TU Dortmund

Master Thesis

Anomaly Detection using an Ensemble with Simple Sub-models

First Reviewer:
Prof Dr. Philipp Doebler

Second Reviewer:
M. Sc. Simon Klüttermann

Vanlal Peka

May 8, 2024

Contents

1 Introduction

Anomaly detection has been effectively used in financial fraud detection(Hilal et al., 2022), cybersecurity threat detection (Jeffrey et al., 2023), industrial equipment monitoring (Chatterjee and Ahmed, 2022), early disease detection (Fernando et al., 2021), and so on. The discipline of anomaly detection has a long history of research as well (Edgeworth, 1887), and the methods to identify anomalies are getting all the more complicated as technologies progressed and data generation exploded in the past decades[1]. The algorithms have grown from simple static threshold-based methods to more complicated user-defined threshold-based methods based on deep learning. As the algorithms get more complicated, it is getting slower and also getting harder to explain the inner workings of the algorithms. This thesis attempts to devise an algorithm that is easy to interpret without compromising on performance.

1.1 Definition of an Anomaly

Although anomaly detection as a discipline has been expanding into several domains and the detection accuracy has been improving, a concrete definition of an anomaly is still elusive because what is considered an anomaly depends on the domain. On top of that, new technologies bring new types of data (e.g. video, spatial, time series, etc.), which makes it harder to have an overarching definition.

In a broad sense, we may define anomalies as observations that fall outside a well-defined notion of normal observations. This then begs the question: what are normal observations? The answer to that, although unsatisfying, is - it depends. Additionally, some researchers differentiate an anomaly from noise or novelty: noise may be defined as unwanted data that hinders data analysis, and common options are to remove it before analyzing the data or make accommodations for it; novelty may be defined as a pattern that is new to the model but is not anomalous, and is usually incorporated into the normal pattern. Moreover, there is no consensus on whether to call it 'anomaly' or 'outlier'. Older publications used the term 'outlier' more, while newer publications used 'anomaly' more (Olteanu et al., 2023). We stuck to calling it 'anomaly' in this thesis.

To add structure to this kind of topic, Chandola et al. (2009) defined three types of anomalies: point anomaly, contextual anomaly, and collective anomaly. A point anomaly

[1] https://www.statista.com/statistics/871513/worldwide-data-created

is a single observation, from a dataset of independent observations, that is different from the majority. Credit card fraud detection using the amount spent is an example of a point anomaly. A contextual anomaly is an observation that may not be an anomaly when considered as a single observation but turned out to be an anomaly under a given context, using time or location or others. Say, the amount spent on a credit card purchase is within the threshold limit so it is not identified as a point anomaly, but that may be identified as a contextual anomaly should the purchase be made in a different country. Collective anomalies are a set of observations that look normal when viewed one by one but exhibit anomalous patterns as a group.

In addition to the three mentioned above, Ruff et al. (2021) defined two types of anomalies to address the hierarchical structure of the features: low-level sensory anomalies and high-level semantic anomalies. These two types are more relevant in high dimensions. Suppose we are trying to identify an anomaly from a dataset of pet images, the low-level sensory anomalies will be identified by comparing pixel values, and the high-level semantic anomalies will be identified by differentiating, say, a cat from a dog.

1.2 Different Labelling, Different Settings

Depending on the availability of labels in the training set, anomaly detection algorithms may be classified into three settings: supervised, unsupervised, and semi-supervised (Chandola et al., 2009). A supervised algorithm assumes the normal observations and the anomalies in the training set are labeled, while an unsupervised algorithm assumes there exists no training data or the training set is not labeled. Between supervised and unsupervised, we have a semi-supervised algorithm that assumes only the normal observations in the training set are labeled and the anomalies in the training set are not labeled. The training sets for this thesis have only normal (unlabelled) observations, so the concept of a semi-supervised algorithm is not applicable to our experiments, and our algorithm falls under the unsupervised algorithms.

1.3 Types of Anomaly Detection Algorithms

There are several anomaly detection algorithms and the number is growing. These algorithms can be grouped into the following types: density estimation and probabilistic methods like Boltzmann Machines, classification methods like One-Class SVM, reconstruction methods like Autoencoders, nearest neighbor methods like k-Nearest Neighbor,

clustering methods like DBSCAN, spectral methods like Principal Component Analysis (PCA), and statistical methods like Gaussian Mixture Model (GMM)(Chandola et al., 2009).

The performance of the algorithms depends on the dataset characteristics. Nearest neighbor and clustering-based methods suffer from the "curse of dimensionality" i.e. the distance measures in high dimensions cannot differentiate an anomaly from normal observations. Spectral methods map from a higher dimension to a lower dimension, so they address the high-dimension problem, but this is only useful if the anomalous observation is separable from normal observations in the low-dimension projection. Statistical methods are only effective for low-dimension data and when the statistical assumptions hold. These show that there may not be an anomaly detection algorithm that has high performance across any kind of dataset.

However, an interesting point is that there is a shared underlying philosophy i.e. "Virtually all outlier detection algorithms create a model of the normal patterns in the data, and then compute an outlier score of a given data point based on the deviations from these patterns. (Aggarwal, 2016)". Unsurprisingly, our anomaly detection algorithm follows the same philosophy: the model is trained on a set with only normal observations, and an outlier score is calculated for each of the observations in the test set, which is a mix of normal and anomalous observations.

1.4 Challenges in Anomaly Detection

Data has been growing not only in size but also in complexity. Most of the time, but not always, new data types and new domains require a new way of detecting anomalies. Additionally, as briefly pointed out in Section 1.1, the concept of what constitutes an anomaly expands with the advent of high dimensional data. In that section, we also explained that to identify an anomaly, we must first define what is normal, and that is not as simple as it sounds because the boundary between "normal" and "anomaly" is not always clear cut. This challenge can be complicated to address because in some domains, like fraud detection, what is considered "normal" evolves as the algorithms incorporate the novelty observations in their pipeline. The dimensionality of the data adds another layer of complications because the larger the data dimension the harder it is to learn the normal representation.

There are also domain-specific challenges. Different domains have different levels of tolerance for misidentification as well as different levels of "tightness" in defining what is normal. In medical research, the preference is to minimize false negatives because the cost of classifying an anomaly as normal can be very high, which may not be the case in analyzing stock prices, for instance. In sentiment analysis, text datasets are sparse and have high dimensions, complicating the boundary that separates normal observations and anomalies; moreover, the documents in a single category can have large variations. In credit card fraud detection and sensor networks, anomaly detection techniques are required to operate online so the algorithms have to be lightweight. Then there are challenges due to imbalanced class distribution, that is, classical machine learning algorithms are biased towards the majority class. In domains like fraud detection, the class imbalance i.e. the anomaly: normal ratio could range from 1:100 to 1:5000 (Krawczyk, 2016). Because of all these, we used a variety of datasets (Section 3.2) to gauge the algorithm performance.

1.5 Thesis Motivation

Existing unsupervised anomaly detection algorithms like One-Class Support Vector Machine (OC-SVM) (Bounsiar and Madden, 2014), local outlier factor (LOF) (Breunig et al., 2000), and other algorithms based on nearest neighbor methods are already quite powerful in low dimension. However, the performance of these algorithms is limited in high-dimension because they struggle to learn the increasingly complex separations. This challenge has been addressed with considerable success using Autoencoder-based models (Borghesi et al., 2019), but the latent space dimensions of Autoencoders are difficult to optimize.

On the other hand, we have Deep Ensemble Anomaly Detection (DEAN), an ensemble-based unsupervised anomaly detection algorithm (Klüttermann and Müller, 2022; Böing et al., 2022). The algorithm is scalable, has deep sub-models, has high variance among the predictions of the sub-models, and has a consistent anomaly score. The algorithm showed better performance (ROC-AUC) when benchmarked against DeepSVDD (Ruff et al., 2018), RandNet (Chawla and Wang, 2017), and Isolation Forest (Liu et al., 2008) on CIFAR-10 (Krizhevsky, 2009) dataset.

This thesis is motivated using DEAN, and the goal is to explore the performance of an ensemble-based unsupervised anomaly detection algorithm, but with shallow ho-

mogenous submodels. Another motivation is that there are some researches where the classical machine learning algorithms outperformed deep learning models across a range of anomaly types (Rewicki et al., 2023). To that end, we used the same loss function and anomaly score calculation from DEAN but implemented them using shallow submodels. We would call our algorithm SEAN i.e. a shallow version of DEAN.

1.6 Organization

Section 2 explains the DEAN algorithm in more detail. Section 3 highlights the motivations for carrying over the concepts from DEAN, and also explains the datasets we used for training and the benchmarking process we followed. Section 4 explains the building blocks of our algorithm starting from the preprocessing steps, all the way to the ensemble of weak models. Section 5 walks through the setup of the experiments, shows the results and interpretations, and then compares the performance with the competitors. Finally, Section 7 summarizes the thesis and closes with further research directions.

2 Deep Ensemble Anomaly Detection (DEAN)

This section explains the components of the DEAN algorithm and their impact on the characteristics of the algorithm. We will start with the three key equations that pin the algorithm.

- The loss function of a (DEAN) submodel:

$$L_{DEAN} = \sum_{\vec{x} \in X_{train}} (f(\vec{x}) - g(\vec{x}))^2 \tag{1}$$

  where $f(\vec{x})$ is a neural network that is learned from the test data, and feature bagging is used to ensure each submodel learns a different representation; $g(\vec{x})$ is an arbitrary function or a constant that $f(\vec{x})$ is compared with. A large value of L_{DEAN} implies the observation $\vec{x}$ is anomalous.

- The equation to calculate the anomaly score of the ith submodel:

$$S_i = \left| \hat{f}\left(X_{test}\right) - \text{mean}\left(\hat{f}(X_{train})\right) \right| \tag{2}$$

where $\hat{f}(.)$ is a prediction function learned using the L_{DEAN} loss function in Equation 1.

- The anomaly score of the ensemble of N submodels:

$$F = \frac{1}{N}\sum_{i=1}^{N} S_i \tag{3}$$

The $g(\vec{x})$ function in Equation 1 affects the flexibility of the (neural network) submodels. For instance, Autoencoders can be imagined as a DEAN model where $g(\vec{x}) = \vec{x}$ i.e. learning the identity function. We can also simplify $g(\vec{x})$ to improve model performance. However, an oversimplified function like $g(\vec{x}) = 0$ learns a trivial solution. Therefore, the experiments in the DEAN paper were run with $g(\vec{x}) = 1$, so the loss function of a submodel essentially simplifies to,

$$L_{DEAN} = \sum_{\vec{x} \in X_{train}} (f(\vec{x}) - 1)^2 \tag{4}$$

The simplified loss function in Equation 4 has several benefits. First, it is easier to optimize, using a neural network, when compared with a more complicated $g(\vec{x})$. Second, it results in fewer local minima, and the local minima, if they appear, can be noticed easily. Third, $g(\vec{x}) = 1$ does not add hyperparameters to the ensemble, simplifying training. Fourth, it has less bias because $g(\vec{x}) = 1$ does not change with the observations. Finally, the simplified loss function combined with feature bagging in each submodel enables the ensemble to scale well (with high dimension). (Klüttermann and Müller, 2022)

3 Premise of the Thesis

This section explains in detail the objective of the thesis, the processes we followed to measure the model performance, and the reasons for following the processes.

3.1 Objective

As mentioned in Section 1.5, this thesis was motivated by the DEAN algorithm to build an ensemble of homogeneous shallow submodels. To that end, we replaced the neural

network submodels of DEAN with simpler submodels, and our submodels use the same DEAN equations from Section 2. The thinking is that using these equations, albeit in shallow submodels, will allow us to reap the benefits of the DEAN model.

- The submodel loss function has the same notations:

$$L_{DEAN} = \sum_{\vec{x} \in X_{train}} (f(\vec{x}) - 1)^2 \tag{5}$$

  where X_{train} is a training set with no anomaly, and $f(\vec{x})$ is a shallow submodel, as opposed to a neural network submodel in DEAN.

- The equation for the ith submodel anomaly score remains the same as DEAN:

$$S_i = \left| \hat{f}\left(X_{test}\right) - \text{mean}\left(\hat{f}(X_{train})\right) \right| \tag{6}$$

  where X_{test} contains normal observations and anomalies.

- The equation for the anomaly score of the ensemble of N submodels also remains the same:

$$F = \frac{1}{N} \sum_{i=1}^{N} S_i \tag{7}$$

Our experiments also incorporate the various combinations of the 'building blocks' listed in Section 4. The homogeneous weak submodels were 'bagged' to form an ensemble, and an anomaly score and an AUROC were calculated (Please refer to Section 4.4 for different methods that can be used to combine the submodels). The model was then benchmarked against DEAN, Isolation Forest, KNN, and CBLOF.

From the list of anomaly types explained in Section 1.3, this thesis focused on point anomalies for tabular datasets and high-level semantic anomalies for image datasets. We also used different types of datasets to mitigate challenges mentioned in Section 1.4 i.e. the challenges coming from the normal-anomaly boundary and the class distribution imbalance. The anomalies in some of our datasets are human experts labeled, but in some datasets, we labeled the anomalies ourselves to have a more balanced class distribution. Finally, all of our trainings were done with no anomalies. Section 3.2 explained the datasets and the criteria used in each dataset to label anomalies.

3.2 Datasets

To address the challenges mentioned in Section 1.4 and to get a better picture of the anomaly detection robustness, it is a usual practice to use different types of datasets. Ruff et al. (2021) grouped the popular benchmarking datasets into three types depending on why observations are labeled as anomalies:

- k-classes-out: A programmer (not necessarily a domain expert) sets which observations are anomalies or normal. For example, labeling a picture of an airplane in the CIFAR-10 dataset as a normal class and the others as anomalies. Out of the three, these datasets, and the anomaly labeling approach may be the least effective in identifying out-of-distribution data.
- Synthetic[2]: Anomalies are synthetically generated from the original datasets e.g. MNIST-C dataset, which is a corrupted version of the standard MNIST dataset. These datasets may give us a better benchmark than the k-classes-out datasets, but they still does not give us the characteristics of true anomalies.
- Real-world: The anomalies are labeled by a human expert e.g. Credit-card Fraud dataset. Out of the three types, these datasets are the best benchmarks to check if an algorithm learns the characteristics of the true anomalies.

In light of the types of datasets listed above, we used the following datasets to get a benchmark on out-of-distribution robustness in our model and in the competitors:

- MNIST (LeCun et al., 2010): This dataset is a collection of 70,000 28x28 grayscaled images of handwritten digits from 0 to 9. There are 10 classes i.e. each digit is a class. There are a total of 60,000 training images i.e. 6,000 training images per class, and 10,000 test images i.e. 1,000 test images per class. For the experiments, an arbitrary class will be set as normal and the remaining class as anomalies.
- CIFAR-10 (Krizhevsky, 2009): This dataset is a collection 60,000 32x32 colored images of 10 classes. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 50,000 training images i.e. 5,000 training images per class, and 10,000 test images i.e. 1,000 test images per class. Similar to MNIST, in our experiments, an arbitrary class was set as normal and the remaining class as anomalies.

[2]The algorithm was not run on synthetic dataset due to schedule constraints.

- Credit card fraud (Pozzolo et al., 2015): This dataset has two days' credit card transactions of European cardholders in September 2013. It has 284,807 transactions and 492 are frauds i.e. 0.172% anomaly.
- Satellite (Goldstein and Uchida, 2016): A dataset of 36 features extracted from satellite observations. The dataset has 5025 normal observations and 75 randomly sampled anomalies i.e. 1.49% anomaly.

Dataset	**Type**	**Classes**	**Observations**	**Features**	**Anomaly**
CIFAR-10	Image	10	60,000	32x32	50%
MNIST	Image	10	70,000	28x28	50%
Credit Card Fraud	Tabular	2 (Fraud, Non-Fraud)	284,807	30	0.172%
Satellite Soil Quality	Tabular	2 (Normal, Anomaly)	5,100	36	1.49%

Table 1: Summary of Datasets

3.3 Performance Measure

There are two popular measures to evaluate the performance of classification algorithms: area under the Receiver Operating Characteristic (ROC) curve, and area under the Precision-Recall (PR) curve. These measures also apply to the anomaly detection algorithms since we classify normal and anomalous classes, similar to the general classification problems.

To explain the two curves, let's start with a confusion matrix (Figure 2). The values in the confusion matrix i.e. true positive, false positive, false negative, and true negative are the number of anomalies correctly predicted as anomalies, observations predicted as anomalies but are normal, the observations predicted as normal but are anomalies, and normal observations correctly predicted as normal, respectively.

		Ground Truth: Positive (Anomaly)	Ground Truth: Negative (Normal)	
Prediction	Positive (Anomaly)	True Positive (TP)	False Positive (FP)	Precision = $\frac{TP}{TP+FP}$
	Negative (Normal)	False Negative (FN)	True Negative (TN)	
		True Positive Rate (TPR) or, Recall = $\frac{TP}{TP+FN}$		

Table 2: The confusion matrix highlights the terms for ROC and PR curves. Recall is also called True Positive Rate or Sensitivity. False Positive Rate is calculated as (1-Precision), and Precision is also called Specificity or True Negative Rate.

The ROC curve is a plot of the True Positive Rate (TPR) against the False Positive Rate (FPR=1 - Precision); plotted with the False Positive Rate on the X-axis, and the True Positive Rate on the Y-axis. AUC-ROC ranges from 0 to 1, where 0.5 corresponds to random chance, and 1 represents a perfect classifier.

The PR curve is a plot of Precision against Recall; plotted with Recall on the X-axis, and Precision on the Y-axis. Precision measures the accuracy of anomaly predictions, while Recall gauges the ability to capture all anomalies.

For a binary classification task with class imbalance, a popular perspective is that the PR curve is a better option than the ROC curve for model comparison. A closer inspection, however, showed that the PR curve is no better than the ROC curve in those situations (McDermott et al., 2024).

3.4 Competitors for Benchmarking

We compared the model performance with a deep algorithm (DEAN), an ensemble ensemble algorithm (Isolation Forest), a nearest-neighbor algorithm (KNN), and a cluster-based algorithm (CBLOF). DEAN is explained in Section 2 above so we do not expand on that here, but we briefly expand on the other algorithms below:

- Isolation Forest: The algorithm begins with a decision tree that isolates anomalies by defining a path from the root to a leaf node. The assumption is that anomalies have shorter paths, therefore, they are easier to isolate. The final model is an ensemble of these decision trees, and the anomaly score of an observation is cal-

culated by averaging its path lengths in all of the trees. Isolation Forest scales in linear time with the size of the dataset, and it performed quite well in high dimensions. However, because the decision boundaries are either vertical or horizontal, there can be regions where these boundaries cross each other, and these regions are then assumed to be normal regions, which may result in misleading anomaly scores for the sparse observations in these regions.

- k-Nearest Neighbors (KNN): The KNN algorithm can be used for both supervised and unsupervised tasks. This thesis used the unsupervised 'variety'. Unlike supervised KNN, where the goal is to classify data points based on labeled training examples, unsupervised KNN aims to partition data into clusters based solely on the input features without any predefined labels. KNN is a non-parametric algorithm, easy to implement and interpret, and doesn't require a training phase; it simply stores the training data and performs computations during the prediction phase. However, the algorithm does not scale well with larger datasets or higher dimensions (it suffers from the curse of dimensionality). It is sensitive to noise, and it is also sensitive to the value of 'K' i.e. the number of nearest neighbors.
- Cluster-Based Local Outlier Factor (CBLOF): The CBLOF algorithm combines aspects of both clustering and density-based methods. It aims to identify local outliers by considering the density of clusters in the feature space. It starts by creating clusters and assigning the observations to one of the clusters, and then the density of each cluster is calculated. The anomaly score of an observation is calculated based on its distance to the centroid and the cluster's density. CBLOF is quite effective in datasets with varying densities, it is also computationally efficient, especially if the clustering algorithm used is scalable. On the other hand, it suffers from the curse of dimensionality as a consequence of the distance-based anomaly score calculation, and the performance is also dependent on the clustering method used.

3.5 Source Code

The code is available at https://github.com/vanlalpeka/thesis. It is written in Python 3 (Van Rossum and Drake, 2009), and in addition to numpy library (Harris et al., 2020), "imgaug" package is used for image pre-processing, scikit-learn (Pedregosa et al., 2011) for feature selection, features bagging, and the submodels.

4 Methodology

Our algorithm has four stages: pre-processing, feature selection, feature bagging, and the ensemble. The following sub-sections explain the stages in detail, including the configuration options at each stage.

