# LATENT-MIXUP : mixing latent variables to train robust classifiers

**Tristan Girard  Benjamin Gundersen  Timothé Laborie**

## Abstract

In recent years there has been an impressive increase in the accuracy of Deep Neural Networks for classification such that for some specific tasks these networks achieve an accuracy above the average human level. At the same time, it has been shown that these networks are usually not robust to distribution shifts or adversarial attacks. A few years ago, the MIXUP method was introduced. It involves training a model using convex combinations of training set inputs and their respective labels. This new method increases the model accuracy for some tasks and makes the models more robust. More recently that method has been generalized to mix the activations of the hidden layers inside a network, which lead to even more robust models. In our work we go one step further and do the mixing at the level of latent variables representing the images. We use both Generative Adversarial Networks (GAN) and Variational Auto-Encoders (VAE), we find the latent variables of the input images and train a standard classifier on images generated/decoded from a convex combination of the latent variables of the input images. We will briefly explain how MIXUP and its generalization MANIFOLD MIXUP work. Building on these methods we then describe our approach. Finally, we show that training with our GAN-MIXUP method yields more robust classifiers on certain datasets than training with MANIFOLD MIXUP.

## 1 Introduction

Current Deep Neural Network can be very accurate on the training set and still miss-classify with high confidence samples that slightly differ from the samples seen during training (Ben-David et al., 2010), are underrepresented in the training set (Hashimoto et al., 2018), or are adversarial (Szegedy et al., 2013). Recently, the MIXUP method has been proposed (Zhang et al., 2017) and empirical evidence showed that classifiers trained with that method were more robust compared to models trained the standard way. The idea of MIXUP is to augment the dataset with new images that are convex combinations of two images from the dataset. The label associated with the new image is the convex combination of the labels of the two original images with the same mixing factor. Generalizing that idea lead to MANIFOLD MIXUP where the hidden representations (activation of intermediate layers) inside a neural network are mixed instead of the raw samples. However, the challenge of developing fast training pipelines that also create robust models is still unsolved as even models trained with the methods above can be fooled with specific perturbations.

In order to increase the robustness even further we designed a training approach that builds on the MIXUP idea but goes one step further. Instead of mixing the input features as in MIXUP or the intermediate representations as in MANIFOLD MIXUP, we mix the latent codes of two samples that we got using a generative model. We then use that mixed latent code to generate a new sample that we associate with a label that is a convex combination of the original labels. Different generative models can be used in order to find the latent variables of images, we used GANs (Goodfellow et al., 2014) which lead to the GAN-MIXUP method and VAEs (Kingma & Welling, 2013) which lead to VAE-MIXUP.

### 1.1 MIXUP METHODS

In this section we will briefly describe the MIXUP and MANIFOLD MIXUP methods, building on these algorithms we present GAN-MIXUP and VAE-MIXUP which mix the latent variables using different generative models. We tested our methods on the datasets MNIST, FashionMNIST and CIFAR-10.

#### 1.1.1 MIXUP

The MIXUP training algorithm as presented in (Zhang et al., 2017) is a data augmentation approach used in classification tasks. Pairs of samples are sampled from the dataset and a new data point is generated as a convex combination of the two original samples where the mixing factor $\lambda$ is drawn from a Beta distribution where the two parameters are set to $\alpha$ being a hyper-parameter. The mixed sample is then fed to the classification model which returns logits. The loss associated to that mixed sample is the convex combination of the loss of the logits with respect to the first label and the loss with respect to the second label, where the mixing factor is the same as the one used for the mixing of the inputs.

#### 1.1.2 MANIFOLD MIXUP

Building on the MIXUP idea the MANIFOLD MIXUP algorithm does not necessarily apply the mixing at the level of the inputs, instead a random hidden layer of the classification network is selected, two samples are sampled from the dataset and are fed to the classifier, for each sample we collect the activation of the selected hidden layer as a vector and then do the mixing over these activations vectors. The mixed activations are then fed to the layer coming directly after the selected layer and the network returns logits. The loss for the two samples is computed the same way as for the MIXUP algorithm. If the input layer is selected to be the one where the representations are mixed, then that approach reduces to the standard MIXUP. This is the reason why MANIFOLD MIXUP is a generalization of the former method.

