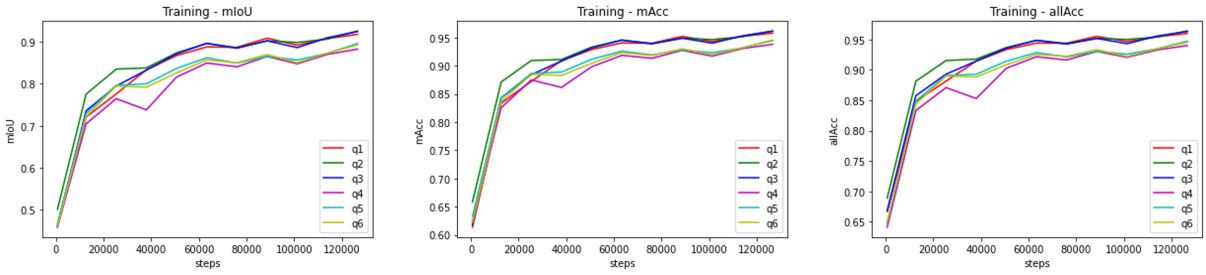

Figure 33: Score along the training steps for the training dataset. Each colour line represents a different quality factor. Left: mIoU. Centre: mAcc. Right: allAcc.

As for the baseline model, a validation set is used to check that no overfitting is occurring in the learning process. After checking that the behaviour is the same as for the training set, and in order not to repeat the same graphs, Figure 34 plots the mIoU of the validation set for the last training step and compares it with the validation set on the decoded anchor. Also Figure 35 shows the IoU for the two classes, foreground on the left and background on the right.

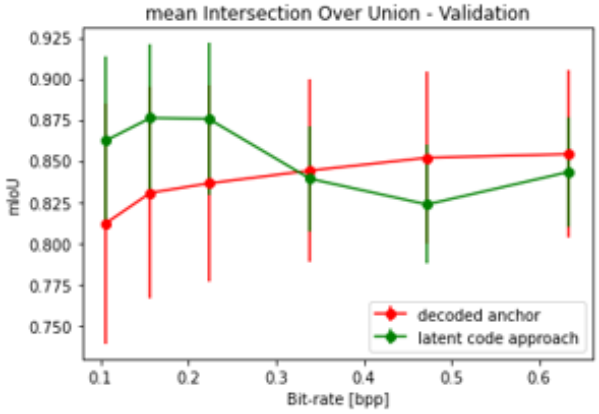

Figure 34: Mean and standard deviation of IoU for the original test set on the latent code approach and the reconstructed test set on the decoded anchor.

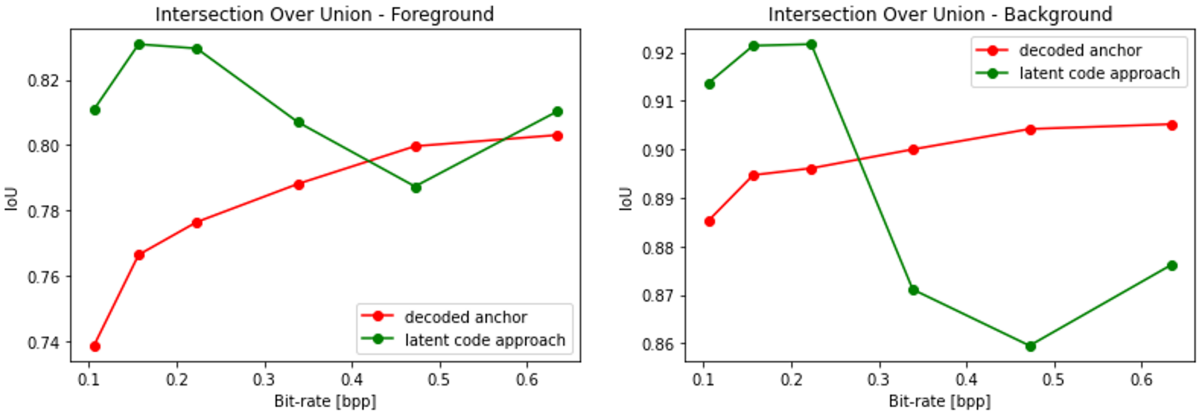

Figure 35: IoU for the original validation set on the latent code approach and the reconstructed validation set for the decoded approach at the last training step in the case of, Left: Foreground class; Right: Background class.

Notice that, instead of having an ascending behaviour as a function of bitrate like the decoded anchor does, this new approach follows a pattern where it performs better for lower bitrates. This is directly related to the latent code dimension explained in 6.3.
The model learns better for low bitrates, as is also observed in the training set throughout the training steps.
This is the first time where the latent code approach is not perceived to depend on the pixel domain but on the code dimension.

These scores in the validation set were used to fine-tune the latent code model. Thus, it was sought to maximise the validation scores in order to finally implement the resulting architecture.

7.4.2 Assessment

Intersection over Union As seen in the decoded anchor, the standard deviation for the mIoU in the latent approach, Figure 36, is also large because the background score is sufficiently higher than the background and produces this deviation.
In this approach, the IoU seems not to follow an increasing pattern, so varies in a range of 0.2 for different bitrates.