4.1 Pre-processing

Normalization and standardization options are available for both the image and the tabular datasets. In addition, there are six pre-processing options for the image datasets i.e. grayscaling, contrast enhancement, noise reduction, edge detection, data augmentation, and skeletonization.

4.1.1 Normalization

Normalization transforms the (numerical) values of features, or columns in the dataset, into a specific range, for instance, a [0,1] range as we do in this thesis. This is done to ensure that different features of a dataset are on a similar scale, preventing certain features from dominating others in machine learning algorithms that are sensitive to the scale of input features. For instance, algorithms that use distance-based metrics (e.g., k-nearest neighbors) or optimization algorithms (e.g., gradient descent) are sensitive to the scale of input features, and training time increases when different features have different ranges so normalizing the features improves the performance and training stability of the machine learning models.

Normalization is a good choice if the data is approximately uniformly distributed in the given range and if there are few extreme values (or none).

For a given feature X that has a minimum value of X_{min} and a maximum value of X_{max}, the normalized feature is calculated as:

$$X_{normalized} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{8}$$

4.1.2 Standardization

Standardization, also known as Z-score normalization, is a process of rescaling the (numerical) features of a dataset so that they have the properties of a standard normal

distribution with a mean (μ) of 0 and a standard deviation (σ) of 1. Similar to normalization, the features are standardized to prevent certain features from dominating the other features. Also similar to normalization, standardization is a good choice if there are few extreme values (or none at all). However, standardization is more robust to extreme values.

For a given feature X that has a mean of μ_X and a variance of σ_X, the standardized feature is calculated as:

$$X_{standardized} = \frac{X - \mu_X}{\sigma_X} \tag{9}$$

4.1.3 Grayscaling

Grayscaling converts the images from three channels (red, green, and blue) to a single channel. This significantly reduces the total features of the images and therefore reduces the training time.

There are several grayscale conversion methods: Gleam, Intensity, Luminance, Luma, etc., but the consensus is that the type of grayscale conversion has little impact on image recognition performance so we chose a version of the *Luma* grayscale conversion that is used in high-definition televisions (HDTVs) (Jack, 2007).

Suppose R, G, and B represent red, green, and blue channels respectively; then, the *Luma* grayscale conversion function $\mathcal{G}$ is given as:

$$\mathcal{G} = 0.2126R + 0.7152G + 0.0722B \tag{10}$$

4.1.4 Contrast Limited Adaptive Histogram Equalization (CLAHE)

A histogram of an image gives us information on the distribution of pixels in terms of their brightness, where darker pixels are on the left side of the histogram and brighter pixels are on the right. If an image is too bright, the histogram will be squeezed to the right i.e. high values. If the image is too dark, it will be squeezed to the left. A technique to increase contrast in these situations is to spread the histogram across the pixel intensity range i.e. 0 to 255. Histogram Equalization is such a technique.

The downside of Histogram Equalization is that it performs global equalization of the histogram, which introduces noise. Contrast Limited Adaptive Histogram Equalization

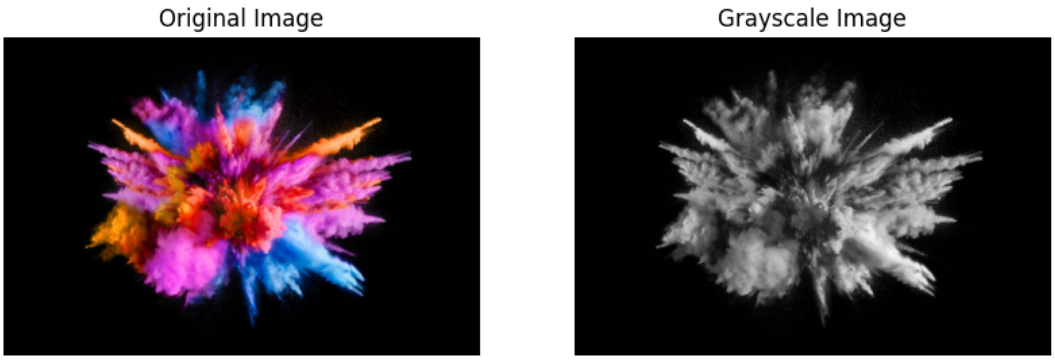

Figure 1: An image before and after applying the grayscale conversion following Equation 10.

(CLAHE) builds on Histogram Equalization but instead of global histogram equalization, it performs the equalization locally and limits the amplification of noise. This noise-limiting characteristic is why we chose CLAHE over other contrast enhancement methods.

CLAHE takes two parameters: contrast limiting threshold and (local) tile size. The algorithm involves three key steps: local enhancements, limited contrast, and interpolation. First, CLAHE divides the image into smaller tiles or patches and performs histogram equalization independently on each of these tiles. This local adaptation prevents over-amplification of intensities in the presence of noise, which can occur with global histogram equalization. Second, CLAHE introduces a contrast limit parameter to prevent excessive amplification of local contrasts. If a pixel value surpasses this limit after the local histogram equalization, the pixel values are redistributed (not clipped) to limit the enhancement. This ensures a controlled and balanced improvement in contrast. Third, as the local enhancements are performed independently on each tile, there might be noticeable separations between adjacent tiles. CLAHE uses an interpolation method that smoothens these transitions, providing a visually cohesive result (Figure 2).

4.1.5 Gaussian Filter for Noise Reduction

Noise in images can lengthen training duration and reduce prediction accuracy. Therefore, noise reduction is important to improve model performance. There are several methods for noise reduction. Popular amongst them are convolution-based filters like mean filter, medial filter, and Gaussian filter. A mean filter takes the average of all the pixels surrounding a given center pixel. Although this makes the calculation simple and

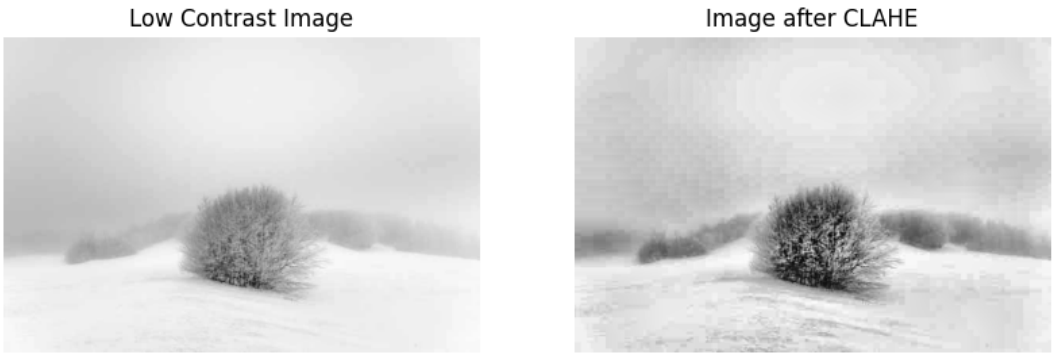

Figure 2: An image before and after applying CLAHE. The image contrast is improved after CLAHE.

quick, the filtered images are blurry around the details where dark and bright pixels are next to each other. A median filter takes the median of the pixels surrounding the center pixel instead of their mean. This prevents blurring but the computations are more expensive when compared to the mean filter. A Gaussian filter is similar to the mean filter but instead of taking the simple average, the pixels are weighted in a normal distribution (Figure 3) with the mean of the distribution coinciding with the center pixel. This makes the Gaussian-filtered images less blurry than mean-filtered images, and also the computation cost is less compared to the median filter, which is why we chose to use the Gaussian filter.

A Gaussian filter is characterized by a kernel, which is essentially a matrix of weights. The weights are determined using a Gaussian function. For instance, a 5x5 Gaussian filter kernel with $\sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ is given by:

$$\frac{1}{273} \times \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

The filter moves over each pixel in the image, and for each pixel, it calculates a weighted average of its neighboring pixels. The Gaussian function determines the weights, with higher weights given to pixels closer to the center. The weighted average is computed by performing a convolution operation between the image and the Gaussian kernel. The

resulting value at each pixel location is a smoothed version of the original pixel value. Since the Gaussian filter gives more importance to nearby pixels, high-frequency noise (spikes or variations) tends to get averaged out, leading to a smoother image.

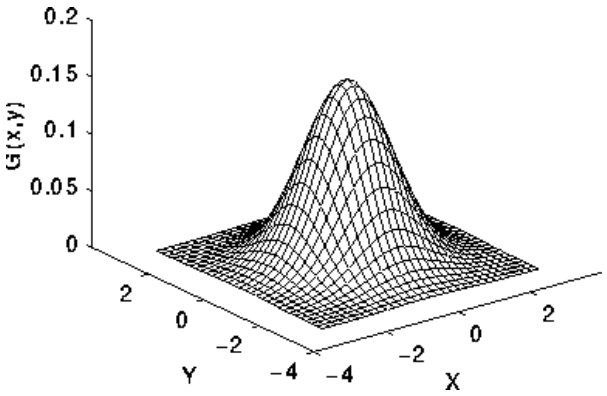

Figure 3: 2-D Gaussian distribution with $\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ and $\sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

4.1.6 Canny Filter for Edge Detection

Edge detection is important in image processing because the edges provide crucial cues for the subsequent processing steps. There are different varieties of edge detection methods: differentiation based, anisotropic diffusion based, active contour based, fuzzy logic based, spatial-frequency based, statistics based, sub-pixel interpolation based, and deep learning based methods. Among these methods, the deep learning based methods have the highest accuracy but the highest computational complexity (Jing et al., 2022). Out of the many edge detection methods, we chose Canny's criteria (Canny, 1986) because it localizes edges fairly well, is also robust against noise, and is computationally lightweight. It is also based on a first-order derivative.

The Canny filtering works in five steps and the algorithm contains a few parameters: the size of the Gaussian filter, and the minimum and the maximum threshold for the last step i.e. hysteris thresholding. First, noise reduction is done using Gaussian smoothing to minimize the impact of noise on the edge detection process. Second, gradients are calculated using a Sobel filter, for instance. The gradient magnitude is computed as the square root of the sum of squares of the horizontal and vertical gradients. Third, non-maximum suppression is used to thin the edges, or in other words, to keep only the local maxima in the gradient magnitude along the direction of the gradient. This process helps in preserving only the most significant edges and suppressing weaker,

less important ones. Fourth, double thresholding is performed to classify edge pixels into three categories: strong edges, weak edges, and non-edges. Pixels with gradient magnitudes above a high threshold are classified as strong edges, and those below a low threshold are classified as non-edges. Pixels with gradient magnitudes between the high and low thresholds are classified as weak edges. Finally, hysteris thresholding is used to build continuous edges i.e. connecting weak edges to strong edges. If a weak edge pixel is connected to a strong edge pixel, it is considered part of the edge. The idea is to trace a path along the edges by accepting weak edges that are part of the same edge structure as strong edges. The final result is a binary image with white pixels representing edges and black pixels representing non-edges as shown in Figure 4.

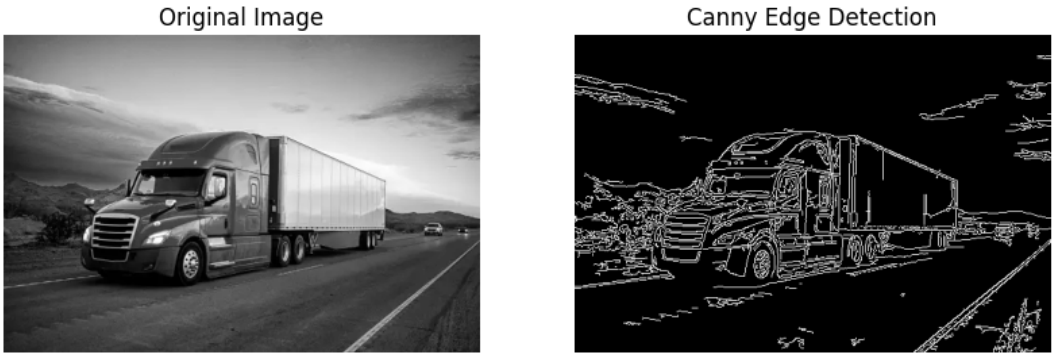

Figure 4: An image before and after applying the Canny filter for edge detection.

4.1.7 Image Augmentation

Image augmentation is a technique widely used in machine learning to artificially increase the diversity of a dataset by applying various transformations to the existing images. Introducing diversity to the training dataset minimizes the risk of overfitting and improves the model's generalization to unseen images. In addition, augmented images expose the model to different variations, making it more robust to changes in lighting conditions, viewpoints, and other factors present in real-world scenarios.

There may be several motivations for augmenting images, for instance, model overfitting, domain shifts between train and test datasets, basic variations like brightness and contrast, class imbalance, and too few images (per class). In this thesis, we split the CIFAR-10 and the MNIST images such that domain shifts and class imbalance are not the main concerns. The datasets also have a good amount of images per class so there is less concern regarding having too few images. Therefore, our main motivations for

augmenting images in this thesis are to prevent overfitting and to train a model that is more robust to the basic image variations.

Deep learning models like Generative Adversarial Network (GAN) can be used to augment images but they consume almost 3X more computation time than traditional augmentations, and on top of that, the performance is not so much better than the traditional methods (Perez and Wang, 2017). Therefore, we opted for the traditional methods in this thesis (Figure 5) and utilized random combinations of the following augmentation techniques:

1. Flipping and Rotation: Images can be horizontally or vertically flipped and rotated to simulate different viewpoints and orientations.
2. Translation: Shifting the image horizontally or vertically helps the model become invariant to changes in position.
3. Zooming: Scaling in or out provides the model with variations in the size of objects in the image.
4. Shearing: Shearing involves tilting the image along one of its axes, introducing distortion.
5. Crop: Cropping allows the model to focus on a specific area within an image and simulate different scales and positions of objects within an image.

Although image augmentation is a powerful technique for enhancing the performance and generalization of machine learning models, excessive augmentation might lead to overfitting, so careful selection and customization of augmentation techniques are essential for achieving optimal results.

4.1.8 Skeletonization

The goal of image skeletonization is to reduce the shape of objects within an image to a simplified, one-pixel-wide representation known as the "skeleton". Skeletons are useful for recognizing and classifying shapes. The simplified representation allows for efficient and robust shape matching.

The various methods to "skeletonize" images are listed below:

1. Sequential Thinning: Algorithms iteratively remove pixels from the boundary of the objects until a one-pixel-wide skeleton is obtained (Zhang and Suen, 1984).

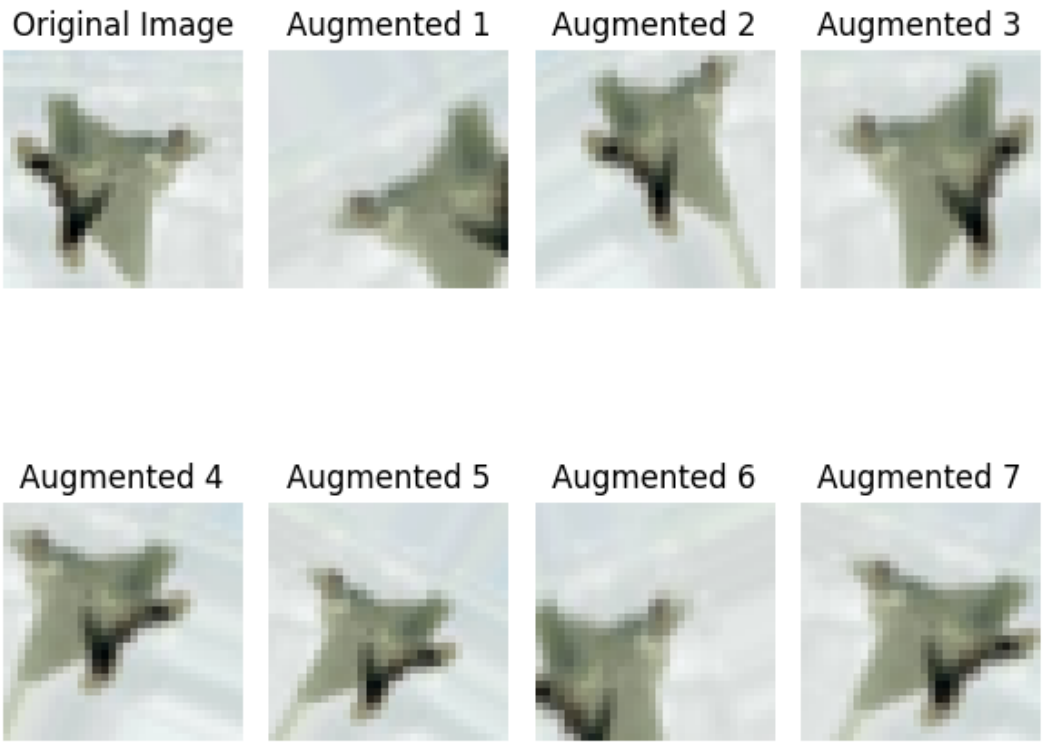

Figure 5: The original image is a random image of an airplane from the CIFAR-10 dataset. The other images are the augmented images of the original. The augmentations are done using random combinations of the traditional augmentation methods.

2. Medial Axis Transform: Skeletons can be derived from the distance transform, where the medial axis represents the center-lines of objects.
3. Morphological Operations: Morphological operations like hit-or-miss transform can be employed for skeletonization.

Among the skeletonization methods above, this thesis used the scikit implementation that follows the sequential thinning method.

4.2 Feature Selection

High-dimensional data causes regression-based algorithms to overfit easily.

The feature selections explored in this thesis are Principal Component Analysis (PCA), Independent Component Analysis (ICA), Non-negative Matrix Factorization (NMF), t-distributed Stochastic Neighbor Embedding (t-SNE), autoencoder, and a combination of

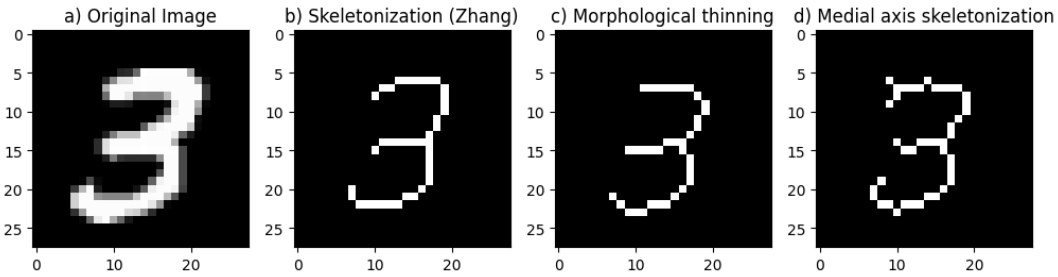

Figure 6: The various skeletonization methods in action: The original image is a random image (28x28 pixels) from the MNIST dataset.

Zero-phase component analysis (ZCA) pre-whitening and Restricted Boltzmann Machine (RBM). In each of these options, we set the default percentage of selected features to be 20% of the original features, except for t-SNE where we selected three features regardless of the number of the original features.

4.2.1 Principal Component Analysis (PCA)

The fundamental concept behind PCA is to reduce the features of a dataset while capturing the maximum information by finding the so-called "principal components" of the data, which are linear combinations of the original features. In other words, the goal is to identify the first k principal components that explain the most amount of variance in the data (Fig. 8). Geometrically, this is akin to trying to find a new feature space (principal axes) to project the data from the original feature space such that the new feature space best explains the variance in the data.

Suppose we have a dataset X with n observations (samples) and p features (variables).

$$X_{(n,p)} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1p} \\ x_{21} & x_{22} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{np} \end{bmatrix}$$

One way to solve a PCA problem is using singular value decomposition of the data matrix $X_{(n,p)}$, as listed below:

1. Standardize each feature of the data so that high variance features do not dominate the other features: $Z_i = \frac{X_i - \mu_i}{\sigma_i}$, where $i = 1, ..., p$, μ_i and σ_i are mean and variance of the ith feature respectively.

2. Perform Singular Value Decomposition (SVD) on $Z_{(n,p)}$ i.e.

$$Z_{(n,p)} = U_{(n,p)}\Sigma_{(p,p)}V^T_{(p,p)}$$

where $U_{(n,p)}$ is called the right singular vectors of $Z_{(n,p)}$ and the columns are orthogonal unit vectors of length n; $\Sigma_{(p,p)}$ is called the singular values of $Z_{(n,p)}$ and it is an *n-by-p* diagonal matrix of the square roots of the eigenvalues; $V_{(p,p)}$ is called the right singular vectors of $Z_{(n,p)}$ and it is a *p-by-p* matrix whose columns are orthogonal unit vectors of length p, these p vectors are the principal axes i.e. the new coordinate system.

