2AMM10 2021-2022 Assignment 4: Group 41

Nikhil Patni Silvia Sultana Aishvarya Viswanathan

June 20, 2022

1 Task 1 : Detect out of distribution data

1.1 Problem formulation

For Assignment 4, we are tasked with learning the underlying distribution of data in order to perform various tasks such as detecting out of distribution data points, learning relevant features of the data, and performing classification when only very few labels are present. We are given access to four datasets: a dataset containing 26000 data points which are known to be in-distribution (i.e. not anomalous) but which are without labels, a dataset containing 2000 labeled data points which are all in-distribution and also two representative datasets containing 1052 data points each which contain roughly 5% out-of-distribution data and which are fully labeled (anomalies being labeled as a sixth class). The provided data points consist of 32×32 pixel gray-scale images of datatype uint8, the labels of the first labeled dataset are one-hot encoded with five classes (datatype is float32), and the labels of the other two sets are one-hot encoded with six classes, (again of type float32). In Task 1, we are to build a deep learning model that can distribution of data from the dataset containing unlabelled datapoints and use the model to classify the datapoints in the representative datasets 1 and 2 into in-distribution and anomolous class.

1.2 Model formulation

Here, since are task is to learn the distribution of unlabelled data, we are in need of a deep learning encoder- decoder model that has provision for unsupervised distribution learning capabilities while also providing image reconstruction properties. Variational autoencoders (VAE) are a good option in this case. They provide an encoder and decoder that can learn the latent space of image encoding similiar to a vanilla autoencoder. But the VAE are better qualified over a autoencoder as it can approximate by virtue of Bayesian Inference. A normal autoencoder just decomposes and tries to re-construct - It's arguably just a transformation process of Deconvolution, Scaling, Linearity and Decompositions whereas Variational Autoencoders have Stochastics. It can learn more precisely while also grasping latent space distribution and can then detect out-of-distribution data once trained. [1]

Our model consists of an encoder and a decoder.[2] The encoder and decoder itself are made up of convolutional and deconvolutional layers which help in compression and decompression of the images. The loss is represented using elbo loss. The elbo loss is the sum of the reconstruction loss and the KL divergence loss. The reconstruction loss corresponds to how much the reconstructred image deviates from the original image. (mean vector) The KL (Kullback–Leibler) [3] divergence calculates how much the reconstruction varies from the gaussian space (standard deviation vector). The backpropagation issue is also handled by reparameterization: where we split the vector into Reconstruction loss, KL divergence and epsilon value. The epsilon handles the stochastic sampling part and doesn't need any backpropagation of gradients. The backpropagation can now travel through the two losses. ELBO losses are always negative and the negation has to be taken for quantification.

So, we can simply define our loss function as the negative ELBO:

$$-\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z})] + KL(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})). \tag{1}$$

where, p(z) is the Latent space prior distribution, p(x—z) is the Generative/conditional distribution and q(z—x) is the Approximate posterior distribution in the equation. The first term constitutes the expectation of Reconstruction loss and the second term is the KL divergence. Reconstruction loss is calculated as:

$$-\log p(\mathbf{x}|\mathbf{z}) = -\log \prod_{d=1}^{D} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_d-\mu_d)^2}{2\sigma^2}} \tag{2}$$

In particular, if we set $\sigma = \frac{1}{\sqrt{2}}$, it becomes:

$$\sum_{d=1}^{D} (x_d - \mu_d)^2 \tag{3}$$

The final equation (3) is similar to an element-wise squared error, similar to the common Mean Squared Error (MSE) loss function. There is one key difference though: the "mean" part of MSE would take the mean/average over all pixels, whereas in this case we sum over all pixels without dividing by the total count. The difference is only a scaling, but is important since we have another loss term (the KL Divergence) that needs to be appropriately weighted.

KL divergence is essentially also an expectation, so it could be approximated with Monte Carlo sampling as well. For two Gaussians (as in our chosen model) however it is possible to compute the KL divergence analytically. It is given as follows:

$$KL(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) = \frac{1}{2}\sum_{k=1}^{K}(\mu_k^2 + \sigma_k^2 - \log\sigma_k^2 - 1). \tag{4}$$

Gradients w.r.t. μ and σ can be computed exactly for this expression, so it can directly be used in a neural network loss function. Note that this loss term encourages μ and σ to stay close to $\mathbf{0}$ and $\mathbf{1}$, respectively.