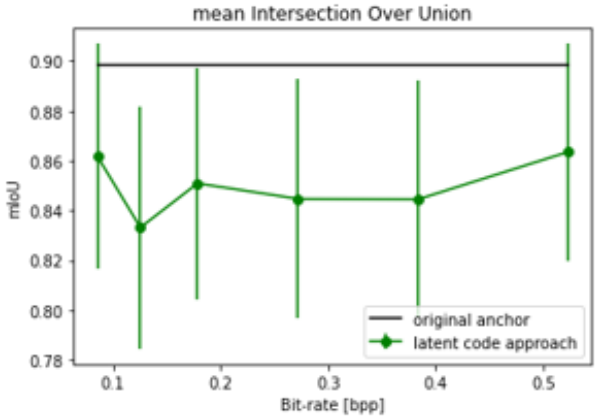

Figure 36: Mean and standard deviation of IoU for the original test set on the latent code approach.

Both classes, foreground and background, follow the same trajectory, achieving the best scores for the hardest and softest compression levels (quality 1 and 6). Between these points, the IoU varies with no clear pattern. The foreground class, Figure 37 left, produces scores in the range [0.79, 0.82] and the background class, Figure 37 right, in the range [0.88, 0.91].

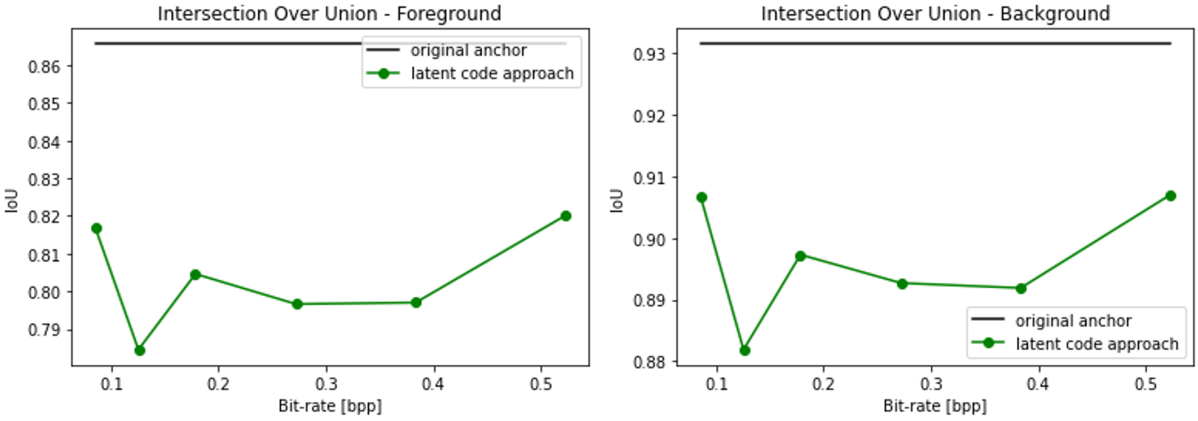

Figure 37: IoU for the test set in the case of, Left: Foreground class; Right: Background class.

Consequently, there is not much difference between the worst and the best performance in terms of IoU, as the difference is only about 3%.
Furthermore, it is observed that both classes follow the same behaviour, so the approach is not operating in a decompensated manner and learns from both classes in the same way. No unbalanced results are generated.
The downward spike caused by the second operating point, between 0.1bpp and 0.2bpp, could be due to the fact that the model for quality 2 predicts more foreground pixels than the prediction of the models for quality 1 and 3. Therefore, many of the foreground pixels predicted for q=2 could be off-target and, consequently, the mIoU and the IoU for both classes decrease.
Finally, it would be interesting to see what happens at the first operational point, as it performs better than the others. This will be discussed in more detail in 7.5.1

Pixel accuracy The pixel accuracy scores also present a relatively flat behaviour when looking at Figure 38. All mean scores are around 0.93, and the largest standard deviation is observed between 0.1bpp and 0.2bpp, as both classes differ at this point.

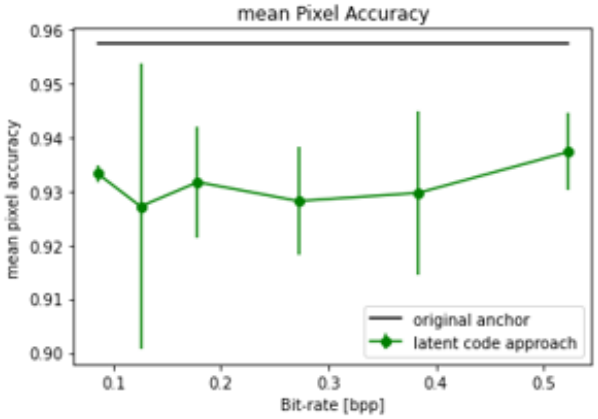

Figure 38: Mean and standard deviation of pixel accuracy for the original test set on the latent code approach.

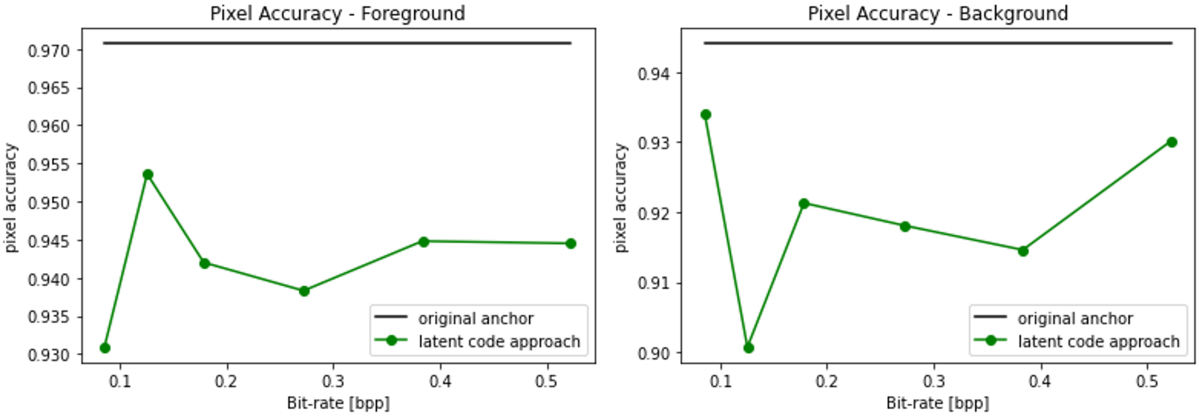

Figure 39: Pixel accuracy for the test set in the case of, Left: Foreground class; Right: Background class.

