

## Research Article

# Few-Shot Learning for Image-Based Nonintrusive Appliance Signal Recognition

L. Matindife <sup>1</sup>, Y. Sun <sup>1</sup>, and Z. Wang <sup>2</sup>

<sup>1</sup>Department of Electrical and Electronic Engineering Science, University of Johannesburg, Auckland Park 2006, South Africa

<sup>2</sup>Department of Electrical Engineering, University of South Africa, Florida 1710, South Africa

Correspondence should be addressed to Y. Sun; sunyanxia@gmail.com

Received 30 March 2022; Revised 19 July 2022; Accepted 29 July 2022; Published 23 August 2022

Academic Editor: Simone Ranaldi

Copyright © 2022 L. Matindife et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

In this article, we present the recognition of nonintrusive disaggregated appliance signals through a reduced dataset computer vision deep learning approach. Deep learning data requirements are costly in terms of acquisition time, storage memory requirements, computation time, and dynamic memory usage. We develop our recognition strategy on Siamese and prototypical reduced data few-shot classification algorithms. Siamese networks address the 1-shot recognition well. Appliance activation periods vary considerably, and this can result in imbalance in the number of appliance-specific generated signal images. Prototypical networks address the problem of data imbalance in training. By first carrying out a similarity test on the entire dataset, we establish the quality of our data before input into the deep learning algorithms. The results give acceptable performance and show the promise of few-shot learning in recognizing appliances in the nonintrusive load-monitoring scheme for very limited data samples.

## 1. Introduction

The nonintrusive load monitoring (NILM) [1–4] has achieved high automatic recognition of appliances' operational status, through the measurement of the complex signal from a single point on the mains supplying the building. Today, a number of issues attribute to the successful implementation of NILM appliance recognition systems. These issues include higher data acquisition throughput and more storage hardware, better simulation software, better imbedded implementation hardware, and the use of deep learning (DL) algorithms. Machine-learning (ML) algorithms premised on hand-engineered features achieve acceptable performance when the data count is relatively low. DL algorithms inherently achieve better feature extraction, and usually, the data count is very high. The performance of DL algorithms greatly outperforms that of the rest of hand-engineered algorithms. As a way of increasing the data count, data augmentation methods [5, 6] normally complement data obtained from direct measurement. Data processing in MILM systems is either time series

(TS) [2] or the image (IM) [7, 8] equivalent of the appliance TS signals. The IM data approach aims at availing more appliance features in a smaller space for improved but simpler identification through convolutional neural network (CNN) computer vision (CV). We can improve the feature base of the IM dataset to mimic a larger dataset by implementing multivariate IMs, information, and IM fusion inputs into the DL algorithms [9, 10]. However, the cost of acquiring large amounts of data becomes high, mainly in terms of increased data acquisition time and increased storage memory requirements. Computation time and dynamic memory usage become higher during model execution in such situations. In addition, if used, data augmentation and fusion inherently add to the complexity of the IM preprocessing stages.

Few-shot learning (FSL) [11, 12] allows the successful implementation of ML recognition algorithms on very limited input datasets. In FSL, ML algorithms mimic the ability of humans to identify an object in a different or new situation, based only on minimal or no prior interaction with that object [11]. The ability to learn-to-learn (also

known as meta-learning (MEL)) from a previous situation makes it possible to achieve this type of recognition capability. Thrun & Pratt [13] give a detailed description of the learn-to-learn process and the expected outcomes of a machine or algorithm that can learn-to-learn. MEL is achieved by using two algorithm approaches, namely, metric [11, 14–16] and gradient-descent [17, 18] learning approaches.

In literature, usually to evaluate the effectiveness of FSL algorithms, we have a comparison to Baseline and Baseline++ models. The building of Baseline models is through a normal transfer learning classification approach. Baseline++ models are an improvement on the standard Baseline models [14, 16, 19]. The Bayesian,  $k$ -nearest neighbours (kNN), and the Siamese network [20, 21] are successful early one-shot learning attempts to classify IMs. The Bayesian method learns the relationship between inputs by using a probabilistic approach to relate the attributes of these inputs. In KNN ( $K=1$ ), the algorithm maps the feature space for two input IMs such that any new input IM outcome is determined by its nearest neighbour. The MEL approach can considerably enhance the performance of Bayesian, KNN, and Siamese classification networks.

There has been successful application of FSL in areas such as robotics, natural language processing, acoustic signal processing, drug recovery, and CV [12]. However, there is scant documentation of FSL as specifically applied to NILM classification [20–22]. The difficulty in realizing one-shot classification has slowed its adoption in NILM systems [23]. Nonetheless, we show some FSL literature developments in NILM. The authors in Ref. [22] proposed the classification of a number of appliance signals using FSL. In Ref. [22], the authors make a comparison of the few-shot performance of the KNN, decision trees (DT), random forest (RF), and long short-term memory (LSTM) models. The models gave  $F1$ -scores that varied from 0.898 to 0.930, which is an assessment of a model’s accuracy on a given dataset. Moreover, the algorithms are not MEL and use power series (PS) appliance signal lengths determined by a sliding window to capture the minimum appliance activations. In Ref. [20], the authors proposed the Siamese neural network for classifying V/I trajectory images. Training of the Siamese networks is based on one-shot pairs of the same and different label V/I trajectory IMs. Similar appliances belonging to the training set form a cluster, with unrecognized appliances forming their own new cluster. The density-based clustering of appliances with noise (DBSCAN) technique provides improved clustering [20]. However, there is still need to improve on the classification performance of the system in [20], as some appliances are not recognized.

In this article, we propose the development of metric-based Siamese and prototypical FSL algorithms for the classification of the limited disaggregated appliance signal images in the NILM recognition. Contrary to the method in Ref. [22], we attack the FSL from a MEL CV perspective to improve the appliance signals’ classification performance. We obtain a very limited in-house input dataset for the intended experiment from fourteen PS appliance operational status signals transformed into the signal IM equivalent form using

Gramian angular summation fields (GASFs) [7, 8]. The fourteen appliances considered in this article are made up of four light-emitting-diode (LED) mains lamps, two compact fluorescent (CFL) mains lamps, three modes of HP laptop operation, a refrigerator, a microwave oven, a desktop computer, a two-plate cooking stove, and a kettle. The contributions of this article are as follows:

- (i) The development of high NILM appliance classification Siamese and prototypical FSL algorithms based on CV. This results in a reduced dataset to as low as one appliance signal sample per class (one-shot) that effectively eliminates the negative voluminous data-related issues to NILM classification systems.
- (ii) To establish the level of closeness between data samples by carrying out a similarity test on the entire dataset. The similarity test value should be  $S_{TV} \geq 0.6$ . A lower similarity value would require data pre-processing. We arrive at a value of 0.6 by attempting to have data that are easily separable at first sight, leaving the extra-involved 0.4 separation to a better-designed metric-learning network structure.

We organized the remaining parts of the paper as follows. In Section 2, we present the similarity, loss functions, and meta-learning theory. Section 3 gives a presentation and detailed design of the proposed system. We also explain how the data are organized. In Section 4, we present and give a discussion of the experimental results. Section 5 gives a closure to the article through the conclusion.

## 2. Similarity Theory, Loss Functions, and Meta-learning

*2.1. Similarity Theory.* Standard ML method classifies objects by assigning a probability or class value to the object in relation to the known class labels. The ML algorithms sample a large number of labeled objects to be able to achieve a good classification. In contrast, the ML similarity approach assesses the level of similarity between two objects to show whether they belong to the same class.

*Definition 1.* Two sets  $X$  and  $Y$  give a Cartesian product between them of  $X \times Y = \{(x, y): x \in X \text{ and } y \in Y\}$ . If  $X = Y$ , the Cartesian product of  $X$  with itself is  $X \times X = X^2$ . A similarity measure  $S$  [24] is a function with nonnegative real values defined on the Cartesian product  $X \times X$ :

$$S: X \times X \longrightarrow \mathcal{R}, \quad (1)$$

such that the following three properties are satisfied:

- (1)  $\exists s_0 \in \mathcal{R}: -\infty < S(x, y) \leq s_0 < +\infty, \quad \forall x, y \in X$
- (2)  $s(x, x) = s_0, \quad \forall x \in X$
- (3)  $s(x, y) = s(y, x), \quad \forall x, y \in X$

If in addition,

- (1)  $s(x, y) = s_0 \leftrightarrow x = y$
- (2)  $s(x, y)s(y, z) \leq [s(x, y) + s(y, z)]s(x, z) \forall x, y, z \in X.$

The  $S$  is called a metric similarity measure [24].

