ABSTRACT

Neural Network Watchdog for Out-of-Distribution Input Mitigation

Justin M. Bui, Ph.D.

Mentor: Robert J. Marks II, Ph.D.

Neural networks have often been described as black boxes. The prevalence of publicly available neural networks and the application of transfer learning has allowed for the development of systems with minimal understanding of the data distribution. For example, a generic neural network trained to differentiate between kittens and puppies will classify a picture of a kumquat as either a kitten or a puppy, despite the kumquat residing outside the known data distribution. The neural network watchdog is a technique which screens trained classifier and regression machine input candidates to determine the distribution validity, and allows for methods of out-of-distribution removal with minimal performance impact.

Neural Network Watchdog for Out-of-Distribution Input Mitigation

by

Justin M. Bui, B.S., M.S.

A Dissertation

Approved by the Department of Electrical and Computer Engineering

_____

Kwang Y. Lee, Ph.D., Chairperson

Submitted to the Graduate Faculty of
Baylor University in Partial Fulfillment of the
Requirements for the Degree
of

Doctor of Philosophy

Approved by the Dissertation Committee

_____

Robert J. Marks II, Ph.D. Chairperson

_____

Kwang Y. Lee, Ph.D.

_____

Joe C. Yelderman, Ph.D.

_____

Mark M. Budnik, Ph.D.

Accepted by the Graduate School

December 2021

_____

J. Larry Lyon, Ph.D., Dean

Page bearing signatures is kept on file in the Graduate School.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank Dr. Robert J. Marks II for his wisdom, guidance, and kindness throughout my research. I am grateful for the support of Mrs. Simcik for all of the behind the scenes work she has provided. I would like to thank my family and friends for their love, support, and encouragement throughout my academic journey. Finally, a thank you to the Baylor University Electrical and Computer Engineering department and faculty for the opportunity to pursue this endeavor.

DEDICATION

To My Mother, Deborah Bui, In Loving Memory.

CHAPTER ONE

Introduction

One of the hottest areas of research and development in the disciplines of electrical and computer engineering is artificial intelligence. John McCarthy defines artificial intelligence as ”the science and engineering of making intelligent machines, especially intelligent computer programs.” [54]. One of the most popular sub-spaces within artificial intelligence is machine learning. Mariam Webster defines machine learning as ”the process by which a computer is able to improve its own performance (as in analyzing image files) by continuously incorporating new data into an existing statistical model” [93]. With the ever-increasing availability of cloud-based and distributed computing, coupled with the proliferation of open-source tools and exciting news headlines, interest in machine learning has grown tremendously.

Machine learning has applications in a variety of fields, from medical systems and financial analysis to robotics and machine vision. The various applications often require different system designs and implementations, however the challenges encountered are often similar. At it’s core, machine learning consists of three primary categories: unsupervised learning, supervised learning, and reinforcement learning.

Unsupervised learning techniques allow for a system to recognize patterns within data, without matching it to specific identifications. Unsupervised learning methods include clustering methods, principal and independent component analysis, autoencoders, generative adversarial networks, and transformers. Applications of unsupervised learning include computer vision systems, recommendation engines, and anomaly detection.

Supervised learning techniques use labeled data to train a system to match particular patterns to a specific identification. Supervised learning methods include Naive Bayes

models, support vector machines, regression models, and neural networks. Applications of supervised learning include object recognition, sentiment analysis, and predictive analytic.

Reinforcement learning techniques differ from those of supervised and unsupervised learning. Reinforcement learning is based on trial and error, where the system, or agent, is rewarded or penalized for actions taken based on the outcome. The agents are provided the capability to interact with their environment through actions and observations. Applications of reinforcement learning include autonomous vehicle and robotics control, game-play systems, and industrial automation.

One driving factor in the new machine learning boom is the development of various open-source machine learning toolboxes, such as FastAI, PyTorch, and TensorFlow. The prevalence of these toolboxes has allowed for the rapid design, development, and evaluation of machine learning models. These models have been used in a variety of applications, including various classification and regression tasks such as object identification, stock price prediction, and component failure prediction.

Many of these models are available as open-source, and can be readily deployed using only a few lines of code. Listing 1.1 shows an example implementation of RESNET50 [29], an open source model which used for image recognition. At this point, the model is ready to be trained, although it is common practice to implement preprocessing on the data to help optimize performance.

```
inputs = tf.keras.Input(shape=(224, 224, 3))
RN50 = tf.keras.applications.resnet50.ResNet50(
include_top=True, input_shape=input_shape,
weights='imagenet', classes=1000,
classifier_activation='softmax'
)(inputs)
model = tf.keras.Model(inputs, RN50)
```

Listing 1.1: Implementation of a RESNET50 Image Classification Neural Network

Many of the open-sourced models are utilized by taking advantage of a technique known as transfer learning. Transfer learning allows existing models to be used by copying some, or all, of their layers, training weights, and biases into new models. This allows newly developed models to take advantage of state-of-the-art performance without having to design an entirely new model from scratch.

While the design and development of models has been made easier with these tool boxes, the quality of a model ultimately depends on the quality of the training data. Data which is used to train and evaluate a model is also known as in-distribution data. In an ideal scenario, a deployed model will only interact with in-distribution data, which results in a high confidence output. One of the biggest challenges to real-world deployments, however, is the identification and mitigation of out-of-distribution data.

Chapter Two provides a historical review of the current techniques to identify out-of-distribution data. To address the issues associated with out-of-distribution data, the neural network watchdog technique has been developed to identify out-of-distribution data and mitigate it's impact on model performance. Chapter Three introduces the high level concepts of the neural network watchdog, which are then demonstrated in Chapter Four.

CHAPTER TWO

Historical Review and Literature Survey

The prevalence of neural networks continues to increase as the tools and techniques for regression and classification tasks become more readily available. Despite their recent popularity, neural networks date back to the 1950's, when Rosenblatt et al. created the first functional perceptron [72]. Research and development associated with neural networks has continued since the development of the perceptron, and in more recent years has garnered public attention with events such as AlphaGo's successful defeat of Lee Sedol [21]. An introductory overview of the history of neural networks is provided by Yadav et al. [99].

The increase in popularity, both academically and industrially, has been spurred by and helped to spur the rapid development of open source tools, such as TensorFlow [1] and PyTorch [65]. Various implementations of neural networks, developed using these tools, can be found throughout a variety of industries. Garg et al. published a survey of neural network techniques present in the medical industry in [20], which discusses the use of various deep neural network techniques, such as Convolutional Neural Networks and Support Vector Machines as they apply to a variety of medical topics. Similar studies have been published which discuss applications such finance [62], oil and gas [27], manufacturing [76], agriculture [39], and autonomous driving [24].

2.1 Out-of-Distribution Detection

As mentioned in Chapter One, one of the biggest challenges associated with neural networks is the impact of identifying and the process to mitigate out-of-distribution data. As systems become more complex, and pipelines are transitioned from evaluation to production environments, the need for online out-of-distribution detection increases [28]. Out-of-distribution detection is often times closely coupled to anomaly detection [22, 31, 60], often referred to as novelty filtering [14, 26, 79, 80].

There have been several techniques applied to identify anomalous data, as demonstrated by the surveys performed by Chalapathy and Chawla [11], Thudumu et al. [82], and Pang et al. [63]. Bulusu et al. [10] and Wang et al. [89] provide a more in-depth analysis of the various techniques used in deep neural network anomaly detection. Several of these techniques are highlighted in the following sections. While the techniques introduced have shown vast improvement in anomaly detection, several challenges still remain, as demonstrated by Nguyen et al. [59].

2.1.1 Probability Based Models

One common approach to anomaly detection is the use of probabilistic models, such as those presented by Ren et al. [71] and Hechtlinger et al. [30]. These approaches have demonstrated improvements over more traditional probability methods, such as the softmax based probabilities discussed by Hendrycks and Gimpel [31]. The probability models are not limited to image tasks, however, other tasks such as network intrusion detection [40] also take advantage of probabilistic models.

2.1.2 Component Analysis Techniques

The use of principal component and independent component analysis have gained traction as methods for anomaly detection. Huang et al. [33] provide a good introduction to principal component analysis based anomaly detection. Wang, Yang, and Li [88] discuss the use of independent component analysis, coupled with Bayesian platforms for anomaly detection. Principal and independent component analysis techniques have commonly been used for anomaly detection in hyper-spectral data analysis [25, 36, 38, 50], as well as in network intrusion detection systems [3, 15, 16, 90, 96].

2.1.3 Clustering Techniques

Many machine learning applications use clustering techniques for classification and regression. Clustering methods allow for anomaly detection by examining several metrics,

including distance [5, 43, 45, 47, 53, 57, 69]. One limitation associated with anomaly detection clustering techniques is the decreased performance in the presence of large numbers of anomalous data.

2.1.4 Autoencoders

The area of anomaly detection that is of most interest to this research is the autoencoder. Autoencoders have demonstrated successful anomaly detection [52, 70, 75, 104] through a variety of methods, including feature extraction [73, 101, 98], probabilistic reconstruction [4, 46, 94], latent representation analysis [66, 77, 86], generative exploitation [51], and reconstruction error analysis [7, 8, 9, 17, 18, 42].

2.2 Autoencoders

The autoencoder is a fundamental component of the neural network watchdog. Autoencoders have demonstrated use in a variety of applications beyond anomaly detection. Some of the more popular applications of autoencoders include noise reduction [12, 23, 49, 84, 102, 103], object detection [19, 44, 55, 68], feature extraction [56, 67, 74, 97, 100], and data generation [32, 35, 41, 87].

As an unsupervised learning method, the autoencoder implicitly learns by estimating a lower dimensional manifold on which training data lives [80, 81]. The feature space dimension is determined by the cardinality of the autoencoder's input and output. The dimension of the latent representation is determined by the size of the bottleneck or waist layer of the autoencoder. As a result, data presented to a properly trained autoencoder will generate an output similar to the input [7].

The ability to generate an output which is similar to an input is leveraged by the neural network watchdog [7, 8, 9]. When out-of-distribution data is presented to a trained autoencoder, an output is generated which deviates from the initial input. This deviation is leveraged by calculating an error and comparing it to a determined threshold. Errors exceeding the threshold are labeled as out-of-distribution and are discarded.

CHAPTER THREE

Methods

The *neural network watchdog* is a technique which can help mitigate the impact of out-of-distribution data on classification and regression networks. This is achieved by using a generative network to reconstruct the input data and using this reconstruction to calculate a difference score. The difference score is compared to a threshold in order to determine whether input should be considered in or out of distribution. The selection of threshold criteria is application specific, and may be adjusted to match desired system performance. The following chapter outlines the design and implementation methodology of a neural network watchdog.

3.1 The Neural Network Watchdog

A neural network watchdog can be implemented in a variety of architectures, and can vary greatly in size, scale, and complexity. These architectures fall in to three primary categories, *Disjoint*, *Symbiotic*, and *Multi-tier*. All three of the watchdog categories are comprised of three core components:

(1) Generative Network or Component
(2) Error Measurement Mechanism
(3) Defined Error Threshold

3.1.1 Generative Component

An autoencoder is used as the generative component of the watchdog. As seen in Chapter Two, autoencoders show remarkable flexibility, but are of specific interest due to their generative capabilities. An autoencoder can be decomposed in to two primary components, an encoder and a decoder, which share a common latent space. Figure 3.1 shows an example structure of an autoencoder.

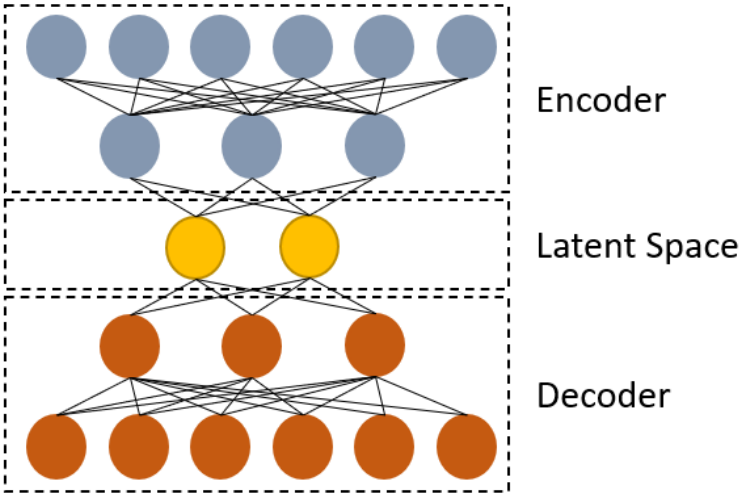

Figure 3.1: An autoencoder is comprised of an encoder and decoder, with a shared latent space.

When out-of-distribution data is presented to a trained autoencoder, an output is generated which deviates from the initial input. This deviation is leveraged by calculating an error and comparing it to a determined threshold. Errors exceeding the threshold are labeled as out-of-distribution, and are discarded.

3.1.2 Difference Measurement

A variety of input difference measurement techniques exist, each with their own advantages and disadvantages. Some of the most common calculations are distance based measures, including the mean squared error (MSE), root mean squared error (RMSE), and mean absolute error (MAE). Equations for these calculations can be found in Equations 3.1, 3.2, and 3.3 .

$$MSE = \frac{\sum(x - \hat{x})^2}{n} \tag{3.1}$$

$$RMSE = \sqrt{\frac{\sum(x - \hat{x})^2}{n}} \tag{3.2}$$

$$MAE = \frac{\sum |x - \hat{x}|}{n} \tag{3.3}$$

These calculations can be readily applied to a variety of different data types such as images and time-series sequences. There are times, however, where more advanced calculations may be required. For example, structural similarity, or SSIM, is a well demonstrated image similarity technique [91, 92], which is commonly used in modern video and image analysis. The calculation for structural similarity can be found in Equation 3.4 [91]. As can be seen, the difference calculations can increase greatly in complexity.

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1) + (2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{3.4}$$

The constants C_1 and C_2 are defined by Equations 3.5 and 3.6, where L represents the data input range value, and K_1 and K_2 are constants with a value $K_n << 1$ [91].

$$C_1 = (K_1 * L)^2 \tag{3.5}$$

$$C_2 = (K_2 * L)^2 \tag{3.6}$$

3.1.3 Defined Error Threshold

The selection of an appropriate threshold is determined based on the desired performance of the system. There are a number of trade-offs or concessions that may be made, depending on the desired level of out-of-distribution filtering. For example, a very tight threshold (i.e., a low RMSE value), may result in fantastic out-of-distribution detection, but may also result in the elimination of a considerable number of in-distribution inputs as well. Conversely, a loose threshold may allow for nearly all in-distribution inputs to be accepted, but also permit a multitude of out-of-distribution inputs to be accepted as well.

In disjoint and symbiotic watchdogs, the threshold comparison aspect of the watchdog is simply an evaluation of the difference measurement and the threshold. In a multi-tier watchdog, several different thresholds may be required, depending on the techniques used

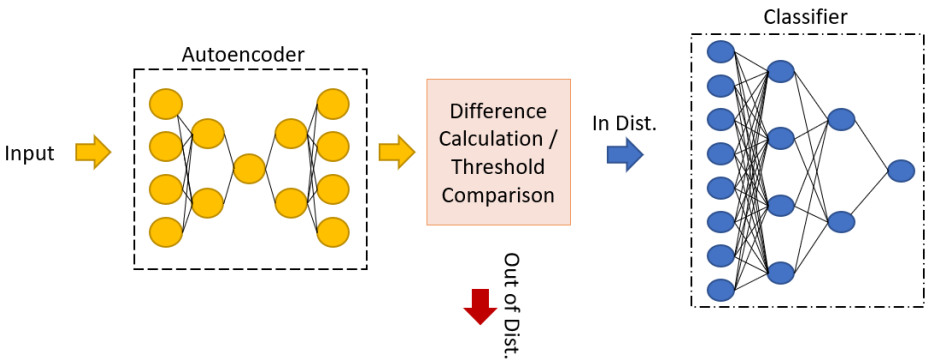

Figure 3.2: The structure of a disjoint watchdog. This structure allows for independent training and deployment of the core and generative networks.

at each tier. Table 3.1 shows a high level summary of some of the trade-offs between design complexity, design flexibility, and training complexity associated with the three architectures.

3.2 Disjoint Neural Network Watchdog

The most simple watchdog architecture is the disjoint watchdog, which is comprised of entirely independent generative and core networks [7]. This allows for the generative network to be designed and developed without impacting the design of the core classification network. Figure 3.2 shows the structure of a typical disjoint watchdog, where the output of the threshold comparison is passed into the classifier if it is determined to be in-distribution, and is otherwise discarded.

One of the advantages to the disjoint watchdog is the ability to train and deploy the generative component and the core classifier independently. This flexibility allows for the continuous improvement of both networks with minimal impact to the other, and greatly simplifies the input pipeline when compared to other watchdog architectures.

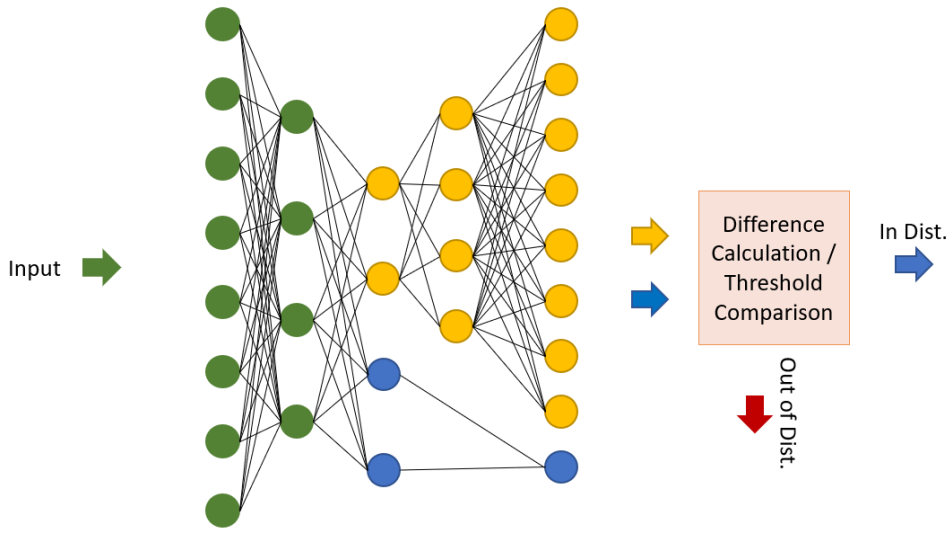

Figure 3.3: The structure of a symbiotic watchdog. This structure takes advantage of shared input layers, which allows for the watchdog to make decisions based on the same latent space representations as the core classifier.

3.3 Symbiotic Neural Network Watchdog

The symbiotic watchdog architecture is comprised of a hybrid generative-classification network, where the generative network and core classifier share common input layers up to the latent space [8]. Figure 3.3 shows the general structure of a symbiotic watchdog. This structure allows the watchdog to determine input validity on the same latent space representations that are used in the core classification network.