Analysing Figure 39, both graphs left and right do not follow the same behaviour as in the IoU case. The main difference between both labels occurs at the first and second operating point. As seen in the mAcc graph (Figure 38), a high standard deviation occurs for the second one, due to an increase in the accuracy of the foreground and a decrease in that of the background. This occurs because the model predicts more foreground noise, and pixels labelled as background are predicted as foreground.

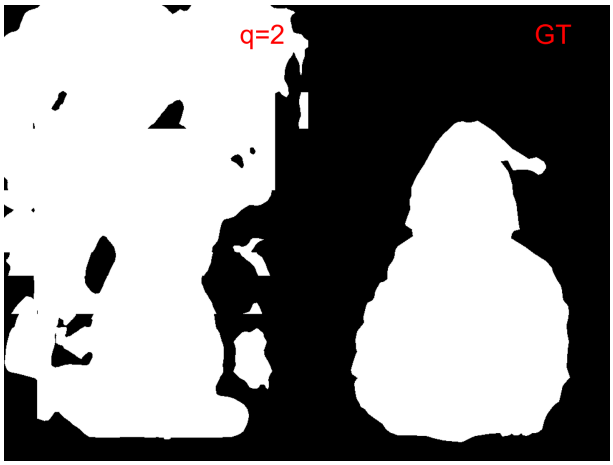

Figure 40: Left: prediction of the quality 2 model. Right: ground-truth.

Figure 40 shows a clear example of what happens at the second operating point for IoU and accuracy. The model prediction labels as foreground a significant part of the background. As this happens for some images, the metrics have a downward peak at this point.
This consequently confirms that the latent code approach does not depend on the pixel domain, since for quality factor 1 (less bitrate than the current example), this phenomenon does not occur. Thus, quality model 2 tends to predict more foreground pixels.

7.4.3 Inference

Below are some examples of the predicted masks for each latent model and their comparison with the ground-truth.

Figure 41: First row: RGB image and ground-truth. Second row: predicted masks for each quality factor 1-6 model from left to right.

Figure 42: First row: RGB image and ground-truth. Second row: predicted masks for each quality factor 1-6 model from left to right.

Figures 41 and 42 analyse a pair of theoretically well-segmented examples. Knowing that the latent code approach does not depend on the pixel domain, it focuses on the fact that in Figure 41 for $q = 5$ and $q = 6$, appears a spot appears on the left player's foot and in Figure 42 some noise appears under the left child. Therefore, in these cases low quality factors predict better than high quality ones. Accordingly, higher compression is applied to obtain better results.

Figure 43: First row: RGB image and ground-truth. Second row: predicted masks for each quality factor 1-6 model from left to right.

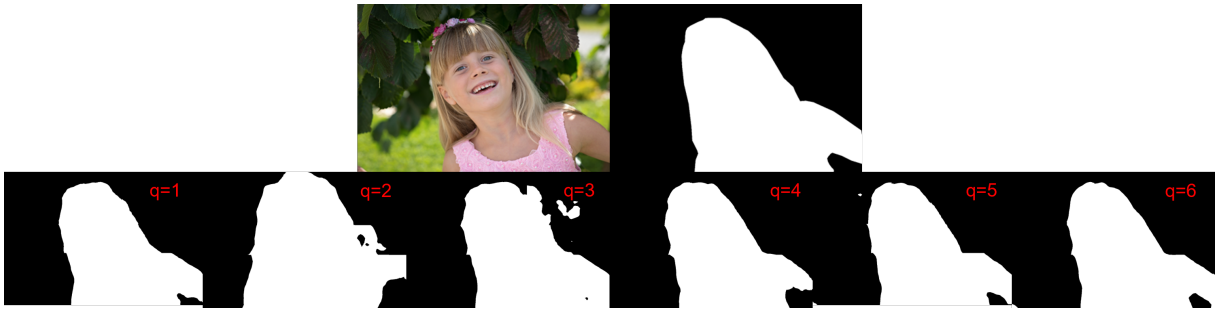

Figure 44: First row: RGB image and ground-truth. Second row: predicted masks for each quality factor 1-6 model from left to right.

In the opposite case, Figures 43 and 44 are theoretically examples of bad segmentation. Looking at them it is interesting to see that not much noise is present and $q = 3$ for Figure 43 and $q = 1$ for Figure 44 achieve really accurate predictions.
Hence, it is shown that for the latent code approach, good predictions can be achieved with aggressive compression rates.

7.5 Comparison

Having seen the performance of the decoded anchor in 7.3 and the latent code approach in 7.4, let's look at the two jointly to see the contributions of each approach and draw conclusions.

This subsection will not explain again how each approach performs, but rather highlight the difference between them, each of which is explained in the appropriate section.

