Variational Autoencoder Classifiers

Ashwin Jeyaseelan
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
ajeyaseelan3@gatech.edu

Tarun Pasumarthi
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
tpasumarthi3@gatech.edu

Abstract

Variational autoencoders (VAE) [4] have become popular in recent times as generative models. More interestingly, they learn to model the distribution of a set of features of observations in a compressed latent space. Given that VAEs have been demonstrated to work well with reconstructing images and modeling them in a latent space, we hypothesized that pretrained VAEs could also be effectively used to learn new information such as classification. Specifically, we use pretrained VAEs to train classifiers and have demonstrated decent results on the MNIST, Quick Draw [3], and Fashion MNIST datasets.

1. Introduction

Typically, classification of images is done by training a deep convolutional network directly on preprocessed images. This can be time-consuming as today's networks depend on thousands to millions of paremeters. Instead of learning the feature masks for images, we formulate an efficient way to train classifiers by training them to make predictions based on encoded latent features of the images.

We used pretrained VAEs to train classifiers. After training a VAE on a specific dataset, we save the weights. Then we train a classifier that learns to predict the correct class of an input image of the same dataset based on the latent representation of that image that has been encoded by the pretrained VAE.

We find that the latent features of autoencoders can be used as effective features to learn information from. In our case, we demonstrate that latent features, or rather the encoded latent representations of input images, serve as useful enough information to efficiently train relatively small classifiers. The only drawback is that a pretrained encoder on the dataset will be required in order to train the classifier.

2. Approach

We conducted experiments on three different datasets, MNIST, quickdraw, and fashion MNIST to test our method. All our models were implemented using the Pytorch framework.

Initially, we tried to test a hybrid model between a VAE and a classifier where the decoder is swapped with a classifier instead. The final loss was modified to be the KL divergence for penalizing the distribution errors and the classification cross entropy error instead of the reconstruction loss. Unfortunately, the hybrid model's training stagnated within a few epochs and didn't converge even after hundreds of epochs. We expected this approach to be straightforward, but further work may have to be done on it to see where we went wrong or if we made any implementation errors.

Next, we decided that it would be easier to train VAEs on datasets first and then use those pretrained models as latent feature extractors that could then be fed as input for the classifiers to train on. We separately trained VAEs on each of the three datasets, with slightly different architectures.

Instead of training traditional VAEs, we used a modified version known as beta-VAE [2], which has a higher penalty for the KL divergence. Thus, the beta-VAE loss becomes:

$$L(\theta, \phi; x, z, \beta) =$$
$$\mathbb{E}_{q_\phi(z|x)}[\log p_\phi(x|z)] - \beta D_{KL}(q_\phi(z|x)||p(z)) \quad [2]$$

where θ refers to the weights of the network, and β is a hyperparameter that enforces a constraint on the KL divergence. A beta value of 1 would be the original VAE loss. We used a beta value of 3 for our VAEs. Our reasoning for using a beta-VAE was that We wanted to use a simple VAE model that tries to learn disentangled representations, such that the features of a latent vector are independent from one another and are therefore more interpretable. This in turn would make it easier for our classifiers to interpret and train on the encoded latent vectors.

After training the VAEs, we trained the classifiers using the pretrained VAEs using the following steps:

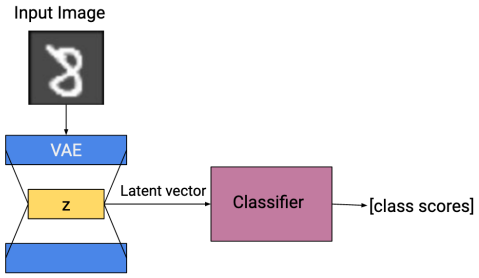


Figure 1: Shown above is the process for feeding data into a classifier. First the input image is fed into the VAE, and then the sampled latent vector from the encoder is fed as input into the classifier, which outputs it predicted class scores.

1. Given input image x, we first feed x into the pretrained VAE and retrieve the encoded representation z, which is sampled from the latent distribution of the class that x belongs to.

