

---

Article

# Closing the performance gap between Siamese networks for dissimilarity image classification and convolutional neural networks

Loris Nanni<sup>1,\*</sup>, Giovanni Minchio<sup>1</sup>, Sheryl Brahnam<sup>2</sup>, Davide Sarraggiotto<sup>1</sup> and Alessandra Lumini<sup>3</sup>

<sup>1</sup> DEI, Via Gradenigo 6, 35131 Padova, Italy; loris.nanni@unipd.it, [giovanni.minchio@studenti.unipd.it](mailto:giovanni.minchio@studenti.unipd.it), and [gianluca.maguolo@phd.unipd.it](mailto:gianluca.maguolo@phd.unipd.it)

<sup>2</sup> Department of Information Technology and Cybersecurity, Missouri State University, 901 S, National Street, Springfield, MO 65804, USA; [sbrahnam@missouristate.edu](mailto:sbrahnam@missouristate.edu)

<sup>3</sup> DISI, University of Bologna, Via dell'Università 50, 47521 Cesena, Italy; [alessandra.lumini@unibo.it](mailto:alessandra.lumini@unibo.it)

\* Correspondence: [loris.nanni@unipd.it](mailto:loris.nanni@unipd.it)

**Abstract:** In this paper, we examine two strategies for boosting the performance of ensembles of Siamese networks (SNNs) for image classification using two loss functions (Triplet and Binary Cross Entropy) and two methods for building the dissimilarity spaces (FULLY and DEEPER). With FULLY, the distance between a pattern and a prototype is calculated by comparing two images using the fully connected layer of the Siamese network. With DEEPER, each pattern is described using a deeper layer combined with dimensionality reduction. The basic design of the SNNs takes advantage of supervised k-means clustering for building the dissimilarity spaces that train a set of support vector machines, which are then combined by sum rule for a final decision. The robustness and versatility of this approach are demonstrated on several cross-domain image data sets, including a portrait data set, two bioimage and two animal vocalization data sets. Results show that the strategies employed in this work to increase the performance of dissimilarity image classification using SNN is closing the gap with standalone CNNs. Moreover, when our best system is combined with an ensemble of CNNs, the resulting performance is superior to an ensemble of CNNs, demonstrating that our new strategy is extracting additional information.

**Keywords:** Siamese networks; Ensemble of classifiers; Loss function; Discrete cosine transform

---

## 1. Introduction

Interest in classification systems based on (dis)similarity spaces is resurging. Unlike the more common technique of classifying samples within a feature space, (dis)similarity classification estimates the class of an unknown pattern by examining its similarities and dissimilarities with a set of training samples and pairwise (dis)similarities between each of the members. This process has come to involve more than the application of standard distance measures; (dis)similarity classification is also a way to build new spaces.

Though the two terms, *similarity* and *dissimilarity* are rarely disambiguated in the literature, classification based on the notion of dissimilarity is an idea first proposed in [1], where the focus was on comparing differences between samples belonging to different classes. Dissimilarity classification can be tackled by using either dissimilarity vectors, as in [2-6], or dissimilarity spaces, as in [7-14]. In the former case, two samples are considered positive if they belong to the same class and negative if they belong to separate classes. The goal of the classifier is to decide which of these two cases a given vector was calculated on. For a more detailed discussion of this approach, see [15].

In contrast, dissimilarity methods that generate dissimilarity spaces, the approach taken here, produce classifiers from within feature vector spaces. Unlike traditional feature vectors representing samples as measured across all features, representation from feature vector spaces is the distance between pairs of samples. In [1], who introduced this

approach, the authors applied prototype selection for training classifiers on dissimilarity spaces. The dissimilarity representations were used as a vector space. This method was applied to image retrieval by [8] using a prototype-based dissimilarity space. In [10], a compact representation based on prototype selection methods was derived from deep convolutional features and learned distance measures.

A loss function commonly used in dissimilarity classification is the Maximum Mean Discrepancy (MMD). In [11], the application of MMD enabled the source and target data in the dissimilarity space to harness the intra-class and inter-class distributions to produce a pairwise matcher. This version of MMD was also shown to work well across several data sets. A modification of the contrastive loss function for a Siamese Neural Network (SNN) [16, 17] was proposed in [18] for brain image classification. The correlation distance of this variant of the loss function predicted the output features of image pairs. This method was expanded for audio classification in [12] and [13]. The audio samples, represented as spectrograms, were transformed by clustering methods into a set of centroids that generated dissimilarity spaces via SNN. The audio samples were then projected into the dissimilarity spaces to obtain a vector space representation that could be used to train Support Vector Machines (SVMs). An improved version of this method was developed for generic image classification in [14], where dissimilarity spaces were produced by a set of clustering methods and a set of SNNs with different CNN backbones. This approach was shown to compete well against state-of-the-art classifiers on several image data sets and obtained the highest classification score on one of them.

