SUPPLEMENTARY MATERIAL TABLE OF CONTENTS

A TOTAL NUMBER OF TRAINABLE NETWORK PARAMETERS

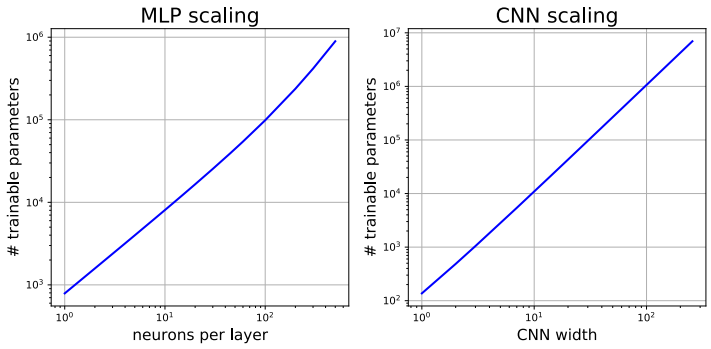


Figure 5: Number of trainable network parameters at increasing network size. The input shape is assumed to be (28,28,1). The structure of the MLP and the CNN is described in section 2.1.

B PSEUDOCODE TO CREATE THE BALANCED PAIRS AND REDUCED DATASET

Below, we introduce the pseudocode of the strategies used to introduce PLN and SLN in the Scenario 1 (sparse connections) and 2 (dense connections). In particular, Algorithm 1 explains how we create balanced positive and negative pairs, Algorithm 2 explains how we create a reduced and balanced version of a dataset. Algorithms 3 and 4 (5 and 6) describe the pipeline to train the network using sparse (dense) dataset relations in presence of PLN and SLN respectively. A pictorial view of the paths leading to training in the different setups considered is depicted in Fig. 6.

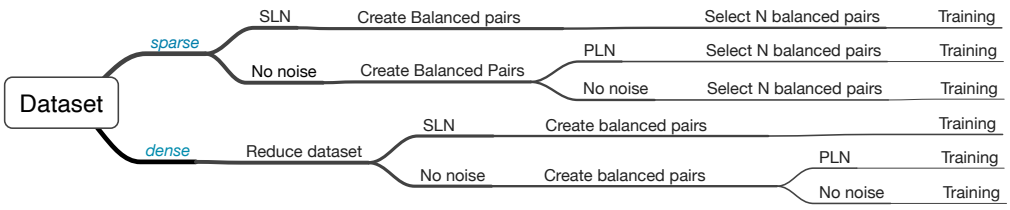


Figure 6: Pipeline guiding from the original dataset to training in the different setups considered.

Algorithm 1 Creating Balanced Pairs

function CREATEBALANCEDPAIRS(X, Y_S, n_c)
  pairs=[]
  Y_P=[]
  $M_d \leftarrow [m_1, \dots m_{n_c}]$ list of number of samples in each class
  $X_{N_d} \leftarrow [X_1, \dots X_{n_c}]$ list of images in each class
  for $d = 1, 2, \dots, n_c$ **do**
    for $i = 1, 2, \dots, m_d$ **do**
      pairs_pos = {X_d[i], X_d[i+1]}
      d_x = (d + random-integer(1,n_c-1)) % n_c select different random class
      j = random-integer(1,m_{d_x}) select random element in the class
      pair_neg = {X_d[i], X_{d_x}[j]}
      pairs = pairs + {pair_pos, pair_neg}
      Y_P = Y_P + [1, 0]
    end for
  end for
  return pairs, Y_P
end function

Algorithm 2 Create Reduced dataset

function REDUCEDATASET(X, Y_S, n_c, NewSize)
  $m \leftarrow$ int(NewSize/n_c)
  indices=[ ]
  for $d = 1, 2, \dots, n_c$ **do**
    class_indices = where(Y_S==d)
    m_class_indices=random.choice(class_indices, m)
    indices.append(m_class_indices)
  end for
  return X[indices], Y_S[indices]
end function

Algorithm 3 *Sparse* Pair Label Noise (PLN)

$N \leftarrow$ number of pairs to train the SNN with
$\tilde{q} \leftarrow$ probability to apply transformation $\mathcal{T}_{\text{PLN}}$
$X, Y_S \leftarrow$load(dataset)
$n_c \leftarrow$len(unique(Y_S))
pairs, $Y_P \leftarrow$ CREATEBALANCEDPAIRS(X, Y_S, n_c)
for $i = 1, 2, \dots, n_s$ **do**
  indices=[]
  $Y_P \leftarrow \mathcal{T}_{\text{PLN}}(\tilde{q}, Y_P)$ fraction of Y_P gets randomized
  indices+=random.choice(where(Y_P ==0), N/2) select balanced N pairs
  indices+=random.choice(where(Y_P ==1), N/2)
  pairs$\leftarrow$ pairs[indices]
  $Y_P \leftarrow Y_P$[indices]
  train
end for

Algorithm 4 *Sparse* Single Label Noise (SLN)

$N \leftarrow$ number of pairs to train the SNN with
$q \leftarrow$ probability to apply transformation $\mathcal{T}_{\text{SLN}}$
$X, Y_S \leftarrow$load(dataset)
$n_c \leftarrow$len(unique(Y_S))
for $i = 1, 2, \ldots, n_s$ **do**
  $Y'_S \leftarrow \mathcal{T}_{\text{SLN}}(\tilde{q}, Y_S)$ — fraction of Y_S gets randomized
  pairs, $Y_P \leftarrow$CREATEBALANCEDPAIRS(X, Y'_S, n_c)
  indices+=random.choice(where(Y_P ==0), N/2) — select balanced N pairs
  indices+=random.choice(where(Y_P ==1), N/2)
  pairs $\leftarrow$ pairs[indices]
  $Y_P \leftarrow Y_P$[indices]
  train
end for

Algorithm 5 *Dense* Pair Label Noise (PLN)

$N \leftarrow$ number of pairs to train the SNN with
$\tilde{q} \leftarrow$ probability to apply transformation $\mathcal{T}_P$
$X, Y_S \leftarrow$load(dataset)
$n_c \leftarrow$len(unique(Y_S))
for $i = 1, 2, \ldots, n_s$ **do**
  $X', Y'_S \leftarrow$ REDUCEDATASET(X, Y_S, n_c, N/2)
  pairs, $Y_P \leftarrow$CREATEBALANCEDPAIRS(X', Y'_S, n_c)
  $Y_P \leftarrow \mathcal{T}_{\text{PLN}}(\tilde{q}, Y_P)$ — fraction of Y_P gets randomized
  train
end for

Algorithm 6 *Dense* Single Label Noise (SLN)

$N \leftarrow$ number of pairs to train the SNN with
$q \leftarrow$ probability to apply transformation $\mathcal{T}_S$
$X, Y_S \leftarrow$load(dataset)
$n_c \leftarrow$len(unique(Y_S))
for $i = 1, 2, \ldots, n_s$ **do**
  $X', Y'_S \leftarrow$ REDUCEDATASET(X, Y_S, n_c, N/2)
  $Y'_S \leftarrow \mathcal{T}_{\text{SLN}}(\tilde{q}, Y'_S)$ — fraction of Y'_S gets randomized
  pairs, $Y_P \leftarrow$CREATEBALANCEDPAIRS(X', Y'_S, n_c)
  train
end for

C EFFECTIVE NOISE DERIVATION

We start by considering the amount of effective noise (real mislabeling) introduced by the pair label transformation

$$\mathcal{T}_{\text{PLN}}(\widetilde{q}) : y_i^P \to \text{Random}(\{0,1\}) \qquad \text{with probability} \quad \widetilde{q}\,. \tag{7}$$

Each time we apply this transformation, the probability of a change in the pair label is 1/2, so the effective error probability is:

$$P_{\text{PLN}} = \frac{\widetilde{q}}{2}\,. \tag{8}$$

This computation is slightly more complicated in the SLN case. Indeed, if we apply the following transformation

$$\mathcal{T}_{\text{SLN}}(q) : {y_i}^S \to \text{Random}(\{1,\ldots,n_c\}) \qquad \text{with probability} \quad q\,, \tag{9}$$

on the initial dataset labels y_i^S, and we then create the pairs, the probability that one (and only one) element in a pair has been operated by $\mathcal{T}_{\text{SLN}}$ is

$$P_{1L} = 2q(1-q)\,, \tag{10}$$

while the probability that both elements have been operated by $\mathcal{T}_{\text{SLN}}$ is

$$P_{2L} = q^2\,. \tag{11}$$

Now the question is: what is the probability that this single label operation (we recall that the term *single label* regards the application of $\mathcal{T}_{\text{SLN}}$ on the label of one or both pair elements and not on the *pair label*) leads to effective pair label corruption? Let us assume that we have a pair of images belonging to different classes $y^P = 0$. The probability that the transformation of a single image label changes the pair label is equal to the likelihood that the same operation over both images effectively changes the pair label. The value of that probability is the following:

$${Q^{0\to1}}_{1L} = {Q^{0\to1}}_{2L} = \frac{1}{n_c}\,. \tag{12}$$

The same reasoning can be applied to pairs of objects belonging to the same class, $y^P = 1$, and leads to

$${Q^{1\to0}}_{1L} = {Q^{1\to0}}_{2L} = \frac{(n_c-1)}{n_c}\,. \tag{13}$$

Creating a balanced dataset where half of the pairs are equal and half are different is common practice. Therefore, we create a dataset where

$$P_{y^P=1} = P_{y^P=0} = \frac{1}{2}\,. \tag{14}$$

Finally, we are now ready to estimate the amount of real noise that is introduced in our dataset corrupting single images labels. This is given by:

$$\begin{aligned} P_{\text{SLN}} \quad &= P_{y^P=1}\,(P_{1L}{Q^{0\to1}}_{1L} + P_{2L}{Q^{0\to1}}_{2L}) \\ &\qquad + P_{y^P=0}(P_{1L}\,{Q^{1\to0}}_{1L} + P_{2L}{Q^{1\to0}}_{2L}) \\ &= \tfrac{1}{2}(P_{1L} + P_{2L})({Q^{0\to1}}_{1L} + {Q^{1\to0}}_{1L}) \\ &= q - \tfrac{1}{2}q^2\,. \end{aligned} \tag{15}$$