7.5.1 Assessment

Intersection over Union The first behaviour that is distinguishable from the very beginning in Figure 45 is the evolution of each of the approaches. As already explained, the decoded anchor follows a rising behaviour as many bits per pixel are used and the latent code approach fluctuates over a range of scores.

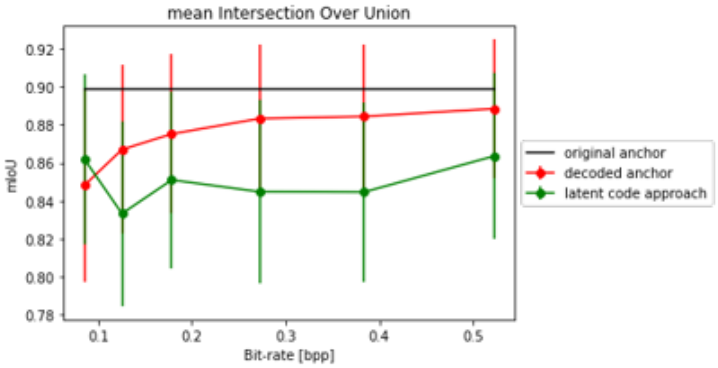

Figure 45: mIoU comparison with std for the latent approach (green), the decoded anchor (red) and the original anchor (black).

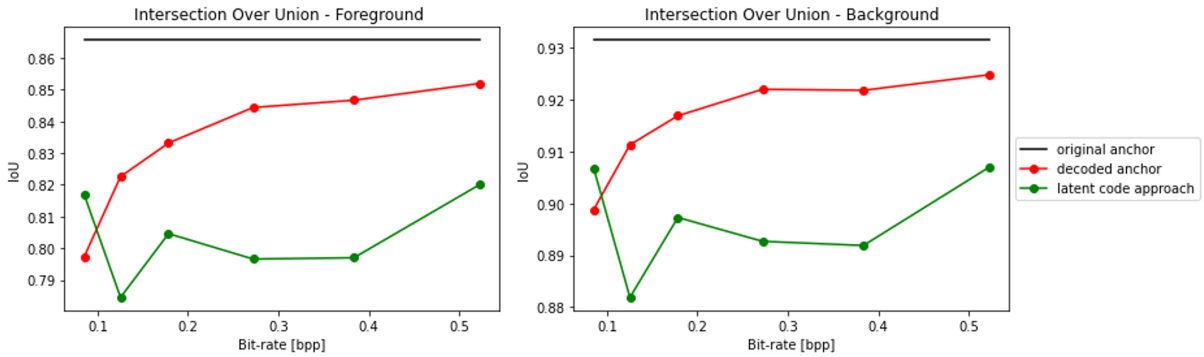

Figure 46: IoU comparison for each class for the latent approach (green), the decoded anchor (red) and the original anchor (black). Left: Foreground. Right: Background.

For the IoU case, both foreground (Figure 46 left) and background (Figure 46 right) follow the same evolution for each of the models. Therefore, the mIoU in Figure X shows that the contribution of both classes follows the same pattern.
If focusing on the scores, it can be seen that there is not much difference between the two approaches, since if looking at the mean, the difference is always between 0.1 and 0.3. For most of the bitrate range, the decoded approach performs slightly better, but it is very interesting to see that for the lowest operating point at 0.085bpp, the latent code approach performs 0.15 points better.
So, the IoU scores suggest that, for the lower bitrates, it is more efficient to use the latent code approach.

Pixel accuracy In the pixel accuracy comparison, at first look it is very similar to the IoU, with higher scores because it is easier to achieve for this metric.

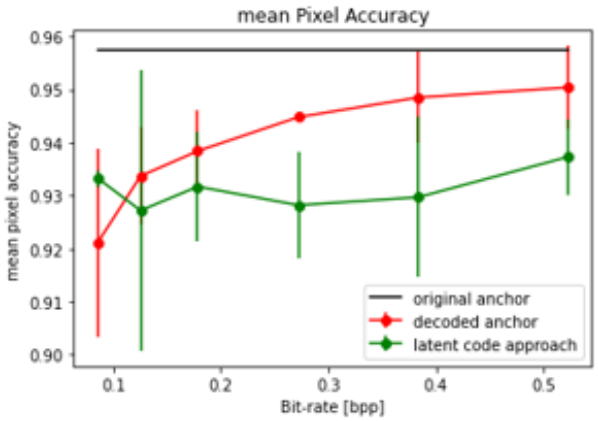

Figure 47: Mean accuracy comparison with std for the latent approach (green), the decoded anchor (red) and the original anchor (black).