The aim of metric similarity measure learning is to decrease the separation between the embedding points of similar inputs. To evaluate the similarity between objects, various similarity measures exist [25]. These include the Euclidean distance, the Pearson's correlation for time series data, the Mahalanobis distance, which is a variation in the Euclidean distance with correlation, Dynamic Time Warping for time series comparison, cosine distance, Jaccard, and Tanimoto similarity measures [25]. For discrete systems, the similarity measures include the Jaccard index, Sorenesen coefficient, and the symmetric difference [24]. The most popular distance metric in ML is the Euclidean. The separation between the embedding of points of dissimilar inputs is to be increased. If  $\{x_i\} \in \mathbb{R}^n$  is a number of points, then for similar points  $x_i$  and  $x_j$  to satisfy similarity [26],

$$S: (x_i, x_j) \in S. \quad (2)$$

The distance learning function in (2) is able to bring similar points together and dissimilar points apart in the embedding space:

$$d(x, y) = d_A(x, y) = |x - y_A| = \sqrt{(x - y)^T A (x - y)}, \quad (3)$$

where  $A$  is an optimum matrix. For metric learning,  $A$  is semi-positive definite,  $A \geq 0$ , and when  $A = I$ , we obtain the Euclidean distance [25, 26].

**2.2. Loss Functions.** The constructive loss function is well suited to metric learning. This function works on pairwise data samples and optimizes the training based on closeness or the absence of it between the samples. Let  $d_W(x, y) = D_W$  be the parameterized Euclidean distance between the outputs embedding of  $x_i, x_j$  as defined in (3). Then, the contrastive loss function is

$$\% L_{\text{contrastive}} = (1 - Y) \frac{1}{2} (D_W)^2 + (Y) \frac{1}{2} \max(0, m - D_W)^2, \quad (4)$$

where  $m > 0$  is a margin. The margin is the radius encapsulating the embedding area, such that dissimilar samples will only contribute to the loss function if the metric distance is within the margin.  $Y$  is a binary indicator for the samples. As an example for a pair of similar inputs, this value is 0 and, for two dissimilar inputs, it is 1 [27]. In other words, the first part of (3) deals with similar points, while the last part of the same equation deals with dissimilar points.

The triplet loss (TL) [28] is another widely used metric loss function mainly in Siamese learning. In this particular case, we identify three input images (an Anchor (AC), a Positive (PO), and a Negative (NE)) each passed through one of three CNN shared weights parallel models (Siamese network) with the three embedding models concatenated. The TL attempts to bring the embedding of the AC and PO closer, while it pushes further apart those of the AC and NE. The distance between the AC encoding ( $f(AC)$ ) and the PO encoding ( $f(PO)$ ) is

$$d(AC, PO) = \|f(AC) - f(PO)\|^2. \quad (5)$$

The distance between the AC encoding ( $f(AC)$ ) and the NE encoding ( $f(NE)$ ) is

$$d(AC, NE) = f(AC) - f(NE)^2. \quad (6)$$

The aim is to have  $d(AC, PO) \leq d(AC, NE)$ , that is,

$$d(AC, PO) - d(AC, NE) \leq 0. \quad (7)$$

To avoid a trivial solution for (6) in which case the embedding would be equal, it is necessary to incorporate a hyperparameter margin  $\alpha$  as shown in the following equation:

$$d(AC, PO) - d(AC, NE) + \alpha \leq 0. \quad (8)$$

The margin makes sure that there is an appreciable separation between  $d(AC, PO)$  and  $d(AC, NE)$ . The TL is

$$L(AC, PO, NE) = \max(d(AC, PO) - d(AC, NE) + \alpha, 0). \quad (9)$$

**2.3. Meta- and Few-Shot Learning.** The meta-learning system works by training a large number of unrelated tasks. Each training task learns to classify images in a query set from the support set of that task (each task has its own support set and query set images. However, all tasks have the same classes and samples in the support set. The query set has the same number of samples across tasks. The images in each task are different). The test task that contains entries completely different from the training tasks would have learned a way of classifying the query test set from the support test set. The generation of a large number of training tasks (to optimize the training model) can only be achieved from datasets that have a large number of classes and relatively few samples per class. In few-shot learning, two popular datasets contain a very large class count to meet the requirements of training few-shot models. The first is the Omniglot dataset comprising 50 alphabets with varying hand-written character (class) numbers each and having 20 samples per character for a total of 1623 characters (classes) [14, 29]. The class count in the Omniglot dataset is high, but the samples per class are few. On the other hand, the Modified National Institute of Standards (MNIST) dataset used as a baseline for testing image ML algorithms has only 10 classes but many samples per class. The second is the miniImagenet dataset that uses 100 image classes divided into 80 training and 20 testing samples [14]. Each class in the miniImagenet has 600 samples. The authors in [23] used the few-shot method on the full Imagenet dataset for 1000 classes to achieve high accuracy, having just a few samples per class that varied from 1 to 3. Although these datasets provide a baseline for developing and testing successful few-shot algorithms, in this article, we have produced a more applicable in-house NILM dataset. As in Ref. [23], only in respect to the number of samples per class, our in-house NILM dataset is processed in three ways: (1) 14 (Way)  $\times$  3 (Shot), (2) 14 (Way)  $\times$  2 (Shot), and (3) 14 (Way)  $\times$  1 (Shot). We evaluate and test the

performance of our system on these three data presentations; however, our ultimate goal is the 1 (Shot) model since this allows for the minimum possible data sample without considering the zero-shot scenario.

With  $N$  (Way) classes in the support set and each class having  $K$  (Shot) images for a total of  $N \times K$  support set images, we aim to classify an image out of  $Q$  images in the query set. The classification problem is one-shot, three-shot, or five-shot when the value of  $K$  is one, three, or five, respectively. In few-shot learning, the dataset samples ( $K$ ) are usually less than ten samples. A special case arises when  $K$  is zero (zero-shot learning (ZSL)). ZSL first learns a projection of labeled (train) data into a new feature space. It then places projections of unseen (test) data into the same feature space and evaluates the distance or similarity value between the train and test entries to establish their relationship [30, 31]. Few-shot is an inductive transfer learning process where we optimize a new task based on previous knowledge about a different task with the same underlying structure. Metric-learning algorithms that include the MatchingNet [14], ProtoNet [15], and RelationNet [16] evaluate the distance or similarity function between images. By so doing, the algorithms can group images together that have smaller distance functions between them.

**2.3.1. Siamese Network.** The Siamese network comprises single-input two-parallel-shared weight CNN networks that are both connected to the same distance function block that in-turn connects to a loss function block. The output of each CNN network before the distance function block is a vector space containing the features or embedding of each input. The similarity between the input embedding points is evaluated in the distance function block through the  $L2$  norm (Euclidean distance)  $\|\mathbf{x} - \mathbf{y}_2\|$ ,  $L1$  norm  $\|\mathbf{x} - \mathbf{y}_1\|$  or cosine similarity  $\cos(\mathbf{x}, \mathbf{y})$ . The loss function implements the contrastive or triplet loss-based model optimization during training. The Siamese network is most appropriate for one-shot learning [20, 21].

**2.3.2. Matching Network.** The matching network uses two different functions  $g_\theta$  and  $f_\theta$  to extract the embedding of the support and query sets, respectively. The cosine similarity function compares each support set image features (embedding) to the query set features, followed by softmax classification. Full context embedding (FCE) through LSTM networks allows the production of an embedding that is the resultant of all the support set image features. FCE improves the performance of the MatchingNet especially in complicated situations [14]. In (9), we show the relationship between the query test sample  $\hat{x}$ , and query predicted label ( $\hat{y}$ ) from classification as [11, 14]

$$P(\hat{y} | \hat{x}, S) = \sum_{i=1}^k a(\hat{x} | x_i) y_i, \quad (10)$$

where  $k$  is number of support set samples,  $x_i, y_i$  represent the support set object-label pairs,  $S = \{(x_i, y_i)\}_{i=1}^k$ , and  $a$  is the attention mechanism. The attention mechanism  $a(\cdot, \cdot)$

chooses the most significant attributes in evaluating the similarity in embedding points.

**2.3.3. Prototypical Networks.** This is a less complex metric-learning algorithm that is capable of higher performance than matching networks. In this algorithm, we first find the prototype ( $c_k$ ) mean class of every object in that class. Secondly, we realize the softmax classification of the test object (query) by establishing the Euclidean distance between the query and prototype embedding [15].