Requiring that the effective dataset noise is the same in SLN and PLN setups, leads us to Eq. (5). We want to stress that PLN and SLN impose different constraints on the training process. PLN is a balanced noise source as the probability of transforming even pairs into odd ones, and vice versa is the same. On the other hand, SLN is an unbalanced source of noise, i.e., the probability that $\mathcal{T}_{\text{SLN}}$ transforms equal pairs into different ones, $(n_c-1)/n_c$, is in general much higher than the opposite case, $1/n_c$. Moreover, as opposed to classification tasks, in Siamese networks and contrastive learning, noise can generally lead to inconsistent relations in the training set. A similarity relation is defined by reflexivity, symmetry, and transitivity, but the appearance of noise can compromise this last property. In fact, PLN, randomly shuffling pair labels, leads to inconsistent relations in the dataset (see Fig. 2). This effect becomes more apparent as we increase the density (number of links) in our training set. On the other hand, similarity breaking does not appear in SLN, where the similarity relations may go against image features but are always self-consistent.

D OUTPUT LAYER AND LOSS FUNCTION

We perform our experiments using the two different output layers and loss functions described below.

Euclidean distance and Contrastive Loss. In this first case, the Siamese NN output layer computes the Euclidean distance between the output vectors of the Siamese branches, $\vec{z}(x)$. Therefore, the model prediction that quantifies the similarity between the two images in a pair is given by:

$$d_i = ||\vec{z}(x_1^i) - \vec{z}(x_2^i)||_2 \,. \tag{16}$$

We then train the network considering the contrastive loss function Hadsell et al. (2006):

$$\mathcal{L}(y^P, d) = \frac{1}{N_{\text{pairs}}} \sum_i \left[y_i^P \, d_i^2 + (1 - y_i^P) \left[\max(0, m - d_i)\right]^2 \right], \tag{17}$$

where y_i^P is the true label and m sets the threshold at which the network classifies a given pair as similar or different. During training, the network tries to minimize $\mathcal{L}$ by collapsing similar samples and pulling apart different samples by a distance equal to the margin, m. The accuracy is given by:

$$\text{acc} = 1 - \text{err} = 1 - \frac{1}{2N} \sum_i |y_i^P - \hat{y}(d_i)| \,, \quad \text{where} \quad \hat{y}(d_i) = [\mathbb{1}_{d<m/2}](d_i). \tag{18}$$

In all experiments, we choose the margin to be $m = 1$.

Cosine similarity and Cosine Embedding Loss. In this setup, the output layer computes the cosine similarity between the output vectors of the Siamese branches. The model prediction is thus given by:

$$s_i = \cos\left(\frac{\vec{z}(x_1^i) \cdot \vec{z}(x_2^i)}{||\vec{z}(x_1^i)||_2 ||\vec{z}(x_2^i)||_2} \right) . \tag{19}$$

We train the network using the Cosine Embedding Loss function,

$$\mathcal{L}(y^P, s) = \frac{1}{N_{\text{pairs}}} \sum_i \left[y_i^P \, (1 - s_i) + (1 - y_i^P) \max(0, s_i - \cos(\alpha)) \right], \tag{20}$$

according to which similar images should give rise to vectors pointing in roughly the same direction. In contrast, the angle between vectors coming from different images should be larger than or equal to α. In this model, we compute the accuracy as:

$$\text{acc} = 1 - \frac{1}{2N} \sum_i |y_i^P - \hat{y}(s_i)|, \quad \text{where} \quad \hat{y}(s_i) = [\mathbb{1}_{s>\cos(\alpha/2)}](s_i). \tag{21}$$

In all experiments, we choose $\alpha = \pi/3$.

E UNBALANCED TRAIN ERROR WITH PLN DIBS

Here, we provide some additional information about the properties of the asymptotic training error arising with PLN on a dense pairs dataset. As mentioned in the main text, PLN breaks transitivity, leading to inconsistent relations that the NN cannot satisfy. In fact, the margin-based loss function tends to collapse images belonging to the same class to a single point and separate different images by a distance equal to or greater than the margin. Some inconsistencies appearing in this setup are shown in Fig. 2. Nevertheless, it is not clear how the network deals with such contradictions at this point. The final training error could arise from three distinct behaviors: collapsing different pairs, separating equal pairs, or a mixture of the two. Contrary to the third option, the first two cases would lead to an unbalanced asymptotic train error (similar pairs systematically mislabeling or vice-versa). In order to answer this question, we compute the confusion matrix on the asymptotic training set, varying the number of classes and PLN. This corresponds to studying the source of the results shown in the top-right corner of Fig. 3. Our findings are shown in Fig. 7. We see that the network is biased towards "False different" classification. That is, the NN tends to classify equal pairs ($y^S = 1$) as different ones ($y^S = 0$). We can provide some intuition about why this happens. As shown in Fig. 8 we can have transitivity breaking inside similar pairs chains and among vertices belonging to different

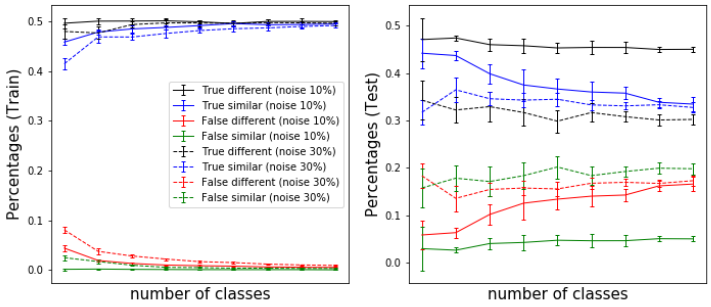


Figure 7: Results about normalized confusion matrix computed on the asymptotic training error (left) and its related test error (right) at varying number of classes and PLN. These results are referred to FMNIST using the dense pair regime (scenario 2). The experimental setup is the same used in Fig. 3.

classes. Notice that more than one mislabeling inside equal images chains (green chains) does not lead to transitivity breaking. Therefore, the impact of this mislabeling on the training error is almost negligible in standard situations, where the number of pairs is much larger than the number of classes. On the other hand, mislabeling on paths involving different classes often leads to inconsistencies. Thus, the dominant contribution to transitivity breaking comes from the diagrams in the last 2 lines of Fig. 8. In this specific case, the configuration that minimizes the amount of training error is the one where the noise-induced similar pair is misclassified, leading to a "False different" kind of error. We also studied the impact of the training error unbalancing on the test error (RHS of Fig. 7). Surprisingly, while in the low noise regime (10%), the test error is also biased towards the "False different" pair classification, increasing the amount of noise to 30%, the total test error increases and becomes more balanced.

Despite the fact that these results are obtained using the dataset preparation discussed in the main text, we believe that changing this procedure would not lead to qualitatively different results as long as it allows for "triangle" configurations as shown in Fig.8.

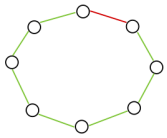
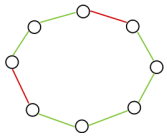
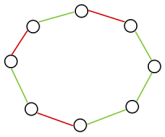
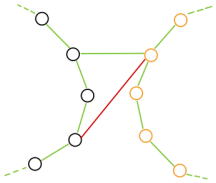
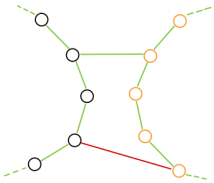
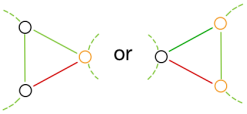
CONFIGURATION	COLLAPSED CONFIGURATION	TRANSITIVITY BREAKING
		YES
		NO
		NO
		YES
		YES

Figure 8: Left: Examples of pairs configurations within the same (first 3 lines) and different classes (last two lines) in the presence of PLN. Center: configuration after similar pair collapse. If this can not be consistently reached, the minimal extended configuration (triangle with 1 red and 2 green edges) is displayed. Right: Presence of transitivity breaking coming from PLN.

