Predify: Augmenting deep neural networks with brain-inspired predictive coding dynamics

Bhavin Choksi*
CerCo CNRS, UMR 5549 &
Université de Toulouse
bhavin.choksi@cnrs.fr

Milad Mozafari*
CerCo CNRS, UMR 5549 &
IRIT CNRS, UMR 5505
milad.mozafari@cnrs.fr

Callum Biggs O'May
CerCo CNRS
UMR 5549

Benjamin Ador
CerCo CNRS
UMR 5549

Andrea Alamia
CerCo CNRS
UMR 5549

Rufin VanRullen
CerCo CNRS, UMR 5549 &
ANITI, Université de Toulouse
rufin.vanrullen@cnrs.fr

Abstract

Deep neural networks excel at image classification, but their performance is far less robust to input perturbations than human perception. In this work we explore whether this shortcoming may be partly addressed by incorporating brain-inspired recurrent dynamics in deep convolutional networks. We take inspiration from a popular framework in neuroscience: "predictive coding". At each layer of the hierarchical model, generative feedback "predicts" (i.e., reconstructs) the pattern of activity in the previous layer. The reconstruction errors are used to iteratively update the network's representations across timesteps, and to optimize the network's feedback weights over the natural image dataset–a form of unsupervised training. We show that implementing this strategy into two popular networks, VGG16 and EfficientNetB0, improves their robustness against various corruptions and adversarial attacks. We hypothesize that other feedforward networks could similarly benefit from the proposed framework. To promote research in this direction, we provide an open-sourced PyTorch-based package called *Predify*, which can be used to implement and investigate the impacts of the predictive coding dynamics in any convolutional neural network.

1 Introduction

Deep convolutional neural networks (DCNNs), initially inspired by the primate visual cortex architecture, have taken big strides in solving computer vision tasks in the last decade. State-of-the-art networks can learn to classify images with high accuracy from huge labeled datasets [1, 2, 3, 4, 5, 6]. This rapid progress and the resulting interest in these techniques have also highlighted their various shortcomings. Most widely studied is the sensitivity of neural networks, not only to perturbations specifically designed to fool them (so-called "adversarial examples") but also to regular noises typically observed in natural scenes [7, 8, 9]. These shortcomings indicate that there is still room for improvement in current techniques.

One possible way to improve the robustness of artificial neural networks could be to take further inspiration from the brain. In particular, one major aspect of the cerebral cortex that is missing from standard feedforward DCNNs is the presence of feedback connections. Recent studies have stressed the importance of feedback connections in the brain [10, 11], and have shown how artificial neural

*Equal Contribution

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

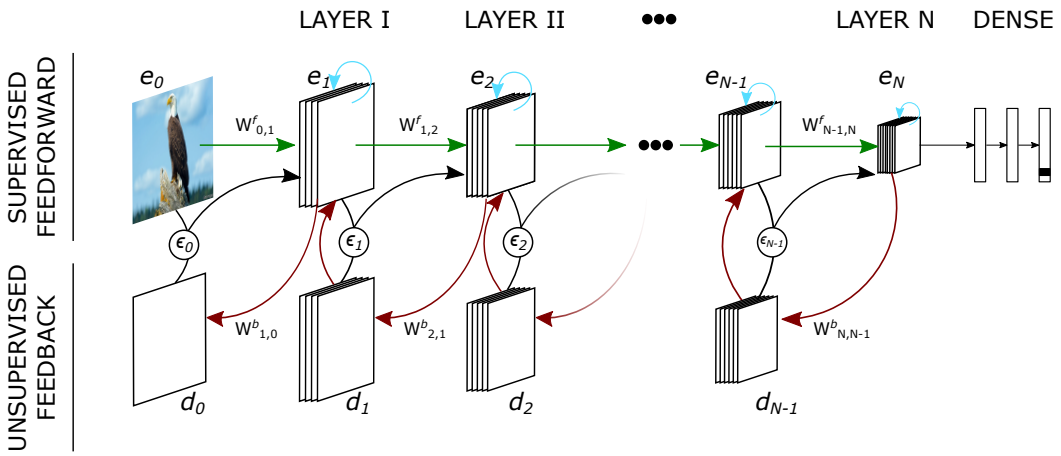


Figure 1: **General overview of our predictive coding strategy** as implemented in a feedforward hierarchical network with generative feedback connections. The architecture (roughly similar to stacked auto-encoders) consists of N encoding layers e_n and N decoding layers d_n. $W_{m,n}$ denotes the connection weights from layer m to layer n, with W^f and W^b for feedforward and feedback connections, respectively. The reconstruction errors at each layer are denoted by ϵ_n. The feedforward connections (green arrows) are trained for image classification (in a supervised fashion), while the feedback weights (red arrows) are optimized for a prediction (i.e. reconstruction) objective (unsupervised). Predictive coding minimizes the reconstruction errors in each layer by updating activations in the next layer accordingly (black arrows). Self-connections (memory) are represented by blue arrows.

networks can take advantage of such feedback for various tasks such as object recognition with occlusion [12], or panoptic segmentation [13]. Feedback connections convey contextual information about the state of the higher layers down to the lower layers of the hierarchy; in this way, they can constrain lower layers to represent inputs in meaningful ways. In theory, this could make neural representations more robust to image degradation [14]. Merely including feedback in the pattern of connections, however, may not always be sufficient; rather, it should be combined with proper mechanistic principles.

To that end, we explore the potential of recurrent dynamics for augmenting deep neural networks with brain-inspired predictive coding (supported by ample neuroscience evidence [15, 16, 17, 18, 19]). We build large-scale hierarchical networks with both feedforward and feedback connections that can be trained using error backpropagation. Several prior studies have explored this interesting avenue of research [20, 21, 22, 23], but with important differences with our approach (see Section 3). We demonstrate that our proposed method adds desirable properties to feedforward DCNNs, especially when viewed from the perspective of robustness.

Our contributions can be summarized as follows:

- We propose a novel strategy for effectively incorporating recurrent feedback connections based on the neuroscientific principle of predictive coding.
- We implement this strategy in two pre-trained feedforward architectures with unsupervised training of the feedback weights, and show that this improves their robustness against different types of natural and adversarial noise.
- We suggest and verify that an emergent property of the network is to iteratively shift noisy representations towards the corresponding clean representations—a form of "projection towards the learned manifold" as implemented in certain adversarial defense methods.
- To facilitate research aimed at using such neuroscientific principles in machine learning, we provide a Python package called *Predify* that can easily implement the proposed predictive coding dynamics in any convolutional neural network with a few lines of code.

2 Our Approach

2.1 The proposed predictive coding dynamics

Predictive coding, as introduced by [24], is a neurocomputational theory positing that the brain maintains an internal model of the world, which it uses to actively predict the observed stimulus. Within a hierarchical architecture, each higher layer attempts to predict the activity of the layer immediately below, and the errors made in this prediction are then utilized to correct the higher-layer activity.

To establish our notation, let us consider a hierarchical feedforward network equipped with generative feedback connections, as represented in Figure 1. The network contains N encoding layers e_n ($n \in \mathbb{N}$) and N corresponding decoding layers d_{n-1}. The feedforward weights connecting layer $n-1$ to layer n are denoted by $W^f_{n-1,n}$, and the feedback weights from layer $n+1$ to n by $W^b_{n+1,n}$. For a given input image, we first initiate the activations of all encoding layers with a feedforward pass. Then, over successive recurrent iterations (referred to as timesteps t), both the decoding and encoding layer representations are updated using the following equations (also refer to Pseudocode 1):

$$\boldsymbol{d}_n(t) = W^b_{n+1,n}\boldsymbol{e}_{n+1}(t) \tag{1}$$

$$\boldsymbol{e}_n(t+1) = \beta_n W^f_{n-1,n}\boldsymbol{e}_{n-1}(t+1) + \lambda_n \boldsymbol{d}_n(t) + (1-\beta_n-\lambda_n)\boldsymbol{e}_n(t) - \alpha_n \nabla\epsilon_{n-1}(t), \tag{2}$$

where β_n, λ_n ($0 \leq \beta_n + \lambda_n \leq 1$), and α_n act as layer-dependent balancing coefficients for the feedforward, feedback, and error-correction terms, respectively. $\epsilon_{n-1}(t)$ denotes the reconstruction error at layer $n-1$ and is defined as the mean squared error (MSE) between the representation $e_{n-1}(t)$ and the predicted reconstruction $d_{n-1}(t)$ at that particular timestep. Layer e_0 is defined as the input image and remains constant over timesteps. All the weights $W^f_{n-1,n}$ and $W^b_{n+1,n}$ are fixed during these iterations.

Each of the four terms in Equation 2 contributes different signals, reflected by different arrow colors in Figure 1: (i) the feedforward term (green arrows; controlled by parameter β) provides information about the (constant) input and changing representations in the lower layers, (ii) the feedback correction term (red arrows; parameter λ), as proposed in [24, 25], guides activations towards their representations from the higher levels, thereby reducing the reconstruction errors over time, (iii) the memory term (blue arrows) acts as a time constant to retain the current representation over successive timesteps, and (iv) the feedforward error correction term (black arrows; controlled by parameter α) corrects representations in each layer such that their generative feedback can better match the preceding layer. For this error correction term, we directly use the error gradient $\nabla\epsilon_{n-1} = [\frac{\partial\epsilon_{n-1}}{\partial e^0_n}, ..., \frac{\partial\epsilon_{n-1}}{\partial e^k_n}]$ to take full advantage of modern machine learning capabilities (where k is the number of elements in e_n). While the direct computation of this error gradient is biologically implausible, it has been noted before that it is mathematically equivalent to propagating error residuals

Pseudocode 1 Predictive Coding Iterations

```
Input image: e_0
for n = 1 to N do
    e_n ← Conv(e_{n-1})
    d_{n-1} ← deConv(e_n)
    ε_{n-1} ← ||d_{n-1} − e_{n-1}||_2^2
end for
for t = 1 to T do
    for n = 1 to N do
        ff ← β_n · Conv(e_{n-1})
        fb ← 0
        if n < N then
            fb ← λ_n · d_n
        end if
        e_n ← ff + fb + (1 − β_n − λ_n) · e_n − α_n · ∇ε_{n-1}
        d_{n-1} ← deConv(e_n)
        ε_{n-1} ← ||d_{n-1} − e_{n-1}||_2^2
    end for
end for
```

up through the (transposed) feedback connection weights $(W^b)^T$, as often done in other predictive coding implementations [22, 24]. Together, the feedforward and feedback error correction terms fulfill the objective of predictive coding as laid out by Rao and Ballard [24]. We discuss the similarities and differences between our equations and those proposed in the original Rao and Ballard implementation in the Appendix A.6.

While it is certainly possible to train such an architecture in an end-to-end fashion, by combining a classification objective for the feedforward weights W^f with an unsupervised predictive coding objective (see Section 2.2) for the feedback weights W^b, we believe that the benefits of our proposed scheme are best demonstrated by focusing on the added value of the feedback pathway onto a pre-existing state-of-the-art feedforward network. Consequently, we implement the proposed strategy with two existing feedforward DCNN architectures as backbones: VGG16 and EfficientNetB0, both trained on ImageNet. We show that predictive coding confers higher robustness to these networks.