3. Calculate the first k principal components using the equation:

$$P_{(n,k)} = U_{(n,k)}\Sigma_{(k,k)}$$

where $U_{(n,k)}$ is the first k columns of $U_{(n,p)}$, and $\Sigma_{(k,k)}$ is the (k, k) upper-left part of $\Sigma_{(p,p)}$. The k principal components are uncorrelated (orthogonal) to each other and the first principal component corresponds to the axis of the highest variance. Fig. 7 shows the principal components from the MNIST dataset projected onto the original features.

Algorithm 1: PCA algorithm to determine the principal components

Input: Number of desired principal components: k

Input: Dataset: $X_{(n,p)}$

Output: Principal Components: $\mathrm{P}_{(n,k)}$

1 Standardize: $\mathrm{Z}_i = \frac{X_i - \mu_i}{\sigma_i}$

2 Solve using SVD: $\mathrm{Z}_{(n,p)} = U_{(n,p)}\Sigma_{(p,p)}V^T_{(p,p)}$

3 $\mathrm{P}_{(n,k)} = U_{(n,k)}\Sigma_{(k,k)}$

Besides SVD, the PCA problem can also be solved using eigen-decomposition of the covariance of the standardized data matrix, $C = \frac{1}{n-1}Z^TZ$. Mathematically, there is no difference between the two approaches. However, there are efficient algorithms for SVD that avoid the calculation of the covariance matrix C, and these calculations are more stable compared to the eigen-decomposition method. Therefore, SVD is now the standard method to solve the PCA problem. We are using PCA implementation in

scikit, where SVD is solved using using LAPACK ("Linear Algebra Package") (Halko et al., 2010).

Although solving for the principal components (using SVD) is quite efficient, the drawback of PCA is that the principal components are not interpretable. The principal components are linear combinations of the original features so they are entirely different from the original features, making it difficult to interpret the principal components and to identify which of the original features are important. Other drawbacks of PCA are that the data is assumed to be normal along the features, and also that the features are assumed to be linearly related; both of which may not always apply.

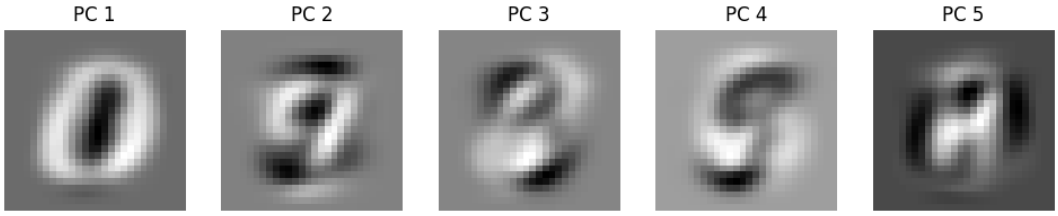

Figure 7: MNIST "eigendigits": Principal components in the original feature space, representing the directions (vectors) of maximum variance in the MNIST data. The leftmost principal component explained the most amount of variance in the data.

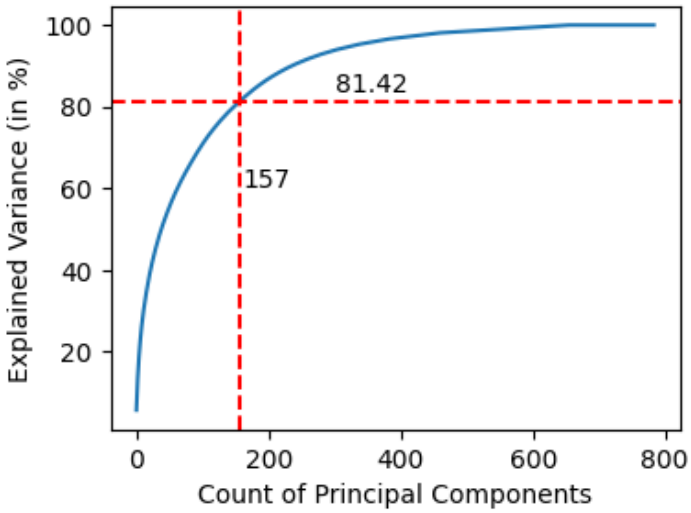

Figure 8: Variance of the MNIST training data explained by the principal components: 157 principal components, which is roughly 20% of the total original features (28x28=784), explained 81.42% of the variance in the MNIST training data (60K images).

4.2.2 Independent Component Analysis (ICA)

Let's briefly juxtapose Independent Component Analysis (ICA) with PCA for clarity: first, the goal of PCA is to find the principal components that are uncorrelated (orthogonal) to each other, while the goal of ICA is to find the components that are (statistically) independent; second, the principal components in PCA are linear combinations of the original features, while the components in ICA are the (assumed) independent latent features that generated the original features; third, PCA assumes normality of the data, while ICA assumes non-normality (to be specific, the components in ICA can still be identified if only one independent component is Gaussian, but the components can not be identified if more than one component is Gaussian); fourth, PCA may be thought of as 'compressing' the information along the principal components, while ICA 'separates' the information; finally, PCA determines the order of magnitude and the variance of the principal components, while ICA cannot determine these two for the (independent) components.

Now, suppose we have a dataset X with n observations (samples) and p features (variables).

$$X_{(n,p)} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1p} \\ x_{21} & x_{22} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{np} \end{bmatrix}$$

The ICA problem can be written in a matrix presentation as below:

$$X_{(n,p)} = A_{(n,n)} S_{(n,p)}$$

where A is the mixing matrix, and S is the independent components matrix. The challenge then is to calculate $A_{(n,n)}$ and $S_{(n,p)}$ while observing $X_{(n,p)}$ only. This can be achieved by assuming non-Gaussian statistically independent components.

Suppose we estimated the the mixing matrix A, we can take its inverse, say, W, and calculate the independent components as:

$$S_{(n,p)} = W_{(n,n)} X_{(n,p)}$$

There are several algorithms to solve for ICA e.g. infomax, FastICA, JADE, and kernel-independent component analysis. We are using the FastICA[3] implementation in *scikit* (Hyvärinen and Oja, 2000).

The Central Limit Theory (CLT) implies the sum of multiple random variables is more Gaussian than the random variables themselves. In other words, CLT implies the original features in our datasets are more Gaussian than the independent latent features. With that in mind, there are two approaches to identifying the latent features: minimizing the mutual information of the components or, maximizing the non-Gaussianity of the components. FastICA follows the second approach, which we explain below and also highlighted in Algorithm 2.

To solve the ICA problem and to identify the independent components, we have to find a weight vector w such that the projection $w^T X_{(n,p)}$ maximizes the non-Gaussianity. This is achieved by performing Newton's optimization on the non-linear function $G(w^T X_{(n,p)}) = logcosh(w^T X_{(n,p)})$.

[3]FastICA convergence speed is cubic, whereas ICA algorithms based on (stochastic) gradient descent methods have only linear convergence.

Algorithm 2: FastICA algorithm to determine the independent components

Input: Number of desired independent components: k

Input: Input dataset: $X_{(n,p)}$

Output: Independent Components: $S_{(k,p)} = W^T_{(n,k)} X_{(n,p)}$

1 Center[a] $X_{(n,p)}$: $x_{ij} = x_{ij} - \frac{1}{n}\sum_{k=1}^{n} x_{kj}$, $i \in n, j \in p$

2 Whiten[b] using eigenvalue decomposition on the covariance matrix of the centered $X_{(n,p)}$: $X_{(n,p)} = D^{\frac{-1}{2}} E^T X_{(n,p)}$

3 Initialize the weight matrix: $W_{(n,k)} = 0$

4 **for** $i \leftarrow 1$ **to** k **do**

5 Create a random vector of weight w_i of size n

6 Perform a Newton iteration on w_i. Approximately: $w_i = \mathbf{E}[X_{(n,p)} G'(w_i^T X_{(n,p)})] - \mathbf{E}[G''(w_i^T X_{(n,p)})] w_i$

7 Perform a deflationary orthogonalization on w_i using the current W: $w_i = w_i - \sum_{m=1}^{i-1}(w_i^T w_m) w_m$

8 Normalize w_i i.e. $w_i = \frac{w_i}{||w_i||}$

9 **if** *w is still changing* **then**

10 Go back to step 6

11 **end**

12 **else**

13 $W_{(n,k)}[i] = w_i$

14 **end**

15 **end**

16 Return $S_{(k,p)} = W^T_{(n,k)} X_{(n,p)}$

[a] This is required because we need $\mathbf{E}(X_{(n,p)} X^T_{(n,p)}) = I$ for FastICA to work

[b] D is the diagonal matrix of eigenvalues and E is the matrix of eigenvectors

4.2.3 Non-negative Matrix Factorization (NMF)

NMF approximates a matrix $X_{(n,p)}$ with n observations and p features such that $X_{(p,n)} \approx W_{(n,k)} H_{(k,n)}$, where W and H are low-ranked non-negative matrices, and $k << n$ and $k << p$. Geometrically, $X^T_{(n,p)}$ is projected onto a new subspace defined by the k columns of $W_{(p,k)}$, and then $H_{(k,n)}$ is the low-dimension equivalent of the high-dimension dataset $X^T_{(n,p)}$ in the new subspace. Say, X is a dataset of facial images where a row is a flattened image, then W is a matrix of parts of the face like an eye, an ear, and a nose; and H is a matrix that specifies which parts of the face are present in an image. This interpretability

is the reason NMF has become quite popular, for instance, in image processing (Gillis, 2020).

Let $V_{(p,n)} = X^T_{(n,p)}$ for ease of notation, then the NMF problem is to find W and H that minimizes the Frobenius norm:

$$||V - WH||^2_{Frobenius} \quad s.t. \quad W \geq 0, H \geq 0 \tag{11}$$

Other divergence measures can be used for the NMF problem formulation but the Frobenius norm is quite commonly used because it corresponds to i.i.d. Gaussian noise assumption and it leads to smooth optimization.

Unlike PCA, NMF does not require the transformed features to be orthogonal to each other. However, there are two challenges with NMF: first, the problem is NP-hard in general, and second, the solution is not unique. Therefore, it can be computationally expensive, especially for large datasets.

We are using a scikit implementation that solves the Frobenius norm in Eqn. 11 with a two-block coordinate descent method, which works by minimizing one coordinate (matrix) at a time. To this end, the method minimizes W first and then H, and repeats to find a local minimum, as such, there is no guarantee of finding global minima. Since NMF is an NP-hard optimization problem, the iterative algorithm is sensitive to the initial value of W and H. A common random initialization is to use a uniform distribution in the range [0,1].

Algorithm 3: Two-block coordinate descent algorithm to solve the NMF problem

Input: Number of desired independent components: k

Input: Input dataset: $V_{(p,n)}$

Output: Non-negative Matrix of Rank-k $\approx \mathrm{W}_{(p,k)} H_{(k,n)}$, where $W \geq 0, H \geq 0$

Randomly initialize the matrices s.t. $W^{(0)} \geq 0, H^{(0)} \geq 0$

for $i \leftarrow 1, 2, ...$ **do**

  $W^{(t)} = update(V, W^{(t-1)}, H^{(t-1)}$ s.t.
  $||V - W^{(t)} H^{(t-1)}||^2_{Frobenius} \leq ||V - W^{(t-1)} H^{(t-1)}||^2_{Frobenius}$

  $H^{(t)^T} = update(V^T, W^{(t)^T}, H^{(t-1)^T}$ s.t.
  $||V - W^{(t)} H^{(t)}||^2_{Frobenius} \leq ||V - W^{(t)} H^{(t-1)}||^2_{Frobenius}$

  if *Stopping Criteria is met* [a] **then**

    break;

  end

end

Return W, H

[a] A stopping criteria can be an upper bound on iteration, or runtime, or if there is no significant change in W and H between the iterations

Figure 9: Plot of NMF components i.e. the matrix W in Eqn. 11 for MNIST dataset and $k = 5$. Each image corresponds to a column in W. Notice the plot of the components resemble digits more than the PCA components in Fig. 7.

4.2.4 t-distributed Stochastic Neighbor Embedding (t-SNE)

t-SNE(van der Maaten and Hinton, 2008) is a t-distributed variant of Stochastic Neighbor Embedding (SNE) (Hinton and Roweis, 2002). It is primarily used for dimensionality reduction and visualization of high-dimensional data in a lower-dimensional space, typically 2D or 3D. It learns a non-parametric mapping and is inherently stochastic due to its probabilistic nature so different runs of t-SNE on the same data may produce slightly different representations. Since a t-distribution is a log-tailed distribution, t-SNE prevents the crowding problem, which is one disadvantage of SNE.

The key idea behind t-SNE is to model pairwise similarities between data points in the high-dimensional space and then map these similarities to a lower-dimensional space. It does so by constructing probability distributions over pairs of points in both the original high-dimensional space and the lower-dimensional embedding space. It then minimizes the Kullback-Leibler divergence between these distributions, effectively optimizing the embedding to reflect the similarities in the original data as closely as possible.

One crucial parameter in t-SNE is **perplexity**, which controls the number of effective neighbors (observations) considered during the optimization process. Perplexity can be seen as a rough measure of the number of close neighbors for each data point. Choosing an appropriate perplexity value is important, as it can significantly affect the resulting embedding. Furthermore, t-SNE is sensitive to the scale of the input features. Therefore, it is essential to scale the data appropriately before applying t-SNE to ensure that features with larger magnitudes do not dominate the optimization process. Standardization is a common preprocessing step for t-SNE.

Unlike linear techniques such as PCA (Principal Component Analysis), t-SNE is nonlinear. It preserves the local (topological) structure of the data by modeling the pairwise similarities between data points in the high-dimensional space and mapping them to a lower-dimensional space. This makes it particularly useful for capturing complex relationships within the data. In addition, t-SNE focuses on preserving the local structure of the data, meaning that nearby points in the original high-dimensional space are likely to remain close in the lower-dimensional embedding. This property makes it well-suited for visualizing clusters and identifying patterns in the data.

However, the distances between points in the t-SNE plot do not directly correspond to distances in the original high-dimensional space. Therefore, qualitative rather than quantitative interpretations should be made based on the relative positions of points and clusters. On top of this, t-SNE can be computationally intensive, especially for large datasets. The algorithm has a time complexity of approximately $O(N^2)$ for the pairwise distance computation between points and $O(NlogN)$ for optimization, where N is the number of data points. For very large datasets, approximate versions of t-SNE or techniques like Barnes-Hut t-SNE may be used to reduce the computational effort.

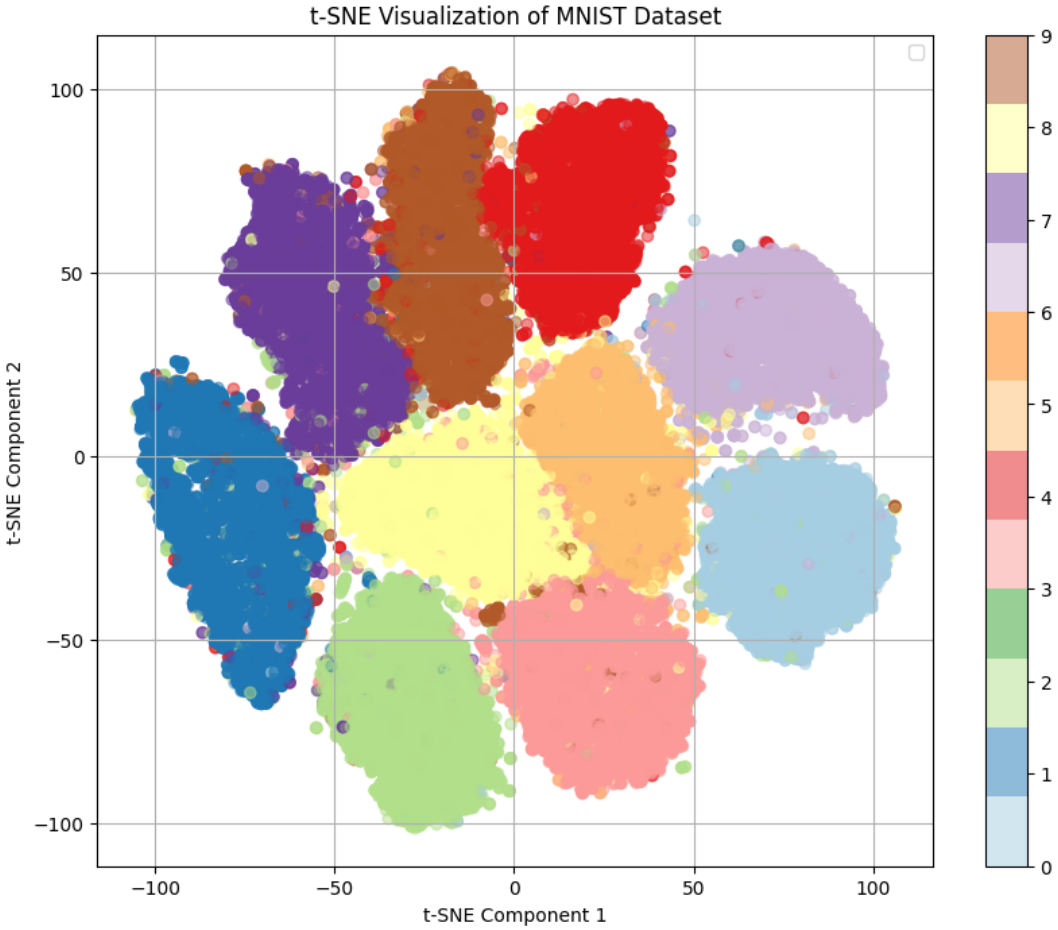

Figure 10: t-SNE visualization of the MNIST dataset. The numbers 0 to 9 in the legend correspond to the digits.

4.2.5 Autoencoders

Autoencoders are a type of neural network used for unsupervised learning. They are quite popular in the realm of dimensionality reduction and data compression. They've gained significant attention due to their ability to learn efficient representations of data.

An autoencoder consists of two main parts: an encoder and a decoder. The encoder compresses the input data into a latent-space representation, while the decoder reconstructs the original input from this representation. The goal is to ensure that the reconstruction closely matches the original input. Autoencoders can be undercomplete (where the dimensionality of the latent space is lower than the input) or overcomplete (where it's higher). Undercomplete autoencoders are primarily used for dimensionality reduction, while overcomplete ones can learn more complex representations.

The choice of loss function plays a crucial role in training autoencoders. Mean Squared Error (MSE) loss is commonly used for reconstruction tasks, but other loss functions like Binary Cross-Entropy or Kullback-Leibler Divergence are employed for specific purposes, such as in variational autoencoders (VAEs). VAEs are a type of generative model that extends the basic autoencoder framework. They incorporate probabilistic concepts, allowing for generating new data samples. VAEs learn a distribution in the latent space, enabling them to generate diverse outputs.

We are using a basic autoencoder, which is a neural network with an encoder and a decoder, with multiple hidden layers in between. There are no probabilistic concepts involved as in the VAEs. We fed the latent space to the next stage in our algorithm, which is the ensemble of simple submodels.

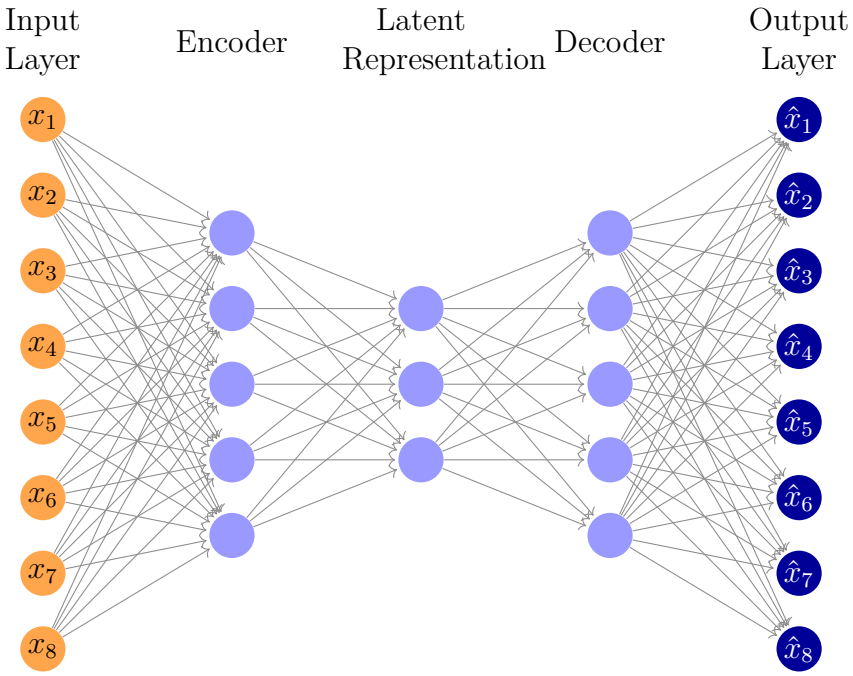

Figure 11: (Basic) Autoencoder with one hidden layer each for the encoder and the decoder. Dimensionality reduction does not require the decoder and the output layer.

