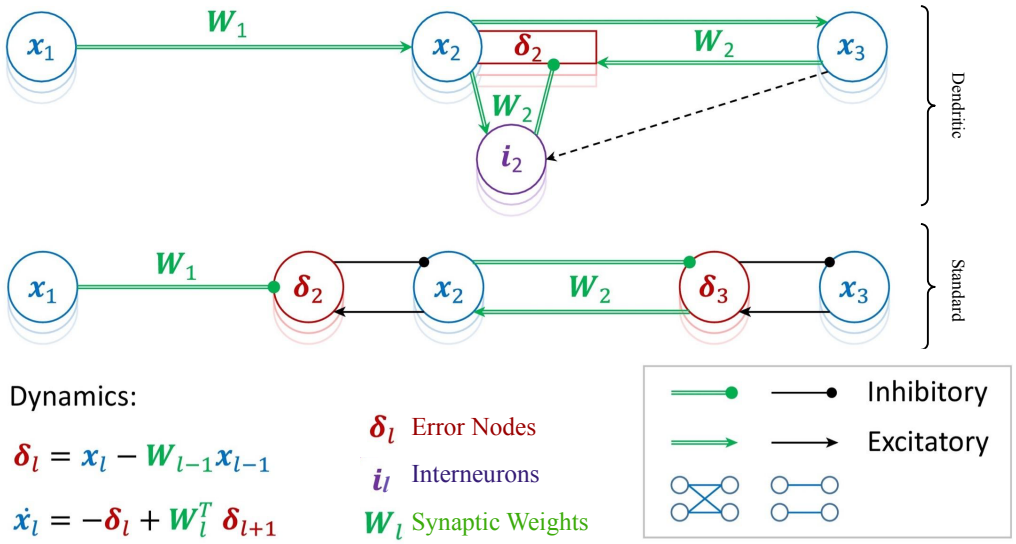

Figure 5: Standard and dendritic neural implementation of predictive coding. The dendritic implementation makes use of interneurons $i_l = W_l x_l$ (according to the notation used in the figure). Both implementations have the same equations for all the updates, and are thus equivalent; however, dendrites allow a neural implementation that does not take error nodes into account, improving the biological plausibility of the model. Figure taken and adapted from (Whittington & Bogacz, 2019).

A A Discussion on Biological Plausibility

In this section, we discuss the biological plausibility of the proposed algorithm, a topic overlooked in the main body of this paper. In the literature, there is often a disagreement on whether a specific algorithm is biologically plausible or not. Generally, it is assumed that an algorithm is biologically plausible when it satisfies a list of properties that are also satisfied in the brain. Different works consider different properties. In our case, we consider as list of minimal properties that include local computations and lack of a global control signals to trigger the operations. Normally, predictive coding networks take error nodes into account, often considered implausible from the biological perspective (Sacramento et al., 2018). Even so, the biological plausibility of our model is not affected by this: it is in fact possible to map PC on a different neural architecture, in which errors are encoded in apical dendrites rather than separate neurons (Sacramento et al., 2018; Whittington & Bogacz, 2019). Graphical representations of the differences between the two implementations can be found in Fig. 5, taken (and adapted) from (Whittington & Bogacz, 2019). Furthermore, our formulation is more plausible than the original formulation of PC, as it is able to learn without the need of external control signals that trigger the weight update.

B PSEUDOCODES OF Z-IL AND PC

Algorithm 2 Learning a dataset $\mathcal{D} = \{\bar{y}_i\}$ with PC.

Require: For every i, $\bar{x}_i^{(0)}$ is fixed to $\bar{y}_i$,
for $t = 0$ to T **do**
  For every i and l, update $\bar{x}^{(l)}$ to minimize F via Eq.(7)
  if t = T **then**
    For every l update each $\bar{\theta}^{(l)}$ to minimize F via Eq. (8)
  end if
end for

Algorithm 3 Learning one training pair $(\bar{s}^{\text{in}}, \bar{s}^{\text{out}})$ with Z-IL

Require: $\bar{x}_0^L$ is fixed to $\bar{s}^{\text{in}}$, $\bar{x}_0^0$ is fixed to $\bar{s}^{\text{out}}$.
Require: $\bar{x}^{(l)} = \bar{\mu}^{(l)}$ for $l \in \{1, ..., L-1\}$, and $t = 0$.
for $t = 0$ to T **do**
  for each level l **do**
    Update $\bar{x}^{(l)}$ to minimize F via Eq.(7)
    if $t = l$ **then**
      Update $\bar{\theta}^{(l)}$ to minimize F via Eq.(8).
    end if
  end for
end for

Table 3: Theoretical Efficiency of PC, Z-IL, BP, and iPC.