2.2 Model architectures and training

We select VGG16 and EfficientNetB0, two different pre-trained feedforward networks on ImageNet, and augment them with the proposed predictive coding dynamics. The resulting models are called PVGG16 and PEfficientNetB0, respectively. The networks' "bodies" (without the classification head) are split into a cascade of N sub-modules, where each plays the role of an e_n in equation (2). We then add deconvolutions as feedback layers d_{n-1} connecting each e_n to e_{n-1}, with kernel sizes accounting for the increased receptive fields of the neurons in e_n or upsampling layers to match the size of the predictions and their targets (see Appendix A.2). We then train the parameters of the feedback deconvolution layers with an unsupervised reconstruction objective (with all feedforward parameters frozen). We minimize the reconstruction errors just after the first forward pass, and after a single deconvolution step (i.e. no error correction or predictive coding recurrent dynamics are involved at this stage):

$$\mathcal{L} = \sum_{n=0}^{N-1} \parallel e_n - d_n \parallel_2^2, \tag{3}$$

where e_n is the output of the nth encoder after the first forward pass and d_n is the estimated reconstruction of e_n via feedback/deconvolution (from e_{n+1}).

For both the networks, after training the feedback deconvolution layers, we freeze all of the weights, and set the values of hyperparameters to $\beta_n = 0.8$, $\lambda_n = 0.1$, and $\alpha_n = 0.01$ for all the encoders/decoders in Equation (2). We also explore various strategies for further tuning hyperparameters to improve the results (see Appendix A.7 for the chosen hyperparameter values).

2.3 *Predify*

To facilitate and automate the process of adding the proposed predictive coding dynamics to existing deep neural networks, we have developed an open-source Python package called *Predify*. The package is developed based on PyTorch [26] and provides a flexible object oriented framework to convert any PyTorch-compatible network into a predictive network. While an advanced user may find it easy to integrate *Predify* in their project manually, a simple text-based user interface (in TOML[2] format) is also provided to automate the steps. For the sake of improved performance and flexibility, *Predify* generates the code of the predictive network rather than the Python object. Given the original network and a configuration file (e.g. `'config.toml'`) that indicates the intended source and target layers for the predictive feedback, three lines of code are enough to construct the corresponding predictive network:

```
from predify import predify

net = # load PyTorch network
predify(net,'./config.toml') # config file indicates the layers that
                             # will act as outputs of encoders.
```

[2] https://toml.io/en/

The Appendix A.1 provides further details on the package, along with a sample config file and certain default behaviours. *Predify* is an ongoing project available on GitHub[3] under GNU General Public License v3.0. Scripts for creating PVGG16 and PEfficientNetB0 from their feedforward instances and reproducing the results presented in this paper, as well as the pre-trained weights are also available on another GitHub repository[4].

3 Related work

There is a long tradition of drawing inspiration from neuroscience knowledge to improve machine learning performance. Some studies suggest using sparse coding, a concept closely related to predictive coding [16, 27, 28, 29, 30], for image denoising [31] and robust deep learning [32, 33], while other studies focus on implementing feedback and horizontal recurrent pathways to tackle challenges beyond the core object recognition [13, 14, 25, 34, 35, 36, 37, 38, 39].

Here, we focus specifically on those studies that tried implementing predictive coding mechanisms in machine learning models [20, 21, 22, 23]. Out of these, our implementation is most similar to the Predictive Coding Networks (PCNs) of [22]. These hierarchical networks were designed with a similar goal in mind: improving object recognition with predictive coding dynamics. However, their network (including the feedback connection weights) is solely optimized with a classification objective. As a result, their network does not learn to uniformly reduce reconstruction errors over timesteps, as the predictive coding theory would mandate. We also found that their network performs relatively poorly until the final timestep (see corresponding Figure S1 in the Appendix A.5), which does not seem biologically plausible: biological systems typically cannot afford to wait until the last iteration before detecting a prey or a predator. In the proposed method, we incorporate the feedforward drive into a similar PC dynamics and train the feedback weights in an unsupervised way using a reconstruction loss. We then show that these modifications help resolve PCNs' issues. We discuss these PCNs [22] further in the Appendix A.5, together with our own detailed exploration of their network's behavior.

Other approaches to predictive coding for object recognition include Boutin et al. [23], who used a PCN with an additional sparsity constraint. The authors showed that their framework can give rise to receptive fields which resemble those of neurons in areas V1 and V2 of the primate brain. They also demonstrated the robustness of the system to noisy inputs, but only in the context of reconstruction. Unlike ours, they did not show that their network can perform (robust) classification, and they did not extend their approach to deep neural networks.

Spratling [40] also described PCNs designed for object recognition, and demonstrated that their network could effectively recognise digits and faces, and locate cars within images. Their update equations differed from ours in a number of ways: they used divisive/multiplicative error correction (rather than additive), and a form of biased competition to make the neurons "compete" in their explanatory power. The weights of the network were not trained by error backpropagation, making it difficult to scale it to address modern machine learning problems. Conversely, our proposed network architecture and PC dynamics are fully compatible with error backpropagation, making them a suitable option for large-scale problems. Indeed, the tasks on which they tested their network are simpler than ours, and the datasets are much smaller.

Huang et al. [41] also aimed to extend the principle of predictive coding by incorporating feedback connections such that the network maximizes "self consistency" between the input image features, latent variables and label distribution. The iterative dynamics they proposed, though different from ours, improved the robustness of neural networks against gradient-based adversarial attacks on datasets such as Fashion-MNIST and CIFAR10.

4 Results

Here we contrast the behavior of feedforward networks with their predictive coding augmentations. When considered at timestep 0 (i.e., after a single feedforward and feedback pass through the model), the deep predictive coding networks (DPCNs) and their accuracy are—by construction—exactly

[3] `https://github.com/miladmozafari/predify`

[4] `https://github.com/bhavinc/predify2021`

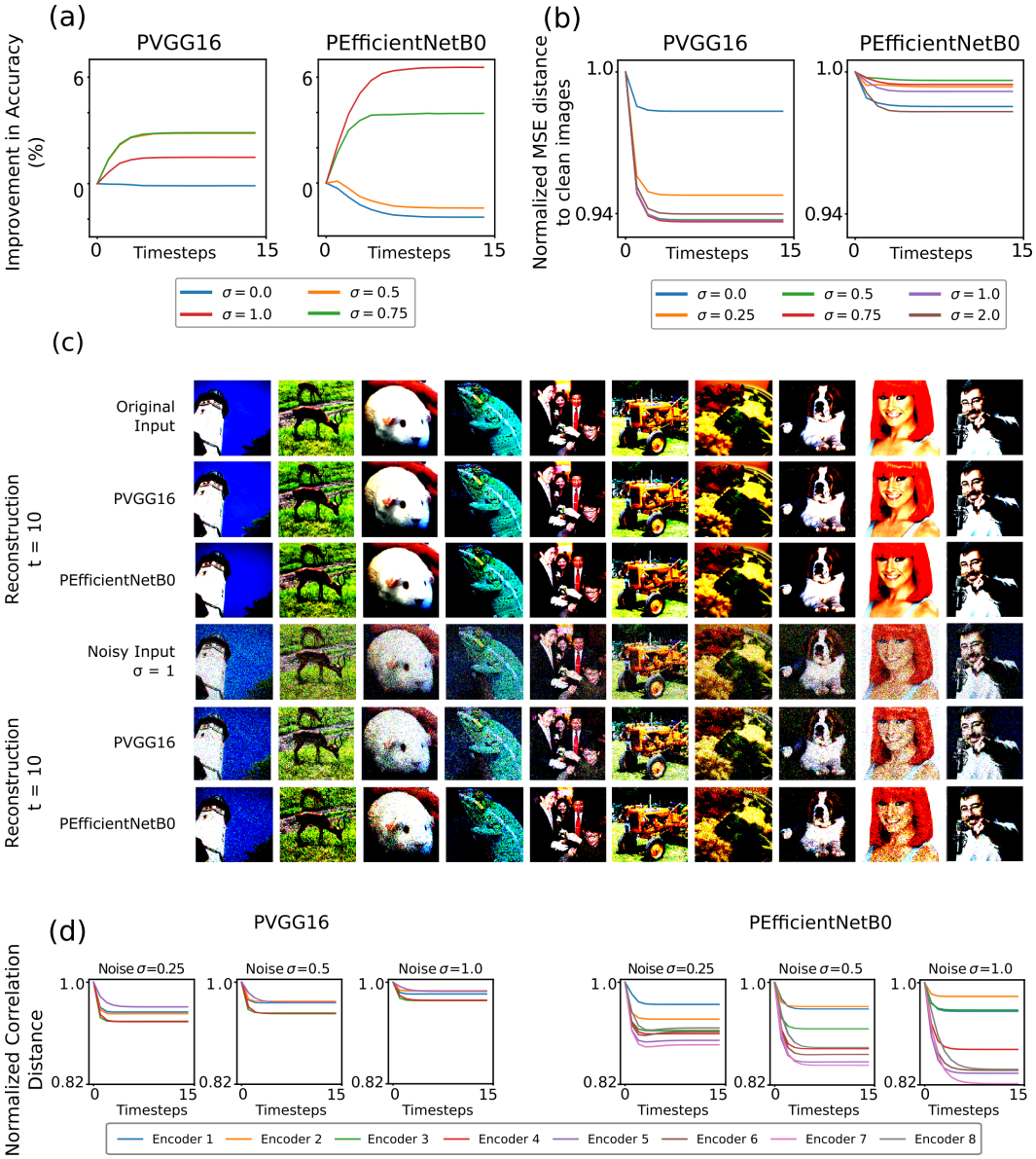


Figure 2: **Performance under Gaussian noise and projection towards the learned manifold.** (a) Improvement in recognition accuracy with reference to the feedforward baseline under various levels of Gaussian noise. Both networks demonstrate significant accuracy improvement across timesteps under noisy conditions, while maintaining a performance close to the feedforward level for clean images. (b) Normalized MSE distance between the image reconstruction (d_0) and the clean image (e_0). Irrespective of the noise level, image reconstruction consistently gets closer to the clean image across timesteps in both models. (c) Examples of clean and noisy input images together with their final reconstruction by the model (the row order from top to bottom is: original image, PVGG16 reconstruction, PEfficientNetB0 reconstruction; noisy image, PVGG16 reconstruction, PEfficientNetB0 reconstruction). For best viewing, we recommend zooming in on the electronic version. (d) Normalized correlation distance between representation of clean and noisy images for each encoder (e_i) across timesteps. The values are normalized with respect to the feedforward baseline (timestep 0). In both models and all encoders, the noisy representations tend to move toward the clean copies.

identical to their standard pretrained feedforward versions. Over successive timesteps, however, the influence of feedback and predictive coding iterations becomes visible. Here, we investigate for both DPCNs (PVGG16 and PEfficientNetB0): (i) how the PC dynamics update the networks' representations across timesteps, and in which direction relative to the learned manifold; (ii) how the networks benefit from PC under noisy conditions, or against adversarial attacks.

