Published in final edited form as:
Conf Comput Vis Pattern Recognit Workshops. 2020 June ; 2020: 3639–3648. doi:10.1109/cvprw50498.2020.00425.

PI-Net: A Deep Learning Approach to Extract Topological Persistence Images

Anirudh Som[1,2], **Hongjun Choi**[1,2], **Karthikeyan Natesan Ramamurthy**[3], **Matthew P. Buman**[4], **Pavan Turaga**[1,2]

[1]School of Arts, Media and Engineering, Arizona State University

[2]School of Electrical, Computer and Energy Engineering, Arizona State University

[3]IBM T. J. Watson Research Center

[4]College of Health Solutions, Arizona State University

Abstract

Topological features such as persistence diagrams and their functional approximations like persistence images (PIs) have been showing substantial promise for machine learning and computer vision applications. This is greatly attributed to the robustness topological representations provide against different types of physical nuisance variables seen in real-world data, such as view-point, illumination, and more. However, key bottlenecks to their large scale adoption are computational expenditure and difficulty incorporating them in a differentiable architecture. We take an important step in this paper to mitigate these bottlenecks by proposing a novel one-step approach to generate PIs directly from the input data. We design two separate convolutional neural network architectures, one designed to take in multi-variate time series signals as input and another that accepts multi-channel images as input. We call these networks Signal PI-Net and Image PI-Net respectively. To the best of our knowledge, we are the first to propose the use of deep learning for computing topological features directly from data. We explore the use of the proposed PI-Net architectures on two applications: human activity recognition using tri-axial accelerometer sensor data and image classification. We demonstrate the ease of fusion of PIs in supervised deep learning architectures and speed up of several orders of magnitude for extracting PIs from data. Our code is available at https://github.com/anirudhsom/PI-Net.

1. Introduction

Deep learning over the past decade has had tremendous impact in computer vision, natural language processing, machine learning, and healthcare. Among other approaches, convolutional neural networks (CNNs) in particular have received great attention and interest from the computer vision community. This is attributed to the fact that they are able to exploit the local temporal and spatial correlations that exist in 1-dimensional (1D) sequential time-series signals, 2-dimensional (2D) data like images, 3-dimensional (3D) data like videos, and 3D objects. In this paper, we refer to these type of data as *input data*. CNNs

asom2@asu.edu.

also have far less learnable parameters than their fully-connected counterparts, making them less prone to over-fitting and have shown state-of-the-art results in applications like image classification, object detection, scene recognition, fine-grained categorization and action recognition [27, 20, 54, 55, 56]. Apart from being good at learning mappings between the input and corresponding class labels, deep learning frameworks are also efficient in discovering mappings between the input data and other output feature representations [49, 51, 30, 16, 13].

While methods for learning features from scratch and mapping image data to desired outputs via neural networks have matured significantly, relatively less attention has been paid to invariance to nuisance low-level physical factors like sensor noise. Topological data analysis (TDA) methods are popularly used to characterize the *shape* of *n*-dimensional point cloud data using representations such as persistent diagrams (PDs) that are robust to certain types of variations in the data [14]. TDA methods have also been successfully applied to different computer vision problems and have shown the ability to incorporate different invariances of interest to the computer vision community [28, 12, 44]. The shape of the data is quantified by properties such as connected components, cycles, high-dimensional holes, level-sets and monotonic regions of functions defined on the data [14]. Topological properties are those invariants that do not change under smooth deformations like stretching, bending and rotation, but without tearing or gluing surfaces. These attractive traits of TDA have renewed interested in this area for answering various fundamental questions, including those dealing with interpretation, generalization, model selection, stability, and convergence [19, 6, 37, 35, 18, 17].

A lot of work has gone into utilizing topological representations efficiently in large-scale machine learning [3, 5, 38, 33, 36, 1, 44]. However, bottlenecks such as computational load involved in discovering topological invariants as well as a lack of a differentiable architecture remain. In this paper we propose simple deep learning architectures to learn approximate mappings between data and their topological feature representations.The gist of our idea is illustrated in Figure 1 and the main contributions are listed below.

Contributions:

1. We propose a novel differentiable neural network architecture called *PI-Net*, to extract topological representations. In this paper we focus on persistence images (PIs) as the desired topological feature.
2. We provide two simple CNN-based architectures called *Signal PI-Net* that takes in multi-variate 1D sequential data and *Image PI-Net* that takes in multi-channel 2D image data.
3. We explore transfer learning strategies to train the proposed *PI-Net* model on a source dataset and use it on a different target dataset, with or without fine-tuning.
4. Through our experiments on human activity recognition using accelerometer sensor data and image classification on standard image datasets, we show the effectiveness of the generated approximations for PIs and compare their performance to PIs generated using analytic TDA methods. We also investigate

the benefits of concatenating PIs with features learnt using deep learning methods.

5. We also evaluate the robustness of classification models to Gaussian noise, with or without fusion with PI representations in image classification tasks.

The rest of the paper is outlined as follows: Section 2 discusses related work. Section 3 provides the necessary background on TDA, PIs and CNNs. In Section 4 we describe the proposed *PI-Net* frameworks in detail and in Section 5 we describe the experimental results. Section 6 concludes the paper.

2. Related Work

Although the formal beginnings of topology is already a few centuries old dating back to Euler, algebraic topology has seen a revival in the past decade with the advent of computational tools and software [39, 2, 4]. Arguably the most popular topological summary is the persistence diagram (PD), which is a multi-set of points in a 2D plane that quantifies the *birth* and *death* times of topological features such as k-dimensional holes or sub-level sets of a function defined on a point cloud [15]. This simple summary has resulted in the adoption of topological methods for various applications [34, 47, 8, 11, 10, 23, 42, 48, 31]. However, TDA methods suffer from two major limitations. First, it is computationally very taxing to extract PDs. The computational load increases both with the dimensionality and with the number of samples in the data being analyzed. The second obstacle is that a PD is a multi-set of points, making it impossible to use machine learning or deep learning frameworks directly on the space of PDs. Efforts have been made to tackle the second issue by attempting to map PDs to spaces that are more favorable for machine learning tools [3, 5, 38, 33, 36, 1, 44]. For further reading, [43] surveys recent topological representations and their associated metrics. To alleviate the first problem, in this paper we propose a simple one-step differentiable architecture called *PI-Net* to compute the desired topological feature representation, specifically persistence images. To the best of our knowledge, we are the first to propose the use of deep learning for computing PIs directly from data.

Our motivation to use deep learning stems from its successful use to learn mappings between input data and different feature representations [49, 51, 30, 16, 13]. However, deep learning and TDA did cross paths before but not in the same context as what we propose in this paper. TDA methods have been used to study the topology [19, 6], algorithmic complexity [37], behavior [18] and selection [35] of deep learning models. Efforts have also been made to use topological feature representations either as inputs or fused with features learned using neural network models [12, 24, 7]. Later in Section 5, we show experimental results on fusing generated PIs with deep learning frameworks for action recognition and image classification.

