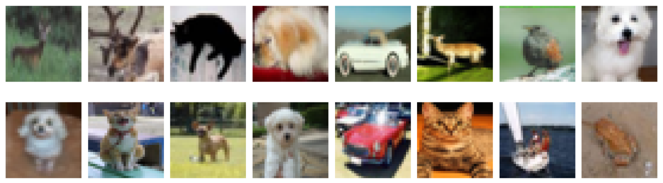
Lab 10 : Predicting CIFAR10 using VAEs

Pau Hidalgo Pujol - APRNS @ GIA-UPC

December 19, 2024



1 Introduction

Variational Autoencoders (VAEs) [1] are a type of neural network architecture that combine the principles of deep learning and probabilistic modeling. By learning a probabilistic latent space, VAEs enable tasks such as image generation, data compression and representation learning.

In this lab, we will use a pretrained VAE from the *Stable Diffusion 1.5* model, accessible through Huggingface [2] and the *diffusers* [3] library. This KL AutoEncoder has been trained on a large-scale dataset, making it versatile enough to encode and decode any type of images (and it also admits arbitrary sizes).

Our focus will be on analyzing the representations produced by this VAE on the CIFAR10 datset [4]. We want to analyze if the latent distribution encodes sufficient information to achieve good classifications on this dataset. Usually VAEs are used the inverse way: they serve as the final step for generation of images, and are evaluated more on reconstruction than on dimensionality reduction.

2 Methodology

The CIFAR10 dataset, obtained using the torchvision library, contains 60.000 images across 10 classes (50.000 allocated for training). Figure1 shows an example of each class. The names are: "airplane", "automobile", "bird", "cat", "deer", "dog", "frog", "horse", "ship", "truck".

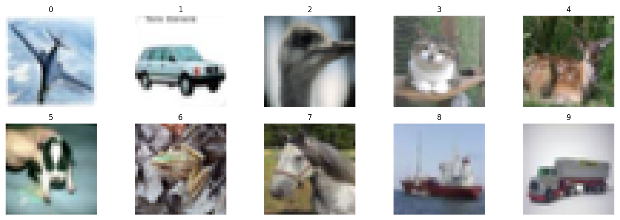


Figure 1: Image example for each class

To apply the pretrained Variational Autoencoder (VAE) to this dataset, we followed these steps:

- Initialize the pretrained VAE from diffusers
- Pass each CIFAR10 image through the VAE encoder using the .encode()
- Convert the latent tensors to NumPy arrays

When doing *.encode()*, the output was of class *AutoencoderKLOutput*, which contained a *.latent_dist*. From that distribution, there were two possible ways to obtain a latent: performing a *.sample()*, effectively sampling from the learned distribution; or saving both *.mean* and *.std*, the parameters of the normal distribution. We performed experiments with both and compared the results.

The latent representations from the VAE were used to train different classification models from the scikit-learn library [**?**]. Specifically, we tried:

- Logistic REgression
- Random Forest
- KNN (with K=25)
- SVC (Support Vector Classifier)
- HistGradientBoosting

Additionally, we also tried LightGBM [5] and XGBoost [6], two boosting algorithms which usually obtain great results.

Each model was trained using the transformed dataset, and performance was evaluated using standard metrics such as accuracy, precision, recall, and F1-score, as provided by the classification report in scikit-learn.

Our goal was to also gain further insights into the latent representations. To do so, we tried to represent them in a two-dimensional PCA [7] (coloring by class) and analyzed each of the 4 features distribution by class.

3 Results

We can observe the results of the different classifiers on 1.

Model	**Accuracy .sample()**	**Accuracy mean & std**
Logistic Regression	0.44	0.46
Random Forest	0.47	0.44
KNN (n=25)	0.53	0.53
SVM	0.61	0.61
HistGradientBoosting	0.57	0.57
LightGBM	0.57	0.58
XGBoost	0.55	0.56
Random predictor	0.10	0.10

Table 1: Model Accuracy Comparison

As we can see, there really isn't a big difference between using .sample() versus mean and std. The latter one sometimes yields slightly better results, but that comes at the expense of double the features, which makes using .sample() a better option.

Observing other scores like precision, recall, and f1-score, we see that the values are very similar to the accuracy. That means that the models aren't really overfitting to one class, but more or less predicting correctly all of them. To assess this claim, we can look at the confusion matrix 2

