Unsupervised Orientation Learning Using Autoencoders

Rembert Daems
IDLab-AIRO
Ghent University – imec
Ghent, Belgium
Rembert.Daems@UGent.be

Francis wyffels
IDLab-AIRO
Ghent University – imec
Ghent, Belgium
Francis.wyffels@UGent.be

Abstract

We present a method to learn the orientation of symmetric objects in real-world images in an unsupervised way. Our method explicitly maps in-plane relative rotations to the latent space of an autoencoder, by rotating both in the image domain and latent domain. This is achieved by adding a proposed *crossing loss* to a standard autoencoder training framework which enforces consistency between the image domain and latent domain rotations. This relative representation of rotation is made absolute, by using the symmetry of the observed object, resulting in an unsupervised method to learn the orientation. Furthermore, orientation is disentangled in latent space from other descriptive factors. We apply this method on two real-world datasets: aerial images of planes in the DOTA dataset and images of densely packed honeybees. We empirically show this method can learn orientation using no annotations with high accuracy compared to the same models trained with annotations.

1 Introduction and related work

Annotating real-world images is costly and error prone. Finding the right people, preferably with experience or expert knowledge of the subject data, can be hard and expensive. Retrieving consistent annotations can be difficult due to ambiguities or unclear instructions. Therefore, many researchers are looking towards unsupervised learning. Various methods exist to use large amounts of unlabeled data to learn hidden structures and useful representations for downstream tasks. An autoencoder (AE) [1] encodes information in a latent space by training an encoder-decoder model with a reconstrucion loss on the decoder output. The variational autoencoder (VAE) extends upon this by imposing a given distribution on the latent space, which makes the model more powerful [2, 3]. These methods are commonly directed at finding disentangled representations, where semantically meaningful properties are represented in individual components in the latent space. Higgins et al. [4] modify the VAE by adding a hyperparameter β to balance latent channel capacity and independence constraints with reconstruction accuracy, claiming it leads to higher degrees of disentanglement. Recently, Locatello et al. [5] provided us with a more sobering look on recent progress in this field, showing that unsupervised learning of disentangled representations is fundamentally impossible without inductive bias.

Our work explicitly disentangles orientation with a self-supervised framework without using any annotations. We control the disentanglement explicitly by rotating the images during training and learning a representation of this orientation in latent space. This is closely related to prior work of Worrall et al. [6]. Lenc et al. [7] also transforms the latent space of an autoencoder for local covariant feature detection.

NeurIPS workshop on Differential Geometry meets Deep Learning.

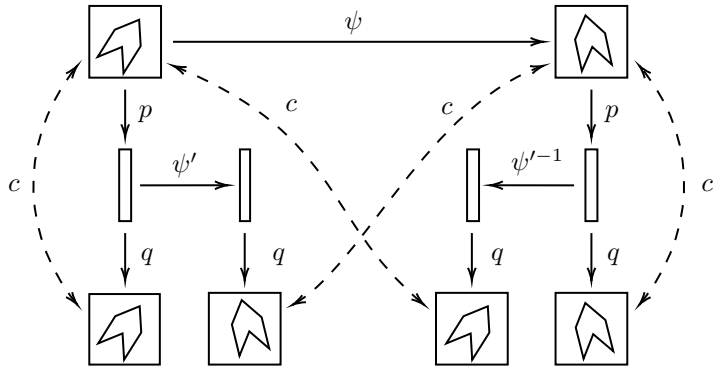


Figure 1: Schematic visualization of $\mathcal{L}_\chi$. $p(x)$ is the encoder function and $q(z)$ the decoder function. $\psi(x, \theta)$ rotates the object in the image domain $\mathcal{X}$ with angle θ and $\psi'(z, \theta)$ rotates the latent representation of the image in the latent domain $\mathcal{Z}$ with the same angle θ. Reconstruction cost functions c are indicated with dashed lines. The standard reconstruction losses to train the autoencoder are indicated on the left and right, acting directly on reconstructions. The crossing lines in the center indicate the proposed $\mathcal{L}_\chi$ loss (hence the name crossing loss), acting on rotated data.