4.1 Performance under Gaussian noise

To understand the evolution of representations and the behavior of the proposed DPCNs, we first investigate their performance under the influence of different levels of Gaussian noise. To this end, we inject additive Gaussian noise to the ImageNet validation set, and monitor the models' performance across timesteps.

In Figure 2a we provide the classification accuracy on these noisy images and absolute values in the Table S4. We observed that both models progressively improve their recognition accuracy relative to their feedforward baseline (timestep 0) over successive iterations while imposing only a minor performance reduction on clean images. In other words, the networks are able to discard some of the noise by leveraging the predictive coding dynamics over timesteps.

4.2 Projection towards the learned manifold

In order to quantify DPCNs' denoising ability, we evaluate the quality of image reconstructions generated by each network using the mean squared error (MSE) between the clean image and its reconstruction generated by the first decoder. For each DPCN, we normalize these distances, by dividing them by the value obtained for the corresponding feedforward network (at t=0). We provide the absolute values in the Table S5. As Figures 2b-c illustrate, the reconstructions become progressively cleaner over timesteps. It should be noted that the feedback connections were trained only to reconstruct clean images; therefore, this denoising property is an emerging feature of the PC dynamics.

Next, we test whether the higher layers of the proposed DPCNs also manifest this denoising property. Hence, we pass clean and noisy versions of all images from the ImageNet validation set through the networks, and measure the average correlation distance between the clean and noisy representations of each encoder at each timestep. As done above, these correlation distances are then normalized with the distance measured at timestep 0 (i.e., relative to the standard feedforward network). For both the networks, the correlation distances decrease consistently over timesteps across all layers (see Figure 2d). This implies that predictive coding iterations help the networks to steer the noisy representations closer to the representations elicited by the corresponding (unseen) clean image.

This is an important property for robustness. When compared to clean images, noisy images can result in different representations at higher layers [42] and consequently, produce significant classification errors. Various defenses have aimed to protect neural networks from perturbations and adversarial attacks by constraining the images to the "original data manifold". Accordingly, studies have used generative models such as GANs [43, 44, 45, 46] or PixelCNNs [47] to constrain the input to the data manifold. Similarly, multiple efforts have been made to clean the representations in higher layers and keep them closer to the learned latent space [42, 48, 49, 50]. Here, we demonstrate that feedback predictive coding iterations can achieve a similar goal by iteratively projecting noisy representations towards the manifolds learned during training, both in pixel (Figure 2b-c) and representation spaces (Figure 2d).

4.3 Benchmarking robustness to ImageNet-C

Given the promising results with additive Gaussian noise (Figure 2), we extend the noise variety and quantify the classification accuracy of the networks under different types of perturbations. We use ImageNet-C, a benchmarking dataset for noise robustness provided by [8], including 19 types of image corruptions across 5 severity levels each. To begin with, we evaluate DPCNs with pre-defined hyperparameter values (as provided in subsection 2.2). We observe that they improve the Corruption Error (CE) scores over timesteps for several of the additive-noise corruptions: Gaussian noise, shot noise, impulse noise or speckle noise (see Figure 3), but fail to improve the overall mean Corruption Error, or mCE score (the recommended score for this benchmark [8]).

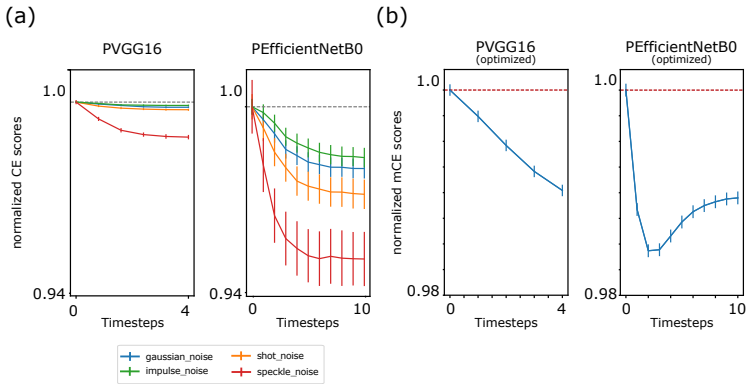


Figure 3: **Benchmarking robustness to ImageNet-C.** (a) Normalized corruption errors (CE) of PVGG16 and PEfficientNetB0 under four types of additive noise corruptions. The values are normalized with respect to the feedforward baseline. Both networks show consistent reductions in the errors across timesteps. (b) Normalized mean Corruption Error (mCE) scores for PVGG16 and PEfficientNetB0 on all the 19 corruptions available in the ImageNet-C dataset, when optimized hyperparameters are used (as described in the Appendix A.7). The values are normalized with respect to the feedforward baseline. In both the panels, error bars represent the standard deviation of the bootstrapped estimate of the mean value.

Thus, instead of using pre-defined hyperparameter values, we fine-tune them using two different methods (see Appendix A.7), and repeat the above experiment. As shown in Figure 3b, when the hyperparameters are more appropriately tuned for the task, the PC dynamics can increase noise robustness more generally across noise types, resulting in improvements of the mean Corruption Error (mCE) score. The CE plots for individual perturbations along with other recommended metrics (values normalized with AlexNet scores, Relative mCE scores) are provided in the Appendix A.8.

Furthermore, in the Appendix A.9, we demonstrate that we can replicate these observations with a version of PEfficientNetB0 provided by [51] that is robust to corruptions in the ImageNet-C dataset. We show that the recurrent dynamics we propose still help in further improving the mCE score of this already robust network.

4.4 Benchmarking robustness to adversarial attacks

Finally, we evaluate the robustness of the networks across timesteps against adversarial attacks. The proposed DPCNs are recurrent models, meaning that their layer representations change on every timestep, and consequently, so do the classification boundaries in the last layer, leading to different accuracy and generalization errors across time (as seen above). To mitigate this effect and properly assess the changes in robustness due to the PC dynamics, for each network we start by selecting 1000 images from the ImageNet validation dataset such that they are correctly classified across all timesteps. Also, we only perform *targeted* attacks so that for each image, the same attack target is given for all timesteps. Using the *Foolbox* library [52], we conduct targeted Basic_Iterative_Method attacks (BIM, with L_∞ norm) [53] for both networks; although it would prove computationally prohibitive to systematically explore all standard types of adversarial attacks, we also evaluated random Projected Gradient Descent attacks (RPGD, with L_2 norm) [54], and non-gradient-based HopSkipJump attacks [55] on a subset of 100 images, specifically for PEfficientNetB0. Across various levels of allowed image perturbations (denoted as ϵs), the predictive coding iterations tend to decrease the success rate of the attacks across timesteps, for both networks and attacks (see Figure 4). That is, DPCNs are more robust against these adversarial attacks than their feedforward counterparts.

5 Discussion and Conclusion

In this work, we explore the use of unsupervised recurrent predictive coding (PC) dynamics, based on neuroscientific principles, to augment modern deep neural networks. The resulting models have an initial feedforward sweep, compatible with visual processing in human and macaque brains [11, 56, 57, 58]. Following this feedforward sweep, consecutive layers iteratively exchange information

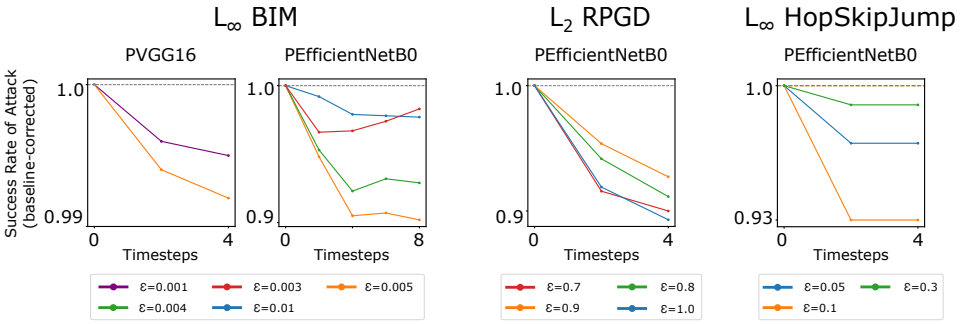


Figure 4: **Benchmarking robustness to adversarial attacks.** Plots show the success rate of targeted adversarial attacks against DPCNs across timesteps. The values are baseline-corrected, relative to the success rate at timestep 0 (feedforward baseline). Both networks demonstrate improved robustness to different types and/or levels of perturbations.

regarding predictions and prediction errors, aiming to converge towards a stable explanation of the input. This dynamic system is inspired by, and reminiscent of, the "canonical microcircuit" (a central component of cortical structure [15]) that relies on feedback signaling between hierarchically adjacent layers to update its activity. Overall, the augmented networks are closer to the architecture of biological visual systems, while gaining some desirable functional properties. For example, in [59], we also demonstrated that the proposed dynamics help the networks perceive illusory contours in a similar way to humans.

Here, we implemented these PC dynamics in two state-of-the-art DCNs, VGG16 and EfficientNetB0, and showed that they helped to improve the robustness of the networks against various corruptions (e.g. ImageNet-C). We demonstrated that this behavior, at least partly, stems from PC's ability to project both the corrupted image reconstructions and neural representations towards their clean counterparts.

We also tested the impact of our network augmentations against adversarial attacks; here again, we showed that PC helps to improve the robustness of the networks. So far, the most promising strategy for achieving robustness has been adversarial training, whereby adversarial datapoints are added to the training dataset. While efficient, this strategy was also shown to be strongly limited [60, 61]. Apart from factors like the choice of the norms used for training, or the high computation requirements, it is ultimately performed with a supervised loss function that can alter the decision boundaries in undesirable ways [61, 62]. Most importantly, adversarial training shares very little, if any, resemblance to the way the brain achieves robustness. Instead, here we start from biological principles and show that they can lead to improved adversarial robustness. It is worth mentioning that both our networks achieved robustness totally via unsupervised training of the feedback connections (while of course, the backbone feed-forward networks that we used were pretrained in a supervised manner). We avoided using costly adversarial training, or tuning our hyperparameters specifically for classification under each attack. This likely explains why the models, while improving in robustness compared to their feedforward versions, remain far from state-of-the-art adversarial defenses. On the other hand, we believe that addition of these methods (adversarial training, hyperparameter tuning) to the training paradigm, in future work, could further improve the networks' adversarial robustness.