This work further expands [14] by proposing additional techniques for improving the performance of an ensemble of SNNs. As in the earlier work, each Siamese network, composed of eight different CNN topologies, generates a dissimilarity space whose features train an SVM, and the SVMs are then combined by sum rule. The strategies investigated here for improving performance further are the following:

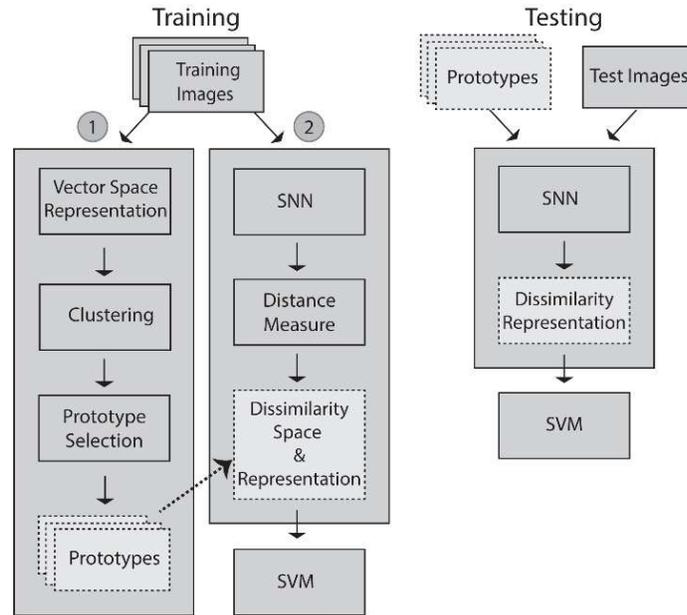
- Two different loss functions are used to train the Siamese networks: the binary cross entropy loss function and the triplet loss function;
- Two different approaches for building the dissimilarity spaces are proposed for extracting features: the first is based on the fully connected layer and the latter on a deeper layer where the size of each channel is reduced by the Discrete Cosine Transform (DCT).
- SNNs are optimized using different variants of Adam, with a new Adam variant proposed in this work.

Systems built with these strategies are compared, fused, and evaluated with previous work on dissimilarity classification. The versatility and robustness of the best ensemble developed using these techniques are demonstrated on five cross-domain image data sets representing medical imaging problems, animal vocalizations (spectrograms), and portrait images.

## 2. Proposed Approach

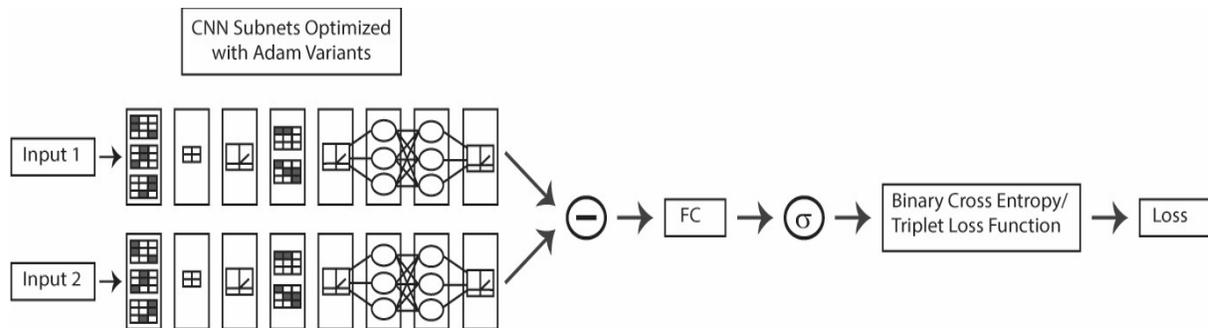
The basic system can be described as follows. The inputs into the system, as in [12-14], are the original images and HASC descriptors [19], extracted to produce a new processed image. If the original image is in color, Hasc is applied separately on each band; if it is grey level, the Hasc image is replicated three times to build an image with three bands.

Starting with the vector space representations, step 1 of the training process, as illustrated in Figure 1, begins by generating a set of clusters that produce a set of prototypes. The prototypes are centroids generated by k-means on the vector space representations. In step 2, a dissimilarity space is generated by an SNN that learns a distance measure from the prototypes that maximizes differences between pairs within class while also minimizing differences of pairs between other classes, a process that produces a feature vector that is trained on an SVM. In the testing stage, an unknown pattern is projected onto the dissimilarity space that was learned by the SNN, which generates the feature vector that is then fed into the trained SVM for a decision.



**Figure 1.** Schematic of the basic dissimilarity architecture using one SNN with the output fed into one SVM.

The SNN, as illustrated in Figure 2, combines two identical deep learners whose outputs are subtracted, which produces a feature vector (the absolute value of the difference) that is passed to a sigmoid and a loss function as in [12-14]. Unlike [12-14], which used binary cross entropy, two different loss functions are tested here (binary cross entropy and triplet loss function), and the CNN subnets are optimized with Adam and some Adam variants.



**Figure 2.** Schematic of SNN.

Though some variations are indicated in Figures 1 and 2, they only show the output of one SNN fed into one SVM. In [12-14] and this work, many SNNs and SVMs are trained, tested, and combined. Eight CNN topologies form the backbone of the SNNs. These are the identical topologies described in [14] (for the reader's convenience, the table in [14] that details the topologies is reprinted in the Appendix). Thus, a large number of SNNs are trained using the different topologies, the two loss functions, and the Adam

optimization algorithms. Each of these systems is tested, fused, and evaluated to build the best-performing system empirically.

The pseudocode for each step in Figure 1 can be found in the following sources: [12], [13], and [14] (see as well the companion source code for this paper available at <https://github.com/LorisNanni>).

Below, we focus on the new techniques proposed in this work: the application of two methods for generating the dissimilarity space (section 2.1), the two different loss functions (section 2.2), and the Adam optimization methods, including a new one proposed here networks (2.3).

### 2.1. Methods for generating the dissimilarity spaces

Both methods for generating the dissimilarity space follow the same basic process used in [12-14]: first, k-means is applied on a vector space representation of the training images, with prototypes calculated as the k centroids of the clusters so produced. Second, a feature vector  $F \in R_k$  is extracted by calculating the distances of image  $x$  from each of the prototypes, where the distance for each  $F_i$  between  $x$  and prototype  $p_i$  is given as  $F_i = d(x, p_i)$ . The resulting feature vector  $F_i$  is fed into the SVM.