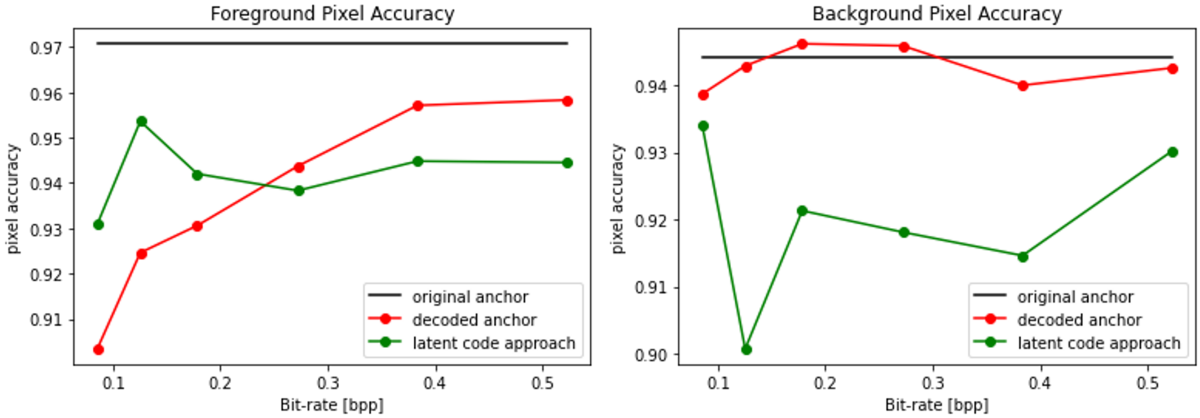

Figure 48: Accuracy comparison for each class for the latent approach (green), the decoded anchor (red) and the original anchor (black). Left: Foreground. Right: Background.

Note that, again, the comparison for the average accuracy in Figure 47 only differs by 0.2 points, obtaining very similar results and repeating the behaviour for the first operating point where the latent code approximation is above the decoded anchor.

For the background accuracy there are two lines without a clear pattern, as explained for the two cases in the corresponding section.
In the foreground case, Figure 48 left, the latent code performs best for the first three operating points. This is because this approach tends to predict more foreground pixels and this conditions the accuracy but does not introduce relevant information to draw conclusions.

First operating point Therefore, after having seen the results for IoU and accuracy metrics, it is clear that the latent code approach is independent of the decoded anchor and is a really suitable option to deal with high compression rates as it leads to good metrics while less bits are needed.
In Figure 49 below, an approach is made for the first operating point which is 0.0853 bits per pixel. Both approaches are compared against each other and all metrics are shown to see how they behave depending on the approach.

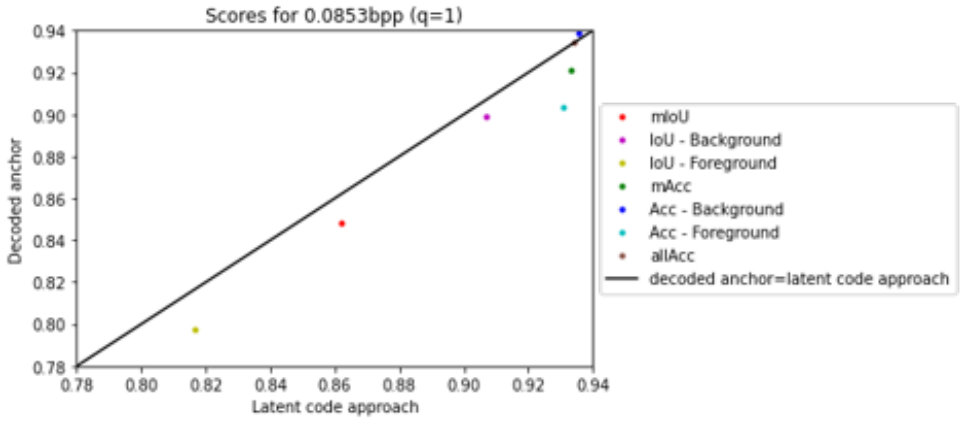

Figure 49: Metric scores at the first operating point (0.0853bpp) for the decoding anchor on the y-axis and the latent code approach on the x-axis.

The black line is the case where both approaches score the same, so to the right of the line shows that the latent approach scores better and to the left of the line the decoded anchor performs better.
The legend of the graph shows what colour each metric is, where the mean and class scores are represented.
Note that of all the metrics studied for this first operating point the latent code approach performs better, except for background accuracy which does not give reliable information.
Thus, the comparison extracts that the best performance is achieved when training in the latent domain for a low bit rate.

7.5.2 Inference

The examples used to illustrate the performance of each approach are used again to compare how both approaches predict against the original anchor.

The first example, Figure 50, is an example of very tight segmentation for the original anchor, as it represents the whole silhouette of the player really accurately, including a small hole between the arm and the body.
For the decoded anchor it is well segmented from the very beginning but the higher the quality factor the more accurate the prediction is by solving some small glitches like a small hole at $q = 1$ in the face or some noise in the foot for $q = 3$.

The latent code approach also segments very close but wider, as the silhouette does not perfectly match the target but the prediction clearly shows that it is the shape of the player. For $q = 5$ and $q = 6$ some noise appears at the bottom, so the lower quality levels perform better.
For this first example both predictions are really good, but perhaps the decoding anchor is a bit more accurate.