	One inference step	PC	Z-IL	BP	iPC
Number of MMs per weight update	$(2L-1)$	$(2L-1)T$	$(2L-1)(L-1)$	$(2L-1)$	$(2L-1)$
Number of SMMs per weight update	2	$2T$	$2(L-1)$	$(2L-1)$	2

C On the efficiency of PC, BP, and iPC

In this section, we discuss the time complexity and efficiency of PC, BP, Z-IL, and iPC. We now start with the first three, and introduce a metric that we use to compute such complexity. This metric is the number of *simultaneous matrix multiplications* (*SMMs*), i.e., the number of non-parallelizable matrix multiplications needed to perform a single weight update. It is a reasonable approximation of running time, as multiplications are by far the most complex operation ($\approx \mathcal{O}(N^3)$) performed by the algorithm.

C.1 Complexity of PC, BP, and Z-IL

Serial Complexity: To complete a single update of all weights, PC and Z-IL run for T and $(L-1)$ inference steps, respectively. To study the complexity of the inference steps we consider the number of *matrix multiplications* (*MMs*) required for each algorithm: One inference step requires $(2L-1)$ MMs: L for updating all the errors, and $(L-1)$ for updating all the value nodes (Eq. equation 6). Thus, to complete one weight update, PC and Z-IL require $(2L-1)T$ and $(2L-1)(L-1)$ MMs, respectively. Note also that BP requires $(2L-1)$ MMs to complete a single weight update: L for the forward, and $(L-1)$ for the backward pass. These numbers are summarized in the first row of Table 3. According to this measure, BP is the most efficient algorithm, Z-IL ranks second, and PC third, as in practice T is much larger than L. However, this measure only considers the total number of matrix multiplications needed, without considering whether some of them can be performed in parallel, which could significantly reduce the time complexity. We now address this problem.

Parallel complexity: The MMs performed during inference can be parallelized across layers. In fact, computations in Eq. equation 6 are layer-wise independent, thus L MMs that update all the error nodes take the time of only one MM if properly parallelized. Similarly, in Eq. equation 6, $(L-1)$ MMs that update all the value nodes take the time of only one MM if properly parallelized. As a result, one inference step only takes the time of 2 MMs if properly parallelized (since, as stated, it consists of updating all errors and values via Eq. equation 6). Thus, one inference step takes 2 SMMs; one weight update with PC and Z-IL takes $2T$ and $2(L-1)$ SMMs, respectively. Since no MM can be parallelized in BP (the forward pass in the network and the backward pass of error are both layer-dependent), before performing a single weight update, $(2L-1)$ SMMs are required. These numbers are summarized in the second row of Table 3. Overall, measured over SMMs, BP and Z-IL are equally efficient (up to a constant factor), and faster than PC.

C.2 Complexity of iPC

To complete one weight update, iPC requires one inference step, thus $(2L-1)$ MMs or 2 SMMs, as also demonstrated in the last column of Table 3. Compared to BP, iPC takes around L times less SMMs per weight update, and should hence be significantly faster in deep networks. Intuitively, this is because matrix multiplications in BP have to be done sequentially along layers, while the ones in iPC can all be done in parallel across layers (Fig. 6). More formally, we have the following theorem, which holds when performing full-batch training:

Theorem 1. *Let M and M' be two equivalent networks with L layers trained on the same dataset. Let M (resp., M') be trained using BP (resp., iPC). Then, the time complexity measured by SMMs needed to perform one full update of the weights is $\mathcal{O}(1)$ and $\mathcal{O}(L)$ for iPC and BP, respectively.*

