TOWARDS IMAGE UNDERSTANDING FROM DEEP COMPRESSION WITHOUT DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT
Motivated by recent work on deep neural network (DNN)-based image compression methods showing potential improvements in image quality, savings in storage, and bandwidth reduction, we propose to perform image understanding tasks such as classification and segmentation directly on the compression representations produced by these compression methods. Since the encoders and decoders in DNN-based compression methods are neural networks with feature-maps as internal representations of the images, we directly integrate these with architectures for image understanding. This bypasses decoding of the compressed representation into RGB space and reduces computational cost. Our study shows that performance comparable to networks that operate on compressed RGB images can be achieved. Furthermore, we show that synergies are obtained by jointly training compression networks with classification networks on the compressed representations, improving image quality, classification accuracy, and segmentation performance. We find that inference from compressed representations is particularly advantageous compared to inference from compressed RGB images at high compression rates.

1 INTRODUCTION
Neural network-based image compression methods have recently emerged as an active area of research. These methods leverage common neural network architectures such as convolutional autoencoders [Ballé et al., 2016; Theis et al., 2017; Rippel & Bourdev, 2017; Agustsson et al., 2017; Li et al., 2017] or recurrent neural networks (Toderici et al., 2015; Johnston et al., 2017) to compress and reconstruct RGB images, and were shown to outperform JPEG2000 (Taubman & Marcellin, 2001) and even BPG (Bellard) on perceptual metrics such as MS-SSIM (Wang et al., 2003). In essence, these approaches encode an image $x$ to some feature-map (compressed representation), which is subsequently quantized to a set of symbols $z$. These symbols are then (losslessly) compressed to a bitstream, from which a decoder reconstructs an image $\hat{x}$ of the same dimensions as $x$ (see Fig.1(a) for an illustration). Quantizing the feature-map imposes a distortion $D$ on $\hat{x}$ w.r.t. $x$, i.e., it increases the reconstruction error. This is traded for a decrease in entropy of $z$, which leads to a decrease of the length of the bitstream as measured by the rate $R$. Thus, to compress images, the goal is to minimize the classical rate-distortion trade-off $D + \beta R$, where $\beta$ controls the trade-off.

Figure 1: We propose to do inference on the learned compressed representation, without decoding.
Besides their outstanding compression performance, learned compression algorithms can—in contrast to engineered compression algorithms—easily be adapted to specific target domains such as stereo images, medical images, or aerial images, leading to even better compression rates on the target domain. In this paper, we explore another promising advantage of learned compression algorithms compared to engineered ones, namely the amenability of the compressed representation they produce to learning and inference without reconstruction (an example is shown in Fig. 1). Specifically, instead of reconstructing an RGB image from the (quantized) compressed representation and feeding it to a network for inference (e.g., classification or segmentation), one uses a modified network that bypasses reconstruction of the RGB image. The rationale behind this approach is that the neural network architectures commonly used for learned compression (in particular the encoders) are similar to the ones commonly used for inference and learned image encoders are hence, in principle, capable of extracting features relevant for inference tasks. The encoder might learn features relevant for inference purely by training on the compression task, and can be forced to learn these features by training on the compression and inference tasks jointly.

The advantage of learning an encoder for image compression which produces compressed representation containing features relevant for inference is obvious in scenarios where images are transmitted, e.g., from a mobile device, before processing, e.g., in the cloud, as it saves reconstruction of the RGB image as well as part of the feature extraction and hence speeds up processing. A typical use case is a cloud photo storage application where every image is processed immediately upon upload for indexing and search purposes. Bypassing RGB image reconstruction and part of the feature extraction hence leads to faster indexing which is of considerable impact given that popular cloud photo storage applications receive millions of uploads each day (e.g., 52m for Instagram).

In addition to this practical aspects, it is interesting to see what types features relevant for (supervised) inference tasks are learned by purely training on the (unsupervised) image compression task. These questions were explored previously in the context of autoencoders for feature learning (see Section 2), where the aim is to learn predictive features for inference as opposed to autoencoders for image compression, which are trained to provide an as accurate and artifact-free reconstruction of the input image as possible.

Our contributions can be summarized as follows:

- We consider two diverse computer vision tasks from compressed image representations, namely image classification and semantic segmentation. Specifically, we use the image compression autoencoder described in [Theis et al., 2017], and adapt ResNet [He et al., 2015] as well as DeepLab [Chen et al., 2016] for inference from the compressed representations.