1.3 Implementation and training

1.3.1 Data Handling and Pre-processing

Our given data set consists of a numpy nd array where we have four sets of nested data in the format of

```
{
'unlabeled_data': 'array',
'labeled_data': {'data': 'array', 'labels': 'array'},
'representative_set_1': {'data': 'array', 'labels': 'array'},
'representative_set_2': {'data': 'array', 'labels': 'array'}
}
```

So, each nested array was extracted and labeled the same as it's original labels. The data contained inside each was in the form of pixel values within a range of 0 to 255. This was normalized by division with 255 to convert the range from (0,255) to (0,1). This is necessary to regulate the loss functions as higher number when squared in KL loss will cause loss to explode even if the deviation is relatively small. Also loss scaled to lower range can reduce number of parameters passed to the convolutional networks, thereby reducing computational time and resulting in faster covergence. [4]

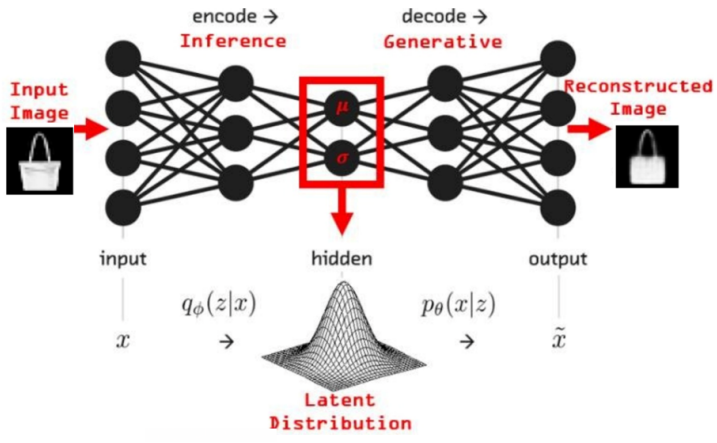

Figure 1: VAE Model architecture

1.3.2 Model

Our model 1 [5] consists of one encoder and one decoder with provision for calculation of mean and standard deviation vector and losses in addition to the latent sampled vector. We pass an image first to the encoder where the three convolution layers will compress the images progressively. The encoder first passes the image through the first convolutional layer which filters the 32*32 image with 32 kernels of size 3. The second convolutional layer takes as input, the output of the first convolutional layer and filters it with 64 kernels of size 3. The third layer has 128 kernels of size 3 connected to the output of the second convolutional layer. This output is then passed through two fully connected layers that work with feature dimension and latent dimension to predict mean (mu) and variance (logVar) which are used for generating middle representation z and KL divergence loss. Using this the ELBO loss is calculated. The decoder takes the latent sampled vector representation (z) of the original image passed through a fully connected layer to convert it to dim = 128 and use that as input to pass it through three transpose convolutional layers to build back the original image. The encoder first passes the image through the first transpose convolutional layer reforms the input image of size 26*26 and the final transpose convolutional layer gives the fully reformed image of size 32*32 again.

Before being the input z is passed through to decoder, the created μ and σ are reparameterized to include ϵ parameter to enable backpropagation. The ELBO scores are normalized and mapped on to a scale of (0,1) to check distribution. And we can recognize anomalies as: Higher the negated ELBO loss tending to 1, the more anomolous the input data.

1.3.3 Loss Function

The VAE learns distribution of data using the ELBO losses calculated. The ELBO loss is calculated as the negation of the sum of the reconstruction loss and the KL divergence loss. The loss equation can be refered to here. (1)

1.3.4 Training the Network

We initialize the network, loss function, and optimizer (Adam) with learning rate and vae parameters. To train the network, we pass batches of unlabeled data with the help of a data_loader with a batch size of 100. The images go through the encoder where after compression, the μ (mean) and σ (standard deviation) vectors are created and passed through to the loss function to calculate reconstruction loss and KL divergence loss, consequently the ELBO score. The latent image sample vector (z) created is passed to the decoder to reimagine. After which the losses are backpropagated to calculate the gradients of our model followed by updation of the weights aided by Adam optimizer.