3. Background

Persistence Diagrams:

Consider a graph $G = \{V,E\}$ constructed from data projected onto a high-dimensional point-cloud space. Here, V is the set of $|V|$ nodes and E denotes the neighborhood relations defined between the samples. Topological properties of the graph can be estimated by first constructing a simplicial complex S over G. S is defined as $S = (G, \Sigma)$, with Σ being a family of non-empty level sets of G, with each element $\sigma \in \Sigma$ is a simplex [15]. This falls under the realm of *persistent homology* where we are interested in summarizing the k-dimensional holes present in the data. The simplices are constructed using the the ϵ-neighborhood rule [15]. It is also possible to quantify the topology induced by a function g defined on the vertices of a graph G by studying the topology of its sub-level or super-level sets. Since $g : V \to \mathbb{R}$ this is referred to as *scalar field topology*. In either case, PDs provide a simple way to summarize the birth vs death time information of the topological feature of interest. In this paper we use *persistent homology* to compute ground-truth PDs for images and *scalar field topology* to compute ground-truth PDs for time-series signals. In a PD the birth-time b refers to the scale at which the feature was formed and death-time d refers to the scale at which it ceases to exist. The difference between d and b gives us the life-time or persistence and is denoted by $l = |d - b|$. Each PD is a multi-set of points (b, d) in $\mathbb{R}^2$. Since $d \geq b$, only one-half of the space in the PD is actually utilized. Points in the PD that lie close to the diagonal represent noise and can be easily discarded by simple thresholding. Plotting the birth-time vs life-time information allows us to utilize the entire 2D space of a PD as shown in Figure 2. Interested readers can refer to the following papers to learn more about the properties of the space of PDs [14, 15].

Persistence Images:

A PI is a finite-dimensional vector representation of a PD [1] and can be computed through the following series of steps. First we map the PD to an integrable function $\rho : \mathbb{R} \to \mathbb{R}^2$ called a persistence surface. The persistence surface ρ is defined as a weighted sum of Gaussian functions that are centered at each point in the PD. Next, a discretization of a sub-domain of the persistence surface is done which results in a grid. Finally, the PI is obtained by integrating the persistence surface over each grid box, giving us a matrix of pixel values. An interesting aspect when computing PIs is the broad range of weighting functions to chose from, to weight the Gaussian functions. Typically, points of high persistence or lifetime are perceived to be more important than points of low persistence. In such cases one may select the weighting function to be non-decreasing with respect to the persistence value of each point in the PD. Adams *et al.* also talk about the stability of persistence images with respect to the 1-Wasserstein distance between PDs [1]. Figure 2 illustrates an example of a PD and its PI where the points are weighted by their life-time.

Convolutional Neural Networks:

CNNs were inspired from the hierarchical organization of the human visual cortex [21] and consist of many intricately interconnected layers of neuron structures serving as the basic units to learn, extract both low-level and high-level features from images. CNNs are particularly more attractive and powerful compared to their connected counterparts because CNNs are able to exploit the spatial correlations present in natural images and each convolutional layer has far less trainable parameters than a fully-connected layer. Several sophisticated CNN architectures have been proposed in the last decade, for example *AlexNet* [27], *VGG* [41], *GoogleNet* [46], *ResNet* [22], *DenseNet* [25], *etc*. Some of these designs are known to surpass humans for object recognition tasks [40]. Apart from discovering features from scratch for classification tasks, CNNs are also popular for learning mappings between input and other feature representations [49, 51, 30, 16, 13]. This motivates us to design simple CNN models for the task of learning mappings between the data and their PI representations. We would like to direct interested readers to the following survey paper to know more about different CNN architectures [29, 45].

Learning Strategies:

Here we will briefly talk about the two learning strategies namely: *Supervised Learning* and *Transfer Learning*. We employ these strategies to train the proposed *PI-Net* model. *Supervised Learning* is concerned with learning complex mappings from X to Y when many pairs of (x, y) are given as training data, with $x \in X$ being the input data and $y \in Y$ being the corresponding label or feature representation. In a classification setting Y corresponds to a fixed set of labels. In a regression setting, the output Y is either a real number or a set of real numbers. In this paper our problem falls under the regression category as we try to learn a mapping between the input data and its PI. *Transfer Learning* is a design methodology that involves using the learned weights of a pre-trained model that is trained on a source dataset D_s for the source task T_s, to initialize the weights of another model that is fine-tuned using a target dataset D_t for the target task T_t [52]. This allows us to leverage the source dataset that the model was initially trained on without having to train the model from scratch. The is useful in cases where the target dataset has a lot less data samples compared to the source dataset. In Section 4 we show how transfer learning is employed in our proposed framework when the target training data is limited.

4. PI-Net Framework

In this section we first describe the steps to generate ground-truth PIs and later discuss the proposed *Signal PI-Net* and *Image PI-Net* configurations.

4.1. Generating Ground-truth Persistence Images

Data Pre-processing: For uni-variate or multi-variate time-series signals, we consider only fixed-frame signals, *i.e.* signals with fixed number of time-steps, and zero-center them. We standardize the training and test sets such that they have unit variance along each time-step. For images we enforce the pixel values to range between [0,1].

Persistence Images for Time-series Data: We use the *Scikit-TDA* python library [39] and use the *Ripser* package for computing PDs. As mentioned earlier, we compute *level-set* filtration PDs for time-series signals. *Scalar field topology* offers a simple way to summarize the different peaks and troughs present in the signal. For example a local minima gives birth to a topological feature (more accurately a 0-dimensional homology group feature) which dies at its local maxima. We compute PDs for each of the *x,y,z* signals in the accelerometer sample. For better use of the 2D space in the PD we consider birth-time vs life-time information. For computing PIs we used the *Persim* package in the *Scikit-TDA* toolbox. In all our experiments we set the grid size of the generated PIs to 50×50 and fit a Gaussian kernel function on each point in the PD. We weight each Gaussian kernel by the life-time of the point. For all time-series datasets we set the standard deviation of the Gaussian kernel to

0.25 and set the birth-time range to [−10, 10]. Once computed we normalize each PI by dividing by its maximum intensity value. This forces the values in the PI to also lie between [0,1].

Persistence Images for Multi-channel Image Data: Here too we use the *Scikit-TDA* library for computing PDs and PIs. We represent each image channel as a 3D point cloud with the three coordinates representing the x-coordinate, y-coordinate and intensity value of each pixel in the image. For example, an image with c channels will result in c 3D point clouds. The x and y coordinate information is also normalized to be within [0, 1]. Finally, we compute the 1-dimensional persistent homology PDs for each channel in the image using the process described in Section 3. For all image datasets in our experiments we discard points in the PD with life-time less than 0.02. For computing PIs we set the grid size of the generated PIs to 50×50 and fit a Gaussian kernel function on each point in the PD. The Gaussian kernel is weighted by the life-time of the point. Other parameters needed to compute PIs like birth-time range and standard-deviation of the Gaussian kernel were set to different values specific to each dataset. We consider the following three datasets in our experiments: *CIFAR10* [26], *CIFAR100* [26] and *SVHN* [32]. For *CIFAR10* and *CIFAR100* we set birth-time range and standard-deviation to [0,0.3] and 0.01. For *SVHN* we set the same parameters to [0,0.2] and 0.005 respectively. Finally, each of the c PIs generated for a c-channel image is further normalized to lie in the range [0,1].

4.2. Network Architecture

Both *PI-Net* architectures were designed using Keras with TensorFlow back-end [9].