Since the core classifier and the watchdog share the layers between the input and the latent space, training the symbiotic network must take in to account the impact of back-propagation from two output sources. The symbiotic network must take bias and weight impacts from both the generative and classification components, which leads to a more complex development, training, and deployment process. The trade off for this complexity is that this architecture leads to higher accuracy and more optimized computational performance.

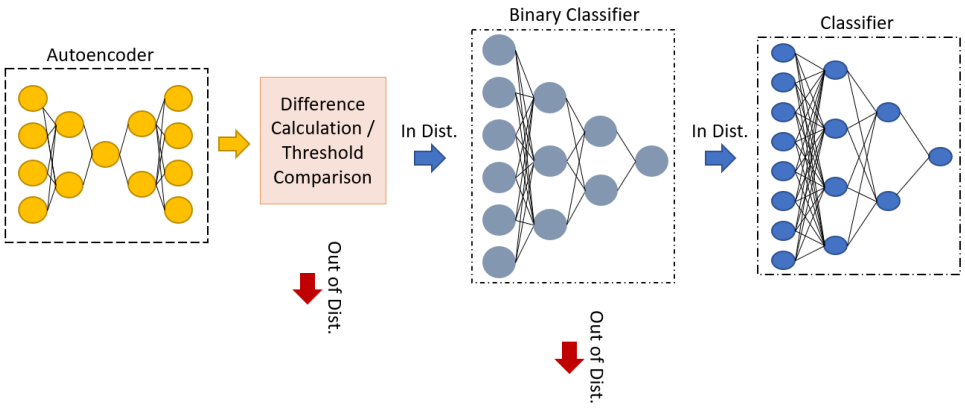

Figure 3.4: An example multi-tier watchdog structure which combines an autoencoder with a binary classifier. This type of structure allows for a less stringent threshold function, while still providing similar out-of-distribution detection performance.

Table 3.1: Watchdog Architecture Summary

Architecture	Design Complexity	Design Flexibility	Training Complexity
Disjoint	Medium	Low	Low
Symbiotic	Medium	Medium	High
Multi-tier	Varies	High	Varies

3.4 Multi-tier Neural Network Watchdog

The multi-tier neural network watchdog is comprised of multiple sub-networks, which are combined to provide more advanced data distribution detection capabilities. This architecture allows for the use of different threshold settings and analysis techniques by combining different neural network types. One example is the combination of an autoencoder and a binary classifier, where the autoencoder uses a standard error function for the threshold, and the binary classifier is then used to further differentiate between low-error out-of-distribution and in-distribution inputs [18]. An example of this type of architecture can be seen in Figure 3.4.

Other potential multi-tier architectures exist as well, such as the example seen in Figure 3.5. Since various sub-networks can be combined based on desired performance

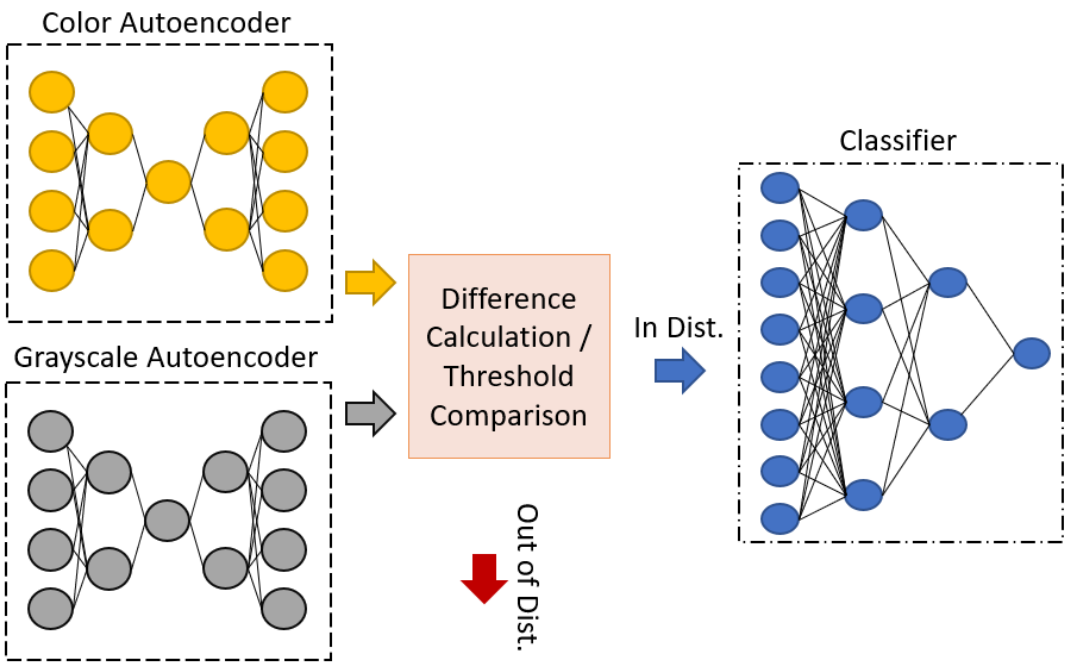

Figure 3.5: An example multi-tier watchdog structure which uses two autoencoders, one designed for grayscale images and one for color images, which both feed into the difference measurement.

criteria, the multi-tier approach allows for the greatest flexibility, at the cost of complexity and scalability. Table 3.1 shows the subjective design complexity and flexibility of the different watchdog architectures, as well as the associated training complexity.

3.5 Implementing a Neural Network Watchdog

The design, development, and deployment of a neural network watchdog can vary depending on the type of watchdog being developed. Disjoint watchdogs may be designed, trained, and deployed entirely separate from the core classification network. Since symbiotic watchdogs share layers with the core classifier, the combined network must be trained prior to deployment. The process for implementing a multi-tier watchdog will vary depending on the specific design.

3.5.1 Training a Neural Network Watchdog

The successful deployment of a neural network watchdog is directly dependent upon the quality of the generative component's training. The training processes for each of the disjoint, symbiotic and multi-tier watchdogs is described below.

3.5.1.1 Disjoint Watchdog Training. The training process for a disjoint watchdog focuses on the autoencoder training process. Once the structure of the autoencoder has been defined, it is trained using the identical training data used to train the core classifier. The training process for the autoencoder is not required to use the same training hyperparameters as the core classifier.

3.5.1.2 Symbiotic Watchdog Training. The training process for a symbiotic watchdog involves simultaneous training of the core classifier and generator components. Since both the generator and classifier share common layers, the back propagation portion of network training adjusts weights based on both loss functions. The loss weights of each component must be adjusted to simultaneously optimize classification accuracy and generator quality. An example of how loss weights are defined in a model can be found Listing in 3.1.

```
symbiotic.compile(
optimizer=tf.keras.optimizers.Adam(learning_rate=.001),
loss={
  "Classifier": tf.keras.losses.categorical_crossentropy,
  "Generator": tf.keras.losses.MSE,
},
loss_weights=[1.0,0.75],
metrics = ['accuracy']
)
```

Listing 3.1: Symbiotic Watchdog Weight Definition

3.5.1.3 *Multi-tier Watchdog Training*

The training process for a multi-tier watchdog is similar to the disjoint watchdog. The training can be performed independently from the core classifier, assuming no symbiotic methods are used. Depending upon the watchdog's design, the training data may require modification to accommodate different distribution detection mechanisms.

3.6 Deploying a Neural Network Watchdog

Once the watchdog has been trained and verified, it is ready for deployment. The deployment can be treated as either an entirely new system, or as a modification to existing systems, depending on the application.

3.6.1 Disjoint and Multi-tier Watchdog Deployment

Disjoint and multi-tier neural network watchdogs are typically deployed prior to the core classifier, often as a part of the input pipeline. The watchdog is placed at the end of the pipeline, enabling distribution mitigation after the preprocessing has been completed. Generative network outputs are compared to their inputs, where any errors exceeding the threshold are discarded. Outputs permitted by the watchdog are then passed directly in to the core classifier.

3.6.2 Symbiotic Watchdog Deployment

Since a symbiotic watchdog is built in to the core classifier, the model can be deployed with no additional modifications to the input pipeline. The classification output is temporarily gated until the generator output has been verified. If the generator's output and input deviate in excess of the permitted threshold, the classification's output is discarded.

CHAPTER FOUR

Experimentation and Results

In order to demonstrate the feasibility of the neural network watchdog, several experiments have been developed and performed to demonstrate the design, development and evaluation of the different watchdog architectures on a variety of different input data types. These experiments include the demonstrations of a disjoint, a symbiotic and a multi-tier watchdog in coordination with an image classification network, as well as a disjoint watchdog used in coordination with a signal analysis and classification network.

4.1 Disjoint Watchdog Image Classifier Experiment

In this experiment, a disjoint watchdog is designed for use with a classification neural network, which is designed to classify hand written digits from the MNIST dataset. The disjoint watchdog will consist of an autoencoder for input reconstruction, combined with a root mean squared error threshold function. In order to evaluate the feasibility of the watchdog, out-of-distribution data from the Fashion MNIST [95] will be added to the evaluation data.

4.1.1 Network Structures

For this experiment, both the core classifier and the autoencoder are convolutional neural networks. The networks are designed independently from one another, and as such, have slightly different architectures. It would be feasible to design the autoencoder using several of the layers of the core classifier, however, for the purposes of demonstration, the autoencoder has been developed from scratch.

4.1.1.1 Classifier Structure. The core classifier is comprised of three 2D convolution layers, with each layer paired with a pooling layer. After the convolution and pooling layers, a flatten layer is used to reduce the representation to our latent space, which is one

dimensional. Dropout is applied at this step to help mitigate over-fitting during training, then a fully connected (dense) layer is added to reduce the latent space to the number of classes present in the training and evaluation (in-distribution) dataset. A softmax activation is then used for our output. Figure 4.1 shows the summary plot of the classifier.

4.1.1.2 Autoencoder Structure. The autoencoder architecture, shown in Figure 4.2, is designed as a separate encoder and decoder, seen in Figures 4.3 and 4.4 which are then combined. The encoder is comprised of two 2D convolution layers, as well as a flatten layer and a dense layer to produce a 16 element latent space. The decoder is comprised of a dense layer to convert the latent space back to the flattened dimension, then a reshape layer to allow for transpose layers to be used. Three layers of transposes are used to reproduce the original input shape.

4.1.2 Training and Evaluation Data

Two subsets of data are used for the training and evaluation of the core classifier, disjoint watchdog, and the combined system. The training dataset is comprised of 60,000 in-distribution images, while the test dataset is comprised of 10,000 in-distribution and 10,000 out-of-distribution images.

4.1.2.1 In-Distribution Data. The MNIST dataset is comprised of 70,000 black and white images of hand-written numbers, with values between 0 and 9. For the purposes of this experiment, the dataset will be split in to 50,000 images for training, 10,000 images for training validation, and 10,000 images for evaluation. The training, validation, and test datasets contain an equal distribution of digit classes. An example of the MNIST digits can be seen in Figure 4.5.

4.1.2.2 Out-of-Distribution Data. The Fashion MNIST dataset is also comprised of black and white images of different types of clothing, such as boots, shoes, jackets, and pants. The mixed distribution dataset will include 10,000 images from the Fashion MNIST

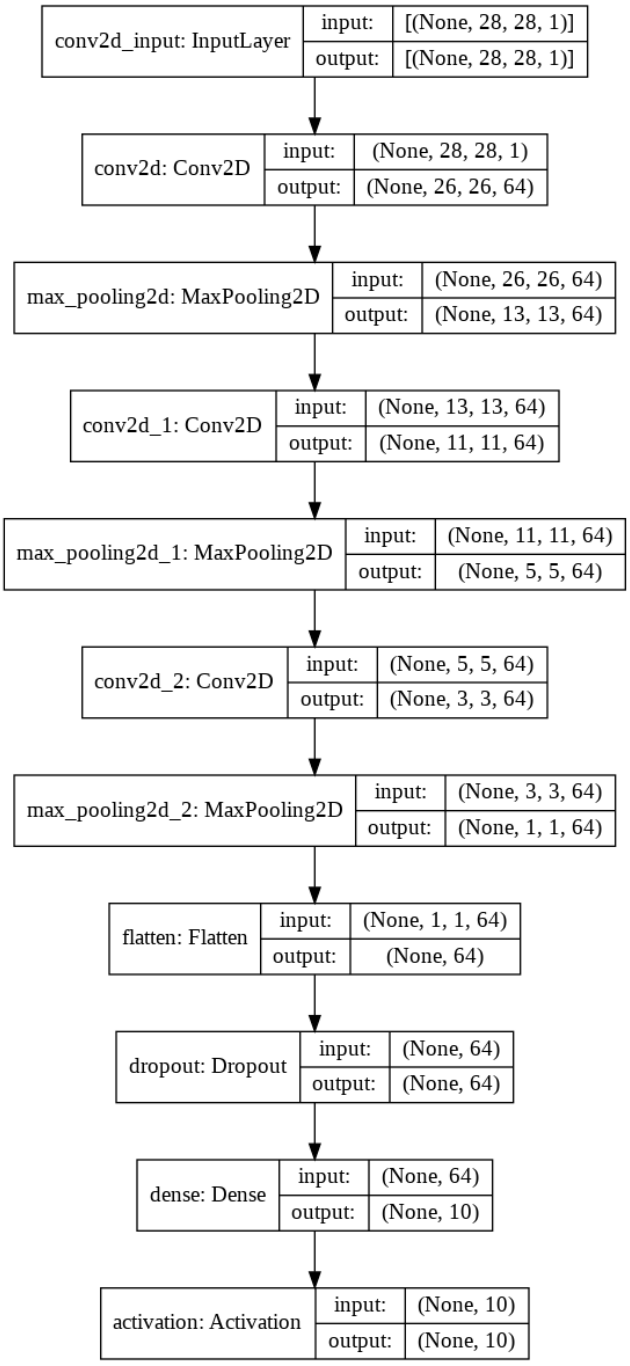

Figure 4.1: The summary plot of the core classification convolutional neural network.

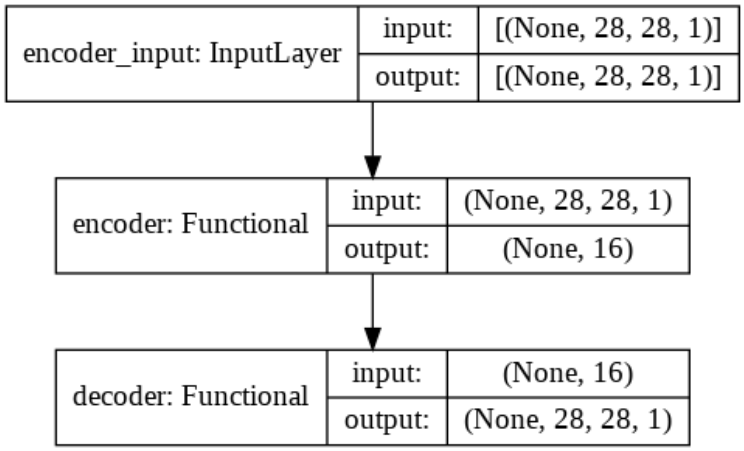

Figure 4.2: The summary plot of the autoencoder neural network, which is the generative component of the watchdog.

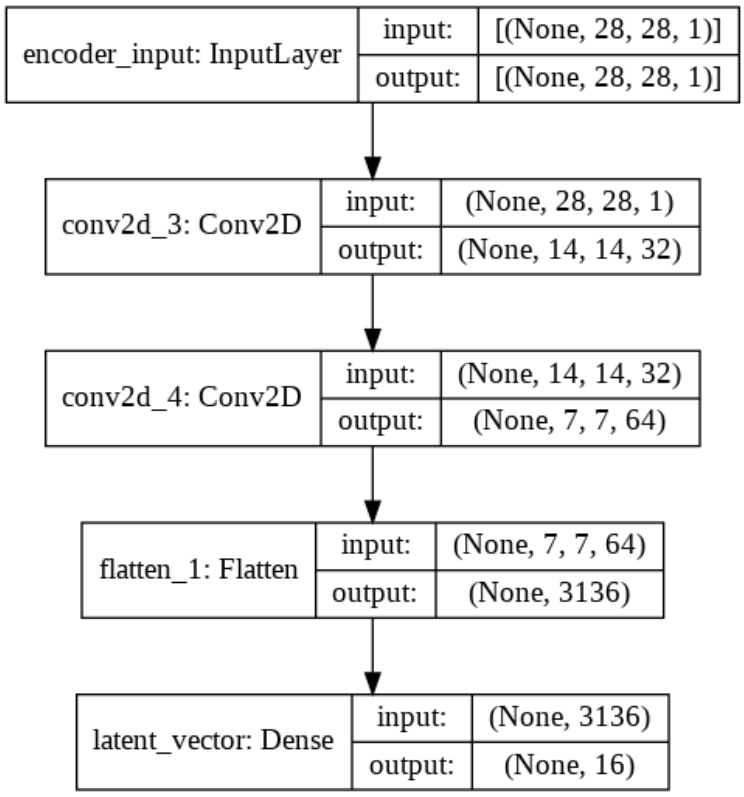

Figure 4.3: The summary plot of the encoder portion of the autoencoder, which is responsible for generating the latent space representation of the input.

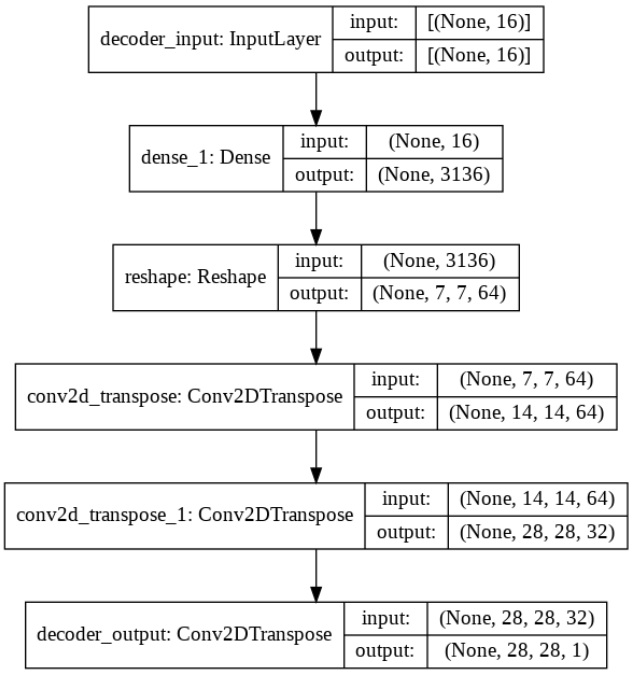

Figure 4.4: The summary plot of the decoder portion of the autoencoder, which is responsible for generating an output from a latent space representation.