F ONLINE/OFFLINE CORRESPONDENCE

The Deep Bootstrap framework Nakkiran et al. (2021) introduces a correspondence between online and offline settings for classification problems. The proposal establishes a relation between the *Real World* (offline), which has a fixed number of samples in the training set, with the *Ideal World* (online), where the optimizer's updates always use fresh samples. The relation between Real and Ideal scenarios is manifested by the following alternative decomposition of the test error Nakkiran et al. (2021):

$$\text{TestError}(f_t) = \text{TestError}(f_t^{\text{iid}}) + [\text{TestError}(f_t) - \text{TestError}(f_t^{\text{iid}})],$$

where, f_t refers to a neural network after t optimizer steps in the Real World with a fixed number of samples, and f_t^{iid} is a network in the Ideal World that is trained as f_t but without re-using samples. This means that the optimization over minibatches gets new samples at each gradient iteration in the online setting. Note that the decomposition in the equation above makes the *bootstrap error* (the subtraction in brackets) explicit.[6] This measures how well models trained offline represent the online regime. Nakkiran et al. (2021) show that the online and offline test errors coincide if the NN is trained for a fixed number of weight updates. This gives empirical evidence that an online/offline correspondence holds for classification problems in several network models for supervised image classification. The conjecture suggests that understanding generalization in the Real World can be effectively reduced to an optimization problem in online learning.

G ADDITIONAL FIGURES

G.1 DOUBLE DESCENT PLOTS

Below we describe the network architectures and the training procedure used to study the DD behavior. Interestingly, the setups for similarity learning are way richer than the one often used for classification tasks. In classification, one can vary the network architecture and hyperparameters, but there is not much freedom regarding the loss function choice, which is usually the cross-entropy loss. In our Siamese setup, besides altering architecture and hyperparameters, we can vary how the distance between the output branches is measured and use their corresponding losses (in particular, we consider Euclidean distance and cosine similarity with contrastive and cosine loss, respectively.)

In addition, we note that all plots showing DD curves in this section are provided in log x scale, and the steepness may be misleading. These plots show the same behaviour observed in Nakkiran et al. (2020a) as the slope is slowly decreasing (see, e.g., Fig. 1 in Nakkiran et al. (2020a) in linear x scale). In our results, the networks extend up to the ratio $\frac{\#\text{parameters}}{\#\text{samples}} = 10^2, 10^3$ (see Fig. 5 in Sec. A), which is comparable to those in Nakkiran et al. (2020a). In fact, we studied the models training dynamics and their convergence, see Sec. G.3 for some examples.

MLP: The MLP architecture we consider has 3 layers of equal size with ReLu activation function. We train the network for 2k epochs also varying the number of neurons per layer. We consider contrastive loss using the Euclidean distance between branch outputs, and we use 6k (9k) train (test) pairs. We use ADAM optimizer to train the network.

CNN-Euclidean Distance: We consider a CNN with 4 layers of different size (see Section 2.1). We vary the CNN with parameter k in the sparse and the dense configuration of input pairs. We consider contrastive loss using Euclidean distance between branch outputs, and we use 6k (9k) train (test) pairs. We train the network for 2k epochs using ADAM optimizer.

CNN-Cosine Distance: We consider a CNN with 4 layers of different size (see Section 2.1). We vary the CNN width parameter k in input pairs' sparse and dense configuration. We consider cosine embedding loss using cosine similarity between branch outputs, and we use 6k (9k) train (test) pairs. We train the network for 2k epochs using ADAM optimizer.

[6] See Appendix C in Nakkiran et al. (2021) for connections to the nonparametric bootstrap in statistics Efron (1979); Efron & Tibshirani (1986). Also, as noted in Nakkiran et al. (2021), the bootstrap error can be related to algorithmic stability (e.g., Bousquet & Elisseeff (2001); Hardt et al. (2016)).

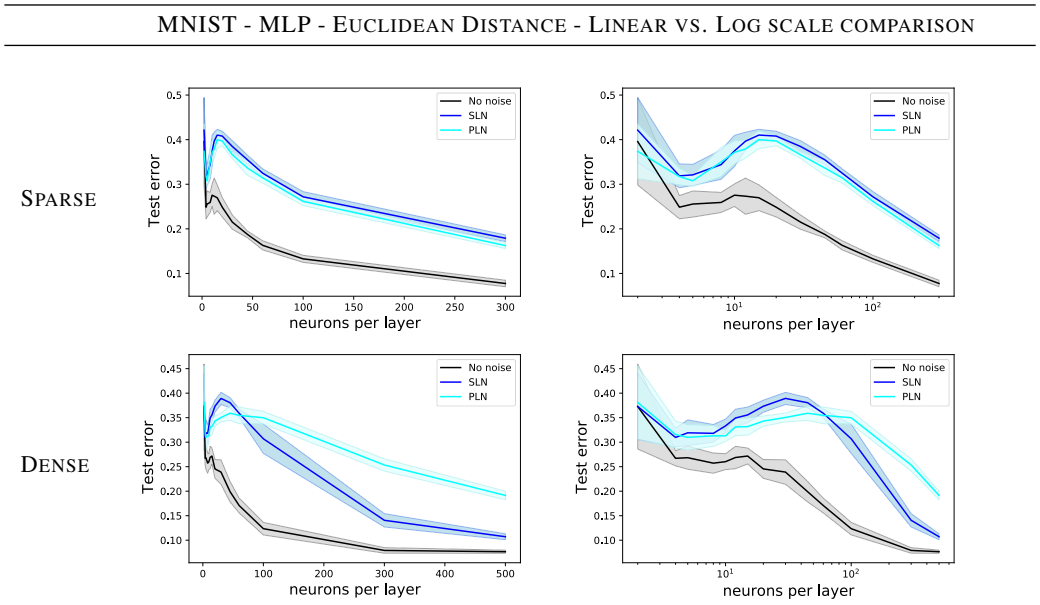


Figure 9: MNIST test error displayed using linear (left) and log scale (right) for MLP architecture.

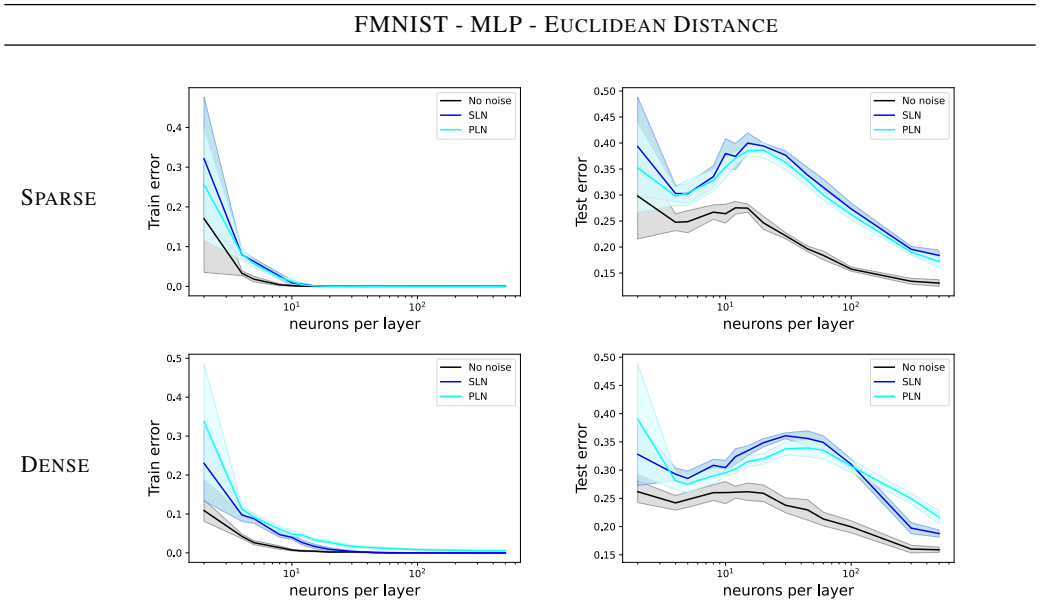


Figure 10: FMNIST training (left) and test error (right) for MLP architecture.