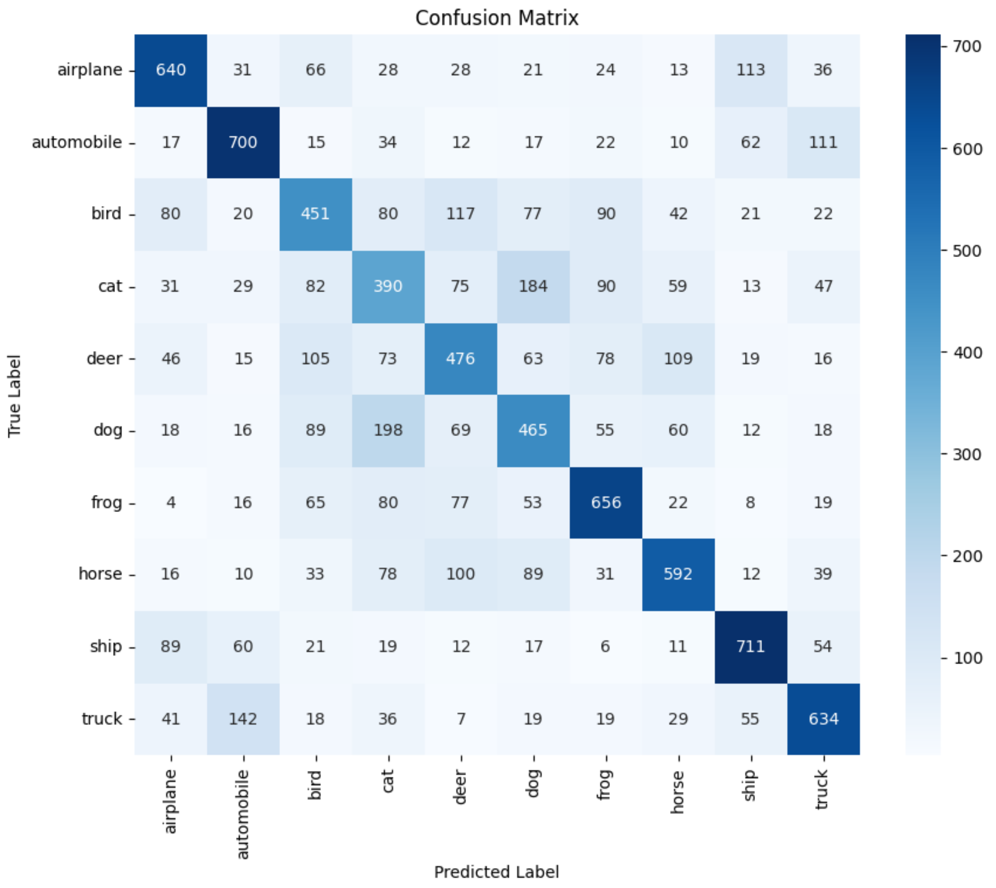


Figure 2: Confusion matrix of XGBoost

While this particular confusion matrix is for the XGBoost model, the ones from other models were very similar. In general, they are able to predict correctly most of the classes. The ones they struggle the most with are cat and dog, often predicting cat when it really is a dog. That makes sense, since usually cats and dogs aren't really that different.

Another two classes which the models often confuse are automobiles and trucks, which again, are pretty similar.

This accuracy results seem to indicate that, even though we encoded 3x32x32 images into 4x4x4 latents, those are good enough to retain information about the class of the object.

We can also try to observe a bit the latents and what do they represent.

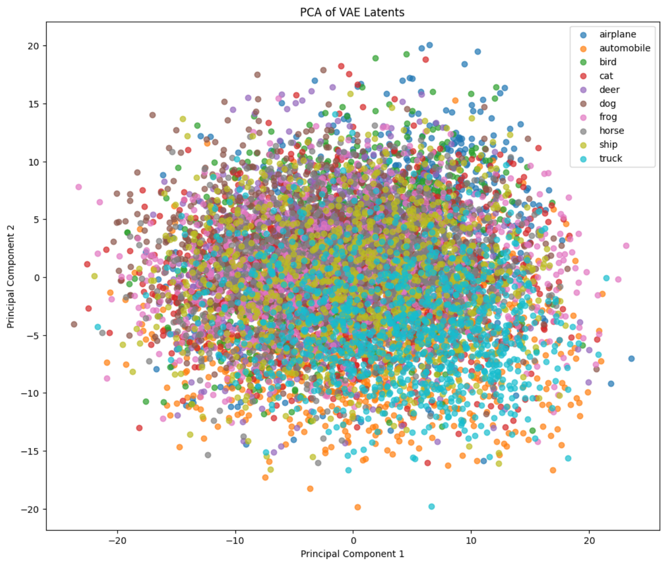


Figure 3: PCA of the latents

If this PCA 3 tells us one thing, it is the fact that most of the classes are very similar, since all of them are on top of each other. We can't really observe a clear difference between classes. This explains why, even though we were able to get a 60% accuracy, the models struggled to get better results, since many of the features aren't clearly separable (at least linearly).

We can also show the KDE plot (similar to an histogram) of some of the 128 features we have (64 in the sample version) in figure 4.