The two methods for generating the dissimilarity space are labeled FULLY and DEEPER. With FULLY, the distance between a pattern and a prototype is obtained directly by comparing the two images using the Siamese network. With DEEPER, each pattern is described using a deeper layer than the fully connected backbone network of the Siamese network. To reduce the high dimensionality of this deeper layer, the Discrete Cosine Transform (DCT) is applied separately to each channel of that layer (see section 2.2). Finally, the distance between a pattern and a prototype is given by the cosine distance. In other words, the backbone of the Siamese network is used as the feature extractor.

For the sake of space, the layers used in DEEPER are reported in the MATLAB toolbox available at <https://github.com/LorisNanni>. This step is not optimized. We have chosen the layer before the last ReLU or fully connected layer to prevent overfitting the results rather than selecting layers optimized for each data set. Optimal layers could have been discovered using a leave-one-out data set, but this procedure was not feasible given the computational power of our GPUs.

### DCT dimensionality reduction

Because DEEPER uses a deeper layer compared to the fully connected backbone to generate the dissimilarity space, a method is needed to reduce dimensionality on each channel (with results combined) of the deeper layer. DCT [20] is the dimensionality transform selected here because 1) its components are typically small in magnitude (most information is located in the low-frequency coefficients), and 2) it balances information packing and computational complexity.

DCT can be expressed as

$$DCTimage(x, y) = \sum_{p,q=1}^n a_p a_q Image(p, q) \cos \frac{2p-1}{2n} \cos \frac{2q-1}{2n}, \quad (1)$$

$$a_p = \begin{cases} \sqrt{\frac{1}{n}}, & n = 1 \\ \sqrt{\frac{2}{n}}, & n > 1 \end{cases}$$

Each channel is reduced to a dimension of 9×9. All the features extracted from each channel are concatenated into a single vector that represents a given pattern/prototype.

### 2.2. Loss functions

### Binary cross entropy loss (Cross)

In the training phase, every pair of images in the training set is fed into the backbone of the Siamese architecture to obtain a feature vector  $F$ . Calculated next is  $Z = |F_1 - F_2|$ , where  $F_1$  and  $F_2$  are the feature vectors of the two images in the pair.  $Z$  is passed through a fully connected layer and a sigmoid function that returns the probability  $Y$  that the two images belong to the same class. Cross is then used for the two-class problem.

In the testing phase, for every sample in the training set, we compute  $F$ . Then, we evaluate  $N$  centroids using k-means clustering. Every image in the training set is expressed as the vector of the distances between its features and the centroids. After that, we train an SVM on those vectors. We then apply this inference algorithm to the images in the test set.

### Triplet loss (Triplet)

With Triplet, we take three images as the inputs, labeled A, P, and N. It is assumed that A and P have the same label and A and N different labels.

In the training phase, for every Triplet in the training set, feature vectors  $F_A, F_P, F_N$  are computed then passed through a sigmoid to obtain  $Y_A, Y_P, Y_N$ . At that point, the loss function is:

$$L = \max(|Y_A - Y_P|_2 - |Y_A - Y_N|_2, -\xi), \quad (2)$$

where  $\xi$  is a positive number, and  $|x|_2$  is the Euclidean norm of the vector. In other words, the loss function encourages the network to create similar representations for samples in the same class and different representations for samples in different classes.

In the testing phase, the process is exactly the same as described for the testing phase of cross-entropy loss.

### 2.3 Adam variants

Introduced in [20, 21], the widely used optimization method Adam (referred to as Base Adam in the experimental section) takes advantage of adaptive gradient and momentum to compute adaptive learning rates for each parameter. It makes use of the gradient at the current step, the exponential moving average of the gradient (first order moment), and the exponential moving average of the square of the gradient (second order moment).

Thus, the first moment  $m_t$  and the second moment  $u_t$  are defined as:

$$m_t = \rho_1 m_{t-1} + (1 - \rho_1) g_t \quad (3)$$

$$u_t = \rho_2 u_{t-1} + (1 - \rho_2) g_t^2 \quad (4)$$

where the hyperparameters  $\rho_1$  and  $\rho_2$  represent the exponential decay rate for the first and second moment (set respectively to 0.9 and 0.99),  $g_t$  is the gradient at time  $t$ , and the square on  $g_t$  is meant to be calculated component-wise. The moments are initialized as  $m_0 = u_0 = 0$ .

To avoid small values of the moving averages due to being initialized to zero, Adam includes a bias-corrected version of the first and second order moments:

$$\hat{m}_t = \frac{m_t}{(1 - \rho_1^t)} \quad (5)$$

$$\hat{u}_t = \frac{u_t}{(1 - \rho_2^t)} \quad (6)$$

The parameter update is computed as follow:

$$\theta_t = \theta_{t-1} - \lambda \frac{\hat{m}_t}{\sqrt{\hat{u}_t + \epsilon}}, \quad (7)$$

where  $\lambda$  is the learning rate and  $\epsilon$  is a very small positive number used to avoid any division by zero (usually set to  $10^{-8}$ ). The operations are supposed to be component-wise.

As noted in [22], Adam performs reasonably well in practice compared to other adaptive learning methods; however, Adam does not utilize the change in immediate past gradient information, a utilization that is incorporated in [22] and [23].

### DGrad

This variant, proposed in [23], makes use of the absolute difference between the current gradient  $g_t$  and the moving average of the element-wise squares of the gradients:

$$\Delta a g_t = |g_t - av g_t| \quad (8)$$

where  $av g_t$  is the moving average of the component-wise squares of the gradient.