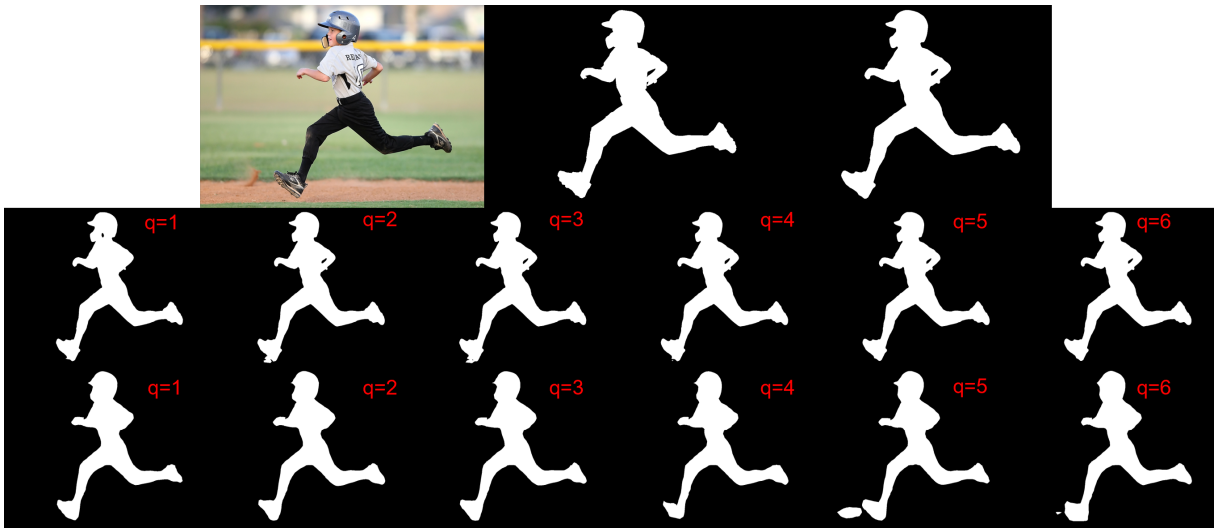

Figure 50: First row: RGB image, ground-truth and original anchor prediction. Second row: decoded anchor prediction for each quality level. Third row: latent code approach prediction for each quality model.

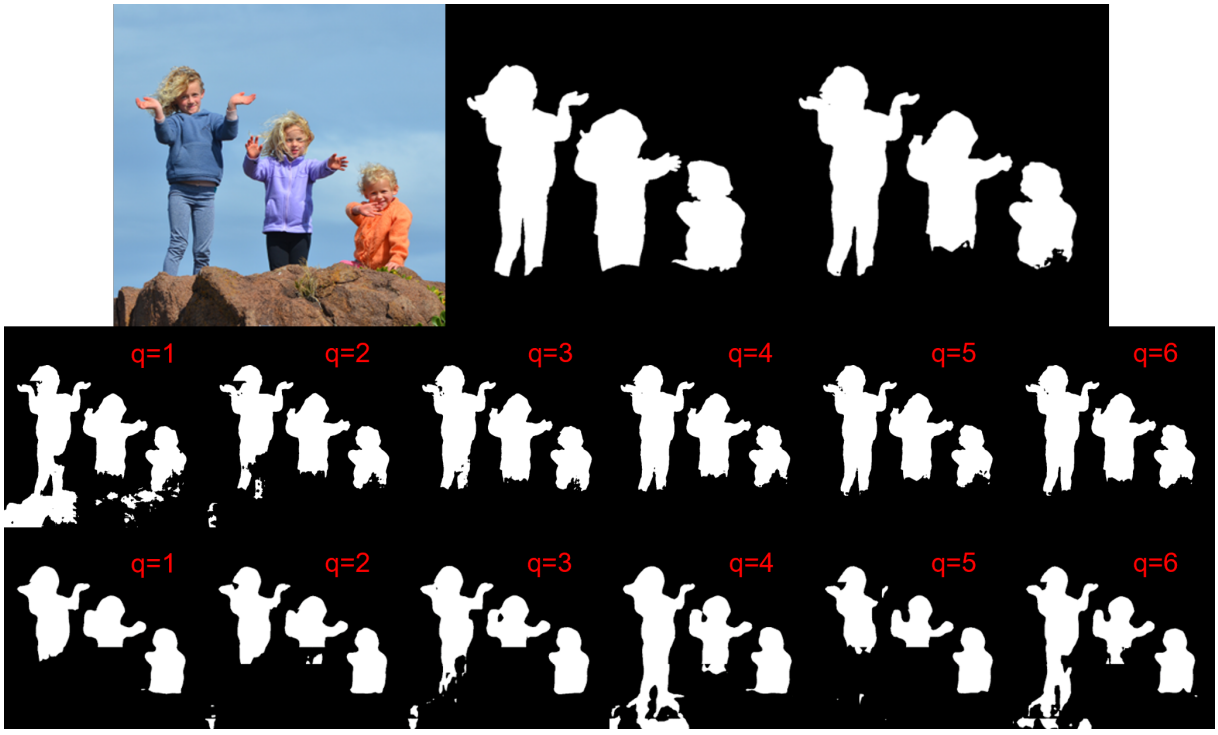

Figure 51: First row: RGB image, ground-truth and original anchor prediction. Second row: decoded anchor prediction for each quality level. Third row: latent code approach prediction for each quality model.

The second example, Figure 51, belongs to a good segmentation for a multiple person case of the original anchor.
The decoded anchor prediction starts to introduce noise at the bottom of the image, but the softer the compression the more accurate the prediction.
Focusing on the latent prediction, it introduces noise for the cases q=4 and q=6, but for example q=1 has no noise. It is true that part of the children's trousers are not segmented as part of them, but all the predicted foreground belongs to people.
In this example, the overall performance is better for the decoded anchor, but focusing on aggressive compression rates (low quality) the segmentation for the latent is more accurate in terms of segmenting people.