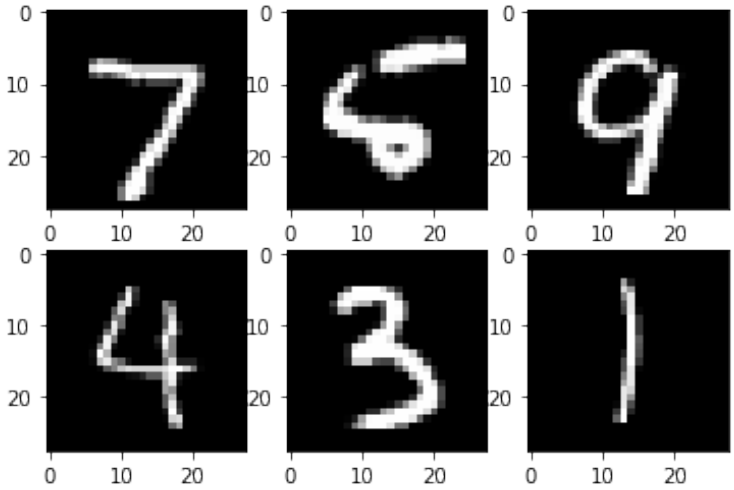

Figure 4.5: Example images from the MNIST Dataset.

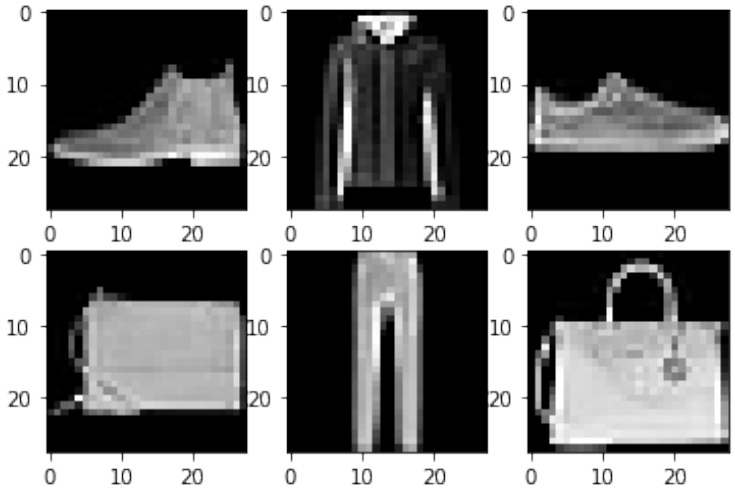

Figure 4.6: Example images from the Fashion MNIST Dataset.

Table 4.1: Disjoint Watchdog Training Hyperparameters

Hyperparameter	Value
Epochs	10
Batch Size	128
Dropout	0.15

dataset. Since the Fashion MNIST and MNIST images are both 28 by 28 pixel images, no additional augmentation is required for this experiment. An example of the Fashion MNIST clothing items can be seen in Figure 4.6

4.1.3 Training the Networks

The training process for both the core classifier and autoencoder is straight forward with a disjoint watchdog. The classification network is trained to match the labels to an input image, whereas the autoencoder is trained to match an input image to itself. Since the networks are independent, they may be trained in either order, and if desired, using different hyperparameters. For this experiment, the autoencoder and core classifier will use identical hyperparameters, which are defined in Table 4.1.

4.1.4 *Threshold Evaluation and Selection*

The threshold value is determined by the desired end performance of the system. Once the autoencoder has been trained, it can be used to reconstruct the test images. Once the test images have all been reconstructed, the difference function may be calculated for each image. For this experiment, the RMSE is calculated for the in-distribution and the out-of-distribution images. Figures 4.7 and 4.8 show examples of in-distribution and out-of-distribution inputs, outputs, and difference images to the autoencoder.

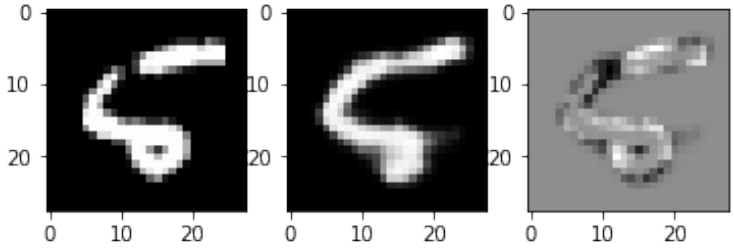

Figure 4.7: Original, regenerated, and difference images of a sample from the MNIST dataset. The RMSE for this example is: ≈ 4.1091

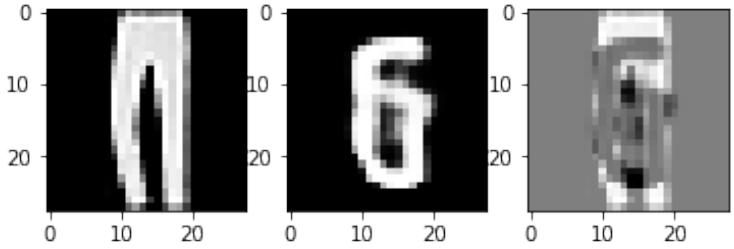

Figure 4.8: Original, regenerated, and difference images of a sample from the Fashion MNIST dataset. The RMSE for this example is: ≈ 8.2966

The easiest method for determining a threshold using a distance based metric, such as the RMSE, is to perform a brute-force calculation of how many images are permitted

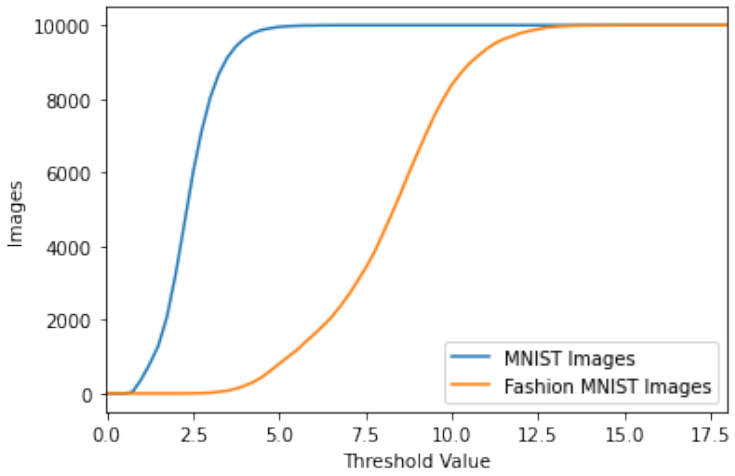

Figure 4.9: Number of images permitted by the watchdog, as a function of RMSE threshold value.

at each threshold value. Figure 4.9 shows a breakdown of how many MNIST and Fashion MNIST images are permitted at each RMSE value. From our observations, a RMSE threshold value of between 4 and 5 is a strong candidate for selection.

4.1.5 Network Performances

With a potential threshold value selected, the network performances can be evaluated. In order to evaluate the quality of the core classifier, the in-distribution accuracy will be used. This accuracy serves as the baseline, and represents a best-case scenario for system performance. The autoencoder and difference score are evaluated on the ability to distinguish between in-distribution and out-of-distribution images from the mixed-distribution dataset. The final evaluation includes the autoencoder with difference score and the core classifier, also known as the guarded classifier. The guarded classifier is evaluated on the mixed-distribution dataset.

4.1.5.1 Core Classifier Performance. After training the core classifier with the parameters described in Table 4.1, the core classifier scores a 98.62% accuracy on the in-distribution test dataset. The Receiver Operator Curves, or ROC curves for the individual

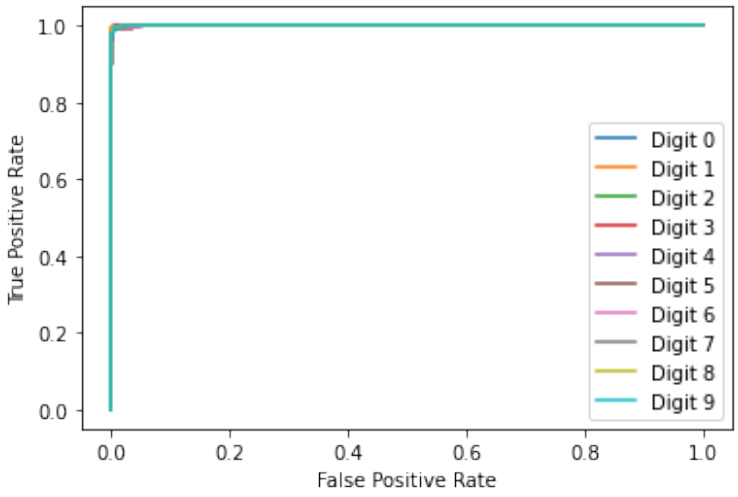

Figure 4.10: The ROC curves for the core classifier, by individual class.

classes, as well as the aggregated performance can be seen in Figures 4.10 and 4.11. These curves indicate a strong ability to correctly classify the in-distribution test data.

4.1.5.2 Autoencoder Performance. After training the autoencoder with the hyperparameters mentioned in Table 4.1, the autoencoder is evaluated in coordination with the difference score. Figure 4.12 shows the autoencoders performance as a function of the RMSE value calculated between the reconstruction and original images. This curve indicates a strong ability to differentiate between in-distribution and out-of-distribution data contained in the mixed-distribution dataset.

4.1.6 Watchdog Guarded Classifier Performance

The watchdog guarded evaluation measures the combined autoencoder, difference score, and core classifier performance as one unit. The input data is first evaluated by the autoencoder and compared to the threshold, prior to being passed along as an input to the core classifier. The performance of the watchdog guarded classifier is evaluated using RMSE values of 4.0, 4.5, 5.0, and 5.5, which can be seen in Figure 4.13.

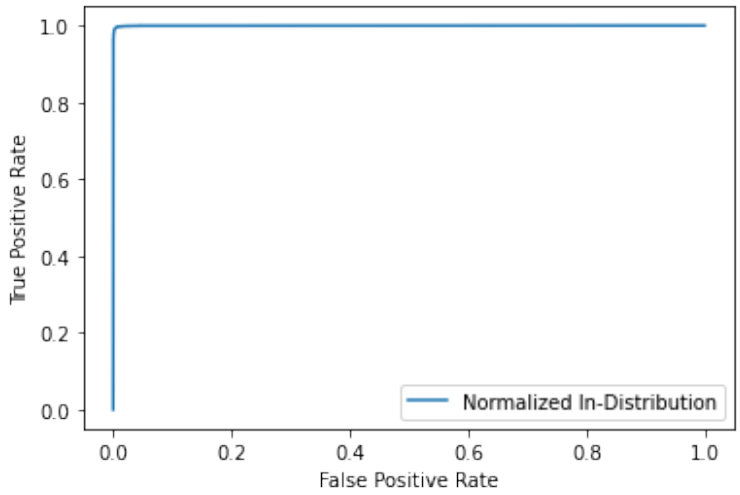

Figure 4.11: The ROC curve for the core classifier's aggregate performance.

Table 4.2: Number of mixed distribution test images permitted at certain threshold values

Threshold Value	In-Distribution	Out-of-Distribution
2.5	6086	5
3.5	9202	61
4.5	9879	433
7.25	10000	3074
12.0	10000	9746

As previously discussed, a threshold value between 4.0 and 5.0 provides strong classifier performance while mitigating the impact of most out-of-distribution data. Table 4.2 shows a breakdown of the number of in- and out-of-distribution images permitted at certain RMSE threshold values. The final selection of the threshold value depends on the application's desired performance, and whether capturing all of the in-distribution data is more valuable than rejecting all of the out-of-distribution data.

4.2 Symbiotic Watchdog Image Classifier Experiment

In this experiment, a symbiotic watchdog is designed to guard and classify images from the MNIST dataset. The symbiotic watchdog will consist of a single input, multiple

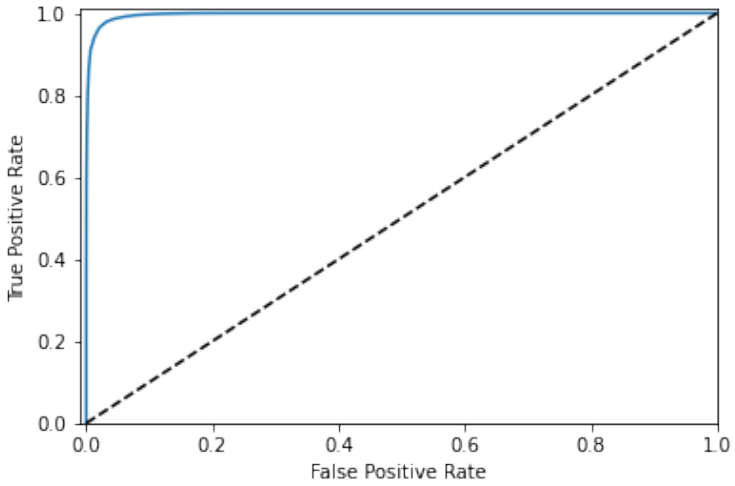

Figure 4.12: The Autoencoder ROC as a function of RMSE threshold value.

output network, combined with a root mean squared error threshold function. In order to evaluate the feasibility of the watchdog, out-of-distribution data from the Fashion MNIST and the KMNIST datasets will be added to the test data.

4.2.1 Network Structure

Unlike the disjoint watchdog, the symbiotic watchdog is comprised of a single input, multi-output neural network. This network combines the generative capabilities of the autoencoder with the classifier. This architecture allows the classifier and generator to share a common latent space, which is also referred to as the waist. Figure 4.14 shows the architecture of the symbiotic neural network. As can be seen, a single 28 by 28 pixel image input produces a classification probability, as well as a reconstructed 28 by 28 pixel image.

4.2.2 Training and Test Data

As with the disjoint experiment, two subsets of data are used for the training and evaluation of the symbiotic watchdog. The training dataset is comprised of 50,000 in-distribution images, with an additional 10,000 images used for validation of the training.

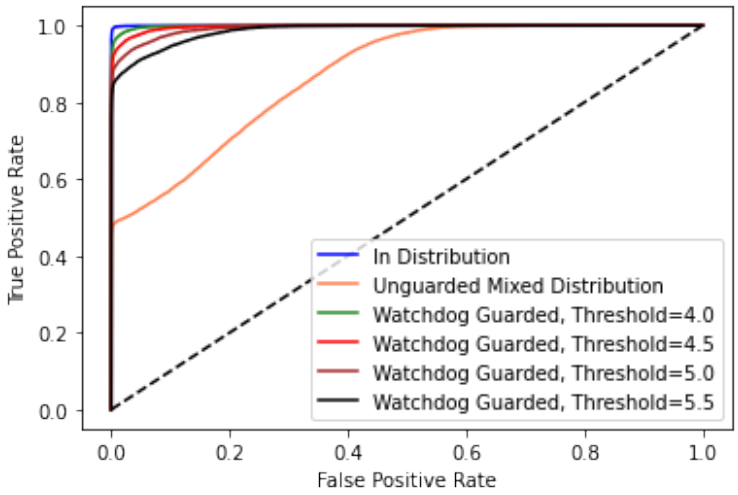

Figure 4.13: The ROC curves of the watchdog guarded classifier with RMSE threshold values of 4.0, 4.5, 5.0, and 5.5.

The mixed distribution test dataset is comprised of 30,000 images, where 10,000 images will be in-distribution and 20,000 will be out-of-distribution images.

4.2.2.1 In-Distribution Data. As with the disjoint watchdog, the symbiotic watchdog experiment will use the MNIST dataset with the same 50,000 training, 10,000 validation, and 10,000 test images. The training, validation, and test datasets contain an equal distribution of digit classes.

4.2.2.2 Out-of-Distribution Data. The out-of-distribution data used in the mixed-distribution test dataset will include 10,000 images from the Fashion MNIST dataset, as well as 10,000 images from the KMNIST dataset. The KMNIST, or Kuzushiji-MNIST [13], is also comprised of 70,000 28 by 28 pixel black and white images of cursive Japanese characters. Since the Fashion MNIST, MNIST, and KMNIST images are all 28 by 28 pixels, this experiment also does not use any additional data augmentation. Examples of the KMNIST images can be seen in Figure 4.15.

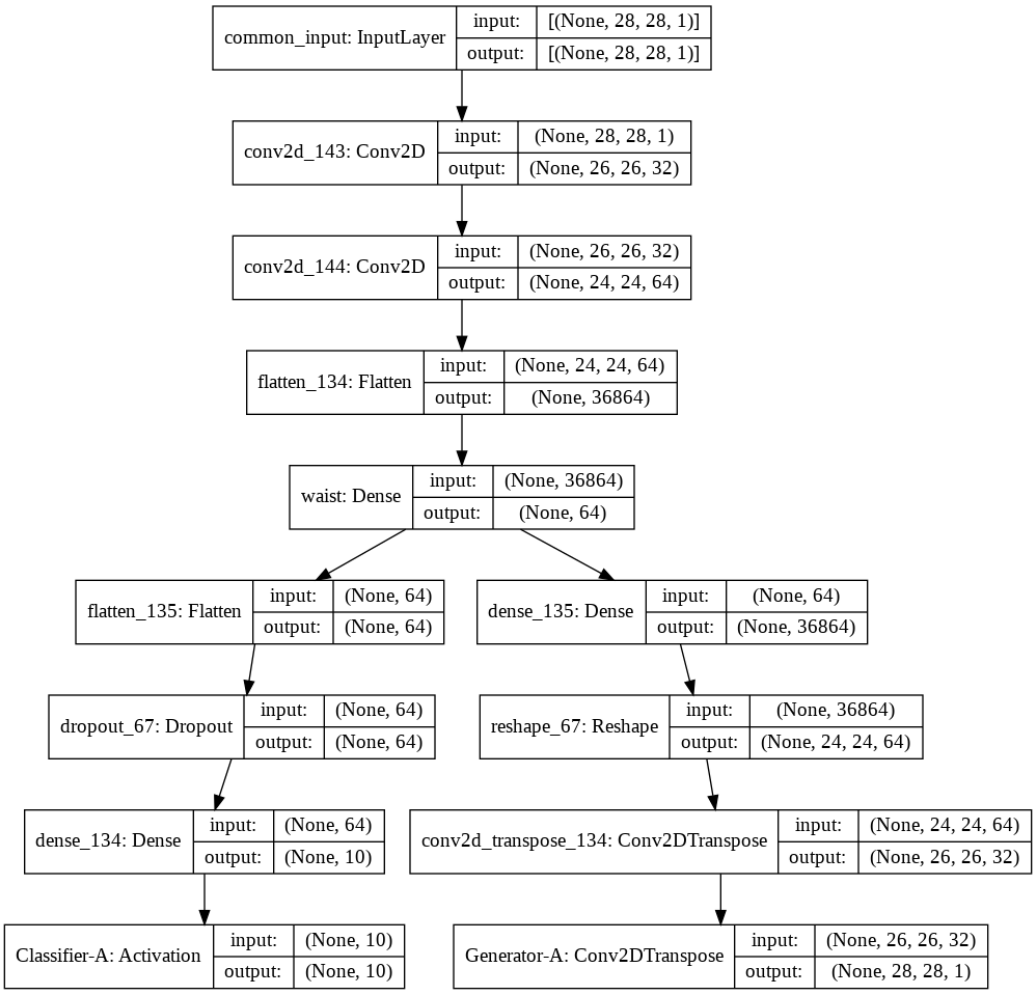

Figure 4.14: The structure of the symbiotic watchdog neural network.