- We show that image classification from compressed representations is essentially as accurate as from the decompressed images (after fine-tuning on decompressed images), while requiring $1.5 \times -2 \times$ fewer operations than reconstructing the image and applying the original classifier.

- Further results indicate that semantic segmentation from compressed representations is as accurate as from decompressed images at moderate compression rate, while being more accurate at high compression rate. This suggests that learned compression algorithms might learn semantic features at high compression rates. Again, segmentation from compressed representation requires $1.5 \times -2 \times$ fewer operations than segmentation from decompressed images.

- When jointly training for image compression and classification, we observe an increase in SSIM and MS-SSIM and, at the same time, an improvement classification accuracy.
Our method only requires minor changes in the original image compression autoencoder and classification/segmentation network architecture, and slight changes in the corresponding training procedures.

The remainder of the paper is organized as follows. We give an overview over related work in Section 2. In Section 3, we review the deep compression architecture proposed in (Theis et al., 2017) as well as the training procedure form (Agustsson et al., 2017), describe the modifications we did to leverage them in our work, and further propose a variant of ResNet (He et al., 2015) amenable to compressed representations. We present and evaluate our methods for image classification and semantic segmentation from compressed representations in Sections 4 and 5 respectively, along with baselines on compressed RGB images. In Section 6, we then address joint training of image compression and classification from compressed representation. Finally, we discuss our findings in Section 7.

2 RELATED WORK

There are a few examples of learning from features extracted from compressed (by engineered codecs) images in the literature. Classification of compressed hyperspectral images was studied in (Hahn et al., 2014; Aghagolzadeh & Radha, 2015). Recently, Fu & Guimaraes (2016) proposed an algorithm based on Discrete Cosine Transform (DCT) to compress the images before feeding them to a neural net for reportedly a 2 to 10 times speed up of the training with minor image classification accuracy loss. Javed et al. (2017) provide a critical review on document image analysis techniques directly in the compressed domain. To our knowledge, inference from compressed representations produced by learned image compression algorithms has not been considered before.

In the context of video analysis different approaches for inference directly from compressed video (obtained using engineered codecs) were proposed, see (Babu et al., 2016) for an overview. The temporal structure of compressed video stream naturally lends itself to feature extraction for many inference tasks. Examples include video classification (Biswas & Babu, 2013; Chadha et al., 2017) and action recognition (Yeo et al., 2008; Kantorov & Laptev, 2014).

Our work is related to unsupervised feature learning using autoencoders in that we propose a method that learns on top of learned feature representations. Hinton & Salakhutdinov (2006) proposed a dimensionality reduction scheme using autoencoders to learn robust image features that can be used for classification and regression. A more robust dimensionality reduction was proposed by Vincent et al. (2008) using denoising autoencoders where input noise is added to make the learned features more robust and Rifai et al. (2011) penalize the Jacobian of the learned representation of the autoencoder to learn more stable features. Convolutional autoencoders to learn hierarchical features were proposed by Masci et al. (2011). Hinton & Salakhutdinov (2006), Vincent et al. (2008), Rifai et al. (2011), Masci et al. (2011) are all motivated by using autoencoder as an pre-training step to initialize the networks that are then finetuned on the target task. Coates et al. (2011) present an analysis of unsupervised feature extraction methods, including autoencoders, for classification.

Finally, compression artifacts from both learned and engineered compression algorithms will compromise the performance of inference algorithms. The effect of JPEG compression artifacts on image classification using neural networks was studied in (Dodge & Karam, 2016).

3 LEARNED DEEPLY COMPRESSED REPRESENTATION

3.1 DEEP COMPRESSION ARCHITECTURE

For image compression, we use the convolutional autoencoder proposed in (Theis et al., 2017) and the training procedure described in (Agustsson et al., 2017). The encoder takes an input image \( x \) and has 2 convolutional layers with spatial subsampling by a factor of 2, followed by 3 residual units and finally a convolutional layer with spatial subsampling by a factor of 2. This results in a \( w/8 \times h/8 \times C \)-dimensional feature volume, where \( w \) and \( h \) are the spatial dimensions of \( x \) and the number of channels \( C \) is a hyperparameter related to the rate \( R \). The feature volume is then quantized to a set of symbols, forming the symbols volume. To get the reconstruction \( \hat{x} \), the symbols
volume is fed into the decoder, which mirrors the encoder but uses upsampling and deconvolutions instead of subsampling and convolutions.