1.4 Experiments and Discussion

1.4.1 Experimental Setup

We used the unlabeled dataset provided to us for training and to calculate elbo loss (reconstruction + kl-divergence loss). We perform experiments on this division and then use the most favorable hyperparameter combination to test our model on representative_set_1 and representative_set_2 to detect anomalies.

For our experimental setup, we chose to first compare between 2-Dimensional, 16-Dimensional and 32-Dimensional latent space. We used these three latent dimensions to see how model bahaves with varying latent dimensions. We then selected the latent dimension in which we are getting the best elbo loss and used that for further experiemnts. For the experiments, we are using the Adam optimizer as it converge faster than SGD and showing better results on our VAE architecture when looking at reconstruction loss. With Adam optimizer, we experimented with multiple learning rate (1e-2, 1e-3 and 1e-4) to see which is showing the better convergence while looking at all three losses (ELBO, Reconstruction and KL). The final combination of hyperparameters is the one used in the final model implementation as shown in Section 1.3.2.

1.4.2 Experimental Results and Analysis

From Figure 2, we can make a remark that increasing the latent_dim will decrease the reconstruction loss but if we increase the latent_dim too much (for e.g. 32), it will start diverging from original images and the reconstruction loss starts to increase. We can also see from 1 that the ELBO and reconstruction loss starts to decrease when we increase latent_dim from 2 to 16 but then the losses slowly starts to increase if we further increase the latent_dim. This is because if we increase latent_dim too much, model will become too flexible and starts negatively imapacting anomaly detection performance.[6] We can also see that from table that if we increase latent_dim, KL loss will starts to increase and that means we are going away from a Gaussian distribution. Without a well defined regularisation term, the model can learn, in order to minimise its reconstruction error, to "ignore" the fact that distributions are returned and behave almost like classic autoencoders (leading to overfitting). So, in order to avoid these effects we have to regularise both the covariance matrix and the mean of the distributions returned by the encoder. In practice, this regularisation is done by enforcing distributions to be close to a standard Gaussian distribution (centred and reduced). With this regularisation term, we prevent the model to encode data far apart in the latent space and encourage as much as possible returned distributions to "overlap", satisfying this way the expected continuity and completeness conditions. Naturally, as for any regularisation term, this comes at the price of a higher reconstruction error on the training data. We need to maintain the tradeoff between the reconstruction error and the KL divergence. That's why we backpropagate ELBO loss which is the combination of Reconstruction error and KL divergence. We showed that for latent_dim=16, this ELBO loss is minimum.

We also compared the original images with the reconstructed one for both latent_dim=2 and latent_dim=16. If we look at 4, we can observe that for latent_dim=2, it didn't reconstruct the images in very detail compared to latent_dim=16 in 5. This is because in latent_dim=2, the VAE model tries to compress the whole information into latent space of 2 dimension which is very difficult. If we look at 4th product of FashionMNIST in these images, we can see that the model trained with latent_dim=16 is able to capture the handle of the bag.

When our network is trained using Adam against different learning rates, we can conjecture that under the same setting, a learning rate of 1e-3 shows best performance with 1e-5 being the worst one. Discovering and analysing the best learning rate is essentially important for stochastic methods to prevent oscillation and slow convergence[7, 8]. From the figure 3 we observe that very low learning rates like 1e-4 and 5e-4 struggle to find the global minima and take much longer time to descent owing to it's smaller steps. Even with 100 epochs we observe that the reconstruction losses are quite high even though it looks like it's near

convergence. Hence for our further evaluations, we select the learning rate of 1e-3 which performs best with Adam optimizer.