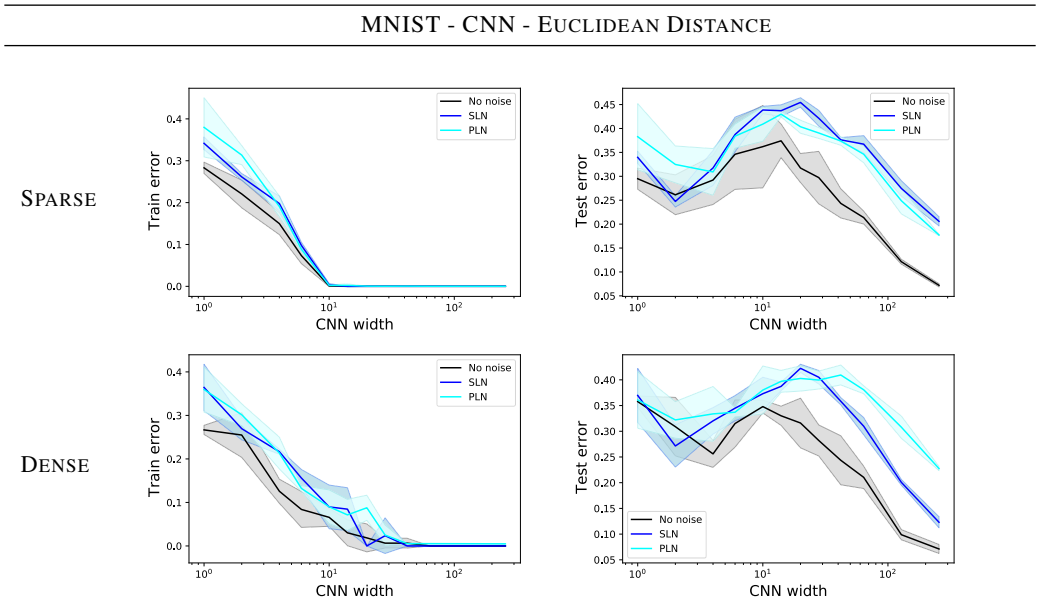


Figure 11: MNIST training (left) and test error (right) for CNN-Euclidean distance configuration.

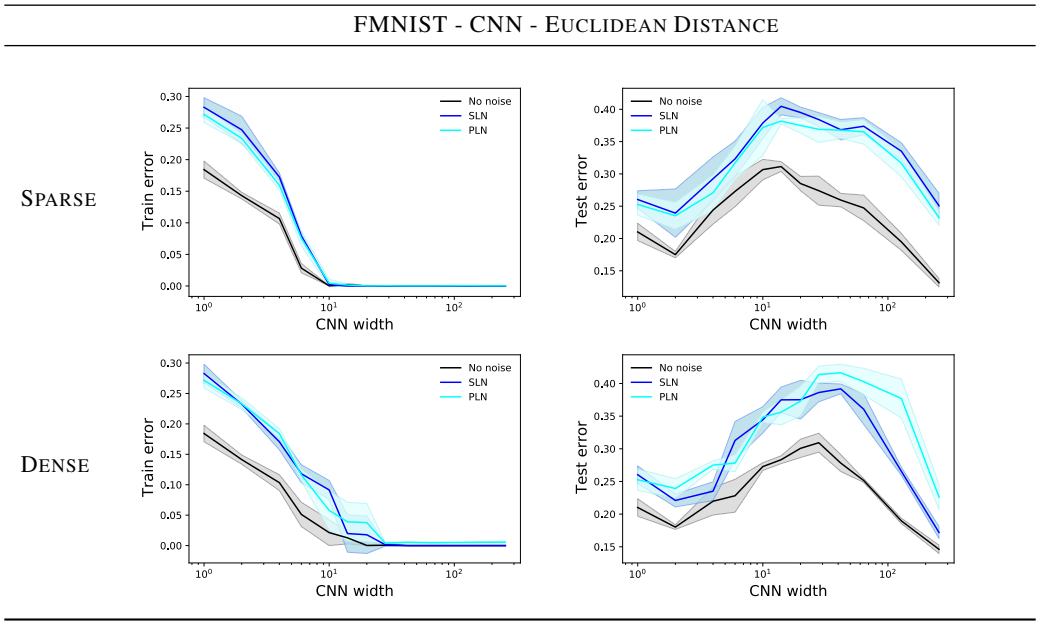


Figure 12: FMNIST training (left) and test error (right) for CNN with Euclidean distance configuration.

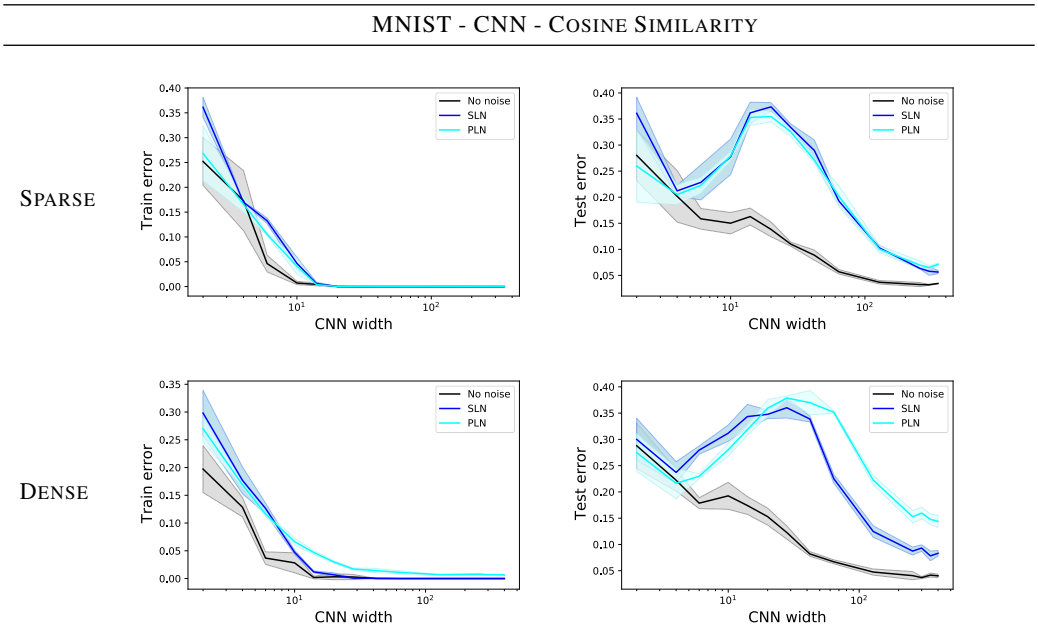


Figure 13: MNIST training (left) and test error (right) for CNN with cosine distance configuration.

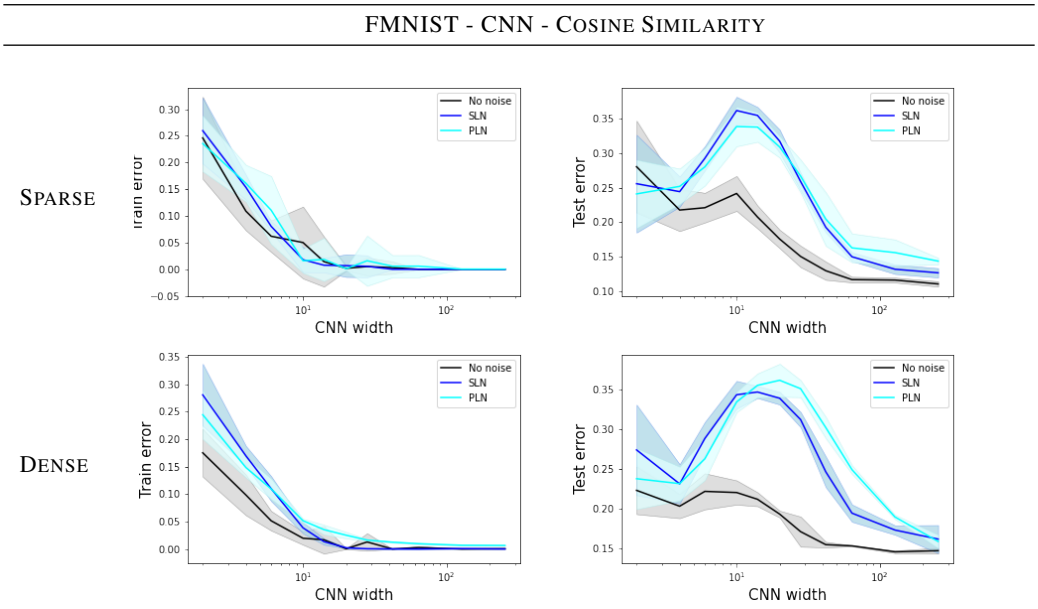


Figure 14: FMNIST training (left) and test error (right) for CNN with cosine distance configuration.

G.2 DOUBLE DESCENT AND COMPLEX DATASETS: THE CIFAR10 CASE

Compared to the case of MNIST and Fashion MNIST, the behavior of DD on CIFAR10 is more varied. First of all, we want to underline that we do not show any MLP results as this architecture is not suitable for the study of image datasets, especially the complex ones. The inadequacy of the neural network leads to poor performances that hide, if not even prevent the appearance of DD. This is precisely what we found in our experiments. We then focused on the CNN architecture, studying the effects of changing the loss function. Our results show that, using the contrastive loss with Euclidean distance (Fig. 15), the dataset pairs density drastically changes the behavior of test error at varying number of network parameters: for sparse connections the DD is absent, while in case of dense connections it is well recognizable. Instead, using the cosine loss with cosine similarity (Fig. 16), a first DD appears in both setups around k = 40 but for larger values of k the two behaviors differ considerably. In particular, in the case of sparse connections we see a wider and well recognizable second DD regardless of the presence of label noise.[7] As for the dense connections, on the other hand, the test error continues to drop in the absence of noise and with SLN, while with PLN the test error curve grows considerably up to values close to 50%. Regardless of the fact that the curve may eventually go back down at higher k values, the deviation of PLN from the other cases shows how the breaking of transitivity compromises effective network training. The large gap that we observe in this plot is representative of the fact that noise turns into corruption.