4.2.6 Zero-phase component analysis (ZCA) pre-whitening and Restricted Boltzmann Machine (RBM)

The idea behind sequencing ZCA and RBM is that ZCA Whitening will be used to remove correlation between nearby pixels i.e. two-way correlation, which will allow us to focus on higher-order correlations, and then RBM will be used for feature extraction.

Zero-phase Component Analysis, often abbreviated as ZCA, is a powerful technique used for whitening data, particularly images. It aims to decorrelate the pixel values of images while preserving their original structure and important features. By doing so, ZCA reduces redundancy in the data and enhances the efficiency of subsequent learning algorithms. ZCA is different from ICA and PCA in that a ZCA-ed image still looks like the original image, while an ICA-ed image or a PCA-ed image no longer looks like the original image.

The process of ZCA involves several steps:

1. Normalization: Initially, the pixel values of the images are normalized to have zero mean and unit variance. This step ensures that the data is centered around zero and has a consistent scale, facilitating further processing.
2. Covariance Matrix Calculation: Next, the covariance matrix of the normalized data is computed. This matrix captures the relationships between different pixel values in the images.
3. Eigen-Decomposition: The covariance matrix is then decomposed into its constituent eigenvectors and eigenvalues. These eigenvectors represent the principal components of the data, while the eigenvalues indicate their importance or variance.
4. Whitening Transformation: Using the eigenvectors and eigenvalues, the data is transformed to a new space where the dimensions are decorrelated and scaled appropriately. This transformation effectively whitens the data, making the features more independent and uniformly distributed.
5. Reconstruction: Finally, the whitened data can be reconstructed back to the original space if necessary, preserving the essential information while removing redundancies.

Restricted Boltzmann Machines, or RBMs, belong to the family of generative neural networks and are widely used for unsupervised learning tasks such as feature learning,

dimensionality reduction, and collaborative filtering. RBMs consist of two layers of neurons: visible units, which represent the input data, and hidden units, which capture higher-level abstractions or features.

The training of RBMs involves two main steps:

1. Contrastive Divergence (CD): This algorithm is used to update the weights between the visible and hidden units iteratively. During training, the model is presented with input data, and the activations of the hidden units are computed using a probabilistic approach based on the current weights. These activations are then used to reconstruct the visible units, and the difference between the actual and reconstructed data is minimized through weight updates.
2. Gibbs Sampling: In the Gibbs sampling process, the states of the visible and hidden units are sampled alternately to generate new configurations of the network. This sampling process allows the RBM to explore the probability distribution of the input data and learn meaningful representations in an unsupervised manner.

4.3 Feature Bagging

Feature bagging is a technique used in machine learning to improve the performance and robustness of models. The idea is to train several models on a subset of the original features to capture different aspects of the dataset. The expectation is that the results of these models have high variance, and taking an average of those results will help minimize overfitting, especially in high-dimension feature spaces.

We used the PolynomialFeatures() function in *scikit* library with no bias-term and only interaction-terms for feature bagging i.e. if a dataset has two features a and b, the new set of features will include $\{a, b, ab\}$, but exclude $\{a^2, b^2\}$ and the intercept. The total amount of features generated from PolynomialFeatures() is further filtered to reduce wall-clock runtime. First, the feature set is reduced to a given percentage of the original feature size. If this still generates too many features, the resulting feature set is further reduced to a fixed maximum size.

4.4 Ensemble

In machine learning, bias and variance of a model are considered complementary to each other, that is, if the bias increases then the variance will decrease, and vice versa.

A good model ought to balance the so-called "bias-variance tradeoff". A model with high bias means the model did not learn the data well enough that the predictions are unrelated to the data, and a model with high variance means the model learned the data too well that the predictions vary with each observation. An individual model tends to have either high bias or high variance. Ensemble learning attempts to mitigate this problem by combining multiple models to reduce the bias or the variance, as applicable. Specifically, the thinking is that an ensemble will have lower variance if each sub-model has high variance and low bias, or have lower bias if each sub-model has low variance and high bias.

The individual sub-models can be combined to form an ensemble using one of the methods below[4]:

- Bagging: This method reduces the variance of the individual sub-models. The ensemble is a collection of homogeneous weak sub-models trained in parallel, independent of each other, and then the output of each sub-model is aggregated using majority voting or averaging. Random Forest is an example of the bagging method, combining several individual decision trees to make an ensemble model. The resulting ensemble has lower variance when compared to a single decision tree.
- Boosting: This method reduces the bias of the individual sub-models. The ensemble is a a collection of homogeneous weak sub-models trained in sequence such that a sub-model output is fed to the next sub-model and this subsequent sub-model improves its performance using the errors from the previous sub-model. The sub-models are aggregated at each step, not at the end. This means the ensemble output is the output of the last sub-model. The weighted average method aggregates the output from each stage (sub-model).
- Stacking: This method increases the predictive accuracy of the sub-models. The ensemble is a collection of heterogeneous strong sub-models trained in parallel, independent of each other. These sub-models utilize the initial training dataset to predict a new training dataset, and a meta-model is trained using the new training dataset. The predictions from the meta-model are aggregated using the weighted average method.

We experimented with four submodels: Linear Regression, LASSO, One-Class SVM, and ElasticNet. The submodels are briefly explained in the following subsections.

[4]The ensembles in this thesis were combined using the bagging method, as mentioned in Section 3.1

4.4.1 Linear Regression

Linear Regression is a statistical method to model the relationship between a dependent feature and one or more independent features. The relationship is assumed to be linear, meaning that changes in the independent variables are associated with proportional changes in the dependent variable.

Suppose we have N observations and p features, the design matrix X of the independent features is given as:

$$X = \begin{bmatrix} 1 & x_{11} & \dots & x_{1p} \\ 1 & x_{21} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \dots & x_{np} \end{bmatrix}$$

Then, a simple Linear Regression model can be expressed as:

$$Y = X\beta + \epsilon \tag{12}$$

where ϵ is the error term.

Equation 12 has a closed form solution for the coefficients estimate $\hat{\beta}$, given below:

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

Then, the dependent variable y can be estimated from the coefficient estimates:

$$\hat{y} = X\hat{\beta}$$

The (classical) Linear Regression has the following underlying assumptions:

1. Linearity: The relationship between the dependent and independent variables is assumed to be linear. If this assumption is violated, the model may produce biased or unreliable estimates.
2. Independence of Errors: The errors (residuals) should be independent of each other and have constant variance (homoscedasticity). Violations of this assumption can lead to inefficient or biased estimates.

3. Normality of Errors: The errors are assumed to be normally distributed. While the central limit theorem suggests that this assumption may not be critical for large sample sizes, departures from normality can still affect the accuracy of confidence intervals and hypothesis tests.
4. No Perfect Multicollinearity: The independent variables should not be highly correlated with each other. Multicollinearity can lead to unstable estimates and difficulty in interpreting the coefficients.

4.4.2 LASSO

LASSO (Least Absolute Shrinkage and Selection Operator) regression is a powerful tool for variable selection and regularization. It is also called L1 regularization. At its core, LASSO regression is a Linear Regression technique that incorporates a penalty term, enforcing sparsity in the coefficient estimates. This penalty term, often denoted as λ, constrains the sum of the absolute values of the coefficients, shrinking some coefficients towards zero. This action of shrinking coefficients (to zero) which essentially performs variable selection is what distinguishes LASSO from traditional Linear Regression methods. This (feature) sparsity makes the resulting model easier to interpret.

Suppose we have N observations with p features and a single outcome y, then the LASSO objective is to solve the following equation:

$$\min_{\beta_0,\beta} \left\{ \sum_{i=1}^{N} \left(y_i - \beta_0 - x_i^T \beta \right)^2 \right\} \text{ subject to } \sum_{j=1}^{p} |\beta_j| \leq t.$$

where t is a user-defined constant that determines the degree of regularization.

The regularization parameter λ controls the trade-off between model complexity and goodness of fit. Techniques such as cross-validation can be employed to automatically tune this parameter, ensuring optimal model performance.

Although there is only one hyperparameter (λ) to tune, choosing the appropriate value can be challenging, and improper tuning may lead to suboptimal models. LASSO also gets computationally prohibitive for large datasets, necessitating specialized algorithms or distributed computing frameworks in those cases.

4.4.3 One-Class SVM

Support Vector Machine (SVM) is a powerful tool in machine learning, primarily used for classification tasks. However, traditional SVMs require labeled data with instances from multiple classes for training. In many real-world scenarios, however, obtaining labeled data from all classes can be challenging or even impossible. This is where One-Class SVMs come into play. One-Class SVM is a variant of the traditional SVM designed to tackle the problem of anomaly detection, where only one class (normal instances) is present during training.

A One-Class SVM is a machine learning algorithm that learns a decision boundary around the normal data points in a dataset. It classifies new instances as either normal or anomalies based on their proximity to this decision boundary. Unlike traditional SVMs, which aim to find the optimal hyperplane that maximizes the margin between different classes, One-Class SVM aims to encapsulate the normal instances within a region of feature space.

One-Class SVM works by mapping the input data into a high-dimensional feature space using a kernel function, typically a radial basis function (RBF) kernel. Then, it tries to find the hyperplane (decision boundary) that best separates the normal instances from the origin in this feature space. This hyperplane is determined by minimizing the margin violations, i.e., instances that fall on the wrong side of the decision boundary. The hyperplane is positioned to encompass the majority of normal instances while minimizing the outliers.

One-Class SVM is particularly well-suited for anomaly detection tasks where only normal data is available during training. It can identify outliers or anomalies in unseen data based on their deviation from the learned normal behavior. It can also handle high-dimensional data efficiently.

One-Class SVM assumes that the majority of instances in the dataset are normal, so the performance of the model may be affected if the dataset is highly imbalanced. Moreover, like many machine learning algorithms, One-Class SVM requires careful selection of hyperparameters such as the kernel function and its parameters. Improper parameter tuning can lead to suboptimal performance or overfitting. Finally, the decision boundary learned by One-Class SVM may not always be interpretable, especially in high-dimensional feature spaces, which can make it challenging to understand the model's behavior.

4.4.4 ElasticNet

Although LASSO simultaneously generates sparse feature space and performs feature selection, it has two noticeable shortcomings: the number of selected features is bounded by the number of observations, and it also tends to select only one (or a few) features from a subset of correlated features and shrinks the rest to zero (Zou and Hastie, 2005). This is where ElasticNet comes into the picture.

ElasticNet blends the penalties of Lasso (L1) and Ridge (L2) regression to offer a solution to the challenges of feature selection, multicollinearity, and model flexibility. The objective function of ElasticNet regression can be expressed as follows:

$$\min_{\beta} \left\{ ||y - X\beta||_2^2 + \lambda_1 ||\beta||_1 + \lambda_2 ||\beta||_2^2 \right\}$$

where, y is a vector of the outcomes, X is the design matrix, β is a vector of coefficients, and λ_1 and λ_2 are the regularization parameters controlling the strength of the L1 and L2 penalties, respectively.

Through the tuning of λ_1 and λ_2, ElasticNet provides users with the flexibility to adjust the balance between L1 and L2 regularization, and address the two shortcomings of LASSO mentioned above.

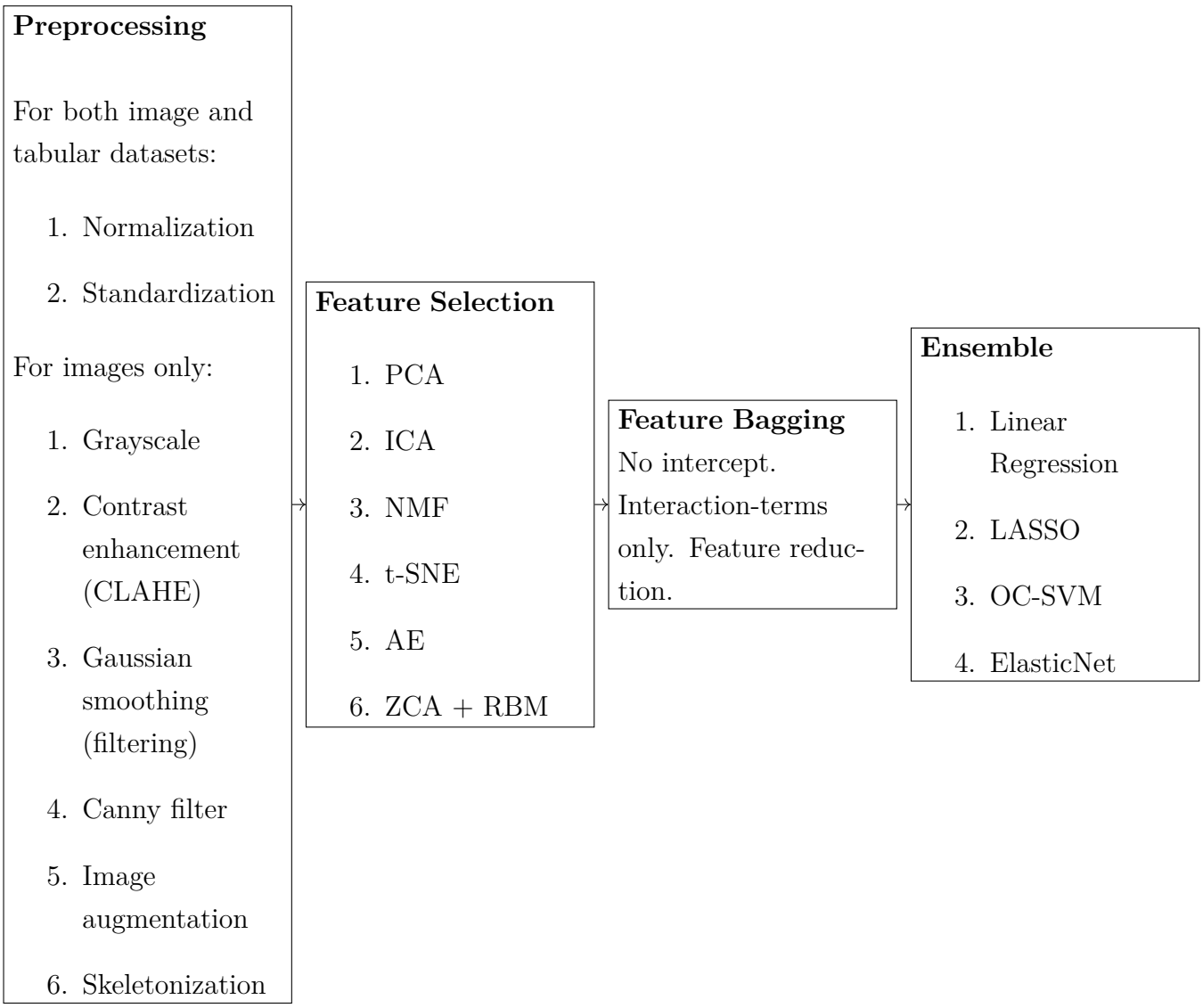

Figure 12: Summarization of the algorithm.

5 Experiments

First, an ensemble size of 3000 is selected so that the results are comparable to the DEAN experiment results. Second, a 10-minute computation budget (per configuration[5]) is set to prevent certain configurations from running too long. Third, each configuration is

[5]For instance, one configuration can be a normalized dataset with PCA feature selection followed with a feature bagging step and with an ensemble of Linear Regression submodels; another configuration may be the same setup except for standardized dataset instead of normalized.

run 10 times and the average AUROC of those 10 runs is taken as the result of that configuration.

There are 72 configurations for tabular datasets and 4608 configurations for image datasets. We ran all 72 configurations for the tabular datasets, but only 504 configurations for the image datasets because it will take too long to run the entire 4608 configurations[6]. The 504 configurations are the sum of the 72 'core' configurations and the 432 'main effects' configurations like 'norm-grayscale' preprocessing. Table 3 lists some of the configurations for clarity.

Pre-Processing	Feature Extraction	Submodel Type	Number of Submodels
Normalize	Autoencoder	OC-SVM	3000
Normalize	ZCA + RBM	Linear Regression	3000
Normalize	ZCA + RBM	ELasticNet	3000
Normalize	ZCA + RBM	LASSO	3000
Normalize	ICA	OC-SVM	3000
Standardize	ZCA + RBM	OC-SVM	3000
Standardize	ZCA + RBM	LASSO	3000
Standardize	ICA	OC-SVM	3000
Standardize	ICA	Linear Regression	3000
Standardize	ICA	ELasticNet	3000
Standardize	ICA	LASSO	3000
None	Autoencoder	OC-SVM	3000
None	Autoencoder	Linear Regression	3000
None	PCA	ELasticNet	3000
None	PCA	LASSO	3000
None	None	OC-SVM	3000
None	None	Linear Regression	3000
...	...	...	...

Table 3: Some of the configurations used to run the jobs. One row represents one configuration. For instance, the first configuration on the list is where the data is normalized first, followed by feature extraction using Autoencoder, and finally an ensemble of 3000 OC-SVM.

[6]Imagine 10 runs for each of the 10 classes in the image datasets, even with each run having a 10-minute computation budget, the total runtimes can grow significantly.

5.1 Satellite

The experiment results (AUROC) for the Satellite dataset are shown in Table 4 below. Each AUROC shown is the average of 10 runs. For instance, the AUROC of 0.8035 on the top-left corner is the average of 10 runs of the same configuration: no preprocessing, features selection using Autoencoder, and ElasticNet submodels.

The best-performing configuration (AUROC = 0.9684) is to normalize the data, perform no feature selection, and then use LASSO submodels. For normalized data and no feature selection, ElasticNet and Linear Regression submodels also showed competitive AUROCs. All in all, regardless of the preprocessing and the feature selection options, LASSO and Linear Regression submodels performed fairly well, while OC-SVM performed worst on average.

AVERAGE of AUROC		Ensemble			
Preprocess	Selection	elastic	lasso	lin	oc-svm
none	ae	0.8035	0.7979	0.8582	0.5000
	ica	0.9140	0.8809	0.9243	0.4318
	none	0.7438	0.7444	0.8953	0.5000
	pca	0.9073	0.8912	0.8990	0.7492
	zca + rbm	0.4556	0.4544	0.4623	0.5000
	tsne	0.6174	0.5599	0.5983	0.7591
norm	ae	0.9075	0.9377	0.9175	0.5000
	ica	0.9121	0.9397	0.9283	0.4817
	none	0.9683	0.9684	0.9672	0.5793
	pca	0.5000	0.5000	0.9200	0.8432
	zca + rbm	0.3778	0.3753	0.3646	0.5000
	tsne	0.6325	0.6053	0.6143	0.7890
std	ae	0.8969	0.9007	0.9056	0.5000
	ica	0.9250	0.9056	0.9383	0.4393
	none	0.9070	0.9070	0.9125	0.6602
	pca	0.6349	0.5000	0.9109	0.8069
	zca + rbm	0.3359	0.3268	0.3451	0.5000
	tsne	0.5894	0.5879	0.5966	0.6994

Table 4: AUROCs for Satellite dataset. Each value is the average of 10 runs. The highlighted cells are the top 10% highest AUROCs. The best-performing configuration of 0.9684 AUROC is normalized data and a LASSO submodel but with no feature selection.

Regardless of the preprocessing and the feature selection options, Linear Regression and OC-SVM submodels ran quicker than ElasticNet and LASSO submodels. In fact, the fastest configurations are the ones with Linear Regression submodel. The wallclock runtimes in seconds are shown in Table 5.

