Exploring CLIP Feature Vectors for Improved Out-of-Distribution Detection

Jessica Li

Follow this and additional works at: https://digitalcommons.uri.edu/theses
Exploring CLIP Feature Vectors for Improved Out-of-Distribution Detection

BY JESSICA LI

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
COMPUTER SCIENCE

UNIVERSITY OF RHODE ISLAND
2023
MASTER OF SCIENCE THESIS

OF JESSICA LI

APPROVED:

Thesis Committee:

Marco Alvarez

Krishna Venkatasubramanian

Yan Sun

Brenton Deboef

DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

2023
ABSTRACT

Out-of-distribution detection in the realm of computer vision is dominated by deep generative models usually trained on raw pixel images. The results of out-of-distribution detection with generative models vary. Among the generative models used today for out-of-distribution detection, diffusion models seem to perform well however, diffusion models tend to be computationally expensive and slow. In an effort to improve out-of-distribution detection using generative models, we introduce a diffusion based autoencoder model. Another area of interest is the use of OpenAI’s Contrastive Language-Image Pre-training (CLIP) image encoder to create CLIP feature vectors of our datasets for use with our model. We compare the performance results of both the use of CLIP feature vectors and the use of raw pixel images with our model. We also test the performance of our model against benchmark results of other models such as a diffusion model.
ACKNOWLEDGMENTS

I would like to start by first thanking my advisor, Dr. Marco Alvarez, for his support and guidance in the completion of my research and thesis. Next, I would like to thank Dr. Krishna Venkatasubramanian and Dr. Yan Sun for being members of my committee. Third, I would like to thank Nancy Eaton for chairing my defense. Lastly, I would like to thank my friends and family for their continued support.
1 Introduction

Computer vision (CV) is a popular area of research today with image processing being one of the leading applications of CV. There are many aspects of image processing such as image recognition, image generation, image classification, and image detection [15]. To tackle these tasks, many machine learning methods have been developed. Among these methods used for CV are deep generative models like autoencoders, generative adversarial networks (GAN), and diffusion models. These models are popular and well documented in the realm of image processing with image generation being one of the main areas of focus. Image generation is the leading application of generative models like diffusion models today. Generative models learn to reconstruct the input images allowing them to generate sample images, in the case of diffusion models [6] and GANs [15], and learn the data distribution. Text to image generation and image manipulation tools based on diffusion models like DALL-E 2 [3] have gained popularity in recent years due to their impressive performance. Diffusion models work well for the task of image generation, but what about anomaly detection? How well do other generative models hold up in comparison for anomaly detection? Does the image representation used for training affect performance? Generative models today largely use raw pixel images for model training and testing model performance. Minor data augmentation and preprocessing may occur prior to use of the data with the model such as image resizing or the conversion of color images to grayscale images. With these changes, the input data is still reminiscent of raw pixel data. What would occur if instead of the use of raw pixel data, the images were encoded into a feature representation of the images for use with a generative model? Would the model perform better for the task of anomaly detection? Contrastive Language-Image Pre-training (CLIP) by OpenAI provides an image encoder that we will use for this purpose. The goal of this study is to use both raw pixel images and CLIP features with a diffusion based autoencoder model inspired by
the diffusion process of diffusion models in the context of anomaly detection. More specifically, out-of-distribution detection.

There is little guarantee in the performance of predictive models when faced with input data that differ from the distribution the model trained on [28]. The authors of [16] discuss the performance issues encountered when trained normalizing flows are introduced to out-of-distribution data. The model assigned higher likelihoods to the out-of-distribution data, MNIST, than in-distribution data, FashionMNIST. Neural Networks classifying out-of-distribution image data with a high confidence score as in-distribution data may lead to serious issues. These issues may even become fatal hazards. For example, given an autonomous vehicle where various traffic signs and cars are considered in-distribution data and birds as out-of-distribution data, if a bird is classified as a stop sign the vehicle may cause a traffic accident. Another application of out-of-distribution detection is in the medical field. The authors of [14] discuss the application of artificial intelligence (AI) and machine learning (ML) for COVID-19 diagnosis and treatment. Accurate predictions of a patient’s diagnosis and severity is crucial in the treatment and recovery of a patient. Supplementing AI/ML models with out-of-distribution detection in the medical field may help avoid inaccurate predictions [4]. Identifying whether the input data is out-of-distribution or anomalous is integral to the success of AI/ML models’ ability to make safe and reliable decisions.

Types of approaches to out-of-distribution detection are categorized as generative model based, uncertainty based, and softmax-based. Softmax-based methods are popular due to their low computational cost and simplicity. Their anomaly detection capability comes from a comparison between a threshold value and the maximum value of the softmax probability. This probability is the confidence score. Softmax-based methods depend heavily on neural networks to separate the confidence scores of out-of-distribution data and in-distribution data. Uncertainty-based methods pro-
duce an uncertainty score for out-of-distribution data. The neural network is modified to include a confidence branch to compute uncertainty scores given an input. For generative models, out-of-distribution data is assumed to have poor image reconstruction in comparison to in-distribution data. The image reconstruction loss is used to determine if the input is out-of-distribution. Variational autoencoders (VAE), generative adversarial networks (GAN), and diffusion models fall under the generative model category and are all considered unsupervised methods [18]. Unsupervised methods, unlike supervised methods, do not require the use of labels and out-of-distribution data during model training. Each model has their own set of strengths and weaknesses. VAEs provide ease of training however, they lack high quality image generation. GANs provide high quality image generation however, are more difficult to work with. GANs are susceptible to mode collapse making them more difficult to train [6]. GANs are also susceptible to gradient disappearance during model training and may have poor diversity with GAN generators [15]. Diffusion models on the other hand produce high quality images, provide easy scalability, and provide better distribution coverage than GANs [6]. However, they can be computationally expensive and are often slow and time consuming to train and use for image generation. Thus, in an attempt to improve the image reconstruction quality of an autoencoder, we combined the structure of an autoencoder with the diffusion process of diffusion models to create a diffusion based autoencoder.