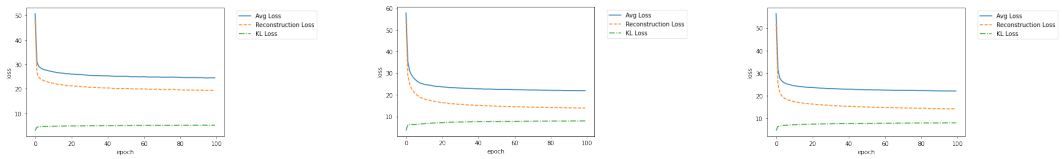

Figure 2: Training the model with different latent space dimension (2-D, 16-D and 32-D)

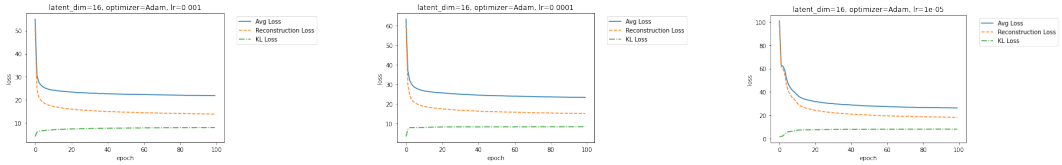

Figure 3: Training the model (optimizer=Adam, latent_dim=16) with varying learning rates (1e-3, 1e-4 and 1e-5)

1.5 Testing and Results

To detect the anomalies in representative_set_1 and 2, we have used the above trained VAE model which was trained on unlabeled dataset to detect anomalies in representative datasets. To detect the anomalies, we passed the representative data through the trained VAE model and calculated the ELBO (= approximate likelihood) scores for each datapoints individually. Lower the ELBO score, the more anomalous the image is. Higher ELBO scores represents normal images as our loss function is negation of ELBO scores. We then took the negation of ELBO scores and normalised scores so that values closer to 0 are normal and values closer to 1 are anomalies. We then plotted the histogram of Normalised Negative ELBO score distribution 6 and as we can see that most of the normal scores distribution lies in the left side and the outliers are on the right side. So we then put a threshold of 0.3 by trading off between True Negatives and True Positives. We then compared the predicted labels with the actual labels in both representative datasets and got the following results:

1. For representative_set_1, Our model is correctly able to detect 23 anomalies out of 52 and 995 normal images out of 1000, giving the roc_auc score of 0.92 and precision_recall curve = 0.59. Confusion matrix, ROC and PR curve is plotted in 7
2. For representative_set_2, Our model is correctly able to detect 18 anomalies out of 52 and 999 normal images out of 1000, giving the roc_auc score of 0.93 and precision_recall curve = 0.65. Confusion matrix, ROC and PR curve is plotted in 8

For both representative datasets, we also plotted negative elbo scores histogram 9 of normal and anomalies and we can clearly see that the model is able to clearly distinguish between normal (green) and anomalies (red).

Loss	latent_dim=2	latent_dim=16	latent_dim=32
ELBO Loss	24.50	**21.86**	22.05
Reconstruction Loss	19.39	**13.77**	14.12
KL Loss	**5.11**	8.08	7.92