Signal PI-Net: The input to the network is a $b \times t \times n$ dimensional time-series signal, where b is the batch-size, t refers to the number of time-steps or frame size. For a uni-variate signal $n = 1$ and for a multi-variate signal $n > 1$. For our experiments in section 5, n is 3 and t is either 250 or 500. After the input layer, the encoder block consists of four 1D convolution layers. Except the final convolution layer, all other convolution layers are followed by batch normalization, ReLU activation and Max-pooling. The final convolution layer is followed by batch normalization, ReLU activation and Global-average-pooling. The number of convolution filters is set to 128, 256, 512 and 1024 respectively. However, the convolution kernel size is same for all layers and is set to 3 with stride set to 1. We use appropriate zero padding to keep the output shape of the convolution layer unchanged. For all Max-pool layers, we set the kernel size to 3 and stride to 2. After the encoder block, we pass the global-average-pooled output into a final output dense layer of size 2500×n. The output of the dense layer is subjected to ReLU activation and reshaped to size $50 \times 50 \times n$.

Image PI-Net: The input to this network is a $b \times h \times w \times c$ dimensional image, where b, h, w, c is the batch-size, the image height, width and number of channels respectively. *Image PI-Net* follows the same architecture as *Signal PI-Net* for the encoder block. However, we now use 2D convolution layers instead. Also, for all the convolution layers the number of filters and kernel size was set to 128 and 3 respectively. We use appropriate zero padding to keep the output shape of the convolution layer unchanged. For all Max-pool layers, we set the kernel size to 3 and stride to 2. We pass the output of the encoder block into a latent variable layer which consists of a dense layer of size 2500. The output of the latent variable layer is reshaped to 50×50 and is passed into the decoder block. The decoder block consists of one 2D deconvolution layer with kernel size set to 50, stride set to 1, number of filters to c. The output of the deconvolution layer is also zero-padded such that the height and width of the output remain unchanged. The deconvolution layer is followed by a final batch normalization and Sigmoid activation. The shape of the output we get is $50 \times 50 \times c$.

5. Experiments

This section can be broadly divided into four parts. First we show results for human activity recognition by using PIs alone and PIs in fusion with different deep learning models on two accelerometer sensor datasets: *GENEactiv* [50] and *USC-HAD* [53]. Second, we show image classification results with and without fusing PIs with a *DenseNet* [25] classifier on the following image datasets: *CIFAR10* [26] and *SVHN* [32]. Third, we show how the generated PIs together with the image classification model can help improve robustness to *Gaussian noise*. Finally, we show improvements in computation time for the task of extracting PIs from image databases using *Image PI-Net*.

5.1. Action Recognition using Accelerometer Data

Dataset Description: The *GENEactiv* dataset consists of 29 different human-activity classes from 152 subjects [50]. The data was collected at a sampling rate of 100Hz using the *GENEactiv* sensor, a light-weight, waterproof, wrist-worn tri-axial accelerometer. Interested readers can refer to the following paper to learn about the data collection protocol [50]. The *USC-HAD* dataset consists of 12 different human-activity classes from 14 subjects [53]. Data was collected using a tri-axial *MotionNode* accelerometer sensor at a sampling rate of 100Hz. The sensor was placed at the front right hip on the body. Both datasets were down-sampled to 50Hz and fixed-length non-overlapping frames were extracted. Figures 5 and 6 show the distribution of the different activity classes in each dataset, with each frame having a duration of 5 seconds or 250 time-steps. For the *GENEactiv* dataset we extracted frames with time-steps = 250 and 500, and used approximately 75% of the frames for training and the rest as the test set. *USC-HAD* being a significantly smaller dataset, we only extracted frames with time-step = 250 and used frames from the first 8 subjects for training and the remaining 6 subjects as the test set.

Training Signal PI-Net: The *Signal PI-Net* model was trained using just the training set of the *GENEactiv* dataset. The batch-size was set to 128 and the model was trained for a 1000 epochs. The learning rate for the first 300 epochs, next 300 epochs and final 400 epochs was set to 10^{-3}, 10^{-4} and 10^{-5} respectively. *Adam* optimizer was used and the *Mean-Squared-Error* loss function was used to quantify the deviation of the generated PIs from the ground-truth PIs. Final training and test loss values are tabulated in Table 1.

Data Characterization and Classification: For characterizing the time-series signals, we consider three different feature representations: (**1**) A 19-dimensional feature vector consisting of different statistics calculated over each 10-second frame [50]; (**2**) Features learnt from scratch using multi-layer-perceptron (MLP) models and 1D CNNs; (**3**)

Persistence Images generated using the traditional filtration technique and the proposed *Signal PI-Net* model. The 19-dimensional feature vector includes *mean*, *variance*, *root-mean-square* value of the raw accelerations on each of X, Y and Z axes, *pearson correlation coefficients* between X-Y, Y-Z and X-Z time series, *difference between maximum and minimum accelerations* on each axis denoted by $\sqrt{dx^2 + dy^2}, \sqrt{dy^2 + dz^2}, \sqrt{dx^2 + dz^2}\sqrt{dx^2 + dy^2 + dz^2}$. From here on out we will refer to this 19-dimensional statistical feature as SF.

The MLP classifier contains 8 dense layers, with each layer having 128 units and ReLU activation. To avoid over-fitting, each dense layer is followed by a dropout layer with a dropout rate of 0.2 and a batch-normalization layer. The output layer is another dense layer with Softmax activation and with number of units equal to the number of classes. The 1D CNN classifier consists of 10 CNN layers with number of filters set to 32, kernel size to 3, stride to 1 and the output is zero-padded. Each CNN layer is followed by batch-normalization, ReLU activation and max-pooling layers. For max-pool layers we set the filter size to 3, the stride was set to 1 for every odd layer and 2 for every even layer. For the final CNN layer we use a global-average-pooling layer instead of a max-pool layer. Here too, the output layer consists of a dense layer with softmax activation and number of units equal to number of target classes.

We used the trained *Signal PI-Net* model to extract PIs for the test set of the *GENEactiv* dataset. We also use the same model to compute PIs for both the training and test sets of the *USC-HAD* dataset. The different classification methods are listed in Table 2. The PIs obtained using traditional analytic methods or using the proposed *Signal PI-Net* model were fused with the MLP and 1D CNN classification models differently. For instance, *MLP - PI* and *MLP - Signal PI-Net* use the MLP classifier to learn features directly from the computed PIs (The PIs were vectorized and passed as inputs). *MLP - SF* uses the MLP classifier with the 19-dimensional statistical feature as input. In *MLP - SF+PI* and *MLP - SF+Signal PI-Net* we first concatenate the SF and PI representations before passing them as input to the MLP model. However, for *1D CNN + PI* and *1D CNN + Signal PI-Net* we use a slightly different approach. Using Principal Component Analysis (PCA) we first reduce the vectorized PI representation (7500-dimensional) to a 32-dimensional feature vector. This was done to reduce the number of additional parameters that would result from the concatenation of the PI feature representations to the *1D CNN* model. The 32-dimensional PI representation is then concatenated to the output of the global-average-pool layer in the *1D-CNN* model.