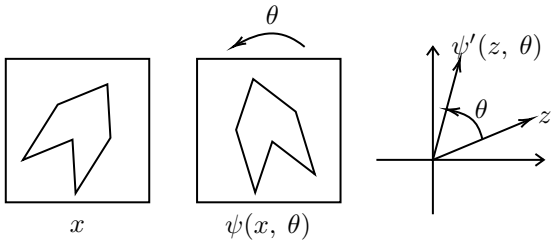


Figure 2: $\psi(x, \theta)$ rotates the image by angle θ, $\psi'(z, \theta)$ rotates the first two components of z by angle θ. Because we explicitly define a one-to-one mapping of θ between $\psi(x, \theta)$ and $\psi'(z, \theta)$, the angle in z directly corresponds to the orientation of the image. In other words, we can unsupervised learn to represent the relative orientation of the image in latent space.

As far as we know, we are the first to use rotations of the latent space to learn orientation in an unsupervised way. We perform experiments on real-world data, with varying and challenging backgrounds (sometimes including parts of other objects).

2 Proposed methods

Our method consists of two steps. First, we extend an autoencoder with explicit learning of a given mapping between a rotation function in the image space $\psi(x, \theta)$ and a rotation function in the latent space $\psi'(z, \theta)$, with the same rotation parameter θ. Second, we propose a method to go from relative rotations to an absolute orientation estimation using object symmetry.

2.1 Crossing Loss $\mathcal{L}_\chi$

Let us consider an autoencoder model with encoder function $p(x) : \mathcal{X} \to \mathcal{Z}$ and decoder function $q(z) : \mathcal{Z} \to \mathcal{X}$. The autoencoder is trained with a cost function $c(x, y) : \mathcal{X} \times \mathcal{X} \to \mathcal{R}_+$, which minimizes the reconstruction error. We then define a rotation function in image space $\psi(x, \theta) : \mathcal{X} \times \mathcal{R} \to \mathcal{X}$ which rotates image x by angle θ, and a counterpart rotation function $\psi'(z, \theta) : \mathcal{Z} \times \mathcal{R} \to \mathcal{Z}$ in latent space.

Our method consists of applying both $\psi(x, \theta)$ and $\psi'(z, \theta)$ in such a way that we can express a crossing loss $\mathcal{L}_\chi$ that enforces an explicit correspondence of $\psi(x, \theta)$ and $\psi'(z, \theta)$ in $\mathcal{X}$ and $\mathcal{Z}$ respectively. Both rotation functions take the same rotation angle $\theta \in \mathcal{R}$ as arguments (Figure 2).

$\mathcal{L}_\chi$ is shown visually in Figure 1 and is defined as:

$$\mathcal{L}_\chi = c[x, (q \circ \psi'^{-1} \circ p \circ \psi)(x, \theta)] + c[(q \circ \psi' \circ p)(x, \theta), \psi(x, \theta)]\,. \tag{1}$$

We also include the standard reconstruction loss over the autoencoder, so that the total loss $\mathcal{L}$ to train the autoencoder with our method is:

$$\mathcal{L} = \mathcal{L}_\chi + c[x, (q \circ p)(x)] + c[\psi(x, \theta), (q \circ p \circ \psi)(x, \theta)] + \mathcal{L}_r \tag{2}$$

where $\mathcal{L}_r$ is a regularization loss depending on the type of autoencoder we choose.

The image rotation function $\psi(x, \theta)$ is implemented as the centered rotation of image x with rotation angle θ, positive in anti-clockwise direction (Figure 2). The corresponding latent space rotation function $\psi'(z, \theta)$ is implemented as a rotation of the first two components of z with angle θ as illustrated in Figure 2. The other $N - 2$ components are not affected by the rotation operation. These components will thus contain information in the image which is rotationally invariant. For clarity, a matrix formulation of $\psi'(z, \theta)$ and $\psi'(z, \theta)^{-1}$ is shown in Appendix A.1.

2.2 Unsupervised estimation of the absolute orientation

In latent space, the orientation is described by the first two components of z: z_0 and z_1. It is represented by angle $\phi = \tan^{-1}(z_1/z_0)$. Because this is only a relative relationship, there is an unknown constant offset θ_0 between the orientation of the object α and the latent angle ϕ: $\alpha = \phi - \theta_0$.