For the present experiments, we made a choice of using different objectives for training the feedforward and feedback weights: pre-trained feedforward weights optimized for classification, feedback weights trained with a reconstruction objective (computed after a single time-step). On the one hand, we note that it is perfectly feasible to train a similar predictive coding architecture with a single objective (classification, reconstruction, or otherwise) for both feedforward and feedback weights [59, 63]. On the other hand, our choice has several advantages. First, using a feedforward backbone pretrained for classification allowed us to demonstrate the effect of our dynamics on pre-existing state-of-the-art neural networks. Some authors have tried training both feedforward and feedback connections together for classification [22] at the final timestep for relatively smaller networks, but as we discussed in our explorations in the Appendix A.5, we found that the resulting network ended up classifying correctly at the last timestep, with very poor performance during early timesteps. This problem could be addressed by training over time-averaged metrics, such as the average cross-entropy loss for N

timesteps. Nonetheless, training the feedback weights for reconstruction instead of classification has the additional advantage that it can be done entirely without supervision. We chose to train the feedback weights for a single time-step, because training with recurrence over multiple timesteps would have required unrolling the network over time. Hence, training a large network like PVGG16 for say 5 or 10 timesteps would incur significant computational challenges. Furthermore, our use of a one-step reconstruction objective allowed us to train the feedback weights independently of the various hyperparameters of our predictive coding dynamics (β, λ, and α), which only influence the model behavior after the second timestep. Training these weights using recurrence would have required to (i) either fix the values of these hyperparameters beforehand, leading to constraints of expensive hyperparameter explorations; (ii) or directly train these hyperparameters as parameters of the model, probably with additional constraints to prevent the network from reaching trivial values (e.g., if all hyperparameters but the feedforward term β converge to zero, the network performs identically to a feedforward one). Finally, from a neuroscience perspective, whether and how the brain combines discriminative and generative representations has been an open question addressed by many researchers, e.g. [5, 64, 65]. Our approach of a discriminative (classification-trained) feedforward coupled with generative (reconstruction-trained) feedback could be considered another attempt in this direction.

We speculate that the proposed PC dynamics could help improve robustness in most feedforward neural architectures. To facilitate further explorations in this direction, we provided a Python package, called *Predify*, which allows users to implement recurrent PC dynamics in any feedforward DCN, with only a few lines of code. *Predify* automates the network building, and thus simplifies experiments. On the other hand, there is as yet no established method or criteria to automate the process of identifying the appropriate number of encoding layers, their source and target layers in the DCN hierarchy, and the corresponding hyperparameter values. This remains an open research question, and a requirement for manual explorations and tuning from *Predify* users. For instance, our own explorations with augmenting ResNets through *Predify* proved difficult, and failed in some situations but succeeded in others. More specifically, as developed in [63] using *Predify*, ResNet augmentations always achieved noise robustness when the hyperparameter values (controlling the feedforward, feedback, and memory terms) could be tuned separately for each noise type; but we found it challenging to identify a single set of hyperparameters that could generalize to all noise types. Nonetheless, we are hopeful that the package will prove useful to the community. The code is structured such that users can readily adapt it to test their hypotheses. In particular, it should allow both proponents and opponents of the predictive coding theory to investigate its effects on any DCN.

Overall, this work contributes to the general case for continuing to draw inspiration from biological visual systems in computer vision, both at the level of model architecture and dynamics. We believe that our user-friendly Python package *Predify* can open new opportunities, even for neuroscience researchers with little background in machine learning, to investigate bio-inspired hypotheses in deep computational models, and thus bridge the gap between the two communities.

6 Broader Impacts

The research discussed above proposes novel ways of using brain-inspired dynamics in current machine learning models. Specifically, it demonstrates a neuro-inspired method for improving the robustness of machine learning models. Given that such models are employed by the general public, and are simultaneously shown to be heavily vulnerable, research efforts to increase (even marginally) or to understand their robustness against mal-intentioned adversaries has high societal relevance.

Importantly, the research also aims to bridge techniques between two different fields–neuroscience and machine learning, which can potentially open new avenues for studying the human brain. For example, it could help better understand the unexplained neural activities in patients, to improve their living conditions, and in the best case, in the treatment of their conditions. While this may also be associated with inherent risks (related to privacy or otherwise), there are clear potential benefits to society.

The likelihood of sentient AI arising from this line of research is estimated to be rather low.

Acknowledgments and Disclosure of Funding

This work was funded by an ANITI (Artificial and Natural Intelligence Toulouse Institute) Research Chair to RV (ANR grant ANR-19-PI3A-0004), as well as ANR grants AI-REPS (ANR-18-CE37-0007-01) and OSCI-DEEP (ANR-19-NEUC-0004).

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

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

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. corr abs/1512.03385 (2015), 2015.

[4] Gao Huang, Zhuang Liu, and Kilian Q Weinberger. Densely connected convolutional networks. corr abs/1608.06993 (2016). *arXiv preprint arXiv:1608.06993*, 2016.

[5] Haider Al-Tahan and Yalda Mohsenzadeh. Reconstructing feedback representations in the ventral visual pathway with a generative adversarial autoencoder. *PLoS Computational Biology*, 17(3):e1008775, 2021.

[6] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[7] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[8] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*, 2019.

[9] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. arxiv e-prints, art. *arXiv preprint arXiv:1412.1897*, 2014.

[10] Tim C Kietzmann, Courtney J Spoerer, Lynn KA Sörensen, Radoslaw M Cichy, Olaf Hauk, and Nikolaus Kriegeskorte. Recurrence is required to capture the representational dynamics of the human visual system. *Proceedings of the National Academy of Sciences*, 116(43):21854–21863, 2019.

[11] Kohitij Kar, Jonas Kubilius, Kailyn Schmidt, Elias B Issa, and James J DiCarlo. Evidence that recurrent circuits are critical to the ventral stream's execution of core object recognition behavior. *Nature neuroscience*, 22(6):974–983, 2019.

[12] Markus Roland Ernst, Jochen Triesch, and Thomas Burwick. Recurrent connections aid occluded object recognition by discounting occluders. In *International Conference on Artificial Neural Networks*, pages 294–305. Springer, 2019.

[13] Drew Linsley, Alekh Karkada Ashok, Lakshmi Narasimhan Govindarajan, Rex Liu, and Thomas Serre. Stable and expressive recurrent vision models. *arXiv preprint arXiv:2005.11362*, 2020.

[14] Dean Wyatte, Tim Curran, and Randall O'Reilly. The limits of feedforward vision: Recurrent processing promotes robust object recognition when objects are degraded. *Journal of Cognitive Neuroscience*, 24(11):2248–2261, 2012.

[15] Andre M Bastos, W Martin Usrey, Rick A Adams, George R Mangun, Pascal Fries, and Karl J Friston. Canonical microcircuits for predictive coding. *Neuron*, 76(4):695–711, 2012.

[16] Yanping Huang and Rajesh PN Rao. Predictive coding. *Wiley Interdisciplinary Reviews: Cognitive Science*, 2(5):580–593, 2011.

[17] Micha Heilbron and Maria Chait. Great expectations: is there evidence for predictive coding in auditory cortex? *Neuroscience*, 389:54–73, 2018.

[18] Laurence Aitchison and Máté Lengyel. With or without you: predictive coding and bayesian inference in the brain. *Current opinion in neurobiology*, 46:219–227, 2017.

[19] Kevin S Walsh, David P McGovern, Andy Clark, and Redmond G O'Connell. Evaluating the neurophysiological evidence for predictive processing as a model of perception. *Annals of the New York Academy of Sciences*, 1464(1):242, 2020.

[20] Rakesh Chalasani and Jose C Principe. Deep predictive coding networks. *arXiv preprint arXiv:1301.3541*, 2013.

[21] William Lotter, Gabriel Kreiman, and David Cox. Deep predictive coding networks for video prediction and unsupervised learning. *arXiv preprint arXiv:1605.08104*, 2016.

[22] Haiguang Wen, Kuan Han, Junxing Shi, Yizhen Zhang, Eugenio Culurciello, and Zhongming Liu. Deep predictive coding network for object recognition. *arXiv preprint arXiv:1802.04762*, 2018.

[23] Victor Boutin, Angelo Franciosini, Frederic Chavane, Franck Ruffier, and Laurent Perrinet. Sparse deep predictive coding captures contour integration capabilities of the early visual system. *arXiv preprint arXiv:1902.07651*, 2019.

[24] Rajesh PN Rao and Dana H Ballard. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature neuroscience*, 2(1):79–87, 1999.

[25] David J Heeger. Theory of cortical function. *Proceedings of the National Academy of Sciences*, 114(8):1773–1782, 2017.

[26] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[27] Bruno A Olshausen and David J Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

[28] Erkki Oja, Aapo Hyvärinen, and Patrik Hoyer. Image feature extraction and denoising by sparse coding. *Pattern Analysis & Applications*, 2(2):104–110, 1999.

[29] Matthew Chalk, Olivier Marre, and Gašper Tkačik. Toward a unified theory of efficient, predictive, and sparse coding. *Proceedings of the National Academy of Sciences*, 115(1):186–191, 2018.

[30] Dylan M Paiton, Charles G Frye, Sheng Y Lundquist, Joel D Bowen, Ryan Zarcone, and Bruno A Olshausen. Selectivity and robustness of sparse coding networks. *Journal of Vision*, 20(12):10–10, 2020.

[31] Xiaoqiang Lu, Yuan Yuan, and Pingkun Yan. Sparse coding for image denoising using spike and slab prior. *Neurocomputing*, 106:12–20, 2013.

[32] Jeremias Sulam, Ramchandran Muthukumar, and Raman Arora. Adversarial robustness of supervised sparse coding. *Advances in neural information processing systems*, 2020.

[33] Edward Kim, Jocelyn Rego, Yijing Watkins, and Garrett T Kenyon. Modeling biological immunity to adversarial examples. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4666–4675, 2020.

[34] Courtney J Spoerer, Patrick McClure, and Nikolaus Kriegeskorte. Recurrent convolutional neural networks: a better model of biological object recognition. *Frontiers in psychology*, 8:1551, 2017.

[35] Drew Linsley, Junkyung Kim, Vijay Veerabadran, Charles Windolf, and Thomas Serre. Learning long-range spatial dependencies with horizontal gated recurrent units. In *Advances in neural information processing systems*, pages 152–164, 2018.

[36] Nicholas Frosst, Sara Sabour, and Geoffrey E. Hinton. Darccc: Detecting adversaries by reconstruction from class conditional capsules. *ArXiv*, abs/1811.06969, 2018.

[37] Dmitry Krotov and John Hopfield. Dense associative memory is robust to adversarial inputs. *Neural computation*, 30(12):3151–3167, 2018.

[38] Jonas Kubilius, Martin Schrimpf, Kohitij Kar, Rishi Rajalingham, Ha Hong, Najib Majaj, Elias Issa, Pouya Bashivan, Jonathan Prescott-Roy, Kailyn Schmidt, Aran Nayebi, Daniel Bear, Daniel L Yamins, and James J DiCarlo. Brain-like object recognition with high-performing shallow recurrent anns. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[39] Karim Rajaei, Yalda Mohsenzadeh, Reza Ebrahimpour, and Seyed-Mahdi Khaligh-Razavi. Beyond core object recognition: Recurrent processes account for object recognition under occlusion. *PLoS computational biology*, 15(5):e1007001, 2019.

[40] Michael W Spratling. A hierarchical predictive coding model of object recognition in natural images. *Cognitive computation*, 9(2):151–167, 2017.