The weighted F1 score classification results for *GENEactiv* and *USC-HAD* is shown in Table 2. For each method we report the mean ± std result over 5 runs. We observe similar results under the different time-step settings in *GENEactiv* and also across the two datasets. PIs computed analytically or using *Signal PI-Net* perform better than SF. Fusing PIs with SF helps significantly improve the classification performance. 1D CNN is a more powerful classifier than MLP, which is made clearly evident from the tabulated results. Fusing PIs with features learnt using 1D CNNs helps marginally improve the overall classification result.

5.2. Image Classification

Dataset Description: We consider the following three datasets in our experiments: *CIFAR10* [26], *CIFAR100* [26] and *SVHN* [32]. *CIFAR10* and *CIFAR100* each contain 50000 images for training and 10000 images for testing, whereas *SVHN* has 73257 images for training and 26032 images for testing. For classification experiments we only show results for *CIFAR10* and *SVHN*. Both datasets have 10 different label categories. Also, the height, width and number of channels for each image is equal to 32, 32 and 3 respectively.

Training Image PI-Net: We develop two kinds of *Image PI-Net* models based on the datasets we chose as source and target datasets: (**1**) In the first kind we set the source and target datasets to be same, *i.e.* we train the *Image PI-Net* model using the *CIFAR10* or *SVHN* dataset. (**2**) For the second type, we use the *CIFAR100* dataset as the source dataset and the target dataset is either *CIFAR10* or *SVHN*. Simply put, we employ transfer learning by first training the *Image PI-Net* model using *CIFAR100* and later use the target dataset to fine-tune the *Image PI-Net* model. For the second case, we further explore two variations: (**2a**) Fine-tune the model using all samples from the training set of the target dataset; (**2b**) fine-tune using just a subset *i.e.* 500 images per class in the training set of the target dataset, to simulate the scenario of having limited training data. We will refer to these variants as *Image PI-Net Fine-tune All* (*Image PI-Net FA*) and *Image PI-Net Fine-tune Subset* (*Image PI-Net FS*) respectively. We explored the above variants to show the use of the proposed *Image PI-Net* model under different scenarios. We set the batch-size to 32. We train the basic *Image PI-Net* model for 415 epochs and set the learning rate for the first 15 epochs, next 200 epochs and final 200 epochs to 10^{-3}, 10^{-5} and 10^{-6} respectively. For *Image PI-Net FA* and *Image PI-Net FS* we first load the weights from the *CIFAR100* pre-trained model and fine-tune the weights for 200 epochs with a learning rate of 10^{-6}. We use the *Adam* optimizer and the *Binary Cross-Entropy* loss function to compile the models. The training and test loss values are tabulated in Table 1.

Data Characterization and Classification: For image classification we use *DenseNet* [25] as our base classification model. PIs alone are not as powerful as features learnt using deep learning frameworks for image classification. However, past research works have shown topological features to carry complementary information that can be exploited to improve the overall performance of a machine learning model [12, 28, 44]. We too show results using *DenseNet* in conjunction with PIs that are generated using traditional filtration techniques and using the proposed *Image PI-Net* model. Figure 7 illustrates how we pass the computed PIs as a secondary input to the base classification network. Our *DenseNet* model has the following specifications: depth = 16, number of dense blocks = 4, number of convolution filters = 16, growth rate = 12, dropout rate = 0.2 and weight decay = 10^{-4}. We pass the generated PIs through a single 2D convolution layer with 32 filters. This is followed by a global-average-pool layer which results in a 32-dimensional feature vector. This feature vector is concatenated with the output of the global-average-pool layer (penultimate layer) of the *DenseNet* model.

The classification results are averaged over three runs and are tabulated in Table 3. We see that fusing PI feature helps improve the overall classification result for the base model on both datasets. PIs generated using the traditional filtration method and the proposed *Image PI-Net* framework achieve similar results. Also, *Image PI-Net FS* being trained on just 500 samples per class, achieves a classification result that is comparable to the other *Image PI-Net* variants. This is useful in cases where there is limited training data for the target task. To check the significance of the different fusion cases we calculate the *P*-value for each case with respect to just the *DenseNet* model. *P*-value is the area of the two-sided *t*-distribution that falls outside ±*t*. We consistently observe a *P*-value of less than 0.05 across all fusion cases. While we only observe marginal improvement in terms of classification accuracy, the advantage of using *PI-Net* with the base classification model is made apparent in the next section.

5.3. Robustness to Gaussian Noise in Images

While data augmentation can help neural networks learn different transforms, TDA methods have the ability to encode different invariances by default. This could help reduce if not completely remove the need for different data variations during the training process. Here we evaluate the robustness of the different *DenseNet* classification models when the test-set images are subjected to Gaussian noise. Note, the classification models were trained using the original training-set images and no data-augmentation was done during the training process. All images were first normalized to lie between [0,1]. For both datasets we apply a zero-mean Gaussian noise and vary the standard deviation to the following levels: 0.02, 0.04, 0.06, 0.08. After applying Gaussian noise we clip the pixel values in the image to lie between [0,1]. We refer to the four increasing severity levels of Gaussian noise as **Level 1, Level 2, Level 3, Level 4** or in short **L1, L2, L3, L4.**

Since computing PDs and PIs using traditional analytic methods is computationally expensive, we were not able to evaluate the *DenseNet + PI* case on all test images. To give some perspective, computing PIs for each severity level on the test-set would take about 10 hours and 24 hours for *CIFAR10* and *SVHN* respectively. More information about the computational complexity is discussed in Section 5.4. To compare all methods we randomly select 500 images from the test set and compare the classification performance. Figure 8 shows the percentage change in the classification performance with respect to the *DenseNet* method in the absence of any Gaussian noise. The effect of Gaussian noise is different for each dataset due to which the *y*-axis is scaled differently. From the bar-plots we see that the overall classification performance decreases as the severity level increases. However, the percentage decrease for *DenseNet + PI* and the different *DenseNet + Image PI-Net* variants is less compared to *DenseNet* alone. Fusing PIs with the *DenseNet* model helps incorporate robustness to different Gaussian noise. We see similar trends between the *500 Test Samples* and *All Test Samples* cases.

5.4. Computation Time to Generate PIs

We used the NVIDIA GeForce GTX Titan Xp graphic card with 12GB memory to train and evaluate all deep learning models. All other tasks were carried out on a standard Intel i7 CPU using Python with a working memory of 32GB. We use the Scikit-TDA software to compute PDs and PIs [39]. Table 4 shows the average time taken to extract PI for one image by conventional TDA methods using one CPU and the proposed *PI-Net* framework on both a

CPU and a GPU. The average is computed over all training images in each dataset. Using the *Image PI-Net* model on a GPU, we see an effective speed up of three orders of magnitude in the computation time. Also, *Image PI-Net* implemented on a CPU is still faster than the analytic method by an order of magnitude. Using a GPU we also check the time taken to compute PIs when the entire training set is passed into *Image PI-Net* as a single batch. It took about 9.77±0.08 seconds for *CIFAR10* and 12.93±0.05 seconds for *SVHN*. This is a fraction of the time compared to the time it takes using conventional TDA tools. So far it had been impossible to compute PIs at real-time using conventional TDA approaches. However, the proposed framework allows us to easily compute PIs in real-time thereby opening doors to new real-time applications for TDA.