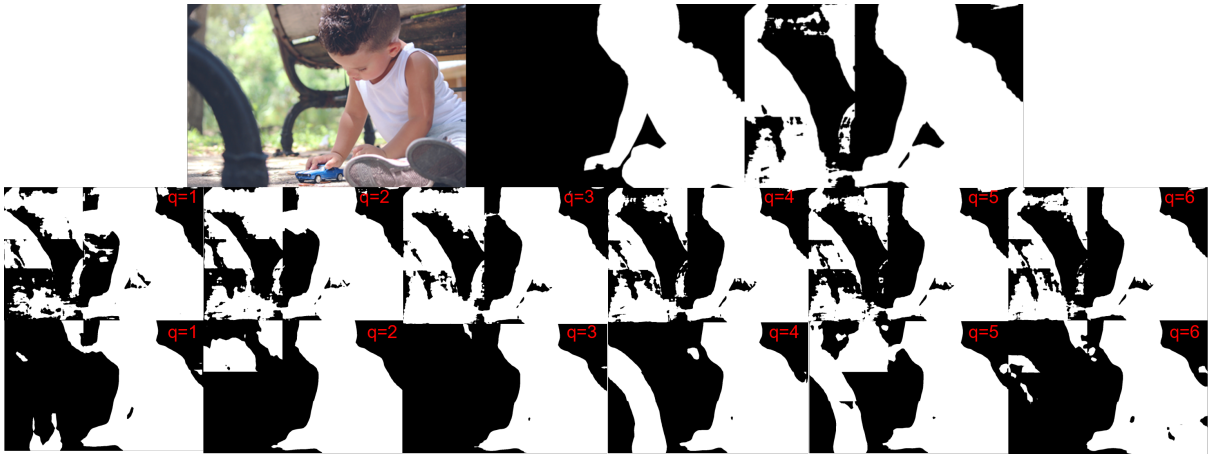

Figure 52: First row: RGB image, ground-truth and original anchor prediction. Second row: decoded anchor prediction for each quality level. Third row: latent code approach prediction for each quality model.

The third example, Figure 52, is the first one where the original anchor predicts with a lot of noise.
The decoded anchor here performs very similarly to the original anchor, so it has noise for each level, reducing it a little when the quality is increased.
Here, the latent actually performs better. It introduces noise at some levels, but not as much as the decoded anchor. Note that $q = 3$ segments very close to the ground-truth, so it outperforms the original anchor. Also $q = 1$ has a good segmentation compared to both anchors.
In this case it is very clear that the latent approach performs much better than the decoded anchor.

Finally, the fourth example (Figure 53) is one in which the difference between approaches is seen in the same way that conclusions are drawn from the metrics.
Again the original anchor is predicting with a lot of noise behind the girl and the decoded anchor reproduces this noise for each quality factor, varying in shape.

Now, this is where it really shows that the latent code approach does not depend on the pixel domain and performs accurately for low bitrates.
For the first quality factor, which we have seen is at the 0.08539bpp operating point, is the segmentation of the girl's silhouette very close to the ground truth and does not reproduce the original anchor noise at any level.

Figure 53: First row: RGB image, ground-truth and original anchor prediction. Second row: decoded anchor prediction for each quality level. Third row: latent code approach prediction for each quality model.

Therefore, it can be concluded after looking at the prediction examples, that the decoded anchor is quite closely related to the original anchor, performing well only when the original anchor has performed well.
The latent code approach introduces that it does not depend on the original anchor and can compensate for bad original segmentations. However, for good segmentations of pixel domain models it is not very accurate, but it does not introduce as much noise that can be annoying as seen in the decoded examples.
So, low bitrates for the latent code approach provide really relevant information.

7.5.3 Runtime benchmark

The following table 3 provides the inference time across the validation dataset for each of the trained models.
Note that the baseline model has been used for the original anchor and the decoded anchor, so the compression stage is not taken into account.
The latent models have the compression stage integrated, so the inference time is for both compression and segmentation.

Model	mean time/image (ms)	std/image (ms)
Baseline	92.6	7.360
Latent Q1	105.2	10.365
Latent Q2	120.8	14.622
Latent Q3	119.15	14.200
Latent Q4	80.05	4.967
Latent Q5	83.55	6.116
Latent Q6	88.85	6.710

Table 3: Mean inference time per image and standard deviation in the validation set for each trained model.

Latent models for $q > 3$ need less time because the latent dimension is larger so they need to apply less compression than the other levels.

Table 4 includes the average time per image and the standard deviation required for the compression of the validation set. This time depends on the compression factor applied to the image and must be added to the Baseline time in the Table 3 to calculate the total end-to-end compression and segmentation inference time.

	mean time/image (ms)	std/image (ms)
Compression	51.32	12.38

Table 4: Mean compression time per image and standard deviation in the validation set.

8 Conclusions

To summarise the work done throughout the project, let's look at the tasks described in the introduction one by one.

- Firstly, it is proposed to use the CompressAI library that implements state-of-the-art compression models for a Pytorch environment. From this platform, the implemented models are analysed and *cheng2020-anchor* is selected as the compression model to be used for the project.
- Secondly, the *Semseg* repository is used, which implements a Pytorch version of the Pyramid Scene Parsing Network, one of the most successful networks for the semantic segmentation task. This implementation is made by the same authors and it is open source, ideal for handling it.
- Thirdly, the Supervisely Person Dataset is presented, which provides a dataset only with images of people with their corresponding binary mask. This dataset fits the foreground/background segmentation approach and it is open and free to use.
- Fourth, two different approaches to deal with image compression and semantic segmentation are proposed. First, an end-to-end compression stage is implemented to apply the result to the semantic segmentation stage. This first approach, called decoded anchor, is useful for later comparison with the second approach. This second approach deals with the integration of compression into the segmentation network. Therefore, a merged approach is proposed in which the semantic segmentation network learns directly from the latent domain.
- Fifth, the comparison of both approaches is presented and analysed in terms of performance metrics and people prediction.