The remaining of this thesis is organized as follows. Chapter two provides an overview of key theories, concepts, and methodologies relevant to my research. Chapter three presents the research methodology used in this research, including benchmarks used, model design and data preprocessing. The results of the research are also presented in Chapter three. Lastly, Chapter four offers a conclusion that summarizes the key findings of the study and discusses potential future paths forward.
2 Background

2.1 Out-of-Distribution Detection

Out-of-distribution detection is sometimes interchangeably referred to as anomaly detection. Although the two methods are related and share similar motivations, there are subtle differences between them. Anomaly detection refers to the testing of data to identify outliers or unexpected behaviors in the data. The goal is to detect any samples that deviate from a predefined normality. What is considered normal or expected behavior must be defined clearly. Out-of-distribution detection refers to detecting sample data from a distribution different from the distribution the model was trained on. The goal of out-of-distribution detection is to identify inputs semantically different from the distribution the model trained on to aid in the prevention of incorrect predictions of AI/ML models [26]. The model is trained on one data distribution (in-distribution data) and tested on a data distribution different from the training distribution (out-of-distribution data). Another way to describe out-of-distribution detection is the task of distinguishing points with low likelihood values, low probability or density, from points with high likelihood under a training distribution that is estimated by a model. Ideally, only in-distribution data will have a high likelihood value while all out-of-distribution data will have low likelihood values. This is unfortunately not always the case. In some instances, a trained model may assign higher likelihood values for out-of-distribution data than in-distribution data [28]. The authors of [28] suggest that this phenomena stems from model estimation errors. For example, given infinite samples of the MNIST dataset, you would not find an image of a horse among them. If the model assigns a high likelihood for the image of a horse, then the model has mistakenly assigned a high probability for the image when the model should have assigned a zero. The authors of [28] suggest the use of adding alternative test statistics to minimize model bias and prevent the model from
assigning high likelihoods for data that should have a likelihood of zero.

2.1.1 Supervised Out-of-Distribution Detection

Supervised methods require the use of out-of-distribution data and labeled data during the training process [11]. This poses two main issues. There may be an imbalance of data instances between the in-distribution data and the out-of-distribution data for training and providing accurate and representative labels may be challenging [20]. Furthermore, collecting large amounts of labeled out-of-distribution data is often expensive and difficult to achieve [27]. As a result, supervised methods are rare. The authors of [17] and [10] both detail supervised out-of-distribution detection methods. In both cases, a \((N + 1)\) classification problem is created. The model has a total of \((N + 1)\) classes where the additional class label is for data unknown to the model [27].

2.1.2 Unsupervised Out-of-Distribution Detection

Unsupervised methods only require the use of in-distribution data for the training process. Unsupervised approaches to out-of-distribution detection either use metrics such as the likelihood estimation from a trained generative model or the reconstruction error from a model trained to reconstruct in-distribution data to detect out-of-distribution data. The approach of evaluating a model’s likelihood assumes that out-of-distribution data will be given a lower likelihood in comparison to in-distribution data. The likelihood value is compared to a threshold to identify out-of-distribution data [11]. The authors of [5] show that a trained model will not always assign a lower likelihood value to out-of-distribution data than in-distribution data. The model trained on CIFAR-10 data produced a higher likelihood value for SVHN data than for CIFAR-10 [5]. Approaches using image reconstruction error require training a model to reconstruct an input. The assumption is that the information bottleneck, latent space, of the trained model will only reconstruct inputs from in-distribution
data well. Out-of-distribution data will have poor reconstruction [11]. Examples of unsupervised models are VAEs, GANs and diffusion models. This study will focus on unsupervised out-of-distribution detection with a diffusion based autoencoder.

2.2 Deep Learning Approaches For Out-of-Distribution Detection

2.2.1 Autoencoders

Autoencoders are neural networks (NN) that are trained to reconstruct the input data given to the model. Their task is to learn a meaningful feature representation of the input data distribution in an unsupervised manner. Autoencoders consists of an encoder and a decoder. The purpose of the encoder is to map the input data into a feature representation of the input, a latent space. The purpose of the decoder is to reconstruct the input given the latent feature representation created by the encoder. Autoencoders are trained to minimize the loss function, a distance comparison between the original input data and the reconstructed output data. There are many variations of autoencoders. One type of autoencoder is a regularized autoencoder. Regularized autoencoders are what most people envision when they think of autoencoders. Regularized autoencoders create an information bottleneck by decreasing the layer size as the data progresses through the encoder. This creates a low dimensional representation of the data. Another type of autoencoder is a denoising autoencoder. Denoising autoencoders share the same architecture of a regularized autoencoder. The main difference is with the input data given to the model. Denoising autoencoders are given an input with added noise and are expected to reconstruct a cleaner version of the input data. There is also a special case where the encoder and decoder are both linear operations. This leads to a linear autoencoder where the latent representation is similar to the latent representation achieved with principle component
A fourth type of autoencoder is the variational autoencoder (VAE) [2]. VAEs function similarly to regular autoencoders. The method consists of an encoder that compresses data to a smaller dimension, a latent space, and a decoder that reverses the compression process. The latent space contains the feature representation of the data. Where VAEs differ from regular autoencoders is the addition of two vectors. One for the mean of the data distribution and another for the standard deviation. This allows for the model to map an input to a distribution opposed to a fixed feature vector. The loss function is a combination of the reconstruction loss and KL divergence. Maximizing this loss is how VAEs train [25].

2.2.2 GANs

GANs consist of a generator tasked with the generation of fake samples and a discriminator tasked with determining if the given input is real or fake. The generator takes a random input noise vector and makes feature predictions, learning the data distribution. The discriminator is trained to recognize the features of the data and predict if the given input is real data or fake data generated by the generator. The nature of the data, real or fake, is revealed to both networks once the discriminator makes its prediction. Based on the accuracy of the prediction, each network will respond accordingly. If the discriminator made a correct prediction, the discriminator network remains unchanged while the generator is updated. If the prediction made by the discriminator is incorrect, the discriminator will update its weights via backpropagation while the generator will remain unchanged. This back and forth interaction, game training, allows for improved generalization of the GAN model through the optimization of model weights between the generator and discriminator. Ideally, the model is able to generate sample data well enough for the discriminator to no longer be able to differentiate the true data from the fake data generated by
the generator [15]. For out-of-distribution detection, the model is trained on only in-distribution data and tested on both in-distribution data and out-of-distribution data. An anomaly score is used based on the reconstruction loss. The assumption is that the anomaly score will be higher for out-of-distribution data than in-distribution data [1].

2.2.3 Diffusion Models

Diffusion models are generative models with high quality image synthesis capabilities. Training the model requires two stages, forward diffusion and reverse diffusion. The diffusion process involves adding noise and removing noise to learn the noise distribution of the data. A trained model is able to generate new data that resembles the data distribution the model was trained on. For out-of-distribution detection, diffusion models are trained on in-distribution data and tested on both in-distribution data and out-of-distribution data. This method shares the same assumptions as GANs. The out-of-distribution data will have a higher reconstruction loss than the in-distribution data.