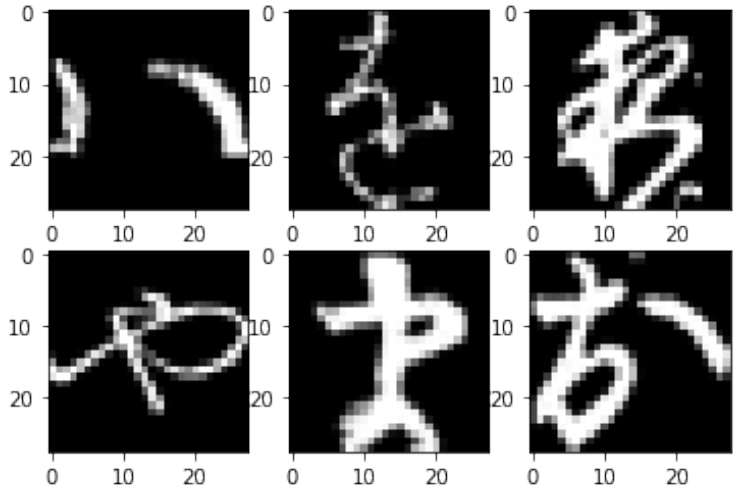

Figure 4.15: Example images from the KMNIST Dataset.

4.2.3 Training the Networks

One of the challenges associated with developing and training a symbiotic neural network is understanding the impact of training bias. To demonstrate how adjusting the weights of the classification and generator outputs of the symbiotic network, nine different networks have been developed with varying weights. Table 4.3 shows the weight percentages for the symbiotic networks. In addition to the nine symbiotic networks, a disjoint watchdog is developed and trained to provide a comparison of watchdog techniques. All of the networks are trained using the 60,000 image in-distribution MNIST dataset with the hyperparameters shown in Table 4.4.

4.2.4 Threshold Evaluation and Selection

Determining the threshold for a symbiotic watchdog takes a similar approach as the disjoint watchdog. The symbiotic network's generator output is compared to the original input image, and the RMSE is calculated. Figure 4.16 shows an example plot of the calculated RMSE values for the entire test dataset, measured on the Symbiotic-E network, which utilizes 100% weights from both the classifier and generator outputs. There is variation in the level of RMSE error with each symbiotic networks, due to how the weights and biases

Table 4.3: Symbiotic Watchdog Training Weight Biases

Network Name	Classifier Weight	Generator Weight
Symbiotic-A	1.00	0.0
Symbiotic-B	1.00	0.25
Symbiotic-C	1.00	0.50
Symbiotic-D	1.00	0.75
Symbiotic-E	1.00	1.00
Symbiotic-F	0.75	1.00
Symbiotic-G	0.50	1.00
Symbiotic-H	0.25	1.00
Symbiotic-I	0.0	1.00

Table 4.4: Symbiotic Watchdog Training Hyperparameters

Hyperparameter	Value
Epochs	10
Batch Size	64
Dropout	0.25

are adjusted during back-propagation. Figure 4.17 shows the number of images permitted by the watchdog, as a function of the RMSE value for several of the symbiotic, as well as the independent watchdogs. From these observations, a RMSE threshold value of between 2.5 and 3.5 is a strong candidate for selection.

4.2.5 Network Evaluations

The symbiotic watchdog is evaluated on several criteria, including training and evaluation times, in-distribution classification accuracy, unguarded mixed distribution classification accuracy, and guarded watchdog classification accuracy.

4.2.5.1 Training and Evaluation Times. One of the proposed advantages to using a symbiotic watchdog is the reduced training and evaluation time. Table 4.5 shows a breakdown of the training and evaluation times for all ten networks used in this experiment. The training and evaluation was performed using the hyperparameters listed above using

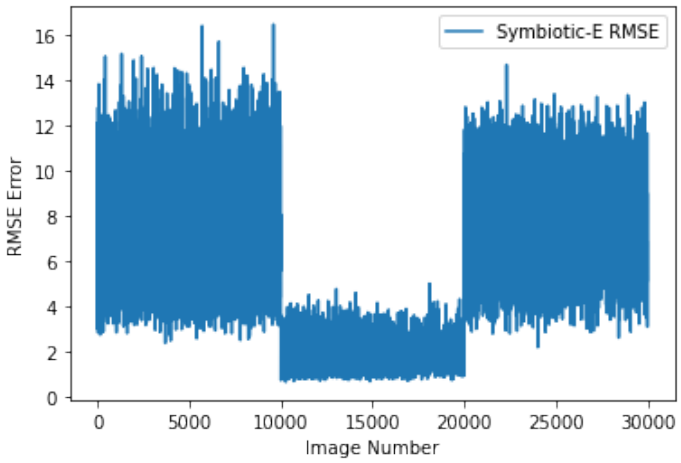

Figure 4.16: Sample plot of the RMSE values of the test dataset, generated by the Symbiotic-E network.

a Google Colab[1] GPU based notebook. While the training and evaluation times do vary depending on resource availability, the symbiotic network has shown an approximate 17% improvement in training time, and an approximate 40% improvement in evaluation time. This improvement can be attributed to two major factors:

(1) Symbiotic watchdogs networks only require one data-loading cycle

(2) Symbiotic watchdog networks have fewer total parameters than disjoint watchdog guarded networks

4.2.5.2 In-Distribution Accuracy. The in-distribution accuracy of the classifiers can be seen in Figure 4.18. The performance for nearly all of the symbiotic classifiers closely matches the performance of the independent classifier. Figure 4.19 shows the upper left quadrant, which further highlights the strong performance of the classifiers. The only observed exception is Symbiotic-I, where the zero percent classifier weight bias does not permit improved accuracy as training progresses.

[1]Google Colab can be found at: https://colab.research.google.com/

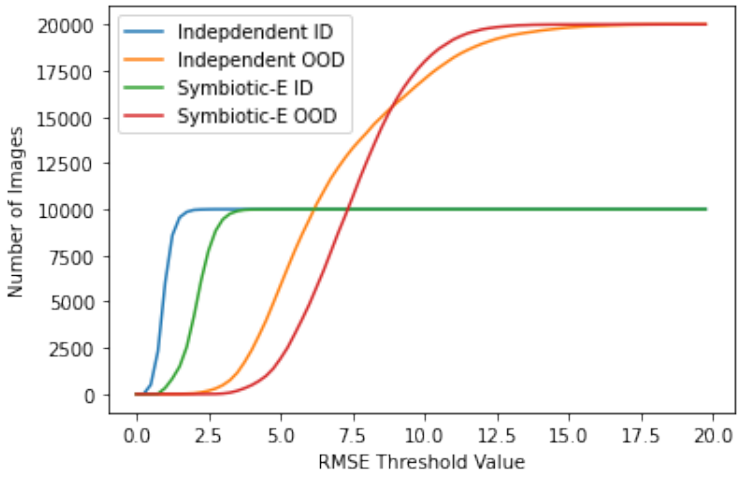

Figure 4.17: Number of images permitted by the watchdog, as a function of RMSE threshold value, for the Independent and Symbiotic-E Watchdogs.

4.2.5.3 Unguarded Mixed-Distribution Accuracy. The unguarded mixed distribution accuracy of the classifiers can be seen in Figure 4.20. The accuracy performance of the symbiotic classifiers closely matches the performance of the independent network, with the exception of Symbiotic-I as expected. The curves also match well with the 2:1 ratio of out-of-distribution to in-distribution data of the mixed distribution test dataset.

Examples of the symbiotic watchdog reconstructions can be found in Figures 4.21, 4.22, and 4.23.

4.2.5.4 Guarded Symbiotic Classifier Performance. The guarded classifier performance is evaluated using RMSE threshold values of 3.5, 4.0, and 4.5. Based on the unguarded mixed distribution results, Symbiotic-I will be removed the remaining analysis. The results of the three threshold evaluations can be seen in Figures 4.24, 4.25, and 4.26.

At first examination, Symbiotic-A shows very weak performance when implementing the threshold function. This can be attributed to the zero percent generator bias during training, which results in poor reconstruction performance. In addition, the independent watchdog performs worse than the symbiotic watchdogs (excluding Symbiotic-A and

Table 4.5: Symbiotic Watchdog Training and Evaluation Times

Network Name	Training Time	Evaluation Time
Symbiotic-A	203s	2.34s
Symbiotic-B	203s	2.65s
Symbiotic-C	202s	2.65s
Symbiotic-D	202s	2.24s
Symbiotic-E	197s	2.23s
Symbiotic-F	203s	2.25s
Symbiotic-G	203s	2.26s
Symbiotic-H	203s	2.24s
Symbiotic-I	204s	2.25s
Independent	244s	3.76s

Symbiotic-I) at similar RMSE thresholds. This difference can be attributed to the lower RMSE values calculated for the autoencoder input regeneration.

4.3 Multi-tier Watchdog Image Classification Experiment

In this experiment, a multi-tier watchdog is designed to guard a traffic sign recognition classification neural network. The multi-tier watchdog used in this experiment consists of an autoencoder first tier and a binary classifier second tier. The autoencoder is trained using identical data and hyperparameters as the core classifier, as seen with a disjoint watchdog experiment. The binary classifier is trained using a generated dataset, and is designed to further differentiate between lower error out-of-distribution images and higher error in-distribution images. In order to evaluate the feasibility of the multi-tier watchdog, out-of-distribution from the CIFAR-10 and CIFAR-100 datasets will be added to the test data.

4.3.1 Network Structures

The multi-tier watchdog experiment consists of three independent (disjoint) neural networks, the autoencoder, the binary classifier, and the core classifier.

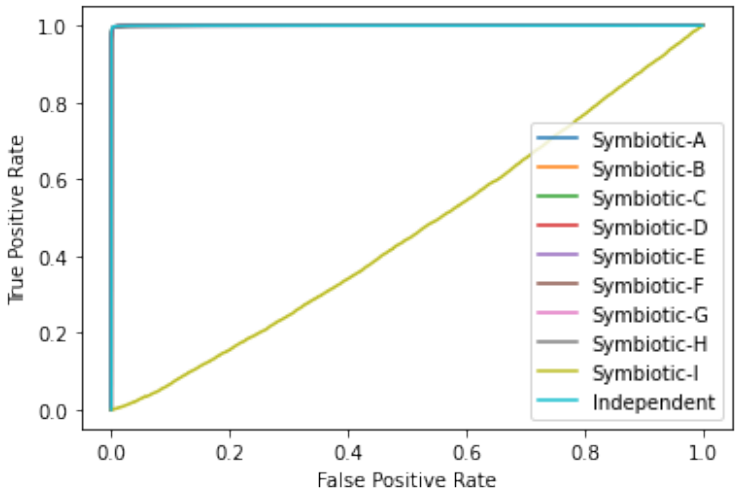

Figure 4.18: The in-distribution test dataset classification ROC curves. The performance of the symbiotic networks A through H, and the independent network are graphically indistinguishable.

4.3.1.1 Autoencoder Structure. The autoencoder structure can be found in Figure 4.27. The encoder portion of the network is comprised of four 2D Convolution layers, with a max pooling layer applied after the second and fourth convolutions. The network is then flattened and reduced to an 800 element latent space dimension. The decoder is comprised of a fully connected layer and a reshape layer, followed by four 2D Transpose layers, with an up sampling layer before the 1st and 3rd transpose layers. This structure allows us to reproduce a standard image size of 32 by 32 pixels across three channels.

4.3.1.2 Binary and Core Classifier Structures. The core and binary classifier structures follow a similar pattern to the encoding portion of the autoencoder, containing four convolution layers, with pooling layers after the second and fourth convolutions. Batch normalization has been implemented after the pooling layers in both of these networks, which helps to reduce over-fitting. The binary and core classifier structures can be found in Figures 4.28 and 4.29. There are two significant differences between the binary and core classifiers:

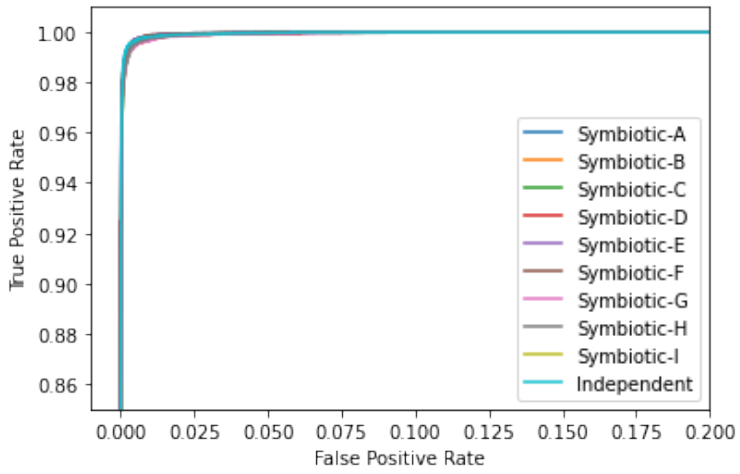

Figure 4.19: The upper left quadrant of the in-distribution test dataset classification ROC curves. The performance of the symbiotic networks and the independent network are graphically indistinguishable.

(1) The core classifier produces a 43 class softmax prediction output, whereas the binary classifier produces a two class softmax prediction output.

(2) The binary classifier includes a 2D cropping layer to remove boundary artifacts.

4.3.2 Training and test Data

The multi-tier watchdog uses two subsets of data for training and evaluation. The training set for the autoencoder and the core classifier will consist of in-distribution images only. The training set for the binary classifier is generated using a modified and independent version of the autoencoder.

4.3.2.1 In-Distribution Data. The German Traffic Sign Recognition Dataset, or GTSRB, is used as the in-distribution data. The GTSRB is a color image dataset comprised of approximately 51,800 images from 43 different classes of road signs [78]. The training dataset for the autoencoder and core classifier will consist of 29,400 images, with another 9,800 images used for training validation. The test dataset will consist of 10,000 images, which have been randomly sampled from the remaining approximately 12,600. To ensure

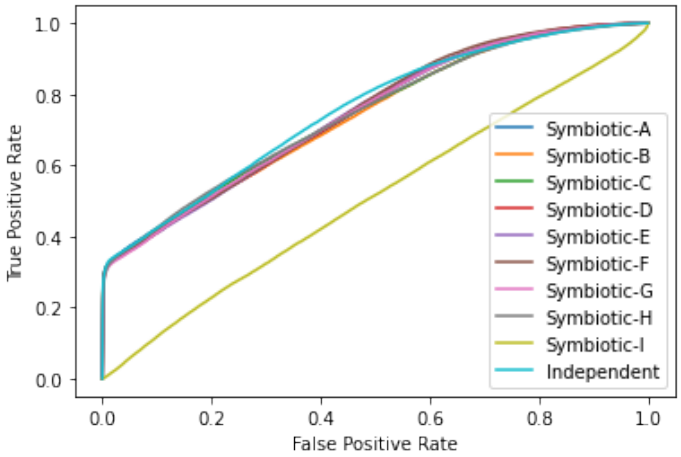

Figure 4.20: The unguarded mixed-distribution classification ROC curves.

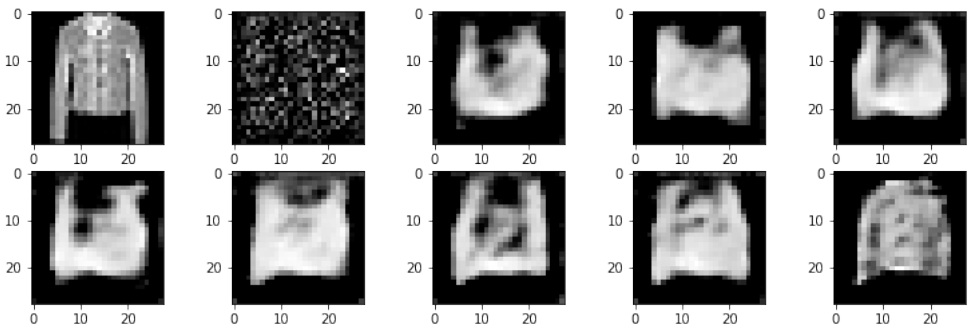

Figure 4.21: Original image and symbiotic reconstructions of a Fashion MNIST jacket.

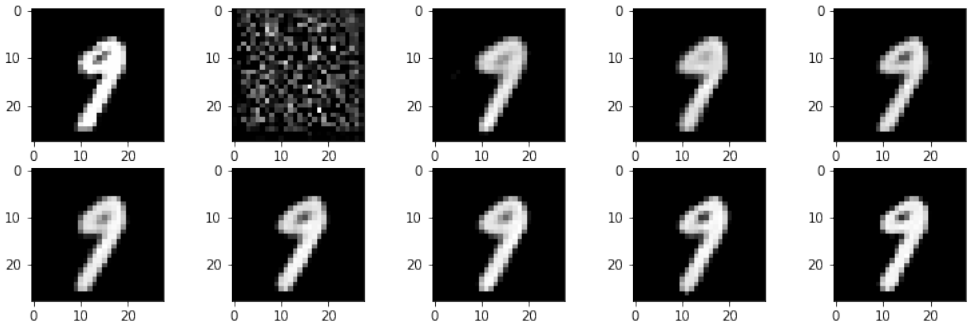

Figure 4.22: Original image and symbiotic reconstructions of a MNIST nine digit.

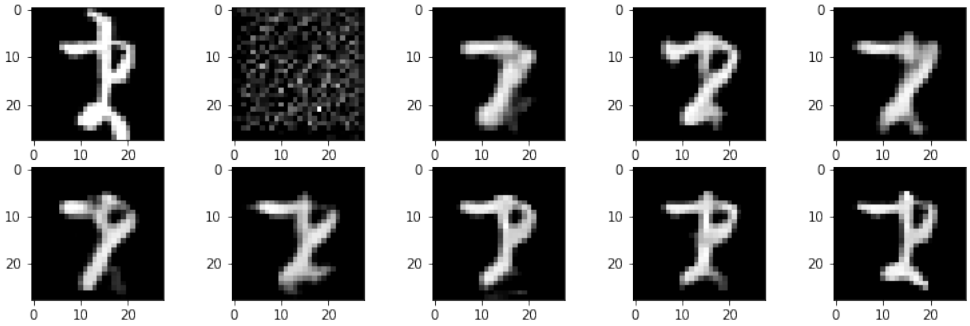

Figure 4.23: Original image and symbiotic reconstructions of a KMNIST Su character.

repeatability, the random sampling uses a seed value which ensures identical performance across system and test iterations. An example of the in-distribution traffic signs can be seen in Figure 4.30

4.3.2.2 Out-of-Distribution Data. The out-of-distribution data used in the mixed-distribution test dataset includes 10,000 images from both the CIFAR-10 and CIFAR-100 datasets[2]. Both the CIFAR-10 and CIFAR-100 consist of 60,000 color images of various animals and objects. Example images from the datasets can be seen in Figures 4.31 and 4.32.