The calculation of the prototype point is as follows:

$$c_k = \frac{1}{|S_k|} \sum_{(x_i, y_i) \in S_k} f_\varnothing(x_i), \quad (11)$$

where  $S_k$  represents the  $k$  class support point,  $x_i$  is feature point with label  $y_i$ , and  $f_\varnothing$  is the embedding function having  $\varnothing$  trainable parameters. The evaluation of the query where  $d$  is the Euclidean distance between the query and prototype class is as follows [15]:

$$p_\varnothing(y = k|X) = \frac{\exp(-d(f_\varnothing(X), C_k))}{\sum_k \exp(-d(f_\varnothing(X), C_k))}. \quad (12)$$

A bigger training class count than that for testing normally achieves better results, but maintains the class samples the same in both training and testing situations [11]. The training episode for the negative log-probability  $\mathcal{F}_\varnothing = -\log p_\varnothing(y = k|X)$  through SGD where  $k$  is the true class [15] is given in Algorithm 1.

**2.3.4. Relation Network.** In this model, there is concatenation of the support set ( $f_\varphi(x_i)$ ) and query set ( $f_\varphi(x_j)$ ) feature maps produced by the same embedding function. The function that concatenates the feature maps is  $\mathcal{C}(f_\varphi(x_i), f_\varphi(x_j))$ . The concatenated result is processed in the relation module to output a similarity measure (relation score) between  $x_i$  and  $x_j$  of value 0 to 1. The number of relation scores depends on the number of classes in the support set. Equation (12) shows the expression for the relation score [16]:

$$r_{i,j} = g_\varnothing(\mathcal{C}(f_\varphi(x_i), f_\varphi(x_j))), i = 1, 2, \dots, \mathcal{C}, \quad (13)$$

where  $\mathcal{C}$  is number of classes in support set,  $x_i$  is support set objects, and  $x_j$  is query set entry during model training.

**2.3.5. Model-Agnostic Meta-Learning (MAML).** The MAML [18, 32] is unique among meta-learning methods since it is implementable on any gradient-descent model. To address the meta-learning problem, if the MAML model successfully solves a previous task, then it should learn to deal with a new task in a faster way with improved performance. The MAML seeks to have a learnable parameter  $\theta$  move close to the optimized  $\theta_i^*$  parameter values of different tasks [18]. This  $\theta$  becomes the initialization value, which is specific task fine-tuned. The  $\theta$  trajectory involves the continuous optimization of the loss functions  $L_i$  for the tasks [18]. We define a task as  $T_i = \{p_i(x), p_i(y|x), L_i\}$  that shows the distribution over the

```

Input: Training set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$ , where each  $y_i \in \{1, \dots, K\}$ .  $D_k$  denotes the subset of  $D$  containing all elements  $(x_i, y_i)$  such that  $y_i = k$ .
Output: The loss  $\mathcal{F}$  for a randomly generated training episode.
 $V \leftarrow \text{RANDOMSAMPLE}(\{1, \dots, K\}, N_C) \triangleright$  Select class indices for episode
for  $k_i$  in  $\{1, \dots, N_C\}$  do
   $S_k \leftarrow \text{RANDOMSAMPLE}(\mathcal{D}_{V_k}, N_S) \triangleright$  Select support examples
   $Q_k \leftarrow \text{RANDOMSAMPLE}(\mathcal{D}_{V_k}/S_k, N_Q) \triangleright$  Select query examples
   $c_k \leftarrow 1/N_C \sum_{(x_i, y_i) \in S_k} f_{\emptyset}(x_i) \triangleright$  Compute prototype from support examples
end for
 $\mathcal{F} \leftarrow 0 \triangleright$  Initialize loss
for  $k$  in  $\{1, \dots, N_C\}$  do
  for  $(x, y)$  in  $Q_k$  do
     $\mathcal{F} \leftarrow \mathcal{F} + 1/N_C N_Q [d(f_{\emptyset}(x), c_k) + \log \sum_{k_i} \exp(-d(f_{\emptyset}(x), c_{k_i}))] \triangleright$  Update loss
  end for
end for

```

ALGORITHM 1: [15] Prototypical single training episode loss computation.  $N$  gives the number of training set samples,  $K$  gives the number of classes in the training set,  $N_C \leq K$  is the number of classes per episode,  $N_S$  is the number of support examples per class, and  $N_Q$  is the number of query examples per class.  $\text{RANDOMSAMPLE}(S; N)$  denotes a set of  $N$  elements chosen uniformly at random from set  $S$ , without replacement [15].

input  $p_i(x)$ , the distribution over the labels given the input  $p_i(y|x)$ , and the loss  $L_i$ . The distribution of tasks is  $p(T)$ . If  $f_{\theta}$  represents the classification model, then the training set loss is

$$\min_{\theta} \sum_{T_i \sim p(T)} L_{T_i}(f_{\theta}). \quad (14)$$

The gradient descent optimizes the loss as

$$\min_{\theta} \sum_{T_i \sim p(T)} \nabla_{\theta} L_{T_i}(f_{\theta}). \quad (15)$$

When the learning rate is  $\alpha$ , the complete gradient-descent update is

$$\theta'_i = \theta - \alpha \nabla_{\theta} L_{T_i}(f_{\theta}). \quad (16)$$

Training of the model to minimize  $L_{T_i}(f_{\theta'_i})$  then follows in the meta-objective as

$$\% \min_{\theta} \sum_{T_i \sim p(T)} \nabla_{\theta} L_{T_i}(f_{\theta'_i}) = \sum_{T_i \sim p(T)} L_{T_i}(f_{\theta} - \alpha \nabla_{\theta} L_{T_i}(f_{\theta})). \quad (17)$$

Covering all the tasks, the stochastic gradient descent (SGD) updates the meta-optimization parameter  $\theta$  as

$$\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{T_i \sim p(T)} L_{T_i} f(\theta_i), \quad (18)$$

where  $\beta$  is the meta step size [18–33].

### 3. Methodology

**3.1. Proposed System.** Based on literature review, compared to other metric networks, the matching network is more involved to realize [14] and normally achieves less performance. Due to this, we do not consider the matching network for application in this article. Due to its simplicity, it is possible to implement a relational metric-learning network. However, for now we only explore the prototypical network.

Appliance activation periods vary considerably, and this can result in imbalance in the number of appliance-specific generated signal images. There is no effect on the performance of few-shot prototypical metric-learning networks by this data imbalance. In this article, we propose the application of prototypical networks. Prototypical networks only produce a prototype (average) value embedding point of the samples in each class during training. A comparison is made of the average prototypes with a test embedding point through the Euclidean distance metric. We first carry out a similarity test on the entire dataset to establish the level of similarity between the data samples. The application stage of the prototypical network will require a similarity test value of at least 0.6 to increase the accuracy of our few-shot learning model.

We give the flowchart of the proposed system in Figure 1. Our proposed system allows for quick determination of the suitability of disaggregated appliance data for metric learning before the actual few-shot learning. By so doing, we are able to preprocess the data before conversion into acceptable TensorFlow file formats, which can result in improved model training. We assign an acceptable data similarity value in the overall data similarity search.

In the proposed train and test few-shot metric model block exploded in Figure 2, we seek to address the recognition of limited appliances signals by employing a model ( $\text{Model}_{\text{metric}}$ ) based on testing the similarity or dissimilarity between a known appliance signal image in the support set ( $D_{\text{support}}$ ) and an unknown disaggregated appliance signal image in the query set ( $D_{\text{query}}$ ). A conventional image-based deep learning neural model would require training by a very large sample count in ( $D_{\text{support}}$ ). The proposed system includes a training dataset ( $D_{\text{train}}$ ) to train the  $\text{Model}_{\text{metric}}$ . Training of the model is through a larger base set split into a specific number of different tasks ( $\{T_i, i \in h\}$  for  $1 \leq i \leq h$ ), for  $h$  tasks to optimize the loss function. The  $\text{Model}_{\text{metric}}$  is the prototypical network. A 1 shot Siamese model can also be realized. The training allows for the realization of a model