Forward Process

The goal of the forward process is to slowly add noise to the input image until the image becomes a standard Gaussian distribution $N(0, I)$. Figure 1 below depicts this process. The noise added is based on a variance schedule $\beta_1, ..., \beta_T$ where $T$ is the total amount of timesteps. The larger the $T$ value, the more noise the image has. The forward process is denoted by the first formula (1) below and the noise formula is denoted by the second formula (2) where $X$ is the image and the subscript is the timestep. $X_0$ is the original image. The noise is a Gaussian distribution where the mean is dependent on the previous image and has a fixed variance. Noise is added
little by little at each timestep.

$$q(X_{1:T}|X_0) = \prod_{t=1}^{T} q(X_t|X_{t-1}) \tag{1}$$

$$q(X_t|X_{t-1}) = N(X_t; \sqrt{1 - \beta_t}X_{t-1}, \beta_tI) \tag{2}$$

The noise formula can be rewritten based on the closed form of the mean and variance from the cumulative variance schedule. Using the notation $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$, the rewritten formula (3) allows sampling of a noisy image at any timestep given the original image $X_0$ [12].

$$q(X_t|X_0) = N(X_t; \sqrt{\bar{\alpha}_t}X_0, (1 - \bar{\alpha}_t)I) \tag{3}$$

![Figure 1: Example of forward diffusion of MNIST zero data](image)

**Backward Process**

The goal of the backwards process is to learn the noise distribution. Learning the distribution allows the U-NET model to predict the noise that was added to an image. The backwards process (4) is a Markov chain starting at $p(X_T) = N(X_T; 0, I)$ with learned Gaussian transitions. Since the variance is fixed to the variance schedule, the model only needs to predict the mean of the Gaussian noise distribution (5).

$$p_\theta(X_{0:T}) = p(X_T) \prod_{t=1}^{T} p_\theta(X_{t-1}|X_t) \tag{4}$$

$$p_\theta(X_{t-1}|X_t) = N(X_{t-1}; \mu_\theta(X_t, t), \Sigma_\theta(X_t, t)) \tag{5}$$
Diffusion models are optimized by the variational lower bound with loss function defined as (6). The distance is calculated between the actual noise and the predicted noise to determine the reconstruction loss [12].

\[ L_{\text{simple}} = E_{t,X_t,\epsilon} \left[ ||\epsilon - \epsilon_\theta(X_t, t)||^2 \right] \] (6)

Figure 2: Model architecture of VAEs, GANs, and Diffusion models.

2.3 Contrastive Language-Image Pre-training (CLIP)

The CLIP model, developed by OpenAI, is a cutting-edge neural network architecture trained on an extensive dataset consisting of diverse text and image pairs. Through the use of multimodal learning, learning through the use of multiple representations of the data (text and image), CLIP excels in discerning the intricate
relationship between images and the corresponding textual description [21]. CLIP also provides zero shot capabilities. Zero-shot learning is the capability of the model to predict a class that it has not seen during training [22]. Inspiration for the model came from the massive success of text-to-text zero-shot capabilities of pre-trained methods that learn from raw text for natural language processing (NLP). These pre-trained methods removed the need for data specific training. Given the success of pre-trained methods for raw text data, OpenAI applied similar techniques to the realm of computer vision [21].

Prior work done by other researchers demonstrated the possibilities and potential of image and text pairings in ML. However, early results of image representation through the use of natural language supervision was not on par with other ML methods of achieving zero-shot capabilities for images. Natural language supervision is the use of natural language during the training process of a model. This allows for easy scalability and for the model to not only learn the representation of the data, but also link that data representation to language. This connection between the data representation and language learned by the model allows for flexible zero-shot learning. OpenAI chose to improve on past research done by others through scaling the amount of data used for image classification when trained with natural language supervision. This resulted in the CLIP model. CLIP is a simplified version of the ConVIRT model trained from scratch on 400 million image and text pairs [21]. ConVIRT is an unsupervised method that learns visual representations of image and text pairs [29].

The CLIP model is trained to predict the image-text pairing that most likely occurred given a batch of image-text pairs. CLIP learns the multi-modal embedding space of the image-text pairings by training an image encoder and a text encoder in parallel. CLIP uses a linear mapping of the encoder to the multi-modal embedding space. CLIP takes the concepts learned from one modality, grammar and sentence
structure of text, into consideration when using the other modality, lines and shapes of images, to embed the input data into a feature space. CLIP learns to embed each text and image pair to as similar feature representations as possible. CLIP aims to maximize the cosine similarity of the embeddings of the real image-text pairs and minimize the similarity of the wrong pairings. A cross entropy loss function is used for optimization of the similarity scores [21].

CLIP’s image encoder consists of two architectures. One of the architectures used for CLIP image encoding is a modified ResNet. ResNet-50 was used as a base with the addition of a rect-2 blur pooling layer to prevent aliasing among other changes. The second architecture used is the Vision Transformer (ViT) introduced in [7]. The image encoder creates a semantically derived feature representation of the images. The CLIP feature vector derived from the image encoder provides a mix of low level and high level image features. CLIP’s text encoder consists of a slightly modified version of the Transformer described in [24]. The Transformer uses a lowercase byte pair encoding [21].

The decision to use CLIP’s capability to encode images into feature vectors for this research came from the desire to leverage CLIP’s ability to generate semantically meaningful feature representations of images and to provide a performance comparison between the use of CLIP features and raw pixel images. CLIP features contain high level feature representations of images such as a face in an image. High level features of an image are semantically meaningful concepts derived from low level features. Low level features are simple key features of an image such as colors, angles, and edges. Using raw pixels does not provide a meaningful representation of an image in the same way CLIP features do. The use of raw pixels do not provide the model an increased depth of knowledge of the input image while CLIP features do.
2.4 Performance Metrics

2.4.1 Receiver Operating Characteristics (ROC) and Area Under a ROC Curve (AUC)

ROC graphs are a way to visualize and organize performance of classifiers. They are two-dimensional graphs where the Y axis is the true positive (TP) rate and the X axis is the false positive (FP) rate. The equations below depict the true positive rate and false positive rate calculations [9].

\[
TP = \frac{\text{Correctly Classified Positives}}{\text{Total Positives}} \tag{7}
\]

\[
FP = \frac{\text{Incorrectly Classified Negatives}}{\text{Total Negatives}} \tag{8}
\]

AUC is a single scalar value derived from the area under the ROC curve that represents the expected performance of a model. The AUC score will always fall between zero and one. The higher the AUC value, the better the model is at correctly classifying data [9].