AVERAGE of Runtime (sec)		Ensemble			
Preprocess	Selection	elastic	lasso	lin	oc-svm
none	ae	255.8762	251.5797	19.6155	20.7705
	ica	443.3754	450.7738	4.4386	8.9327
	none	600.1985	600.1952	29.933	16.7359
	pca	600.1914	600.1093	4.4311	70.7904
	zca + rbm	266.3904	265.2003	5.1736	7.9466
	tsne	546.7106	545.6989	325.0739	350.4549
norm	ae	267.9164	278.3019	22.4142	22.9126
	ica	468.903	457.6327	4.4746	8.8707
	none	600.3472	600.2436	29.9491	16.6389
	pca	234.2943	233.7825	4.4003	8.0973
	zca + rbm	268.5004	267.5895	5.2173	7.8262
	tsne	556.9002	552.8689	332.4173	354.7121
std	ae	263.83	263.1137	21.7297	24.4521
	ica	457.7685	453.0444	4.4422	8.8504
	none	600.2496	600.3135	30.1164	16.9414
	pca	282.2199	236.9244	4.4228	10.3045
	zca + rbm	267.1621	267.1687	5.1355	7.8112
	tsne	548.6326	549.1985	326.1366	352.3072

Table 5: Runtimes for the Satellite dataset. Each value is the average of 10 runs. ElasticNet and LASSO run significantly longer compared to Linear Regression and OC-SVM. The highlighted cells are the best 10% runtimes.

The ensemble size of 3000 ran to completion within the 10-minute budget for most configurations, except for some configurations related to ElasticNet and LASSO submodels. Table 6 showed the counts of submodels that ran within the 10-minute computation budget. This table is complementary to the runtimes in Table 5.

AVERAGE of Count		Ensemble			
Preprocess	Selection	elastic	lasso	lin	oc-svm
none	ae	3000	3000	3000	3000
	ica	3000	3000	3000	3000
	none	1619	1618.3	3000	3000
	pca	2309.4	2299.3	3000	3000
	zca + rbm	3000	3000	3000	3000
	tsne	3000	3000	3000	3000
norm	ae	3000	3000	3000	3000
	ica	3000	3000	3000	3000
	none	1636.6	1745.2	3000	3000
	pca	3000	3000	3000	3000
	zca + rbm	3000	3000	3000	3000
	tsne	3000	3000	3000	3000
std	ae	3000	3000	3000	3000
	ica	3000	3000	3000	3000
	none	1253.7	1258.8	3000	3000
	pca	3000	3000	3000	3000
	zca + rbm	3000	3000	3000	3000
	tsne	3000	3000	3000	3000

Table 6: Count of the submodels that were run within the 10-minute computation budget for the Satellite dataset. Each value is the average of 10 runs. The highlighted cells are the smallest 10%.

To validate the relationship between AUROC and ensemble size, several ensemble sizes at a step of 100 were run for a particular configuration (normalized data, no feature selection, and Linear Regression submodels) as shown in Figure 13. We can see that the relationship between ensemble size and the AUROCs is neither linear nor logarithmic, but rather almost like a sawtooth wave.

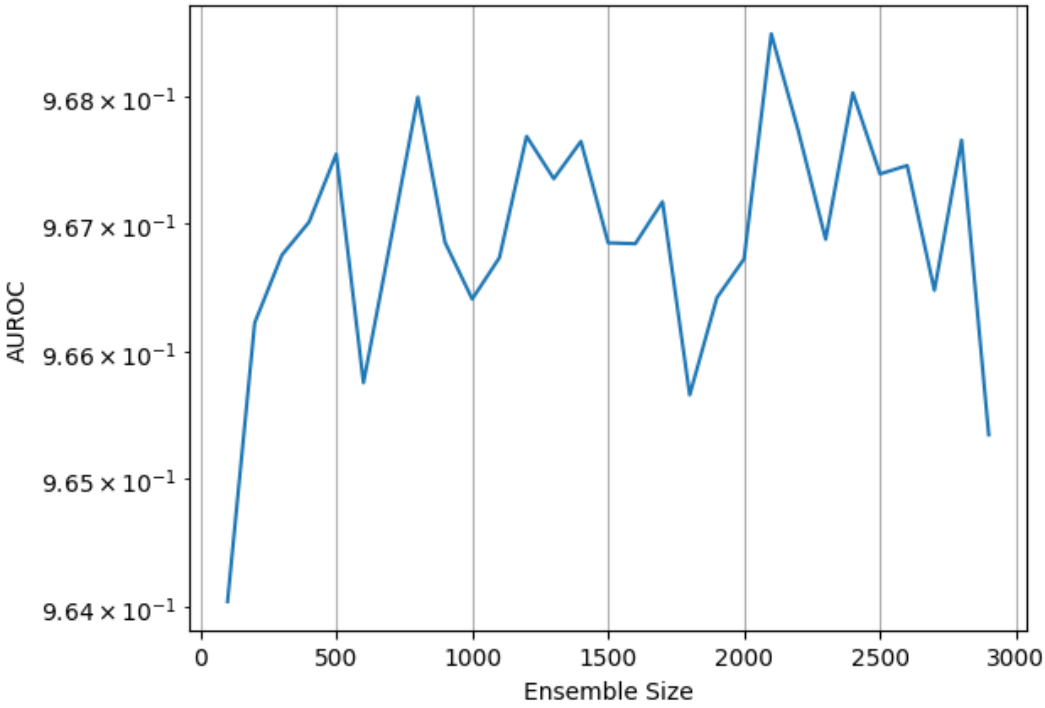

Figure 13: AUROC vs. ensemble size of the Satellite dataset with normalized data, no feature selection, and an ensemble of Linear Regression submodels.

5.2 Credit Card Fraud

The experiment results (AUROC) for the Credit Card Fraud dataset are shown in Table 7 below. The best-performing configuration (AUROC = 0.9567) is with normalized data, no feature selection, and Linear Regression submodels.

Regarding feature selection, RBM and t-SNE perform lowest regardless of the preprocessing and the submodel types; and not doing feature selection at all produces the highest AUROCs across the configurations.

AVERAGE of AUROC		Ensemble			
Preprocess	Selection	elastic	lasso	lin	oc-svm
none	ae	0.5055	0.5056	0.5511	0.5000
	ica	0.7549	0.7467	0.7731	0.7151
	none	0.8812	0.8665	0.9420	0.7544
	pca	0.7246	0.7149	0.7650	0.5351
	zca + rbm	0.4713	0.4699	0.4701	0.5000
	tsne	0.5000	0.5000	0.5000	0.5000
norm	ae	0.6615	0.6928	0.7166	0.5000
	ica	0.7696	0.7628	0.7812	0.5688
	none	0.9512	0.9480	0.9567	0.5000
	pca	0.5000	0.5000	0.7613	0.5030
	zca + rbm	0.4687	0.4668	0.4677	0.5000
	tsne	0.5000	0.5000	0.5000	0.5000
std	ae	0.8844	0.8686	0.8991	0.5000
	ica	0.8592	0.8629	0.8639	0.6187
	none	0.9365	0.9376	0.9399	0.9146
	pca	0.6981	0.6092	0.8382	0.6099
	zca + rbm	0.4718	0.4700	0.4701	0.5000
	tsne	0.5000	0.5000	0.5000	0.5000

Table 7: AUROCs for Credit Card Fraud dataset. Each value is the average of 10 runs. The best-performing configuration of 0.9567 AUROC is normalized data and Linear Regression submodels but with no feature selection. The highlighted cells are the top 10% AUROCs.

The wallclock runtimes in seconds are shown in Table 8. t-SNE feature selection increased the runtimes by orders of magnitude irrespective of the other options. As a whole, the Linear Regression submodel with ICA feature selection runs the fastest.

AVERAGE of Runtime (sec)		Ensemble			
Preprocess	Selection	elastic	lasso	lin	oc-svm
none	ae	600.8602	601.1312	98.9414	311.7588
	ica	603.604	601.4035	98.5179	381.5936
	none	603.6166	603.7823	601.4229	605.9302
	pca	601.5444	601.205	98.8311	614.8519
	zca + rbm	601.3747	601.3187	541.4898	428.3078
	tsne	30203.9222	30031.5495	29996.6158	41059.2288
norm	ae	600.9412	600.7636	262.9384	441.2298
	ica	602.356	601.6201	99.6406	602.3667
	none	608.295	608.5644	601.0178	602.0415
	pca	601.2456	602.0464	99.1412	378.9214
	zca + rbm	601.1502	601.124	146.4145	424.4468
	tsne	46178.7255	45849.5332	46664.729	46142.0382
std	ae	600.9211	600.8541	336.4428	535.597
	ica	601.5637	601.4412	98.7027	603.962
	none	603.8183	605.2633	601.1035	600.9925
	pca	601.9737	601.0422	244.9774	397.4282
	zca + rbm	601.0975	601.1652	274.7261	602.3055
	tsne	35875.9548	37322.5751	36279.8834	36343.0288

Table 8: Runtimes for the Credit Card Fraud dataset. Each value is the average of 10 runs. Regardless of the preprocessing and the submodel type, t-SNE feature selection increased the runtimes by an order of magnitude. The highlighted cells are the fastest 10%.

AVERAGE of Count		Ensemble			
Preprocess	Selection	elastic	lasso	lin	oc-svm
none	ae	552.6	515.3	3000	3000
	ica	169.9	441.6	3000	3000
	none	185.1	111.6	588.1	45.5
	pca	299.4	309.4	3000	33.3
	zca + rbm	430.4	427.6	3000	3000
	tsne	0	0	0	0
norm	ae	474.2	403.8	3000	3000
	ica	263.9	430.1	3000	593.1
	none	38.7	42	590.6	302.4
	pca	518.3	215.6	3000	3000
	zca + rbm	430.9	428.5	3000	2971.4
	tsne	0	0	0	0
std	ae	355.9	250.9	2926.7	2787.4
	ica	390	432.1	3000	1337.6
	none	97.9	90.2	565.9	747.7
	pca	286	501.2	3000	3000
	zca + rbm	427.2	431.2	3000	519.7
	tsne	0	0	0	0

Table 9: Count of the submodels that were run within the 10-minute computation budget for the Credit Card Fraud dataset. Each value is the average of 10 runs. The highlighted cells i.e., configurations with t-SNE feature selection did not run even one submodel within the 10-minute budget.

5.3 MNIST

Table 10 lists the AUROCs for the 72 'core' configurations, and the top 10% are highlighted. The best-performing class mostly occurred for the digit '1', which is not surprising considering the simplicity of the digit. Other high AUROCs occurred for the digits '5', '6', '7', and '9' for Linear Regression and OC-SVM submodels. The configurations with the most number of high AUROCs is standardized data and OC-SVM submodel but with no feature selection.

AVERAGE of AUROC			Ensemble			
Preprocess	Selection	Digit	elastic	lasso	lin	oc-svm
none	ae	0	0.8173	0.8147	0.8907	0.5000
		1	0.9545	0.9518	0.9825	0.5000
		2	0.6781	0.6757	0.7584	0.5000
		3	0.6831	0.6703	0.7605	0.5000
		4	0.6738	0.6696	0.7834	0.5000
		5	0.6368	0.6328	0.7287	0.5000
		6	0.7589	0.7491	0.8517	0.5000
		7	0.7179	0.7033	0.8139	0.5000
		8	0.7459	0.7483	0.7882	0.5000
		9	0.7275	0.7403	0.8310	0.5000
	ae Average		0.7394	0.7356	0.8189	0.5000
	ica	0	0.7987	0.8114	0.8298	0.5523
		1	0.9629	0.9667	0.9814	0.6399
		2	0.6524	0.6716	0.6666	0.5174
		3	0.6265	0.6323	0.6587	0.5688
		4	0.8042	0.8218	0.8063	0.5349
		5	0.6538	0.6525	0.6621	0.5292
		6	0.7895	0.7931	0.8268	0.5552
		7	0.7331	0.7086	0.7547	0.5939
		8	0.6139	0.6014	0.6127	0.5450
		9	0.6816	0.6752	0.7336	0.5957
	ica Average		0.7317	0.7335	0.7533	0.5632
	none	0	0.8508	0.8513	0.9050	0.5000
		1	0.9362	0.9369	0.9856	0.5000
		2	0.6921	0.6920	0.7915	0.5000
		3	0.7001	0.7013	0.7631	0.5000
		4	0.7737	0.7721	0.8513	0.5000
		5	0.6643	0.6656	0.7185	0.5000
		6	0.7587	0.7582	0.9170	0.5000
		7	0.7862	0.7850	0.8957	0.5000
		8	0.7371	0.7392	0.7897	0.5000
		9	0.7601	0.7607	0.8545	0.5000
	none Average		0.7659	0.7662	0.8472	0.5000

Preprocess	Selection	Digit	elastic	lasso	lin	oc-svm
	pca	0	0.7665	0.7683	0.9581	0.8443
		1	0.9871	0.9866	0.9978	0.8251
		2	0.5373	0.5927	0.8384	0.6354
		3	0.7160	0.7045	0.8824	0.7397
		4	0.7061	0.7040	0.9260	0.7406
		5	0.6574	0.6477	0.8566	0.6925
		6	0.6462	0.6844	0.9402	0.6949
		7	0.8118	0.8199	0.9365	0.7511
		8	0.6433	0.6460	0.8037	0.6679
		9	0.7437	0.7664	0.9251	0.7774
	pca Average		0.7215	0.7321	0.9065	0.7369
	zca + rbm	0	0.5781	0.5781	0.5781	0.5000
		1	0.6230	0.6227	0.6235	0.5000
		2	0.5547	0.5547	0.5547	0.5000
		3	0.5443	0.5444	0.5442	0.5000
		4	0.5280	0.5283	0.5282	0.5000
		5	0.5359	0.5359	0.5362	0.5000
		6	0.6796	0.6800	0.6801	0.5000
		7	0.6667	0.6665	0.6666	0.5000
		8	0.5411	0.5418	0.5417	0.5000
		9	0.6104	0.6108	0.6115	0.5000
	zca + rbm Average		0.5862	0.5863	0.5865	0.5000
	tsne	0	0.6248	0.6246	0.6149	0.5131
		1	0.5000	0.5000	0.5000	0.5000
		2	0.4360	0.4429	0.4616	0.4043
		3	0.5562	0.5553	0.5933	0.4857
		4	0.5212	0.5263	0.5160	0.5165
		5	0.4695	0.4625	0.4615	0.4208
		6	0.5196	0.5145	0.5246	0.4341
		7	0.4501	0.4395	0.4436	0.3671
		8	0.5477	0.5776	0.5619	0.5276
		9	0.4776	0.4616	0.4632	0.4479
	tsne Average		0.5103	0.5105	0.5141	0.4617
norm	ae	0	0.9574	0.9614	0.9791	0.5000

Preprocess	Selection	Digit	elastic	lasso	lin	oc-svm
		1	0.9919	0.9921	0.9961	0.5000
		2	0.8083	0.8110	0.8663	0.5000
		3	0.8381	0.8420	0.8964	0.5000
		4	0.8665	0.8673	0.9144	0.5000
		5	0.7965	0.7992	0.8539	0.5000
		6	0.9333	0.9327	0.9640	0.5000
		7	0.9091	0.9058	0.9367	0.5000
		8	0.8635	0.8656	0.8985	0.5000
		9	0.9043	0.9084	0.9398	0.5000
	ae Average		0.8869	0.8886	0.9245	0.5000
	ica[7]	0	0.8177	0.8144	0.8324	0.4818
		1	0.9809	0.9656	0.9951	0.9117
		2	0.6765	0.6843	0.6915	0.5259
		3	0.7546	0.7402	0.7573	0.5419
		4	0.8006	0.8133	0.8160	0.5153
	ica Average		0.8061	0.8035	0.8184	0.5953
	none	0	0.8506	0.8522	0.9048	0.5000
		1	0.9339	0.9391	0.9849	0.5000
		2	0.6897	0.6948	0.7926	0.5000
		3	0.7019	0.7028	0.7620	0.5000
		4	0.7694	0.7738	0.8514	0.5000
		5	0.6641	0.6675	0.7167	0.5000
		6	0.7580	0.7625	0.9182	0.5000
		7	0.8037	0.8089	0.8959	0.5000
		8	0.7400	0.7431	0.7892	0.5000
		9	0.7619	0.7665	0.8569	0.5000
	none Average		0.7673	0.7711	0.8473	0.5000
	pca	0	0.7865	0.8088	0.9734	0.8984
		1	0.9801	0.9626	0.9975	0.8335
		2	0.6136	0.5436	0.8421	0.6352
		3	0.6618	0.6777	0.8941	0.7511
		4	0.7315	0.7119	0.9258	0.7143
		5	0.6064	0.5720	0.8221	0.6870

[7]FastICA failed to converge for digits 5 and up.

Preprocess	Selection	Digit	elastic	lasso	lin	oc-svm
		6	0.6932	0.7170	0.9506	0.6800
		7	0.7680	0.7669	0.9306	0.7864
		8	0.6635	0.6773	0.8779	0.6604
		9	0.7389	0.7984	0.9295	0.7769
	pca Average		0.7243	0.7236	0.9144	0.7423
	zca + rbm	0	0.5719	0.5719	0.5719	0.5000
		1	0.5887	0.5887	0.5891	0.5000
		2	0.5547	0.5547	0.5547	0.5000
		3	0.5326	0.5326	0.5326	0.5000
		4	0.5239	0.5239	0.5239	0.5000
		5	0.5357	0.5357	0.5363	0.5000
		6	0.6749	0.6749	0.6751	0.5000
		7	0.6441	0.6446	0.6447	0.5000
		8	0.5400	0.5400	0.5400	0.5000
		9	0.6016	0.6017	0.6015	0.5000
	zca + rbm Average		0.5768	0.5769	0.5770	0.5000
	tsne	0	0.6124	0.6058	0.6059	0.5269
		1	0.5000	0.5000	0.5000	0.5000
		2	0.4620	0.4537	0.4706	0.4253
		3	0.5672	0.5867	0.6099	0.5246
		4	0.5135	0.5083	0.5281	0.5113
		5	0.4705	0.4711	0.4474	0.4397
		6	0.5230	0.5175	0.5208	0.4358
		7	0.4388	0.4254	0.4308	0.3608
		8	0.5672	0.5733	0.5520	0.5390
		9	0.4847	0.4435	0.4504	0.4489
	tsne Average		0.5139	0.5085	0.5116	0.4712
std	ae	0	0.9512	0.9539	0.9571	0.5000
		1	0.9894	0.9892	0.9895	0.5000
		2	0.8397	0.8282	0.8455	0.5000
		3	0.8215	0.8221	0.8265	0.5000
		4	0.8594	0.8612	0.8619	0.5000
		5	0.7937	0.7878	0.7961	0.5000
		6	0.9476	0.9480	0.9516	0.5000

Preprocess	Selection	Digit	elastic	lasso	lin	oc-svm
		7	0.9274	0.9259	0.9316	0.5000
		8	0.7812	0.7771	0.7724	0.5000
		9	0.9160	0.9119	0.9196	0.5000
	ae Average		0.8827	0.8805	0.8852	0.5000
	ica	0	0.7941	0.7908	0.8572	0.7615
		1	0.7810	0.9044	0.9891	0.9550
		2	0.6061	0.6559	0.7038	0.6354
		3	0.6886	0.6909	0.7883	0.7055
		4	0.7181	0.6897	0.8073	0.6381
		5	0.6167	0.6193	0.6553	0.5906
		6	0.7668	0.7954	0.8785	0.7688
		7	0.7868	0.8248	0.8775	0.7143
		8	0.6940	0.6815	0.7092	0.6028
		9	0.7976	0.7760	0.8240	0.8043
	ica Average		0.7250	0.7429	0.8090	0.7176
	none	0	0.9486	0.9493	0.9490	0.9684
		1	0.9921	0.9919	0.9907	0.9934
		2	0.7877	0.7890	0.8091	0.8769
		3	0.8115	0.8149	0.8188	0.9263
		4	0.8913	0.8923	0.8961	0.8522
		5	0.8003	0.8005	0.8033	0.9552
		6	0.8908	0.8927	0.9231	0.9501
		7	0.9143	0.9153	0.9267	0.9307
		8	0.7466	0.7466	0.7592	0.7942
		9	0.8809	0.8819	0.8910	0.9346
	none Average		0.8664	0.8674	0.8767	0.9182
	pca	0	0.9418	0.9462	0.9507	0.6382
		1	0.9900	0.9899	0.9912	0.9771
		2	0.7462	0.7421	0.7615	0.7281
		3	0.7553	0.7506	0.7661	0.7292
		4	0.8211	0.8215	0.8472	0.6167
		5	0.6887	0.6890	0.6918	0.6350
		6	0.9274	0.9218	0.9346	0.8805
		7	0.9337	0.9287	0.9377	0.8498

Preprocess	Selection	Digit	elastic	lasso	lin	oc-svm
		8	0.7330	0.7391	0.7501	0.7254
		9	0.8730	0.8760	0.8899	0.8312
	pca Average		0.8410	0.8405	0.8521	0.7611
	zca + rbm	0	0.5700	0.5707	0.5706	0.5000
		1	0.5943	0.5940	0.5970	0.5000
		2	0.5547	0.5547	0.5547	0.5000
		3	0.5275	0.5278	0.5279	0.5000
		4	0.5219	0.5222	0.5221	0.5000
		5	0.5362	0.5365	0.5362	0.5000
		6	0.6723	0.6725	0.6727	0.5000
		7	0.6641	0.6641	0.6643	0.5000
		8	0.5390	0.5389	0.5391	0.5000
		9	0.6016	0.6015	0.6024	0.5000
	zca + rbm Average		0.5782	0.5783	0.5787	0.5000
	tsne	0	0.5238	0.5222	0.5344	0.5024
		1	0.3893	0.3628	0.3700	0.4858
		2	0.5010	0.5293	0.4650	0.4764
		3	0.5113	0.5344	0.5035	0.5146
		4	0.4887	0.5022	0.4854	0.5404
		5	0.5453	0.5488	0.5203	0.5272
		6	0.5451	0.5435	0.5351	0.4092
		7	0.5504	0.5430	0.5362	0.4602
		8	0.6118	0.6162	0.6062	0.5536
		9	0.5708	0.5508	0.5697	0.4358
	tsne Average		0.5237	0.5253	0.5126	0.4905

Table 10: AUROCs for the MNIST dataset. Each value is the average of 10 runs.