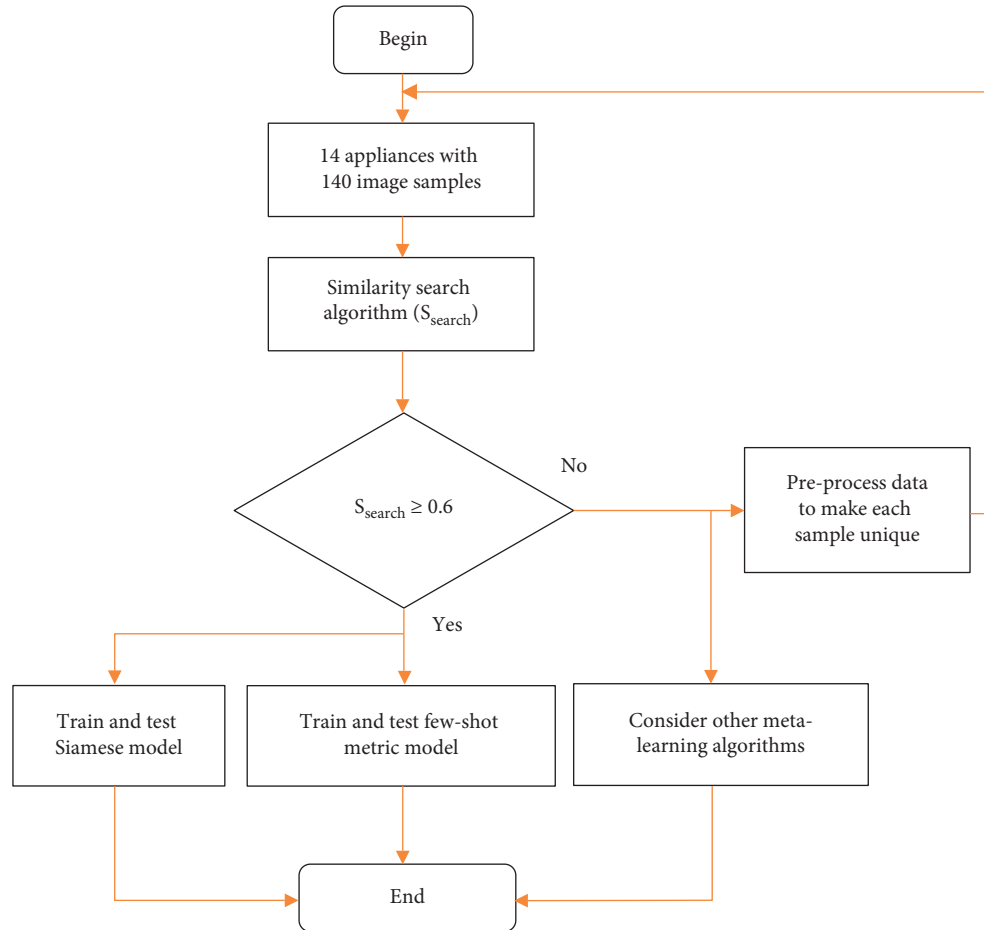


FIGURE 1: Proposed system with data presimilarity test.

that learns to learn to place the embedding of similar classes together in the test task.

**3.2. Dataset Preparation.** The dataset is made up of fourteen appliance categories or classes placed in an ALL\_IMAGES main directory on the computer. These appliances include four light-emitting-diode (LED) mains lamps (LED1-1 (5W), LED1-2(5W), LED2-1 (5W), and LED3-1 (5.5W)), two compact fluorescent (CFL) mains lamps (CFL1-1(12W), CFL2-1(14W)), three modes of HP laptop operation (laptop\_boot, laptop\_ms\_word, laptop\_video), a refrigerator (fridge), a microwave oven (microwave), a desktop computer (desktop), a two-plate cooking stove (stove), and a kettle (K). A sample of our raw few-shot train support dataset is shown in Figure 3 and is comprised of GASF IMs initially in RGB format and shape  $400 \times 400 \times 3$ . In Figure 3, we have shown only two samples out of ten samples per class. Using a PA1000 Tektronix [34] power analyzer in a laboratory setup, we measure the operational TS signals over the complete activation of the appliance. We then transform the appliance’s activation signals to IM equivalent by using GASF.

Figure 4 shows the images used in the test support and never seen before by the few-shot model.

As clearly seen in Figures 3 and 4, the sample images have different features and this property is used to successfully train and test the few-shot meta-learning model. It is important at this point to note that for the similarity test model, the samples in Figures 4 and 5 are considered one dataset, which is then split using the sklearn train\_test\_split. Converting the RGB images to grayscale and reducing image size helps to decrease the complexity of developed algorithms, speed up the process, and use less computation resources.

The use of both the omniglot and miniImagenet datasets for evaluating developed FSL algorithms is widespread. We observe that typical file formats in FSL algorithms include the NumPy array (.npy), tar.gz (.tgz), or just straight image file folder. However, the IMs in these FSL algorithms are normally converted to grayscale ( $L$ ) and resized to  $28 \times 28$ . The .npy grayscale images have an initial shape of  $28 \times 28$ . To take advantage of existing few-shot coding in literature, we prepare our custom dataset more or less in the same manner as for the omniglot and miniImagenet datasets. TFRecords files present data in binary record sequences. TFRecords is the recommended TensorFlow data format. We do not evaluate our final models on TFRecords as the conversion of various data formats to TFRecords requires different coding approaches. However, we do experiment with TFRecord files.

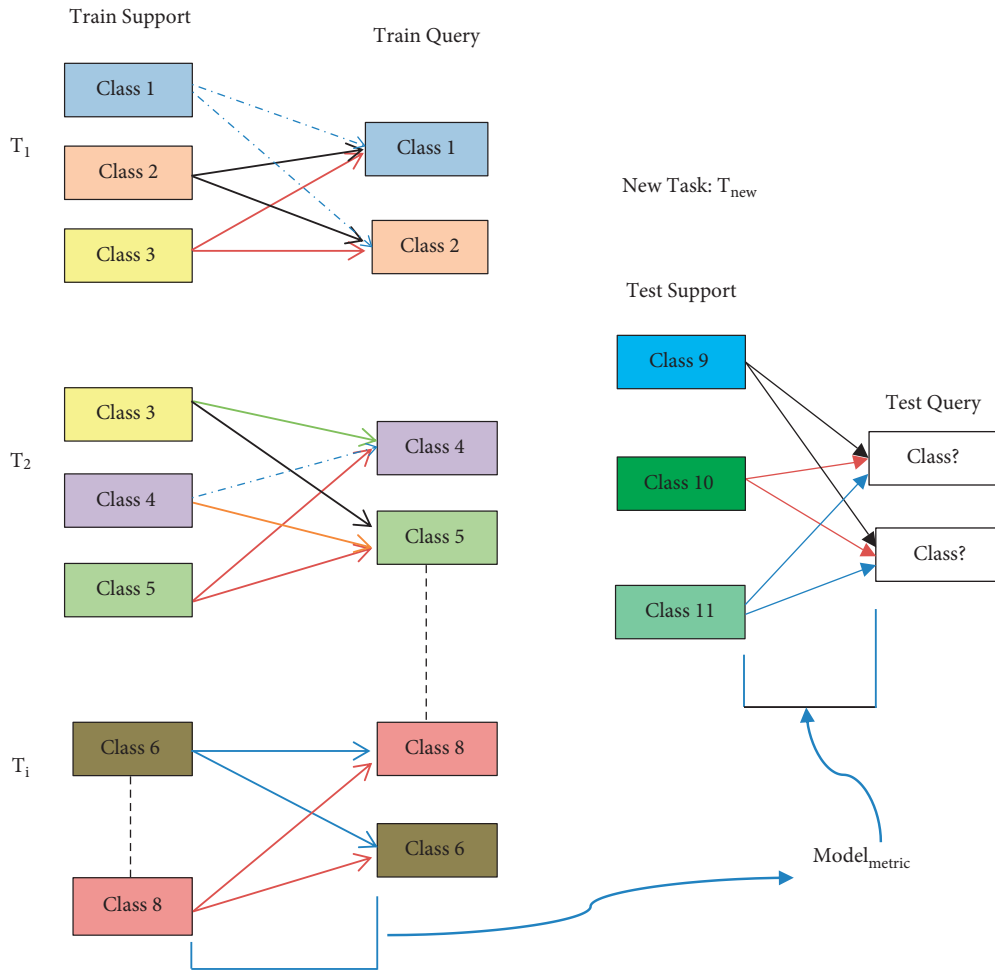


FIGURE 2: Few-shot metric-learning nonintrusive appliance recognition framework.

For each appliance, we are able to capture at least ten activation signals, which results in ten signal IM samples per appliance class. The total number of resized IMs per one measurement exercise is 140. When we take the NumPy route, the produced IM NumPy array is reshaped to [number of classes, samples per class, IM width, IM height, channels] to give shape (14, 10, 28, 28, 1). We then split this into train, test, and validation data. We follow the directory structure of the omniglot dataset to achieve the NumPy and reshape above. The execution of the create-miniimagenet.py converts the image miniimagenet folder into train.npy and test.npy. On the other hand, the helper.py script converts the omniglot dataset to .npy.