2.4.2 Mean Squared Error (MSE) and Negative Log Likelihood (NLL)

MSE is calculated by the summation of all error values divided by the total number of data points. Each error value is calculated by taking the difference of the observed value and the predicted value and squaring the result [13]. MSE is the average of a set of errors.

\[
MSE = \frac{1}{\text{Total Data}} \sum_{i=1}^{\text{Total Data}} (\text{Observed Values} - \text{Predicted Values})^2 \tag{9}
\]

NLL is the negative log of the likelihood value. Likelihood is the probability of an outcome, B, given a true instance, A [8]. NLL is equal to \(-log(P(B|A))\).
3 Methodology

3.1 Problem Statement

More often than not, classification ML methods train and test the model with the same data distribution. The data used for both training and testing contain the same classes. This is impractical when deployed for real world use. It is unrealistic to be able to account for all possible encounters that the trained model will observe when deployed. Therefore, the model is likely to encounter data that is unknown to the trained model. These unknowns, out-of-distribution (OOD) data, can lead to inaccurate predictions. Inaccurate predictions can be detrimental for systems that are safety critical such as, autonomous vehicles and medical applications. The goal of OOD detection is to identify inputs to the model that are not part of the data distribution the model trained on. By identifying OOD data, the model can reject inputs that are inappropriate for the model [23].

The capabilities of a model for OOD detection can be tested through the data reconstruction loss. This loss is determined by how similar the reconstructed data is to the original input data. In-distribution data is assumed to have a lower reconstruction loss than OOD data. Alternatively, log-likelihood values can also be used to determine OOD data from in-distribution data. Using the reconstruction loss, we can calculate the likelihood value. It is assumed that OOD data will have a lower likelihood value than the in-distribution data [16]. In both cases, the loss or likelihood values can be used in a binary classification problem. If the value falls within a threshold, then the data is considered in-distribution or OOD depending on the definition of the threshold.

The focus of this paper is OOD detection through the use of a diffusion based autoencoder model. The performance of our model will be tested against other benchmark model results. Model performance and our model’s capability of OOD
detection will be focused on model assigned likelihood estimations of the input data. The likelihood values are determined by the reconstruction loss of the original input and the reconstructed image outputted by the model.

3.2 Datasets

A brief description of each dataset used and a preliminary analysis of the similarities of the datasets can be found below.

The MNIST dataset from the National Institute of Standard and Technology consists of 70,000 28 by 28 grayscale images of handwritten numbers from zero to nine. This dataset is split into 60,000 training images and 10,000 testing images.

Figure 3: Sample MNIST data

The Fashion-MNIST dataset created by Zalando consists of ten classes of 28 by 28 grayscale images of various articles of clothing. These classes are T-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, and ankle boot. There are a total of 70,000 images split into 60,000 training images and 10,000 testing images.

Figure 4: Sample FashionMNIST data
The CIFAR-10 dataset from the Canadian Institute for Advanced Research consists of 32 by 32 color images of ten different object classes. These classes are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. There are a total of 60,000 images split into 50,000 training images and 10,000 testing images.

![Sample CIFAR10 data](image)

Figure 5: Sample CIFAR10 data

The SVHN dataset is a set of 32 by 32 color images of the numbers zero through nine. These images differ from the MNIST dataset as they come from real world images of house number plates. The dataset is split into 73,257 training images and 26,032 testing images.

![Sample SVHN data](image)

Figure 6: Sample SVHN data

The CelebA dataset is a set of 218 by 178 color images of celebrity faces. The dataset is split into 162,770 training images, and 19,962 testing images.
For each of these datasets, two additional sets of data were created through the application of a vertical flip on the images and a horizontal flip of the images.

These datasets all have similarities and differences to each other. MNIST and FashionMNIST are similar in their image color and size. They are both black and white 28 by 28 pixel images while the other three datasets contain color images. CIFAR10 and SVHN are both 32 by 32 pixels while CelebA is the most different in image size. All the images are square with the exception of CelebA. CelebA images are much larger in comparison to the other datasets. In terms of the subject of the images, SVHN and MNIST are similar. Both datasets contain numbers. FashionMNIST and CIFAR10 are also similar in structure. They both contain ten different classes of various distinct objects.

### 3.3 Experiment Details

Each of the five datasets were used to train two models. One model was trained on the raw pixel images while the second model was trained using CLIP feature vectors. This resulted in a total of ten trained models. Each model trained for 100 epochs with a batch size of 2048. For each of these models, the dataset it was trained on was considered the in-distribution data while the remaining four datasets were considered OOD data. The vertical flipped images and horizontal flipped images from each re-
spective dataset were also used as OOD data. Each of the trained models were tested against their respective OOD data and the results of some of these comparisons were compared with the results found in [11]. The authors of [11] benchmarked the results of their research against several generative based approaches and reconstruction based approaches to OOD detection done by other researchers. The authors of [11] define generative based approaches as approaches that focus on the use of likelihoods to evaluate unknown samples after the generative model has been fit to a data distribution. In contrast, reconstruction based approaches use the reconstruction loss based on the similarity of the original input and the reconstructed input developed by the trained model. A description of each benchmark test can be found in [11]. The area under the ROC curve (AUC) score is used to determine the performance of each method. ROC is a probability curve while AUC measures the accuracy of the model by determining the degree of separability. The larger the AUC score, the better the model is performing.

3.4 Data Preprocessing

CLIP feature vectors of size 1024 were produced for each of the five datasets. The feature vector size may change depending on the trained CLIP model used for feature extraction. For this study, we used open_clip’s pretrained ViT-g-14 model. This model was trained on the LAION-2B dataset. The model had an accuracy of 78.5%. The LAION-2B dataset is a large image-text dataset. It contains two billion image-text pairs filtered by CLIP.

Data preprocessing for the raw pixel images required more work than creating the CLIP feature vectors for each dataset. For the grayscale image datasets, MNIST and FashionMNIST, the only data processing required was flattening the images to a vector of size 784. For the rest of the datasets, they were converted to grayscale and resized to 28 by 28 pixel images to match the MNIST and FashionMNIST dataset.
sizes. Then, the datasets were also flattened to a vector of size 784 for use with our diffusion autoencoder model. The flattening, resizing, and conversion to grayscale is necessary to structure the input data into a format usable by our model.