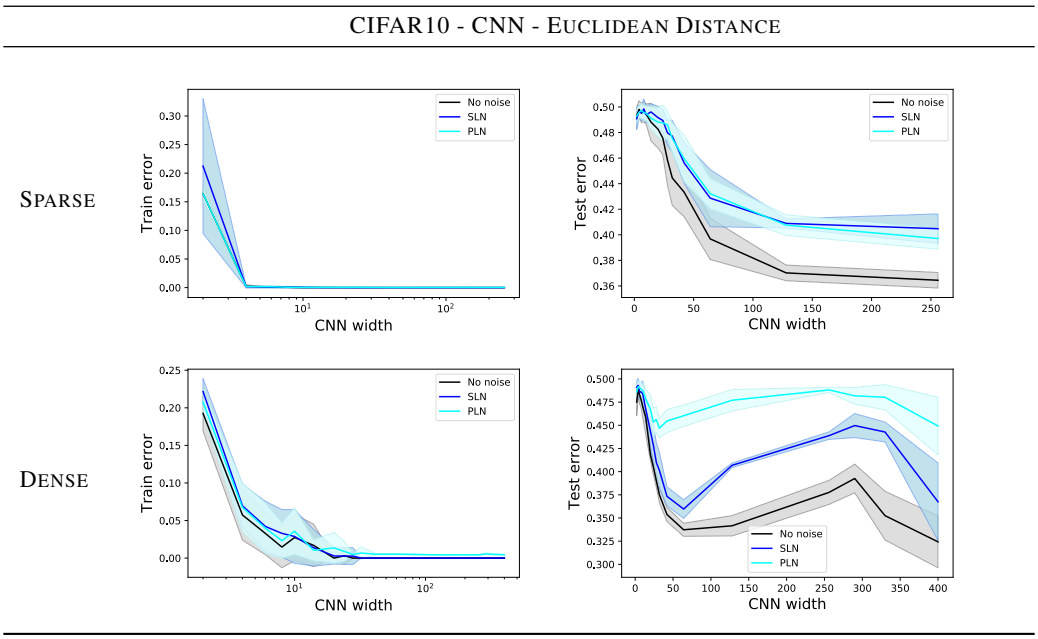


Figure 15: CIFAR10 training (left) and test error (right) for CNN with Euclidean distance configuration.

[7]The presence of multiple DD is not completely unexpected. In classification tasks it was shown to appear in d'Ascoli et al. (2020b); Adlam & Pennington (2020). Nevertheless, we cannot explain the real origin and the position of these peaks as it is not possible to treat Siamese networks using techniques that allow an analytical approach such as Random Fourier Features. We leave this analysis for future works.

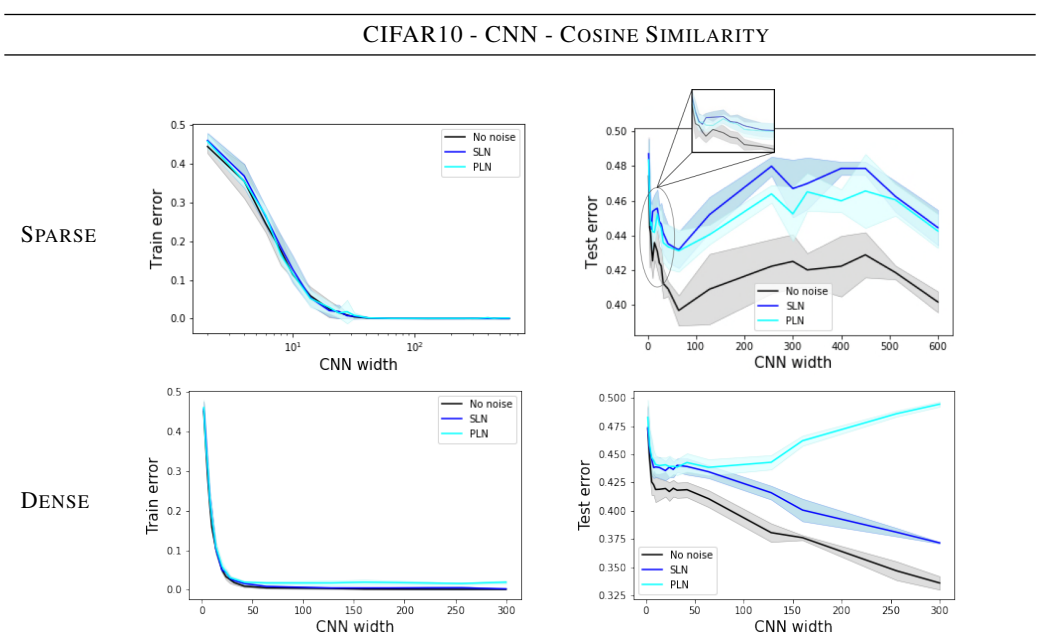


Figure 16: CIFAR10 training (left) and test error (right) for CNN with cosine distance configuration.

G.3 EPOCH WISE DOUBLE DESCENT

For completeness and to show that each point of the parameter-wise DD curve is obtained at model convergence, we show some results related to the epoch-wise training/test error and the training loss evolution.

MNIST - MLP - TIME EVOLUTION

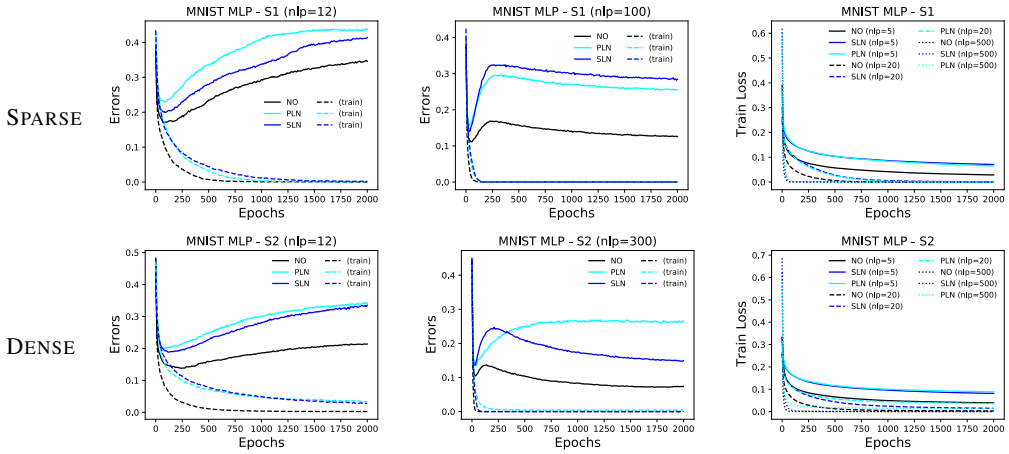


Figure 17: MNIST training (left) and test error (right) and training loss function evolution for MLP architecture.

FMNIST - MLP - TIME EVOLUTION

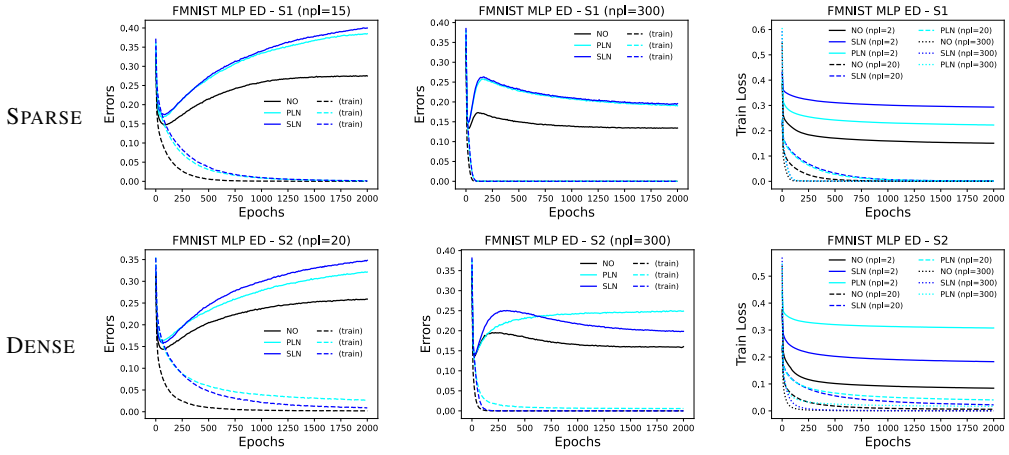


Figure 18: FMNIST training (left) and test error (right) and training loss function evolution for MLP architecture.