Table 11 lists the average AUROCs for the 72 'core' configurations and the 432 'main effects' configurations. Instead of listing the AUROCs by the class (or by the digit), each number in the table is the average of the 10 runs of the 10 classes for a given configuration. The highlighted cells are the top 10% and we can see that the majority of them fall in the configurations with Linear Regression and OC-SVM submodels. In fact, the largest AUROC (0.9461) is from a "std,augment-none-SVM" configuration.

Among the preprocessing options of standardization, normalization, and no pre-preprocessing, standardization seems to better the AUROCs more than the other two. For the image-specific preprocessing options, augmentation, blurring, and CLAHE seem to improve the AUROCs. When it comes to feature selection, Autoencoder, no feature selection, and PCA contributed the most to increasing the AUROCs.

AVERAGE of AUROC		Ensemble			
Preprocess	Selection	elastic	lasso	lin	oc-svm
none	ae	0.7394	0.7356	0.8189	0.5000
	ica	0.7317	0.7335	0.7533	0.5632
	none	0.7659	0.7662	0.8472	0.5000
	pca	0.7215	0.7321	0.9065	0.7369
	zca + rbm	0.5862	0.5863	0.5865	0.5000
	tsne	0.5103	0.5105	0.5141	0.4617
none,augment	ae	0.6499	0.6486	0.6948	0.5000
	ica	0.6899	0.6899	0.8012	0.7965
	none	0.7248	0.7246	0.7325	0.5000
	pca	0.6329	0.6269	0.8494	0.8139
	zca + rbm	0.5094	0.5060	0.4985	0.5000
	tsne	0.5000	0.5000	0.5000	0.5000
none,blur	ae	0.6823	0.6833	0.7754	0.5000
	ica	0.7545	0.7550	0.8487	0.3350
	none	0.8349	0.8347	0.8656	0.5000
	pca	0.7080	0.7118	0.8756	0.1511
	zca + rbm	0.5482	0.5484	0.5391	0.5000
	tsne	0.5572	0.5540	0.5540	0.5278
none,canny	ae	0.6493	0.6320	0.5268	0.5000
	ica	0.5066	0.4907	0.4623	0.8464
	none	0.6083	0.6079	0.7132	0.5000
	pca	0.3020	0.2997	0.4261	0.8873
	zca + rbm	0.4890	0.4891	0.4936	0.5000
	tsne	0.5000	0.5000	0.5000	0.5000
none,clahe	ae	0.7368	0.7416	0.7775	0.5000
	ica	0.7888	0.7767	0.8998	0.7671
	none	0.7669	0.7662	0.8124	0.5000
	pca	0.6976	0.7061	0.8919	0.7769

Preprocess	Selection	elastic	lasso	lin	oc-svm
	zca + rbm	0.4939	0.4968	0.4913	0.5000
	tsne	0.5191	0.5253	0.5289	0.5149
none,gray	ae	0.5011	0.5004	0.5003	0.5000
	ica	0.4996	0.4995	0.4996	0.4996
	none	0.4999	0.4999	0.5001	0.5000
	pca	0.4989	0.4987	0.4997	0.4996
	zca + rbm	0.5009	0.5023	0.5016	0.5000
	tsne	0.5082	0.5037	0.4992	0.4924
none,skel	ae	0.5399	0.5331	0.4836	0.5000
	ica	0.5182	0.5162	0.5327	0.4980
	none	0.4523	0.4536	0.4799	0.5000
	pca	0.4134	0.3913	0.5338	0.3536
	zca + rbm	0.5523	0.5524	0.5527	0.5000
	tsne	0.5498	0.5478	0.5545	0.5064
norm	ae	0.8869	0.8886	0.9245	0.5000
	ica	0.8061	0.8035	0.8184	0.5953
	none	0.7673	0.7711	0.8473	0.5000
	pca	0.7243	0.7236	0.9144	0.7423
	zca + rbm	0.5768	0.5769	0.5770	0.5000
	tsne	0.5139	0.5085	0.5116	0.4712
norm,augment	ae	0.7782	0.7797	0.8278	0.5000
	ica	0.6852	0.6921	0.8328	0.7947
	none	0.7242	0.7268	0.7325	0.5000
	pca	0.4765	0.4769	0.7864	0.8360
	zca + rbm	0.5104	0.5053	0.4971	0.5000
	tsne	0.5000	0.5000	0.5000	0.5000
norm,blur	ae	0.7954	0.7977	0.8329	0.5000
	ica	0.7869	0.7702	0.8488	0.3504
	none	0.8254	0.8438	0.8652	0.5000
	pca	0.7075	0.7151	0.8675	0.1565
	zca + rbm	0.5825	0.5807	0.5761	0.5000
	tsne	0.5735	0.5678	0.5741	0.5386
norm,canny	ae	0.1798	0.1835	0.2744	0.5000
	ica	0.4907	0.4896	0.4616	0.8398

Preprocess	Selection	elastic	lasso	lin	oc-svm
	none	0.6577	0.6610	0.7267	0.5717
	pca	0.4148	0.3994	0.4604	0.8775
	zca + rbm	0.5289	0.5322	0.5341	0.5000
	tsne	0.5000	0.5000	0.5000	0.5000
norm,clahe	ae	0.8704	0.8744	0.8972	0.5000
	ica	0.8135	0.8125	0.8759	0.7706
	none	0.7089	0.7383	0.7911	0.6513
	pca	0.7764	0.7799	0.9032	0.6257
	zca + rbm	0.6584	0.6586	0.6619	0.5000
	tsne	0.5335	0.5453	0.5483	0.4760
norm,gray	ae	0.5016	0.5012	0.5008	0.5000
	ica	0.4997	0.4998	0.4996	0.4996
	none	0.4999	0.4999	0.4997	0.5000
	pca	0.5000	0.5000	0.4996	0.4996
	zca + rbm	0.5026	0.5019	0.5036	0.5000
	tsne	0.5106	0.4973	0.4846	0.5014
norm,skel	ae	0.5486	0.5327	0.5969	0.5000
	ica	0.5201	0.5075	0.5121	0.4906
	none	0.4528	0.4526	0.4801	0.5000
	pca	0.4333	0.3961	0.5288	0.3656
	zca + rbm	0.5527	0.5526	0.5528	0.5000
	tsne	0.5553	0.3803	0.5608	0.5051
std	ae	0.8827	0.8805	0.8852	0.5000
	ica	0.7250	0.7429	0.8090	0.7176
	none	0.8664	0.8674	0.8767	0.9182
	pca	0.8410	0.8405	0.8521	0.7611
	zca + rbm	0.5782	0.5783	0.5787	0.5000
	tsne	0.5237	0.5253	0.5126	0.4905
std,augment	ae	0.7686	0.7679	0.7861	0.5000
	ica	0.6196	0.6190	0.6522	0.8339
	none	0.8508	0.8521	0.8754	0.9461
	pca	0.4776	0.4675	0.7645	0.9039
	zca + rbm	0.5288	0.5274	0.5219	0.5000
	tsne	0.5000	0.5000	0.5000	0.5000

Preprocess	Selection	elastic	lasso	lin	oc-svm
std,blur	ae	0.8775	0.8723	0.8771	0.5000
	ica	0.7477	0.7497	0.8005	0.5098
	none	0.8394	0.8378	0.8420	0.9296
	pca	0.7258	0.7328	0.8023	0.7230
	zca + rbm	0.6166	0.6157	0.6182	0.5000
	tsne	0.4856	0.4916	0.4890	0.4954
std,canny	ae	0.4741	0.0863	0.0947	0.5000
	ica	0.4249	0.4961	0.4603	0.8468
	none	0.3746	0.3920	0.6547	0.5130
	pca	0.3002	0.2910	0.4241	0.8850
	zca + rbm	0.5907	0.5897	0.5891	0.5000
	tsne	0.5000	0.5000	0.5000	0.5000
std,clahe	ae	0.8152	0.7329	0.7356	0.5000
	ica	0.7717	0.8020	0.8672	0.7767
	none	0.4486	0.5299	0.6421	0.7379
	pca	0.7895	0.7872	0.8540	0.7691
	zca + rbm	0.6392	0.6378	0.6384	0.5000
	tsne	0.4948	0.5027	0.5083	0.5563
std,gray	ae	0.5013	0.5016	0.5010	0.5000
	ica	0.4999	0.4999	0.4999	0.4996
	none	0.4996	0.4996	0.5009	0.4996
	pca	0.4997	0.4997	0.4997	0.4996
	zca + rbm	0.5031	0.5025	0.5018	0.5000
	tsne	0.5015	0.4966	0.4993	0.4827
std,skel	ae	0.3026	0.3041	0.3059	0.5000
	ica	0.4995	0.5129	0.5051	0.5956
	none	0.4020	0.4008	0.4068	0.6441
	pca	0.4910	0.4914	0.4775	0.4303
	zca + rbm	0.5405	0.5402	0.5408	0.5000
	tsne	0.5341	0.5239	0.5426	0.4839

Preprocess	Selection	elastic	lasso	lin	oc-svm

Table 11: Average AUROCs of 10 classes for a given configuration for the MNIST dataset. This table lists a combination of the 'main effects' (432 configurations) and the 72 configurations from Table 10 above, which totals to 504 configurations. The highlighted cells are the top 10% AUROCs. The configuration "std,augment-none-SVM" has the highest AUROC at 0.9461.

The sizes of the ensembles that were run in the 10-minute computation budget are listed in Table 12. All of the configurations with zero ensemble size, or in other words 'the slowest', were with t-SNE feature selection, which is no different from what happened in the Credit Card Fraud dataset in Table 9. However, for the Satellite dataset in Table 6, t-SNE did not perform the slowest. This implies the impact of t-SNE slowing down the runs is more pronounced at higher-dimension datasets.

Without considering the preprocessing or the feature selection options, the configurations with higher frequencies of small ensemble sizes i.e. configurations with longer runtimes are the configurations with ElasticNet and LASSO submodels.

AVERAGE of Count		Ensemble			
Preprocess	Selection	elastic	lasso	lin	oc-svm
none	ae	2993	2996	3000	3000
	ica	2998	2990	3000	3000
	none	3000	3000	3000	3000
	pca	551	552	3000	3000
	zca + rbm	841	1032	3000	3000
	tsne	525	609	2515	1875
none,augment	ae	2658	2673	3000	3000
	ica	778	861	3000	3000
	none	2403	2381	3000	3000
	pca	456	414	3000	1607
	zca + rbm	547	593	3000	3000
	tsne	0	0	0	0
none,blur	ae	2984	2989	3000	3000
	ica	1966	1869	3000	3000
	none	2994	3000	3000	3000
	pca	491	477	3000	3000

Preprocess	Selection	elastic	lasso	lin	oc-svm
	zca + rbm	869	1029	3000	3000
	tsne	725	749	2612	2102
none,canny	ae	2988	2991	3000	3000
	ica	1050	1079	3000	3000
	none	1091	1090	3000	3000
	pca	418	473	3000	3000
	zca + rbm	923	1038	3000	3000
	tsne	0	0	0	0
none,clahe	ae	2655	2637	3000	3000
	ica	990	1013	3000	3000
	none	1676	1389	3000	3000
	pca	545	526	3000	3000
	zca + rbm	951	1057	3000	3000
	tsne	334	983	2958	2701
none,gray	ae	3000	3000	3000	3000
	ica	2997	3000	3000	3000
	none	3000	2991	3000	3000
	pca	2996	2998	3000	3000
	zca + rbm	3000	3000	3000	3000
	tsne	2992	2995	3000	3000
none,skel	ae	1405	1381	3000	3000
	ica	3000	3000	3000	3000
	none	3000	3000	3000	3000
	pca	703	2429	3000	3000
	zca + rbm	875	1023	3000	3000
	tsne	1012	982	3000	2516
norm	ae	1248	1237	3000	3000
	ica	2661	2702	3000	3000
	none	3000	3000	3000	3000
	pca	1027	1261	3000	3000
	zca + rbm	895	1035	3000	3000
	tsne	514	598	2564	1940
norm,augment	ae	777	792	3000	3000
	ica	841	885	3000	3000

Preprocess	Selection	elastic	lasso	lin	oc-svm
	none	2502	2463	3000	3000
	pca	711	888	3000	3000
	zca + rbm	552	594	3000	3000
	tsne	0	0	0	0
norm,blur	ae	1354	1382	3000	3000
	ica	1807	2011	3000	3000
	none	3000	3000	3000	3000
	pca	1270	1587	3000	3000
	zca + rbm	888	1035	3000	3000
	tsne	811	832	2804	2515
norm,canny	ae	2997	3000	3000	3000
	ica	1148	1157	3000	3000
	none	1036	1107	3000	3000
	pca	1462	2054	3000	3000
	zca + rbm	937	1036	3000	3000
	tsne	0	0	0	0
norm,clahe	ae	1620	1540	3000	3000
	ica	1792	1974	3000	3000
	none	1367	1674	3000	3000
	pca	1977	2329	3000	3000
	zca + rbm	919	1031	3000	3000
	tsne	722	785	2274	1899
norm,gray	ae	3000	3000	3000	3000
	ica	3000	3000	3000	3000
	none	3000	3000	3000	3000
	pca	3000	3000	3000	3000
	zca + rbm	3000	3000	3000	3000
	tsne	3000	3000	3000	3000
norm,skel	ae	1407	1401	3000	3000
	ica	3000	2995	3000	3000
	none	3000	3000	3000	3000
	pca	733	2456	3000	3000
	zca + rbm	889	1035	3000	3000
	tsne	1041	1094	3000	2491

Preprocess	Selection	elastic	lasso	lin	oc-svm
std	ae	2956	2946	3000	3000
	ica	2520	2546	3000	3000
	none	3000	3000	3000	3000
	pca	349	328	3000	3000
	zca + rbm	876	1038	3000	3000
	tsne	746	747	3000	2457
std,augment	ae	1746	1745	3000	3000
	ica	1139	1286	3000	3000
	none	2517	2538	3000	3000
	pca	486	460	3000	3000
	zca + rbm	539	581	3000	3000
	tsne	0	0	0	0
std,blur	ae	2880	2894	3000	3000
	ica	1636	1931	3000	3000
	none	2946	2928	3000	3000
	pca	323	356	3000	3000
	zca + rbm	894	1025	3000	3000
	tsne	1278	1277	3000	2976
std,canny	ae	2705	2656	3000	3000
	ica	1056	1046	3000	3000
	none	1014	1230	3000	3000
	pca	406	833	3000	3000
	zca + rbm	927	1037	3000	3000
	tsne	0	0	0	0
std,clahe	ae	2892	2927	3000	3000
	ica	1015	996	3000	3000
	none	1236	1702	3000	3000
	pca	488	459	3000	3000
	zca + rbm	926	1050	3000	3000
	tsne	692	691	1800	1649
std,gray	ae	3000	3000	3000	3000
	ica	3000	3000	3000	3000
	none	2983	2973	3000	3000
	pca	2973	2972	3000	3000

Preprocess	Selection	elastic	lasso	lin	oc-svm
	zca + rbm	3000	3000	3000	3000
	tsne	2994	2989	3000	3000
std,skel	ae	2688	2783	3000	3000
	ica	2988	2986	3000	3000
	none	3000	3000	3000	3000
	pca	367	320	3000	3000
	zca + rbm	897	1021	3000	3000
	tsne	1334	1343	3000	2954

Table 12: Average count (rounded-off to nearest decimal) of submodels in an ensemble within the 10-minute computation budget. The highlighted cells are the lowest 10%. Feature selection using t-SNE has the most number of zero ensemble sizes, which means t-SNE significantly slowed down the algorithm compared to the other feature selection methods.

5.4 CIFAR-10

The average AUROCs for the CIFAR-10 dataset with the 72 'core' configurations are listed in Table 13. Out of the 10 classes of images, the higher AUROCs occurred most frequently for deer and frog pictures. If we look at only the submodel types, Linear Regression has the most amount of higher AUROCs. On the contrary, OC-SVM has the least amount of higher AUROCs. OC-SVM also did not produce high AUROCs for the deer and the frog pictures, which have high AUROCs in other submodel types. A key problem with CIFAR-10 dataset is that, if no feature selection is done, the algorithm fails to run most of the time due to memory allocation issues.