[41] Yujia Huang, James Gornet, Sihui Dai, Zhiding Yu, Tan Nguyen, Doris Tsao, and Anima Anandkumar. Neural networks with recurrent generative feedback. *Advances in Neural Information Processing Systems*, 33, 2020.

[42] Cihang Xie, Yuxin Wu, Laurens van der Maaten, Alan L Yuille, and Kaiming He. Feature denoising for improving adversarial robustness. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 501–509, 2019.

[43] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. Defense-gan: Protecting classifiers against adversarial attacks using generative models. *International Conference on Learning Representations*, 2018.

[44] Guoqing Jin, Shiwei Shen, Dongming Zhang, Feng Dai, and Yongdong Zhang. Ape-gan: Adversarial perturbation elimination with gan. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3842–3846, 2019.

[45] Dongyu Meng and Hao Chen. Magnet: a two-pronged defense against adversarial examples. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 135–147, 2017.

[46] Ajil Jalal, Andrew Ilyas, Constantinos Daskalakis, and Alexandros G Dimakis. The robust manifold defense: Adversarial training using generative models. *arXiv preprint arXiv:1712.09196*, 2017.

[47] Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. In *International Conference on Learning Representations*, 2018.

[48] Guanhong Tao, Shiqing Ma, Yingqi Liu, and Xiangyu Zhang. Attacks meet interpretability: Attribute-steered detection of adversarial samples. In *Advances in Neural Information Processing Systems*, pages 7717–7728, 2018.

[49] A Emin Orhan and Brenden M Lake. Improving the robustness of imagenet classifiers using elements of human visual cognition. *arXiv preprint arXiv:1906.08416*, 2019.

[50] Kevin Roth, Yannic Kilcher, and Thomas Hofmann. The odds are odd: A statistical test for detecting adversarial examples. In *International Conference on Machine Learning*, pages 5498–5507. PMLR, 2019.

[51] Cihang Xie, Mingxing Tan, Boqing Gong, Jiang Wang, Alan L Yuille, and Quoc V Le. Adversarial examples improve image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 819–828, 2020.

[52] Jonas Rauber, Wieland Brendel, and Matthias Bethge. Foolbox: A python toolbox to benchmark the robustness of machine learning models. In *Reliable Machine Learning in the Wild Workshop, 34th International Conference on Machine Learning*, 2017.

[53] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

[54] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

[55] Jianbo Chen, Michael I Jordan, and Martin J Wainwright. Hopskipjumpattack: A query-efficient decision-based attack. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1277–1294. IEEE Computer Society, 2020.

[56] Simon Thorpe, Denis Fize, and Catherine Marlot. Speed of processing in the human visual system. *nature*, 381(6582):520–522, 1996.

[57] Chou P Hung, Gabriel Kreiman, Tomaso Poggio, and James J DiCarlo. Fast readout of object identity from macaque inferior temporal cortex. *Science*, 310(5749):863–866, 2005.

[58] Rufin VanRullen. The power of the feed-forward sweep. *Advances in Cognitive Psychology*, 3(1-2):167, 2007.

[59] Zhaoyang Pang, Callum Biggs O'May, Bhavin Choksi, and Rufin VanRullen. Predictive coding feedback results in perceived illusory contours in a recurrent neural network. *Neural Networks*, 144:164–175, 2021.

[60] Ludwig Schmidt, Shibani Santurkar, Dimitris Tsipras, Kunal Talwar, and Aleksander Madry. Adversarially robust generalization requires more data. In *Advances in Neural Information Processing Systems*, pages 5014–5026, 2018.

[61] Haichao Zhang and Jianyu Wang. Defense against adversarial attacks using feature scattering-based adversarial training. In *Advances in Neural Information Processing Systems*, pages 1831–1841, 2019.

[62] Thomas Tanay and Lewis Griffin. A boundary tilting persepective on the phenomenon of adversarial examples. *arXiv preprint arXiv:1608.07690*, 2016.

[63] Andrea Alamia, Milad Mozafari, Bhavin Choksi, and Rufin VanRullen. On the role of feedback in visual processing: a predictive coding perspective. *arXiv preprint arXiv:2106.04225*, 2021.

[64] James J DiCarlo, Ralf Haefner, Leyla Isik, Talia Konkle, Nikolaus Kriegeskorte, Benjamin Peters, Nicole Rust, Kim Stachenfeld, Joshua B Tenenbaum, Doris Tsao, et al. How does the brain combine generative models and direct discriminative computations in high-level vision? *CCN 2021 Workshop GAC*, 2021. `https://openreview.net/forum?id=zlTiwFtLlR4`.

[65] Derek J Huffman and Craig EL Stark. Multivariate pattern analysis of the human medial temporal lobe revealed representationally categorical cortex and representationally agnostic hippocampus. *Hippocampus*, 24(11):1394–1403, 2014.

[66] Michael W Spratling. Predictive coding as a model of biased competition in visual attention. *Vision research*, 48(12):1391–1408, 2008.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] Please refer to Section 6
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A] We draw comparisons with Rao and Ballard's implementation, and provide step-by-step derivations in the Appendix.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] The scripts and data of the main experiments will be available online on GitHub. At the review stage, we included the source code in the supplementary materials for the purpose of anonymization.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] All the important details are provided in Section 2 and the Appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] The corresponding details are given in Figure captions.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes] See subsection 2.3
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We include the proposed Predify package in the supplemental material.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] All assets used were open-source with appropriate licenses.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Appendix

A.1 Getting Started with Predify

Both VGG16 and EfficientNetB0 are converted to predictive coding networks PVGG16 and PEfficientNetB0, using the Predify package. The fastest and easiest way to convert a feedforward network into its predictive coding version is to use Predify's text-based interface which supports configuration files in TOML format.

The current version of Predify assumes that there is no gap between the encoders. Therefore, in the minimal case, one only needs to provide a list of sub-module names in the target feedforward network. Then, Predify takes care of the rest by converting each of them into an encoder and assigning default decoders. More precisely, let x and y denote the input and output of a layer (or complex sub-module, potentially including multiple layers) that is selected to be an encoder (e_n). If x and y respectively have the size (c_{in}, h_{in}, w_{in}) and $(c_{out}, h_{out}, w_{out})$; then, the default decoder's structure that predicts this encoder (d_{n+1}) is a 2D upscaling operation by the factor of $(h_{in}/h_{out}, w_{in}/w_{out})$ followed by a transposed convolutional layer with c_{out} channels and 3×3 window size. The values of hyperparameters will be set to $\beta_n = 0.3$, $\lambda_n = 0.3$, and $\alpha_n = 0.01$

In Predify, each encoder (e_n) and the decoder that uses its output to predict the activity of the encoder below (d_{n-1}) is called a *PCoder*. To verify the functionality of Predify's default settings, we applied it for PEfficientNetB0 used in this work. Here is the corresponding minimal configuration file:

```
name = "PEfficientNetB0"

input_size = [3,224,224]
gradient_scaling = true
shared_hyperparameters = false

[[pcoders]]
module = "act1"
[[pcoders]]
module = "blocks[0]"
[[pcoders]]
module = "blocks[1]"
[[pcoders]]
module = "blocks[2]"
[[pcoders]]
module = "blocks[3]"
[[pcoders]]
module = "blocks[4]"
[[pcoders]]
module = "blocks[5]"
[[pcoders]]
module = "blocks[6]"
```

One can easily override the default setting by providing all the details for a PCoder. Here is the configuration corresponding to the PVGG16 used in this work:

```
imports = [
"from torch.nn import Sequential, ReLU, ConvTranspose2d",
]

name = "PVGG16"

input_size = [3, 224, 224]
gradient_scaling = true
shared_hyperparameters = false

[[pcoders]]
module = "features[3]"
predictor = "ConvTranspose2d(64, 3, kernel_size=(5, 5), stride=(1, 1),
    padding=(2, 2))"
hyperparameters = {feedforward=0.2, feedback=0.05, pc=0.02}
```

```
[[pcoders]]
module = "features[8]"
predictor = "Sequential(ConvTranspose2d(128, 64, kernel_size=(10, 10),
    stride=(2, 2), padding=(4, 4)), ReLU(inplace=True))"
hyperparameters = {feedforward=0.4, feedback=0.1, pc=0.05}

[[pcoders]]
module = "features[15]"
predictor = "Sequential(ConvTranspose2d(256, 128, kernel_size=(14, 14)
    , stride=(2, 2), padding=(6, 6)), ReLU(inplace=True))"
hyperparameters = {feedforward=0.4, feedback=0.1, pc=0.008}

[[pcoders]]
module = "features[22]"
predictor = "Sequential(ConvTranspose2d(512, 256, kernel_size=(14, 14)
    , stride=(2, 2), padding=(6, 6)), ReLU(inplace=True))"
hyperparameters = {feedforward=0.5, feedback=0.1, pc=0.0024}

[[pcoders]]
module = "features[29]"
predictor = "Sequential(ConvTranspose2d(512, 512, kernel_size=(14, 14)
    , stride=(2, 2), padding=(6, 6)), ReLU(inplace=True))"
hyperparameters = {feedforward=0.6, feedback=0.0, pc=0.006}
```

The network configuration files (in TOML format) are available to download on GitHub[5].

A.2 Network Architectures

VGG16 consists of five convolution blocks and a classification head. Each convolution block contains two or three convolution+ReLU layers with a max-pooling layer on top. For each e_n in PVGG16, we selected the max-pooling layer in block $n-1$ and all the convolution layers in block n of VGG16 (for $n \in \{1, 2, 3, 4, 5\}$) as the sub-module that provides the feedforward drive. Afterwards, to predict the activity of each e_n, a deconvolution layer d_n is added which takes the e_{n+1} as the input. Here, deconvolution kernel sizes are set by taking the increasing receptive field sizes into account.

In the case of PEfficientNetB0, we used PyTorch implementation of EfficientNetB0 provided in `https://github.com/rwightman/pytorch-image-models`. This implementation of EfficientNetB0 consists of eight blocks of layers (considering the first convolution and batch normalization layers as a separate block). Similar to PVGG16, we convert each of these blocks into an encoder (e_n) and add deconvolution layers accordingly. This time we set the kernel size of all deconvolution layers to 3x3 and use upsampling layers to compensate the shrinkage of layer size through the feedforward pathway (i.e. Predify's default setting).

Table S1 summarizes PVGG16's architecture. Moreover, the hyperparameter values are provided in Tables S2 and S3.

[5] `https://github.com/bhavinc/predify2021`

Table S1: Architectures of e_ns and d_ns for PVGG16 and PEfficientNetB0. Conv (channel, size, stride), MaxPool (size, stride), Deconv (channel, size, stride), Upsample (scale_factor), BN is BatchNorm, $[\,]_+$ is ReLU, and $[\,]_*$ is SiLU. EfficientBlock corresponds to each block in PyTorch implementation of EfficientNetB0.