2. Then z is fed into the classifier, which outputs its scores for each class, with the max argument score being the predicted class.

3. When training, the predicted output is compared to the label. We use cross entropy as our loss function for all our classifiers. Backpropagation occurs as normal, but the weights of encoder of the VAE that's being used as a feature extractor are kept frozen.

3. Experiments and Results

3.1. MNIST

For our first experiment we decided to test it on the MNIST dataset, which is comprised of images of numbers from 10 classes. We felt like this would be a good starting point for testing the possibility of our approach since the dataset is fairly easy to train on.

Initially, we tried to train a hybrid VAE model as discussed in the approach. After that didn't work, we decided to use a pretrained VAE model online. We used a pretrained VAE model that was only made up of fully connected layers. The model was simple enough for us to quickly test our idea. Since the pretrained VAE's reconstructed images were very blurry, our classifiers were only able to reach up to 84% accuracy. Afterwards, we decided to train out own beta-VAEs on the data. [6]

The encoder of our MNIST beta-VAE is comprised of 3 convolutional layers that output 32 feature maps, 64 feature maps, and 64 feature maps respectively, followed by two fully connected nets that compute the mean and corresponding standard deviation of each of the latent features.

(a) sample test images

(b) beta-VAE reconstructed images

Figure 2: beta-MNIST VAE performance

All three convolutional layers have kernels of size 4, with zero-padding of size 1. The first two kernels have a stride of 2 and the last kernel has a stride of 1. The latent dimension is of size 20, meaning that the VAE learns the means and distributions of 20 latent features. The decoder is identical to the encoder. It is composed of 3 transposed convolutional layers that follow the same design as the encoder convolutional layers in the opposite order. We also put batch normalization between all layers for improved performance.

The VAE was trained for 80 epochs. All MNIST models used the same data split of 60000 training images and 10000 test images. It's performance on reconstructing a set of samples in the test image set can be seen in Figure 2.

After training the VAE, we instantiated a random train and test MNIST dataset with the same split as mentioned above for our classifier. We were able to train a small classifier with only two fully connected layers (first layer of 15 units and second layer of 10 units) and get a test accuracy of 96% within 6 epochs of training.

To get better performance, we added three more fully connected layers for a total of 5 layers, with batch norm in between each layer. We used the SELU activation function [5] on the output of each layer. We were able to reach a test accuracy of 97% within 8 epochs of training. It's performance on each of the 10 classes can be seen in Figure 3.

3.2. Quick Draw

Upon seeing our classifiers train seamlessly on the MNIST dataset, we decided to test our idea on more rigorous datasets. The quick draw dataset was appealing because it had vector based images instead of fully pixelized images, and we thought it would also be a fun challenge to design a beta-VAE capable of generating sketches. We were curious

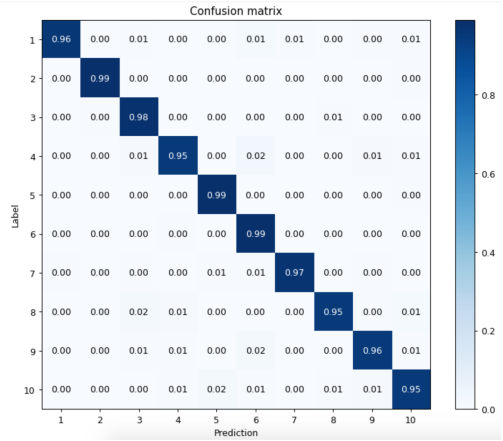
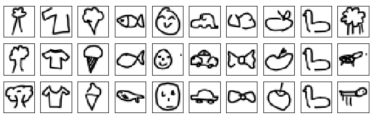


Figure 3: Shown above is the confusion matrix of our MNIST classifier. Its performance on each of the 10 classes can be seen. The numbers on the y-axis are the true labels and the numbers on the x-axis are our classifier's predicted labels.