### 3.5 Model Parameters

Two hyperparameter searches were completed for each dataset. One search was done using the CLIP features extracted from the dataset and another search was completed on the raw pixel images of the dataset. The parameters used in the hyperparameter search were the learning rate, the number of layers for the encoder, the number of timesteps used for the diffusion process, and the hidden layer size. Table 1 shows the values used in the hyperparameter search for CLIP features. These values were also used for the raw pixel images aside from the hidden layer size. The hidden layer size values used for the raw pixel images were [392, 588, 784]. The values for the hidden layer size for both CLIP features and raw pixels were derived from the input size. I used the input size, 3/4 of the input size, and 1/2 of the input size as the values for the hidden layer.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_rate</td>
<td>0.0001, 0.001, 0.01</td>
</tr>
<tr>
<td>n_layers</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>n_timesteps</td>
<td>5, 10, 20, 40</td>
</tr>
<tr>
<td>hid_sz</td>
<td>512, 768, 1024</td>
</tr>
</tbody>
</table>

Table 1: Hyperparameters

Based on the hyperparameter search for the CLIP feature based models, all datasets trained best when there was only one layer in the encoder portion of the model structure with the hidden layer size equaling 1024. This is the same size as our input and output layer. For the CIFAR10, CelebA, and SVHN datasets, a learning rate of 0.0001 worked best. In contrast, for the MNIST and FashionMNIST datasets, a larger learning rate of 0.001 worked best. These two datasets also shared the same
number of timesteps, 10, for optimal results. SVHN also used a timestep of 10. CelebA had the smallest timestep value, 5, while CIFAR10 used a timestep of 40.

The hyperparameter search results for the raw pixel images show that much like the CLIP features, all datasets trained best when there was only one layer in the encoder and the hidden layer size matched the input and output layer size, 784 in this case. All datasets performed best with a learning rate of 0.001 and all datasets except FashionMNIST used a timestep value of 10. FashionMNIST performed best with a timestep value of 5.

3.6 Model Structure

The model structure of our diffusion autoencoder is similar to that of a standard autoencoder. The main difference between the two models is the addition of a diffusion process similar to a diffusion model in our autoencoder structure. The encoder portion of our model remains the same as most standard autoencoder models. The encoder structure starts with a linear neural network input layer that receives either a flatten image array, or a CLIP feature vector. The input layer is followed by the rectified linear unit (ReLU) activation function. This activation function is a piecewise linear function. Given a positive input, the function returns that exact value as the output. If the function is given a negative input, the function will output zero. After the ReLU function is applied to the input layer, a dropout layer is introduced. The task of the dropout layer is to help prevent overfitting by preventing all neurons in a layer from converging to the same point. The dropout layer randomly sets inputs sent to the next layer to zero at a rate specifiable in the model parameters. This structure repeats for the hidden layers of the encoder. The total number of layers in the encoder is determined by the user. Our model takes as a parameter, the number of layers we want the encoder to have.

In the case of our experiments discussed in this paper, based on the hyperparameter
search described above, our encoder consisted of only one set of the three piece structure described above for the input layer. We did not have any hidden layers. The next portion of our model structure are the diffusion layers. The diffuser, much like the encoder, is a sequential structure and uses the same three components. The diffuser structure is as follows, \( \text{linear neural network layer} \Rightarrow \text{ReLU} \Rightarrow \text{dropout} \Rightarrow \text{linear neural network layer} \Rightarrow \text{ReLU} \Rightarrow \text{dropout} \Rightarrow \text{linear neural network layer} \Rightarrow \text{ReLU} \). Our model takes as a parameter the number of timesteps for the diffusion process. Based on the timestep value, the following process is repeated. Noise is sampled from a standard Gaussian distribution much like with a diffusion model. This sampled noise is then added to the resulting output of the diffuser given the previous layer’s output. This process of adding noise to distort the input aids in the prevention of our model from learning the identify function especially, since our model layer size remains consistent throughout. Once the diffusion is completed, the noisy output is sent to the decoder where the decoder learns to remove the added noise and reconstruct the original input data. The decoder is also a sequential structure made up of the same three components as the rest of the model. The decoder structure is as follows, \( \text{linear neural network layer} \Rightarrow \text{ReLU} \Rightarrow \text{dropout} \Rightarrow \text{linear neural network layer} \). The model is trained to minimize the mean square error (MSE) loss function. Similarly to diffusion models, the image reconstruction loss, MSE in this case, is used to determine the performance of the model and the OOD score.

![Figure 8: Model Architecture](image_url)
3.7 Results and Analysis

A summary of the AUC scores for each baseline and comparison that was portrayed in [11] with the addition of our model is shown in Table 2 and Table 3. The authors of [11] implemented seven models for a performance comparison with their model. They followed the original implementation of the creators of each model whenever possible and used the same sets of data as well as the same performance metric for each comparison. The results of my models use the same performance metric and data for my comparison with the other eight models. All five datasets are easily accessible to the public making this test fairly repeatable given ample time and computational resources.

Our model that trained on CLIP features performed better than all other models in 12 out of the 15 comparisons and tied for best performance with the DDPM model on an additional comparison. Our model that trained on CLIP features out performed all of the other models with an average performance ranking of 1.6. Our model that trained on raw pixel images only performed better than all other models on one comparison and had an average performance ranking of 6.2. The overall placement of our model out of the ten models was first place for the CLIP feature trained model and seventh place for the raw pixel trained model.