To handle the non-differentiability of the quantization step during training, Agustsson et al. (2017) employ a differentiable (soft) approximation of quantization and anneal it to the actual (hard) quantization during training to prevent inversion of the soft quantization approximation. Here, we replace this procedure by a different quantization step, $Q$, which behaves like $\hat{Q}$ in the forward pass but like $\hat{Q}$ in the backward pass (using the notation of Agustsson et al. (2017)). Like annealing, $Q$ prevents inversion of the soft quantization approximation, but facilitates joint training of the autoencoder for image compression with an inference task (see Section 6). Additionally, we chose to use scalar quantization (i.e., $p_n = \bar{p}_n = 1$ in the notation of Agustsson et al. (2017)) to further simplify joint training (while losing some compression performance, see Appendix A.3 in Agustsson et al. (2017)).

We train the network to minimize the rate-distortion trade-off $D + \beta R$. To do so, we use the mean squared error (MSE) between $x$ and $\hat{x}$ as a metric for $D$. We estimate $R$ using $H(q)$, where $q$ is estimating the probability distributions of the symbols using a histogram (see Agustsson et al., 2017) for details. Training our network for a certain $\beta$ results in an operating point with a certain MSE and a certain bit rate as measured by bits per pixel (bpp). To better control the bpp, we introduce the target entropy $H_t$ to formulate our loss:

$$L_c = \text{MSE} + \beta \max \left( H(q) - H_t, 0 \right)$$

(1)

We train compression networks for three different bpp operating points by choosing different values for $H_t$ and different values for $C$. In theory, changing $H_t$ is enough to change the resulting average bpp of the network, but we found it beneficial to also change $C$. We obtain three operating points at 0.0983 bpp ($C = 8$, low), 0.330 bpp ($C = 16$, medium), 0.635 bpp ($C = 32$, high). We use learning rates of $1e^{-3}$, $1e^{-5}$, and $1e^{-3}$ for the low, medium, and high operating points, respectively. We train on the images from the ILSVRC2012 dataset (see Section 4.2), using a batch size of 30. We train each operating point for 600k iterations. Fig. 3 depicts the performance of our deep compression models vs. standard JPEG and JPEG2000 methods on ILSVRC2012 data.

![Figure 3: MS-SSIM, SSIM and PSNR as a function of rate for the reported Deep Compression operating points, JPEG 2000 and JPEG](image)

The resulting learned compressed representation is visualized in Fig. 4. The original image is shown along with compressed versions, reconstructed from the compressed representation, and the 4 channels with the highest entropy of the compressed representation, for each compression operating point. The total number of channels for the compressed representation are 8, 16, and 32 for the 0.0983, 0.330, and 0.635 bpp operating points, respectively. As the bpp gets progressively lower the entropy cost of the network forces the compressed representation to use fewer symbols, as can clearly be seen in Fig. 4. For the most aggressive compression, the channel maps are almost binary.

### 3.2 ResNet for compressed representations

For input images with spatial dimension $224 \times 224$ the resulting compressed representation from the compression network is a tensor with dimension $28 \times 28 \times C$, where $C$ is the number of channels.
To adapt the ResNet architecture to using the compressed feature tensors with spatial dimension $28 \times 28$ as input, we propose a simple variant of ResNets which we refer to as cResNet-$k$, where $k$ is the number of convolutional layers of the network. These networks are constructed by simply ‘cutting’ off the front of the regular (RGB) ResNets, removing the root-block and the residual layers that have a larger spatial dimension than $28 \times 28$. To adjust the number of layers $k$, we stick to the recipe of [He et al., 2015] and only adjust the number of $14 \times 14$ (conv4, x) residual blocks.

Employing this method we get 3 different architectures: (i) cResNet-39 is ResNet-50 with the first 11 layers chopped off as described above, significantly reducing computational costs; (ii) cResNet-51 and (iii) cResNet-72 are then obtained by adding $14 \times 14$ residual blocks to match the computational cost of ResNet-50 and ResNet-71, respectively.

A detailed description of these architectures and their computational complexity is given in Table 1.