Table 1: Training losses after 100 epochs for different latent space dimensions

Figure 4: Original Vs. Reconstructed images for latent_dim=2

Figure 5: Original Vs. Reconstructed images for latent_dim=16

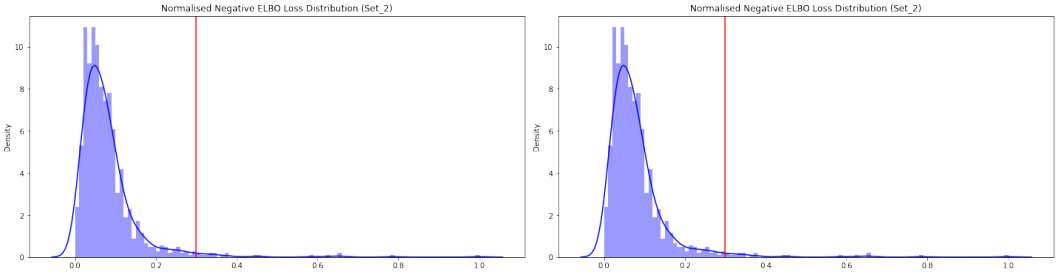

Figure 6: Normalised (-ELBO) loss distribution

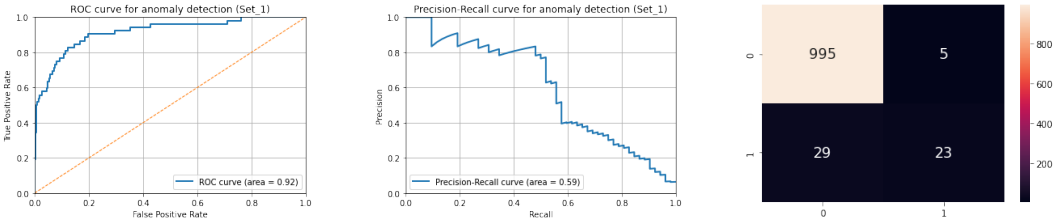

Figure 7: Evaluation matrix on Representative set 1

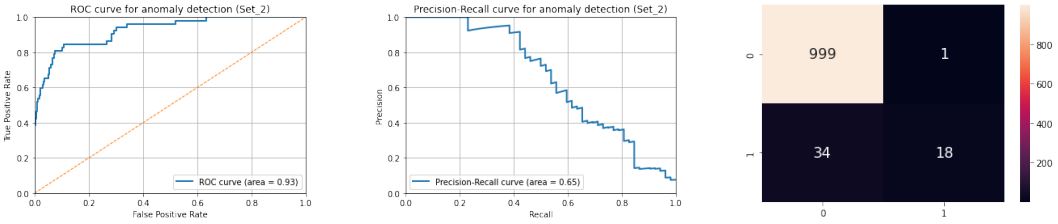

Figure 8: Evaluation matrix on Representative set 2

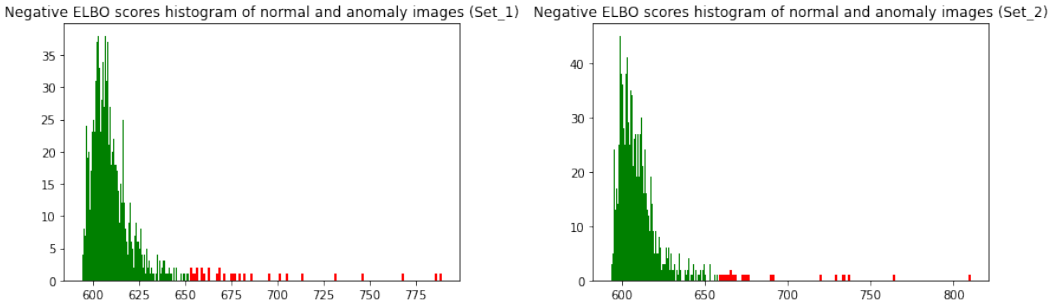

Figure 9: (-ELBO) scores histogram of normal (green) and anomaly (red) images predicted by model

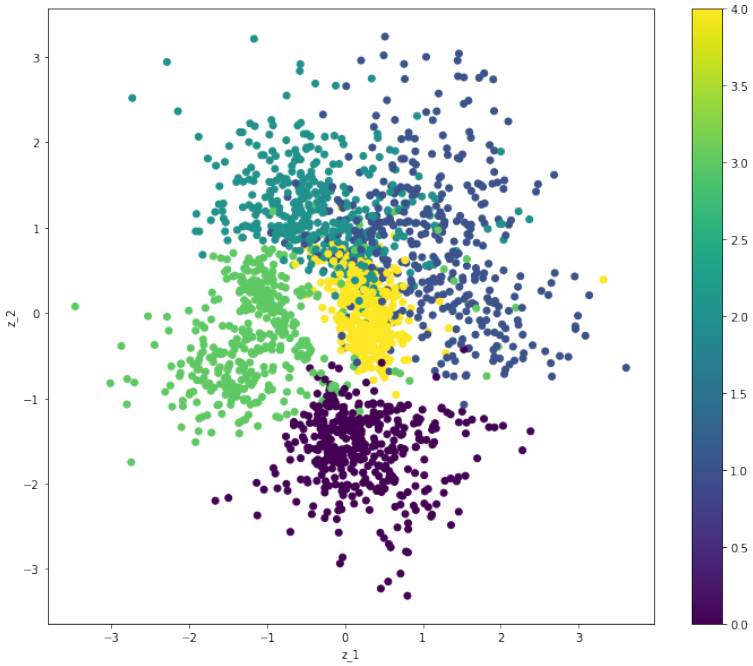

Figure 10: 2-dimensional description of the dataset in terms of five different MNIST classes (The distribution of latent values over the whole labeled data set)