AVERAGE of	AUROC		Ensemble			
Preprocess	Selection	Class	elastic	lasso	lin	oc-svm
none	ae	Airplane	0.5611	0.5604	0.5787	0.5000
		Automobile	0.5244	0.5255	0.5295	0.5000
		Bird	0.4903	0.4929	0.5211	0.5000
		Cat	0.5050	0.5046	0.5085	0.5000
		Deer	0.5617	0.5596	0.5724	0.5000
		Dog	0.5425	0.5427	0.5399	0.5000

Preprocess	Selection	Class	one. elastic	lasso	lin	oc-svm
		Frog	0.5688	0.5717	0.5936	0.5000
		Horse	0.5838	0.5796	0.5720	0.5000
		Ship	0.5985	0.5979	0.6150	0.5000
		Truck	0.5834	0.5840	0.5862	0.5000
	ae Average		0.5520	0.5519	0.5617	0.5000
	ica	Airplane	0.6039	0.6097	0.6293	0.6063
		Automobile	0.4221	0.4289	0.3933	0.5406
		Bird	0.6287	0.6248	0.6485	0.5644
		Cat	0.4665	0.4666	0.4724	0.4704
		Deer	0.6820	0.6855	0.7238	0.5896
		Dog	0.4357	0.4238	0.4331	0.4408
		Frog	0.6771	0.6861	0.7288	0.5187
		Horse	0.4487	0.4305	0.4262	0.5158
		Ship	0.5966	0.5914	0.6182	0.5708
		Truck	0.4454	0.4412	0.4190	0.4790
	ica Average		0.5407	0.5389	0.5493	0.5296
	pca	Airplane	0.6117	0.6174	0.6105	0.6447
		Automobile	0.4255	0.4335	0.4086	0.5361
		Bird	0.6231	0.6223	0.6330	0.5604
		Cat	0.4899	0.4892	0.4829	0.4287
		Deer	0.6802	0.6848	0.7125	0.5812
		Dog	0.4685	0.4580	0.4504	0.4387
		Frog	0.7111	0.7113	0.6991	0.4595
		Horse	0.4862	0.4839	0.4525	0.6165
		Ship	0.6173	0.6063	0.6312	0.5648
		Truck	0.4665	0.4610	0.4215	0.5019
	pca Average		0.5580	0.5568	0.5502	0.5333
	zca + rbm	Airplane	0.3950	0.3885	0.3851	0.5000
		Automobile	0.5429	0.5476	0.5478	0.5000
		Bird	0.5058	0.5087	0.5055	0.5000
		Cat	0.5329	0.5350	0.5373	0.5000
		Deer	0.5743	0.5788	0.5798	0.5000
		Dog	0.5204	0.5198	0.5193	0.5000
		Frog	0.6010	0.6069	0.6076	0.5000

Preprocess	Selection	Class	one. elastic	lasso	lin	oc-svm
		Horse	0.5395	0.5407	0.5384	0.5000
		Ship	0.4621	0.4602	0.4580	0.5000
		Truck	0.5322	0.5340	0.5296	0.5000
	zca + rbm Average		0.5206	0.5220	0.5208	0.5000
	tsne	Airplane	0.5072	0.5043	0.5121	0.4984
		Automobile	0.4329	0.4534	0.4312	0.5044
		Bird	0.5931	0.5927	0.5888	0.5374
		Cat	0.4711	0.4853	0.4991	0.4994
		Deer	0.5755	0.5793	0.5616	0.5063
		Dog	0.4871	0.4738	0.4747	0.4946
		Frog	0.6058	0.5829	0.5866	0.5154
		Horse	0.4889	0.4763	0.4775	0.4657
		Ship	0.5140	0.5148	0.5061	0.4940
		Truck	0.4814	0.4814	0.4772	0.5038
	tsne Average		0.5157	0.5144	0.5115	0.5019
norm	ae	Airplane	0.5857	0.5781	0.5863	0.5000
		Automobile	0.4755	0.4707	0.4649	0.5000
		Bird	0.5825	0.5771	0.5801	0.5000
		Cat	0.5061	0.5029	0.5041	0.5000
		Deer	0.6199	0.6121	0.6173	0.5000
		Dog	0.5334	0.5360	0.5266	0.5000
		Frog	0.6723	0.6718	0.6757	0.5000
		Horse	0.5379	0.5426	0.5373	0.5000
		Ship	0.6719	0.6935	0.6921	0.5000
		Truck	0.5496	0.5397	0.5490	0.5000
	ae Average		0.5735	0.5724	0.5733	0.5000
	ica	Airplane	0.6113	0.6187	0.6526	0.6100
		Automobile	0.4164	0.4142	0.3976	0.5254
		Bird	0.6352	0.6192	0.6479	0.5632
		Cat	0.4778	0.4662	0.4743	0.4760
		Deer	0.6814	0.6771	0.7189	0.5852
		Dog	0.4417	0.4389	0.4418	0.4380
		Frog	0.6857	0.6859	0.7280	0.5234
		Horse	0.4463	0.4863	0.4818	0.5304

Preprocess	Selection	Class	one. elastic	lasso	lin	oc-svm
		Ship	0.5967	0.5736	0.6148	0.5632
		Truck	0.4461	0.4458	0.4412	0.4885
	ica Average		0.5439	0.5426	0.5599	0.5303
	pca	Airplane	0.6025	0.5938	0.6305	0.6507
		Automobile	0.4360	0.4219	0.4061	0.5506
		Bird	0.6113	0.6001	0.6238	0.5728
		Cat	0.4880	0.4983	0.5111	0.4516
		Deer	0.6659	0.6718	0.6992	0.6176
		Dog	0.4640	0.4774	0.4665	0.4313
		Frog	0.6762	0.6797	0.7260	0.5152
		Horse	0.4806	0.4940	0.5098	0.6254
		Ship	0.6103	0.6131	0.6059	0.5994
		Truck	0.4771	0.4953	0.6129	0.5217
	pca Average		0.5512	0.5545	0.5792	0.5536
	zca + rbm	Airplane	0.3889	0.3852	0.3838	0.5000
		Automobile	0.5493	0.5511	0.5507	0.5000
		Bird	0.5071	0.5049	0.5043	0.5000
		Cat	0.5280	0.5353	0.5335	0.5000
		Deer	0.5740	0.5822	0.5778	0.5000
		Dog	0.5212	0.5204	0.5204	0.5000
		Frog	0.5956	0.6015	0.6018	0.5000
		Horse	0.5410	0.5418	0.5412	0.5000
		Ship	0.4525	0.4545	0.4514	0.5000
		Truck	0.5320	0.5316	0.5316	0.5000
	zca + rbm Average		0.5190	0.5208	0.5196	0.5000
	tsne	Airplane	0.4878	0.5083	0.5093	0.5000
		Automobile	0.4239	0.4491	0.4331	0.5204
		Bird	0.5870	0.5842	0.5919	0.5094
		Cat	0.4981	0.4810	0.4938	0.5283
		Deer	0.5600	0.5633	0.5672	0.5047
		Dog	0.4836	0.4883	0.4735	0.5213
		Frog	0.5978	0.5982	0.5946	0.5195
		Horse	0.4707	0.4749	0.4763	0.4868
		Ship	0.5173	0.5044	0.5124	0.4946

Preprocess	Selection	Class	one. elastic	lasso	lin	oc-svm
		Truck	0.4907	0.4790	0.4772	0.4873
	tsne Average		0.5117	0.5131	0.5129	0.5072
std	ae	Airplane	0.5371	0.5396	0.5417	0.5000
		Automobile	0.4949	0.4876	0.4936	0.5000
		Bird	0.5574	0.5612	0.5651	0.5000
		Cat	0.5137	0.5108	0.5076	0.5000
		Deer	0.6357	0.6336	0.6387	0.5000
		Dog	0.5070	0.5084	0.5100	0.5000
		Frog	0.6710	0.6703	0.6732	0.5000
		Horse	0.4997	0.4973	0.5085	0.5000
		Ship	0.5724	0.5746	0.5735	0.5000
		Truck	0.5404	0.5285	0.5251	0.5000
	ae Average		0.5529	0.5512	0.5537	0.5000
	ica	Airplane	0.6333	0.6286	0.6370	0.6044
		Automobile	0.4222	0.4160	0.3935	0.5313
		Bird	0.6332	0.6212	0.6381	0.5571
		Cat	0.4558	0.4715	0.4751	0.4789
		Deer	0.6865	0.6897	0.7227	0.5976
		Dog	0.4319	0.4355	0.4384	0.4490
		Frog	0.7031	0.6993	0.7442	0.5341
		Horse	0.4409	0.4200	0.4422	0.5398
		Ship	0.6039	0.5798	0.6295	0.5585
		Truck	0.4460	0.4238	0.4232	0.4905
	ica Average		0.5457	0.5385	0.5544	0.5341
	pca	Airplane	0.6082	0.6100	0.6114	0.6296
		Automobile	0.4239	0.4198	0.4059	0.5493
		Bird	0.6232	0.6147	0.6281	0.5574
		Cat	0.4942	0.5036	0.4884	0.4346
		Deer	0.6805	0.6866	0.7135	0.5891
		Dog	0.4597	0.4572	0.4547	0.4365
		Frog	0.6995	0.7008	0.7071	0.4464
		Horse	0.4867	0.5056	0.4619	0.6172
		Ship	0.6015	0.6041	0.6318	0.5770
		Truck	0.4516	0.4550	0.4168	0.5039

Preprocess	Selection	Class	one. elastic	lasso	lin	oc-svm
	pca Average		0.5529	0.5557	0.5520	0.5341
	zca + rbm	Airplane	0.3931	0.3884	0.3821	0.5000
		Automobile	0.5454	0.5462	0.5445	0.5000
		Bird	0.5079	0.5063	0.5101	0.5000
		Cat	0.5348	0.5361	0.5366	0.5000
		Deer	0.5769	0.5795	0.5784	0.5000
		Dog	0.5212	0.5223	0.5215	0.5000
		Frog	0.5943	0.6031	0.6025	0.5000
		Horse	0.5385	0.5386	0.5399	0.5000
		Ship	0.4586	0.4558	0.4551	0.5000
		Truck	0.5308	0.5302	0.5298	0.5000
	zca + rbm Average		0.5202	0.5206	0.5201	0.5000
	tsne	Airplane	0.4956	0.5048	0.5057	0.5260
		Automobile	0.4222	0.4297	0.4300	0.4971
		Bird	0.5898	0.5767	0.5895	0.4944
		Cat	0.4838	0.4896	0.4777	0.4934
		Deer	0.5836	0.5875	0.5874	0.5165
		Dog	0.4685	0.4785	0.4880	0.5019
		Frog	0.6159	0.6028	0.5912	0.4890
		Horse	0.4721	0.4712	0.4664	0.4833
		Ship	0.5284	0.5056	0.5277	0.4672
		Truck	0.4823	0.5116	0.4758	0.5145
	tsne Average		0.5142	0.5158	0.5140	0.4983

Table 13: AUROCs for CIFAR-10 dataset, with no image related preprocessing. Each value is the average of 10 runs. The highlighted cells are the top 10% highest AUROCs, and the majority of them have Linear Regression submodels.

Table 14 lists the average AUROCs for the 72 'core' configurations and the 432 'main effects' configurations. Each cell is an average of 10 runs of the 10 classes for a given configuration. The configuration "norm,augment-pca-lin" has the highest AUROC at 0.5867.

Among the preprocessing options of normalizing, standardizing, and no preprocessing, normalizing the data seemed to have the most impact on AUROC. Image-specific prepro-

cessing like augmentation, blurring, and grayscaling also add to improve the AUROCs. This is more pronounced for normalized data. In feature selection, Autoencoder, no feature selection, and PCA contributed the most to increasing the AUROCs. Finally, if we were to consider only the submodel types, Linear Regression has the highest frequency of larger AUROCs among the submodel types.

AVERAGE of AUROC		Ensemble			
Preprocess	Selection	elastic	lasso	lin	oc-svm
none	ae	0.5520	0.5519	0.5617	0.5000
	ica	0.5407	0.5389	0.5493	0.5296
	pca	0.5580	0.5568	0.5502	0.5333
	zca + rbm	0.5206	0.5220	0.5208	0.5000
	tsne	0.5157	0.5144	0.5115	0.5019
none,augment	ae	0.5447	0.5441	0.5539	0.5000
	ica	0.5291	0.5268	0.5442	0.5740
	pca	0.5494	0.5519	0.5436	0.5606
	zca + rbm	0.5203	0.5199	0.5205	0.5000
	tsne	0.5000	0.5000	0.5000	0.5000
none,blur	ae	0.5527	0.5518	0.5650	0.5000
	ica	0.5316	0.5351	0.5465	0.4866
	pca	0.5574	0.5579	0.5448	0.5309
	zca + rbm	0.5292	0.5290	0.5284	0.5000
	tsne	0.5097	0.5120	0.5133	0.5007
none,canny	ae	0.5088	0.5084	0.5106	0.5000
	ica	0.4922	0.4907	0.4961	0.5215
	pca	0.4999	0.5011	0.5020	0.4909
	zca + rbm	0.5141	0.5155	0.5152	0.5000
	tsne	0.5151	0.5165	0.5247	0.4944
none,clahe	ae	0.5147	0.5151	0.5187	0.5000
	ica	0.5280	0.5240	0.5367	0.5209
	pca	0.5295	0.5285	0.5461	0.5791
	zca + rbm	0.5093	0.5084	0.5082	0.5000
	tsne	0.5016	0.4998	0.5084	0.4959
none,gray	ae	0.5397	0.5390	0.5477	0.5000
	ica	0.5243	0.5269	0.5301	0.5065

Preprocess	Selection	elastic	lasso	lin	oc-svm
	none	0.5556	0.5556	0.5594	0.5000
	pca	0.5308	0.5313	0.5264	0.5320
	zca + rbm	0.5197	0.5193	0.5202	0.5000
	tsne	0.5078	0.5095	0.5078	0.4964
none,skel	ae	0.5203	0.5225	0.5247	0.5000
	ica	0.5339	0.5317	0.5415	0.5376
	none	0.5348	0.5352	0.5394	0.5000
	pca	0.5349	0.4983	0.5363	0.5162
	zca + rbm	0.5111	0.5120	0.5114	0.5000
	tsne	0.5201	0.5298	0.5216	0.5028
norm	ae	0.5735	0.5724	0.5733	0.5000
	ica	0.5439	0.5426	0.5599	0.5303
	pca	0.5512	0.5545	0.5792	0.5536
	zca + rbm	0.5190	0.5208	0.5196	0.5000
	tsne	0.5117	0.5131	0.5129	0.5072
norm,augment	ae	0.5832	0.5821	0.5851	0.5000
	ica	0.5497	0.5454	0.5659	0.5695
	pca	0.5489	0.5498	0.5867	0.5851
	zca + rbm	0.5202	0.5206	0.5200	0.5000
	tsne	0.5000	0.5000	0.5000	0.5000
norm,blur	ae	0.5686	0.5694	0.5648	0.5000
	ica	0.5386	0.5421	0.5519	0.4875
	pca	0.5439	0.5435	0.5582	0.5505
	zca + rbm	0.5199	0.5211	0.5211	0.5000
	tsne	0.5130	0.5107	0.5157	0.5004
norm,canny	ae	0.5053	0.5064	0.5093	0.5000
	ica	0.4935	0.4936	0.4973	0.5179
	pca	0.4958	0.4983	0.4963	0.4855
	zca + rbm	0.5152	0.5153	0.5155	0.5000
	tsne	0.5163	0.5151	0.5212	0.4934
norm,clahe	ae	0.4840	0.4781	0.4768	0.5000
	ica	0.5280	0.5391	0.5411	0.5186
	pca	0.5393	0.5299	0.5581	0.5619
	zca + rbm	0.5090	0.5084	0.5082	0.5000

Preprocess	Selection	elastic	lasso	lin	oc-svm
	tsne	0.5001	0.5029	0.5038	0.4960
norm,gray	ae	0.5590	0.5580	0.5600	0.5000
	ica	0.5328	0.5229	0.5306	0.5050
	none	0.5567	0.5572	0.5594	0.5000
	pca	0.5310	0.5297	0.5268	0.5488
	zca + rbm	0.5192	0.5207	0.5202	0.5000
	tsne	0.5090	0.5112	0.5114	0.5043
norm,skel	ae	0.5255	0.5251	0.5284	0.5000
	ica	0.5288	0.5310	0.5443	0.5335
	none	0.5343	0.5342	0.5400	0.5000
	pca	0.5325	0.5009	0.5365	0.5169
	zca + rbm	0.5126	0.5115	0.5115	0.5000
	tsne	0.5285	0.5245	0.5328	0.4984
std	ae	0.5529	0.5512	0.5537	0.5000
	ica	0.5457	0.5385	0.5544	0.5341
	pca	0.5529	0.5557	0.5520	0.5341
	zca + rbm	0.5202	0.5206	0.5201	0.5000
	tsne	0.5142	0.5158	0.5140	0.4983
std,augment	ae	0.5169	0.5179	0.5202	0.5000
	ica	0.5277	0.5255	0.5400	0.5644
	pca	0.5446	0.5471	0.5358	0.5650
	zca + rbm	0.5186	0.5186	0.5199	0.5000
	tsne	0.5000	0.5000	0.5000	0.5000
std,blur	ae	0.5536	0.5522	0.5540	0.5000
	ica	0.5339	0.5322	0.5467	0.4821
	pca	0.5562	0.5570	0.5458	0.5341
	zca + rbm	0.5258	0.5259	0.5266	0.5000
	tsne	0.5261	0.5204	0.5195	0.4973
std,canny	ae	0.4637	0.4612	0.4601	0.5000
	ica	0.4899	0.4919	0.4957	0.5215
	pca	0.5002	0.4943	0.5049	0.4873
	zca + rbm	0.5149	0.5150	0.5155	0.5000
	tsne	0.5202	0.5202	0.5229	0.4970
std,clahe	ae	0.5345	0.5362	0.5390	0.5000

Preprocess	Selection	elastic	lasso	lin	oc-svm
	ica	0.5314	0.5324	0.5421	0.5224
	pca	0.5381	0.5354	0.5481	0.5739
	zca + rbm	0.5096	0.5087	0.5089	0.5000
	tsne	0.5070	0.4994	0.5002	0.4888
std,gray	ae	0.5370	0.5403	0.5400	0.5000
	ica	0.5277	0.5265	0.5268	0.5087
	none	0.5473	0.5473	0.5491	0.4802
	pca	0.5289	0.5326	0.5278	0.5376
	zca + rbm	0.5203	0.5201	0.5198	0.5000
	tsne	0.5059	0.5121	0.5050	0.4993
std,skel	ae	0.5429	0.5428	0.5422	0.5000
	ica	0.5380	0.5463	0.5464	0.5353
	none	0.5353	0.5354	0.5363	0.5380
	pca	0.5455	0.5445	0.5472	0.5333
	zca + rbm	0.5007	0.5013	0.5007	0.5000
	tsne	0.5244	0.5237	0.5231	0.4873

Table 14: AUROCs for CIFAR-10 dataset, including image-related preprocessing. Each cell in this table is an average AUROCs of 10 classes for a given configuration. This table lists a combination of the 'main effects' (432 configurations) and the 72 'core' configurations from Table 13 above, which totals to 504 configurations. The configuration "norm,augment-pca-lin" has the highest AUROC at 0.5867. The highlighted cells are the top 10% highest AUROCs. Image augmentation appears to boost performance.

The average sizes of the ensembles that ran in the 10-minute computation budget for the CIFAR-10 dataset are listed in Table 15. Similar to the other datasets above, the ensemble with zero submodel count occurred only for t-SNE feature selection. Regardless of the preprocessing and the feature selection options, ElasticNet and LASSO have the highest frequency of small ensemble sizes among the submodel types i.e. they are slower than Linear Regression and OC-SVM submodels.

AVERAGE of Count		Ensemble			
Preprocess	Selection	elastic	lasso	lin	oc-svm
none	ae	3000	3000	3000	3000

Preprocess	Selection	elastic	lasso	lin	oc-svm
	ica	1035	1030	3000	3000
	pca	527	557	3000	3000
	zca + rbm	676	999	3000	3000
	tsne	1452	1531	3000	2987
none,augment	ae	2633	2607	3000	3000
	ica	905	1020	3000	3000
	pca	494	519	3000	2520
	zca + rbm	233	313	3000	3000
	tsne	0	0	0	0
none,blur	ae	2999	2989	3000	3000
	ica	1003	1110	3000	3000
	pca	520	537	3000	3000
	zca + rbm	692	1020	3000	3000
	tsne	1296	1346	3000	2938
none,canny	ae	3000	3000	3000	3000
	ica	919	924	3000	3000
	pca	460	433	3000	3000
	zca + rbm	664	976	3000	3000
	tsne	2774	2531	3000	3000
none,clahe	ae	2998	3000	3000	3000
	ica	935	1029	3000	3000
	pca	561	564	3000	3000
	zca + rbm	611	1084	3000	3000
	tsne	1494	1507	2730	2647
none,gray	ae	3000	2999	3000	3000
	ica	1156	1131	3000	3000
	none	1174	1185	3000	3000
	pca	534	538	3000	3000
	zca + rbm	1001	1167	3000	3000
	tsne	1140	1182	2864	2393
none,skel	ae	1720	1711	3000	3000
	ica	1763	1727	3000	3000
	none	3000	3000	3000	3000
	pca	984	2803	3000	3000

Preprocess	Selection	elastic	lasso	lin	oc-svm
	zca + rbm	944	1206	3000	3000
	tsne	2783	2608	2730	2693
norm	ae	3000	3000	3000	3000
	ica	1110	1146	3000	3000
	pca	770	898	3000	3000
	zca + rbm	688	1007	3000	3000
	tsne	1485	1350	3000	2996
norm,augment	ae	3000	2975	3000	3000
	ica	1065	1086	3000	3000
	pca	753	844	3000	3000
	zca + rbm	234	316	2981	3000
	tsne	0	0	0	0
norm,blur	ae	3000	3000	3000	3000
	ica	1174	1182	3000	3000
	pca	839	1000	3000	3000
	zca + rbm	672	1033	3000	3000
	tsne	1281	1272	3000	2949
norm,canny	ae	3000	3000	3000	3000
	ica	926	919	3000	3000
	pca	1011	1179	3000	3000
	zca + rbm	682	965	3000	3000
	tsne	2612	2560	3000	3000
norm,clahe	ae	3000	3000	3000	3000
	ica	1069	1032	3000	3000
	pca	798	947	3000	3000
	zca + rbm	599	1090	3000	3000
	tsne	1445	1500	2700	2676
norm,gray	ae	3000	3000	3000	3000
	ica	1176	1168	3000	3000
	none	1120	1171	3000	3000
	pca	872	1259	3000	3000
	zca + rbm	989	1157	3000	3000
	tsne	1144	1201	2861	2368
norm,skel	ae	1657	1731	3000	3000

Preprocess	Selection	elastic	lasso	lin	oc-svm
	ica	1753	1645	3000	3000
	none	3000	3000	3000	3000
	pca	937	2760	3000	3000
	zca + rbm	952	1203	3000	3000
	tsne	2575	2571	2790	2749
std	ae	2902	2950	3000	3000
	ica	939	1039	3000	3000
	pca	529	546	3000	3000
	zca + rbm	683	1006	3000	3000
	tsne	1564	1498	3000	3000
std,augment	ae	1808	1783	3000	3000
	ica	880	967	3000	3000
	pca	489	509	3000	2989
	zca + rbm	236	311	2977	3000
	tsne	0	0	0	0
std,blur	ae	2961	2981	3000	3000
	ica	1079	1065	3000	3000
	pca	517	535	3000	3000
	zca + rbm	671	1025	3000	3000
	tsne	1391	1411	3000	2994
std,canny	ae	2792	2806	3000	3000
	ica	892	850	3000	3000
	pca	451	494	3000	3000
	zca + rbm	673	933	3000	3000
	tsne	2648	2672	3000	3000
std,clahe	ae	3000	2996	3000	3000
	ica	925	968	3000	3000
	pca	547	561	3000	3000
	zca + rbm	626	1075	3000	3000
	tsne	1551	1546	2664	2527
std,gray	ae	2726	2771	3000	3000
	ica	1137	1129	3000	3000
	none	2834	2823	3000	3000
	pca	513	477	3000	3000

Preprocess	Selection	elastic	lasso	lin	oc-svm
	zca + rbm	959	1162	3000	3000
	tsne	1274	1339	3000	2667
std,skel	ae	2924	2924	3000	3000
	ica	1362	1362	3000	3000
	none	2723	2647	3000	3000
	pca	339	361	3000	3000
	zca + rbm	970	1182	3000	3000
	tsne	2399	2320	2610	2643

Table 15: Average count (rounded-off to nearest decimal) of submodels in an ensemble within the 10-minute computation budget. The highlighted cells are the lowest 10%. ElasticNet and LASSO submodels have the highest frequency of small ensemble size, which implies that the ensembles with these submodels took the longest to run.