Proof. Consider training on an MLP with L layers, and update weights for multiple times on a single datapoint. Generalizations to multiple datapoints and multiple mini-batches are similar and will be provided after. We first write the equations needed to be computed for iPC to produce one weight

update:

$$
\begin{aligned}
x_{i,t}^{(L)} &= s_i^{in} \text{ and } x_{i,t}^{(0)} = s_i^{out} \\
\hat{x}_{i,t}^{(l)} &= \sum_{j=1}^{n^{l-1}} \theta_{i,j}^{(l+1)} f(x_{j,t}^{(l+1)}) && \text{for } l \in \{1, \dots, L\} \\
\varepsilon_{i,t}^{(l)} &= x_{i,t}^{(l)} - \hat{x}_{i,t}^{(l)} && \text{for } l \in \{1, \dots, L\}
\end{aligned} \tag{9}
$$

$$
x_{i,t+1}^{(l)} = x_{i,t}^{(l)} + \gamma \cdot \left(-\varepsilon_{i,t}^{(l)} + x_{i,t}^{(l)} \sum_{k=1}^{n^{(l+1)}} \varepsilon_{k,t}^{(l+1)} \theta_{k,i}^{(l)} \right) \quad \text{for } l \in \{1, \dots, L\} \tag{10}
$$

$$
\theta_{i,j,t+1}^{(l)} = \theta_{i,j,t}^{(l)} - \alpha \cdot \varepsilon_{i,t}^{(l+1)} f(x_{j,t}^{(l)}) \quad \text{for } l \in \{1, \dots, L\}. \tag{11}
$$

We then write the three equations needed to be computed for BP to produce one weight update:

$$
\begin{aligned}
x_{i,t}^{0} &= s_i^{in} \\
x_{i,t}^{(l)} &= \sum_{j=1}^{n^{l-1}} \theta_{i,j}^{(l+1)} f(x_{j,t}^{(l+1)}) \text{ for } l \in \{1, \dots, L\} \\
\varepsilon_{i,t}^{(L)} &= s_i^{out} - x_{i,t}^{(L)}
\end{aligned} \tag{12}
$$

$$
\varepsilon_{i,t}^{(l)} = f'\left(x_{i,t}^{(l)}\right) \sum_{k=1}^{n^{(l+1)}} \varepsilon_{k,t}^{(l+1)} \theta_{k,i}^{(l)} \text{ for } l \in \{L, \dots, 1\} \tag{13}
$$

$$
\theta_{i,j,t+1}^{(l)} = \theta_{i,j,t}^{(l)} - \alpha \cdot \varepsilon_{i,t}^{(l+1)} f(x_{j,t}^{(l)}) \text{ for } l \in \{1, \dots, L\}.
$$

First, we notice that the matrix multiplication (MM) is the most complex operation. Specifically, for two adjacent layers with the size of n^l and n^l, the complexity of MM is $\mathcal{O}(n^l n^l)$, but the maximal complexity of the other operations is $\mathcal{O}(\max n^l, n^l)$. In the above equations, only equations with MM are numbered, which are the equations that we investigate in our complexity analysis.

Eq. equation 9 for iPC takes L MMs, but one SMM, since the the for-loop for $l \in \{1, \dots, L\}$ can run in parallel for different l. This is further because the variables on the right side of Eq. equation 9 are immediately available. Differently, Eq. equation 12 for iPC takes L MMs, and also L SMMs, since the for-loop for $l \in \{1, \dots, L\}$ has to be executed one after another, following the specified order $\{2, \dots, L\}$. This is further because the qualities on the right side of Eq. equation 12 are immediately available, but require to solve Eq. equation 12 again for another layer. That is, to get $x_{i,t}^{(L)}$, Eq. equation 12 has to be solved recursively from $l = 1$ to $l = L$.

Similar sense applies to the comparison between Eqs. equation 10 and equation 13. Eq. equation 10 for iPC takes $L - 1$ MMs but 1 SMMs; Eq. equation 13 for BP takes $L - 1$ MMs and also $L - 1$ SMMs.

Overall, Eqs. equation 9 and equation 10 for iPC take $2L - 1$ MMs but 2 SMMs; Eqs. equation 12 and equation 13 for BP take $2L - 1$ MMs and also $2L - 1$ SMMs. Then, the time complexity measured by SMMs needed to perform one full update of the weights is $\mathcal{O}(1)$ and $\mathcal{O}(L)$ for iPC and BP, respectively.