2 Task 2 : Low dimensional description of the dataset in terms of the five modes of the distribution

To visualize the latent space of our MNIST dataset, we trained the same model as in Task 1 1 with latent_dim = 2 on unlabeled dataset and extracted the z (sample latent space vector) for all datapoints in labeled dataset during evaluation. We used latent_dim of 2 in this task so that we can easily visualize the latent space using scatter-plot. The plot 10 shows the sample representations in latent space for data points from the labeled set. Although VAEs are unsupervised, we do have label information for the MNIST data, so we used this to give data points a different colour depending on their label.

The diagram gives us insight into the spatial distribution of the latent values. As we can see clearly in plot that the model learned the distribution of dataset pretty well and the VAEs values are (by design) distributed around the origin. They build up smaller, spherical clusters which are contiguous and share a similar value range. We can also observe in the plot that the VAE model learned the different classes of dataset and plotted them into different isolated clusters. Also the interpolations between multiple images work quite well with VAE.

3 Task 3 : Classify Unlabeled data into five classes

3.1 Problem formulation

For Assignment 4, we are tasked with learning the underlying distribution of data in order to perform various tasks such as detecting out of distribution data points, learning relevant features of the data, and performing classification when only very few labels are present. We are given access to four datasets: a dataset containing 26000 data points which are known to be in-distribution (i.e. not anomalous) but which are without labels, a dataset containing 2000 labeled data points which are all in-distribution and also two representative datasets containing 1052 data points each which contain roughly 5% out-of-distribution data and which are fully labeled (anomalies being labeled as a sixth class). The provided data points consist of 32×32 pixel gray-scale images of datatype uint8, the labels of the first labeled dataset are one-hot encoded with five classes (datatype is float32), and the labels of the other two sets are one-hot encoded with six classes, (again of type float32). In Task 3, we are to build a deep learning model that can learn to classify unlabelled data into one of the five different classes.

3.2 Model formulation

Here, our task is to create a model that can classify unlabeled data into one of five classes. In order to achieve this, we can utilize the initially created VAE model in Task 1. Transfer learning is great way to solve this task with minimal changes to the initially created model. For this task, we require the use of only the encoder and not the decoder in the model. So we borrow the encoder and freeze these layers as they are already trained for the same data and similar case and does not require retraining. We drop the decoder modules and instead append two fully connected layers to the transfered and frozen model. The model would take in unlabelled data and the final fully connected layer will spit out a tensor of size 5 with 1 for the class the data was identified as and 0 for the other 4. The working of the model can be referred to from the section 1.3.2

This model uses Adam as optimizer with a learning rate of 1e-3 and Cross-entropy (CE) as the loss function. Cross entropy compares each of the predicted probabilities to actual class output which can be either 0 or 1. It then calculates the score that penalizes the probabilities based on the distance from the expected value. That means how close or far from the actual value. We are now measuring loss between network prediction value and actual label using below CE loss equation:

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) \tag{5}$$

where, M is the number of classes, log is the natural log, y is the binary indicator (0 or 1) if class label c is the correct classification for observation o and p is the predicted probability observation o is of class c.

3.3 Implementation and training

3.3.1 Data Handling and Preprocessing

For modifying provided data to fit our use case and model, we perform a steps of preprocessing to the data. For this task 3, we scale the 'labeled_data' dataset, 'representative_set_1' and 'representative_set_2' from the pixel scale of (0,255) to (0,1) for training and validation similar to Task 1 1.3.1. Also we remove the anomalies data from 'representative_set_1' and 'representative_set_2' as it is not useful or necessary for our use case. (i.e) we remove all data that have 1 in the 6th class label. Then the datasets of 'labeled_data' and 'representative_set_1'are concatenated to form one set of data and then split into training and validation sets in 80:20 split using train_test_split.