We define orientation angle α for a symmetric object such that if $\alpha = 0$, the symmetry axis of the object coincides with the x-axis. This means that if the image is mirrored along the x-axis, the object is rotated by angle -2α and becomes $-\alpha$ (see Figure 3). We use this property to find θ_0. A full explanation and theoretical derivation is found in Appendix A.2.

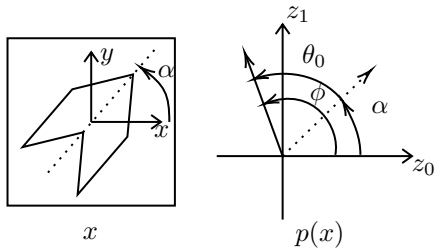


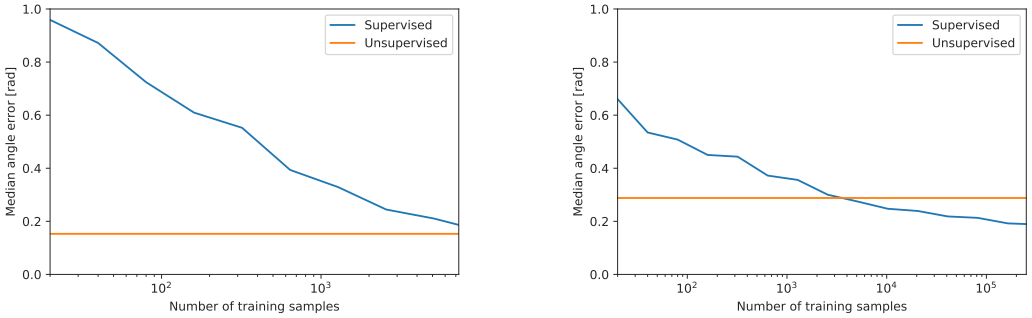
Figure 3: The orientation of the object is described by angle α. If $\alpha = 0$, the object is aligned with the x-axis. z_0 and z_1 are the first two components of $z = p(x)$ describing the orientation in latent space. However, the angle $\phi = \tan^{-1}(z_1/z_0)$ is not equal to α because we never gave the model any information to learn this definition, only relative rotation angles. This means we have to find an offset angle θ_0 so that $\alpha = \phi - \theta_0$.

3 Empirical validation

We performed experiments on two real-world datasets to prove our methods usefulness in real-world applications. DOTA is a large-scale dataset for object detection in aerial images [8]. We extracted images of the plane category and constructed a training set with 7305 images, and a test set with 2234 images. The dataset consists of several types of planes (fighter jets, commercial airplanes, sport planes). This is challenging because the autoencoder needs to generalize the concept of a plane over these different types.

The Honeybee Segmentation and Tracking Datasets [9] contains grayscale annotated images of honeybees in a bee hive. We extracted images of the individual bees and constructed a dataset with 251289 images in the training dataset and 80566 images in the test dataset. Many of the images contain other bees close to the main bee because the bees are densely packed.

See Appendix A.4 for some exemplar images from the datasets.



(a) Planes dataset

(b) Honeybees dataset

Figure 4: Results of the benchmarks: median orientation angle error compared between supervised method (blue line) and unsupervised method (orange line). The horizontal axis shows the number of annotated training images used for the range of supervised benchmarks. The error on the supervised benchmark declines when more annotations are available. The unsupervised benchmark is shown as a horizontal line because it does not depend on annotations. For the planes dataset (a), the supervised error never surpasses the accuracy of the unsupervised method. For the honeybees dataset (b), the error catches up to the unsupervised method at 5120 training samples.

We used a VAE model with $N = 16$ latent dimensions for the planes and $N = 8$ latent dimensions for the bees. By limiting the number of latent dimensions, we found that the model is forced to be efficient in encoding the objects and to reuse concepts of different types in its encodings (e.g. a sport plane is similarly encoded as a fighter jet). Appendix A.5 details the architecture of the VAE models.