6 Benchmarking

Table 16 lists the AUROCs of our best-performing configurations against CBLOF, KNN, Isolation Forest, and DEAN. For the Satellite dataset, the largest AUROC is normalized data and LASSO submodel with no features selection; for Credit Card Fraud dataset it is from normalized data and Linear Regression submodel with no feature selection; for MNIST dataset it is standardized and augmented data, and OC-SVM submodel with no feature selection; and for CIFAR-10 dataset it is normalized and augmented data with feature selection using PCA and Linear Regression submodel.

Table 16 shows that for tabular datasets i.e. Satellite and Credit Card Fraud, our algorithm performance is quite competitive and came in second only to KNN. However, for high-dimension datasets such as MNIST and CIFAR-10 datasets, deep learning models, like DEAN, produced the highest AUROCs among the algorithms. For CIFAR-10 dataset, our algorithm performed the worst among the algorithms, but it came in a close second position after DEAN for the MNIST dataset.

AVERAGE of AUROC		Algorithm				
Dataset	Class	CBLOF	KNN	IForest	DEAN	SEAN
CCFraud		0.9458	0.9656	0.9496		0.9567
CCFraud Average		0.9458	0.9656	0.9496		0.9567
Satellite		0.9534	0.9736	0.9511		0.9684
Satellite Average		0.9534	0.9736	0.9511		0.9684
MNIST	0	0.9780	0.9924	0.9626	0.9917	0.9923
	1	0.9963	0.9982	0.9919	0.9988	0.9969
	2	0.8682	0.9099	0.7302	0.9527	0.9318
	3	0.8648	0.9073	0.7978	0.9625	0.9344
	4	0.9003	0.9253	0.8510	0.9732	0.9204
	5	0.8851	0.9321	0.6936	0.9660	0.9289
	6	0.9667	0.9721	0.8362	0.9867	0.9778
	7	0.9482	0.9619	0.8948	0.9743	0.9700
	8	0.8507	0.8596	0.7039	0.9314	0.8612
	9	0.9299	0.9508	0.8628	0.9680	0.9468
MNIST Average		0.9188	0.9409	0.8325	0.9705	0.9461
CIFAR-10	Airplane	0.6454	0.6693	0.6515	0.6863	0.5834
	Automobile	0.4273	0.4376	0.4423	0.6267	0.4878
	Bird	0.6599	0.6807	0.6503	0.5798	0.5579
	Cat	0.5218	0.5192	0.5327	0.6357	0.5890
	Deer	0.7680	0.7658	0.7507	0.6876	0.6798
	Dog	0.5184	0.5021	0.5338	0.6299	0.5619
	Frog	0.7378	0.7394	0.7308	0.7292	0.6171
	Horse	0.5180	0.5065	0.5466	0.6291	0.5715
	Ship	0.6946	0.6917	0.6918	0.7305	0.7000
	Truck	0.4526	0.4269	0.5221	0.6989	0.5190
CIFAR-10 Average		0.5944	0.5939	0.6052	0.6634	0.5867

Table 16: Our best setups (parameter sets) compared with CBLOF, KNN, Isolation Forest, and DEAN in each dataset. For Credit Card Fraud dataset our highest AUROC is from a norm-none-Lin configuration, for Satellite dataset it is norm-none-LASSO, for MNIST dataset it is std,aug-none-SVM, and for CIFAR-10 dataset it is norm,aug-pca-Lin. The AUROCs for DEAN are pulled from the DEAN paper (Klüttermann and Müller, 2022).

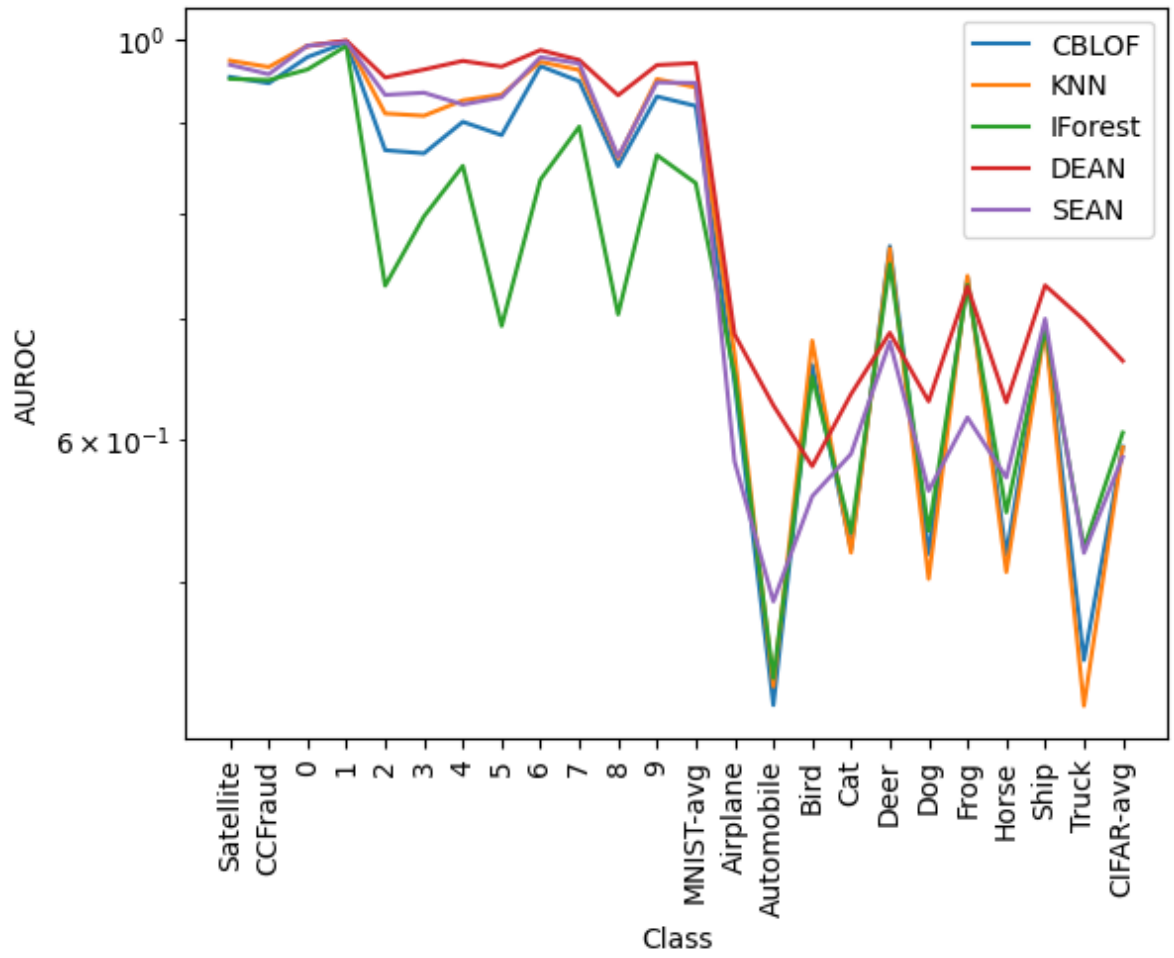

Figure 14: Plot of the benchmark results from Table 16.

7 Conclusion

Motivated by the DEAN algorithm and the fact that there are some researches out there with findings that say a deep learning based anomaly detection algorithm may be no better than its shallow counterpart, this thesis explored the performance of an anomaly detection algorithm built from an ensemble of simple submodels. The algorithm works in four stages: pre-processing, feature selection, feature bagging, and an ensemble of simple submodels, where each submodel of the ensemble uses the loss functions and the anomaly scores from the DEAN algorithm.

The experiments were run on four datasets: satellite, credit card fraud, MNIST, and CIFAR-10, and the finding is that there is no "one size fits all" configuration that performs

the best across all of the datasets, as mentioned in Section 6. For the satellite dataset, LASSO and Linear Regression submodels produced the higher AUROCs on average, and OC-SVM produced the worst average AUROCs. However, Linear Regression and OC-SVM submodels ran quicker than ElasticNet and LASSO submodels. This suggests choosing an ensemble of Linear Regression submodels to have a good balance between AUROCs and runtimes in lower-dimension datasets like the Satellite dataset. Similar results were obtained from the Credit Card Fraud dataset, a slightly higher dimension dataset, where Linear Regression submodel struck a good balance between high AUROCs and short runtimes. This is the case for MNIST and CIFAR-10 datasets as well.

Pre-processing the dataset also had a significant impact on performance. Either normalization or standardization improved the algorithm performance in all of the four datasets. Adding image-specific preprocessing like augmentation or blurring or CLAHE further improved the performance in MNIST and CIFAR-10 datasets. One key point to mention is that the experiments did not cover the exhaustive combination of all the preprocessing options due to scheduling constraints so exploring those options may be an idea for further research experiments.

When it comes to the feature selection options, performing no feature selection produced higher AUROCs in satellite, credit card fraud, and MNIST datasets. However, for a higher dimension dataset like the CIFAR-10 dataset, there were memory allocation failures if no feature selection was done; but if grayscaling or skeletonization was done on the dataset, the jobs ran fine with no feature selection and produced competitive AUROCs.

In conclusion, we found from the experiments that the anomaly detection algorithm using simple submodels produced competitive performance but lagged behind its deep-learning counterpart, and preprocessing the dataset using either normalization or standardization had a significant impact on performance, and an ensemble of Linear Regression submodels struck a good balance between accuracy and runtime.

Bibliography

[1] Charu C. Aggarwal. *Outlier Analysis*. Springer Publishing Company, Incorporated, 2nd edition, 2016. ISBN 3319475770.

[2] Andrea Borghesi, Andrea Bartolini, Michele Lombardi, Michela Milano, and Luca Benini. Anomaly detection using autoencoders in high performance computing systems. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01): 9428–9433, jul 2019. doi: 10.1609/aaai.v33i01.33019428. URL https://doi.org/10.1609%2Faaai.v33i01.33019428.

[3] Abdenour Bounsiar and Michael G. Madden. One-class support vector machines revisited. In *2014 International Conference on Information Science Applications (ICISA)*, pages 1–4, 2014. doi: 10.1109/ICISA.2014.6847442.

[4] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: Identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, page 93–104, New York, NY, USA, 2000. Association for Computing Machinery. ISBN 1581132174. doi: 10.1145/342009.335388. URL https://doi.org/10.1145/342009.335388.

[5] Benedikt Böing, Simon Klüttermann, and Emmanuel Müller. Post-robustifying deep anomaly detection ensembles by model selection. In *2022 IEEE International Conference on Data Mining (ICDM)*, pages 861–866, 2022. doi: 10.1109/ICDM54844.2022.00098.

[6] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986.

[7] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3), jul 2009. ISSN 0360-0300. doi: 10.1145/1541880.1541882. URL https://doi.org/10.1145/1541880.1541882.

[8] Ayan Chatterjee and Bestoun S. Ahmed. Iot anomaly detection methods and applications: A survey. *Internet of Things*, 19:100568, 2022. ISSN 2542-6605. doi: https://doi.org/10.1016/j.iot.2022.100568. URL https://www.sciencedirect.com/science/article/pii/S2542660522000622.

[9] Nitesh Chawla and Wei Wang. *Proceedings of the 2017 SIAM International Conference on Data Mining (SDM)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2017. doi: 10.1137/1.9781611974973. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611974973.

[10] F.Y. Edgeworth. Xli. on discordant observations. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 23(143):364–375, 1887. doi: 10.1080/14786448708628471. URL https://doi.org/10.1080/14786448708628471.

[11] Tharindu Fernando, Harshala Gammulle, Simon Denman, Sridha Sridharan, and Clinton Fookes. Deep learning for medical anomaly detection – a survey. *ACM Comput. Surv.*, 54(7), jul 2021. ISSN 0360-0300. doi: 10.1145/3464423. URL https://doi.org/10.1145/3464423.

[12] Nicolas Gillis. *Nonnegative Matrix Factorization*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2020. doi: 10.1137/1.9781611976410. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611976410.

[13] Markus Goldstein and Seiichi Uchida. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PLOS ONE*, 11(4):1–31, 04 2016. doi: 10.1371/journal.pone.0152173. URL https://doi.org/10.1371/journal.pone.0152173.

[14] Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions, 2010.

[15] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

[16] Waleed Hilal, S. Andrew Gadsden, and John Yawney. Financial fraud: A review of anomaly detection techniques and recent advances. *Expert Systems with Applications*, 193:116429, 2022. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2021.116429. URL https://www.sciencedirect.com/science/article/pii/S0957417421017164.

[17] Geoffrey Hinton and Sam Roweis. Stochastic neighbor embedding. In *Proceedings of the 15th International Conference on Neural Information Processing Systems*, NIPS'02, page 857–864, Cambridge, MA, USA, 2002. MIT Press.

[18] Aapo Hyvärinen and Erkki Oja. Independent component analysis: algorithms and applications. *Neural networks : the official journal of the International Neural Network Society*, 13 4-5:411–30, 2000. URL https://api.semanticscholar.org/CorpusID:11959218.

[19] Keith Jack. *Video Demystified: A Handbook for the Digital Engineer*. 2007. ISBN 9780750683951.

[20] Nicholas Jeffrey, Qing Tan, and José R. Villar. A review of anomaly detection strategies to detect threats to cyber-physical systems. *Electronics*, 12(15), 2023. ISSN 2079-9292. doi: 10.3390/electronics12153283. URL https://www.mdpi.com/2079-9292/12/15/3283.

[21] Junfeng Jing, Shenjuan Liu, Gang Wang, Weichuan Zhang, and Changming Sun. Recent advances on image edge detection: A comprehensive review. *Neurocomputing*, 503:259–271, 2022. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2022.06.083. URL https://www.sciencedirect.com/science/article/pii/S0925231222008141.

[22] Simon Klüttermann and Emmanuel Müller. Dean: Deep ensemble anomaly detection. 2022. URL https://github.com/KDD-OpenSource/DEAN.

[23] Bartosz Krawczyk. Learning from imbalanced data: open challenges and future directions. *Progress in Artificial Intelligence*, 5(4):221 – 232, 2016. doi: 10.1007/s13748-016-0094-0. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85043605198&doi=10.1007%2fs13748-016-0094-0&partnerID=40&md5=048d4390e4eb698c3b66bafbbc7e724c. Cited by: 1379; All Open Access, Hybrid Gold Open Access.

[24] Alex Krizhevsky. Learning multiple layers of features from tiny images. pages 32–33, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[25] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[26] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422, 2008. doi: 10.1109/ICDM.2008.17.

[27] Matthew B. A. McDermott, Lasse Hyldig Hansen, Haoran Zhang, Giovanni Angelotti, and Jack Gallifant. A closer look at auroc and auprc under class imbalance, 2024.

[28] Madalina Olteanu, Fabrice Rossi, and Florian Yger. Meta-survey on outlier and anomaly detection. *Neurocomputing*, 555:126634, 2023. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2023.126634. URL https://www.sciencedirect.com/science/article/pii/S0925231223007579.

[29] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[30] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning, 2017.

[31] Andrea Dal Pozzolo, Olivier Caelen, Reid A. Johnson, and Gianluca Bontempi. Calibrating probability with undersampling for unbalanced classification. In *2015 IEEE Symposium Series on Computational Intelligence*, pages 159–166, 2015. doi: 10.1109/SSCI.2015.33.

[32] Ferdinand Rewicki, Joachim Denzler, and Julia Niebling. Is it worth it? comparing six deep and classical methods for unsupervised anomaly detection in time series. *Applied Sciences*, 13(3), 2023. ISSN 2076-3417. doi: 10.3390/app13031778. URL https://www.mdpi.com/2076-3417/13/3/1778.

[33] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. Deep one-class classification. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4393–4402. PMLR, 10–15 Jul 2018. URL `https://proceedings.mlr.press/v80/ruff18a.html`.

[34] Lukas Ruff, Jacob R. Kauffmann, Robert A. Vandermeulen, Grégoire Montavon, Wojciech Samek, Marius Kloft, Thomas G. Dietterich, and Klaus-Robert Müller. A unifying review of deep and shallow anomaly detection. *Proceedings of the IEEE*, 109(5):756–795, 2021. doi: 10.1109/JPROC.2021.3052449.

[35] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008. URL `http://www.jmlr.org/papers/v9/vandermaaten08a.html`.

[36] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009. ISBN 1441412697.

[37] T. Y. Zhang and Ching Y. Suen. A fast parallel algorithm for thinning digital patterns. *Commun. ACM*, 27(3):236–239, 1984. URL `http://dblp.uni-trier.de/db/journals/cacm/cacm27.html#ZhangS84`.

[38] Hui Zou and Trevor Hastie. Regularization and Variable Selection Via the Elastic Net. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 67 (2):301–320, 03 2005. ISSN 1369-7412. doi: 10.1111/j.1467-9868.2005.00503.x. URL `https://doi.org/10.1111/j.1467-9868.2005.00503.x`.

Eidesstattliche Versicherung

(Affidavit)

Peka, Vanlal

Name, Vorname
(surname, first name)

231125

Matrikelnummer
(student ID number)

☐ Bachelorarbeit
(Bachelor's thesis)

■ Masterarbeit
(Master's thesis)

Titel
(Title)

Anomaly Detection using an Ensemble with Simple Sub-models

Ich versichere hiermit an Eides statt, dass ich die vorliegende Abschlussarbeit mit dem oben genannten Titel selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

I declare in lieu of oath that I have completed the present thesis with the above-mentioned title independently and without any unauthorized assistance. I have not used any other sources or aids than the ones listed and have documented quotations and paraphrases as such. The thesis in its current or similar version has not been submitted to an auditing institution before.

Aizawl, 08.05.2024

Ort, Datum
(place, date)

Unterschrift
(signature)

Belehrung:
Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -).

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird ggf. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin") zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Official notification:
Any person who intentionally breaches any regulation of university examination regulations relating to deception in examination performance is acting improperly. This offense can be punished with a fine of up to EUR 50,000.00. The competent administrative authority for the pursuit and prosecution of offenses of this type is the Chancellor of TU Dortmund University. In the case of multiple or other serious attempts at deception, the examinee can also be unenrolled, Section 63 (5) North Rhine-Westphalia Higher Education Act (*Hochschulgesetz, HG*).

The submission of a false affidavit will be punished with a prison sentence of up to three years or a fine.

As may be necessary, TU Dortmund University will make use of electronic plagiarism-prevention tools (e.g. the "turnitin" service) in order to monitor violations during the examination procedures.

I have taken note of the above official notification:*

Aizawl, 08.05.2024

Ort, Datum
(place, date)

Unterschrift
(signature)

***Please be aware that solely the German version of the affidavit ("Eidesstattliche Versicherung") for the Bachelor's/ Master's thesis is the official and legally binding version.**