C.3 Efficiency on One Data Point

To make the difference more visible and provide more insights, we explain this in detail with a sketch of this process on a small network in Fig. 6, where the horizontal axis of m is the time step measured by simultaneous matrix multiplications (SMMs), i.e., within a single m, one can perform one matrix multiplication or multiple ones in parallel; if two matrix multiplications have to be executed in order (e.g., the second needs results from the first), they will need to be put into

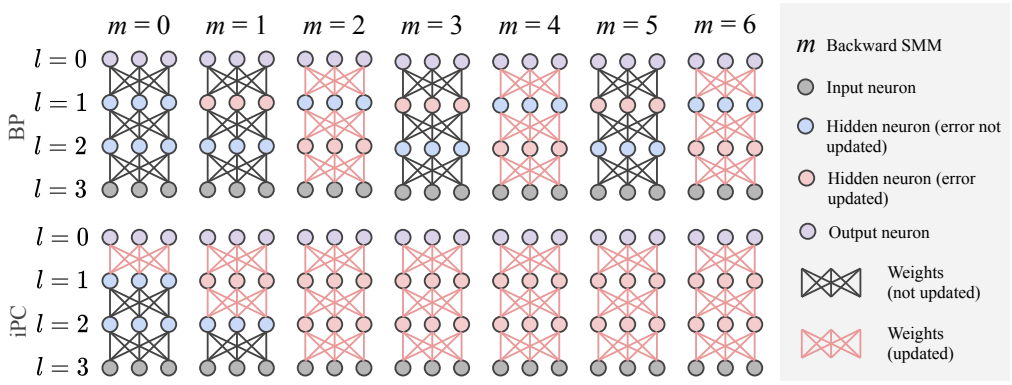

Figure 6: Graphical PClustration of the efficiency over backward SMMs of BP and iPC on a 3-layer network. iPC never clears the error (red neurons), while BP clears it after every update. This allows iPC to perform 5 full and 2 partial updates of the weights in the first 6 SMMs. In the same time frame, BP only performs 3 full updates. Note that the SMMs of forward passes are excluded for simplicity, w.l.o.g., as the insight from this example generalizes to the SMMs of the forward pass.

two steps of m. Note that we only consider the matrix multiplications for the backward pass, i.e., the matrix multiplications that backpropagate the error of a layer from an adjacent layer for BP and the inference of Eq. equation 6 for iPC, thus the horizontal axis m is strictly speaking "Backward SMM". The insight for the forward pass is similar as that of the backward pass. As it has been said, for BP, backpropagating the error from one layer to an adjacent layer requires one matrix multiplication; for iPC, one step of inference on one layer via Eq. equation 6 requires one matrix multiplication. BP and iPC are presented in the first and second rows, respectively. Before both methods are able to update weights in all layers, they need two matrix multiplications for spreading the error through the network, i.e., a weights update of all layers occurs for the first time at $m = 2$ for both methods. After $m = 2$, BP cleared all errors on all neurons, so at $m = 3$, BP backpropagates the error from $l = 0$ to $l = 1$, and at $m = 4$, BP backpropagates the error from $l = 1$ to $l = 2$ after which it can make an update of weights at all layers again for the second time. Note that the matrix multiplication that backpropagates errors from $l = 1$ to $l = 2$ at $m = 4$ cannot be put at $m = 3$, as it requires the results of the matrix multiplication at $m = 3$, i.e., it requires the error to be backpropagated to $l = 1$ from $l = 0$ at $m = 3$. However, this is different for iPC. After $m = 2$, iPC does not reset $x^l_{i,t}$ to $\mu^l_{i,t}$, i.e., the error signals are still held in $\varepsilon^l_{i,t}$. At $m = 3$, iPC performs two matrix multiplications in parallel, corresponding to two inferences steps at two layers, $l = 1$ and $l = 2$, updating $x^l_{i,t}$, and hence the error signals are held in $\varepsilon^l_{i,t}$ of these two layers. Note that the above two matrix multiplications of two inference steps can run in parallel and be put into a single m, as inference requires only locally and immediately available information. In this way, a weight update in iPC is able to be performed at every m ever since the very first few steps of m.

D Training Details

We now list some additional details to reproduce our results.

D.1 Experiments of Efficiency

The experiments for the efficiency of generative models were run on fully connected networks with 128, 256 or 512 hidden neurons, and $L \in \{4, 5\}$. Every network was trained on CIFAR10 or Tiny Imagenet with learning rates $\alpha = 0.00005$ and $\gamma = 0.5$, and $T \in \{8, 12, 16\}$. The experiments on discriminative models are performed using networks with 64 hidden neurons, depth $L \in \{3, 4, 6\}$, and learning rates $\alpha = 0.0001$ and $\gamma = 0.5$. The networks trained with BP have the same learning rate α. All the plots for every combination of hyperparameters can be found in Figures 8 and 7.