As a supervised benchmark, we trained supervised models with the same architecture as the encoder part of the VAE models, with an output vector with two components. The orientation α was directly trained on these two components in the form of $(\cos(\alpha), \sin(\alpha))$ with an L2 loss. This is a fair comparison because the models have the exact same architecture. We varied the number of training samples over an exponential range to compare the strength of the unsupervised model. The VAE models were trained with binary cross-entropy as reconstruction loss and KL divergence loss. All models were trained using ADAM [10] with a learning rate of 10^{-4} and a batch size of 64.

Figure 4 shows the results on the test sets. Median angle error is used as a metric to compare results. A clear trend is visible where the supervised accuracy increases with number of training samples. For the planes dataset, our unsupervised method has a lower error, even when the full trainingset of 7305 images is used to train the supervised benchmark. For the honeybee dataset, our unsupervised method has a lower error when the supervised benchmark is trained with less than 5120 images. Appendix A.3 shows a detailed distribution of the errors of the unsupervised and the supervised results.

We also compare our results on the honeybee dataset with the orginal authors [9]. Bozek et al. report a median angle error of 0.56 radians (32.2 degrees) on the same dataset. Our unsupervised result has a median error of 0.28 radians.

4 Conclusion and further work

We proposed a method to learn in-plane orientation of objects in images without using any labels and experimentally validated it on planes in the DOTA dataset and honeybees in the Honeybee Segmentation and Tracking dataset. The comparison with our supervised benchmark shows impressive results surpassing the accuracy of the supervised model trained on the whole trainingset of 7305 images in the planes dataset, and comparable to the accuracy of the supervised model trained on 5120 annotated images in the honeybees dataset. The benchmark was compared using the median angle error.

Though our method is limited in this work to in-plane orientation in 2D images, we believe its simplicity is worth investigating possible applications on 3D data in future work.

Acknowledgments

Rembert Daems is supported by VLAIO Baekeland Mandate HBC.2018.2102. We sincerely thank Iryna Korshunova for proofreading this paper. We thank the anonymous reviewers whose comments helped improve the manuscript.

References

[1] Geoffrey E Hinton and Ruslan R Salakhutdinov. "Reducing the dimensionality of data with neural networks". In: *science* 313.5786 (2006), pp. 504–507.

[2] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[3] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. "Stochastic backpropagation and approximate inference in deep generative models". In: *arXiv preprint arXiv:1401.4082* (2014).

[4] Irina Higgins et al. "beta-vae: Learning basic visual concepts with a constrained variational framework". In: *International Conference on Learning Representations*. Vol. 3. 2017.

[5] Francesco Locatello et al. "Challenging common assumptions in the unsupervised learning of disentangled representations". In: *arXiv preprint arXiv:1811.12359* (2018).

[6] Daniel E Worrall et al. "Interpretable transformations with encoder-decoder networks". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 5726–5735.

[7] Karel Lenc and Andrea Vedaldi. "Learning covariant feature detectors". In: *European conference on computer vision*. Springer. 2016, pp. 100–117.

[8] Gui-Song Xia et al. "DOTA: A large-scale dataset for object detection in aerial images". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 3974–3983.

[9] Katarzyna Bozek et al. "Towards dense object tracking in a 2D honeybee hive". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4185–4193.

[10] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

A Appendix

A.1 Matrix formulation of $\psi'(z,\theta)$ and $\psi'(z,\theta)^{-1}$

We can express $\psi'(z,\theta)$ and $\psi'(z,\theta)^{-1}$ using a matrix multiplication:

$$\psi'(z,\,\theta) \;=\; \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 & \dots & 0 \\ -\sin(\theta) & \cos(\theta) & 0 & \dots & 0 \\ 0 & 0 & 1 & & 0 \\ \vdots & \vdots & & \ddots & \\ 0 & 0 & 0 & & 1 \end{bmatrix} z \tag{3}$$

and

$$\psi'(z,\,\theta)^{-1} \;=\; \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & \dots & 0 \\ \sin(\theta) & \cos(\theta) & 0 & \dots & 0 \\ 0 & 0 & 1 & & 0 \\ \vdots & \vdots & & \ddots & \\ 0 & 0 & 0 & & 1 \end{bmatrix} z\,. \tag{4}$$