(a) sample test images

(b) beta-VAE reconstructed images

Figure 4: beta-Quick Draw VAE performance

if latent vectors of sketches would provide useful enough information about the class of images that a sketch belongs to.

The quick draw dataset has 50 million drawings and 345 categories, but we decided to constrain the number of classes to 10 classes, which were tree, t-shirt, ice cream, fish, face, car, bowtie, apple, flamingo, and sheep. Since the VAE based Sketch-RNN model by Ha et. al [1] (which was the first sketch VAE to be trained on vector stroke based data (on the quick draw dataset) wasn't very successful in generating sketches when trained on multiple categories, we decided to limit the number of classes we'd train on to 10.

Our beta-VAE on the quick draw dataset has the same architecture as the MNIST beta-VAE except for an extra final convolutional and deconvolutional layer with kernel size 3, stride of 1, and zero padding of 1 in the encoder and decoder. We increased our latent dimension to 30 to allow our VAE to learn more latent attributes. We trained the model

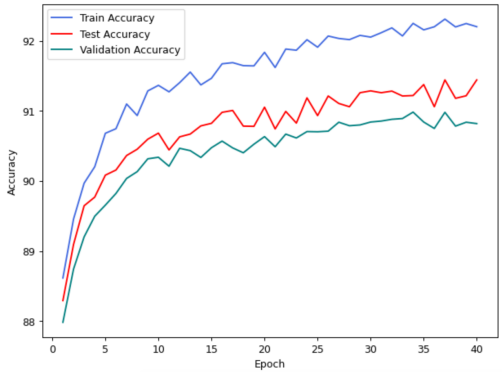


Figure 5: Shown above is the performance of our classifier in terms of accuracy with the pretrained quickdraw beta-VAE on the quick draw dataset

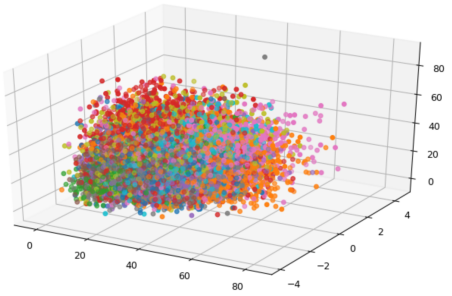


Figure 6: 3D Visualization of 3 of our beta VAE's latent attributes across all 10 classes.

for 456 epochs. It's performance on reconstructing a set of sample test image is shown in Figure 4. Clearly, there's room for improvement in the VAE, as the reconstructed images are more pixelated or have denser strokes than the original images. At the minimum, it can at least be seen what the reconstructed images represent.

To make sure that our encoder was properly learning the latent distributions of the data, we visualized 3 randomly selected latent attributes across all 10 classes. The beta-vae constraints the distributions of the attributes so they end up being close together. Fortunately, we can see some separation of the 10 classes with only 3 of the 30 attributes being visualized. For instance, the red dots are clustered together, while the green dots are clustered below them, and the orange dots are clustered on the right side.

Next, we trained our classifier with the same architecture as the 5 layer classifier that was used on the MNIST dataset. Once again, we initialized a train, validation, and test set of images belonging to the 10 classes of interest, which were randomly selected and shuffled prior to training. Our classi-

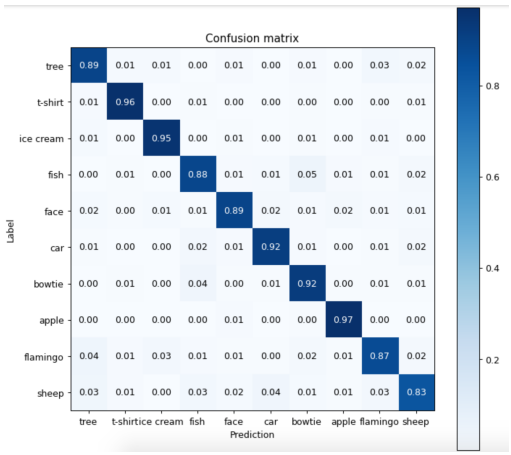
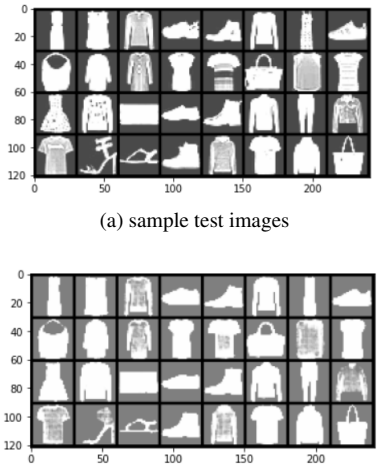