#### 1.1.3 LATENT-MIXUP

Numerical experiments done in the original paper show that training with MIXUP yields models with a higher accuracy and which are more robust compared to models trained the standard way. In fact, very recent research shows that that training technique can be seen as a regularization technique (Carratino et al., 2020). The generalization MANIFOLD MIXUP performs better both in terms of accuracy and robustness (Verma et al., 2018). It is well-know that the deeper a layer is, the more semantics its activation layers have (Zeiler & Fergus, 2014). Coupling that insight with the fact that mixing the representations works better on deeper layer than on the inputs lead to the idea that the mixing should be done on vectors containing an important share of the semantics of the image. Generative models such as GANs (Goodfellow et al., 2014) and VAEs (Kingma & Welling, 2013) represent complex images in a low-dimensional space that summarises the most important aspects of the image. This inspired the decision to perform mixing on latent vectors of images instead of the inputs themselves or the activations of hidden layers. More specifically, we find latent vectors of the images in the training set and train a standard classifier on images generated/decoded from a convex combination of random pairs of latent vectors.

# 2 Models and Methods

## 2.1 GAN-MIXUP

For each dataset, the GAN-MIXUP process requires a GAN and a set of latent vectors that can accurately reconstruct the images from the training set.

#### 2.1.1 MNIST and FashionMNIST

We train a GAN ourselves using a latent dimension of 1024. To invert the train set images, we use the methods described in (Insights) which involves training an initializer model and a feature extractor. We optimize the initial latent vectors using the L2 loss on the extracted visual features using 1000 gradient steps and AdamW (Ilya Loshchilov, 2017) with a learning rate of 0.03. After inverting the images, some of them had gotten stuck in a local minima due to a bad initialization, so we exclude the images where the reconstruction error is below average.

##### 2.1.1.1 TRAINING

For the classifier we use a generic convolution neural network with 575k parameters. When training each of the four methods described above, we use Adam (Diederik P. Kingma, 2014) with 50 epochs, a learning rate of 0.001 and a learning rate decay of 0.9 per epoch.

#### 2.1.2 CIFAR-10

As CIFAR-10 is a larger dataset than MNIST, we chose a StyleGAN2 model pre-trained on CIFAR-10 (Karras et al., 2021). This model has a FID of 2.384 on the CIFAR-10 dataset.

##### 2.1.2.1 StyleGAN Inversion

To perform the inversion we employ the same approach as with the MNIST dataset, but this time in the W space of StyleGAN2. The encoder is a copy of existing code from StyleGAN3-editing (Alaluf et al., 2022) modified to work on the StyleGAN2 model that is pre-trained on CIFAR-10. After training different configurations we decided to use a ReStyle-pSp encoder using a ResNetBackBone, which was then trained for 5 days on a NVIDIA RTX 3090 GPU.

We then encode all 50-thousand training images and fine tune the latent vectors using gradient descent with the latent vectors as parameters and as loss a mixture ($\lambda_{LPIPS} = 0.9, \lambda_{L2} = 0.1$) between LPIPS (Zhang et al., 2018) and L2 of the original and reconstructed images. As optimizer we use AdamW with beta1 equal to $0.95$ and a learning rate of $0.03$. Using 1000 optimization steps, this takes about 30 seconds per image on a NVIDIA GeForce GTX 1080ti GPU. Doing this batched seemed to result in worse inversions, but it seems to be worthwhile to explore methods to do this batched without suffering from worse inversion. We ran this step on multiple GPUs in parallel.