A.2 Using symmetry to find the absolute orientation

During training, the model learned to represent a rotation in the image domain the same way as a rotation in the latent domain. That means if we change orientation α with any angle θ, ϕ will change with the same angle θ. There is however no way that the representation in latent space would learn that $\alpha = \phi$. There is a constant difference θ_0 between α and ϕ, arbitrarily chosen by the model because it was never taught the meaning of $\phi = 0$:

$$\alpha = \phi - \theta_0\,. \tag{5}$$

θ_0 is inherent to the trained model, and is the same for all objects in the dataset, because the model maps the objects to the same latent space. This means we need to find only this one parameter θ_0 to have a correct orientation estimate. θ_0 can be learned from limited annotated data, or unsupervised by using the symmetry of the object. If we define x_m as the mirroring of x around the x-axis, and α and α_m as their respective orientations, we know that:

$$\alpha = -\alpha_m \tag{6}$$

because at $\alpha = 0$ the symmetry axis coincides with the x-axis (see Figure 3).

If we then define ϕ and ϕ_m as the orientation angles in latent domain of respectively x and x_m, we can combine equations (5) and (6) to find an expression for θ_0:

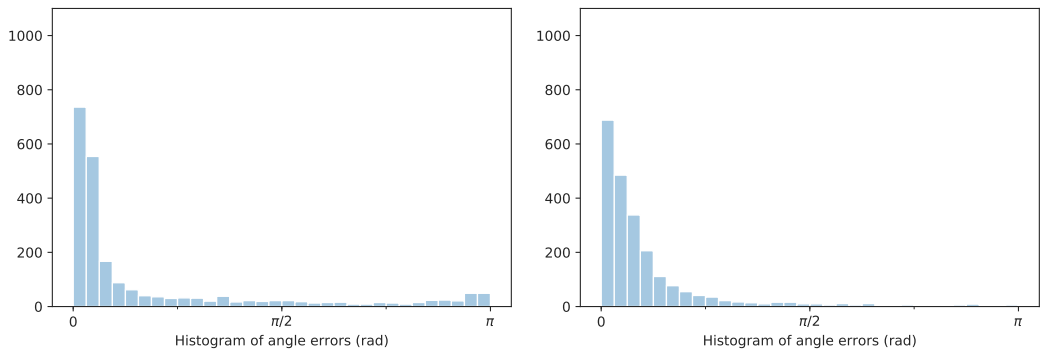
$$\begin{aligned} \alpha &= \phi - \theta_0 \\ \alpha_m &= \phi_m - \theta_0 \\ \theta_0 &= \frac{\phi + \phi_m}{2} \end{aligned}. \tag{7}$$

Since α and α_m are not present anymore, this equation provides us with a straightforward method to learn θ_0 unsupervised from the training data.

It is important to note that this equation has two solutions: θ_0 and $\theta_0 + \pi$. Intuitively, this corresponds to the fact that the arrow in Figure 3 could just as well be pointing in the negative direction of x for these equations to hold. The model has no way of knowing what the semantic meaning is of 'front' or 'back'. Deciding what the forward direction of the object is and correspondingly choosing θ_0 is not possible in an unsupervised manner and is done by hand or by a single binary label. Please note that our method does more than finding the symmetry axis in the image, it does have an internal representation which contains a direction. We only have to define this direction as 'front' or 'back' once (not for every image).

A.3 Detailed error histograms

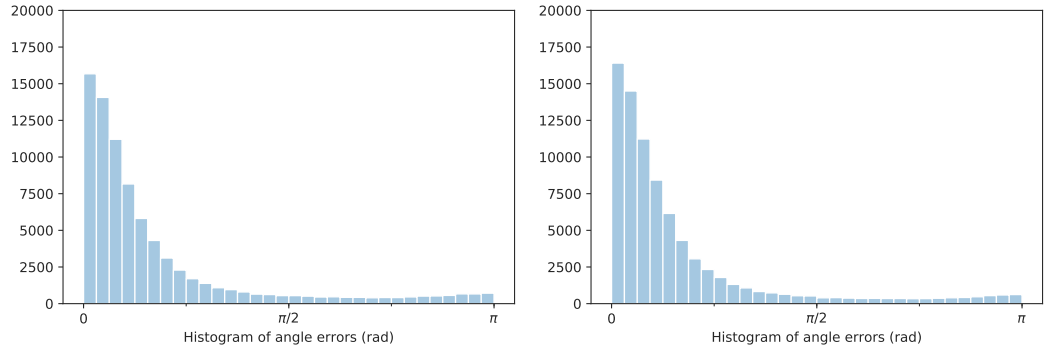
Figures 5 and 6 show detailed histograms of the angle error for the unsupervised results and corresponding supervised results.