4.3.3 Training the Networks

The training process for a multi-tier watchdog will vary, depending on the final architecture and development process. For the watchdog proposed in this experiment, a three phase training process is implemented. The hyperparameters for training the networks can be found in Tables 4.6 and 4.7.

4.3.3.1 First Phase. The first training phase is responsible for the training of both the autoencoder and core classifier. A data augmentation pipeline is developed to artificially increase the amount of training data through the use of various image manipulation

[2]More information about these datasets can be found at https://www.cs.toronto.edu/ kriz/cifar.html

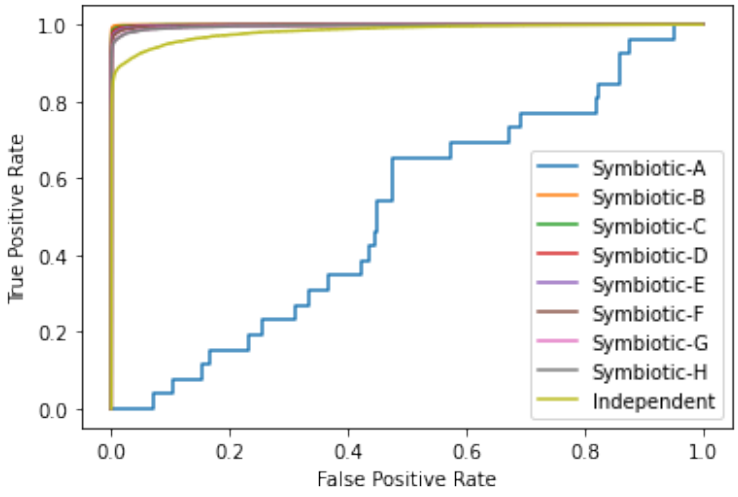

Figure 4.24: ROC curves for the Symbiotic and Independent Watchdogs, using an RMSE threshold of 3.5.

Table 4.6: Multi-tier Watchdog Training Hyperparameters for the Autoencoder and Classifier

Hyperparameter	Value
Epochs	25
Batch Size	128
Dropout	0.35

techniques. This augmentation allows both the autoencoder and the core classifier to be trained for longer while mitigating the risk of over-fitting associated with small training sample sizes. The parameters used for the image augmentation can be found in Table 4.8.

4.3.3.2 Second Phase. The second training phase is responsible for the training and execution of an image generator network to build the dataset used to train the binary classifier. In this phase, a generative network is developed using the same architecture as the autoencoder [18]. The previously trained autoencoder and error function are used to establish the training criteria for the generative network. The generative network is

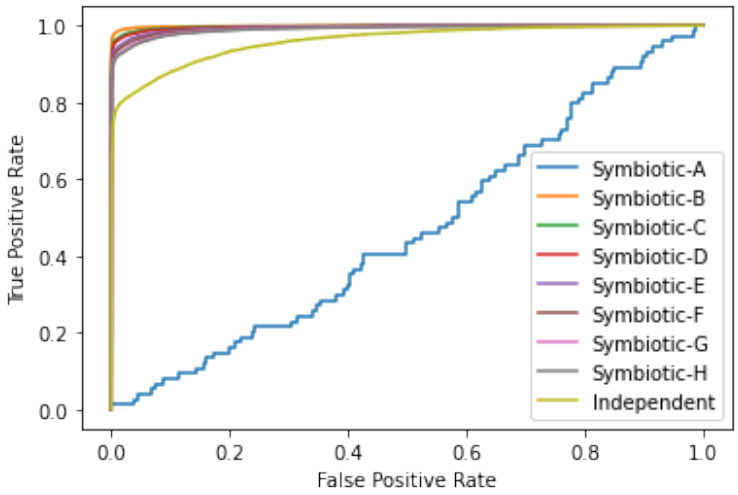

Figure 4.25: ROC curves for the Symbiotic and Independent Watchdogs, using an RMSE threshold of 4.0.

Table 4.7: Multi-tier Watchdog Training Hyperparameters for the binary classifier

Hyperparameter	Value
Epochs	5
Batch Size	64
Dropout	0.35

then trained to produce images within a certain percentage of the error determined by the threshold. An example of the generated images can be seen in Figure 4.33.

4.3.3.3 Third Phase. The third phase of training uses the generator network to produce a subset of the data used to train the binary classifier. The generator produces images that are considered to be out-of-distribution, which are then combined with the original training images to produce a two-class dataset. The binary classifier is then trained on the two-class dataset.

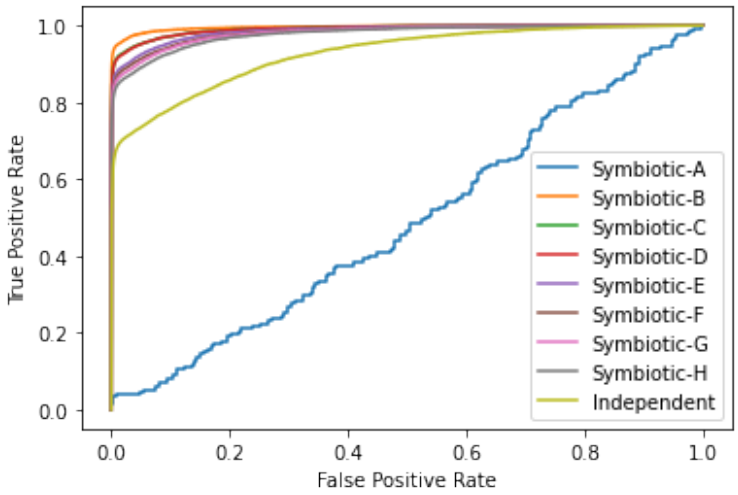

Figure 4.26: ROC curves for the Symbiotic and Independent Watchdogs, using an RMSE threshold of 4.5.

Table 4.8: Training Image Augmentation Parameters

Augmentation Type	Augmentation Value	Description
Rotation Range	20	Allows for random rotations of up to 20 degrees
Width Shift Range	.15	Allows for up to 15% horizontal shift
Height Shift Range	.15	Allows for up to 15% vertical shift
Zoom Range	.15	Allows for up to 15% image zoom
Horizontal Flip	True	Enables random horizontal flipping
Vertical Flip	True	Enables random vertical flipping
Rescale	1/255.0	Scales the pixel values to [0,1]

4.3.4 Threshold Evaluation and Selection

The threshold function for the multi-tier watchdog is a structural similarity measurement [91, 92]. Initial calculations were made using an RMSE threshold, however, after further evaluation, the performance of the RMSE system was not sufficient. Transitioning the threshold function to the structural similarity measurement allowed for more reliable image difference measurements across multiple color channels.

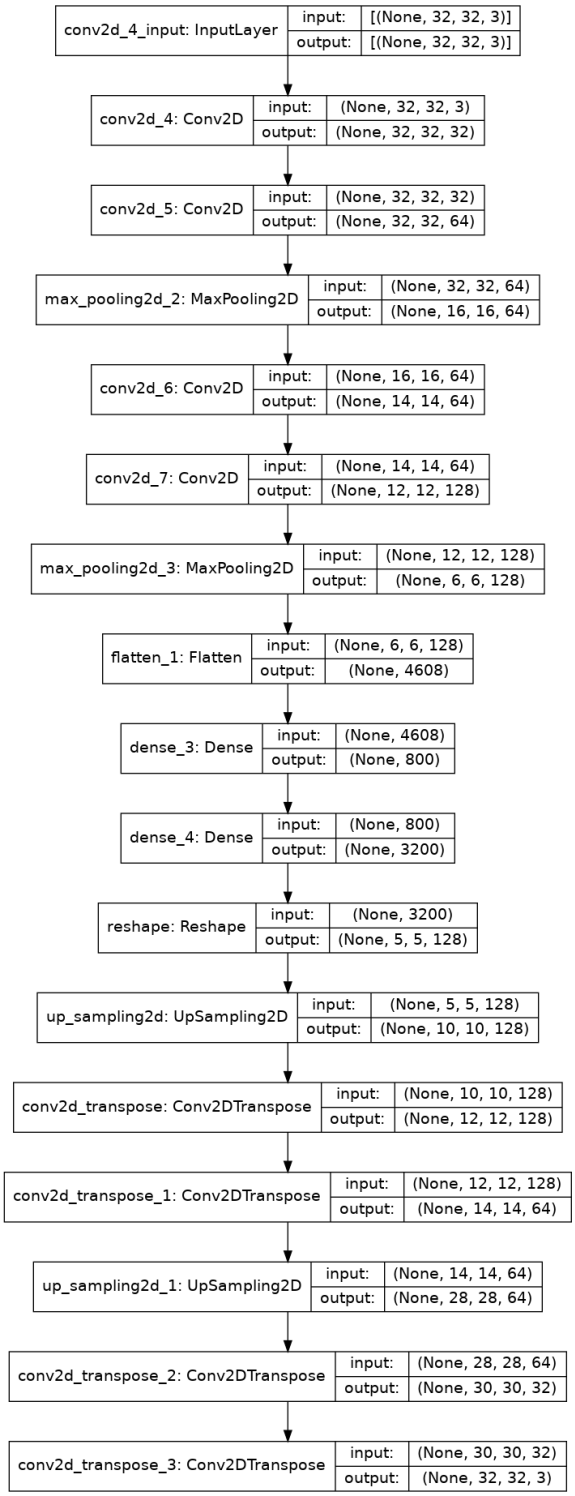

Figure 4.27: Network structure of the multi-tier watchdog's autoencoder.

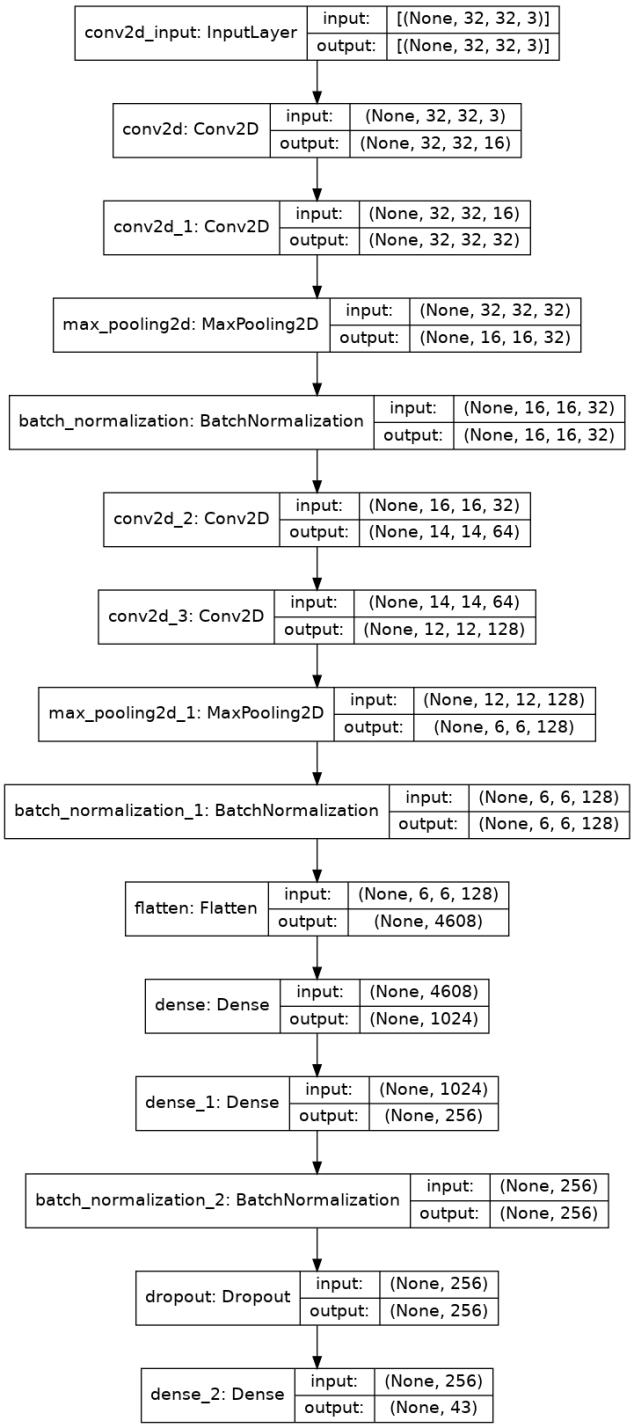

Figure 4.28: Network structure of the multi-tier watchdog's core classifier.

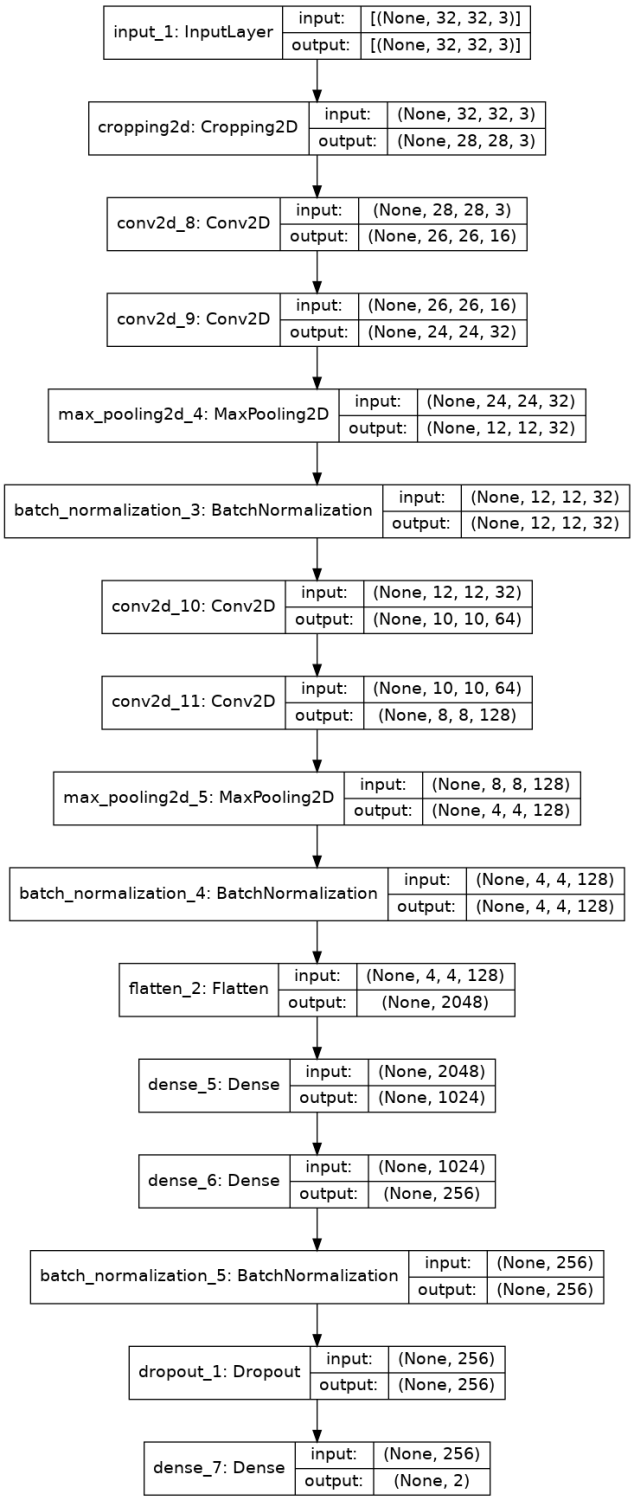

Figure 4.29: Network structure of the multi-tier watchdog's binary classifier.

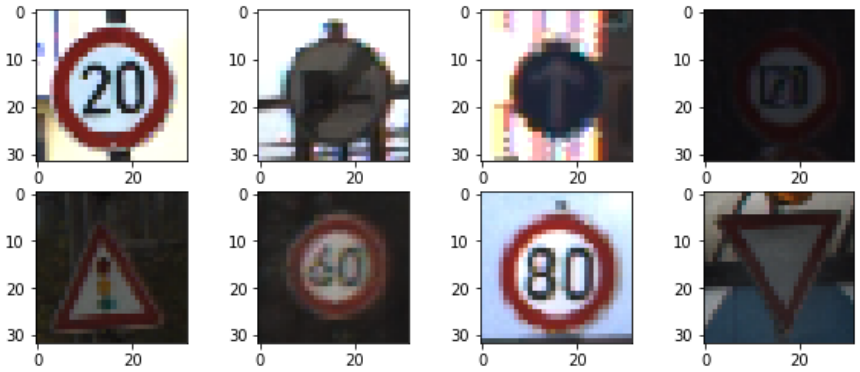

Figure 4.30: Example images from the GTSRB dataset.

Figure 4.31: Example images from the CIFAR-10 dataset.

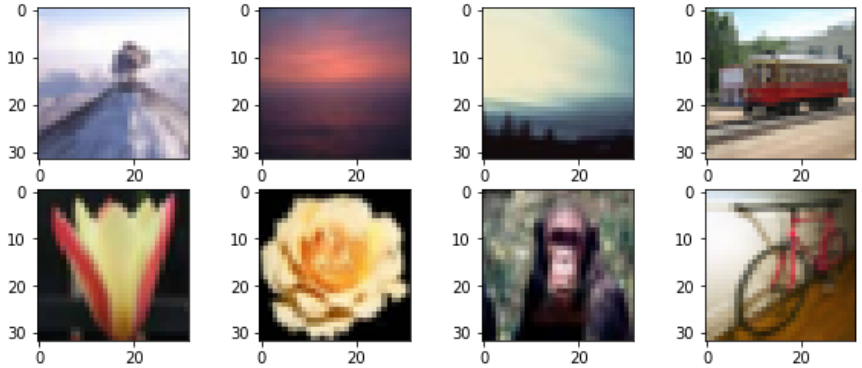

Figure 4.32: Example images from the CIFAR-100 dataset.

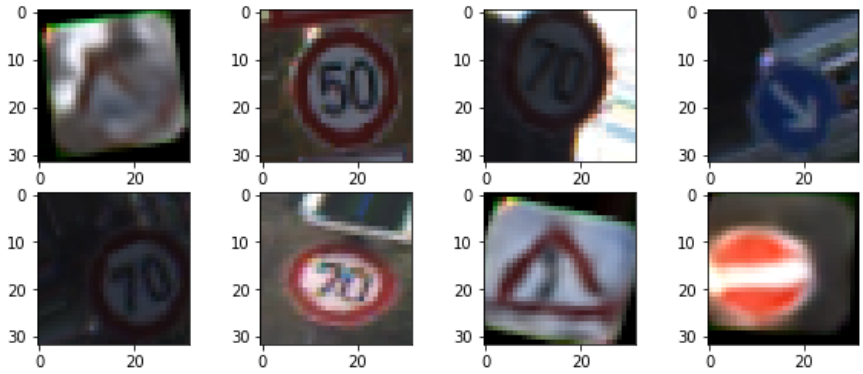

Figure 4.33: Example images generated for the binary classification training dataset.

