Improved Disentanglement through Aggregated Convolutional Feature Maps

Maximilian Seitzer MAXIMILIAN.SEITZER@IIS.FRAUNHOFER.DE
Fraunhofer Institute for Integrated Circuits IIS, Intelligent Systems Group, Erlangen, Germany

Abstract

We present a simple image preprocessing method for training VAEs leading to improved disentanglement compared to directly using the images. In particular, we propose to use regionally aggregated feature maps extracted from CNNs pretrained on ImageNet. Our method achieves the first rank on 3 of 5 metrics on the challenge's public leaderboard.

1. Introduction

The representational power and utility of feature representations obtained from deep CNNs trained on large image datasets such as ImageNet (Russakovsky et al., 2014) is well-known. Amongst others, they are routinely used by practitioners to improve performance in transfer learning scenarios, and form the basis for perceptual loss functions (Johnson et al., 2016). A common view explaining the success of deep convolutional representations is that they describe an image in an abstract, concise way, simplifying downstream tasks such as classification (Bengio et al., 2012). Thus, a natural hypothesis to draw is that it is easier for a Variational Autoencoder (VAE) (Kingma and Welling, 2013) to disentangle the latent factors of variations from this abstract description than from image itself. Therefore, in our challenge submission, we employ pretrained CNNs to extract convolutional feature maps as a preprocessing step before training the VAE. To reduce the high-dimensional feature maps and fit the challenge's resource restrictions, we propose to aggregate the feature maps using a regional pooling technique from the context of image retrieval.

2. Method

Our method consists of the following three steps: (1) from each image in the dataset, extract a convolutional feature map using a CNN pretrained on ImageNet (section 2.1), (2) each feature map is aggregated into a feature vector and stored in memory (section 2.2), (3) a VAE is trained to reconstruct the feature vectors and disentangle the latent factors of variation (section 2.3).

Due to limited personal time leading up to the challenge deadline, most of our hyperparameter choices are based on trial-and-error and thus we can unfortunately not provide detailed studies and justifications for now. We plan to provide a more detailed study about our method in the future. Appendix A contains further comments about our choices and the things we additionally tried out.

2.1. Feature Map Extraction

To extract convolutional feature maps from the images, we use the VGG19-BN[1] architecture (Simonyan and Zisserman, 2014) in the `torchvision` package. In particular, we use the pretrained weights stemming from training on ImageNet without further finetuning them in any way. Input images are transformed to the format the pretrained networks expect, i.e. we bilinearly resize them to 224×224 pixels and standardize them using mean and variance across each channel computed from the ImageNet dataset. We use the outputs of the last layer before the final average pooling, resulting in a spatial feature map of size $512 \times 7 \times 7$.

2.2. Feature Aggregation

As the memory limitations of the challenge prohibit us to store the full feature maps in memory, we choose to aggregate them into feature vectors. This also appears sensible as the dimensionality of the full feature maps is actually larger than of the input images ($3 \times 64 \times 64$), and thus learning the latent factors from feature maps might actually be harder than from the original images.

To perform the feature aggregation, we adapt a technique introduced in the context of object retrieval, called *regional maximum activations of convolutions (RMAC)* (Tolias et al., 2015). In object retrieval, the goal is to find the image an object appears on from a collection of images. Tolias et al. (2015) achieve this by matching a feature vector carrying the object's "signature" against an RMAC feature vector for each image. To allow matching against all the different objects that appear in an image, RMAC aggregates the signatures of objects at different scales and locations into the image feature vector. We assume that this property of RMAC is also useful in our case, as we need to consider different objects (e.g. in MPI3D, the robotic arm and the object) to find the latent factors of variation from feature maps, but we do not know the scale and location of these object a priori.

We compute RMAC by applying max-pooling operations with different kernel sizes and strides to the feature maps (without any padding), resulting in a set of 512-dimensional feature vectors. Concretely, we use kernel sizes 1×1, 3×3, 5×5 and 7×7 with strides 1, 2, 2, 1 respectively. These values were experimentally found to result in good performance. We then $\ell 2$-normalize each of the feature vectors, sum all vectors up and apply a final $\ell 2$-normalization, resulting in the aggregated feature vector. In contrast to Tolias et al. (2015), we do not apply PCA-whitening to the feature vectors before the summation.

To reduce the computational overhead, we would like to extract and aggregate the features for each image only once before training, and store them in memory. But because the challenge only allows to sample from the dataset (rather than selectively accessing each available image), there will be some amount of duplicates among the stored feature vectors. To increase the amount of unique latent factor combinations available during training, we sample 1000000 images from the dataset for feature extraction, albeit the dataset having only 460800 images. We note that this sampling introduces an unnecessary source of randomness increasing the variance between runs.