(a) Unsupervised

(b) Supervised

Figure 5: Histograms of error distributions of (a) the unsupervised and (b) the supervised result on the planes dataset. The supervised result shown is the result with the full training set of 7305 images.



(a) Unsupervised (b) Supervised

Figure 6: Histograms of error distributions of (a) the unsupervised and (b) the supervised result on the honeybee dataset. The supervised result shown is the result with 5120 training samples wich has comparable median error to the unsupervised result.

A.4 Additional images

Figures 7 and 8 show some image from the training sets.

Figure 7: Some images from the planes training set. Planes are photographed while on the ground. There are several types of planes in the dataset: fighter jets, commercial airplanes, sport planes. This makes the dataset more challenging because the model should encode orientation the same way for all types of planes. The images include background information and sometimes other planes are visible in the same image.

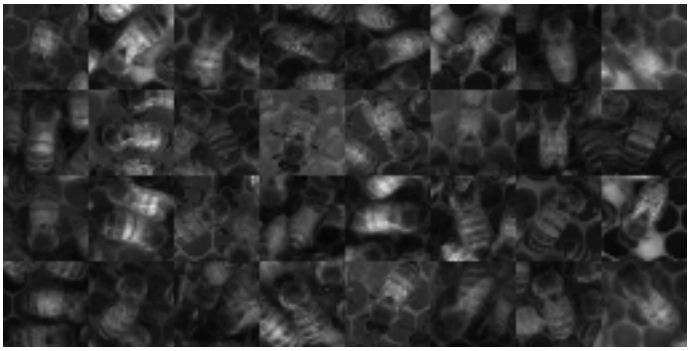


Figure 8: Some images from the training set. Many images contain other bees close to the main bee because the bees are densely packed.

A.5 Model architectures

We present the model architecture in PyTorch code. The supervised model is exactly the same as the encoder, with $N = 2$ outputs. The model has 3 input channels for the planes dataset (RGB) and 1 input channel for the bees dataset (grayscale).

```
from torch import nn


class VAE(nn.Module):
    def __init__(self, num_channels, num_latent_dim):
        super(VAE, self).__init__()
        self.num_latent_dim = num_latent_dim

        self.encoder = nn.Sequential(
            nn.Conv2d(num_channels, 32, 4, 2, 1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.Conv2d(32, 64, 4, 2, 1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.Conv2d(64, 64, 4, 2, 1),
            nn.BatchNorm2d(64),
        )

        self.conv_mu = nn.Conv2d(64, num_latent_dim, 4)
        self.conv_logvar = nn.Conv2d(64, num_latent_dim, 4)

        self.fc_up = nn.Linear(num_latent_dim, 64 * 4 * 4)

        self.decoder = nn.Sequential(
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.ConvTranspose2d(64, 64, 4, 2, 1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.ConvTranspose2d(64, 32, 4, 2, 1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.ConvTranspose2d(32, num_channels, 4, 2, 1),
            nn.Sigmoid(),
        )

    def encode(self, x):
        h = self.encoder(x)
        mu = self.conv_mu(h)
        logvar = self.conv_logvar(h)
        mu = mu[:, :, 0, 0]
        logvar = logvar[:, :, 0, 0]
        return mu, logvar

    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5 * logvar)
        eps = torch.randn_like(std)
        return mu + eps * std

    def decode(self, z):
        h = self.fc_up(z)
        h = h.view(h.shape[0], -1, 4, 4)
        x = self.decoder(h)
        return x
```