##### 2.1.2.2 Improving the latent vectors

We additionally improve the worst latent vectors. First, we do an additional four thousand steps on the latent vectors which produce normalized images with a mean squared

| Variant | Data Augmentation | Accuracy | DeepFool score | Blurred accuracy |
|---|---|---|---|---|
| Standard | None | 0.8164 (0.0012) | 0.5322 (0.0297) | 0.3449 (0.0108) |
| MIXUP | None | 0.8416 (0.0038) | 0.5602 (0.0308) | 0.3623 (0.0098) |
| MANIFOLD MIXUP | None | **0.8556 (0.0020)** | 0.6409 (0.0271) | 0.3759 (0.0046) |
| GAN-MIXUP | None | 0.8341 (0.0004) | 0.8675 (0.0269) | **0.4424 (0.0119)** |
| GAN-MIXUP (MSE $\geq$ 0.025) | None | 0.8326 (0.0059) | **0.9188 (0.0351)** | 0.4308 (0.0115) |
| GAN-MIXUP (MSE $\geq$ 0.020) | None | 0.8320 (0.0047) | 0.8701 (0.0968) | 0.4236 (0.0181) |
| Standard | Random Flips + AutoAugment | **0.9139 (0.0005)** | 0.5327 (0.0171) | 0.4992 (0.0246) |
| MIXUP | Random Flips + AutoAugment | 0.9135 (0.0044) | 0.8514 (0.0355) | 0.4605 (0.0080) |
| MANIFOLD MIXUP | Random Flips + AutoAugment | **0.9139 (0.0035)** | 0.7107 (0.0224) | 0.4895 (0.0085) |
| GAN-MIXUP | Random Flips + AutoAugment | 0.8910 (0.0029) | 0.9576 (0.0022) | 0.5551 (0.0230) |
| GAN-MIXUP (MSE $\geq$0.025) | Random Flips + AutoAugment | 0.8875 (0.0045) | 0.9701 (0.0522) | 0.5557 (0.0198) |
| GAN-MIXUP (MSE $\geq$0.020) | Random Flips + AutoAugment | 0.8960 (0.0116) | **1.0213 (0.0404)** | **0.5581 (0.0270)** |

Table 1: Results on the CIFAR-10 dataset

error (MSE) greater or equal to $0.025$ in comparison to the normalized original image. Then we do another 4000 steps with MSE greater or equal to $0.02$. The distributions of the MSE between normalized original and inverted images, as well as a few examples of the inversion process can be seen in Figures 1, 2, 3 and 4, 5, 6 in the appendix. The inverted images recover the original images with high fidelity.

#### 2.1.2.3 TRAINING

We use a ResNet18 (He et al., 2015) as a classifier for all CIFAR-10 experiments. As optimizer we used SGD with 270 epochs, a batch size of 32 and an initial learning rate of 0.1 with ReduceLROnPlateau with a factor of 0.1.

### 2.2 VAE-MIXUP

We trained a VAE the standard way on the dataset of interest using the standard loss. Once the VAE converged we augmented the dataset as follows: for random pairs of input images we got latent codes for the two images using the encoder network, we then did the mixing of the latent codes using some mixing factor $\lambda$, which lead to a unique latent code which we passed through the decoder network in order to get an image. This newly generated image is then sent to the classifier to get logits, the loss for that pair of image is then compute the same way as for MIXUP. VAEs are known to generate blurry images, this is problematic as the images generated from the mixing of the latent codes were very blurry and thus negatively affected the accuracy of the trained classifier. One attempt to tackle that issue was to apply a sharpening filter to the images generated by the VAE.

### 2.3 Training

In order to find good hyperparameters for VAE-MIXUP we ran a grid search and selected the hyperparameters with the highest validation accuracy.

## 3 Results

We implemented our methods GAN-MIXUP and VAE-MIXUP using the PyTorch library (Paszke et al., 2019). We compare our methods with three different baselines. The first baseline is to simply train the classifier the standard way, the second baseline is MIXUP, the third baseline is MANIFOLD MIXUP. For each variant we ran a grid-search on the hyperparameters and selected the ones yielding the highest classification accuracy on the validation set. We then reported the classification accuracy on the testing set as well as the DeepFool (Moosavi-Dezfooli et al., 2016) score and the accuracy on images blurred with a Gaussian kernel of size 5 from the testing set.

VAE-MIXUP got an accuracy similar to the baselines, however the DeepFool scores and accuracy on the blurred images were much worse than the baselines on every dataset, thus the result section does not include the results of VAE-MIXUP.