4 IMAGE CLASSIFICATION FROM COMPRESSED REPRESENTATIONS

4.1 DEEP METHOD

For image classification from RGB-images we use the ResNet-50 (V1) architecture [He et al., 2015]. It is composed of so-called bottleneck residual units where each unit has the same computational cost regardless of the spatial dimension of the input tensor (with the exception of blocks that subsample spatially, and the root-block). Constant computational cost is maintained by doubling the number of channels when subsampling spatially by a factor 2. The network is fully convolutional and its structure can be seen in Table 1 for inputs with spatial dimension $224 \times 224$.

Following the architectural recipe of [He et al., 2015], we adjust the number of $14x14$ (conv4, x) blocks (see Table 1) to obtain ResNet-71, an intermediate architecture between ResNet-50 and the well known ResNet-101.
The image to the network and use the output from the encoder as input to the Nets and use the mean over the ImageNet dataset.

For pre-processing we do random-mirroring of inputs, random-cropping of inputs for RGB images, and 28 × 28 for feature volumes. Downsampling is performed by conv3-1, conv4-1, and conv5-1 with a stride of 2.

### Table 1: Structure of the ResNet and the cResNet architectures

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>conv2_x</td>
<td>56 × 56</td>
<td>1 × 1, 64</td>
<td>1 × 1, 64</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 × 3, 64</td>
<td>3 × 3, 64</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 × 1, 256</td>
<td>1 × 1, 256</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3_x</td>
<td>28 × 28</td>
<td>1 × 1, 128</td>
<td>1 × 1, 128</td>
<td>1 × 1, 128</td>
<td>1 × 1, 128</td>
<td>1 × 1, 128</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 × 3, 128</td>
<td>3 × 3, 128</td>
<td>3 × 3, 128</td>
<td>3 × 3, 128</td>
<td>3 × 3, 128</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 × 1, 512</td>
<td>1 × 1, 512</td>
<td>1 × 1, 512</td>
<td>1 × 1, 512</td>
<td>1 × 1, 512</td>
</tr>
<tr>
<td>conv4_x</td>
<td>14 × 14</td>
<td>1 × 1, 256</td>
<td>1 × 1, 256</td>
<td>1 × 1, 256</td>
<td>1 × 1, 256</td>
<td>1 × 1, 256</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 × 3, 256</td>
<td>3 × 3, 256</td>
<td>3 × 3, 256</td>
<td>3 × 3, 256</td>
<td>3 × 3, 256</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 × 1, 1024</td>
<td>1 × 1, 1024</td>
<td>1 × 1, 1024</td>
<td>1 × 1, 1024</td>
<td>1 × 1, 1024</td>
</tr>
<tr>
<td>conv5_x</td>
<td>7 × 7</td>
<td>1 × 1, 512</td>
<td>1 × 1, 512</td>
<td>1 × 1, 512</td>
<td>1 × 1, 512</td>
<td>1 × 1, 512</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 × 3, 512</td>
<td>3 × 3, 512</td>
<td>3 × 3, 512</td>
<td>3 × 3, 512</td>
<td>3 × 3, 512</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 × 1, 2048</td>
<td>1 × 1, 2048</td>
<td>1 × 1, 2048</td>
<td>1 × 1, 2048</td>
<td>1 × 1, 2048</td>
</tr>
<tr>
<td>FLOPs</td>
<td></td>
<td>5.38 × 10⁷</td>
<td>3.66 × 10⁷</td>
<td>3.36 × 10⁸</td>
<td>3.33 × 10⁸</td>
<td>2.95 × 10⁷</td>
</tr>
</tbody>
</table>

Table 1: Structure of the ResNet and the cResNet architectures. The input to the ResNet networks are RGB images with a spatial dimensions 224 × 224. The input to the cResNet networks is a compressed feature tensor with spatial dimensions 28 × 28. Building blocks are shown in brackets, with the numbers of blocks stacked. Downsampling is performed by conv3-1, conv4-1, and conv5-1 with a stride of 2.

### 4.2 Benchmark

We use the ImageNet dataset from the Large Scale Visual Recognition Challenge 2012 (ILSVRC2012) (Russakovsky et al. 2014) to train the compression network and for image classification. It consists of 1.28 million training images, 50k validation images and 100k test images (not available/closed source). These images are distributed across 1000 diverse classes. For image classification we report top-1 classification accuracy (or simply just accuracy) and top-5 classification accuracy. Since the test images are not openly available all numbers are reported on the validation set.