We then run a custom-developed script to convert our ALL\_IMAGE folder into train.py and test.py. In some instances, we performed the data split by producing train, test, and validation CSV files that contain the IM file name and label. The labeled IMs in this case are stored in their own separate folder.

**3.3. Training Procedure.** Due to the extensive coding required for the FSL algorithms, we had to experiment with code examples from numerous GitHub repositories and from keras.io code examples [35, 36]. We implemented the

code in both python and keras in IPython and Jupyter platforms, respectively. In training some code, we used the Google Colaboratory (Google Colab.) notebooks platform in which we could easily install such packages as PyTorch. Google Colab also allowed us to use the graphics processing unit (GPU) facility not available on our HP 650 Notebook to speed up training. However, in these codes, we modified the utilities (utils) data handling part to accommodate our in-house datasets. We also modified or added code for specific data results' visualization and experimented with various hyper-parameters. In some instances, we experimented with different number of convolutional layers. We also experimented with different epochs, episodes, and different number of support and query set classes and shots. However, our target system was the 1-shot model.

We trained the similarity test model in colab, with data loaded into My Drive in Google Drive. The RGB images of size  $400 \times 400$  are resized to  $28 \times 28$ . Training is performed with a train-to-test split ratio 0.75:0.25 and batch size of one. The train shape is  $(105 \times 28 \times 28 \times 3)$ , and test shape is  $(35 \times 28 \times 28 \times 3)$ . High numbers of batch size did not improve system performance, probably due to our limited training samples. The embedding model is a three-layer VGG 2D convolutional network with filter sizes 32, 64, and 128 from the input and kernel size of 3. During training, we

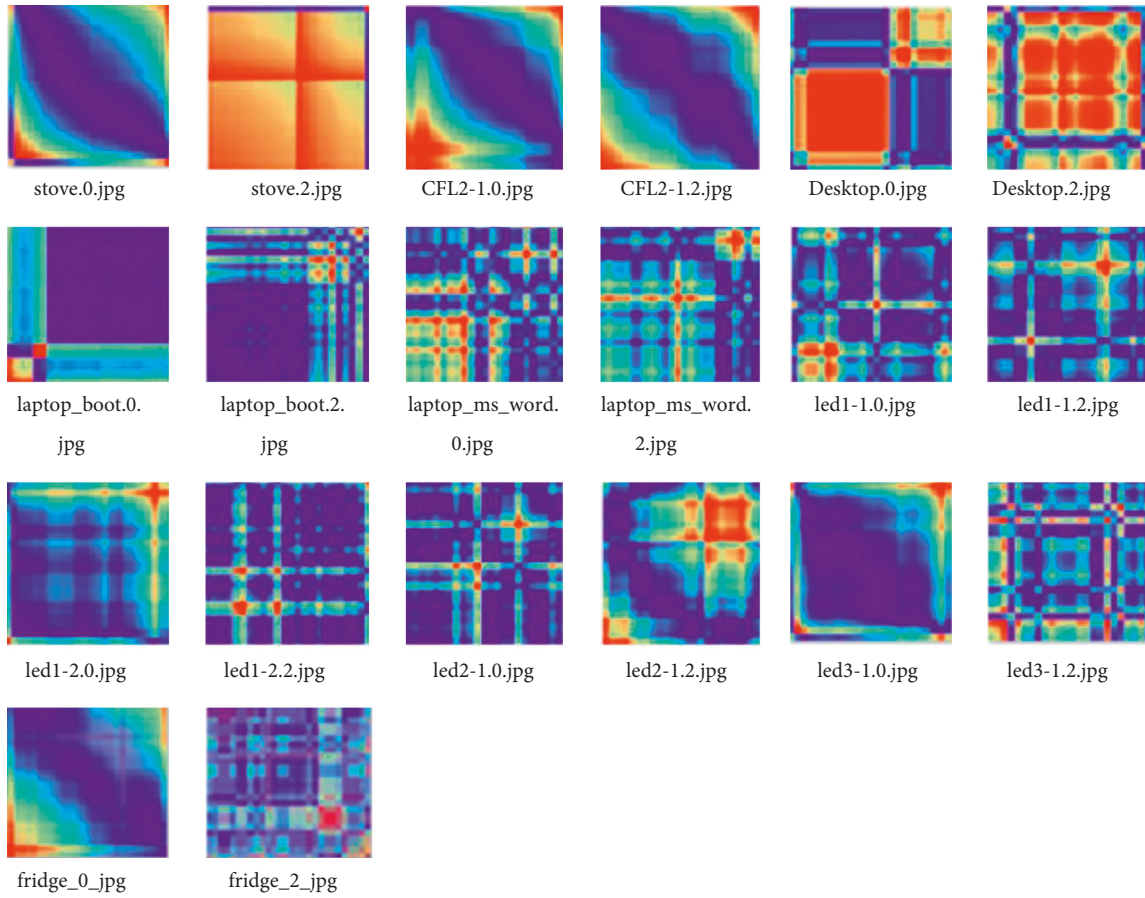


FIGURE 3: Two samples each of the ten (10) train support classes.

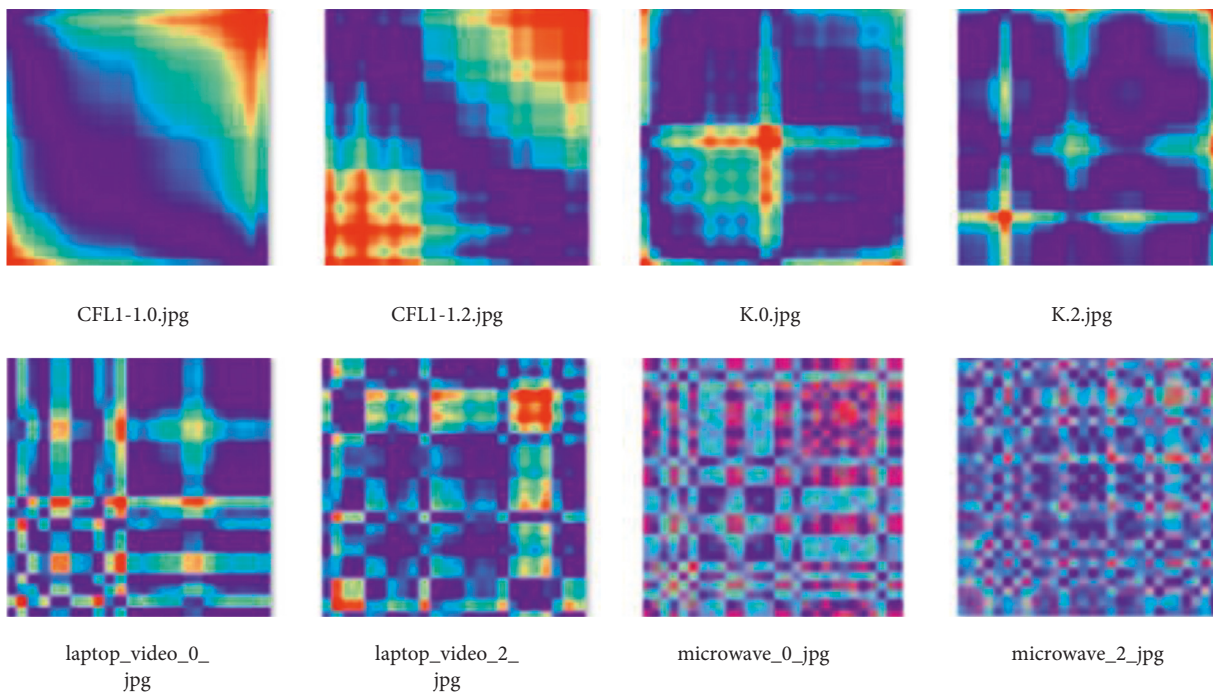


FIGURE 4: Two samples each of four (4) test support classes.