The absolute difference  $\Delta a g_t$  is then normalized by its maximum component as follows:

$$\Delta \widehat{a} g_t = \frac{\Delta a g_t}{\max(\Delta a g_t)} \quad (9)$$

Then,  $\xi_t$  is defined as:

$$\xi_t = Sig(4 \cdot \Delta \widehat{a} g_t) \quad (10)$$

where  $Sig(\cdot)$  is the sigmoid function:

$$Sig(x) = \frac{1}{1+e^{-x}} \quad (11)$$

Each parameter of the network is finally updated following the equation:

$$\theta_{t+1} = \theta_t - \lambda \cdot \xi_t \frac{\widehat{m}_t}{\sqrt{\widehat{u}_t + \epsilon}} \quad (12)$$

where  $\widehat{m}_t$  and  $\widehat{u}_t$  are the first and second order moments seen in Adam.

### DecayDGrad (New)

This DGrad variant introduces a learning rate decay, both locally and in the whole training process. The local decay can be achieved with a periodic impulse, defined as follows:

$$imp_t = e^{-\left(\frac{2 \times \text{mod}(t,s)}{s}\right)^2} \quad (13)$$

where  $s = 10$  is the period (number of iterations between each impulse).

The impulse  $imp_t$  is then multiplied by a global decay factor  $d_t$ , shown in the equation:

$$d_t = e^{-\frac{2 \times (t - c \cdot n_{iter})^2}{n_{iter}^2}} \quad (14)$$

where  $n_{iter}$  is the total number of iterations in the training process. The parameter  $c = 0.25$ , multiplied by  $n_{iter}$ , determines the iteration whereby  $d_t$  assumes its maximum value.

The parameter  $\xi_t$  is therefore defined as:

$$\xi_t = Sig(4 \cdot \Delta \widehat{a} g_t) \cdot imp_t \cdot d_t. \quad (15)$$

Each parameter of the network is updated as shown in (DGrad).

Notice that  $imp_t$  only has values in range 0 to 1, and its maximum value is assumed for iterations, which are multiples of  $s$ . The purpose of these restraints is to attenuate the value calculated by DGrad locally, namely progressively in the span of  $s$  iterations, to get a better evaluation of the local minimum, thereby avoiding an eventual overshoot of the global minimum.

The reason behind the learning rate decay factor  $d_t$  is to keep the learning rate high in the initial part of the training, which accelerates training and avoids the memorization of noisy data while at the same time extending the decay in later iterations. In this way, DGrad can learn complex patterns, as shown in [24]. The plot of  $d_t$  and  $imp_t \cdot d_t$  is reported in Figure 2.

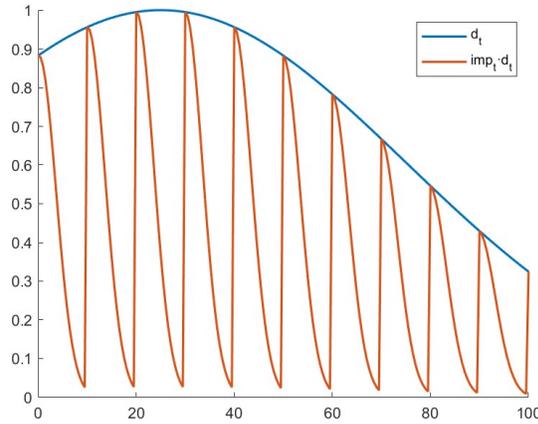


Figure 2. Plot of  $d_t$  and  $imp_t \cdot d_t$ .

### 3. Data Sets

The following five image data sets, representing very different classification tasks, were selected to demonstrate the versatility of the proposed method:

- BIRDz [25]: This balanced dataset is a real-world benchmark for bird species vocalizations. The testing protocol is ten runs using the data split in [25]. The audio tracks were extracted from the Xeno-Canto Archive (<http://www.xeno-canto.org/>). BIRDz contains a total of 2762 acoustic samples from eleven North American bird species, along with 339 unclassified audio samples (consisting of noise and unknown bird vocalizations). The bird classes vary in size from 246 to 259. Each observation is represented by five spectrograms: 1) constant frequency, 2) frequency modulated whistles, 3) broadband pulses, (4) broadband with varying frequency components, and 5) strong harmonics;
- CAT [26, 27]: This dataset has ten balanced classes of cat vocalizations, with each one containing ~300 samples for a total of 2962 samples taken from Kaggle, Youtube, and Flickr. The testing protocol is 10-fold cross-validation. The average duration of each sample is 4s.
- InfLar [28]: This dataset contains eighteen Narrow-Band Imaging (NBI) endoscopic videos of eighteen different patients with laryngeal cancer. The videos were retrospectively analyzed and categorized into four classes (informative, blurred, containing saliva or specular reflections, and underexposed). The average video length is 39s. The videos were acquired with an NBI endoscopic system (Olympus Visera Elite S190 video processor and an ENF-VH rhino-laryngo videoscope) with a frame rate of 25fps and an image size of  $1920 \times 1072$  pixels. A total of 720 video frames, 180 for each of the four classes was extracted and labeled. The testing protocol is three-fold cross-validation with data separated at the patient level to ensure that the frames from the same class were classified based on the features characteristic of each class and not due to features linked to the individual patient (e.g., vocal fold anatomy).
- RPE [29]: This is a medical image classification data set that intends to distinguish the maturation of human stem cell-derived retinal pigmented epithelium. RPE is based on 195 images that were divided into sixteen subwindows. These subwindows were then assigned to one of four classes: 1) Fusiforms, 2) Epithelioid, 3) Cobblestone, and 4) Mixed. Subwindows that were out of focus or that contained background information exclusively were discarded. This division of images into four and the exclusion process produced a total of 1862 images.