	PVGG16 Input Size: 3x224x224		PEfficientNetB0 Input Size: 3x224x224	
	e_n	d_{n-1}	e_n	d_{n-1}
PCoder1	$[\text{Conv (64, 3, 1)}]_+$ $[\text{Conv (64, 3, 1)}]_+$	Deconv (3, 5, 1)	$[\text{BN (Conv (32, 3, 2))}]_*$	Upsample (2) Deconv (3, 3, 1)
PCoder2	MaxPool (2, 2) $[\text{Conv (128, 3, 1)}]_+$ $[\text{Conv (128, 3, 1)}]_+$	$[\text{Deconv (64, 10, 2)}]_+$	EfficientBlock0	Deconv (32, 3, 1)
PCoder3	MaxPool (2, 2) $[\text{Conv (256, 3, 1)}]_+$ $[\text{Conv (256, 3, 1)}]_+$ $[\text{Conv (256, 3, 1)}]_+$	$[\text{Deconv (128, 14, 2)}]_+$	EfficientBlock1	Upsample (2) Deconv (16, 3, 1)
PCoder4	MaxPool (2, 2) $[\text{Conv (512, 3, 1)}]_+$ $[\text{Conv (512, 3, 1)}]_+$ $[\text{Conv (512, 3, 1)}]_+$	$[\text{Deconv (256, 14, 2)}]_+$	EfficientBlock2	Upsample (2) Deconv (24, 3, 1)
PCoder5	MaxPool (2, 2) $[\text{Conv (512, 3, 1)}]_+$ $[\text{Conv (512, 3, 1)}]_+$ $[\text{Conv (512, 3, 1)}]_+$	$[\text{Deconv (512, 14, 2)}]_+$	EfficientBlock3	Upsample (2) Deconv (40, 3, 1)
PCoder6	-	-	EfficientBlock4	Deconv (80, 3, 1)
PCoder7	-	-	EfficientBlock5	Upsample (2) Deconv (112, 3, 1)
PCoder8	-	-	EfficientBlock6	Deconv (192, 3, 1)

A.3 Execution Time

Since we used a variable number of GPUs for the different experiments, an exact execution time is hard to pinpoint. Briefly, depending on the number of timesteps, analysing mCE scores and adversarial attacks on PEfficientNetB0 took around 15-20 hours on an NVIDIA TitanV gpu. These numbers were about three to four times higher for experiments on PVGG16. For both the networks, training the feedback weights on the ImageNet dataset generally finished before 5 epochs, which took approximately 7-8 hours for a single GPU.

A.4 Gradient Scaling

In our dynamics, the error (ϵ_{n-1}) is defined as a scalar quantity whose gradient is taken with respect to the activation of the higher layer (e_n). That is,

$$\nabla \epsilon_{n-1} = \begin{bmatrix} \frac{\partial \epsilon_{n-1}}{\partial e_n^1} \\ \vdots \\ \frac{\partial \epsilon_{n-1}}{\partial e_n^L} \end{bmatrix} \tag{4}$$

where L denotes the number of elements in e_n. The partial derivative with respect to e_n^j can then be written as,

$$\frac{\partial \epsilon_{n-1}}{\partial e_n^j} = \frac{1}{K} \sum_i^K \frac{\partial (e_{n-1}^i - d_{n-1}^i)^2}{\partial e_n^j} \tag{5}$$

$$\tag{6}$$

where K is the number of elements in e_{n-1} (= channels x width x height). Equation 5 highlights how the dimensionality of the prediction (equivalently the error term) affects the gradients, scaling them down by a factor K.

This can be easily seen by supposing that the gradients with respect to e_n^j are i.i.d normally distributed around 0 with standard deviation σ,

$$\frac{\partial(e_{n-1}^i - d_{n-1}^i)^2}{\partial e_n^j} \sim \mathcal{N}(0, \sigma^2) \tag{7}$$

$$\sum_i^K \frac{\partial(e_{n-1}^i - d_{n-1}^i)^2}{\partial e_n^j} \sim \mathcal{N}(0, K\sigma^2) \tag{8}$$

Thus,

$$\frac{\partial \epsilon_{n-1}}{\partial e_n^j} = \frac{1}{K}\sum_i^K \frac{\partial(e_{n-1}^i - d_{n-1}^i)^2}{\partial e_n^j} \sim \mathcal{N}(0, \frac{\sigma^2}{K}) \tag{9}$$

This scaling is further troublesome in DCNs, where most gradients are zero since they are not part of the receptive field of the element e_n^j. Hence assuming that there are only C elements (kernel*channels) that are part of the receptive field of e_n^j,

$$\sum_i^K \frac{\partial(e_{n-1}^i - d_{n-1}^i)^2}{\partial e_n^j} = \sum_i^C \frac{\partial(e_{n-1}^i - d_{n-1}^i)^2}{\partial e_n^j} \sim \mathcal{N}(0, C\sigma^2) \tag{10}$$

Hence,

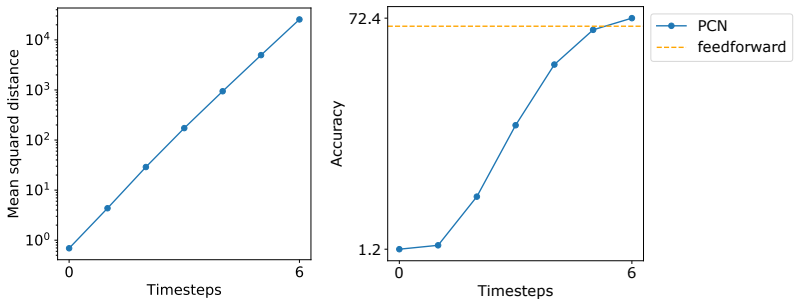
$$\frac{\partial \epsilon_{n-1}}{\partial e_n^j} = \frac{1}{K}\sum_i^C \frac{\partial(e_{n-1}^i - d_{n-1}^i)^2}{\partial e_n^j} \sim \mathcal{N}(0, \frac{C\sigma^2}{K^2}) \tag{11}$$

We use Equation 11 to, at least partly, counteract this effect due to the dimensionality of the prediction errors. We multiply the gradient by a factor of $\sqrt{K^2/C}$ to scale them in a way that is more comparable across layers, and thus apply a more meaningful step size for correcting the errors.

A.5 Prior work: PCNs

To better understand the model proposed by Wen et al. [22] and its differences to ours, we conducted several experiments using the code that they provided, and report here our most compelling observations. A first striking shortcoming was that the accuracy of their feedforward baseline was far from optimal. Using their code, with relatively minor tweaks to the learning rate schedule, we were able to bring it up from 60% to 70% – just a few percentage points below their recurrent network. We expect that this could be further improved with a more extensive and systematic hyperparameter search. In other words, their training hyperparameters appeared to have been optimised for their predictive coding network, but not – or not as much – for their feedforward baseline. We further found that a minor change to the architecture - using group normalisation layers after each ReLU – leads to a feedforward network which performs on par with the recurrent network, with a mean over 6 runs of 72% and best of 73%. Adding the same layers to the recurrent network did not lead to a corresponding improvement in accuracy.

We also found that the network had poor accuracy (underperforming the optimized feedforward baseline) until the final timestep, as can been seen in Figure S1b. This can be clarified by a closer reading of Figure 3 of their paper: the reported improvements over cycles from 60% at timestep 0 to more than 70% at timestep 6 are for seven distinct networks, each evaluated only at the timestep they were trained for. So in fact, in their model the predictive coding updates do not gradually improve on an already reasonably guess. This is clearly not biologically plausible: visual processing would be virtually useless if the correct interpretation of a scene only crystallised after a number of "timesteps". By the time a person has identified an object that object is likely to have disappeared or, in a worst



(a) Reconstruction error over timesteps

(b) Accuracy over timesteps on CIFAR100 testset

Figure S1: **PCN**: Panel (a) shows the reconstruction errors of the model over timesteps. It does not decrease over timesteps, as would be expected in a predictive coding system. Panel (b) depicts the accuracy of the model on the CIFAR100 test dataset. The model performs at chance level at early timesteps and then becomes better in the last few timesteps.

case scenario, eaten them. We also experimented with feeding the classification error at each timestep into an aggregate loss function, but this lead to a network which, while performing well, essentially did not improve over timesteps.

Figure S1a shows that the network does not uniformly minimise reconstruction errors over time for all layers, and thus is not performing correct predictive coding updates. In fact the total reconstruction error (across all layers) increases exponentially over timesteps. There are a number of possible explanations for this. Firstly, in the case of the network with untied weights, the authors choose to make a strong assumption in the update equations (seen as the equivalence of their Equations 5 and 6): that the feedback weights can be assumed to be the transpose of the feedforward weights, i.e. $W^b = (W^f)^T$. They thus propagate the feedforward error through the feedforward weights. However, it might be that the network learns feedback weights which essentially invert the feedforward transformation as assumed, but this is not guaranteed, and nor is it explicitly motivated through the classification loss function. Indeed, because the network is not motivated to learn a representation at earlier timesteps which produces a good prediction, it does not necessarily need to learn the inverse transformation: it can instead learn some other transformation which, when applied with the update equations, leads the network to *end up* in the right place. That being said, this assumption is valid for the network with tied weights, and this network also does not uniformly reduce reconstruction error over timesteps. Possibly, the presence of ReLU non-linearities means that the forward convolution may still not be perfectly invertible by a transposed convolution. Finally, in line with this unexpected increase of reconstruction errors over time, we have also failed to extract good image reconstructions from the network as seen in Figure 5 of their paper, although in private communication the authors indicated that this was possible with some other form of normalisation.

In short, while the ideas put forward in [22] share similarities with our own, their exact implementation did not support the claims of the authors, and the question of whether predictive coding can benefit deep neural networks remained an open one. We hope that our approach detailed in the present study can help resolve this question.

A.6 Comparing with Rao and Ballard

This section aims to start from the equations initially provided in Rao and Ballard [24] and compare them to ours. The parallels drawn will help to highlight the similarities and the differences between both the approaches.