MNIST - CNN - COSINE SIMILARITY - TIME EVOLUTION

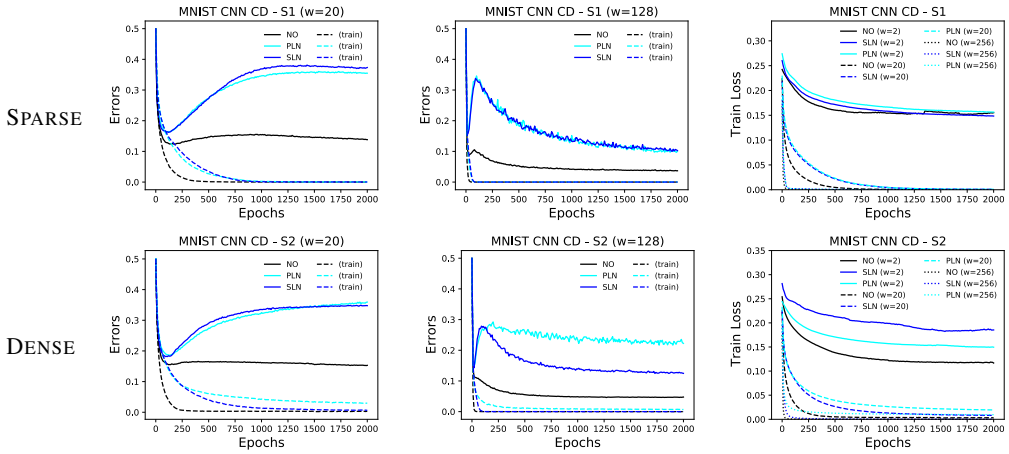


Figure 19: MNIST training (left) and test error (right) and training loss function evolution for CNN -cosine similarity architecture.

FMNIST - CNN - COSINE SIMILARITY - TIME EVOLUTION

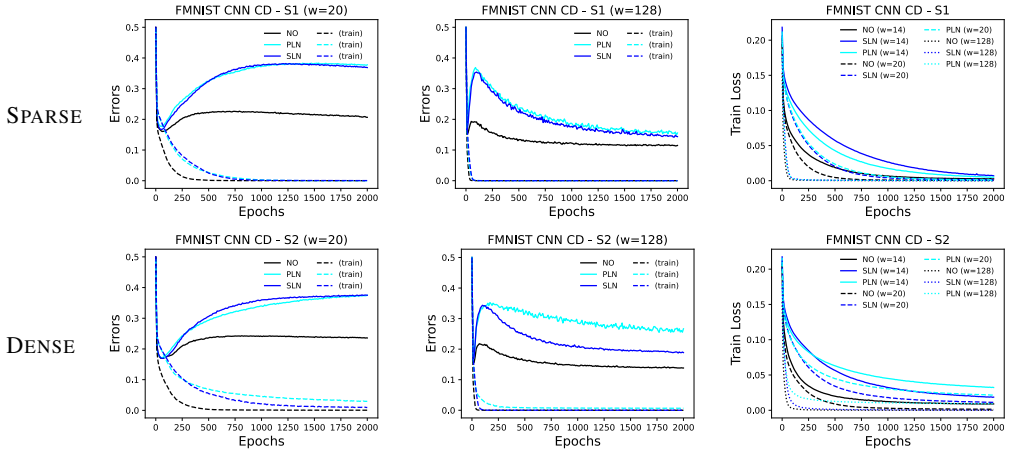


Figure 20: FMNIST training (left) and test error (right) and training loss function evolution for CNN -cosine similarity architecture.

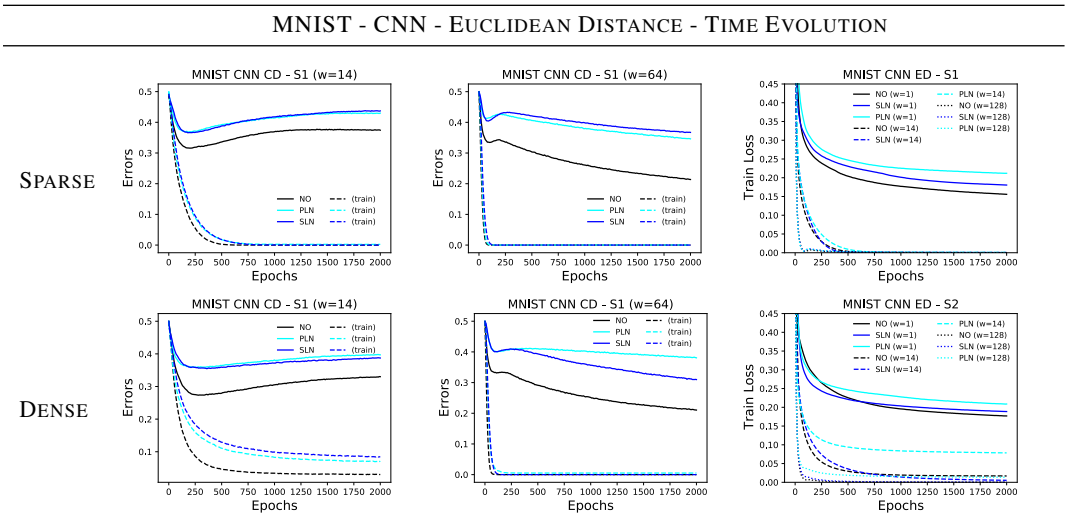


Figure 21: MNIST training (left) and test error (right) and training loss function evolution for CNN - Euclidean distance architecture.

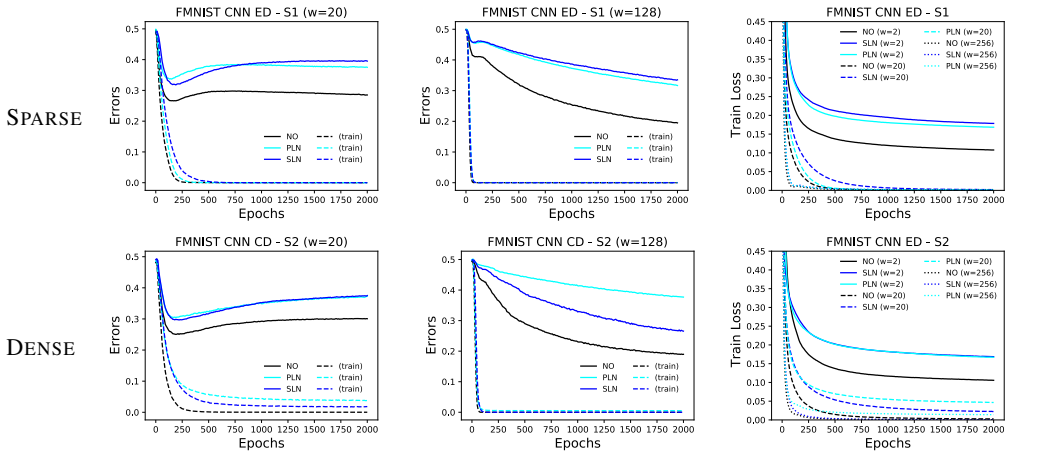


Figure 22: FMNIST training (left) and test error (right) and training loss function evolution for CNN - Euclidean distance architecture.

G.4 LEARNING RATE CHOICE MAY AFFECT DD

In this section we briefly point out that choosing different learning rate values may affect DD behaviour. We experimentally found that this effect is particularly prominent in the training setup where the Siamese CNN is trained using Euclidean distance and contrastive loss. In particular, choosing a larger learning rate, makes the DD more noisy. Detecting DD thus requires more runs and the final result may hide the shift between SLN and PLN curves in the dense pairs setup. In Figs. 23 and 24 we provide few examples where we considered lr$= 10^{-4}$ (20 iterations), as opposed to the previous section where all CNN were trained with lr$= 2 \cdot 10^{-5}$.

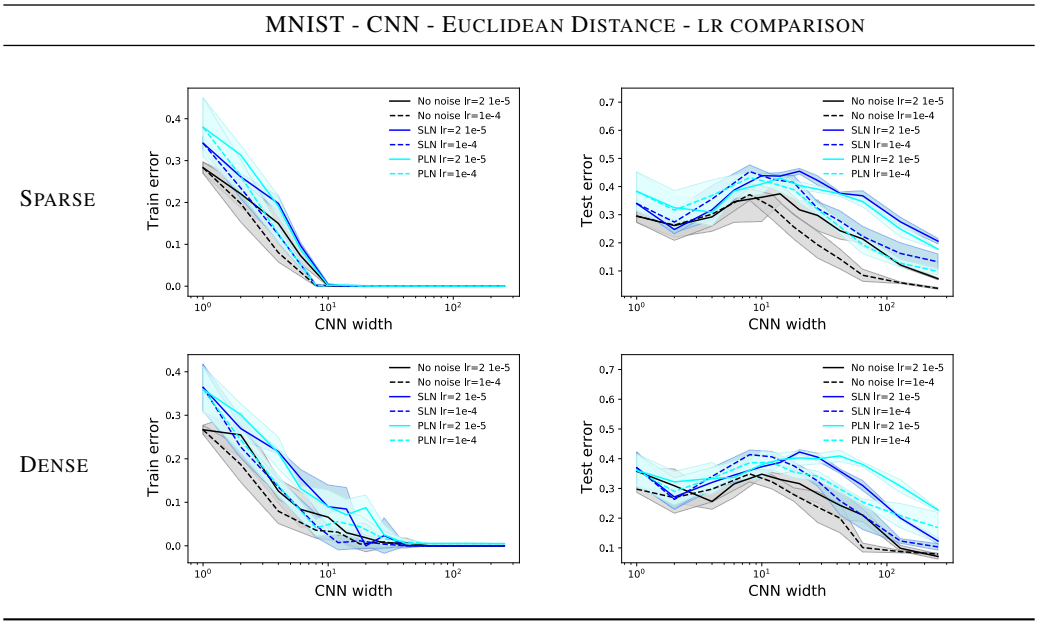


Figure 23: MNIST training (left) and test error (right) for CNN-Euclidean distance configuration.