- Port [30]: This dataset contains 927 paintings from six different art movements: 1) High Renaissance, 2) Impressionism, 3) Northern Renaissance, 4) Post-Impressionism, 5) Rococo, and 6) Ukiyo-e. Ten-fold cross-validation is the testing protocol. The same testing protocol presented in the papers introducing each data set is used in the experimental section, with accuracy the performance indicator.

### 3. Experimental results

The default settings in the MATLAB framework for Siamese networks were used to train the SNNs in all experiments to ensure no overfitting for any given data set. For ADAM optimization and its variants, the number of iterations was set to 3000 with no stop criterion, the gradient decay factor to 0.9, the squared gradient decay factor to 0.99, and the learning rate to 0.0001.

The first run of experiments is reported in Table 1. In these tests, we use all the data sets. Each performance cell in Table 1 contains three rows of values for each data set:

1. Top: the performance obtained using the method named FULLY for SVM input;
2. Middle: the performance obtained using the method named DEEPER for SVM input;
3. Bottom: the fusion by average rule of the SVMs in 1 and 2.

**Table 1.** Performance of the two tested loss functions (boldface represents the best performance).

	Cross	CrossDD	Triplet	Triplet+Cross	Triplet+Cross +CrossDD
CAT	<b>83.05</b>	80.00	77.29	82.37	<b>83.05</b>
	69.15	71.19	72.20	71.53	76.27
	80.00	79.32	76.61	80.00	80.00
InfLar	86.94	93.75	90.42	91.25	<b>93.47</b>
	87.78	88.33	88.89	89.17	91.39
	89.44	92.36	90.56	91.39	92.64
BIRDz	94.49	93.35	94.08	94.90	94.56
	92.92	92.53	94.02	94.36	94.16
	94.52	93.91	94.84	<b>95.21</b>	94.88
RPE	84.52	84.58	85.43	85.75	84.97
	84.15	84.49	85.10	85.00	85.16
	84.73	85.08	85.17	<b>85.92</b>	85.48
Port	70.99	74.44	68.72	72.82	74.33
	69.57	70.54	70.96	71.83	73.22
	70.55	74.53	72.59	74.11	<b>74.65</b>

The clustering method is k-means for all methods, and the number of prototypes is in the set {15, 30, 45, 60}. Thus, four networks are trained using the four numbers of prototypes in the set; the four SVMs trained in this way are combined by average rule.

For the sake of computation time, we use a single network topology in this test, which is the first topology tested in [14] and the Siamese topology recommended by Mathworks (see the Appendix).

The columns of Table 1 report the following approaches:

- Cross: Binary Cross Entropy loss function coupled with base Adam (this is the best approach proposed [14]);
- CrossDD: Binary Cross Entropy loss function coupled with our new Adam variant DecayDGrad;
- Triplet: Triplet loss function coupled with base Adam.
- X+Y (columns 5 and 6): the fusion between X and Y.

From the results reported in Table 1, the following conclusions can be drawn:

- Triplet produces a result that is similar to Cross on three datasets but performs better than Cross in InfLar and worst in CAT;
- The fusion between Cross and Triplet boosts the performance of the base loss

- functions, except in the case of CAT;
- The fusion among all the different approaches (see bottom cells in the column Triplet+Cross and Triplet+Cross+CrossDD) produces the best average performance.

Table 2 reports results using combinations of the two loss functions on all eight topologies. Because running experiments on all five data sets was computationally too expensive, we chose to run them only on InfLar and Port because they are very different application problems.

In each cell of Table 2, the following four results are reported:

1. Top: Cross function coupled with FULLY for SVM input (the best approach proposed in [14]);
2. Upper: Triplet loss function coupled with FULLY for SVM input;
3. Lower: Fusion by average rule among Cross coupled with FULLY, Cross coupled with DEEPER, Triplet coupled with FULLY, and Triplet coupled with DEEPER;
4. Bottom: This is the fusion by average rule of SVMs 1 and 2 described for the method reported at the bottom of Table 1 but with the addition of CrossDD coupled with both FULLY and DEEPER.

The last row of Table 2 reports the fusions of #4 above for the numbered topologies.

**Table 2.** Performance varying the network topologies (topologies are described in [14] and reprinted in the Appendix; boldface represents the best performance).

Topology	InfLar	Port
Topology #1 -	86.94	70.99
	90.42	68.72
	91.39	74.11
	<b>92.64</b>	<b>74.65</b>
Topology #2 -	85.56	68.73
	<b>92.78</b>	70.02
	92.08	72.49
	91.67	<b>72.92</b>
Topology #3 -	79.44	60.23
	83.75	68.17
	<b>85.42</b>	68.41
	84.03	<b>69.17</b>
Topology #4 -	87.50	69.69
	91.25	68.29
	<b>92.22</b>	73.58
	90.97	<b>74.65</b>
Topology #5 -	84.03	60.00
	<b>89.44</b>	65.03
	87.64	64.95
	85.14	<b>69.69</b>
Topology #6 -	87.64	<b>73.48</b>
	88.61	68.07
	<b>91.25</b>	73.25
	90.56	73.24
Topology #7 -	79.44	66.03
	<b>91.39</b>	70.85
	85.00	70.85
	84.72	<b>71.85</b>
Topology #8 -	86.39	65.58
	87.22	66.55
	<b>91.11</b>	66.55
	90.56	<b>72.16</b>
Fusion 1-4	<b>92.78</b>	<b>75.09</b>
Fusion 1-6	91.53	74.45
Fusion 1-8	91.81	74.98

In [13], we showed that combining more than four networks using the same topology (but varying the clustering algorithm) failed to improve performance. Examining Table 2, we discover that changing the loss function and the method for building the dissimilarity space is beneficial when making an ensemble. We also observe that for all topologies

except #6 in the Portrait data set (Port), the best performance is not obtained by contrastive loss coupled with FULLY (as was the case in [14]); instead, on average, the new method DEEPER succeeds in boosting performance. Finally, we learn that adding CrossDD, our new Adam variant, to the ensemble for InfLar generally does not increase performance; CrossDD works very well with the first topology but performs worst with the other topologies. On Port, however, the addition of CrossDD generally does improve performance.