6. Conclusion and Future Work

In this paper we took the first step in using deep learning to extract topological feature representations. We developed a simple, effective and differentiable architecture called to extract PIs directly from time-series and image data. *PI-Net* has a significantly lower computational complexity compared to using conventional topological tools. We show improvements in classification performance on two accelerometer and two image datasets. Despite observing marginal improvement in image classification accuracy, the benefit of using *PI-Net* with the base classification network is made apparent through the robustness to Gaussian noise experiment.

For future work we would like to explore more sophisticated deep learning architectures that can allow us to learn mappings between higher dimensional data and their corresponding topological feature representations. We would also like to see how deep learning can be further used to generate other kinds of topological representations and test their robustness to different image deformations like contrast, blur and affine transformations.

Acknowledgments

This work was supported in part by NIH R01GM135927 and NSF CAREER 1452163. Arizona State University's institutional review board approved all study materials and procedures (protocol number 1304009121).

References

[1]. Adams Henry, Emerson Tegan, Kirby Michael, Neville Rachel, Peterson Chris, Shipman Patrick, Chepushtanova Sofya, Hanson Eric, Motta Francis, and Ziegelmeier Lori. Persistence images: A stable vector representation of persistent homology. Journal of Machine Learning Research, 18(8):1–35, 2017.

[2]. Adams Henry, Tausz Andrew, and Vejdemo-Johansson Mikael. Javaplex: A research software package for persistent (co) homology. In International Congress on Mathematical Software, pages 129–136. Springer, 2014.

[3]. Anirudh Rushil, Venkataraman Vinay, Ramamurthy Karthikeyan Natesan, and Turaga Pavan. A riemannian framework for statistical analysis of topological persistence diagrams. In The IEEE Conference on Computer Vision and Pattern Recognition Workshops, pages 68–76, 2016.

[4]. Bauer Ulrich, Kerber Michael, and Reininghaus Jan. Distributed computation of persistent homology. In Proceedings of the Workshop on Algorithm, Engineering and experiments, pages 31–38. SIAM, 2014.

[5]. Bubenik Peter. Statistical topological data analysis using persistence landscapes. The Journal of Machine Learning Research, 16(1):77–102, 2015.

[6]. Bubenik Peter and Holcomb John. Statistical inferences from the topology of complex networks. Technical report, Cleveland State University, Cleveland, United States, 2016.

[7]. Cang Zixuan and Wei Guo-Wei. Topologynet: Topology based deep convolutional and multi-task neural networks for biomolecular property predictions. PLoS Computational Biology, 13(7), 2017.

[8]. Chintakunta Harish, Gentimis Thanos, Gonzalez-Diaz Rocio, Jimenez Maria-Jose, and Krim Hamid. An entropy-based persistence barcode. Pattern Recognition, 48(2):391–401, 2015.

[9]. Chollet François et al. Keras. https://keras.io, 2015.

[10]. Chung Moo K, Bubenik Peter, and Kim Peter T. Persistence diagrams of cortical surface data. In International Conference on Information Processing in Medical Imaging, pages 386–397. Springer, 2009.

[11]. Dabaghian Yuri, Memoli Facundo, Frank Loren, and Carlsson Gunnaŕ. A topological paradigm for hippocampal spatial map formation using persistent homology. PLoS Computational Biology, 8(8):1–14, 2012. [PubMed: 22629235]

[12]. Dey Tamal Krishna, Mandal Sayan, and Varcho William. Improved Image Classification using Topological Persistence. In Vision, Modeling & Visualization The Eurographics Association, 2017.

[13]. Dong Chao, Loy Chen Change, He Kaiming, and Tang Xiaoou. Learning a deep convolutional network for image super-resolution. In European Conference on Computer Vision, pages 184–199. Springer, 2014.

[14]. Edelsbrunner Herbert and Harer John. Computational topology: an introduction. American Mathematical Society, 2010.

[15]. Edelsbrunner Herbert, Letscher David, and Zomorodian Afra. Topological persistence and simplification. Discrete & Computational Geometry, 28(4):511–533, 2002.

[16]. Eigen David, Puhrsch Christian, and Fergus Rob. Depth map prediction from a single image using a multi-scale deep network. In Advances in Neural Information Processing Systems, pages 2366–2374, 2014.

[17]. Ferri Massimo. Why topology for machine learning and knowledge extraction? Machine Learning and Knowledge Extraction, 1(1):115–120, 2018.

[18]. Gabella Maxime, Afambo Nitya, Ebli Stefania, and Spreemann Gard. Topology of learning in artificial neural networks. arXiv preprint arXiv:1902.08160, 2019.

[19]. Gabrielsson Rickard Bruel and Carlsson Gunnar. Exposition¨ and interpretation of the topology of neural networks. arXiv preprint arXiv:1810.03234, 2018.

[20]. Girshick Ross, Donahue Jeff, Darrell Trevor, and Malik Jitendra. Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 580–587, 2014.

[21]. Grill-Spector Kalanit and Malach Rafael. The human visual cortex. Annu. Rev. Neurosci, 27:649–677, 2004. [PubMed: 15217346]

[22]. He Kaiming, Zhang Xiangyu, Ren Shaoqing, and Sun Jian. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 770–778, 2016.

[23]. Heath Kyle, Gelfand Natasha, Ovsjanikov Maks, Aanjaneya Mridul, and Guibas Leonidas J. Image webs: Computing and exploiting connectivity in image collections. In IEEE Conference on Computer Vision and Pattern Recognition, 2010.

[24]. Hofer Christoph, Kwitt Roland, Niethammer Marc, and Uhl Andreas. Deep learning with topological signatures. In Advances in Neural Information Processing Systems, pages 1634–1644. 2017.

[25]. Huang Gao, Liu Zhuang, Van Der Maaten Laurens, and Weinberger Kilian Q. Densely connected convolutional networks. In Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, pages 4700–4708, 2017.

[26]. Krizhevsky Alex and Hinton Geoffrey. Learning multiple layers of features from tiny images. Technical report, Cite-seer, 2009.

[27]. Krizhevsky Alex, Sutskever Ilya, and Hinton Geoffrey E. Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems, pages 1097–1105, 2012.

[28]. Li Chunyuan, Ovsjanikov Maks, and Chazal Frederic. Persistence-based structural recognition. In IEEE Conference on Computer Vision and Pattern Recognition, pages 1995–2002, 2014.

[29]. Liu Weibo, Wang Zidong, Liu Xiaohui, Zeng Nianyin, Liu Yurong, and Alsaadi Fuad E. A survey of deep neural network architectures and their applications. Neurocomputing, 234:11–26, 2017.

[30]. Long Jonathan, Shelhamer Evan, and Darrell Trevor. Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 3431–3440, 2015.

[31]. Nawar Afra, Rahman Farhan, Krishnamurthi Narayanan, Som Anirudh, and Turaga Pavan. Topological descriptors for parkinson's disease classification and regression analysis. arXiv preprint arXiv:2004.07384, 2020.

[32]. Netzer Yuval, Wang Tao, Coates Adam, Bissacco Alessandro, Wu Bo, and Ng Andrew Y. Reading digits in natural images with unsupervised feature learning. 2011.