The training time of our method is significantly longer than for the other baselines as we have to find latent codes for the images and to generate a new image from the mixed latent codes. The main complexity overhead lies in that step. On the other hand, the inference is not affected at all as our method is a data augmentation technique which doesn't affect the architecture of the final classifier.

### 3.1 CIFAR-10

We trained models using our method on the CIFAR-10 dataset as described in the previous section and reported the results with and without data augmentation. The data augmentation consists of random horizontal flips and the augmentation framework AutoAugment (Cubuk et al., 2018). In addition to the standard GAN-MIXUP we also included the results when we improved the worst latent codes as described in section 2.1.2.2, these correspond to the entries GAN-MIXUP (MSE $\geq$ 0.02) and GAN-MIXUP (MSE $\geq$

| Variant | Accuracy | DeepFool score | Blurred accuracy |
|---|---|---|---|
| Standard | **0.9921 (0.0015)** | 1.4080 (0.0304) | **0.9343 (0.0115)** |
| MIXUP | 0.9884 (0.0031) | **4.0896 (0.0279)** | 0.6657 (0.0092) |
| MANIFOLD MIXUP | 0.9886 (0.0025) | 3.8588 (0.0286) | 0.8429 (0.0046) |
| GAN-MIXUP | 0.9739 (0.0005) | 0.5328 (0.0271) | 0.7265 (0.0130) |

Table 2: Results on the MNIST dataset

| Variant | Accuracy | DeepFool score | Blurred accuracy |
|---|---|---|---|
| Standard | **0.9252 (0.0016)** | 0.3429 (0.0314) | 0.6727 (0.0123) |
| MIXUP | 0.8883 (0.0034) | **0.9098 (0.0216)** | 0.7163 (0.0087) |
| MANIFOLD MIXUP | 0.9044 (0.0028) | 0.6736 (0.0265) | **0.7175** (0.0037) |
| GAN-MIXUP | 0.9192 (0.0014) | 0.3559 (0.0242) | 0.6769 (0.0135) |

Table 3: Results on the FashionMNIST dataset

0.025) in the table showing the results. The number in the parenthesis is the standard deviation over the different runs, we marked the best results with a bold font. We can read from the result table 1 that MANIFOLD MIXUP yields the best classification accuracy both with and without data augmentation. Our method GAN-MIXUP has an accuracy slightly lower than the MANIFOLD MIXUP but still outperforms the standard training without data augmentation in terms of accuracy. Looking at the robustness measures we see that GAN-MIXUP outperforms the other variants both in terms of DeepFool score and accuracy on the blurred images. Training the classifier on a NVIDIA GeForce RTX 2080ti GPU takes about 30ms per batch for the standard method and 104ms for GAN-MIXUP.

### 3.2 MNIST and FashionMNIST

We trained models using our method on the MNIST and FashionMNIST datasets as described in the previous sections. On FashionMNIST, we set it up so that half of the data comes from original images, as we found it improves all of the metrics. We can read from the result table 2 that the standard training procedure yields the best classification accuracy. Our method GAN-MIXUP did not improve the robustness on either dataset compared to MANIFOLD-MIXUP, but it does get a better accuracy than both other Mixup methods on FashionMNIST. Looking at the robustness measures we see that MIXUP outperforms the other variants in terms of DeepFool score.

## 4 Discussion

In this research, we present a method for improving the adversarial robustness of a classifier using GAN and VAE interpolation. Our experiments show that the GAN-Mixup method was effective on the CIFAR-10 dataset, where it is able to significantly improve the classifier's robustness against adversarial attacks as measured by DeepFool, while keeping the accuracy comparable or a bit lower than the standard training procedure. However, when we tried to apply a similar method on the MNIST and FashionMNIST datasets, we did not see the same improvement.

One possible explanation for this difference in performance is that the StyleGAN2 model trained on CIFAR-10 is a lot more complex and fits the data distribution closer. The StyleGAN2 model we used has a FID of 2.384. Another explanation is that we spent far more computational resources to invert the CIFAR-10 GAN and retrieve the latent vectors. It might also be the case that the StyleGAN2 latent space is more well behaved for interpolation. When we plotted some interpolated images from the MNIST GAN, most of them just looked like one of the original labels.