### 4.3 Training Procedure

We use the ResNet implementation from the slim library in TensorFlow (Abadi et al. 2015) with some modifications for the custom architectures. For a fair comparison when using different bitrates, we train all classifications networks from scratch in our experiments. To scale the training down so that it can be done on a single GPU we use batch size 64 (compared to 256 in (He et al., 2015)). Goyal et al. (2017) showed that when scaling the batch size for training ResNets the learning rate should be scaled linearly with the batch size. We adopt that method and use a learning rate 0.025 with batch size 64 (in the original paper (He et al., 2015) ResNet uses batch size 256 and learning rate 0.1). We employ the same learning rate schedule as in (He et al., 2015), but for a speedup of training iterations we decay the learning rate at a faster pace (3.75× faster). We use a constant learning rate that is reduced by a factor 10 at 8, 16, and 24 epochs and the network is trained for a total of 28 epochs. A stochastic gradient descent (SGD) optimizer is used with momentum 0.9. We use weight decay of 0.0001. For pre-processing we do random-mirroring of inputs, random-cropping of inputs (224 × 224 for RGB images, 28 × 28 for feature volumes) and center the images using per channel mean over the ImageNet dataset.

When training on feature volumes and compressed RGB images we train the compression network for each operating point. Then we fix the the compression network (encoder and decoder), feed the image to the network and use the output from the encoder as input to the cResNets and use the output from the decoder as inputs the the ResNets.

### 4.4 Classification results

In Table 2 and in Fig 5 the results for the classification accuracy of the different architectures at each operating point is listed, both classifying from the compressed representation and the corresponding reconstructed images.
We look at top-5 classification accuracy as a function of computational complexity for the 0.0983 bpp compression operating point. This is done by doing inference on the reconstructed images with the ResNet-50 and the ResNet-71 architecture and plotting their computational cost vs. the resulting accuracy. For the compressed representation the same was done with cResNet-39, cResNet-51 and cResNet-72. As discussed before the comparison is in favor of the reconstructed images when looking at computational cost without the cost of decoding, but as soon as the decoding cost is accounted for doing inference from the compressed representations performs better.

Table 2: Image classification accuracies - 3.75x training schedule after 28 epochs. Note that the input to the cResNet are compressed representation and the inputs to ResNet are RGB images. Results shown for 3 different operating points and the original images (top row)

<table>
<thead>
<tr>
<th>bpp</th>
<th>Network architecture</th>
<th>Top 5 accuracy</th>
<th>Top 1 accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0983</td>
<td>ResNet-50</td>
<td>89.96</td>
<td>71.06</td>
</tr>
<tr>
<td></td>
<td>cResNet-51</td>
<td>87.85</td>
<td>67.68</td>
</tr>
<tr>
<td></td>
<td>cResNet-39</td>
<td>87.47</td>
<td>67.17</td>
</tr>
<tr>
<td>0.635</td>
<td>ResNet-50</td>
<td>86.25</td>
<td>65.18</td>
</tr>
<tr>
<td></td>
<td>cResNet-51</td>
<td>85.87</td>
<td>64.78</td>
</tr>
<tr>
<td></td>
<td>cResNet-39</td>
<td>85.46</td>
<td>64.14</td>
</tr>
<tr>
<td>0.330</td>
<td>ResNet-50</td>
<td>78.52</td>
<td>55.30</td>
</tr>
<tr>
<td></td>
<td>cResNet-51</td>
<td>78.20</td>
<td>55.18</td>
</tr>
<tr>
<td></td>
<td>cResNet-39</td>
<td>77.65</td>
<td>54.31</td>
</tr>
</tbody>
</table>

Fig 5 shows how for the 2 classification architectures with the same computational complexity (ResNet-50 and cResNet-51) the validation curves at the 0.635 bpp compression operating point are almost the same and as the bpp lowers the gap gets even smaller. Table 2 shows for the final point the ResNet-50 only performs 0.5% percentage points better in top-5 accuracy at the 0.635 bpp operating point. And the gap closes as the bpp gets lower: for the 0.0983 bpp operating point this difference is only 0.3% percentage points.

Using the same pre-processing and the same learning rate schedule but starting from the original RGB images yields 89.96% top-5 accuracy. So the performance of the compressed representation at the 0.635 bpp compression operating point is even competitive with using the original images for inference at a significantly lower memory cost.