4.3.5 Network Evaluations

The multi-tier watchdog performance is evaluated on several criteria. The core classifier is evaluated on the in-distribution classification accuracy. The autoencoder and threshold function are evaluated on the ability to differentiate between in-distribution and out-of-distribution data. The binary classifier is evaluated on the ability to differentiate between near-threshold generated images and the original in-distribution images. The watchdog is evaluated on the combined autoencoder, binary classifier and core classifier performance.

4.3.5.1 Autoencoder and Threshold Evaluation. The autoencoder and difference score are evaluated using the mixed distribution dataset. Figure 4.34 shows the ROC curve of the autoencoder, as a function of the structural similarity score. Figures 4.35 and 4.36 show examples of original, reconstructed, and structural similarity difference images. Table 4.9 shows the structural similarity score of both example images, as well as the RMSE value as a point of reference.

4.3.5.2 Core Classifier Performance. The core classifier's performance is evaluated on the in-distribution GTSRB images. The core classifier scores a 95.7% accuracy across all 43 classes of the dataset. The normalized ROC curve can be seen in Figure 4.37.

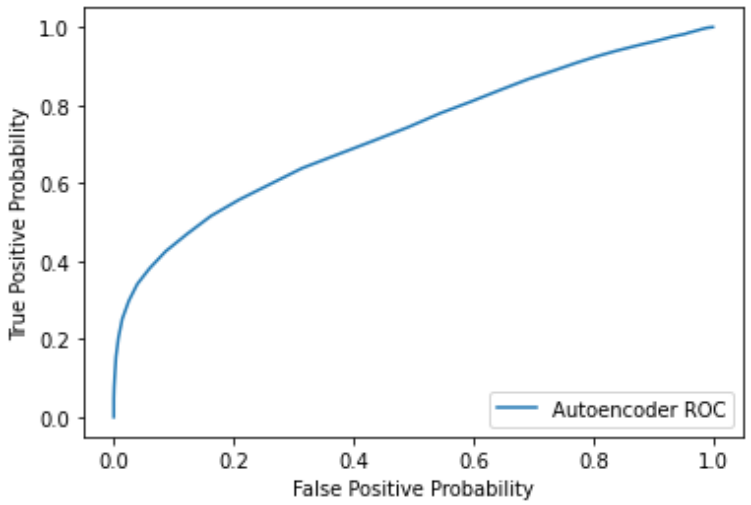

Figure 4.34: The ROC curve for the multi-tier watchdog's autoencoder, evaluated on the mixed-distribution test dataset.

Table 4.9: Structural Similarity and RMSE values for Figures 4.35 and 4.36

Image	SSIM Difference Score	Calculated RMSE
Direction of Travel Sign	0.8689	3.8384
CIFAR Boat Image	0.7081	5.8583

4.3.5.3 Binary Classifier Performance. The binary classifier is evaluated on a test subset of the generated near threshold and the original in-distribution. The binary classifier shows a strong ability to differentiate between the near-threshold out-of-distribution and in-distribution images.

4.3.5.4 Unguarded Mixed Distribution Performance. The unguarded mixed distribution accuracy of the system can be seen in Figure 4.39. As expected, the performance of the core classifier rapidly degrades with the introduction of out-of-distribution data. The curve matches the 2:1 ratio of out-of-distribution to in-distribution data of the mixed distribution test dataset.

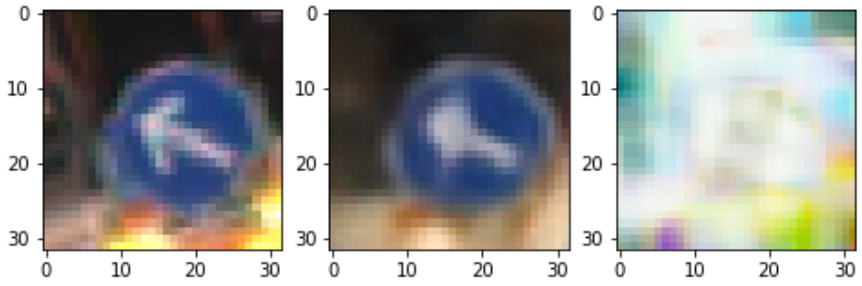

Figure 4.35: The original, autoencoder reconstructed, and difference images for a sample sign image.

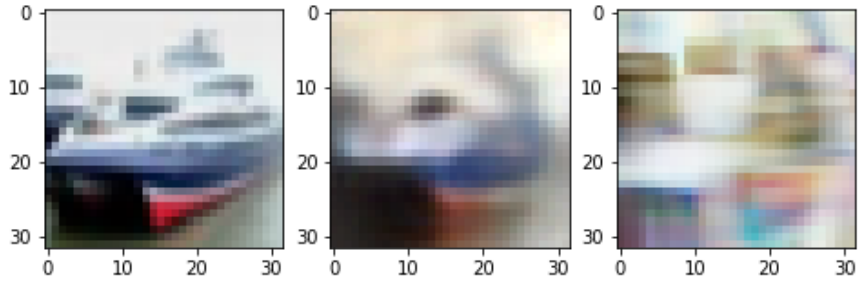

Figure 4.36: The original, autoencoder reconstructed, and difference images for a sample CIFAR image.

4.3.5.5 Guarded Mixed Distribution Performance. The guarded classifier performance is evaluated using a structural similarity score threshold of 0.85, 0.87, and 0.90. The performance of the core classifier improves as the structural similarity score threshold increases, as seen in Figure 4.40. As expected, the performance of the system increases as the watchdog threshold becomes more stringent.

4.4 Disjoint Watchdog Signal Identification and Classification Experiment

In this experiment, a disjoint watchdog is designed in coordination with colleagues at Virginia Tech [17] to guard a communications signal classification neural network. These networks are designed to identify and classify various communication and radar based waveforms. The disjoint watchdog consists of an autoencoder used for power spectral density reconstruction, coupled with a root mean squared error threshold function. In order to

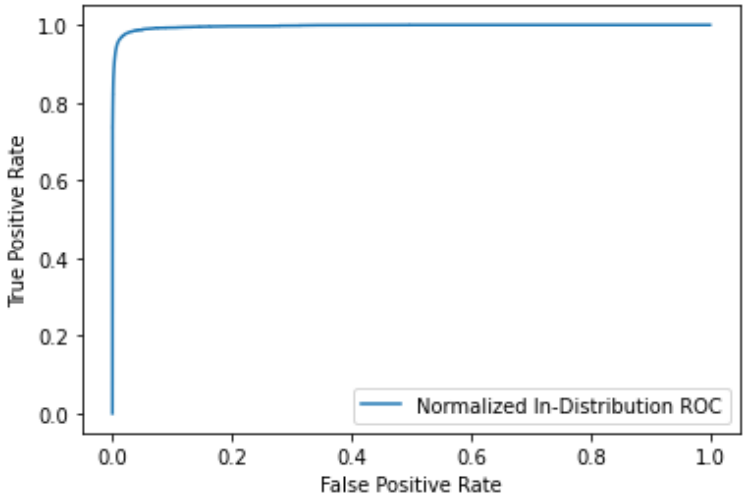

Figure 4.37: The ROC curve for the core classifier, evaluated on the in-distribution test dataset.

evaluate the feasibility of the signal detection watchdog, out-of-distribution data is generated using MATLAB's Time-Frequency toolbox.

4.4.1 Network Structures

Unlike the previous experiments, the network architectures of the signal classifier and watchdog differ. The primary difference in network architecture is attributed to the different input data types. The classifier is designed to work with raw waveform data, whereas the autoencoder is designed to work with power spectral density data.

4.4.1.1 Classifier Structure. The classification network is a deep multi-layer perceptron comprised of several fully connected, or Dense layers used in combination with dropout layers. The network is designed to work with a 4,096 element Fast Fourier Transform signal input to produce one of four waveform classifications. The structure of the signal classification network can be seen in Figure 4.41.

4.4.1.2 Autoencoder Structure. The autoencoder network is comprised two subnetworks, an encoder and decoder. The encoder is comprised of three 2D convolutional

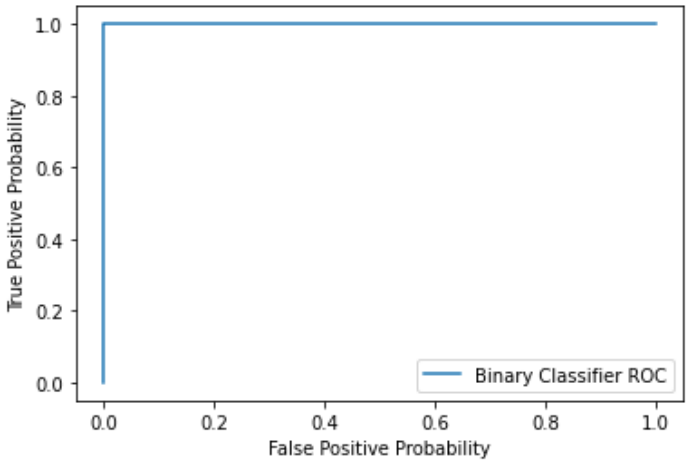

Figure 4.38: The ROC curve for the binary classifier, evaluated on the generated near-threshold dataset.

layers, combined with a dense layer to produce a 16 element latent space representation. The decoder is comprised of a dense layer which feeds in to four 2D transpose layers. The autoencoder is designed to work with a power spectral density analysis of the signal. The structure of the autoencoder can be seen in Figure 4.42.

4.4.2 Training and Evaluation Data

Two sets of data are used for the evaluation of the watchdog and signal classifier. The training dataset is comprised of 40,000 in-distribution waveforms, and the test dataset is comprised of 12,800 in-distribution waveforms, combined with 12,800 out-of-distribution waveforms.

4.4.2.1 In-Distribution Data. The in-distribution dataset is comprised of waveform signals which fall in to one of the following four categories:

(1) Single Carrier

(2) Single-Carrier Frequency Division Multiple Access

(3) Orthogonal Frequency Division Multiplexing

(4) Linear Frequency Modulation

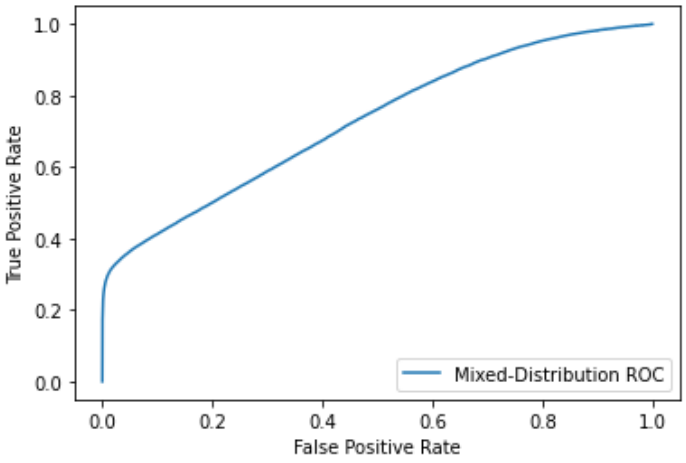

Figure 4.39: The ROC curve for the unguarded core classifier, evaluated on the mixed-distribution test dataset.

These signals categories are then further divided into 8 equal sets of 1,250 waveforms training and 400 test waveforms respectively, using the following modulation techniques:

(1) BPSK

(2) QPSK

(3) 16-PSK

(4) 64-PSK

(5) 4-QAM

(6) 16-QAM

(7) 64-QAM

(8) 256-QAM

4.4.2.2 Out-of-Distribution Data. The out-of-distribution test data is comprised of waveform signals which fall in to one of the following four categories:

(1) FM Radio Signals

(2) AM Radio Signals

(3) Bluetooth Low Energy 5.0 (BLE)

(4) White Noise

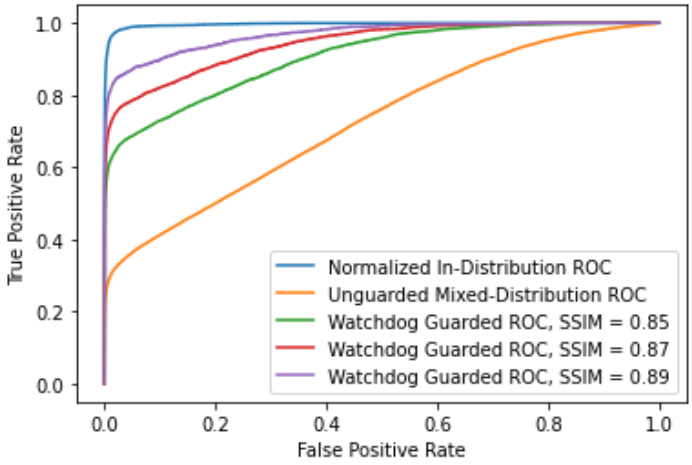

Figure 4.40: The ROC curve for the multi-tier watchdog guarded core classifier with structural similarity scores of 0.85, 0.87, and 0.89.

As with the in-distribution dataset, these four categories are further divided into 8 equal sets of 400 test waveforms using the previously mentioned modulation techniques.

4.4.3 Training the Networks

The training process for the classification network is performed by first applying various signal impairments to the waveforms, including phase offset, frequency offset and various signal fading techniques [17]. Once these impairments have been applied, the classifier is trained using a batch size of 128 waveforms, trained over 50 epochs.

The training process for the autoencoder network is performed similarly, with the same signal impairments being applied to the training waveforms, prior to converting to the power spectral density of the signal. The power signal density of each signal is then used to train the autoencoder, using the same batch size and training epoch parameters as the classification network.

4.4.4 Threshold Evaluation and Selection

The selection of a threshold function in this experiment varies from the previous experiments, which relied on a signal value threshold for distribution determination. Based

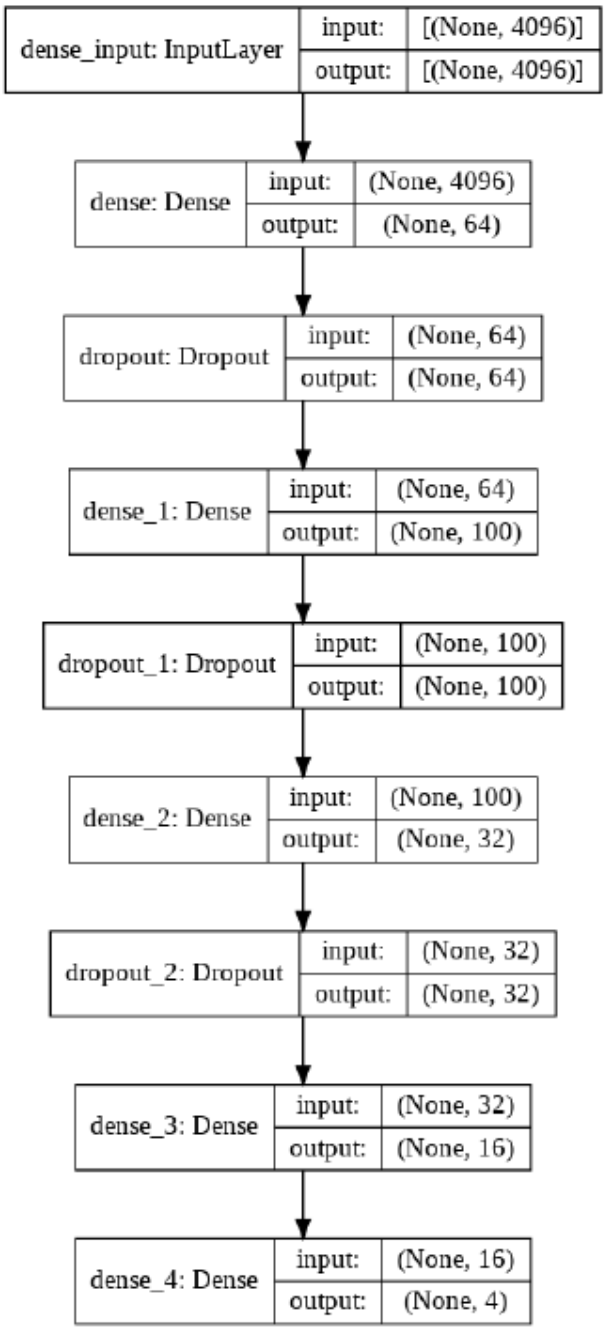

Figure 4.41: The signal classification network structure.

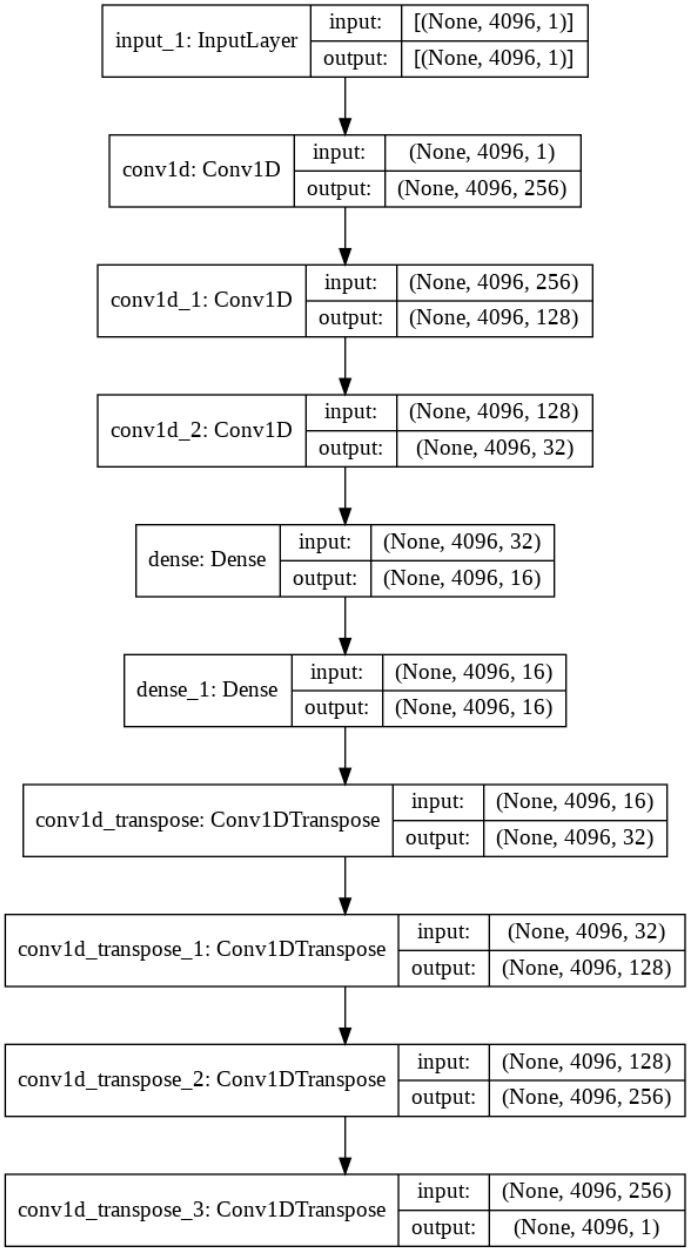

Figure 4.42: The power spectral density autoencoder network structure.