The VAE-Mixup method performed worse in every situation. This may be because the VAEs that we used have a much lower model complexity than StyleGAN2.

Overall, our results suggest that GAN interpolation is a promising method for improving adversarial robustness, but more work is needed to understand its limitations and to explore other ways of improving the robustness of classifiers to adversarial attacks.

## 5 Summary

Our experiments show that the GAN interpolation method was effective on the CIFAR-10 dataset. From this, we conclude that GAN-MIXUP is a promising method to improve robustness. This method is computationally demanding and so is the inversion. Implementing and comparing different methods of inversion is a time consuming endeavour.

In conclusion, GAN-MIXUP is a promising method to provide robustness against adversarial attacks and potentially improves the accuracy if the latent vectors are of high quality. This improvement comes with a cost in computational time, complexity of code and the time needed to compare and come up with ways on how to invert the images.

# References


Alaluf, Y., Patashnik, O., Wu, Z., Zamir, A., Shechtman, E., Lischinski, D., and Cohen-Or, D. Third time's the charm? image and video editing with stylegan3, 2022. URL https://github.com/yuval-alaluf/stylegan3-editing.

Ben-David, S., Blitzer, J., Crammer, K., Kulesza, A., Pereira, F., and Vaughan, J. W. A theory of learning from different domains. *Machine learning*, 79(1):151–175, 2010.

Carratino, L., Cissé, M., Jenatton, R., and Vert, J. On mixup regularization. *CoRR*, abs/2006.06049, 2020. URL https://arxiv.org/abs/2006.06049.

Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., and Le, Q. V. Autoaugment: Learning augmentation policies from data, 2018. URL https://arxiv.org/abs/1805.09501.

Diederik P. Kingma, J. B. Adam: A method for stochastic optimization, 2014. URL https://arxiv.org/abs/1412.6980.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial networks, 2014. URL https://arxiv.org/abs/1406.2661.

Hashimoto, T., Srivastava, M., Namkoong, H., and Liang, P. Fairness without demographics in repeated loss minimization. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1929–1938. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/hashimoto18a.html.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition, 2015. URL https://arxiv.org/abs/1512.03385.

Ilya Loshchilov, F. H. Decoupled weight decay regularization, 2017. URL https://arxiv.org/abs/1711.05101.

Insights, A. Editing faces using artificial intelligence. URL https://youtu.be/dCKbRCUyop8?t=938.

Karras, T., Aittala, M., Laine, S., Härkönen, E., Hellsten, J., Lehtinen, J., and Aila, T. Alias-free generative adversarial networks. In *Proc. NeurIPS*, 2021.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes, 2013. URL https://arxiv.org/abs/1312.6114.

Moosavi-Dezfooli, S.-M., Fawzi, A., and Frossard, P. Deepfool: A simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks, 2013. URL https://arxiv.org/abs/1312.6199.

Verma, V., Lamb, A., Beckham, C., Najafi, A., Mitliagkas, I., Courville, A., Lopez-Paz, D., and Bengio, Y. Manifold mixup: Better representations by interpolating hidden states, 2018. URL https://arxiv.org/abs/1806.05236.

Zeiler, M. D. and Fergus, R. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pp. 818–833. Springer, 2014.

Zhang, H., Cisse, M., Dauphin, Y. N., and Lopez-Paz, D. mixup: Beyond empirical risk minimization, 2017. URL https://arxiv.org/abs/1710.09412.

Zhang, R., Isola, P., Efros, A. A., Shechtman, E., and Wang, O. The unreasonable effectiveness of deep features as a perceptual metric, 2018. URL https://arxiv.org/abs/1801.03924.