In Table 3, we compare our best results on InfLar and Port with the best ensembles reported in [12-14] that tested ensembles of SNNs and CNN subnets using all eight topologies. In addition, the performance of four well-known CNNs is reported for baseline comparison, along with their fusion (eCNN) by average rule. The fine-tuning of the CNNs pretrained on ImageNet was performed with the following training options: batch size: 30; max epoch: 20 (for all the networks with no freezing). The row "Fusion x-y + eCNN" is the sum rule between Fusion x-y (see Table 2) and eCNN. Before the fusion, the score of Fusion x-y and eCNN are normalized to mean 0 and standard deviation 1.

**Table 3.** Performance accuracy obtained considering different standard CNNs and other Siamese approaches (xxx\* means that it does not converge and boldface represents best performance).

Method	InfLar	Port
[12]	74.86	xxx*
[13]	89.86	71.42
[14]	91.10	73.05
Fusion 1-4	92.78	75.09
Fusion 1-6	91.53	74.45
Fusion 1-8	91.81	74.98
GoogLeNet	90.42	80.38
VGG16	91.53	86.51
VGG19	92.22	82.42
GoogLeNetP365	93.61	80.91
eCNN	94.03	86.41
Fusion 1-4 + eCNN	<b>94.44</b>	<b>86.84</b>
Fusion 1-6 + eCNN	<b>94.44</b>	<b>86.84</b>
Fusion 1-8 + eCNN	94.31	<b>86.84</b>

As can be observed in Table 3, the proposed ensembles outperform previous methods based on Siamese networks and boosts the performance of the ensemble of CNNs. On the data set InfLar, the performance of the best standalone topology (see Table 2) is 92.78, which is comparable with the performance obtained by a CNN; however, on the Port data set, where our new Adam variant increased performance, the performance gap between the CNNs and Siamese networks is still significant. The approach proposed in this work also greatly improves previous Siamese methods applied to this dataset.

## 5. Conclusion

This paper proposes an image classification system that like several recent studies generates dissimilarity spaces from which features are extracted and trained on a set of SVMs. The objective of this study was to produce a high performing ensemble of Siamese networks based on combining different topologies, loss functions, and optimization methods (with one new Adam variant proposed here) from which features could be extracted for training the SVMs.

Results on five cross-domain image data sets demonstrate the superior power of the proposed approach compared with previous works using ensembles of Siamese networks. Comparison with the state-of-the-art confirms that the fusion of the different topologies, loss functions, and optimization approaches methods is a feasible way for generating a robust and highly generalizable image classification system.

In the future, we intend to validate our approach on additional cross-domain image data sets and investigate more techniques for building an ensemble of Siamese networks.

**Author Contributions:** Conceptualization, A.L. and L.N.; methodology, L.N.; software, G.M., D.S., and L.N.; validation, L.N.; formal analysis, A.L.; resources—S.B.; writing—original draft preparation, A.L., S.B., and L.N.; and writing—review and editing A.L., S.B., and L.N.

**Funding:** This research received no external funding.

**Acknowledgments:** The authors are grateful to NVIDIA Corporation for supporting this research with the donation of a Titan Xp GPU.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## References

1. Pełalska, E. and R.P. Duin, *The Dissimilarity Representation for Pattern Recognition - Foundations and Applications*. 2005, Singapore: World Scientific.
2. Cha, S. and S. Srihari. *Writer Identification: Statistical Analysis and Dichotomizer*. in *SSPR/SPR*. 2000.
3. Oliveira, L., E. Justino, and R. Sabourin, *Off-line Signature Verification Using Writer-Independent Approach*. 2007 International Joint Conference on Neural Networks, 2007: p. 2539-2544.
4. Hanusiak, R.K., et al., *Writer verification using texture-based features*. *International Journal on Document Analysis and Recognition (IJ DAR)*, 2011. **15**: p. 213-226.
5. Zottesso, R.H.D., et al., *Bird species identification using spectrogram and dissimilarity approach*. *Ecological Informatics*, 2018. **48**: p. 187-197.
6. Souza, V.L.F., A. Oliveira, and R. Sabourin, *A Writer-Independent Approach for Offline Signature Verification using Deep Convolutional Neural Networks Features*. 2018 7th Brazilian Conference on Intelligent Systems (BRACIS), 2018: p. 212-217.
7. Pełalska, E. and R.P. Duin, *Dissimilarity representations allow for building good classifiers*. *Pattern Recognit. Lett.*, 2002. **23**: p. 943-956.
8. Nguyen, G., M. Worring, and A. Smeulders. *Similarity learning via dissimilarity space in CBIR*. in *MIR '06*. 2006.
9. Theodorakopoulos, I., et al., *HEp-2 cells classification via sparse representation of textural features fused into dissimilarity space*. *Pattern Recognit.*, 2014. **47**: p. 2367-2378.
10. Hernández-Durán, M., Y.P. Calaña, and H.M. Vazquez. *Low-Resolution Face Recognition with Deep Convolutional Features in the Dissimilarity Space*. in *IWAIPR*. 2018.
11. Mekhazni, D., et al. *Unsupervised Domain Adaptation in the Dissimilarity Space for Person Re-identification*. in *ECCV*. 2020.
12. Nanni, L., et al., *Spectrogram classification using dissimilarity space*. *Sensors*, 2020. **10**(12): p. 4176.
13. Nanni, L., et al., *Animal sound classification using dissimilarity spaces*. *Applied Sciences*, 2020. **10**(23): p. 8578.
14. Nanni, L., et al., *Experiments of image classification using dissimilarity spaces built with siamese networks*. *Sensors*, 2021. **21**(1573): p. 2-18.
15. Costa, Y.M.G., et al., *The dissimilarity approach: a review*. *Artificial Intelligence Review*, 2019. **53**: p. 2783-2808.
16. Chicco, D., *Siamese neural networks: An overview*, in *Artificial Neural Networks. Methods in Molecular Biology*, H. Cartwright, Editor. 2020, Springer Protocols: Humana, New York, NY. p. 73-94.
17. Bromley, J., et al. *Signature Verification Using A "Siamese" Time Delay Neural Network*. in *Int. J. Pattern Recognit. Artif. Intell.* 1993.
18. Agrawal, A., *Dissimilarity learning via Siamese network predicts brain imaging data*. arXiv: Neurons and Cognition, 2019.
19. San Biagio, M., et al., *Heterogeneous auto-similarities of characteristics (hasc): Exploiting relational information for classification*, in *IEEE Computer Vision (ICCV13)*. 2013: Sydney, Australia. p. 809-816.
20. Feig, E. and S. Winograd, *Fast algorithms for the discrete cosine transform*. *IEEE Transactions on Signal Processing*, 1992. **49**(9): p. 2174-2193.