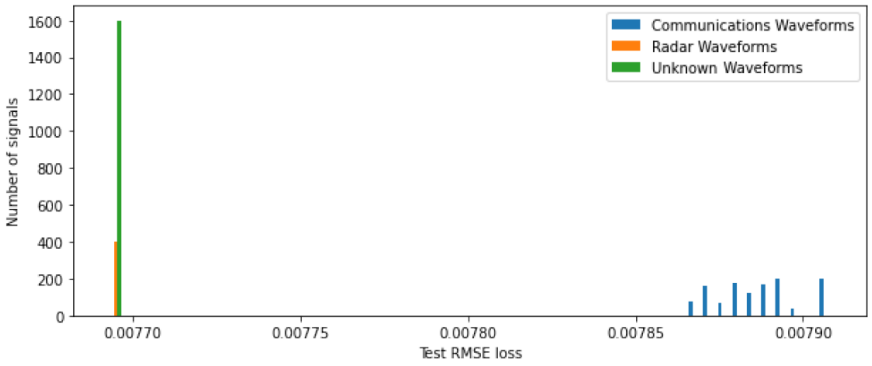

Figure 4.43: An example plot of the RMSE values of communication, radar, and unknown signals.

on the experiments results, an RMSE range is selected, as opposed to a single value. An example of the RMSE ranges for various signal types can be seen in Figure 4.43. The threshold range is selected based on the minimum and maximum RMSE values calculated on the in-distribution dataset.

4.4.5 Signal Classification Performance Evaluation

4.4.5.1 Core Classifier Performance. The performance of the signal classification network is evaluated based the ability to classify signals which are subject the previously mentioned perturbations. The performance of the classifier on the in-distribution dataset can be seen in Figure 4.44.

4.4.5.2 Watchdog Performance. The addition of the watchdog for improves the mixed-distribution when using higher fidelity FFT analysis. Figure 4.45 shows the aggregated performance of the watchdog guarded classification network when using a 16,384 element FFT prior to autoencoder regeneration.

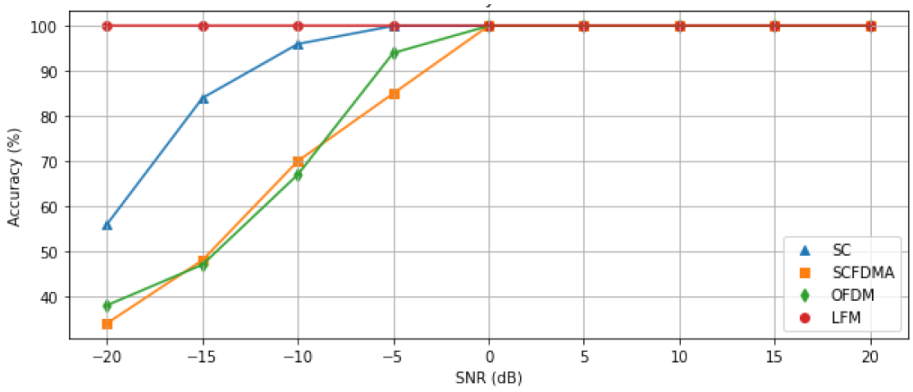

Figure 4.44: The classifier performance of in-distribution data, as a function of SNR.

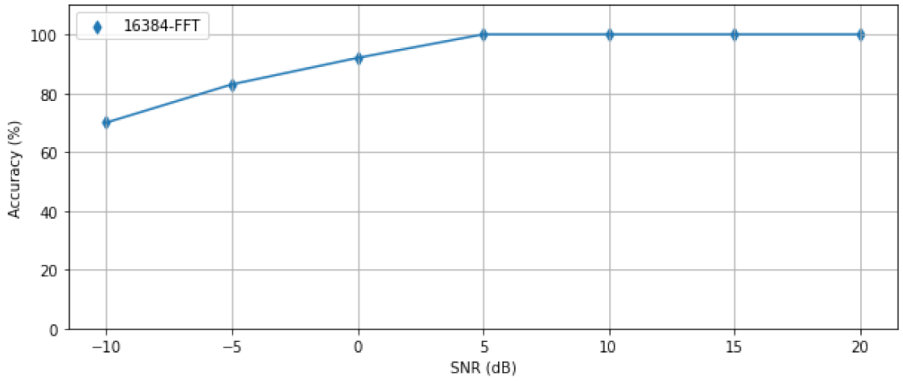

Figure 4.45: The watchdog guarded mixed-distribution performance of the classifier, as a function of SNR.

CHAPTER FIVE

Discussion and Conclusion

The design, development, and evaluation of four neural network watchdogs are presented. The neural network watchdog allows for the on-line detection of out-of-distribution data in a machine learning system without the need for additional preprocessing. The data presented demonstrates the proof of concept on image and waveform data, however, the technique can be applied other data types as well.

The value of out-of-distribution data detection and filtering is evident when the validity of input data cannot be determined. The neural network watchdog is deployed to allow for in-situ distribution detection, by being built into the input pipeline or designed directly in to the neural network. The techniques can be specialized to accommodate desired system performance.

The three primary categories of neural network watchdogs, disjoint, symbiotic, and multi-tier have all demonstrated the ability to detect and remove out-of-distribution data from classification networks. The degree of performance improvement is related to several factors, including complexity of the data, availability of training and evaluation data, and the complexity of the core and watchdog networks. The three categories have advantages and disadvantages compared to one another.

The disjoint watchdog is the most simple architecture to develop and deploy. It's independent architecture allows for easy deployment in the data pipeline with minimal execution impact. This fact allows the disjoint watchdog to be developed for an existing system, including black box networks, if access to the training data is available.

The symbiotic watchdog is more constrained in terms of development and deployment when compared to the disjoint architecture. Since the symbiotic network is designed

directly into the core network, a symbiotic watchdog cannot be implemented on an existing system without first retraining the network. With shared latent space representations, however, the symbiotic neural network has demonstrated higher reliability detection with reduced computational costs.

The multi-tier watchdog approach allows for the most design flexibility. The composition of multiple watchdog layers allows for the fine tuning for optimal performance, however, the costs of development and deployment increase as tiers become more complex. The deployment of a multi-tier watchdog is similar to that of a simple disjoint watchdog, allowing the system to be applied to an existing neural network.

The selection of a difference function and the threshold value will also vary with the type and complexity of data. For example, an RMSE threshold may be appropriate for single channel data types, where as more complex analysis may be required for multi-channel data. A tight threshold is appropriate for a system where classification on in-distribution only is critically important. On the other hand, a more relaxed threshold is appropriate when it is desired to capture all of the in-distribution data at the cost of permitting some out-of-distribution data.

5.1 Future Work

The neural network watchdog concept can be further improved for more reliable out-of-distribution detection, using a number of different methods. The generative components can make use of deeper neural network architectures to produce more precise latent space representations. The training process for the generative components can also be expanded to include various augmentation techniques to greatly increase the amount of training data. The generative components may also be improved by implementing additional input pre-processing, which is common practice in the deployment of neural network based systems.

Additional watchdog techniques can also be developed to accommodate different data input types, such as text, audio, video, or sensor data. These data types can take

advantage of different neural network types, such as recurrent neural networks or Long-Short Term Memory neural networks, which allow for more advanced input representations to be learned.

For successful commercialization, a neural network watchdog would need to have minimal impact to the training and execution time of a machine learning system, while providing ideal distribution detection. Research aimed at reducing the computational complexity of the watchdog, as well as pipeline optimization will allow for reduced development and operation costs of machine learning systems which implement a neural network watchdog.

APPENDIX

APPENDIX

Python Code

```
import tensorflow as tf
import tensorflow.keras.layers as layers


def define_simple_model(shape, filters=8, kernel_size=(3, 3),
    pool_size=(2, 2), dropout=.25, num_classes=1):
"""
This function creates a simple Convolutional Neural Network for
     image classification

:param shape: Input Image Shape
:param filters: The base number of filters to be used in the
    convolution layers
:param kernel_size: kernel size of the convolutions
:param pool_size: MaxPooling pool size
:param dropout: Dropout settings for training
:param num_classes: Number of Classes in the dataset
:return: Returns a CNN model based on the input parameters
"""

model = tf.keras.models.Sequential()
model.add(layers.Conv2D(input_shape=shape, filters=filters,
    kernel_size=kernel_size,
activation=tf.keras.activations.relu))
```

```
    model.add(layers.MaxPool2D(pool_size=pool_size))
    model.add(layers.Conv2D(filters=filters * 2, kernel_size=
        kernel_size,
    activation=tf.keras.activations.relu))
    model.add(layers.MaxPool2D(pool_size=pool_size))
    model.add(layers.Conv2D(filters=filters * 4, kernel_size=
        kernel_size,
    activation=tf.keras.activations.relu))
    model.add(layers.MaxPool2D(pool_size=pool_size))
    model.add(layers.Flatten())
    model.add(layers.Dropout(dropout))
    model.add(layers.Dense(num_classes))
    model.add(layers.Activation(tf.keras.activations.softmax))

    return model
```

Listing A.1: An example function used to create a convolutional neural network image classifier

```
    # Network design and training hyperparameters
    input_shape = img_shape
    batch_size = 64
    kernel_size = (3,3)
    pool_size = (2,2)
    latent_dim = 64
    filters = 64
    dropout = .25
    Epochs = 10

    # Define The CNN Model
```

```
cnn = tf.keras.models.Sequential()

cnn.add(tf.keras.layers.Conv2D(filters = filters, kernel_size =
    kernel_size, activation = 'relu',  input_shape =
   input_shape))
cnn.add(tf.keras.layers.MaxPooling2D(pool_size))

cnn.add(tf.keras.layers.Conv2D(filters = filters,
kernel_size = kernel_size, activation = 'relu'))
cnn.add(tf.keras.layers.MaxPooling2D(pool_size))

cnn.add(tf.keras.layers.Conv2D(filters = filters,
kernel_size = kernel_size, activation = 'relu'))
cnn.add(tf.keras.layers.MaxPooling2D(pool_size))

cnn.add(tf.keras.layers.Flatten())
cnn.add(tf.keras.layers.Dropout(dropout))
cnn.add(tf.keras.layers.Dense(num_labels))
cnn.add(tf.keras.layers.Activation('softmax'))

cnn.compile(loss = 'categorical_crossentropy', optimizer = '
   adam',metrics = ['accuracy'])

# Define the Autoencoder Model
inputs = tf.keras.layers.Input(shape = input_shape, name = '
   encoder_input')
x = tf.keras.layers.Conv2D(filters = filters,
kernel_size= kernel_size, activation = 'relu')(inputs)
```

```
x = tf.keras.layers.Conv2D(filters = filters*2,
kernel_size= kernel_size, activation = 'relu')(x)
shape = tf.keras.backend.int_shape(x)
x = tf.keras.layers.Flatten()(x)

latent = tf.keras.layers.Dense(latent_dim, name = '
   latent_vector') (x)

encoder = tf.keras.Model(inputs, latent, name = 'encoder')

latent = tf.keras.layers.Input(shape = (latent_dim, ),
name = 'decoder_input')

x = tf.keras.layers.Dense(shape[1] * shape[2] * shape[3])(
   latent)
x = tf.keras.layers.Reshape((shape[1], shape[2], shape[3]))(x)
x = tf.keras.layers.Conv2DTranspose(filters = filters*2,
kernel_size = kernel_size, activation = 'relu')(x)

x = tf.keras.layers.Conv2DTranspose(filters = filters,
kernel_size = kernel_size, activation = 'relu')(x)

outputs = tf.keras.layers.Conv2DTranspose(filters = 1,
kernel_size = kernel_size,
activation = 'sigmoid', padding = 'same',
name = 'decoder_output')(x)

decoder = tf.keras.Model(latent, outputs, name = 'decoder')
```

```
autoencoder = tf.keras.Model(inputs, decoder(encoder(inputs)),
name = 'autoencoder')
autoencoder.compile(loss='mse', optimizer = 'adam')
```

Listing A.2: The code used to define the models of a disjoint neural network watchdog

```
import tensorflow as tf

def make_symbiotic_model(input_shape, filters, kernel_size,
latent_dim, dropout, num_labels):
"""
:param input_shape: Input image shape
:param filters: The base number of filters used in the
   convolutional layers
:param kernel_size: kernel size of the convolutions
:param latent_dim: dimension of latent space/waist layer
:param dropout: dropout amount for classification sub-layers
:param num_labels: number of labels for classifier sub-layers
:return: Returns a symbiotic neural network model based on
   input parameters
"""

inputs = tf.keras.layers.Input(shape=input_shape, name='
   common_input')
x = tf.keras.layers.Conv2D(filters=filters,
kernel_size=kernel_size, activation='relu')(inputs)

x = tf.keras.layers.Conv2D(filters=filters * 2,
kernel_size=kernel_size, activation='relu')(x)
```

```
shape = tf.keras.backend.int_shape(x)
x = tf.keras.layers.Flatten()(x)

core = tf.keras.layers.Dense(latent_dim, name='waist')(x)

# Classifier Network sub-layers
flat = (tf.keras.layers.Flatten())(core)
drop = (tf.keras.layers.Dropout(dropout))(flat)
dense = (tf.keras.layers.Dense(num_labels))(drop)
class_out = (tf.keras.layers.Activation('softmax'))(dense)

# Generator Network sub-layers
x = tf.keras.layers.Dense(shape[1] * shape[2] * shape[3])(core)
x = tf.keras.layers.Reshape((shape[1], shape[2], shape[3]))(x)
x = tf.keras.layers.Conv2DTranspose(filters=filters,
kernel_size=kernel_size, activation='relu')(x)
gen_out = tf.keras.layers.Conv2DTranspose(filters=1,
kernel_size=kernel_size, activation='relu')(x)

symbiotic_model = tf.keras.Model(inputs=inputs,
outputs=[class_out, gen_out], )

return symbiotic_model
```

Listing A.3: A function used to create a Symbiotic Neural Network Watchdog.

```
# Create a TensorBoard Callback to display training logs
tb_cb = tf.keras.callbacks.TensorBoard(log_dir=log_dir,
   histogram_freq=1)
```

```
# Network Training Hyperparameters
batch_size = 64
Epochs = 10

# Compile the symbiotic model, using 1.0 for both classifier
# and generator training weights

symbiotic_model.compile(
optimizer=tf.keras.optimizers.Adam(learning_rate=.001),
loss={
  "Classifier": tf.keras.losses.categorical_crossentropy,
  "Generator": tf.keras.losses.MSE,
},
loss_weights=[1.0,1.0],
metrics = ['accuracy'])

# Train the symbiotic neural network
symbiotic_model.fit(image_training_data,
{"Classifier" : labels_training_data,
 "Generator": image_training_data},
epochs = Epochs, batch_size = batch_size,
callbacks=[tb_cb])
```

Listing A.4: Example code used to compile and train a symbiotic watchdog model.

```
import numpy as np
```

```
# Define a function to calculate image RMSE values
def img_rmse(image1, image2):
  """
  :param image1: First Image
  :param image2: Second Image
  :return: Root Mean Squared Error, RMSE
  """

  error = np.sqrt(np.sum((image1.astype('float')-image2.astype(
     'float'))**2))
  return error

# Determine whether input image should be permitted or rejected
def permit_input(original_image, generated_image, threshold):
  """
  :param original_image: Original input image
  :param generated_image: Watchdog regenerated image
  :param threshold: Threshold Value
  :return: Whether the image should be permitted
  """

  if img_rmse(original_image, generated_image) < threshold:
    return True
  else:
    return False
```

Listing A.5: An example function used determine whether an image should be permitted or rejected by a watchdog.

BIBLIOGRAPHY

[1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Zheng, X. (2016). Tensorflow: A system for large-scale machine learning. In 12th USENIX symposium on operating systems design and implementation (OSDI 16) (pp. 265-283).

[2] Abbasi, M., Shui, C., Rajabi, A., Gagné, C., & Bobba, R. (2019). Toward metrics for differentiating out-of-distribution sets. arXiv preprint arXiv:1910.08650.

[3] Abokifa, A. A., Haddad, K., Lo, C. S., & Biswas, P. (2017). Detection of cyber physical attacks on water distribution systems via principal component analysis and artificial neural networks. In World Environmental and Water Resources Congress 2017 (pp. 676-691).

[4] An, J., & Cho, S. (2015). Variational autoencoder based anomaly detection using reconstruction probability. Special Lecture on IE, 2(1), 1-18.

[5] Ariyaluran Habeeb, R. A., Nasaruddin, F., Gani, A., Amanullah, M. A., Abaker Targio Hashem, I., Ahmed, E., & Imran, M. (2019). Clustering-based real-time anomaly detection—A breakthrough in big data technologies. Transactions on Emerging Telecommunications Technologies, e3647.

[6] Brauckhoff, D., Salamatian, K., & May, M. (2009, April). Applying PCA for traffic anomaly detection: Problems and solutions. In IEEE INFOCOM 2009 (pp. 2866-2870). IEEE.

[7] Bui, J., & Marks II, R. J. (2021). Autoencoder Watchdog Outlier Detection for Classifiers. Proceedings of the 13th International Conference on Agents and Artificial Intelligence - Volume 2 2021 (pp. 990-996) .

[8] Bui, J., & Marks II, R. J. (2021). Symbiotic Hybrid Neural Network Watchdog For Outlier Detection Proceedings: Machine Learning and Data Mining in Pattern Recognition, IBAI Publishing, New York, July 18-22, 2021

[9] Bui, J. M., Amigo, G. A., & Marks II, R. J. (2021). Generatively Augmented Neural Network Watchdog for Image Classification Networks. arXiv preprint arXiv:2109.06168.

[10] Bulusu, S., Kailkhura, B., Li, B., Varshney, P. K., & Song, D. (2020). Anomalous example detection in deep learning: A survey. IEEE Access, 8, 132330-132347.

[11] Chalapathy, R., & Chawla, S. (2019). Deep learning for anomaly detection: A survey. arXiv preprint arXiv:1901.03407.

[12] Chiang, H. T., Hsieh, Y. Y., Fu, S. W., Hung, K. H., Tsao, Y., & Chien, S. Y. (2019). Noise reduction in ECG signals using fully convolutional denoising autoencoders. IEEE Access, 7, 60806-60813.

[13] Clanuwat, T., Bober-Irizar, M., Kitamoto, A., Lamb, A., Yamamoto, K., & Ha, D. (2018). Deep learning for classical japanese literature. arXiv preprint arXiv:1812.01718.