Figure 7: Shown above is the confusion matrix of our quick draw classifier. Its performance on each of the 10 classes can be seen. The numbers on the y-axis are the true labels and the numbers on the x-axis are our classifier's predicted labels.



(a) sample test images

(b) beta-VAE reconstructed images

Figure 8: beta-MNIST VAE performance

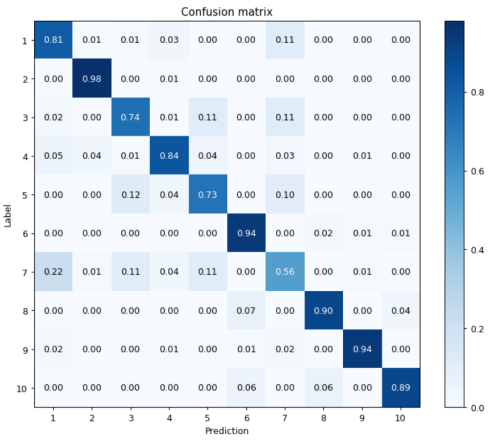


Figure 9: Shown above is the confusion matrix of our Fashion MNIST classifier.

fier reached a test accuracy of 91% at 9 epochs of training. Training further up to 40 epochs did not improve the performance, as shown in Figure 5.

From looking at the confusion matrix, our classifier performed well with predicting images of most of the classes with a minimum prediction accuracy of 87%, except for the sheep class where it got an accuracy of 83%. Considering that our quick draw beta-VAE still has a lot of fine tuning to do and only has a latent dimension of size 30, it was pretty impressive to see how our classifier was able to learn from the encoded latent representation of the quickdraw images. To make things more difficult, many of the quickdraw images are noisy. Nonetheless, our classifier was able to clearly learn from the latent codes of the quick draw dataset.

3.3. Fashion MNIST

We wanted to see if we could replicate the MNIST results with the same architecture on other datasets, so we decided to use the Fashion MNIST dataset due to its similarity to MNIST. Like MNIST, Fashion MNIST consists of 28 by 28 images of 10 classes (T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot) and contains the same amounts of train and test data (60,000 training and 10,000 test).

Using the same beta-VAE architecture, the model was trained for 80 epochs, and its performance on reconstructing a set of samples in the test image set can be seen in Figure 2. After training the VAE, we trained our classifier and got a test accuracy of 83% within 5 epochs of training. The resulting confusion matrix can be seen in Figure 4.

Unlike the MNSIT and Quick draw datasets, the result of the for Fashion MNIST dataset was 10% lower than state of the art benchmarks. Although we initially assumed that this was due to the latent vectors not being able to accurately represent more complex- real world images compared to those presented in the other two datasets, Figure 2 shows that the images learned by the VAE were fairly blurry, indicating that the problem might be due to the MNSIT architecture not being perfectly compatible with the Fashion MNIST dataset. Therefore, we predict that state of the art accuracy from our latent vector classification is possible with more fine-tuning of the hyperparameters and the architecture. Even so, the 83% accuracy does indicate good performance, especially given quick convergence and the training time of just 195 seconds over 10 epochs, which is a significant speed upgrade over most deep classifiers.