[33]. Pachauri Deepti, Hinrichs Chris, Chung Moo K, Johnson Sterling C, and Singh Vikas. Topology-based kernels with application to inference problems in alzheimer's disease. IEEE transactions on Medical Imaging, 30(10):1760–1770, 2011. [PubMed: 21536520]

[34]. Perea Jose A and Harer John. Sliding windows and persistence: An application of topological methods to signal analysis. Foundations of Computational Mathematics, 15(3):799–838, 2015.

[35]. Ramamurthy Karthikeyan Natesan, Varshney Kush, and Mody Krishnan. Topological data analysis of decision boundaries with application to model selection. In Proceedings of the International Conference on Machine Learning, pages 5351–5360, 2019.

[36]. Reininghaus Jan, Huber Stefan, Bauer Ulrich, and Kwitt Roland. A stable multi-scale kernel for topological machine learning. In IEEE Conference on Computer Vision and Pattern Recognition, 2015.

[37]. Rieck Bastian, Togninalli Matteo, Bock Christian, Moor Michael, Horn Max, Gumbsch Thomas, and Borgwardt Karsten. Neural persistence: A complexity measure for deep neural networks using algebraic topology. In International Conference on Learning Representations, 2019.

[38]. Rouse David, Watkins Adam, Porter David, Harer John, Bendich Paul, Strawn Nate, Munch Elizabeth, DeSena Jonathan, Clarke Jesse, Gilbert Jeffrey, et al. Feature-aided multiple hypothesis tracking using topological and statistical behavior classifiers. In SPIE Defense +Security, 2015.

[39]. Saul Nathaniel and Tralie Chris. Scikit-TDA: Topological data analysis for python. 10.5281/zenodo.2533369, 2019.

[40]. Silver David, Huang Aja, Maddison Chris J, Guez Arthur, Sifre Laurent, Van Den Driessche George, Schrittwieser Julian, Antonoglou Ioannis, Panneershelvam Veda, Lanctot Marc, et al. Mastering the game of go with deep neural networks and tree search. nature, 529(7587):484, 2016. [PubMed: 26819042]

[41]. Simonyan Karen and Zisserman Andrew. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

[42]. Singh Gurjeet, Memoli Facundo, Ishkhanov Tigran, Sapiro Guillermo, Carlsson Gunnar, and Ringach Dario L. Topological analysis of population activity in visual cortex. Journal of Vision, 2008.

[43]. Som Anirudh, Ramamurthy Karthikeyan Natesan, and Turaga Pavan. Geometric metrics for topological representations. Handbook of Variational Methods for Nonlinear Geometric Data, page 415.

[44]. Som Anirudh, Thopalli Kowshik, Ramamurthy Karthikeyan Natesan, Venkataraman Vinay, Shukla Ankita, and Turaga Pavan. Perturbation robust representations of topological persistence diagrams. In Proceedings of the European Conference on Computer Vision, pages 617–635, 2018.

[45]. Srinivas Suraj, Sarvadevabhatla Ravi Kiran, Mopuri Konda Reddy, Prabhu Nikita, Kruthiventi Srinivas SS, and Babu R Venkatesh. A taxonomy of deep convolutional neural nets for computer vision. Frontiers in Robotics and AI, 2:36, 2016.

[46]. Szegedy Christian, Liu Wei, Jia Yangqing, Sermanet Pierre, Reed Scott, Anguelov Dragomir, Erhan Dumitru, Vanhoucke Vincent, and Rabinovich Andrew. Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1–9, 2015.

[47]. Tralie Christopher J and Perea Jose A. (quasi) periodicity quantification in video data, using topology. SIAM Journal on Imaging Sciences, 11(2):1049–1077, 2018.

[48]. Venkataraman Vinay, Ramamurthy Karthikeyan Natesan, and Turaga Pavan. Persistent homology of attractors for action recognition. In IEEE International Conference on Image Processing, pages 4150–4154. IEEE, 2016.

[49]. Walker Jacob, Gupta Abhinav, and Hebert Martial. Dense optical flow prediction from a static image. In Proceedings of the IEEE International Conference on Computer Vision, pages 2443–2451, 2015.

[50]. Wang Qiao, Lohit Suhas, Toledo Meynard John, Buman Matthew P, and Turaga Pavan. A statistical estimation framework for energy expenditure of physical activities from a wrist-worn accelerometer. In Annual International Conference of the IEEE Engineering in Medicine and Biology Society, pages 2631–2635. IEEE, 2016.

[51]. Wang Xiaolong, Fouhey David, and Gupta Abhinav. Designing deep networks for surface normal estimation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 539–547, 2015.

[52]. Yosinski Jason, Clune Jeff, Bengio Yoshua, and Lipson Hod. How transferable are features in deep neural networks? In Advances in Neural Information Processing Systems, pages 3320–3328, 2014.

[53]. Zhang Mi and Sawchuk Alexander A. USC-HAD: a daily activity dataset for ubiquitous activity recognition using wearable sensors. In Proceedings of the ACM Conference on Ubiquitous Computing, pages 1036–1043. ACM, 2012.

[54]. Zhang Ning, Donahue Jeff, Girshick Ross, and Darrell Trevor. Part-based R-CNNs for fine-grained category detection. In European Conference on Computer Vision, pages 834–849. Springer, 2014.

[55]. Zhang Ning, Paluri Manohar, Ranzato Marc'Aurelio, Darrell Trevor, and Bourdev Lubomir. Panda: Pose aligned networks for deep attribute modeling. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1637–1644, 2014.

[56]. Zhou Bolei, Lapedriza Agata, Xiao Jianxiong, Torralba Antonio, and Oliva Aude. Learning deep features for scene recognition using places database. In Advances in Neural Information Processing Systems, pages 487–495, 2014.

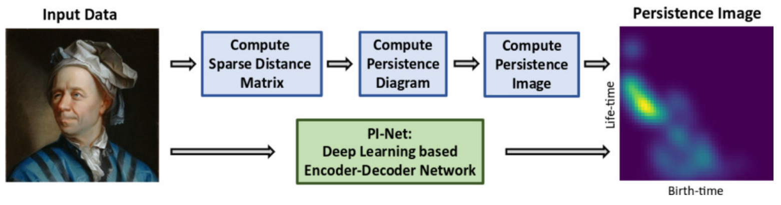
Input Data
Compute Sparse Distance Matrix
Compute Persistence Diagram
Compute Persistence Image
Persistence Image
PI-Net: Deep Learning based Encoder-Decoder Network
Life-time
Birth-time

Figure 1.

Illustration of the proposed PI-Net model to directly compute persistence images from input data. Traditional analytic methods (illustrated in the top half of the figure) consist of a sequence of steps that are computationally expensive.

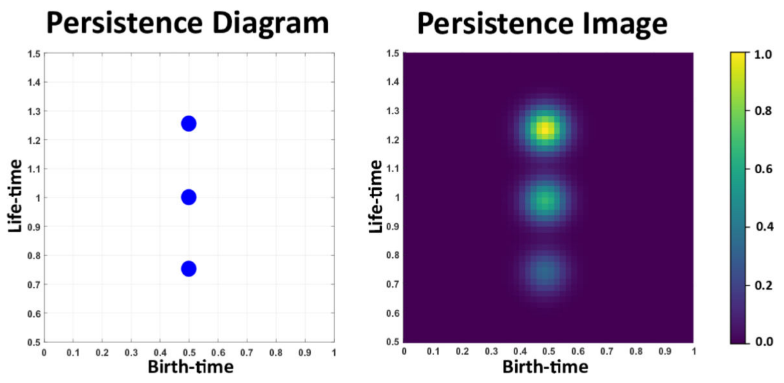

Figure 2.