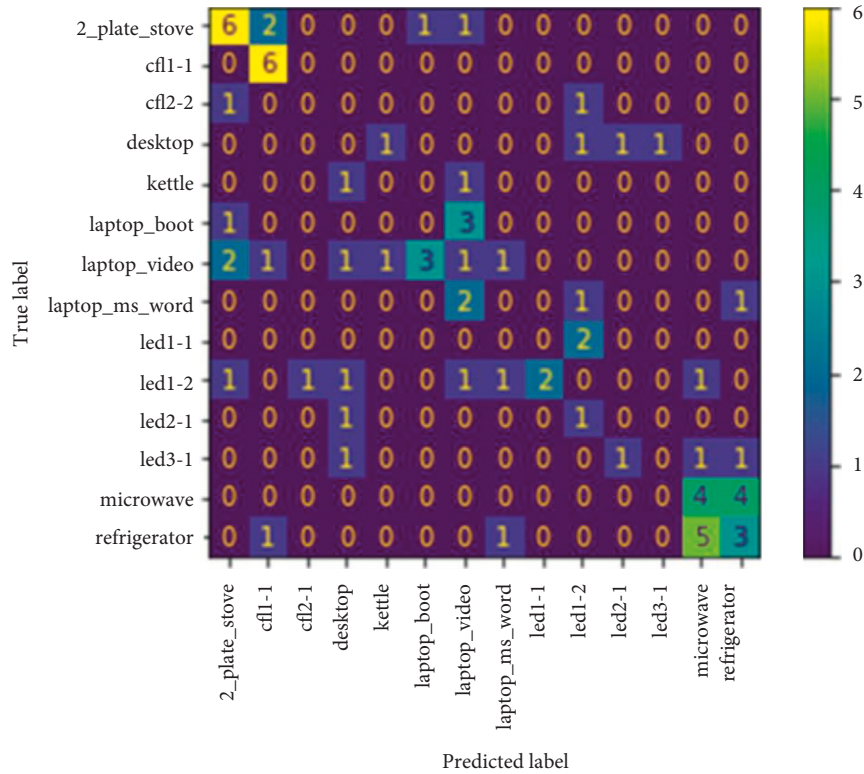


FIGURE 5: Similarity test confusion matrix.

experimented with various hyper-parameters and we would get slightly different classification results. However, for best classification results, we settled for the Adam optimizer with a learning rate of  $1e-3$  as in [35], and we used the sparse-categorical-cross-entropy loss function.

To train the Siamese network, we create two directories each with four grayscale images of dimension  $400 \times 400 \times 1$ . The directories are the 2\_plate\_stove and CFL1-1(12W) appliance classes. After preprocessing, the new data shape is 16, 2, 1, 200, 200, where 32 is the total sample size, 16 for 2 pairs input into the Siamese network. The 1, 200, 200 represents the new dimensions of the images in pgm format. As in Ref. [37], the base model consists of two 2D convolutional layers followed by flatten operation and two dense layers. The first 2D CNN layer had 6 filters each of size 3, ReLU activation, followed by max. Pooling (2, 2) and dropout 0.25. The second 2D CNN layer had 12 filters each of size 3, ReLU activation, followed by max. Pooling (2, 2) and dropout 0.25. The first dense layer is 128 units with dropout 0.1, and the last layer has 50 units with reLu activation. We used the contrastive loss and RMS optimizer. In addition, we developed a Siamese model based on the triplet loss function with a margin (alpha) of 0.2, 98 train grayscale samples and 42 grayscale test samples. We experimented with different image formats that proved to be difficult to implement in the coding of [37].

In the prototypical recognition system, the data were based on the RGB IMs of size  $400 \times 400$ . These IMs from a total IM count of 140 are reshaped to  $28 \times 28 \times 3$ . Two approaches were then used to format the input data into the prototypical network. The first approach involved the

internal model augmentation through rotation at different angles to obtain a final train set shape of (400, 28, 28, 3) and a test shape of (160, 28, 28, 3). The second approach took the 140 IM samples and reshaped to train set shape (10, 10, 28, 28, 3) and test set shape (4, 10, 28, 28, 3), where 4 represents the test classes and 10 the number of samples per class for the test set. The second approach implementable on CPU because of the low memory requirements provided the results captured in this article. It was necessary to combine the different modules available in GitHub to come up with a TensorFlow prototypical network [38], which was also executed in colab under GPU. The codes are executed with the SGD optimizer and a learning rate of 0.1. The recognition efficiency generally increases as the number of episodes increases. In the first approach, the highest accuracy was at 30000 episodes for a frame size of 1000. However, in the second CPU approach, the maximum episode count was 600 episodes.

## 4. Experimental Results

**4.1. Similarity Search.** In Figure 5, we see the appliance recognition results based on similarity.

To evaluate the suitability of our dataset for metric learning, we use the code in [35]. In this similarity test algorithm, we infer images of the same class as being similar, while those between classes are not. A requirement in model training is the pairing of images in the same class for the similarity test. One image is the AC and the other the PO [35]. There are 38 test samples out of the 140 total appliance

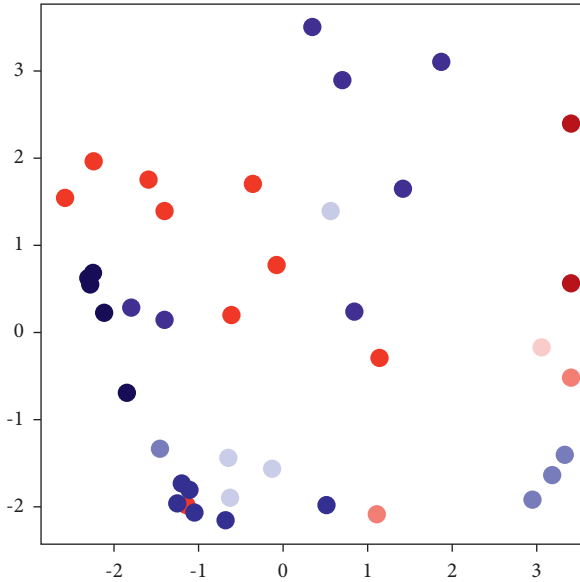


FIGURE 6: Triplet loss test embedding results.

IM samples. By pairing the test samples, we obtain 76 samples. For 14 classes, that translates to six (6) samples in the test set assigned to each class.

The `2_plate_stove` and `cf11-1` attained one hundred per cent recognition. The refrigerator and microwave oven also attained high levels of recognition. However, their relative high powers especially with the inclusion of the refrigerator switching spike some activations are almost similar between the two. The system also had difficulty in recognizing between the different operating modes of the laptop. However, the system was able to cluster all sample points as laptop among `laptop_boot`, `laptop_video` and `laptop_ms_word`, which in itself represents the success of the similarity test. With improved network and data, the similarity test system has potential for attaining high classification and hence passes the test criteria of 0.6. We give the similarity loss plot in Section 4.4

**4.2. Siamese Network 1 Shot Learning.** In Siamese network training, we have two authentic (similar) images to which we assign an authentic label of 1 to the pair. For a pair of images between classes, we assign a not-authentic label of zero (0). During training, the input into the Siamese network is either the pair of authentic images or a pair of not-authentic images. The trained model provides the set coordinate of the embedding of each similar pair per class. In our case, we obtain the Siamese 1 shot experimental results on both the contrastive and triplet loss functions. When the contrastive loss function trained Siamese network is tested against the compact fluorescent lamp (CFL2-1(14W)), it returns a true target value of 1 as given in the part sample code:

```
In [34]: x_testsia[target_index:target_index + 1,
0].shape
Out [32]:
(1, 1, 200, 200)
```

TABLE 1: Prototypical network few-shot learning test results.

Test	Ns	Nc	Nq	Test episodes	Avg. test accuracy %
1	1	2	9	600	91.343
				250	89.844
				100	91.167
2	2	2	8	600	94.406
				250	94.95
				100	94.50
3	3	2	7	600	94.976
				250	94.743
				100	95.429
4	5	2	5	600	96.10
				250	96.8
				100	94.7
5	7	2	3	600	96.722
				250	96.867
				100	97.83
6	8	2	2	600	97.417
				250	96.5
				100	96.5

TABLE 2: N-way  $k$ -shot accuracies for the prototypical network.

N-way	$k$ -shot	Average accuracy (%)
2-way	1-shot	91.343
2-way	2-shot	94.95
2-way	3-shot	95.429.
2-way	5-shot	96.8
2-way	7-shot	97.83
2-way	8-shot	97.417
3-way	1-shot	90.543
3-way	2-shot	93.361
3-way	3-shot	93.643
3-way	5-shot	94.378
3-way	7-shot	95.333
3-way	8-shot	95.889
4-way	1-shot	88.17
4-way	2-shot	93.005
4-way	3-shot	93.374
4-way	5-shot	93.892
4-way	7-shot	93.708
4-way	8-shot	94.958

```
In [34]: predsia = model.predict([x_testsia[target_index:target_index + 1, 0],
x_testsia[target_index:target_index + 1, 1]])
predsia = predsia < 0.5
print("y_test[target_index:], y_testsia[target_index, 0] == True, "pred:" ,predsia)
y_testsia[target_index]: True predsia: [[True]].
```

In triplet loss Siamese model, the train set shape is (98, 28, 28, 1) and the test shape is (42, 28, 28, 1). On the other hand, a part test code for the triplet loss is

```
In [15]: batch_size = 9
epochs = 200
steps_per_epoch = int(x_train.shape[0]/batch_size)
```

TABLE 3: Comparison with published results.