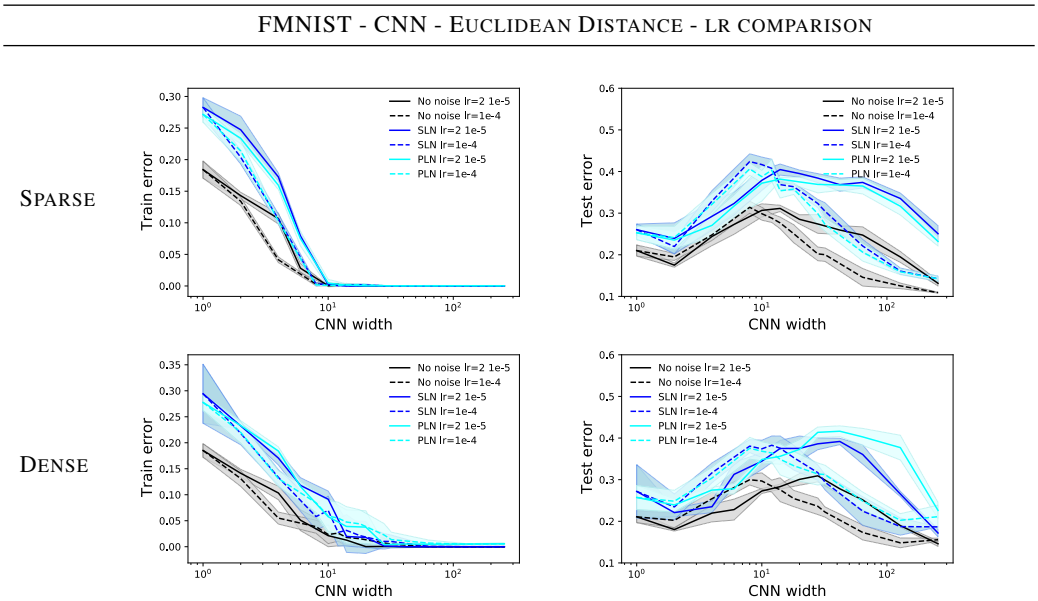


Figure 24: FMNIST training (left) and test error (right) for CNN with Euclidean distance configuration.

G.5 COMPARISON BETWEEN ONLINE AND OFFLINE SETTINGS

Here we present several comparisons between online (*Ideal World*) and offline (*Real World*) settings for different metrics. For all plots we use the EMNIST dataset as in Sec. 3. The experimental setup details are given in Sec. 2.1.

- **Ideal Worlds.** In Fig. 25, we compare Ideal world cases with different percentages of single label noise (SLN) and pair label noise (PLN) for the MLP-Euclidean Distance architecture trained with the contrastive loss. As expected, the test error gets worse as label noise increases, but the model still improves generalization with noisy data. Figure 26 compares Ideal world test errors for different learning rates for the CNN-Cosine-Similarity case trained with the cosine loss and data without noise.
- **10% Noise – MLP-Euclidean Distance (Contrastive Loss).** Figure 27 (Figure 28) shows the Real and Ideal Test Errors and Test Losses for the sparse (dense) scenario.[8] Note that the Test Losses follow the same behavior as their corresponding Test Errors. The PLN Ideal (gray) and PLN Real (cyan) test errors for the dense case (left panel of Fig. 28) are also shown in the left-hand side of Fig. 4 in the main text.
- **10% Noise – CNN-Cosine Similarity (Cosine Loss).** Likewise to the MLP case above, Figure 29 (Figure 30) shows the Real and Ideal Test Errors and Test Losses for the sparse (dense) scenario. The PLN Ideal (gray) and PLN Real (cyan) test errors for dense connections (left panel of Fig. 30) are also shown in the right-hand side of Fig. 4 in the main text.
- **No-Noise – MLP-CNN Comparisons.** In Fig. 31 we compare the MLP and CNN architectures. For both cases we calculate the Euclidean distance in the output layer and use the contrastive loss for training with learning rate $\lambda = 10^{-4}$. Top-left of figure 31 shows that the MLP Real test error for dense connections is slightly higher than the sparse one, and there is a constant gap between the two scenarios, which converges after about 2k iterations. Both curves present a small deviation from the Ideal World, which marginally increases as the online test error improves with fresh samples. On the other hand, there is a larger bootstrap gap between the CNN Real scenarios and the corresponding ideal case, see bottom-left of Fig. 31. This indicates that the Real test errors for the CNN-Euclidean-Distance architecture overfit earlier than the ones for the MLP-Euclidean-Distance case,[9] signalizing that a correspondence between online and offline settings is affected by the network setup. Nevertheless, in order to achieve the best performance, the right NN setup should be used (i.e., the natural architecture-loss function matching). Indeed, the architecture-loss matching choice is the reason why CNN performs worse than MLP in Fig. 31, where both cases are trained with the contrastive loss using the Euclidean distance between output branches. In fact, as shown in Fig. 32, CNN-Cosine-Similarity trained with the cosine embedding loss outperforms MLP-Euclidean-Distance trained with the contrastive loss as expected.

[8] Note that the test errors in Figs. 27 and 28 (and also in Figs. 29 and 30) should not be directly compared to the DD curves in Sec. 3. In particular, in Sec. 3, there is a clear difference between PLN and SLN for the scenario with dense connections. While in Sec. 3 the network is trained over 2k epochs, such that training continues after the training error has reached its asymptotic value; here we want to probe the Real World when its optimizer is still updating significantly. As pointed out in previous works (see, e.g., Nakkiran et al. (2020a)), setups with early stopping training usually do not exhibit DD. For the Real-world models in this section, the training always stops at 40 epochs.

[9] This can be avoided by stopping training earlier in the CNN case. However, since we want to compare the architectures in Fig. 31, we train both MLP and CNN scenarios using the same loss and learning rate for the same number of optimizer iterations. We stress, though, that Nakkiran et al. (2021) compare soft-errors (continuous error versions), instead of the hard-errors (discrete errors coming from using the margin/threshold) that we use here. In the classification examples discussed in Nakkiran et al. (2021), they found that the bootstrap gap is often smaller for soft-errors.

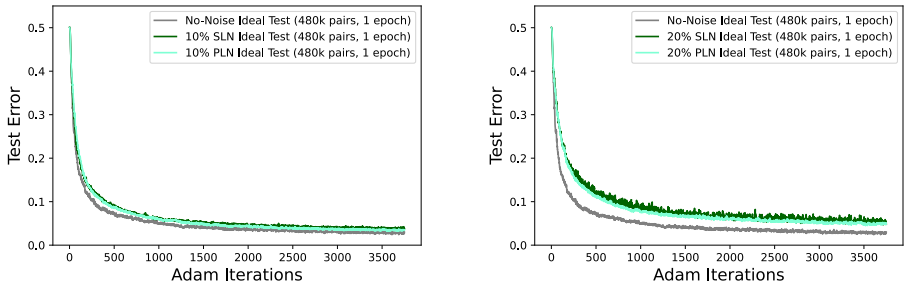


Figure 25: Comparison between Ideal worlds for different levels of noise. Plots show the Test Errors as a function of minibatch Adam iterations. We plot the median over 5 trials for the scenarios with SLN and PLN label noise (10% (left) and 20% (right)) for the MLP architecture with 200 nodes per layer trained with contrastive loss using learning rate $\lambda = 10^{-4}$.

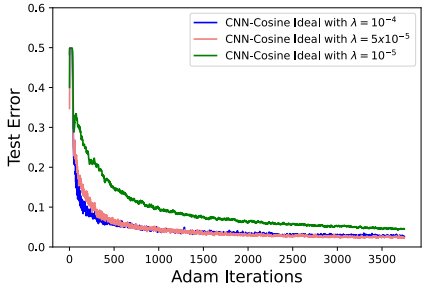


Figure 26: Comparison between Ideal worlds for different learning rates for data without noise. The plot shows the Test Errors as a function of minibatch Adam iterations for the CNN architecture trained once with the cosine loss using 480k pairs.