Illustration of a PD and its weighted PI for three points with same birth-time but different life-time. Due to the weighting function points with higher life-time appear more brighter.

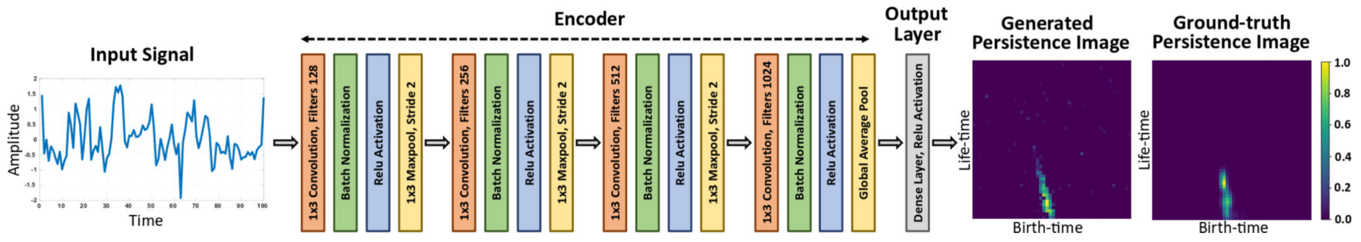
Input Signal
Amplitude
Time
Encoder
1x3 Convolution, Filters 128
Batch Normalization
Relu Activation
1x3 Maxpool, Stride 2
1x3 Convolution, Filters 256
1x3 Convolution, Filters 512
1x3 Convolution, Filters 1024
Global Average Pool
Output Layer
Dense Layer, Relu Activation
Generated Persistence Image
Ground-truth Persistence Image
Life-time
Birth-time

Figure 3.
Illustration of Signal PI-Net for computing PIs directly from multi-variate time-series signals.

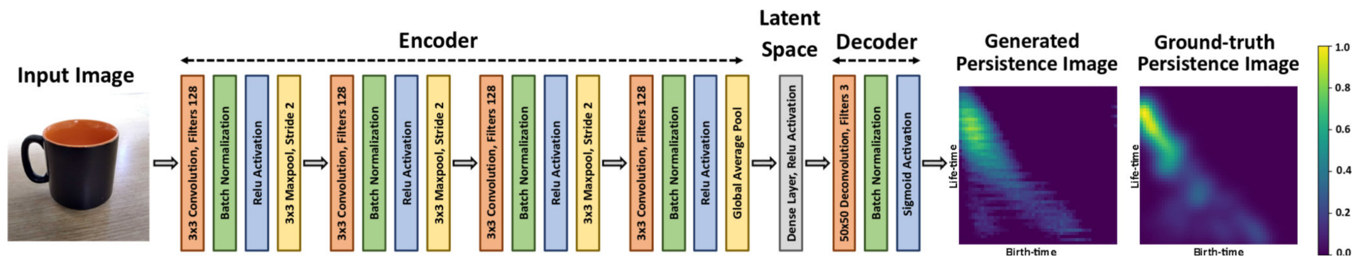

Figure 4.
Illustration of Image PI-Net for computing PIs directly from multi-channel image data.

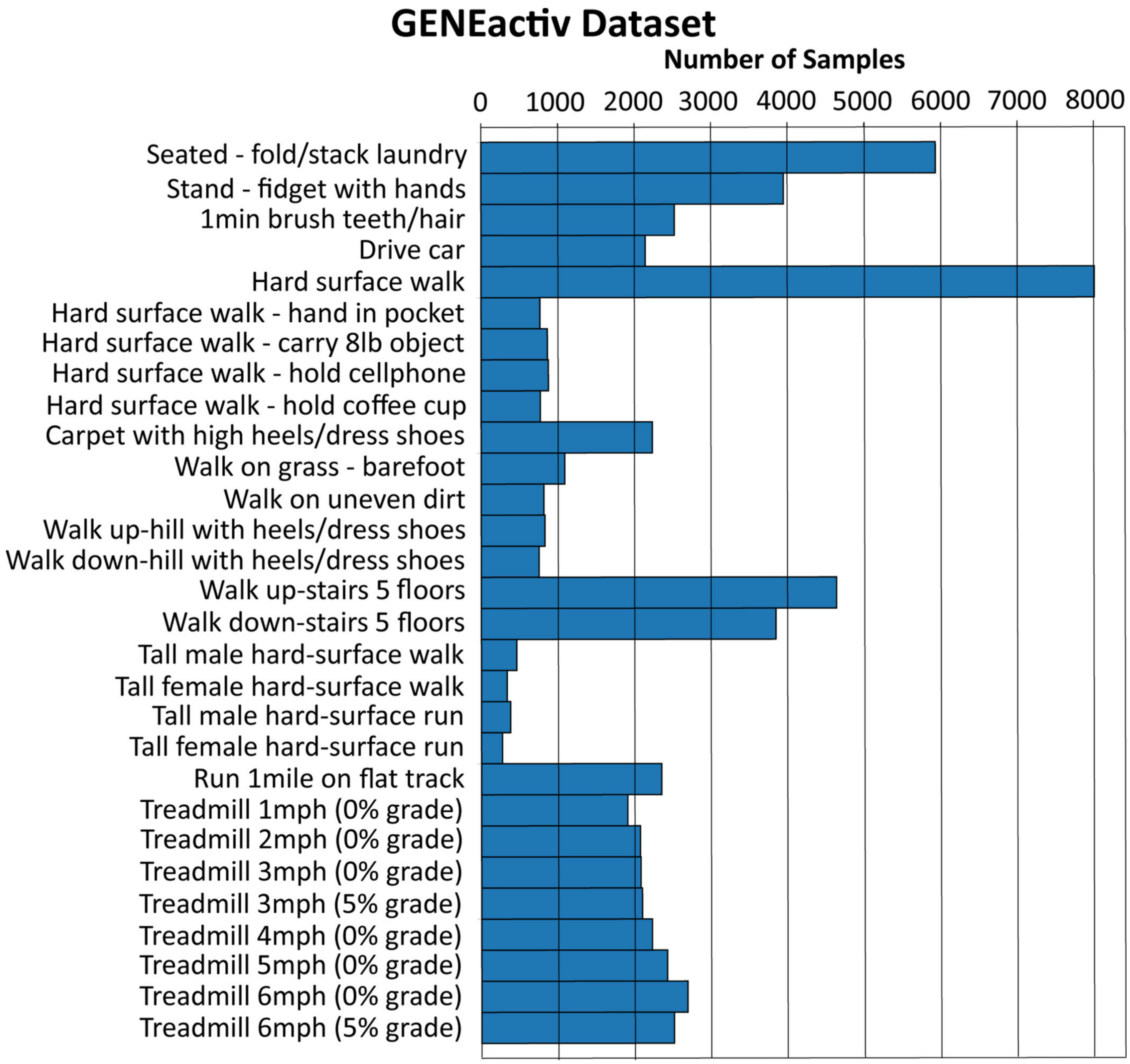

Figure 5.