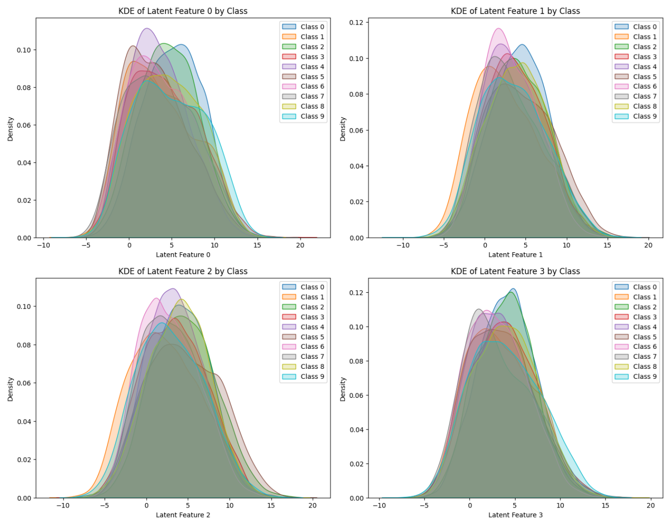


Figure 4: KDE of the first 4 features

Looking at this distributions, there seem to really exist some differences between classes. For example, class 6 tens to have lower values in all of the 4 features, while class 8 usually has higher values. This shows that there really are differences in the latents of different classes.

We can also compare different encoders: in fact, one can use, from the same family of models, sd-vae-ft-mse (instead of sd-vae-ft-ema), which is more widely used.

They are trained on the same basis VAE, but fine-tuned slightly differently. When comparing training classification models on features by each model, we found no significant differences (since both VAEs are pretty much the same).

We also tried stabilityai/sdxl-vae, which follows the same architecture but in theory was trained better. However, when using this model, we found worse results. This seems to be due to some kind of problem with how we used the model, which was further confirmed when observing the reconstructions 5. We weren't able to find the cause of the error.

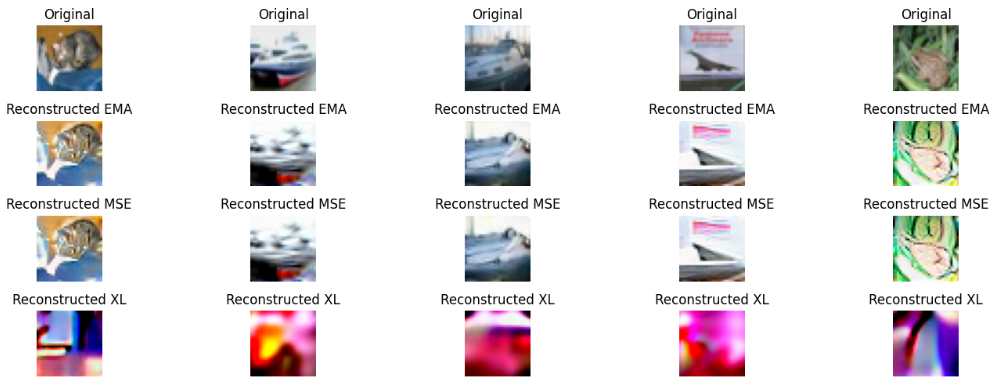


Figure 5: Reconstructions from VAEs

4 Conclusions

We were able to show that VAEs demonstrate a strong ability to retain meaningful latent representations, even if the dimensionality reduction is extreme. Despite the drastic reduction in data, we were able to achieve a reasonable classification accuracy.

It is important to note two things: first of all, the variational auto-encoder was trained on an entirely different dataset, which shows its great capacity to generalize. Secondly, even though we got an accuracy of 61%, this is still a lot less than what one could get by using the entire image with a CNN (in https://paperswithcode.com/sota/image-classification-on-cifar-10, we see that many models obtain +90% accuracy on the same benchmark). The PCA visualization of the latent space illustrated significant overlap between classes, explaining the models' struggles to achieve higher accuracy. This overlap suggests that while the VAE encodes meaningful information, the latent features for some classes (e.g., cats vs. dogs or automobiles vs. trucks) remain too similar for robust separability.

Observing the reconstructions confirmed that the VAE effectively captures the primary structure of images and the main idea, but occasionally loses finer details (as expected, due to the dimensionality reduction).

Overall, we were able to use the VAE to effectively reduce the dimensionality of our input images and perform classification, which was our goal. So we can conclude that Variational AutoEncoders are much more than simple generative models, and their latents encode rich semantic information that we can use to make predictions.

References

[1] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” *arXiv preprint arXiv:1312.6114*, 2013.

[2] H. Face, “Hugging face transformers,” 2020.
urlhttps://huggingface.co/stabilityai/sd-vae-ft-ema.

[3] H. Face, “Diffusers: State-of-the-art diffusion models in pytorch,” 2022.
urlhttps://github.com/huggingface/diffusers.

[4] A. Krizhevsky, “Learning multiple layers of features from tiny images,” 2009. Technical report, University of Toronto.

[5] Microsoft, “Lightgbm: A highly efficient gradient boosting decision tree.” `https://lightgbm.readthedocs.io/`, 2017. Accessed: 2024-12-19.

[6] T. Chen and C. Guestrin, “Xgboost: A scalable tree boosting system.” `https://xgboost.readthedocs.io/`, 2016. Accessed: 2024-12-19.

[7] I. T. Jolliffe, *Principal Component Analysis*. New York: Springer, 2002.