21. Kingma, D.P. and J. Ba, *Adam: A Method for Stochastic Optimization*. CoRR, 2015. **abs/1412.6980**.
22. Dubey, S., et al., *diffGrad: An Optimization Method for Convolutional Neural Networks*. IEEE Transactions on Neural Networks and Learning Systems, 2020. **31**(11): p. 4500-4511.
23. Nanni, L., G. Maguolo, and A. Lumini, *Exploiting Adam-like Optimization Algorithms to Improve the Performance of Convolutional Neural Networks*. ArXiv, 2021. **abs/2103.14689**.
24. You, K., M. Long, and M.I. Jordan, *How Does Learning Rate Decay Help Modern Neural Networks*. arXiv: Learning, 2019.
25. Zhang, S.-h., et al., *Automatic Bird Vocalization Identification Based on Fusion of Spectral Pattern and Texture Features*. 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2018: p. 271-275.
26. Pandeya, Y.R., D. Kim, and J. Lee, *Domestic cat sound classification using learned features from deep neural nets*. Applied Sciences, 2018. **8**(10): p. 1949.
27. Pandeya, Y.R. and J. Lee, *Domestic Cat Sound Classification Using Transfer Learning*. Int. J. Fuzzy Logic and Intelligent Systems, 2018. **18**: p. 154-160.
28. Moccia, S., et al., *Learning-based classification of informative laryngoscopic frames*. Computer Methods and Programs in Biomedicine, 2018. **158**: p. 21-30.
29. Nanni, L., et al., *Texture descriptors ensembles enable image-based classification of maturation of human stem cell-derived retinal pigmented epithelium*. PLoS ONE, 2016(e0149399).
30. Liu, S., et al., *Novel features for art movement classification of portrait paintings*. Image and Vision Computing, 2021. **108**: p. 104121.

## Appendix

For ease of reference, a description of the eight SN networks in [14] are reprinted below.

CNN Siamese Networks (1–8) layers.

Siamese Network 1				
Layers	Activations	Learnable	Filter Size	Num. of Filters
Input Layer	224 × 224			
2D Convolution	215 × 215 × 64	6464	10 × 10	64
ReLU	215 × 215 × 64	0		
Max Pooling	107 × 107 × 64	0	2 × 2	
2D Convolution	101 × 101 × 128	401,536	7 × 7	128
ReLU	101 × 101 × 128	0		
Max Pooling	50 × 50 × 128	0	2 × 2	
2D Convolution	47 × 47 × 128	262,272	4 × 4	128
ReLU	47 × 47 × 128	0		
Max Pooling	23 × 23 × 128	0	2 × 2	
2D Convolution	19 × 19 × 64	204,864	5 × 5	64
ReLU	19 × 19 × 64	0		
Fully Connected	4096	94,638,080		
Siamese Network 2				
Layers	Activations	Learnable	Filter Size	Num. of Filters
Input Layer	224 × 224	0		
2D Convolution	220 × 220 × 64	1664	5 × 5	64
LeakyReLU	220 × 220 × 64	0		
2D Convolution	216 × 216 × 64	102,464	5 × 5	64