1. `https://download.pytorch.org/models/vgg19_bn-c79401a0.pth`

2.3. VAE Training

Finally, we train a standard VAE on the set of aggregated feature vectors resulting from the previous step. The encoder network consists of 4 fully-connected layers with 256, 128, 64, and 32 neurons, followed by two fully-connected layers parametrizing mean and log variance of the factorized Gaussian approximate posterior $q(\boldsymbol{z} \mid \boldsymbol{x}) = \mathcal{N}\left(\boldsymbol{\mu}, \boldsymbol{\sigma}^2\right)$ with $C = 19$ latent factors. The number of latent factors was experimentally determined. The decoder network consists of 4 fully-connected layers with 32, 64, 128, and 256 neurons, followed by a fully-connected layer parametrizing the mean of the factorized Gaussian conditional distribution $p(\boldsymbol{x} \mid \boldsymbol{z}) = \mathcal{N}(\hat{\boldsymbol{\mu}}, \boldsymbol{I})$. All fully-connected layers but the final ones use batch normalization and are followed by ReLU. We use the standard Pytorch initialization for all layers and assume a factorized Gaussian prior on the latent variables $p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$.

For optimization, we use the Adam optimizer with a learning rate of 0.001, $\beta_0 = 0.999$, $\beta_1 = 0.9$ and a batch size of 256. The VAE is trained for $N = 23$ epochs by minimizing

$$\sum_{i=1}^{512} (\hat{\mu}_i - x_i)^2 - 0.5 \frac{\beta}{C} \sum_{j=1}^{C} 1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2$$

where β is a hyperparameter to balance between the MSE reconstruction and the KLD penalty terms. As the scale of the KLD term depends on the numbers of latent factors C, we normalize it by C such that β can be varied independently of C. It can be harmful to start training with too much weight on the KLD term (Bowman et al., 2015). Therefore, we use the following cosine schedule to smoothly anneal β from 0 to $\beta_{\text{end}} = 0.13 - 10^{-5}$ over the course of training[2]:

$$\beta(t) = \frac{\beta_{\text{end}}}{2}\left(1 - \cos\frac{\pi t}{N}\right)$$

where $\beta(t)$ is the value for β in training episode $t \in \{0, \ldots, N-1\}$. Instead of $\beta(t)$, we use a warm-up value of $\beta_{\text{start}} = 10^{-5}$ for the first 2 epochs. This schedule lets the model initially learn to reconstruct the data and only then puts pressure on the latent variables to be factorized which we found to considerably improve performance.

3. Conclusion

At the time of writing, results for *MPI3D-real* (Waleed Gondal et al., 2019) were not yet available. On the public leaderboard (i.e. on *MPI3D-realistic*), our best submission achieves the first rank on the FactorVAE (Kim and Mnih, 2018), SAP (Kumar et al., 2017) and DCI (Eastwood and Williams, 2018) metrics. See appendix B for a discussion of the results.

As Locatello et al. (2018) point out, for successful unsupervised disentanglement, some kind of inductive biases are required. We suggest that pretrained feature extractors can play the role of a strong inductive bias for natural image data. Our method could also be a straight-forward avenue to scale disentanglement techniques to larger image sizes. This report only provides exploratory results, but we think that the initial results are promising enough to warrant further investigation.

2. Due to a bug, this is the schedule we used, but not the one we intended to use (see appendix A.3.2).

References

Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35:1798–1828, 2012.

Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Józefowicz, and Samy Bengio. Generating Sentences from a Continuous Space. In *CoNLL*, 2015.

Tian Qi Chen, Xuechen Li, Roger Baker Grosse, and David Kristjanson Duvenaud. Isolating Sources of Disentanglement in Variational Autoencoders. In *ICLR*, 2018.

Cian Eastwood and Christopher K. I. Williams. A Framework for the Quantitative Evaluation of Disentangled Representations. In *ICLR*, 2018.

Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, 2016.

Hyunjik Kim and Andriy Mnih. Disentangling by Factorising. In *ICML*, 2018.

Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. *ArXiv*, abs/1312.6114, 2013.

Abhishek Kumar, Prasanna Sattigeri, and Avinash Balakrishnan. Variational Inference of Disentangled Latent Concepts from Unlabeled Observations. *ArXiv*, abs/1711.00848, 2017.

Francesco Locatello, Stefan Bauer, Mario Lucic, Gunnar Rätsch, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. Challenging Common Assumptions in the Unsupervised Learning of Disentangled Representations. In *RML@ICLR*, 2018.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *ICML*, 2018.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115:211–252, 2014.

Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *ICLR*, 2014.

Raphael Suter, ore Miladinovic, Bernhard Schölkopf, and Stefan Bauer. Robustly Disentangled Causal Mechanisms: Validating Deep Representations for Interventional Robustness. In *ICML*, 2019.

Giorgos Tolias, Ronan Sicre, and Hervé Jégou. Particular object retrieval with integral max-pooling of CNN activations. *ICLR*, 2015.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017.

Muhammad Waleed Gondal, Manuel Wüthrich, Dorde Miladinović, Francesco Locatello, Martin Breidt, Valentin Volchkov, Joel Akpo, Olivier Bachem, Bernhard Schölkopf, and Stefan Bauer. On the Transfer of Inductive Bias from Simulation to the Real World: a New Disentanglement Dataset. *arXiv e-prints*, art. arXiv:1906.03292, Jun 2019.