D.2 EXPERIMENTS OF GENERALIZATION QUALITY

As already stated in the paper body, to make sure that our results are not the consequence of a specific choice of hyperparameters, we performed a comprehensive grid search on hyperparameters, and reported the highest accuracy obtained, and the search is further made robust by averaging over 5 seeds. Particularly, we tested over 8 learning rates (from 0.000001 to 0.01), 4 values of weight decay $(0.0001, 0.001, 0.01, 0.1)$, and 3 values of the integration step γ $(0.1, 0.5, 1.0)$. We additionally verified that the optimized value of each hyperparameter lies within the searched range of that hyperparameter. As for additional details, we used standard Pytorch initialization for the parameters. For the hardware, we used a single Nvidia GeForce RTX 2080 GPU on an internal cluster. Despite the large search, most of of the best results were obtained using the following hyperparameters: $\gamma = 0.5$ ($\gamma = 1$ for Alexnet), $\alpha = 0.00005$.

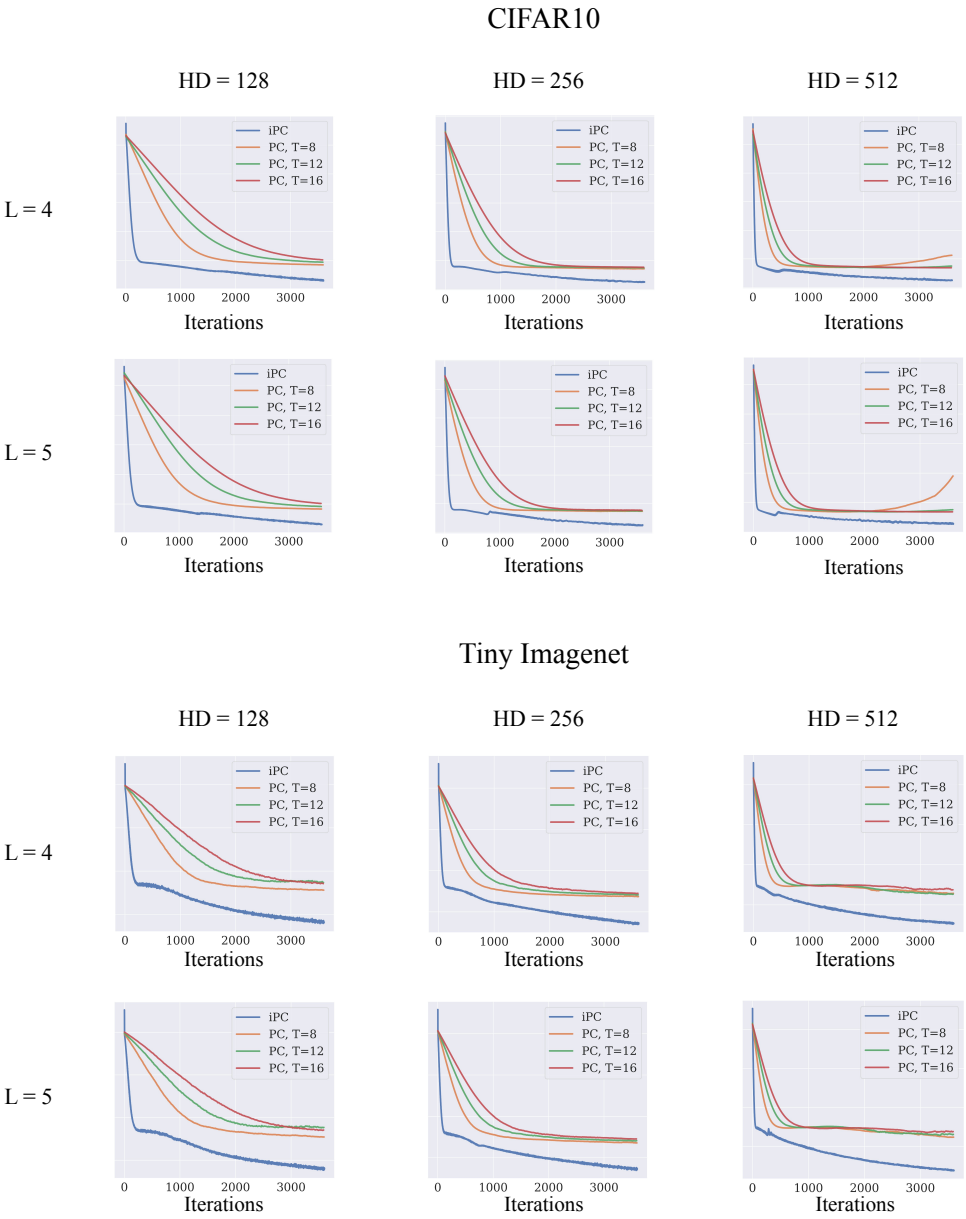

Figure 7: Efficiency of multiple generative networks trained with PC.