<table>
<thead>
<tr>
<th></th>
<th>FashionMNIST</th>
<th>CIFAR10</th>
<th>Ret</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>8.4</td>
<td>55.5</td>
<td>51.1</td>
<td></td>
</tr>
<tr>
<td>VFlip</td>
<td>6.1</td>
<td>52.4</td>
<td>50.4</td>
<td>50.0</td>
</tr>
<tr>
<td>HFlip</td>
<td>59.6</td>
<td>40.4</td>
<td>50.0</td>
<td>50.0</td>
</tr>
<tr>
<td>SVHN</td>
<td>50.0</td>
<td>50.0</td>
<td>50.0</td>
<td>50.0</td>
</tr>
<tr>
<td>Typicality</td>
<td>50.0</td>
<td>50.0</td>
<td>50.0</td>
<td>50.0</td>
</tr>
<tr>
<td>Density-of-States</td>
<td>98.1</td>
<td>67.1</td>
<td>55.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>96.4</td>
<td>54.6</td>
<td>51.0</td>
<td>50.1</td>
</tr>
<tr>
<td></td>
<td>50.0</td>
<td>50.0</td>
<td>4.0</td>
<td></td>
</tr>
<tr>
<td>Generative-based</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Likelihood</td>
<td>8.8</td>
<td>55.4</td>
<td>51.1</td>
<td></td>
</tr>
<tr>
<td>WAIC</td>
<td>6.0</td>
<td>52.5</td>
<td>50.4</td>
<td>50.0</td>
</tr>
<tr>
<td>Typicality</td>
<td>88.6</td>
<td>40.4</td>
<td>50.0</td>
<td>50.0</td>
</tr>
<tr>
<td>Density-of-States</td>
<td>98.1</td>
<td>67.1</td>
<td>55.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>96.4</td>
<td>54.6</td>
<td>51.0</td>
<td>50.1</td>
</tr>
<tr>
<td></td>
<td>50.0</td>
<td>50.0</td>
<td>4.0</td>
<td></td>
</tr>
<tr>
<td>Reconstruction-based</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AutoEncoder (AE)</td>
<td>75.0</td>
<td>50.0</td>
<td>50.4</td>
<td></td>
</tr>
<tr>
<td>AutoEncoder Mahalanobis</td>
<td>94.9</td>
<td>79.5</td>
<td>63.0</td>
<td></td>
</tr>
<tr>
<td>MemAE</td>
<td>56.9</td>
<td>59.0</td>
<td>48.7</td>
<td></td>
</tr>
<tr>
<td>Diffusion AE + CLIP (ours)</td>
<td>99.5</td>
<td>93.7</td>
<td>55.7</td>
<td></td>
</tr>
<tr>
<td>Diffusion AE + raw pixels (ours)</td>
<td>97.9</td>
<td>50.0</td>
<td>50.0</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: AUC scores were found for each comparison and displayed in the table for FashionMNIST and CIFAR10. All comparisons made for each trained model is with OOD data. The highest value in each column is indicated by bold text numbers. The Rank column is the average rank across all 15 comparisons spanned across this table and the following table.
Table 3: AUC scores were found for each comparison and displayed in the table for CelebA and SVHN. All comparisons made for each trained model is with OOD data. The highest value in each column is indicated by bold text numbers. The Rank column is the average rank across all 15 comparisons spanned across this table and the previous table.

The diffusion autoencoder model that trained on the CLIP features had the highest average rank out of all the methods displayed in Table 2 and Table 3 with the DDPM model having the second highest overall rank. Our model that trained on raw pixel images performed poorly in comparison to the CLIP feature trained model. The raw pixel image trained model ranked seventh overall on performance out of the ten methods used for comparison. The large gap in performance between our models show that using CLIP features instead of raw pixel images may lead to better results.

When strictly comparing the results of each comparison between only our models, The model trained on CLIP features out performed the model trained on raw pixel images in all comparisons except one. When testing the trained CIFAR10 models against the horizontally flipped CIFAR10 images, the raw pixel model out performed the CLIP feature model by a small margin of one percent as seen in Table 2. The model trained on CLIP features is the clear winner between our two methods of training our model. This shows promise in the use of CLIP feature vectors to improve model performance.

By analyzing the results of Table 2 and Table 3, we see that for all datasets except for SVHN, the results of the trained model tested against the horizontally flipped images performed worse than the model tested against the vertically flipped images. This is likely due to the fact that most images in these datasets still look like the in-distribution data when flipped horizontally. For example, a face in the
CelebA dataset will still look like a proper face when flipped horizontally. Similarly, a deer in the CIFAR10 dataset will still look like a deer. The deer would just be facing the opposite direction when compared to the original image. This does not hold true when the image is flipped vertically for these datasets. For the CLIP feature trained model, the difference in accuracy between horizontally flipped images and vertically flipped images is large for FashionMNIST and CelebA. A face or a pair of pants flipped vertically would look much different from the original image so it makes sense for the trained model to be able to accurately identify the image as OOD while it would struggle to identify the horizontally flipped images as OOD due to the similarities of the flipped image to the original image. The difference in accuracy between the horizontally flipped images and vertically flipped images is smaller for the CIFAR10 data due to the fact that there are many variations of the same class leading to more ambiguity. For example, a cat can be standing in one image and curled up in another. Despite this, the vertically flipped images still had a higher degree of accuracy. The only case where the model tested against their horizontally flipped images did better than the vertically flipped images was for the SVHN dataset. This makes sense since the dataset contains numbers. There are more numbers that look the same when flipped vertically than there are numbers that look the same when flipped horizontally. The numbers one, eight and zero will likely look like in-distribution data when flipped horizontally while the same three numbers and the number three will likely look the same when flipped vertically. Similarly to the CIFAR10 dataset, there is a smaller gap in accuracy between the two sets of flipped images due to the many variations of the image of the number. The number’s pairing with other numbers and the angle of the numbers in the image may cause an increase in ambiguity.
Figure 9: The first row shows the raw pixel input image sent through the trained model of each dataset. The models used here were all trained on raw pixel images. The second row shows the recreated image from the trained model. From left to right, the images are from MNIST, FashionMNIST, CIFAR10, CelebA, and SVHN.

Figure 9 above provides a comparison of the original input image and the reconstructed image provided by the raw pixel trained model. Looking at Figure 9, we see that only MNIST and FashionMNIST were able to reconstruct the input image well with CelebA having a vague resemblance of a face. The other two datasets resemble noise. Seeing these poor input reconstruction results fall in line with the poor overall performance of the raw pixel trained models seen in Table 2 and Table 3. The poor image reconstruction of the model trained on the raw pixel images of some of the datasets is likely due to the fact that raw pixel images struggle to capture high level features of an image. Raw pixels are just that, raw pixels sent to the model. Due to the lack of feature representation and the diversity of the images of the three datasets our model performed poorly on, the reconstructed images are more reminiscent of noise than an image. MNIST and FashionMNIST are more simple and clean cut. Every image in every class for both datasets contain very similar objects and have a solid black background. There is not much diversity. In contrast, CIFAR10, CelebA, and SVHN all have varying backgrounds, object locations, object shapes, and object sizes for images within the same class. This complexity and diversity make it difficult for
the model to learn specific patterns and the data distribution of the datasets through raw pixels alone. The CelebA dataset provided slightly better image reconstruction than CIFAR10 and SVHN due to the fact that the CelebA images have less diversity than the other two datasets. CelebA images all have headshots of people roughly centered in the image while CIFAR10’s automobile class contains different models of cars from various angles. Seeing the results of the input image reconstruction of the raw pixel trained models makes the results of MNIST and FashionMNIST in Table 4 below intriguing.