Appendix A. Further Notes

A.1. Notes on Feature Map Extraction

We experimented with features from pretrained ResNet, ResNeXt, DenseNet and VGG-19 architectures. On *MPI3D-simple*, ResNeXt-101 and VGG-19 outperformed ResNet and DenseNet in terms of the metrics used in the challenge. Between the two of them, we could not clearly detect which architecture works better. On *MPI3D-realistic* (i.e. on the evaluation server), VGG-19 showed better performance based on our limited number of trials, and thus we chose it as our feature extraction network. However, we expect that ResNeXt-101 can also be used given the right kind of hyperparameter settings.

A.2. Notes on Feature Aggregation

We first experimented with simple average-pooling (as usually done in classification architectures) and max-pooling over the feature maps to aggregate the feature maps. This did not result in satisfying performance. We conjecture this is because global pooling loses the information of the spatial locations of objects in the image identifying some of the factors of variations. For example, the degrees of freedom of the robotic arm can easily be derived by the relative positions of object and manipulator.

Compared to global pooling, RMAC enhances the ability of the VAE to infer the factors of variations by better representing the properties of different objects in the image in the aggregated representation. For example, the degrees of freedom of the robotic arm can also be derived by the specific orientation of the manipulator. However, like global pooling, RMAC also does not directly encode the spatial location of objects. An approach to do so could be to overlay a positional encoding onto the feature maps before aggregation, for example similarly to how spatial information is encoded in self-attention mechanisms (Parmar et al., 2018).

We also experimented with PCA-whitening the regionally pooled vectors before summing them up as in the original RMAC formulation (Tolias et al., 2015). We found that this made it harder for the VAE to reconstruct the feature vectors, and thus disentanglement performance suffered.

Finally, instead of hand-designing the aggregation operation, it could also be beneficial to learn the optimal aggregation as part of the VAE training process, e.g. using a transformer-based approach (Vaswani et al., 2017). This was not feasible within the challenge constraints as it would have required to store the full feature map in memory.

Table 1: Summary of scores and ranks of our best submission on the public leaderboard at the end of stage 1. The normalized score is provided to show the performance relative to the best submission and is calculated by dividing the score by the highest achieved score on each metric.

	FactorVAE	**DCI**	**SAP**	**IRS**	**MIG**	
Score	0.8476	0.5364	0.1831	0.5979	0.3473	$\sum 2.5123$
Normalized Score	1	1	1	0.8046	0.8873	$\sum 4.6919$
Rank	1	1	1	26	4	∅ 6.6

A.3. Notes on VAE Training

A.3.1. Number of Latent Factors

The number of latent factors C plays an important role for the performance: if C is chosen too low, the reconstruction error can not be reduced sufficiently whereas if C is chosen to high, there is not enough pressure on the latent bottleneck to disentangle the latent factors; in both cases, performance suffers. Our best model uses $C = 19$. This is considerably higher than the number of latent factors of the MPI3D dataset (i.e. 7), which presumably is because feature vectors encode more information (e.g. about textures) than raw images, and thus a larger latent bottleneck is required to reconstruct the data.

A.3.2. Beta Schedule

The β-schedule in the main text is the one we used for the submission, but does not exactly match the behavior we wanted to achieve due to a late-discovered bug. We actually intended to anneal β from an initial value β_{start} to a final value β_{end} over the N training epochs.

Appendix B. Discussion of Results on the Public Leaderboard

As at the time of writing, results on *MPI3D-real* were not yet released, we can only discuss results from *MPI3D-realistic*. We summarize the results of our best submission on the public leaderboard in table 1. This submission uses slightly different settings than the one described in the main text (which we use as the entry for the private leaderboard), but reaches comparable results. In particular, encoder and decoder have fully-connected layers with 256, 128, and 64 respectively 64, 128, and 256 neurons, the number of latent factors is $C = 18$, the number of training epochs is 20, and the settings for the β-schedule are $\beta_{\text{start}} = 10^{-4}$ and $\beta_{\text{end}} = 0.12 - 10^{-4}$.

Our method achieves the first rank on FactorVAE (Kim and Mnih, 2018), SAP (Kumar et al., 2017) and DCI (Eastwood and Williams, 2018). On FactorVAE, there is a particularly large margin of 0.19 absolute and 29% relative difference to the second ranked method. On MIG (Chen et al., 2018), our method achieves the fourth rank, with 0.044 absolute and 11% relative difference to the best method on this metric. Our method only falls behind on IRS (Suter et al., 2019), where the method is ranked 26th, with 0.145 absolute and 19% relative distance to the best method. In our experiments, there seemed to be a correlation between IRS and the amount of pressure on factorizing the latent factors (i.e. the β value in the loss function). As a consequence, if training collapses and the KLD loss term approaches zero, the IRS can still reach high values. This explains the number of submissions with higher IRS values (but considerably lower scores on the other metrics) than our method. In particular, the default submission has an IRS value of 0.6199, but fails to provide good disentanglement otherwise.

Overall, we think that the results show the potential of our approach. However, as the challenge uses the mean rank as its ranking criterion, our method is penalized by the mediocre rank on IRS and only places fourth overall. Using an alternative ranking criterion which takes into account the numerical distance between the different methods (see table 1), our method places at the first rank (on the public leaderboard).