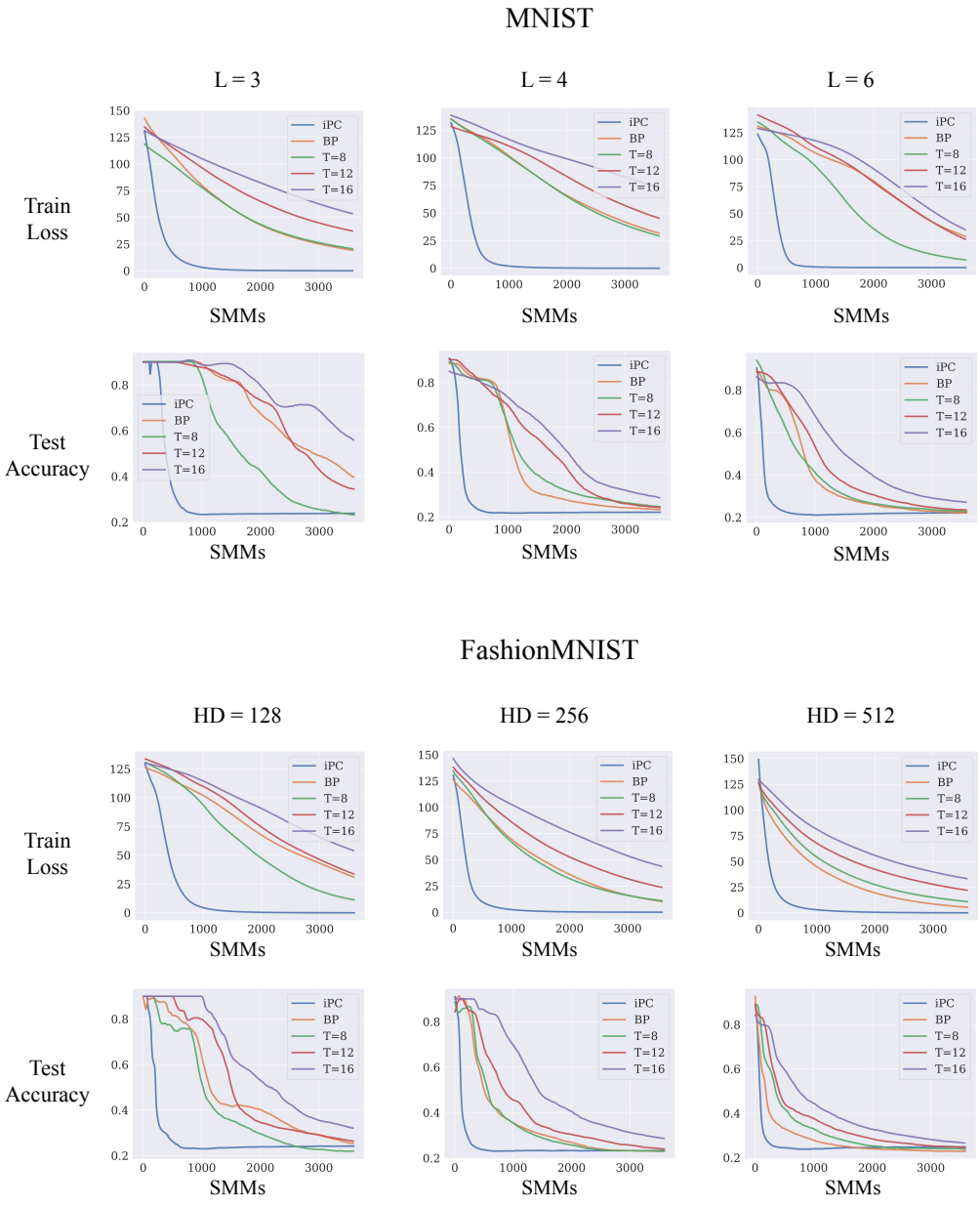

Figure 8: Efficiency of multiple discriminative networks trained with PC and BP.