Rao and Ballard consider a two-layer system, and start with the assumption that the brain possesses a set of internal causes, denoted as $\mathbf{r}$ (in matrix notation), that it uses to predict the visual stimulus, for example an input image $\mathbf{I}$, such that

$$\mathbf{I} \approx f(U\mathbf{r}) \tag{12}$$

where $f(.)$ is some nonlinear activation function. This $\mathbf{r}$ can be equalled to encoding layer e_1 in our equations, with $\mathbf{I}$ being the input image e_0 or its reconstruction d_0. U here, represents the top-down weight matrix (equivalent to $W^b_{1,0}$) that helps to make a prediction about the input image. That is,

$$\mathbf{I} \approx f(U\mathbf{r}) \equiv e_0 \approx d_0 = W^b_{1,0} e_1 \tag{13}$$

In this two-layer hierarchical architecture, $\mathbf{r}$ itself is predicted by the higher layer $\mathbf{r}^h$ using the weight matrix U^h, equivalent to how e_1 is predicted by e_2 using $W^b_{2,1}$ in our model. This prediction denoted as $\mathbf{r}^{td}$ in Rao and Ballard's original implementation can be equalled to d_1 in our equations.

$$\mathbf{r}^{td} = f(U^h \mathbf{r}^h) \equiv d_1 = W^b_{2,1} e_2 \tag{14}$$

The errors made in making the predictions are defined, like ours, as the mean squared distance,

$$\epsilon_0 = (\mathbf{I} - f(U\mathbf{r}))^T (\mathbf{I} - f(U\mathbf{r})) \tag{15}$$

$$\epsilon_1 = (\mathbf{r} - \mathbf{r}^{td})^T (\mathbf{r} - \mathbf{r}^{td}) \tag{16}$$

Please note that differentiating the prediction error ϵ_0 with respect to $\mathbf{r}$ (similar to taking the gradient of ϵ_{n-1} with respect to e_n as done in our error-correction term) gives us,

$$\nabla \epsilon_{\mathbf{0}} = -2U^T \frac{\partial f}{\partial U\mathbf{r}}^T (\mathbf{I} - fU\mathbf{r}) \tag{17}$$

$$= -kU^T (\mathbf{I} - f(U\mathbf{r})) \tag{18}$$

which will be useful later.

As per the predictive coding theory, the brain tries both to learn parameters (U and U^h) over a dataset of natural inputs, and tries to modify its neural activations ($\mathbf{r}$ and $\mathbf{r}^h$) over time given a particular input, in such a way as to minimize the total error E, defined as:

$$E = a \cdot \underbrace{(\mathbf{I} - f(U\mathbf{r}))^T (\mathbf{I} - f(U\mathbf{r}))}_{\epsilon_0} + b \cdot \underbrace{(\mathbf{r} - \mathbf{r}^{td})^T (\mathbf{r} - \mathbf{r}^{td})}_{\epsilon_1} \tag{19}$$

Here a and b act as constants that weigh the errors in this two-level hierarchichal network. Equation 19 is reflected as Equation 4 on Page 86 of the original paper [24]. The original implementation also contains terms that account for the prior probability distributions of $\mathbf{r}$ and U; these terms can be equated to regularization terms, and thus we omit them for the sake of simplicity.

Equation 19 represents the overall error, calculated as sum of the mean squared errors across the hierarchy of the network. It should be noted that we use this same objective function ($-E$) to train the feedback weights of our networks.

As stated above, the predictive coding dynamics aim to modify neural representations $\mathbf{r}$ so as to minimize the error E, i.e., differentiating the above equation:

$$\frac{d\mathbf{r}}{dt} = -\frac{\partial E}{\partial \mathbf{r}} = a \cdot U^T \frac{\partial f^T}{\partial U\mathbf{r}}(\mathbf{I} - f(U\mathbf{r})) + b \cdot (\mathbf{r}^{td} - \mathbf{r}) \tag{20}$$

Barring a regularization term, the above equation is equivalent to Equation 7 on page 86 of [24]. One can see that the first term in the RHS of equation 20 can be substituted with our error-correction term $\nabla\epsilon_0$ (see Eq. 18). Hence, Equation 20 after simultaneously expanding the LHS becomes,

$$\frac{\mathbf{r}(t+dt) - \mathbf{r}(t)}{dt} = -a_1 \cdot \nabla\epsilon_r(t) + b \cdot (\mathbf{r}^{td}(t) - \mathbf{r}(t)) \tag{21}$$

We use subscript r for ϵ to emphasize that this error can be calculated at any level/stage $\mathbf{r}$ represents in a multi-layer hierarchical system, and is not restricted to just the first layer of the hierarchy. Similarly, the time resolution dt can be equated to 1 timestep (of arbitrary duration) for simulations. Hence, rearranging the equation further,

$$\mathbf{r}(t+1) = \underbrace{b \cdot \mathbf{r}^{td}(t)}_{feedback} + \underbrace{(1-b)\mathbf{r}(t)}_{memory} - \underbrace{a_1 \nabla\epsilon_r(t)}_{error-correction} \tag{22}$$

In the above equation, the first term corresponds to our feedback term, the second term corresponds to our memory term and the last term corresponds to our feedforward error-correction term. That is, exchanging constants to match our notation:

$$\mathbf{r}(t+1) = \underbrace{\boxed{}}_{feedforward} + \underbrace{\lambda \cdot \mathbf{r}^{td}(t)}_{feedback} + \underbrace{(1-\lambda)\mathbf{r}(t)}_{memory} - \underbrace{a_1 \nabla\epsilon_r(t)}_{error-correction} \tag{23}$$

This can be directly compared to our main Equation 2.

Equation 23 also highlights the fact that our approach has an extra feedforward term that is not present in the original Rao and Ballard proposal. We believe that such a modification allows for rethinking the role of error-correction in network dynamics; where error-correction constituted the predominant mode of feed-forward communication in the Rao and Ballard implementation, it plays a more supporting role in our implementation, iteratively correcting the errors made by the feedforward convolutional layers. We empirically found that the feedforward term helped to improve the stability of the training. Interestingly, a common criticism of predictive coding lies in its inability to explain the dominance of feedforward brain activity compared to prediction error signals [17, 18]. We believe that our proposed implementation allows for a flexible modulation of these two terms, and thus systematic investigation of these factors–as done in [63].

From a practical perspective, we expect that our framework can be readily used by both proponents and opponents of the predictive coding theory. Setting the feedforward term β equal to zero produces a pure predictive coding network as proposed in Rao and Ballard [24]. Alternatively, one can set the error-correction term α equal to zero to study a bidirectional network with feedback and feedforward drives, in the style of Heeger [25]. The framework has been implemented such that the basic update rule (as class `Pcoder` in the package) is easily adaptable, allowing one to try other complex interactions between these terms; for example, one could easily include multiplicative interactions between feedback and feedforward terms to emulate forms of biased competition (see [40, 66]).

A.7 Tuning hyperparameters

In addition to the fixed set of hyperparameters used in our initial experiments (Figures 2, 3a and 4), we also experimented with optimizing our hyperparameters. To tune the hyperparameters for the models, we applied two different strategies for both the models–tuning hyperparameters for the whole network vs tuning hyperparameters for each pcoder separately. After a few initial explorations on clean images, we discovered that the hyperparameters dictate where the network dynamics converge, and consequently its performance for noisy situations. This effect is characterized and investigated thoroughly in [63]. Thus, in this study, we decide to use gaussian noise of standard deviation 0.5 to tune the hyperparameters and test it on all other types of noises from the ImageNet-C dataset.

For PVGG16, we start by fixing the value of alpha for each layer to zero and only search for β_n's and λ_n's. We calculate the average cross-entropy loss for 4 timesteps on 2000 images and use it as a metric for choosing the hyperparameters. The hyperparameters chosen are as follows :

Table S2: Values of the Hyperparameters

n	β_n	λ_n	α_n
1	0.2	0.05	0.01
2	0.4	0.10	0.01
3	0.4	0.10	0.01
4	0.5	0.10	0.01
5	0.6	0.00	0.01

For PEfficientNetB0, we take a different approach. Instead of the whole network, we start by finetuning each pcoder using the same metric (average crossentropy for 4 timesteps) on 4050 images. We then combine all hyperparameters found for each pcoder. The hyperparameters chosen are as follows :

Table S3: Values of the Hyperparameters

n	β_n	λ_n	α_n
1	0.77	0.08	0.01
2	0.76	0.11	0.01
3	0.83	0.03	0.01
4	0.94	0.01	0.01
5	0.73	0.25	0.01
6	0.81	0.01	0.01
7	0.85	0.10	0.01
8	1.0	0.00	0.01

We then, calculate the mCE scores using all the 19 noises for both the networks. The CE scores for each noise are shown below :

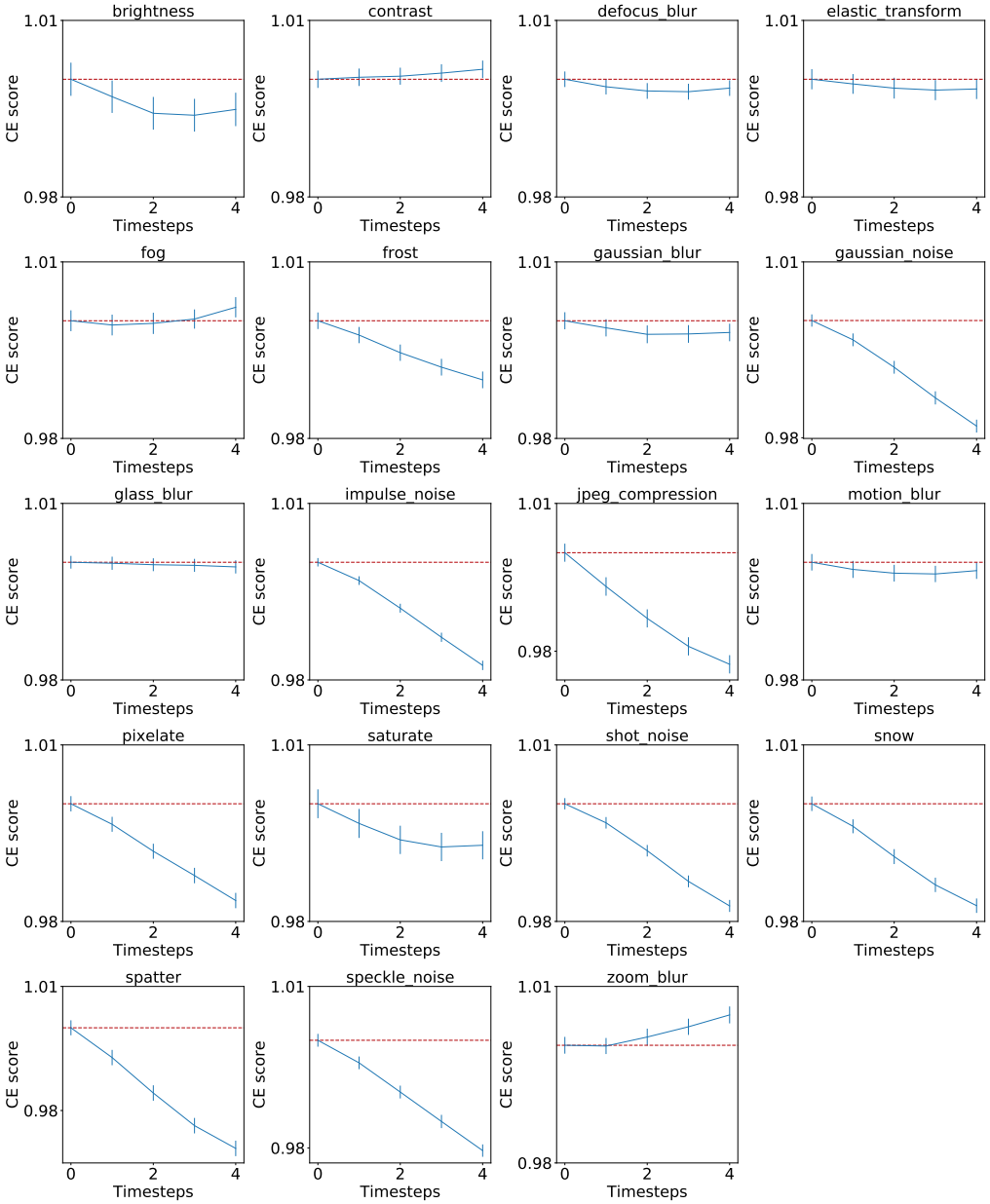


Figure S2: **PVGG16 (optimised) Corruption Error (CE) scores for all distortions:** The panel shows the CE scores calculated on the distorted images provided in the ImageNet-C dataset. The values are normalized with the CE score obtained for the feedforward VGG. The error bars denote the standard deviation of the means obtained from bootstrapping (resampling multiple binary populations across all severities.)