After having made a proposal for each of the defined tasks, some conclusions can be drawn from the comparison of the approaches presented throughout the project.

- The decoded anchor seems to reconstruct the shape of the person better but adding noise while the latent code approach is not as precise on the silhouette but it is very reliable with the right predictions.
- The decoded anchor improves as less compression is applied (better results for better quality), while the latent code approach does not follow a clear pattern.
- The decoded anchor is extremely related to the original anchor in the pixel domain. The latent code approach breaks this dependency and proposes better segmentations for cases where the original anchor segmentation was poor.
- The latent code approach depends on the dimension of the latent dimension, but not on the number of bits per pixel.
- For fewer bits per pixel, the latent code approach performs better than the decoded anchor, both in terms of mIoU and mAcc.
- It is feasible to apply the latent code approach directly in the latent space.
- The inference in the latent code approach is faster than the decoded anchor as it skips the decoding step and the CNN module from the PSPNet.

Finally, some future work is proposed that may be useful for further development based on the results obtained from this project.

- Using CompressAI's *cheng2020-attention* model will improve the current compression stage results. The pre-trained model is not available yet, but will be available soon.
- Review the entire Supervisely Person dataset and re-tag, if necessary, low quality labelling to extend the current one or combine it with other datasets containing person labels, such as the COCO dataset.
- Rethink as an instance segmentation approach to focus only on the segmentation of the person(s) and not be contaminated by the background.
- Extend the current approach to the case of semantic segmentation for multiple tags instead of treating it as a binary semantic segmentation.

As a final note, the current training has been conditioned by the limited computational resources and in order to be able to develop some of the proposals for future work it would be necessary to be able to increase the computing capacity.

References

[1] Supervisely person dataset. https://www.kaggle.com/datasets/tapakah68/supervisely-filtered-segmentation-person-dataset.

[2] Jean Bégaint, Fabien Racapé, Simon Feltman, and Akshay Pushparaja. Compressai: a pytorch library and evaluation platform for end-to-end compression research. *arXiv preprint arXiv:2011.03029*, 2020.

[3] Jpeg-ai call for evidence ieee mmsp2020 challenge. https://jpegai.github.io/test_images/.

[4] Zhengxue Cheng, Heming Sun, Masaru Takeuchi, and Jiro Katto. Learned image compression with discretized gaussian mixture likelihoods and attention modules. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[5] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890, 2017.

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

[7] Robert Torfason, Fabian Mentzer, Eirikur Agustsson, Michael Tschannen, Radu Timofte, and Luc Van Gool. Towards image understanding from deep compression without decoding. *arXiv preprint arXiv:1803.06131*, 2018.

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

[9] Fabrice Bellard. Bpg image format (2014). *Volume*, 1:2, 2016.

[10] Yueyu Hu, Wenhan Yang, Zhan Ma, and Jiaying Liu. Learning end-to-end lossy image compression: A benchmark. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[11] Alain Horé and Djemel Ziou. Image quality metrics: Psnr vs. ssim. In *2010 20th International Conference on Pattern Recognition*, pages 2366–2369, 2010.

[12] Zhou Wang, Eero P Simoncelli, and Alan C Bovik. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, volume 2, pages 1398–1402. Ieee, 2003.

[13] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. *arXiv preprint arXiv:1802.01436*, 2018.

[14] David Minnen, Johannes Ballé, and George D Toderici. Joint autoregressive and hierarchical priors for learned image compression. *Advances in neural information processing systems*, 31, 2018.

[15] Johannes Ballé, Valero Laparra, and Eero P Simoncelli. End-to-end optimized image compression. *arXiv preprint arXiv:1611.01704*, 2016.

[16] Changyue Ma, Zhao Wang, Ruling Liao, and Yan Ye. A cross channel context model for latents in deep image compression. *arXiv preprint arXiv:2103.02884*, 2021.

[17] Benjamin Bross, Ye-Kui Wang, Yan Ye, Shan Liu, Jianle Chen, Gary J Sullivan, and Jens-Rainer Ohm. Overview of the versatile video coding (vvc) standard and its applications. *IEEE Transactions on Circuits and Systems for Video Technology*, 31(10):3736–3764, 2021.

[18] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*, 2014.

[19] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[20] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[21] Hang Zhang, Kristin Dana, Jianping Shi, Zhongyue Zhang, Xiaogang Wang, Ambrish Tyagi, and Amit Agrawal. Context encoding for semantic segmentation. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7151–7160, 2018.

[22] Juntang Zhuang, Junlin Yang, Lin Gu, and Nicha Dvornek. Shelfnet for fast semantic segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[23] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. Lossy image compression with compressive autoencoders. *arXiv preprint arXiv:1703.00395*, 2017.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[25] Supervisely: Unified os/platform for computer vision. `https://supervise.ly`.

[26] T Boutell. Rfc2083: Png (portable network graphics) specification version 1.0, 1997.

[27] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[28] Juana Valeria Hurtado and Abhinav Valada. Chapter 12 - semantic scene segmentation for robotics. In Alexandros Iosifidis and Anastasios Tefas, editors, *Deep Learning for Robot Perception and Cognition*, pages 279–311. Academic Press, 2022.

[29] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[30] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy,

Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[31] Tianfan Xue, Baian Chen, Jiajun Wu, Donglai Wei, and William T Freeman. Video enhancement with task-oriented flow. *International Journal of Computer Vision*, 127(8):1106–1125, 2019.

[32] Hengshuang Zhao. semseg. `https://github.com/hszhao/semseg`, 2019.