<table>
<thead>
<tr>
<th>OOD Data</th>
<th>MNIST</th>
<th>FashionMNIST</th>
<th>CIFAR10</th>
<th>CelebA</th>
<th>SVHN</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>-</td>
<td>97.9</td>
<td>100</td>
<td>100</td>
<td>99.9</td>
</tr>
<tr>
<td>FashionMNIST</td>
<td>99.7</td>
<td>-</td>
<td>100</td>
<td>99.9</td>
<td>99.9</td>
</tr>
<tr>
<td>CIFAR10</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>43.4</td>
<td>65.2</td>
</tr>
<tr>
<td>CelebA</td>
<td>0</td>
<td>0</td>
<td>74.4</td>
<td>-</td>
<td>82.4</td>
</tr>
<tr>
<td>SVHN</td>
<td>0</td>
<td>0</td>
<td>42.0</td>
<td>28.1</td>
<td>-</td>
</tr>
<tr>
<td>VFlip</td>
<td>49.9</td>
<td>50.0</td>
<td>53.0</td>
<td>61.8</td>
<td>49.5</td>
</tr>
<tr>
<td>HFlip</td>
<td>49.9</td>
<td>50.0</td>
<td>50.9</td>
<td>50.8</td>
<td>49.6</td>
</tr>
</tbody>
</table>

Table 4: AUC scores for all comparison made with each of the five raw pixel trained models against their respective OOD data in this experiment. The OOD data for each dataset here comes from the test set of each dataset.

Table 4 shows the results of AUC scores for each comparison made of a trained model and their respective OOD data for the raw pixel trained models. Given the good image reconstruction capabilities of the two trained models, MNIST and FashionMNIST seen in Figure 9, it is a surprise to see an accuracy of zero when these two models were tested against CIFAR10, CelebA, and SVHN. The likelihood of an input being OOD is determined by the MSE loss value between the input image and the reconstructed image. For these two trained models, the results of the MSE loss coincidentally led to a low loss value causing the model to classify the input as in-distribution when that is not the case. This major misclassification is also seen in Figure 11 and Figure 13. The authors of [28] suggest that this phenomena is a result of disjoint support. If the support of the MNIST or FashionMNIST dataset is greater
than the support of the other three datasets, than the trained model can assign high probability to the OOD data. Support is defined as the set of possible values of the data. Alternatively, the authors of [19] suggest that poor performances may be a result of unfair biases learned by the model. Spurious correlations may be the cause of our results. Some deep networks rely on superficial features/ways to predict labels such as the image background. In cases like that, minor shifts in the background greatly deteriorates model accuracy and performance [19]. For the CIFAR10, CelebA, and SVHN models, they were able to accurately classify the MNIST and FashionMNIST datasets well with less good results for the other OOD datasets. This is likely due to the fact that the images in the MNIST and FashionMNIST are much different in comparison to the other datasets. These two datasets are more simple and clean in comparison. They do not have major deviations and image backgrounds to create additional complexity like with the other datasets. This might have made it easier for the other trained models to differentiate it as OOD.

Table 5 below show the results of AUC scores for each comparison made of a trained model and their respective OOD data for the CLIP feature trained models.

<table>
<thead>
<tr>
<th>OOD Data</th>
<th>MNIST</th>
<th>FashionMNIST</th>
<th>CIFAR10</th>
<th>CelebA</th>
<th>SVHN</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>-</td>
<td>99.5</td>
<td>99.6</td>
<td>99.9</td>
<td>99.9</td>
</tr>
<tr>
<td>FashionMNIST</td>
<td>99.9</td>
<td>-</td>
<td>99.4</td>
<td>100</td>
<td>99.9</td>
</tr>
<tr>
<td>CIFAR10</td>
<td>100</td>
<td>99.9</td>
<td>-</td>
<td>100</td>
<td>99.9</td>
</tr>
<tr>
<td>CelebA</td>
<td>100</td>
<td>100</td>
<td>99.9</td>
<td>-</td>
<td>100</td>
</tr>
<tr>
<td>SVHN</td>
<td>100</td>
<td>99.9</td>
<td>99.5</td>
<td>100</td>
<td>-</td>
</tr>
<tr>
<td>VFlip</td>
<td>82.3</td>
<td>93.7</td>
<td>68.9</td>
<td>95.2</td>
<td>58.8</td>
</tr>
<tr>
<td>HFlip</td>
<td>81.9</td>
<td>55.7</td>
<td>49.9</td>
<td>54.2</td>
<td>63.1</td>
</tr>
</tbody>
</table>

Table 5: AUC scores for all comparison made with each of the five CLIP feature trained models against their respective OOD data in this experiment. The OOD data for each dataset here comes from the test set of each dataset.

Table 5 shows the accuracy of the CLIP feature trained models when tested against other datasets as well as their respective vertically flipped and horizontally flipped images. All models performed well with near perfect results when they were tested...
against images from another dataset. The results are not as good when tested against the flipped images of its own dataset. However, these results are still much better overall than the results of the raw pixel trained models. CLIP features provide a new way of feature identification that greatly improved the results of our model when compared to the results of the raw pixel trained models. Using CLIP for feature extraction also helps eliminate the need of additional data preprocessing such as color to grayscale image conversion. Thus, allowing for ease of data comparison and model use with various types of image structures.

The AUC scores presented in the above tables are a result of using dynamically derived threshold values based on all the data of each corresponding dataset. A non-binned method is used.

Figure 10 through Figure 19 below depict histograms of negative log likelihoods assigned to each dataset by a model trained on one of the five datasets. The figures alternate between CLIP feature trained models and raw pixel trained models.

![Figure 10](image)

**Figure 10:** Histogram of the negative log likelihood values assigned to each dataset (test and train) by the CLIP feature trained MNIST model. This graph shows good separability between the in-distribution data, MNIST, and OOD data. The SVHN and FashionMNIST datasets show the most similarity to the in-distribution data of MNIST. This is likely due to the similarity in data or data structure. SVHN and MNIST both contain images of numbers and FashionMNIST and MNIST both visually look similar, black background with a single white ‘object’ roughly centered in the image.
Figure 11: Histogram of the negative log likelihood values assigned to each dataset (test and train) by the raw pixel trained MNIST model. This graph shows poor separability between the in-distribution data, MNIST, and OOD data.

Figure 12: Histogram of the negative log likelihood values assigned to each dataset (test and train) by the CLIP feature trained FashionMNIST model. This graph shows good separability between the in-distribution data, FashionMNIST, and OOD data. However, the level of separability is not as good as the results for MNIST.
Figure 13: Histogram of the negative log likelihood values assigned to each dataset (test and train) by the raw pixel trained FashionMNIST model. This graph shows poor separability between the in-distribution data and the OOD data.