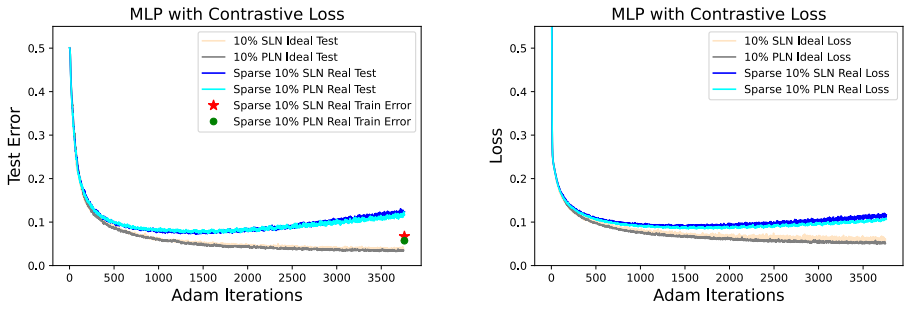


Figure 27: Ideal *vs.* Sparse Real worlds with 10% of label noise for the MLP architecture with 200 nodes per layer trained with learning rate $\lambda = 10^{-4}$. Plots show the Test Errors (left) and the Test Losses (right) as a function of minibatch Adam iterations. The star (dot) on the left panel corresponds to the SLN (PLN) Real world Train Error at the end of training.

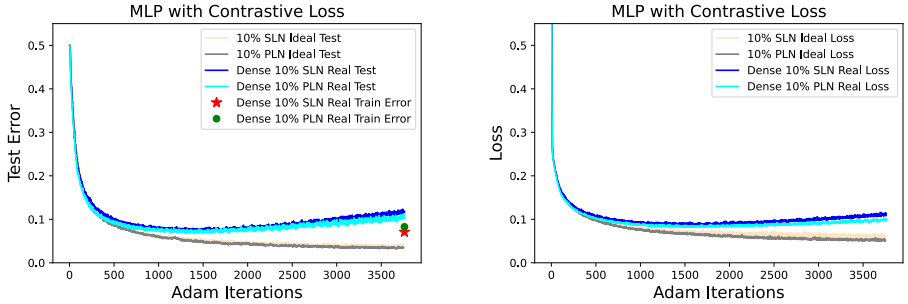


Figure 28: Ideal *vs.* Dense Real worlds with 10% of label noise for the MLP architecture with 200 nodes per layer trained with learning rate $\lambda = 10^{-4}$. Plots show the Test Errors (left) and the Test Losses (right) as a function of minibatch Adam iterations. The star (dot) on the left panel corresponds to the SLN (PLN) Real world Train Error at the end of training. PLN Ideal (gray) and PLN Real (cyan) Test Errors are also shown in the left panel of Fig. 4.

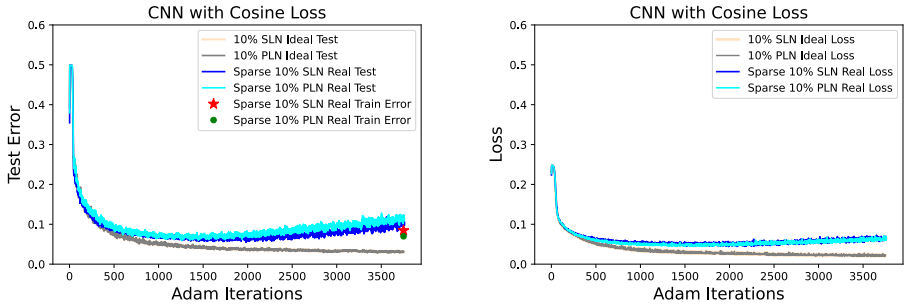


Figure 29: Ideal *vs.* Sparse Real worlds with 10% of label noise for the CNN architecture with $k = 47$ trained with cosine loss using learning rate $\lambda = 5 \times 10^{-5}$. Plots show the Test Errors (left) and the Test Losses (right) as a function of minibatch Adam iterations. The star (dot) on the left panel corresponds to the SLN (PLN) Real world Train Error at the end of training. Note that the Ideal world curves almost completely overlap to each other.

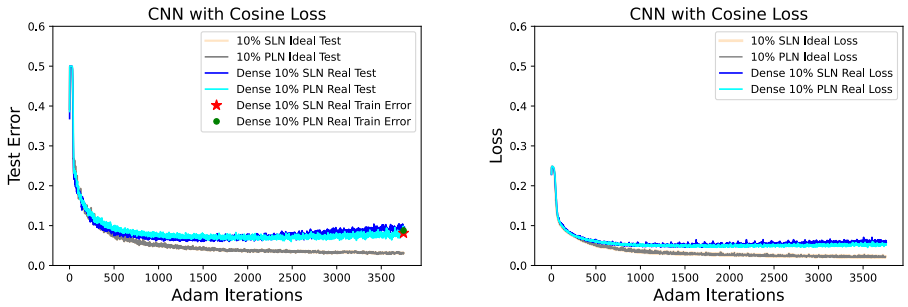


Figure 30: Ideal *vs.* Dense Real worlds with 10% of label noise for the CNN architecture with $k = 47$ trained with the cosine loss using learning rate $\lambda = 5 \times 10^{-5}$. Plots show the Test Errors (left) and the Test Losses (right) as a function of minibatch Adam iterations. The star (dot) on the left corresponds to the SLN (PLN) Real world Train Error at the end of training. Note again that the Ideal world curves overlap to each other. PLN Ideal (gray) and PLN Real (cyan) Test Errors are also shown in the right panel of Fig. 4.

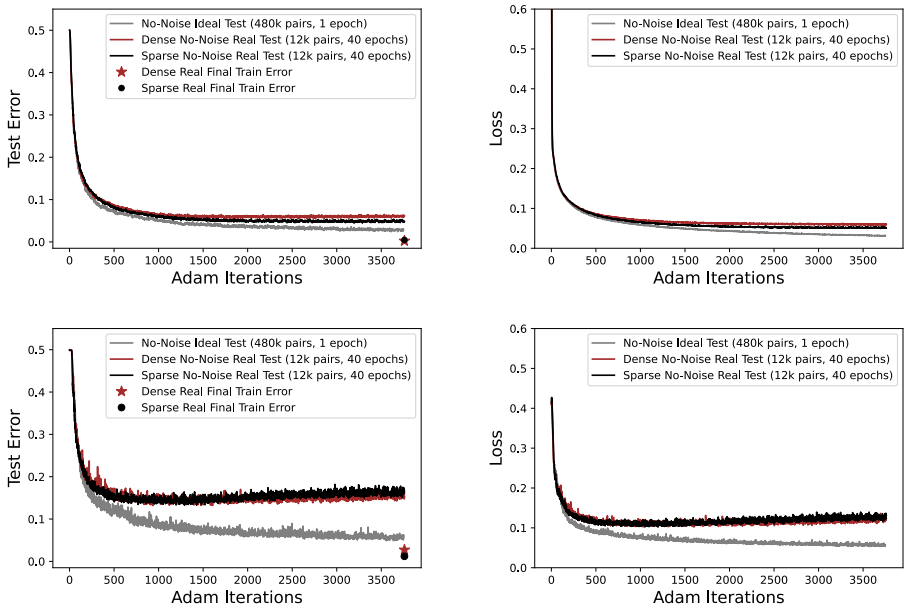


Figure 31: Ideal *vs.* Dense/Sparse Real worlds in the absence of label noise. Plots show Test Errors (left) and the Test Losses (right) as a function of minibatch Adam iterations for the MLP (top) and CNN (bottom) cases. The stars (dots) correspond to the Dense (Sparse) Real world train error at the end of training. For both architectures we calculate the Euclidean distance in the output layer and use the contrastive loss for training with learning rate $\lambda = 10^{-4}$. We use 200 nodes per layer for the MLP cases and width $k = 47$ for the CNN cases.

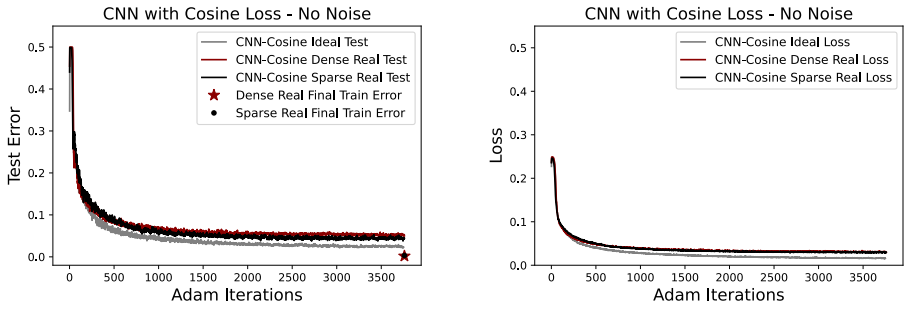


Figure 32: Ideal *vs.* Dense/Sparse Real worlds in the absence of label noise. Plots show the Test Errors (left) and the Test Losses (right) as a function of minibatch Adam iterations for the CNN-Cosine-Similarity (with $k = 47$) trained with the cosine loss using learning rate $\lambda = 5 \times 10^{-5}$. The star (dot) corresponds to the Dense (Sparse) Real World train error at the end of training. Note that the train errors (star/dot) overlap to each other.