Distribution of activity classes in the *GENEactiv* dataset for time-step length = 250.

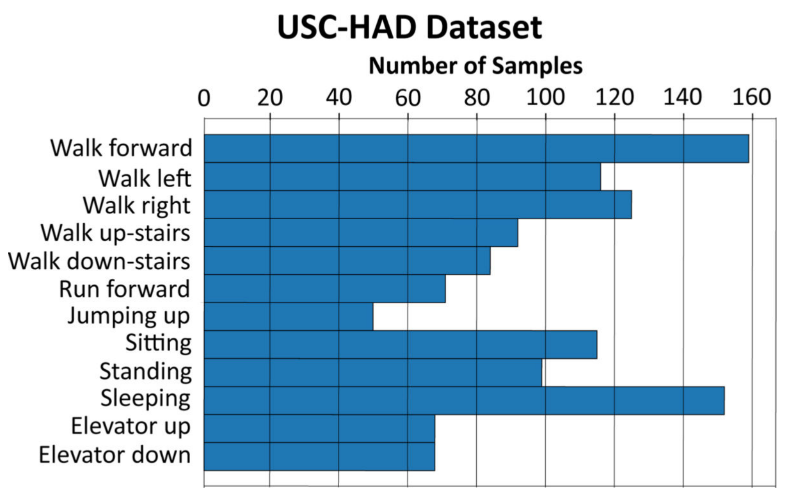

Figure 6.

Distribution of activity classes in the *USC-HAD* dataset for time-step length = 250.

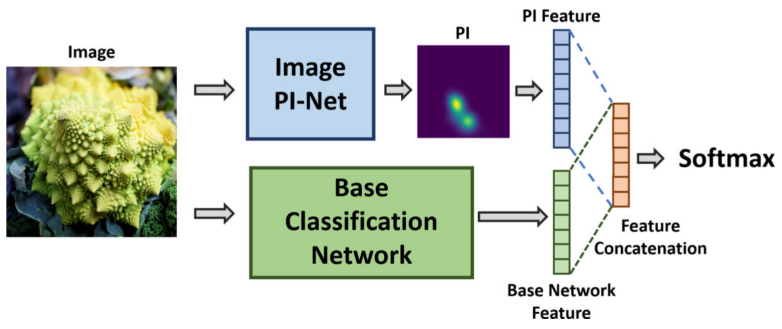

Figure 7.
Illustration of the modified base model where we concatenate PI feature with features learnt using the base classification network.

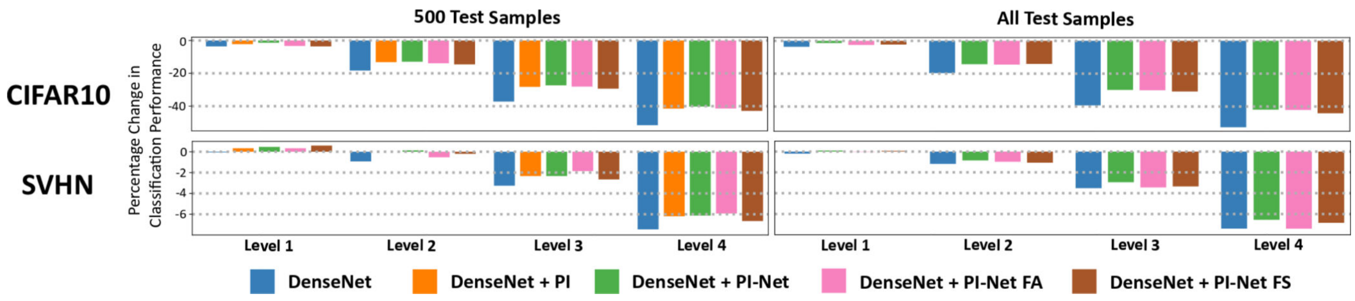

Figure 8.

Percentage point drop in the classification performance on CIFAR10 (top) and SVHN (bottom) as the Gaussian noise severity increases. The percentage drop is calculated with respect to the classification performance of the DenseNet model in the absence of any Gaussian noise. Without noise, the DenseNet classification performance for 500 Test Samples and All Test Samples for CIFAR10 is 82.93% and 83.80%, and for SVHN is 96.06% and 95.65%. While the performance of all models drop as degradation increases, the drop of topological fusion models is less compared to just the DenseNet model. Note, the *y*-axis is scaled different for each dataset.

Table 1.

Final train and test loss values after training the different Signal PI-Net and Image PI-Net models.

PI-Net	Train Loss	Test Loss
Signal PI-Net Time-steps = 250	0.00159	0.00158
Signal PI-Net Time-steps = 500	0.00187	0.00187
CIFAR10 Image PI-Net	1.99793	2.06193
CIFAR10 Image PI-Net FA	2.02095	2.04441
CIFAR10 Image PI-Net FS	0.51734	0.52560
SVHN Image PI-Net	1.53533	1.51732
SVHN Image PI-Net FA	1.57923	1.54195
SVHN Image PI-Net FS	0.41519	0.40955

Table 2.

Weighted F1 score classification results for the GENEactiv and USC-HAD datasets. The mean ± std values were calculated over five runs.

Method	GENEactiv		USC-HAD
	Time-steps = 250	**Time-steps = 500**	**Time-steps = 250**
MLP - PI	46.45±0.32	49.67±0.63	43.21±0.66
MLP - Signal PI-Net	**49.47±0.69**	**53.69±1.08**	**48.15±0.67**
MLP - SF	41.70±0.41	42.01±0.42	35.68±0.11
MLP - SF + PI	48.57±0.37	49.82±0.62	44.31±0.36
MLP - SF + Signal PI-Net	**50.66±0.78**	**54.44±0.80**	**48.97±0.30**
1D CNN	53.56±0.31	54.97±1.35	54.58±0.64
1D CNN + PI	54.28±0.23	56.38±0.23	54.64±0.62
1D CNN + Signal PI-Net	**54.41±0.21**	**56.41±0.22**	**57.82±0.78**

Table 3.

Image classification accuracy results for CIFAR10 and SVHN datasets, with the mean ± std values calculated over three runs. *P*-value is calculated with respect to the base DenseNet model.

Method	**CIFAR10**		**SVHN**	
	Mean±SD	**p-Value**	**Mean±SD**	**p-Value**
DenseNet	83.80±0.12	-	95.65±0.00	-
DenseNet + PI	**84.37±0.21**	0.0153	**95.86±0.01**	<0.0001
DenseNet + Image PI-Net	**84.82±0.19**	0.0160	**95.95±0.08**	0.0038
DenseNet + Image PI-Net FA	**84.69±0.38**	0.0195	**95.84±0.06**	0.0063
DenseNet + Image PI-Net FS	**84.59±0.17**	0.0032	**95.95±0.07**	0.0020

Table 4.

Comparison of the average time taken to compute PIs for one image using conventional TDA tools and the proposed PI-Net model. The time reported is averaged over all images present in. the training set of each dataset.

Method	**Time (10^{-3} seconds)**	
	CIFAR10 (50000 images)	**SVHN (73257 images)**
Conventional TDA - CPU	3567.29±867.74	3433.06±622.21
PI-Net - CPU	**125.45±5.30**	**125.49±5.34**
PI-Net - GPU	**2.52±0.02**	**2.19±0.02**