Figure 14: Histogram of the negative log likelihood values assigned to each dataset (test and train) by the CLIP feature trained CIFAR10 model. This graph shows good separability between the in-distribution data and OOD data.
Figure 15: Histogram of the negative log likelihood values assigned to each dataset (test and train) by the raw pixel trained CIFAR10 model. This graph shows poor separability between the in-distribution data and OOD data.

Figure 16: Histogram of the negative log likelihood values assigned to each dataset (test and train) by the CLIP feature trained CelebA model. This graph shows good separability between the in-distribution data and OOD data.
Figure 17: Histogram of the negative log likelihood values assigned to each dataset (test and train) by the raw pixel trained CelebA model. This graph shows poor separability between the in-distribution data and the OOD data SVHN.

Figure 18: Histogram of the negative log likelihood values assigned to each dataset (test and train) by the CLIP feature trained SVHN model. This graph shows good separability between the in-distribution data and OOD data.
Figure 19: Histogram of the negative log likelihood values assigned to each dataset (test and train) by the raw pixel trained SVHN model. This graph shows good separability between the in-distribution data and the OOD data CelebA.

Figure 20 through Figure 29 below are histograms of negative log likelihood values assigned by a trained model to images from the same dataset it trained on. The figures alternate between CLIP feature trained models and raw pixel trained models.

Figure 20: Histogram of the negative log likelihood values assigned by the CLIP feature trained MNIST model tested with only MNIST images. The MNIST training set and testing set were used along with the vertically flipped images of the testing set and the horizontally flipped images of the testing set. This graph shows slight separability between the four different sets of MNIST data.
Figure 21: Histogram of the negative log likelihood values assigned by the raw pixel trained MNIST model tested with only MNIST images. The MNIST training set and testing set were used along with the vertically flipped images of the testing set and the horizontally flipped images of the testing set. This graph shows no separability between the four different sets of MNIST data.

Figure 22: Histogram of the negative log likelihood values assigned by the CLIP feature trained FashionMNIST model tested with only FashionMNIST images. The FashionMNIST training set and testing set were used along with the vertically flipped images of the testing set and the horizontally flipped images of the testing set. This graph shows some separability between the four different sets of FashionMNIST data with the vertical flipped images showing the greatest separation. This likely is due to the fact that the vertical flip of a clothing item will result in the item being upside down. This results in a more distinct difference to the in-distribution data.
Figure 23: Histogram of the negative log likelihood values assigned by the raw pixel trained FashionMNIST model tested with only FashionMNIST images. The Fashion-MNIST training set and testing set were used along with the vertically flipped images of the testing set and the horizontally flipped images of the testing set. This graph shows no separability between the four different sets of FashionMNIST data.

Figure 24: Histogram of the negative log likelihood values assigned by the CLIP feature trained CIFAR10 model tested with only CIFAR10 images. The CIFAR10 training set and testing set were used along with the vertically flipped images of the testing set and the horizontally flipped images of the testing set. This graph shows slight separability between the four different sets of the data.
Figure 25: Histogram of the negative log likelihood values assigned by the raw pixel trained CIFAR10 model tested with only CIFAR10 images. The CIFAR10 training set and testing set were used along with the vertically flipped images of the testing set and the horizontally flipped images of the testing set. This graph shows no separability between the four different sets of the data.

Figure 26: Histogram of the negative log likelihood values assigned by the CLIP feature trained CelebA model tested with only CelebA images. The CelebA training set and testing set were used along with the vertically flipped images of the testing set and the horizontally flipped images of the testing set. This graph shows slight separability between the four different sets of the data with a greater separation between the vertical flipped images and the in-distribution data much like the results of the FashionMNIST model.
Figure 27: Histogram of the negative log likelihood values assigned by the raw pixel trained CelebA model tested with only CelebA images. The CelebA training set and testing set were used along with the vertically flipped images of the testing set and the horizontally flipped images of the testing set. This graph shows no separability between the four different sets of the data.

Figure 28: Histogram of the negative log likelihood values assigned by the CLIP feature trained SVHN model tested with only SVHN images. The SVHN training set and testing set were used along with the vertically flipped images of the testing set and the horizontally flipped images of the testing set. This graph shows no separability between the four different sets of the data.
Figure 29: Histogram of the negative log likelihood values assigned by the raw pixel trained SVHN model tested with only SVHN images. The SVHN training set and testing set were used along with the vertically flipped images of the testing set and the horizontally flipped images of the testing set. This graph shows no separability between the four different sets of the data.

Through observation of the above sets of histograms of the negative log likelihood values for each trained model, we see that the CLIP feature models learned the features of their respective dataset better than the raw pixel trained models. The plots for the CLIP feature models show greater separability between the in-distribution data and the OOD data than the models trained on raw pixel images. With the CLIP feature models, you can more clearly see distinct sections for each dataset the trained model was tested on in comparison to the raw pixel trained counterparts.
4 Conclusion

In this study, we introduced OOD detection with a diffusion autoencoder. OOD detection is a method used to determine if the given data is part of the data distribution the model trained on. Knowing this information allows for a model to accept or reject the input sent to the model based on if the input is OOD. This is integral for safety critical systems. Diffusion models are a type of generative model commonly used today for image synthesis due to their ability to produce high quality images. However, diffusion models are often large, computationally expensive, and slow. Another generative model is an autoencoder. Autoencoders are fast but produce low quality images. In combining the two model structures, we developed a diffusion based autoencoder. We trained the diffusion autoencoder on each of the five datasets twice. Once with the CLIP feature vectors and another instance with the raw pixel images. We tested each model against a range of OOD data and compared the results of our model not only to each other, but also to the benchmark results found in [11]. Lastly, we analyzed the results from our experiment and determined that training our model with CLIP features produced the best results and given the accuracies produced, there is potential in using this method for OOD detection.

4.1 Future Work

Research into OOD detection, different uses of generative models, and ways of improving autoencoders are still open and ongoing research problems. One potential way of improving our model is to create a noise scheduler much like what modern diffusion models use. This will speed up the diffusion process. Another potential way of improving our model would be a more extensive hyperparameter search to further optimize the model. This may further improve the results found in this paper. Training the model more, especially the raw pixel image trained models may have led to different results. Alternatively, implementing some of the OOD detection
improvement suggestions found in [28] may also be beneficial.
Bibliography