3.3.2 Model

Our transferred model from 1.3.2 consists of one frozen encoder with provision for calculation of mean and standard deviation vector and losses in addition to the latent sampled vector. We pass an image first to the encoder where the three convolution layers will compress the images progressively. The encoder first passes the image through the first convolutional layer which filters the 32*32 image with 32 kernels of size 3. The second convolutional layer takes as input, the output of the first convolutional layer and filters it with 64 kernels of size 3. The third layer has 128 kernels of size 3 connected to the output of the second convolutional layer. This output is then passed through two fully connected layers that work with feature dimension and latent dimension to generate middle representation z(sampled latent vector). The latent sample vector of dimension 16 is then passed to an appended fully connected layer that converts 16 to 256 dimensions and that is passed to a final fully connected layer that converts the 256 dimensions to 512 dimensions which is then converted to 5 dimensions through a softmax function to decide class label.

3.3.3 Loss Function

The VAE based network here performs classification to classify inputs as one of the five classes; hence, using cross-entropy loss function is a good choice. The loss equation used to calculate loss between network prediction and true label has been defined here. (5)

3.3.4 Training the Network

We initialize the network, loss function, and optimizer (Adam) with learning rate of 1e-3 and vae parameters. To train the network, we pass batches of train data with the help of train_loader with a batch size of 100. The images go through the encoder where after compression, the sampled latent vetor is passed through the fully connected layers and finally through a softmax function for multi-class classification. After which the losses are backpropagated to calculate the gradients of our model followed by updation of the weights aided by Adam optimizer. The architecture is demonstrated in Section 1.3.2. We are averaging out the train_running_loss over the length of the train_loader to get training_loss. Finally the train and val losses are stored for all epochs and plotted in Figure 11. We used the validation dataset to calculate validation loss of our model so that we can tune the hyperparameters and also to correct overfitting in network.

After the experimentation explained in Section 3.4, we finalized Adam as optimizer with learning_rate=1e-3. We also used Cross-entropy loss as the loss function with batch_size of 100 for both train and validation loader. With these parameters, we got the Min_validation loss of **1.0602** at Epoch 87. We chose 100 as epoch size as we can see that losses started to converge and we are also not seeing any over-fitting happening in the model. This is currently the best result we have achieved.

3.4 Experimental Analysis and Discussion

We used the labeled dataset with representative_set_1 without anomalies for training and to calculate cross-entropy loss. We then split this combined data into train set and validation set using an 80-20 train-validation split. We perform experiments on this division and then use the most favorable hyperparameter combination to evaluate our model on representative_set_2 (without anomalies). We can then use this model to classify the unlabeled dataset into five classes of MNIST. To train the model, we used Adam optimizer with lr=1e-3 as we shown in 1.4.2 that this is the best hyperparameter combination for this model. We then trained the model for 100 epochs and calculated the cross-entropy loss between predicted label and actual label and then backpropagate the loss to update the weights and biases. We can see in the train and val losses plot 11 that the losses are decreasing and converging to a point and we are also not seeing any overfitting in the model.

We used transfer learning to classify the unlabeled dataset into one of five classes of MNIST. To accomplish this task, we freeze the encoder of our previously trained VAE model 1.3.2 and