We look at top-5 classification accuracy as a function of computational complexity for the 0.0983 bpp compression operating point in Fig. 6. This is done by doing inference on the reconstructed images with the ResNet-50 and the ResNet-71 architecture and plotting their computational cost vs. the resulting accuracy. For the compressed representation the same was done with cResNet-39, cResNet-51 and cResNet-72. As discussed before the comparison is in favor of the reconstructed images when looking at computational cost without the cost of decoding, but as soon as the decoding cost is accounted for doing inference from the compressed representations performs better.

Figure 5: Top 5 accuracy on the validation set for different architectures at each compression operating point. The input to the cResNet are the compressed representations and the input to the ResNet are corresponding RGB images reconstructed from the compressed representations.
5 SEMANTIC SEGMENTATION FROM COMPRESSED REPRESENTATIONS

5.1 DEEP METHOD

For semantic segmentation we use the ResNet based Deeplab architecture (Chen et al., 2016) and start from the codes from https://github.com/DrSleep/tensorflow-deeplab-resnet. When doing fine-grained 'per-pixel' tasks, such as segmentation, it is important not to subsample the feature maps too much spatially. The cResNet and ResNet image classification architectures from Sections 4.1 and 3.2, respectively, are hence repurposed with atrous convolutions to increase their receptive field as described in (Chen et al., 2016) to prevent the network from subsampling the feature maps below the layers of spatial dimension of 28 × 28 (224/8 × 224/8). The last 1000-way classification layer of these classification architectures is then replaced by an atrous spatial pyramid pooling (ASPP) with four parallel branches with rates \{6, 12, 18, 24\} which provides the final classification of each pixel from the feature map.

5.2 BENCHMARK

The PASCAL VOC-2012 dataset Everingham et al. (2015) for semantic segmentation was used for image segmentation tasks. It has 20 object foreground classes and 1 background class. The dataset consists of 1464 training, 1449 validation and 1456 test (not available/closed source) images. Every pixel in each image is annotated with one of the 20 + 1 classes. The original dataset is furthermore augmented with extra annotations provided by Hariharan et al. (2011) so the final dataset has 10,582 images for training. All performance is measured in pixelwise intersection-over-union (IoU) averaged over all the classes, or mean-intersection-over-union (mIoU). As the test images are not openly available we report all numbers on the validation set.

5.3 TRAINING PROCEDURE

For the training of the segmentation architecture we use same settings as in Chen et al. (2016). We use batch size 10 and perform 20k iterations for training using SGD optimizer with momentum 0.9. The initial learning rate is 0.001 (0.01 for final classification layer) and the learning rate policy is that at each step the original learning rate is multiplied by \((1 - \frac{\text{iter}}{\text{max iter}})^{\text{power}}\) where \(\text{power} = 0.9\). We use a weight decay of 0.0005. For preprocessing we do random-mirroring of inputs, random-cropping of inputs (320 × 320 for RGB images, 40 × 40 for the compressed representation) and center the images using per channel mean over the dataset. The cResNet/ResNet networks are pre-trained on the ImageNet dataset using the procedure described in ?? on the image classification task, the encoder and decoder are again fixed and then the architectures are finetuned on the image segmentation task.
5.4 SEGMENTATION RESULTS

In Table 3 and in Fig. 3 the results for the semantic segmentation of the different architectures at each operating point is listed, both doing segmentation from the compressed representation and the corresponding reconstructed images. In contrast to the classification performance for the 2 classification architectures with the same computational complexity (ResNet-50 and cResNet-51) they perform equally well at the 0.635 bpp compression operating point. At the 0.0983 bpp operating point the semantic segmentation from the compressed representation is actually better than for the reconstructed RGB images. The mIoU of for the compressed representations is 54.62% vs. 52.97% for the reconstructed RGB images from the feature volumes.

Fig. 8 and Fig. 9 show the segmentation visually for 2 different images for both the cResNet-51 and the ResNet-50 architecture at each operating point. The highlight the fact that these are difficult task which can be performed reasonably well using compressed representation. They also indicate how details in the segmentation get lost as the bpp gets lower, the segmentation suffers from the lack of detail in the images.

The segmentation masks for the 0.0983 bpp operating point in Fig. 8 show where the reconstructed RGB image fails to capture the back part of the train, while the compressed representation manages to capture that aspect of the image in the segmentation.