[14] El-Sharkawi, M.A., R.J. Marks II, Robert J. Streifel and I. Kerszenbaum "Detection and Localization of Shorted-Turns in the DC-Field Winding of Turbine-Generator Rotors Using Novelty Filters and Fuzzified Neural Networks," in Fuzzy System Theory in Electrical Power Engineering, M.E. El-Hawary, editor (IEEE Press, 1998), pp.85-111.

[15] Fernandes Jr, G., Rodrigues, J. J., & Proenca Jr, M. L. (2015). Autonomous profile-based anomaly detection system using principal component analysis and flow analysis. Applied Soft Computing, 34, 513-525.

[16] Fernandes Jr, G., Carvalho, L. F., Rodrigues, J. J., & Proença Jr, M. L. (2016). Network anomaly detection using IP flows with principal component analysis and ant colony optimization. Journal of Network and Computer Applications, 64, 1-11.

[17] Fredieu, C. T., Bui, J., Martone, A., Marks II, R. J., Baylis, C., & Buehrer, R. M. (2021). Classification of Common Waveforms Including a Watchdog for Unknown Signals. arXiv preprint arXiv:2108.07339.

[18] Galán, G. A. A., Bui, J., & Marks, R. J. (2021). Cascade Watchdog: A Multi-tiered Adversarial Guard for Outlier Detection. arXiv preprint arXiv:2108.09375.

[19] García-Gonzàlez, J., Molina-Cabello, M. A., Luque-Baena, R. M., Ortiz-de-Lazcano-Lobato, J. M., & López-Rubio, E. (2020, October). Deep autoencoder architectures for foreground object detection in video sequences based on probabilistic mixture models. In 2020 IEEE International Conference on Image Processing (ICIP) (pp. 3199-3203). IEEE.

[20] Garg, A., & Mago, V. (2021). Role of machine learning in medical research: A survey. Computer Science Review, 40, 100370.

[21] Gibney, E. (2016). Google AI algorithm masters ancient game of Go. Nature News, 529(7587), 445.

[22] Golan, I., & El-Yaniv, R. (2018). Deep anomaly detection using geometric transformations. arXiv preprint arXiv:1805.10917.

[23] Gondara, L. (2016, December). Medical image denoising using convolutional denoising autoencoders. In 2016 IEEE 16th international conference on data mining workshops (ICDMW) (pp. 241-246). IEEE.

[24] Grigorescu, S., Trasnea, B., Cocias, T., & Macesanu, G. (2020). A survey of deep learning techniques for autonomous driving. Journal of Field Robotics, 37(3), 362-386.

[25] Gu, Y., Liu, Y., & Zhang, Y. (2006, May). A selective kernel PCA algorithm for anomaly detection in hyperspectral imagery. In 2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings (Vol. 2, pp. II-II). IEEE.

[26] Guttormsson, S. E., Marks, R. J., El-Sharkawi, M. A., & Kerszenbaum, I. (1999). Elliptical novelty grouping for on-line short-turn detection of excited running rotors. IEEE Transactions on Energy Conversion, 14(1), 16-22.

[27] Hanga, K. M., & Kovalchuk, Y. (2019). Machine learning and multi-agent systems in oil and gas industry applications: A survey. Computer Science Review, 34, 100191.

[28] Haug, S., Marks, R. J., & Dembski, W. A. (2021). Exponential Contingency Explosion: Implications for Artificial General Intelligence. IEEE Transactions on Systems, Man, and Cybernetics: Systems.

[29] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[30] Hechtlinger, Y., Póczos, B., & Wasserman, L. (2018). Cautious deep learning. arXiv preprint arXiv:1805.09460.

[31] Hendrycks, D., & Gimpel, K. (2016). A baseline for detecting misclassified and out-of-distribution examples in neural networks. arXiv preprint arXiv:1610.02136.

[32] Huang, K., & Wang, X. (2021). ADA-INCVAE: Improved data generation using variational autoencoder for imbalanced classification. Applied Intelligence, 1-16.

[33] Huang, L., Nguyen, X., Garofalakis, M., Jordan, M. I., Joseph, A., & Taft, N. (2006, December). In-network PCA and anomaly detection. In NIPS (Vol. 2006, pp. 617-624).

[34] Huang, Y., Jin, W., Yu, Z., & Li, B. (2021). A robust anomaly detection algorithm based on principal component analysis. Intelligent Data Analysis, 25(2), 249-263.

[35] Islam, Z., Abdel-Aty, M., Cai, Q., & Yuan, J. (2021). Crash data augmentation using variational autoencoder. Accident Analysis & Prevention, 151, 105950.

[36] Jablonski, J. A., Bihl, T. J., & Bauer, K. W. (2015). Principal component reconstruction error for hyperspectral anomaly detection. IEEE Geoscience and Remote Sensing Letters, 12(8), 1725-1729.

[37] Jiang, D., Yao, C., Xu, Z., & Qin, W. (2015). Multi-scale anomaly detection for high-speed network traffic. Transactions on Emerging Telecommunications Technologies, 26(3), 308-317.

[38] Johnson, R. J., Williams, J. P., & Bauer, K. W. (2012). AutoGAD: An improved ICA-based hyperspectral anomaly detection algorithm. IEEE Transactions on Geoscience and Remote Sensing, 51(6), 3492-3503.

[39] Kamilaris, A., & Prenafeta-Boldú, F. X. (2018). Deep learning in agriculture: A survey. Computers and electronics in agriculture, 147, 70-90.

[40] Keshk, M., Sitnikova, E., Moustafa, N., Hu, J., & Khalil, I. (2019). An integrated framework for privacy-preserving based anomaly detection for cyber-physical systems. IEEE Transactions on Sustainable Computing, 6(1), 66-79.

[41] Kim, K., & Myung, H. (2018). Autoencoder-combined generative adversarial networks for synthetic image data generation and detection of jellyfish swarm. IEEE Access, 6, 54207-54214.

[42] Kim, S., Jo, W., & Shon, T. (2020). APAD: Autoencoder-based payload anomaly detection for industrial IoE. Applied Soft Computing, 88, 106017.

[43] Kiss, I., Genge, B., Haller, P., & Sebestyén, G. (2014, September). Data clustering-based anomaly detection in industrial control systems. In 2014 IEEE 10th International Conference on Intelligent Computer Communication and Processing (ICCP) (pp. 275-281). IEEE.

[44] Li, B., Sun, Z., & Guo, Y. (2019, July). Supervae: Superpixelwise variational autoencoder for salient object detection. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 33, No. 01, pp. 8569-8576).

[45] Li, P., Niggemann, O., & Hammer, B. (2018, October). A geometric approach to clustering based anomaly detection for industrial applications. In IECON 2018-44th Annual Conference of the IEEE Industrial Electronics Society (pp. 5345-5352). IEEE.

[46] Lin, Y., & Wang, J. (2019). Probabilistic deep autoencoder for power system measurement outlier detection and reconstruction. IEEE Transactions on Smart Grid, 11(2), 1796-1798.

[47] Liu, F. T., Ting, K. M., & Zhou, Z. H. (2012). Isolation-based anomaly detection. ACM Transactions on Knowledge Discovery from Data (TKDD), 6(1), 1-39.

[48] Lore, K. G., Akintayo, A., & Sarkar, S. (2017). LLNet: A deep autoencoder approach to natural low-light image enhancement. Pattern Recognition, 61, 650-662.

[49] Lu, X., Tsao, Y., Matsuda, S., & Hori, C. (2013, August). Speech enhancement based on deep denoising autoencoder. In Interspeech (Vol. 2013, pp. 436-440).

[50] Ma, L., Crawford, M. M., & Tian, J. (2010). Anomaly detection for hyperspectral images based on robust locally linear embedding. Journal of Infrared, Millimeter, and Terahertz Waves, 31(6), 753-762.

[51] Mao, J., Wang, H., & Spencer Jr, B. F. (2021). Toward data anomaly detection for automated structural health monitoring: Exploiting generative adversarial nets and autoencoders. Structural Health Monitoring, 20(4), 1609-1626.

[52] Marchi, E., Vesperini, F., Eyben, F., Squartini, S., & Schuller, B. (2015, April). A novel approach for automatic acoustic novelty detection using a denoising autoencoder with bidirectional LSTM neural networks. In 2015 IEEE international conference on acoustics, speech and signal processing (ICASSP) (pp. 1996-2000). IEEE.

[53] Marcos Alvarez, A., Yamada, M., Kimura, A., & Iwata, T. (2013, October). Clustering-based anomaly detection in multi-view data. In Proceedings of the 22nd ACM international conference on Information & Knowledge Management (pp. 1545-1548).

[54] McCarthy, J. (2007). What is artificial intelligence?.

[55] Meena, S. D., & Agilandeeswari, L. (2020). Stacked convolutional autoencoder for detecting animal images in cluttered scenes with a novel feature extraction framework. In Soft computing for problem solving (pp. 513-522). Springer, Singapore.

[56] Meng, Q., Catchpoole, D., Skillicom, D., & Kennedy, P. J. (2017, May). Relational autoencoder for feature extraction. In 2017 International Joint Conference on Neural Networks (IJCNN) (pp. 364-371). IEEE.

[57] Münz, G., Li, S., & Carle, G. (2007, September). Traffic anomaly detection using k-means clustering. In GI/ITG Workshop MMBnet (pp. 13-14).

[58] Narayanan, S., Marks, R. J., Vian, J. L., Choi, J. J., El-Sharkawi, M. A., & Thompson, B. B. (2002, May). Set constraint discovery: missing sensor data restoration using autoassociative regression machines. In Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No. 02CH37290) (Vol. 3, pp. 2872-2877). IEEE.

[59] Nguyen, A., Yosinski, J., & Clune, J. (2015). Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 427-436).

[60] Nguyen, D. T., Lou, Z., Klar, M., & Brox, T. (2019, May). Anomaly detection with multiple-hypotheses predictions. In International Conference on Machine Learning (pp. 4800-4809). PMLR.

[61] Nguyen, H. D., Tran, K. P., Thomassey, S., & Hamad, M. (2021). Forecasting and Anomaly Detection approaches using LSTM and LSTM Autoencoder techniques with the applications in supply chain management. International Journal of Information Management, 57, 102282.

[62] Ozbayoglu, A. M., Gudelek, M. U., & Sezer, O. B. (2020). Deep learning for financial applications: A survey. Applied Soft Computing, 93, 106384.

[63] Pang, G., Shen, C., Cao, L., & Hengel, A. V. D. (2021). Deep learning for anomaly detection: A review. ACM Computing Surveys (CSUR), 54(2), 1-38.

[64] Park, S., Yu, S., Kim, M., Park, K., & Paik, J. (2018). Dual autoencoder network for retinex-based low-light image enhancement. IEEE Access, 6, 22084-22093.

[65] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems, 32, 8026-8037.

[66] Pidhorskyi, S., Almohsen, R., Adjeroh, D. A., & Doretto, G. (2018). Generative probabilistic novelty detection with adversarial autoencoders. arXiv preprint arXiv:1807.02588.

[67] Polic, M., Krajacic, I., Lepora, N., & Orsag, M. (2019). Convolutional autoencoder for feature extraction in tactile sensing. IEEE Robotics and Automation Letters, 4(4), 3671-3678.

[68] Prystavka, P., Cholyshkina, O., Dolgikh, S., & Karpenko, D. (2020, September). Automated object recognition system based on convolutional autoencoder. In 2020 10th International Conference on Advanced Computer Information Technologies (ACIT) (pp. 830-833). IEEE.

[69] Pu, G., Wang, L., Shen, J., & Dong, F. (2020). A hybrid unsupervised clustering-based anomaly detection method. Tsinghua Science and Technology, 26(2), 146-153.

[70] Qi, Y., Shen, C., Wang, D., Shi, J., Jiang, X., & Zhu, Z. (2017). Stacked sparse autoencoder-based deep network for fault diagnosis of rotating machinery. Ieee Access, 5, 15066-15079.

[71] Ren, J., Liu, P. J., Fertig, E., Snoek, J., Poplin, R., DePristo, M. A., ... & Lakshminarayanan, B. (2019). Likelihood ratios for out-of-distribution detection. arXiv preprint arXiv:1906.02845.

[72] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65(6), 386.

[73] Roy, M., Bose, S. K., Kar, B., Gopalakrishnan, P. K., & Basu, A. (2018, November). A stacked autoencoder neural network based automated feature extraction method for anomaly detection in on-line condition monitoring. In 2018 IEEE Symposium Series on Computational Intelligence (SSCI) (pp. 1501-1507). IEEE.

[74] Ryu, S., Choi, H., Lee, H., & Kim, H. (2019). Convolutional autoencoder based feature extraction and clustering for customer load analysis. IEEE Transactions on Power Systems, 35(2), 1048-1060.

[75] Sakurada, M., & Yairi, T. (2014, December). Anomaly detection using autoencoders with nonlinear dimensionality reduction. In Proceedings of the MLSDA 2014 2nd workshop on machine learning for sensory data analysis (pp. 4-11).

[76] Sharp, M., Ak, R., & Hedberg Jr, T. (2018). A survey of the advancing use and development of machine learning in smart manufacturing. Journal of manufacturing systems, 48, 170-179.

[77] Slavic, G., Campo, D., Baydoun, M., Marin, P., Martin, D., Marcenaro, L., & Regazzoni, C. (2020, May). Anomaly detection in video data based on probabilistic latent space models. In 2020 IEEE Conference on Evolving and Adaptive Intelligent Systems (EAIS) (pp. 1-8). IEEE.

[78] Stallkamp, J., Schlipsing, M., Salmen, J., & Igel, C. (2012). Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. Neural networks, 32, 323-332.

[79] Streifel, R. J., Marks, R. J., El-Sharkawi, M. A., & Kerszenbaum, I. (1996). Detection of shorted-turns in the field winding of turbine-generator rotors using novelty detectors-development and field test. IEEE Transactions on Energy Conversion, 11(2), 312-317.

[80] Thompson, B. B., Marks, R. J., Choi, J. J., El-Sharkawi, M. A., Huang, M. Y., & Bunje, C. (2002, May). Implicit learning in autoencoder novelty assessment. In Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No. 02CH37290) (Vol. 3, pp. 2878-2883). IEEE.

[81] Thompson, B. B., Marks, R. J., & El-Sharkawi, M. A. (2003, July). On the contractive nature of autoencoders: Application to missing sensor restoration. In Proceedings of the International Joint Conference on Neural Networks, 2003. (Vol. 4, pp. 3011-3016). IEEE.

[82] Thudumu, S., Branch, P., Jin, J., & Singh, J. J. (2020). A comprehensive survey of anomaly detection techniques for high dimensional big data. Journal of Big Data, 7(1), 1-30.

[83] Ullah, W., Ullah, A., Haq, I. U., Muhammad, K., Sajjad, M., & Baik, S. W. (2021). CNN features with bi-directional LSTM for real-time anomaly detection in surveillance networks. Multimedia Tools and Applications, 80(11), 16979-16995.

[84] Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P. A. (2008, July). Extracting and composing robust features with denoising autoencoders. In Proceedings of the 25th international conference on Machine learning (pp. 1096-1103).

[85] Vu, H. S., Ueta, D., Hashimoto, K., Maeno, K., Pranata, S., & Shen, S. M. (2019). Anomaly detection with adversarial dual autoencoders. arXiv preprint arXiv:1902.06924.

[86] Vu, L., Nguyen, Q. U., Nguyen, D. N., Hoang, D. T., & Dutkiewicz, E. (2020). Learning Latent Representation for IoT Anomaly Detection. IEEE Transactions on Cybernetics.

[87] Wan, Z., Zhang, Y., & He, H. (2017, November). Variational autoencoder based synthetic data generation for imbalanced learning. In 2017 IEEE symposium series on computational intelligence (SSCI) (pp. 1-7). IEEE.

[88] Wang, G., Yang, J., & Li, R. (2016). An anomaly detection framework based on ICA and Bayesian classification for IaaS platforms. KSII Transactions on Internet and Information Systems (TIIS), 10(8), 3865-3883.

[89] Wang, R., Nie, K., Wang, T., Yang, Y., & Long, B. (2020, January). Deep learning for anomaly detection. In Proceedings of the 13th International Conference on Web Search and Data Mining (pp. 894-896).

[90] Wang, W., & Battiti, R. (2006, April). Identifying intrusions in computer networks with principal component analysis. In First international conference on availability, reliability and security (ARES'06) (pp. 8-pp). IEEE.

[91] Wang, Z., & Bovik, A. C. (2009). Mean squared error: Love it or leave it? A new look at signal fidelity measures. IEEE signal processing magazine, 26(1), 98-117.

[92] Wang, Z., Bovik, A. C., Sheikh, H. R., & Simoncelli, E. P. (2004). Image quality assessment: from error visibility to structural similarity. IEEE transactions on image processing, 13(4), 600-612.

[93] "Machine learning." Merriam-Webster.com Dictionary, Merriam-Webster, https://www.merriam-webster.com/dictionary/machine

[94] Wu, J., Zhao, Z., Sun, C., Yan, R., & Chen, X. (2020). Fault-attention generative probabilistic adversarial autoencoder for machine anomaly detection. IEEE Transactions on Industrial Informatics, 16(12), 7479-7488.

[95] Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747.

[96] Xie, K., Li, X., Wang, X., Cao, J., Xie, G., Wen, J., ... & Qin, Z. (2018). Online anomaly detection with high accuracy. IEEE/ACM transactions on networking, 26(3), 1222-1235.

[97] Xing, C., Ma, L., & Yang, X. (2016). Stacked denoise autoencoder based feature extraction and classification for hyperspectral images. Journal of Sensors, 2016.

[98] Xiong, Y., & Zuo, R. (2021). Robust Feature Extraction for Geochemical Anomaly Recognition Using a Stacked Convolutional Denoising Autoencoder. Mathematical Geosciences, 1-22.

[99] Yadav, N., Yadav, A., & Kumar, M. (2015). History of neural networks. In An Introduction to Neural Network Methods for Differential Equations (pp. 13-15). Springer, Dordrecht.

[100] Yan, B., & Han, G. (2018). Effective feature extraction via stacked sparse autoencoder to improve intrusion detection system. IEEE Access, 6, 41238-41248.

[101] Yao, R., Liu, C., Zhang, L., & Peng, P. (2019, June). Unsupervised anomaly detection using variational auto-encoder based feature extraction. In 2019 IEEE International Conference on Prognostics and Health Management (ICPHM) (pp. 1-7). IEEE.

[102] Yasenko, L., Klyatchenko, Y., & Tarasenko-Klyatchenko, O. (2020, May). Image noise reduction by denoising autoencoder. In 2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT) (pp. 351-355). IEEE.

[103] Zhang, C., Zhou, L., Zhao, Y., Zhu, S., Liu, F., & He, Y. (2020). Noise reduction in the spectral domain of hyperspectral images using denoising autoencoder methods. Chemometrics and Intelligent Laboratory Systems, 203, 104063.

[104] Zhou, C., & Paffenroth, R. C. (2017, August). Anomaly detection with robust deep autoencoders. In Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining (pp. 665-674).