LeakyReLU	216 × 216 × 64	0		
Max Pooling	108 × 108 × 64	0	2 × 2	
2D Convolution	106 × 106 × 128	73,856	3 × 3	128
LeakyReLU	106 × 106 × 128	0		
2D Convolution	104 × 104 × 128	147,584	3 × 3	128
LeakyReLU	104 × 104 × 128	0		
Max Pooling	52 × 52 × 128	0	2 × 2	
2D Convolution	49 × 49 × 128	262,272	4 × 4	128
LeakyReLU	49 × 49 × 128	0		
Max Pooling	24 × 24 × 128	0	2 × 2	
2D Convolution	20 × 20 × 64	204,864	5 × 5	64
LeakyReLU	20 × 20 × 64	0	5 × 5	
Fully Connected	2048	52,430,848		
<b>Siamese Network 3</b>				
<b>Layers</b>	<b>Activations</b>	<b>Learnable</b>	<b>Filter Size</b>	<b>Num. Filters</b>
Input Layer	224 × 224			
2D Convolution	55 × 55 × 128	6400	7 × 7	128
Max Pooling	27 × 27 × 128	0	2 × 2	
2D Convolution	23 × 23 × 256	819,456	5 × 5	256
ReLU	23 × 23 × 256	0		
2D Convolution	19 × 19 × 128	819,328	5 × 5	128
Max Pooling	9 × 9 × 128	0	2 × 2	
2D Convolution	7 × 7 × 64	73,792	3 × 3	64
ReLU	7 × 7 × 64	0		
Max Pooling	3 × 3 × 64	0	2 × 2	
Fully Connected	4096	2,363,392		
<b>Siamese Network 4</b>				
<b>Layers</b>	<b>Activations</b>	<b>Learnable</b>	<b>Filter Size</b>	<b>Num. of Filters</b>
Input Layer	224×224			
2D Convolution	218 × 218 × 128	6400	7 × 7	128
Max Pooling	54 × 54 × 128	0	4 × 4	
ReLU	54 × 54 × 128	0		
2D Convolution	50 × 50 × 256	819,456	5 × 5	256
ReLU	50 × 50 × 256	0		
2D Convolution	48 × 48 × 64	147,520	3 × 3	64
Max Pooling	24 × 24 × 64	0	2 × 2	
2D Convolution	22 × 22 × 128	73,856	3 × 3	128
ReLU	22 × 22 × 128	0		
2D Convolution	18 × 18 × 64	204,864	5 × 5	64
Fully Connected	4096	84,938,752		
<b>Siamese Network 5</b>				
<b>Layers</b>	<b>Activations</b>	<b>Learnable</b>	<b>Filter Size</b>	<b>Num. of Filters</b>
Input Layer	224×224			
2D Convolution	215 × 215 × 64	6464	10 × 10	64
Max Pooling	107 × 107 × 64	0	2 × 2	
ReLU	107 × 107 × 64	0		
2D Convolution	26 × 26 × 128	401,536	7 × 7	128
ReLU	26 × 26 × 128	0		
2D Convolution	9 × 9 × 128	409,728	5 × 5	128
ReLU	9 × 9 × 128	0		

2D Convolution	$6 \times 6 \times 64$	131,136	$4 \times 4$	64
ReLU	$6 \times 6 \times 64$	0		
Fully Connected	4096	9,441,280		
<b>Siamese Network 6</b>				
<b>Layers</b>	<b>Activations</b>	<b>Learnable</b>	<b>Filter Size</b>	<b>Num. of Filters</b>
Input Layer	224x224			
2D Convolution	$218 \times 218 \times 64$	3200	$7 \times 7$	64
Max Pooling	$109 \times 109 \times 64$	0	$2 \times 2$	
ReLU	$109 \times 109 \times 64$	0		
2D Convolution	$107 \times 107 \times 128$	73,856	$3 \times 3$	128
Max Pooling	$53 \times 53 \times 128$	0	$2 \times 2$	
ReLU	$53 \times 53 \times 128$	0		
2D Convolution	$53 \times 53 \times 64$	8,256	$1 \times 1$	64
ReLU	$53 \times 53 \times 64$	0		
2D Convolution	$51 \times 51 \times 128$	73,856	$3 \times 3$	128
ReLU	$51 \times 51 \times 128$	0		
Max Pooling	$25 \times 25 \times 128$	0	$2 \times 2$	
2D Convolution	$25 \times 25 \times 128$	16,512	$1 \times 1$	128
ReLU	$25 \times 25 \times 128$	0		
2D Convolution	$22 \times 22 \times 64$	131,136	$4 \times 4$	64
Max Pooling	$11 \times 11 \times 64$	0	$2 \times 2$	
ReLU	$11 \times 11 \times 64$	0		
Fully Connected	4096	31,723,520		
<b>Siamese Network 7</b>				
<b>Layers</b>	<b>Activations</b>	<b>Learnable</b>	<b>Filter Size</b>	<b>Num. of Filters</b>
Input Layer	224x224			
Dropout Layer	224x224	0		
2D Convolution	$218 \times 218 \times 64$	3200	$7 \times 7$	64
Max Pooling	$109 \times 109 \times 64$	0	$2 \times 2$	
2D Convolution	$105 \times 105 \times 128$	204,928	$5 \times 5$	128
Max Pooling	$52 \times 52 \times 128$	0	$2 \times 2$	
2D Convolution	$48 \times 48 \times 64$	204,864	$5 \times 5$	64
Max Pooling	$24 \times 24 \times 64$	0	$2 \times 2$	
2D Convolution	$22 \times 22 \times 256$	147,712	$3 \times 3$	256
Max Pooling	$11 \times 11 \times 256$	0	$2 \times 2$	
2D Convolution	$9 \times 9 \times 256$	590,080	$3 \times 3$	256
Fully Connected	4096	16,781,312		
<b>Siamese Network 8</b>				
<b>Layers</b>	<b>Activations</b>	<b>Learnable</b>	<b>Filter Size</b>	<b>Num. of Filters</b>
Input Layer	224x224			
2D Convolution	$215 \times 215 \times 32$	3232	$10 \times 10$	32
Max Pooling	$107 \times 107 \times 32$	0	$2 \times 2$	
ReLU	$107 \times 107 \times 32$	0		
2D Grouped Convolution	$101 \times 101 \times 64$	50,240	$7 \times 7$	64
2D Convolution	$97 \times 97 \times 128$	204,928	$5 \times 5$	128
Max Pooling	$48 \times 48 \times 128$	0	$2 \times 2$	
ReLU	$48 \times 48 \times 128$	0		
2D Grouped Convolution	$46 \times 46 \times 256$	147,712	$3 \times 3$	256
Fully Connected	4096	2,218,790,912		