In Fig. 6 the mIoU validation performance as a function of computational complexity for the 0.0983 bpp compression operating point is reported. This is done in the same way as is reported in Section 4 but for segmentation. Here, even without accounting for the decoding cost of the reconstructed images, the compressed representation still performs better. At a fixed computational cost, segmentation from the compressed representation gives about 1.6% better mIoU. And at a fixed mIoU the computational cost is about $1.5 \cdot 10^9$ FLOPs less expensive for compressed representation. Accounting for the decoding costs this difference becomes $4.3 \cdot 10^9$ FLOPs.

![Figure 7: mIoU validation performance for different segmentation architectures at each compression operating point. The input to the cResNet are the compressed representations and the input to the ResNet are corresponding RGB images reconstructed from the compressed representations](image-url)

6 JOINT TRAINING FOR COMPRESSION AND IMAGE CLASSIFICATION

6.1 FORMULATION

To train for compression and classification jointly, we combine the compression network and the cResNet-51 architecture. The quantized compressed feature tensors are fed to the decoder to optimize for mean-squared reconstruction error and also to a cResNet-51 network to optimize for classification using a softmax loss. The combined loss function takes the form

$$
\mathcal{L}_c = \gamma (\text{MSE} + \beta \max (H(q) - H_t, 0)) + \ell_{ce}(y, \hat{y}),
$$

where the loss terms for the compression network are the same as in training for compression only, see Section 3 $\ell_{ce}$ is the cross-entropy loss for classification, and $\gamma$ controls the trade-off between the mean-squared reconstruction error and the classification loss.
Table 3: Image segmentation performance. Using the Deeplab training schedule starting from pre-trained ResNets described in Section 4. The input to the cResNet are compressed representations and the inputs to ResNet are RGB images. Results shown for 3 different operating points and the original images (top row).

<table>
<thead>
<tr>
<th>bpp</th>
<th>Network architecture</th>
<th>mIoU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ResNet-50</td>
<td>65.75</td>
</tr>
<tr>
<td>0.635</td>
<td>ResNet-50</td>
<td>62.97</td>
</tr>
<tr>
<td></td>
<td>cResNet-51</td>
<td>62.86</td>
</tr>
<tr>
<td></td>
<td>cResNet-39</td>
<td>61.85</td>
</tr>
<tr>
<td>0.330</td>
<td>ResNet-50</td>
<td>60.75</td>
</tr>
<tr>
<td></td>
<td>cResNet-51</td>
<td>61.12</td>
</tr>
<tr>
<td></td>
<td>cResNet-39</td>
<td>60.78</td>
</tr>
<tr>
<td>0.0983</td>
<td>ResNet-50</td>
<td>52.97</td>
</tr>
<tr>
<td></td>
<td>cResNet-51</td>
<td>54.62</td>
</tr>
<tr>
<td></td>
<td>cResNet-39</td>
<td>53.51</td>
</tr>
<tr>
<td></td>
<td>ResNet-71</td>
<td>54.55</td>
</tr>
<tr>
<td></td>
<td>cResNet-72</td>
<td>55.78</td>
</tr>
</tbody>
</table>

Original image/mask

Figure 8: **Top:** Images at different compression operating points. **Middle:** Segmentation mask from compressed feature tensor with architecture cResNet-51. **Bottom:** Segmentation mask from RGB images reconstructed from the compressed representation with architecture ResNet-50

When training the cResNet-51 networks for image classification as described in Section 4.3 the compression networks are fixed (they were previously trained as described in Section 3). When doing joint training the compression network is initialized from this fixed state and the cResNet-51 is initialized from this trained state. This state of the networks is used as the initialization and then the network is ‘finetuned’.

The learning rate schedule is similar to the one used in the image classification setting. It starts with an initial learning rate of 0.0025 that is divided by 10 every 3 epochs using a SGD optimizer with momentum 0.9. The joint network is then trained for a total of 9 epochs.
We set the hyperparameters to $\gamma = 0.001$, $\beta = 150$ and $m = 1.265$ for the 0.635 bpp operating point and $\gamma = 0.001$, $\beta = 600$ and $m = 0.8$ for the 0.0983 bpp operating point.