# A MSE of latent vectors

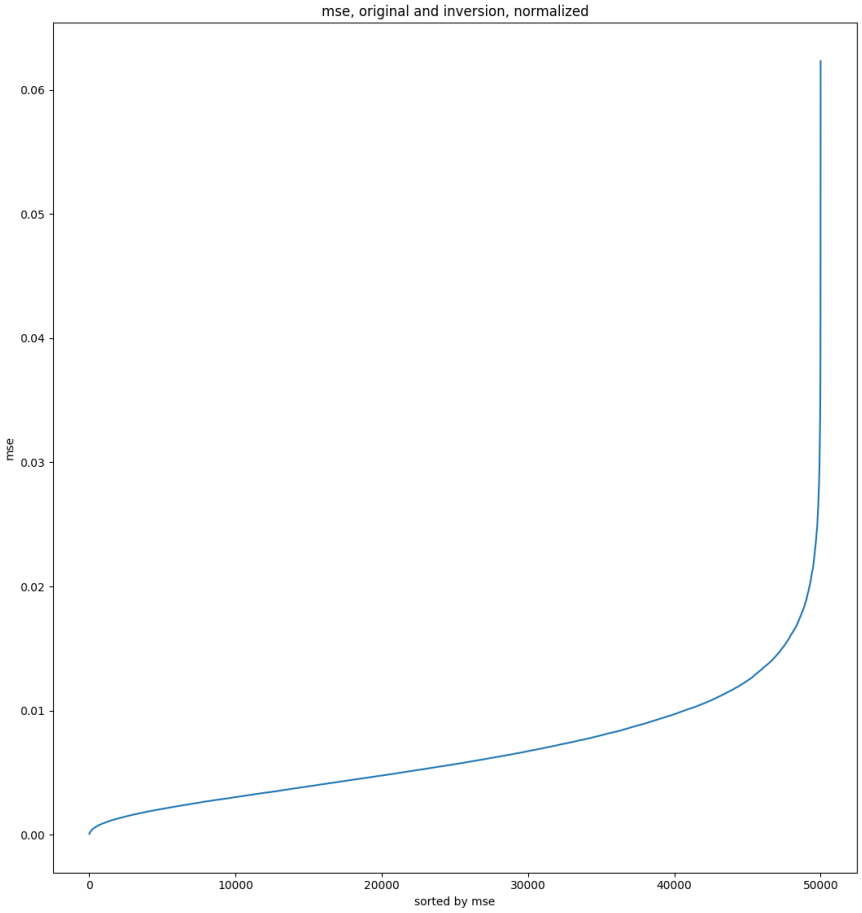


Figure 1: MSE of latent vectors, no additional steps

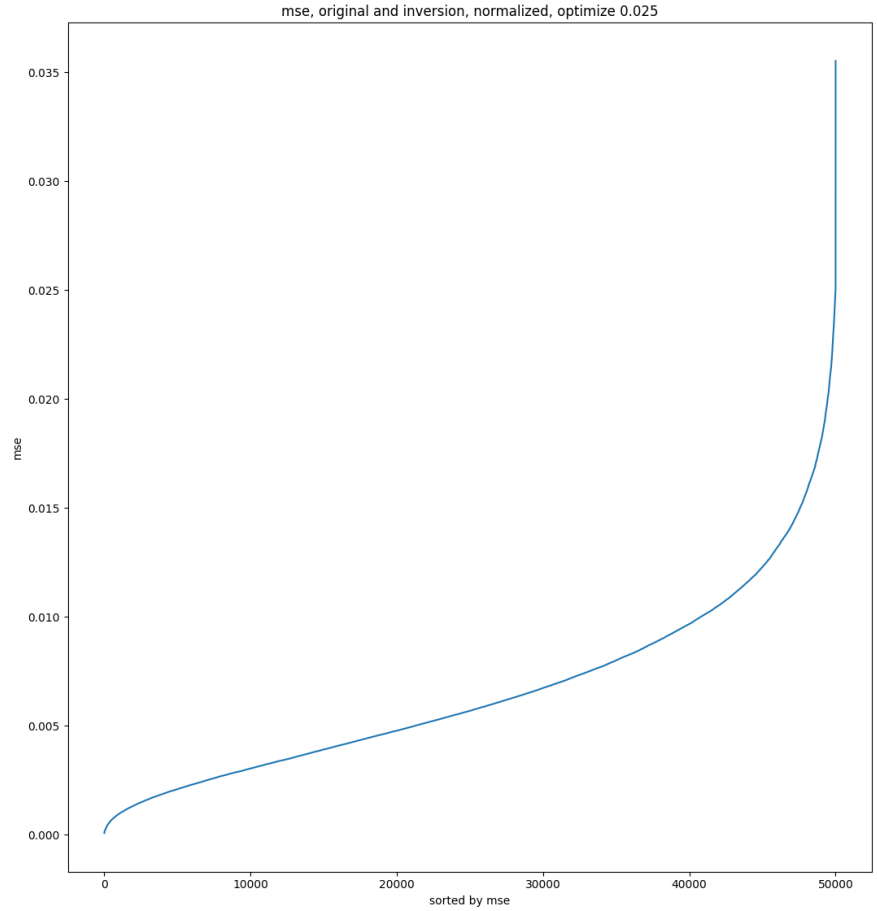


Figure 2: MSE of latent vectors, additional steps for $0.025$

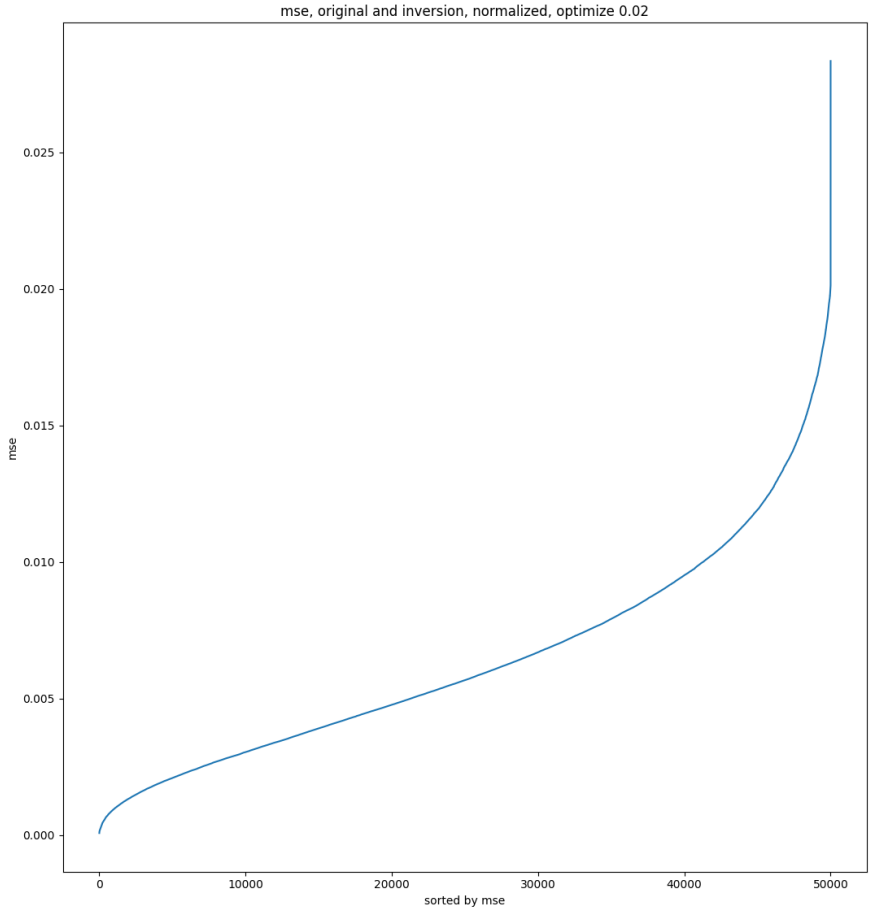


Figure 3: MSE of latent vectors, additional steps for 0.02

# B CIFAR-10 Inversion Examples

Figure 4: original image (left), inverted image (center), encoded image (right)

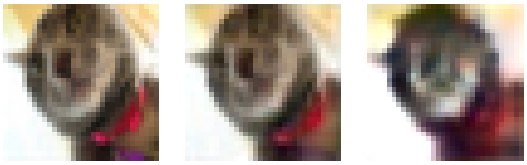

Figure 5: original image (left), inverted image (center), encoded image (right)

Figure 6: original image (left), inverted image (center), encoded image (right)