	Accurcay	CE Loss	Training time
Quickdraw	0.91	0.28	210s
MNIST	0.96	0.12	164s
FashionMNIST	0.83	0.44	195s

Table 1: Evaluation metrics

4. Discussion and Conclusion

In this paper we introduce a novel approach to image classification using latent vector representation of images. Unlike typical convolutional neural network classifiers that utilize the entire images as vectors, we first train our beta-VAE to learn these latent vectors over our dataset and then feed them into our simple two layer feed forward classifier.

Using this pipeline, we evaluated our approach on three datasets, the MNIST, Quick Draw, and Fashion MNIST, and got 96%, 91%, and 84% testing accuracies respectively. Although these results fall just shy of state of the art benchmarks[5], they offer significant speed upgrades as the validation accuracies converge within 10 epochs and less than 300 seconds of training. Therefore we can conclude that our latent vector approach was indeed successful is effective for image classification.

References

[1] David Ha and Douglas Eck. A neural representation of sketch drawings. *CoRR*, abs/1704.03477, 2017.

[2] I. Higgins, Loïc Matthey, A. Pal, C. Burgess, Xavier Glorot, M. Botvinick, S. Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *ICLR*, 2017.

[3] T. Kawashima J. Kim J. Jongejan, H. Rowley and N. Fox-Gieg. The quick, draw! - a.i. experiment, 2016.

[4] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2014.

[5] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. *CoRR*, abs/1706.02515, 2017.

[6] Chandan Singh. gan-vae-pretrained-pytorch, 2020. $\text{mnist}_v aepretrainedmodel$.

A. Appendix

A.1. Algorithms

Our pipeline for our project consists of training a variational auto encoder on a dataset of images primarily for the purpose of learning the latent vectors, training these latent vectors on a feed-forward classifier network, and finally using the latent vectors of the test data to get the test accuracy and other experimental metrics. Our VAE's encoder and decoder consist of 4 2d convolutional layers each with batch normalization and a relu activation function. Our feed forward network 5 linear layers each with a selu activation function. The metrics we used for our evaluation are test accuracy, cross entropy loss, and training time.

A.2. Datasets

We used 3 datasets in these projects: MNIST, Fashion MNIST, and Quick Draw. The MNIST dataset consists of images of handwritten digits of size 28 by 28 and contains 60,000 training images and 10,000 testing images. The Fashion MNIST consists of images of 10 classes of clothing of size 28 by 28 and contains 60,000 training images and 10,000 testing images. The Quick Draw dataset consists of sketches by human users of various classes which are of size 28 by 28. For our project we use ten classes, (tree, t-shirt, ice cream, fish, face, car, bowtie, apple, flamingo, sheep), 84,000 training images, 36000 validation images, and 30000 test images. The MNIST and Fashion MNIST datasets can be downloaded programmatically from the Pytorch Torchvision datasets, and the Quick Draw dataset can be downloaded from: https://quickdraw.withgoogle.com/data.

Prior to training our models, all data was normalized between 0 and 1 by the standard deviation and mean of its data set.

A.3. Source Code

Although we started out with a pretrained MNIST VAE model, due to its poor performance, we trained our own VAE's and Classifiers for all the datasets. The specific implementations of our algorithms, metrics, and overall pipeline are publicly accessible at https://github.com/8Gitbrix/CS7643_Final_Project/.

A.4. Parameters

The hyper parameters we experimented with for our VAE were batch size, kernel size, and number of convolutional layers. The range of batch sizes were: 32, 64, 128; The range of kernel sizes were: 3, 4, 5; The range of layers were: 2, 3, 4, 5. The best accuracy we got was with batch size of 128, kernel size of 4, and 4 convolutional layers, and this was determined by the mode that resulted in

the best validation accuracy among all these permutations. As a result, our training run consisted running our training loop over each of these permutations once and after than our evaluation run was ran once. Once we selected these parameters and architecture for the MNIST dataset we used the same parameters on the Fashion MNIST and Quick Draw datasets for consistency.

A.5. Evaluation Metrics

We evaluated our model based on test accuracy, cross entropy loss of the test data, and training time of the classifier on the pretrained VAE models. These results can be see in Table 1.