Method	TRDS	TRCL	TRQS	TRACC %	TEDS	TECL	TEACC % Avg:
Ours: Protonet	4	5	6	100.00	7	2	97.83
	4	5	6	100.00	8	2	97.41
	4	5	6	100.00	1	2	91.343
Reference [39]	22	5	—	75.00	8	2	93.75
Reference [40]	30	2	—	83.33	6	2	86.11

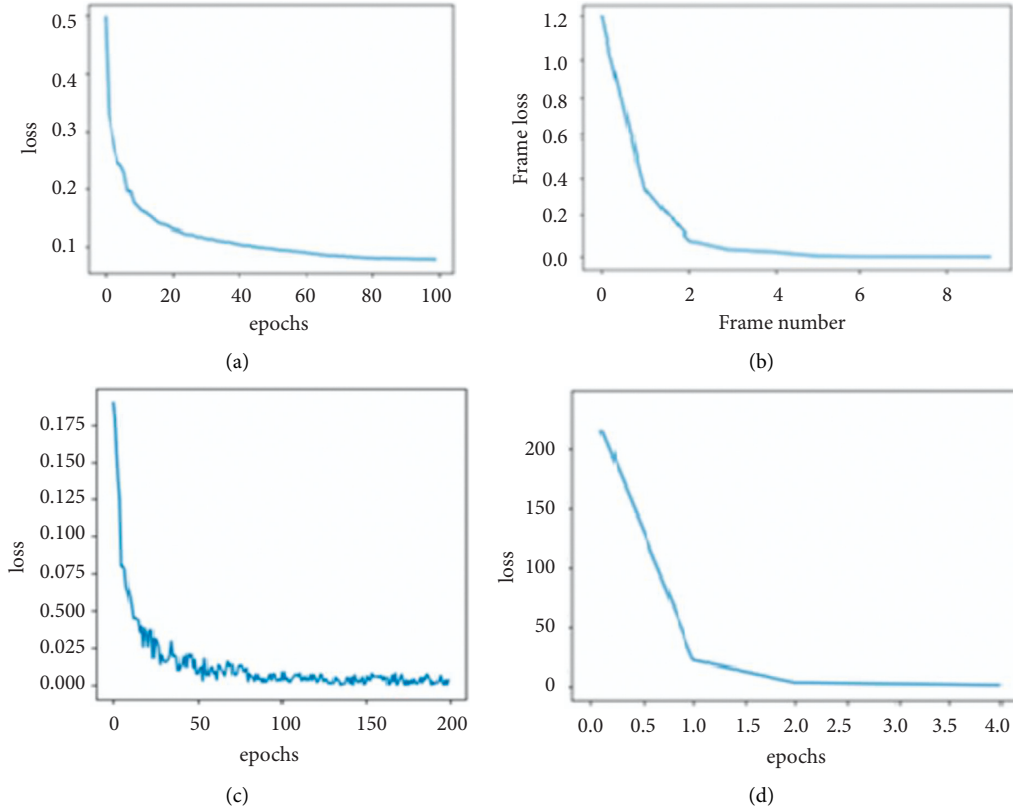


FIGURE 7: (a). Similarity network loss, (b) prototypical network loss, (c) triplet loss of Siamese network, and (d) constructive loss of the Siamese network.

```

net.compile (loss = triplet_loss, optimizer="adam")
_ = net.fit(
    data_generator(btch_size),
    steps_per_epoch = steps_per_epch,
    epochs = epchs, verbose = False,
    callbacks = [
        PCAEmdPlotter(
            plt, embedding_model,
            x_test[:40], y_test[:40]))].

```

We give both the contrastive and triplet loss plots in Section 4.4. Figure 6 shows the results of the test imbedding in the triplet loss model.

From Figure 6, there is a tendency for clustering of the embedding in any class. These results here show that there is a need to increase the train dataset or redesign the model for deeper DL. These results are in synchronization with the results shown in Figure 5 where the model tries with effort to obtain the classification of different appliance signals that are almost the same.

**4.3. Prototypical Network.** The prototypical model achieves high accuracy early in the training and converges well. The train loss and accuracy plots are given in Section 4.4. To test the model, we specify different values of number of samples or shots in the support set ( $N_s$ ), the different number of classes in the support set or ways ( $N_c$ ), and the number of

samples in the query set ( $N_q$ ) whose class value is unknown. Table 1 shows the relation between test support and query set sample entries per given appliance class number.

Table 1 gives all the few-shot learning test results collected for the prototypical network. In Tables 2, we show the summary of 2-way  $k$ -shot accuracy results for the prototypical network. In Table 2, the 2-way 7 shot gives the highest performance at 97.83% average test accuracy. Our data have four (4) test classes and ten (10) training classes. Hence, in the test and support class, the number of samples is either 4-way  $k$ -shot, 3-way  $k$ -shot, 2-way  $k$ -shot, or 1-way  $k$ -shot. Likewise, the training test set can vary from 10-way  $k$ -shot to 1- $k$ -shot. The 1-way  $k$ -shot is a theoretical postulation, since in reality, we cannot have a model that trains to detect one class in our case. However, a multiple sample within one class is feasible. The limited classes in our experiment will result in slight model overfit of 100% train accuracy to 97.83%. The 2-way 1-shot system gives a reasonable average test accuracy of 91.343%. The results in Tables 1 and 2 are based on a 5-way  $k$ -shot training set. A 10-way  $k$ -shot training set did not produce satisfactory results. Comparing the performance of the prototypical network with the metric similarity search at the beginning, we see an agreement between the two systems. Table 2 also shows the average accuracies of the 3-way  $k$ -shot and 4-way  $k$ -shot FSL test models.

In Tables 2, we see that the test accuracy goes up as the  $k$ -shot value goes up in the support set. The number of classes is the same in the test support and query sets. The four test cases belong to the refrigerator, kettle, LED2-1(5W), and CFL2-1(14W) mains lamps.

We now make a comparison of the results of this article to published results that use the same datasets. In Table 3, we show training dataset (TRDS), training classes (TRCL), training query set (TRQS), training accuracy (TRACC), test dataset (TEDS), testing classes (TECL), and the testing accuracy (TEACC).

From Table 3, we can clearly see that with a very limited training dataset, our model achieves a higher training accuracy and higher test accuracy than from publications that use the same data in different model architectures. Reference [39] is IM classification based on the capsule network, while Reference [40] is classification based on the ConvNet VGG image classifier. From the similarity metric test results at the beginning, we see that we have a number of classes whose embedding is very close to each other. We need to investigate how we can improve further the accuracy of our model by considering such issues as hybrid MEL systems [41, 42]. There is need to consider the visualization of the embedding including the actual class objects and labels from the query set.

**4.4. Model Loss Plots.** Figure 7 gives the training loss plots for the models developed in this article. The plots show the attempt by the models to reach convergence through stable training. Due to the limited training data points, the models tend to overfit; however, they do produce acceptable performance.

## 5. Conclusion

In this article, we investigate the application of few-shot learning in the form of a Siamese and prototypical network for the classification of disaggregated appliance signal images. By first carrying out a similarity test on the entire image dataset, we see that there are some appliances whose embedding is extremely close to each other. We observe a clear separation of embedding in other instances. We infer this information from the given confusion matrix. Nonetheless, the results show that we can achieve acceptable recognition of appliance signals using the Siamese and prototypical few-shot learning network. Two major challenges have been encountered in this study. The first is the inadequate number of available training classes so that the models could provide improved generalization. The second challenge is the closeness of some of the appliance signals to each other resulting in impaired discrimination between appliances. Program execution on normal CPU was extremely slow, or the system would just crash. Fortunately, we were able to make use of the GPU facility on Google Colab platform.

In future, we investigate the application of the MAML algorithm and the possible application of hybrid metric and gradient-descent few-shot learning methods for improved recognition performance. We will also consider increasing the training data classes and examples. Of particular interest is the improvement of the  $N$ -way 1-shot recognition setup.

## Data Availability

The data used in this study are derived from public domain resources.

## Conflicts of Interest

The corresponding author states that there are no conflicts of interest.