To ensure a fair comparison we performed the following control experiments. 1) To compare compression metrics the compression network is trained using the same learning rate schedule as the joint network without the classification network, to make sure that the improvements are not solely due to increased training and a different training schedule from before. 2) To compare the classification metrics the cResNet-51 is trained using the same learning rate schedule with the fixed encoder and decoder to make sure that the classification improvements are not due to longer training of the network. 3) To compare if the classification performance increase is merely due to the improvement of the operating point, a full cResNet-51 was trained from scratch using the operating point obtained from 1)

6.2 Joint training results

In Fig. 9 and in Fig. 11 the results of finetuning the joint network, compression and classification at the same time, are compared to finetuning the compression network only, with the same learning rate schedule. As is evident, at the same bpp compared to finetuning the compression network only, the joint training slightly improves both MS-SSIM and SSIM. However the PSNR goes down slightly. The same results apply for 0.0983 and the 0.635 bpp compression operating points.

When doing joint training the classification performance of the network also increases. For the 0.635 bpp compression operating point the top-5 performance starts at 87.85%. When doing joint finetuning the operating points moves to 0.62 bpp but the top-5 classification accuracy improves to 88.34%. Now if we continue training the cResNet with the fixed operating point, 0.635 bpp, the performance improves to 88.11% and when training a full ResNet for the improved operating point that was gotten by finetuning the compression network first and then training a ResNet the top-5 accuracy ends up at 88.10%, so the joint training clearly improves the accuracy.
For the 0.0983 bpp operating point we get 78.20% top-5 accuracy. When training only the cResNet with a fixed compression network it ends up at 78.50%, a slight improvement. Training an improved operating point by only training the compression network and then training a full cResNet on top of the now operating points results in a top-5 accuracy of 79.20% at 0.099 bpp. Now when doing the joint training we get an operating point at 0.10 that gives a top-5 performance of 81.25%, which is an improvement of 2%.

Figure 10: The change in image metrics when finetuning the compression network only vs. finetuning jointly the compression network with the classification architecture. The image shows the starting point in image metrics before finetuning and after. This is done for the 0.645 bpp compression operating point.

Figure 11: The change in image metrics when finetuning the compression network only vs. finetuning jointly the compression network with the classification architecture. The image shows the starting point in image metrics before finetuning and after. This is done for the 0.0983 bpp compression operating point.
7 DISCUSSION

We proposed and explored inference when starting directly from learned compressed representations without the need to decode, for two fundamental computer vision tasks: classification and semantic segmentation of images.

In our experiments we departed from a very recent state-of-the-art deep compression architecture proposed by [Theis et al., 2017] and showed that the obtained compressed representations can be easily fed to variants of standard state-of-the-art DNN architectures while achieving comparable performance to the unmodified DNN architectures working on the decoded/reconstructed RGB images (see Figure 6). In particular, only minor changes in the training procedures and hyperparameters of the original compression and classification/segmentation networks were necessary to obtain our results.

The main strong points of the proposed method for image understanding from deep compression without decoding are the following:

**Runtime** Our approach saves decoding time and also DNN inference time as the DNN adapted models can be shorter than those using the decoded RGB images for comparable performance.

**Memory** Removing the need for reconstructing the image is a feat with large potential for real-time memory constrained applications which use specialized hardware such as in the automotive industry. Complementary, we have the benefit of shorter DNN models and high compression rates with proved good performance.

**Robustness** The approach was successfully validated for image classification and semantic segmentation with minimal changes in the specialized DNN models, which make us to believe that the approach can be extended to most of the related image understanding tasks, such as object detection or structure-from-motion.

**Synergy** The joint training of compression and understanding DNN models led to synergistic improvements in both compression quality and classification/segmentation accuracy.

**Performance** According to our experiments and the top performance achieved, compressed representations are a promising alternative to the largely common use of decoded images as starting point in image understanding tasks.

At the same time the approach has a couple of shortcomings:

**Complexity** In comparison with the current standard compression methods (such as JPEG, JPEG2000) the deep encoder we used and the learning process have higher time and memory complexities. However, research on deep compression is in its infancy while techniques such as JPEG are matured. Recently, [Rippel & Bourdev, 2017] have shown that deep compression algorithms can achieve the same or higher (de)compression speeds as standard compression algorithms on GPUs. As more and more devices are being equipped with dedicated deep learning hardware deep compression could become commonplace.

**Performance** The proposed approach is particularly suited for high compression rates and wherever the memory constraints and storage are critical. Medium to high compression rates are also the regime where deep compression algorithms considerably outperform standard ones.

Extending our method for learning from compressed representation to other computer vision tasks is an interesting direction for future work. Furthermore, gaining a better understanding of the features/compressed representations learned by image compression networks might lead to interesting applications in the context of unsupervised/semisupervised learning.

REFERENCES