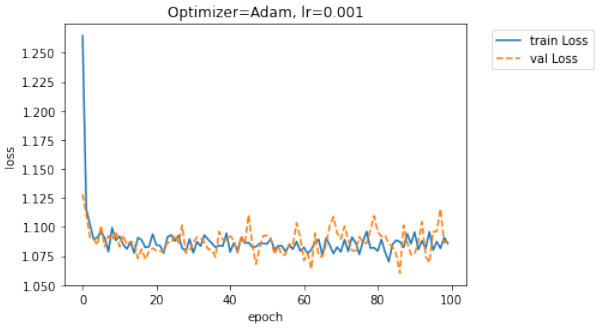

Figure 11: Epoch Vs. loss plot for VAE classification

then applied two fully connected layers with softmax activation function to get the output as probability distribution of five classes. We were able to do so because our VAE model trained on the unlabeled dataset and learned the distribution of different classes of dataset as shown in latent space visualization in 10. We can still say that the model didn't learn the differentiation between classes very well as the validation loss is high in this case. This might be also because model learned the distribution of data very well as it is trained with 26000 unlabeled datapoints and it is able to classify well between either an image comes from this distribution or an anomaly image but it didn't train with sufficient number of labeled datapoints (onlt 3000). That's why it can't perform well in classification tasks. This transfer learning technique also don't perform well on labeling out of distribution datapoints like anomalies because the model didn't learn the distribution of those datapoints.

3.5 Testing and Results

We tested our transfer learned VAE classification model to predict the labels of representative_set_2 (without anomalies) and then compared it with the actual labels to calculate average test loss and test accuracy. After testing it on 1000 datapoints, we got the average test loss of **1.0983** and achieved the test accuracy of **0.81**. We can then use this model to classify unlabeled datapoints into five classes of MNIST.

4 Conclusion

We discussed and implemented a Variational Autoencoder (VAE) to detect anomalies in dataset after learning the distribution of data using unlabeled_dataset. We were able to detect most of the anomalies in both the represntative datasets by putting a threshold of 0.3 in Normalised Negative ELBO score distribution. The results are mentioned in 1. VAE models don't giving perfect results because the latent dimension is so small and they have to compress the entire information into the latent dimension vector. That's why the generated images are not exactly same as original images but still performed well in compressing the information. Another limitation which we observed in the training phase is that VAE models take a lot of time and resources to train. That is the reason we trained it for 100 epochs. They will perform a bit better if trained for more epochs.

We then showed the latent space visualization of the dataset in 2 dimensional. We saw there that the model is able to learn the distribution of different classes and able to segregate them in clusters. We represented the dataset by taking the latent dimension of 2. We can also represent it by taking higher latent dimension and then do dimensionality reduction using PCA and plot t-SNE. Higher latent dimension might be able to segregate between classes a little bit better as it is in less compressed form and have more information.

After that for task 3, we transfer the previously trained VAE encoder and used its weights and biases to classify the unlabeled dataset. One limitation we saw while doing transfer learning is that the transfered model is trained very well on unlabeled dataset and is able

to learn the distribution but didn't perform well for classifying labels. We figured that it is because it is not trained for many labeled datapoints and will perform better if trained on more datapoints. Another limitation of our network is that even though we performed extensive experimentation, we missed many combinations between the hyperparameters (like dynamic changing between learning rate and momentum values for SGD) and hence there remains enough scope for hyperparameter tuning in our case. For example, we could have used better techniques like grid search or AutoML. A future direction of work can be based on Generative adversarial networks as showed in [9] to detect anomalies and to generate new data with the same statistics as the training set.

References

[1] Thilo Spinner, Jonas Körner, Jochen Görtler, and Oliver Deussen. Towards an interpretable latent space: an intuitive comparison of autoencoders with variational autoencoders. In *IEEE VIS 2018*, 2018.

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

[3] James M Joyce. Kullback-leibler divergence. In *International encyclopedia of statistical science*, pages 720–722. Springer, 2011.

[4] Nikhil B. Image data pre-processing for neural networks, Jan 2019.

[5] Kevin Frans. Variational autoencoders explained, Sep 2019.

[6] Christopher Vogelsanger and Christian Federau. Latent space analysis of vae and intro-vae applied to 3-dimensional mr brain volumes of multiple sclerosis, leukoencephalopathy, and healthy patients. *arXiv preprint arXiv:2101.06772*, 2021.

[7] James Martens. New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193*, 2014.

[8] Yuhuai Wu, Mengye Ren, Renjie Liao, and Roger Grosse. Understanding short-horizon bias in stochastic meta-optimization. *arXiv preprint arXiv:1803.02021*, 2018.

[9] Houssam Zenati, Chuan Sheng Foo, Bruno Lecouat, Gaurav Manek, and Vijay Ramaseshan Chandrasekhar. Efficient gan-based anomaly detection. *arXiv preprint arXiv:1802.06222*, 2018.