## Acknowledgments

This research was supported partially by South African National Research Foundation (Grants nos. 120106, 141951, and 132797) and South African National Research Foundation Incentive (Grant no. 132159).

## References

- [1] D. deP. Penha and A. R. C. Castro, "Home Appliance Identification for NILM systems based on deep neural network," *International Journal of Artificial Intelligence and Applications (IJAIA)*, vol. 9, no. 2, pp. 69–80, 2018.
- [2] S. Houidi, F. Auger, H. B. A. Sethom, D. Fourer, and L. Miegerville, "Multivariate event detection methods for non-intrusive load monitoring in smart homes and residential buildings," *Energy and Buildings*, vol. 208, Article ID 109624, 2020.
- [3] J. Kelly and W. Knottenbelt, "Neural NILM: deep neural networks applied to energy disaggregation," in *Proceedings of the 2nd 31 ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*, pp. 55–64, South Korea, November, 2015.

- [4] L. De Baets, J. Ruysinck, C. Develder, T. Dhaene, and D. Deschrijver, "Appliance classification using VI trajectories and convolutional neural networks," *Energy and Buildings*, vol. 158, pp. 32–36, 2018.
- [5] A. Mikolajczyk and M. Grochowski, "Data augmentation for improving deep learning image classification problem," in *International Interdisciplinary PhD Workshop*, pp. 117–122, IIPhDW), Swinoujcie, 2018.
- [6] L. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning. Computer Vision and Pattern Recognition," 2017, <https://arxiv.org/abs/1712.04621>.
- [7] M. Mottahedi and S. Asadi, "Non-intrusive load monitoring using imaging time series and convolutional neural networks," in *Proceedings of the 16th International Conference on Computing in Civil and Building Engineering*, pp. 705–710, Osaka, Japan, July, 2016.
- [8] Z. Wang and T. Oates, "Imaging time-series to improve classification and imputation," in *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 3939–3945, Montreal, August, 2015.
- [9] A. Alofi, A. Alghamdi, R. Alahmadi, and N. Aljuaid, "A review of data fusion techniques," *International Journal of Computer Application*, vol. 167, no. 7, pp. 37–41, 2017.
- [10] Y. Shi-tong, "An effective image fusion rule based on consistency checking," in *Proceedings of the International Conference on Computer Science and Network Technology*, pp. 2443–2446, Dalian, China, October, 2011.
- [11] S. Yang, F. Liu, N. Dong, and J. Wu, "Comparative analysis on classical meta-metric models for few-shot learning," *IEEE Access*, vol. 8, Article ID 127065, 2020.
- [12] Y. Wang, Q. Yao, J. T. Kwok, L. M. Ni, and L. M. Ni, "Generalizing from a few examples: a survey on few-shot learning," *ACM Computing Surveys*, vol. 53, no. 3, pp. 1–34, 2021.
- [13] S. Thrun and L. Pratt, "Learning to learn: introduction and overview," in *Learning to Learn*, S. Thrun and L. Pratt, Eds., pp. 3–17, Springer, Boston, MA, 1998.
- [14] O. Vinyals, C. Blundell, T. Lillicrap, and D. Wierstra, "Matching networks for one shot learning," in *Proceedings of the Adv. Neural Inf. Process. Syst.*, pp. 3630–3638, Granada Spain, December, 2016.
- [15] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," in *Proceedings of the Adv. Neural Inf. Process. Syst.*, pp. 4077–4087, Granada Spain, November, 2017.
- [16] Sung, "Learning to compare: relation network for few-shot learning," in *Proceedings of the IEEE Conf. Comput. Vis. Pattern Recognit.*, pp. 1199–1208, Honolulu, HI, USA, June, 2018.
- [17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [18] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," *Proceedings of the 34th International Conference on Machine Learning PMLR*, vol. 70, pp. 1126–1135, 2017.
- [19] W.-Y. Chen, "A closer look at few-shot classification," 2019, <https://arxiv.org/abs/1904.04232>.
- [20] L. De Baets, C. Develder, T. Dhaene, and D. Deschrijver, "Detection of unidentified appliances in non-intrusive load monitoring using siamese neural networks," *International Journal of Electrical Power & Energy Systems*, vol. 104, pp. 645–653, 2019.
- [21] G. Koch, R. Zemel, and R. Salakhutdinov, *Siamese neural networks for one-shot image recognition ICML Deep Learning Workshop*, vol. 2, 2015.
- [22] R.-C. Mihailescu, D. Hurtig, and C. Olsson, "End-to-end anytime solution for appliance recognition based on high-resolution current sensing with few-shot learning," *Internet of Things*, vol. 11, Article ID 100263, 2020.
- [23] F. Alqadah and R. Bhatnagar, "Similarity measures in formal concept analysis," *Annals of Mathematics and Artificial Intelligence*, vol. 8, 2011.
- [24] F. Iglesias and W. Kastner, "Analysis of similarity measures in times series clustering for the discovery of building energy patterns," *Energies*, vol. 6, no. 2, pp. 579–597, 2013.
- [25] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. Russell, "Distance metric learning, with application to clustering with side-information," 2002, <https://ai.stanford.edu/%7Eang/papers/nips02-metric.pdf>.
- [26] A. A. Gupta, D. P. Foster, and L. H. Ungar, "Unsupervised Distance Metric Learning Using Predictability," 2008, [https://repository.upenn.edu/cgi/viewcontent.cgi?article=1928&context=cis\\_reports](https://repository.upenn.edu/cgi/viewcontent.cgi?article=1928&context=cis_reports).
- [27] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, pp. 1735–1742, New York, NY, USA, June, 2006.
- [28] Suprpto and J. A. Polela, "The influence of loss function usage at SIAMESE network in measuring text similarity," (IJACSA), *International Journal of Advanced Computer Science and Applications*, vol. 11, no. 12, pp. 787–792, 2020.
- [29] B. M. Lake, R. Salakhutdinov, J. Gross, and J. Tenenbaum, "One shot learning of simple visual concepts proceedings of the annual meeting of the cognitive science," *Society*, vol. 33, no. 33, 2011.
- [30] C. Pan, J. Huang, J. Hao, and J. Gong, "Towards zero-shot learning generalization via a cosine distance loss," *Neuro-computing*, vol. 381, pp. 167–176, 2020.
- [31] A. Mishra, A. Pandey, and H. A. Murthy, "Zero-shot learning for action recognition using synthesized features," *Neuro-computing*, vol. 390, pp. 117–130, 2020.
- [32] H. J. Ye, X.-R. Sheng, and D. C. Zhan, "Few-shot learning with adaptively initialized task optimizer: a practical meta-learning approach," *Machine Learning*, vol. 109, no. 3, pp. 643–664, 2020.
- [33] S. Qiao, C. Liu, W. Shen, and A. Yuille, "Few-shot image recognition by predicting parameters from activations," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7229–7238, Salt Lake City, June, 2018.
- [34] Tektronix, *PA1000 Power Analyzer User Manual*, Tektronix, Inc, USA, 2018.
- [35] M. Kelcey, "Metric learning for image similarity search," 2020, [https://keras.io/examples/vision/metric\\_learning/](https://keras.io/examples/vision/metric_learning/).
- [36] A. D. Moreau, "Few-Shot learning with Reptile," 2020, <https://keras.io/examples/vision/reptile/>.
- [37] P. Ramesh, "Face recognition using siamese networks [Tutorial]," 2019, <https://hub.packtpub.com/face-recognition-using-siamese-networks-tutorial/>.
- [38] Omniglot, "Prototypical Net," 2019, <https://www.kaggle.com/hsankesara/prototypical-net>.
- [39] L. Matindife, Y. Sun, and Z. Wang, "Disaggregated power system signal recognition using capsule network," in *Neural Computing for Advanced Applications. NCAA 2020. Communications in Computer and Information Science*, H. Zhang,

- Z. Zhang, Z. Wu, and T. Hao, Eds., vol. vol 1265, Singapore, Springer, 2020.
- [40] L. Matindife, Y. Sun, and Z. Wang, "Image-based mains signal disaggregation and load recognition," *Complex & Intelligent Systems*, vol. 7, no. 2, pp. 901–927, 2021.
- [41] X. Li, L. Yu, C. W. Fu, M. Fang, and P. A. Heng, "Revisiting metric learning for few-shot image classification," *Neurocomputing*, vol. 406, pp. 49–58, 2020.
- [42] D. Wang, Y. Cheng, M. Yu, X. Guo, and T. Zhang, "A hybrid approach with optimization-based and metric-based meta-learner for few-shot learning," *Neurocomputing*, vol. 349, pp. 202–211, 2019.