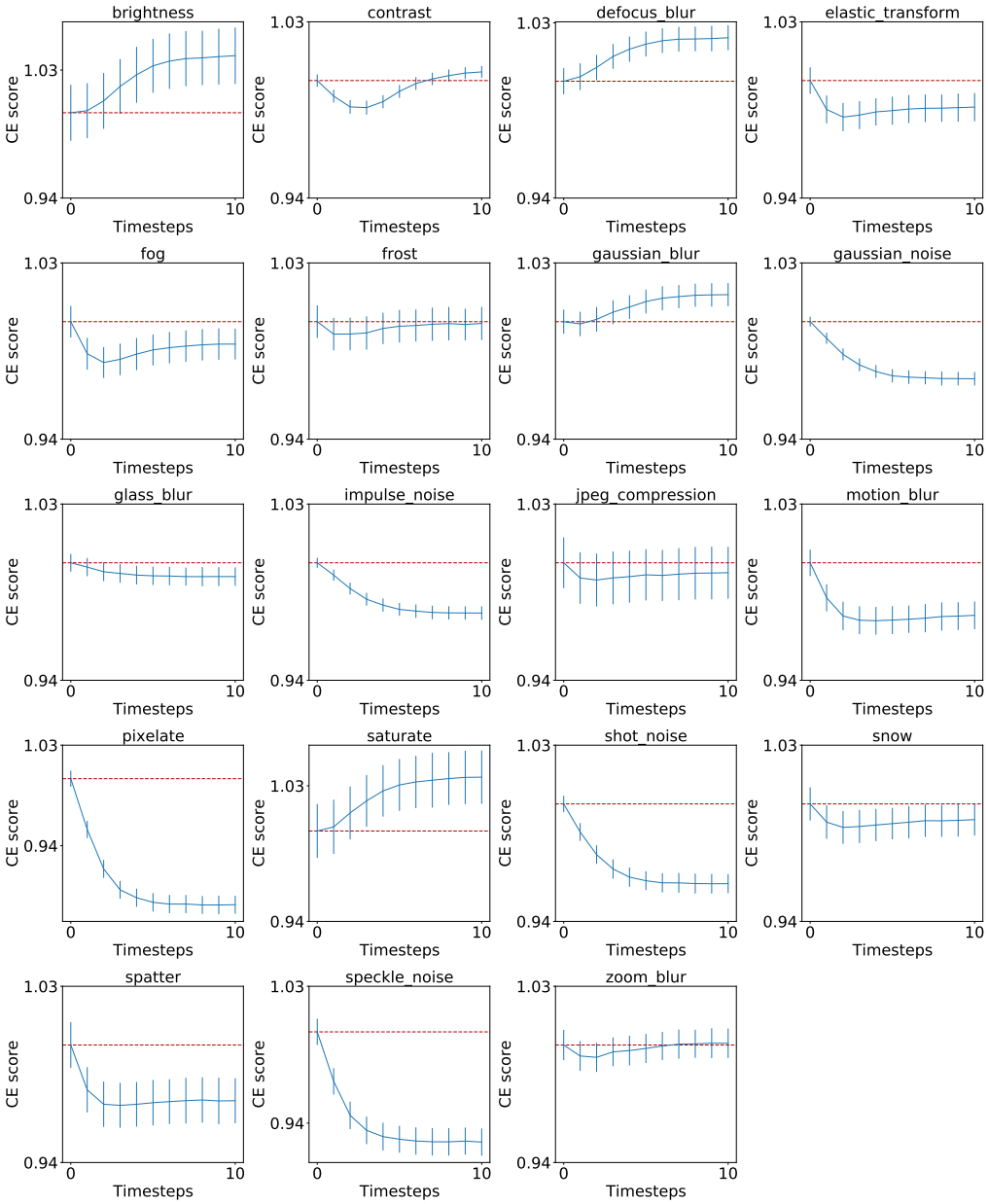


Figure S3: **PEfficientNetB0 (optimised) Corruption Error (CE) scores for all distortions:** The panel shows the CE scores calculated on the distorted images provided in the ImageNet-C dataset. The values are normalized with the CE score obtained for the feedforward EfficientNetB0. The error bars denote the standard deviation of the means obtained from bootstrapping (resampling multiple binary populations across all severities.)

A.8 mCE scores of the optimized networks using AlexNet as a baseline

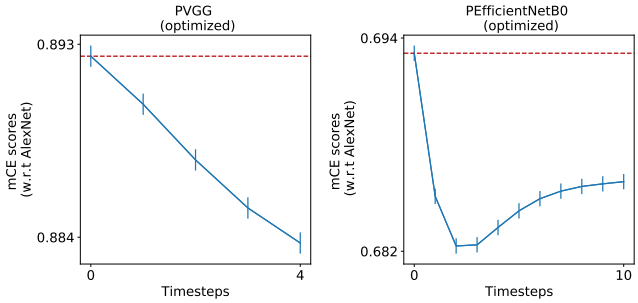


Figure S4: The mCE scores of the optimized networks (as shown in Figure 3) normalized using the score of the AlexNet network. Instead of normalizing using the score for the feedforward version of our recurrent network, to facilitate comparison with other works, we here normalize the scores using the score obtained for AlexNet network.

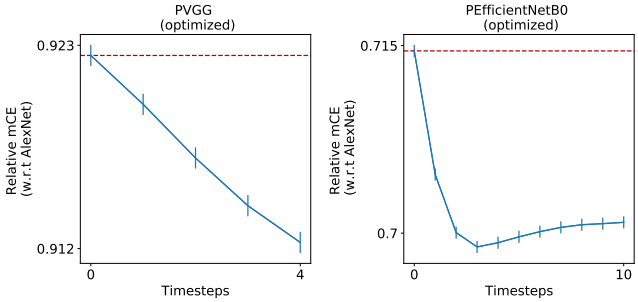


Figure S5: The Relative mCE scores of the optimized networks (as shown in Figure 3) normalized using the score of the AlexNet network. As suggested by [8], we use Relative mCE score which accounts for the changing baseline accuracy on the clean images over timesteps.

A.9 mCE scores of a predified robust network

We also incorporated our recurrent dynamics in an already robust PEfficientNet network. As a simple approach, we just used the hyperparameters (α, β and λ) that were optimized for the non-robust version of PEfficientNEtB0 (on 0.25 gaussian noise) and measured its robustness against the corruptions in ImageNet-C dataset. We observed that the proposed predictive coding dynamics further helped in improving the robustness of this already robust network.

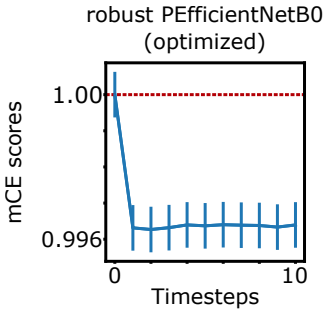


Figure S6: mCE scores of a predified version of an already robust PEfficientNetB0

A.10 Original data for Adversarial Attacks

We provide here the non-baseline corrected versions of the data presented for adversarial attacks in Figure 4. The panels below show the success rate of the targeted attacks across timesteps calculated on 1000 images. The perturbations allowed (ϵ) and the type of attack are denoted at the top.

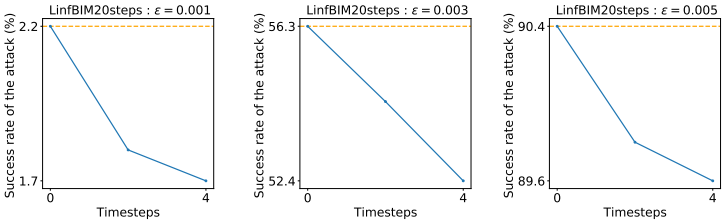


Figure S7: L_∞BIM attacks on PVGG16 network

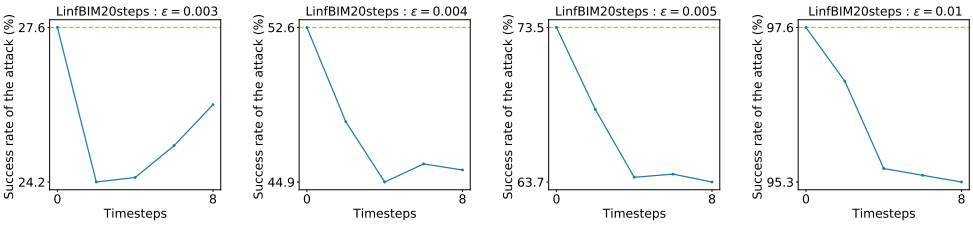


Figure S8: L_∞BIM attacks on PEfficientNetB0 network

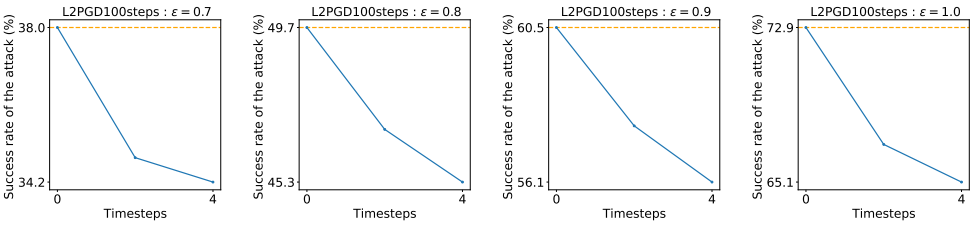


Figure S9: L_2RPGD attacks on PEfficientNetB0 network

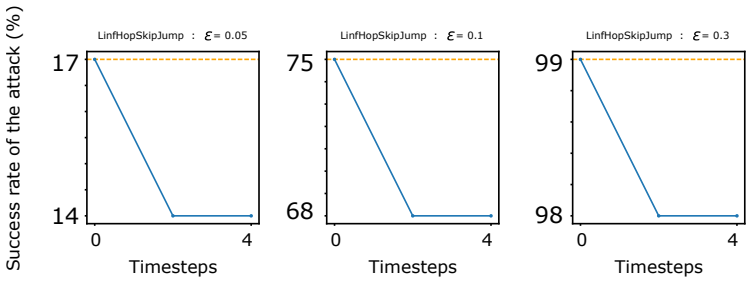


Figure S10: L_∞ HopSkipJump attacks on PEfficientNetB0

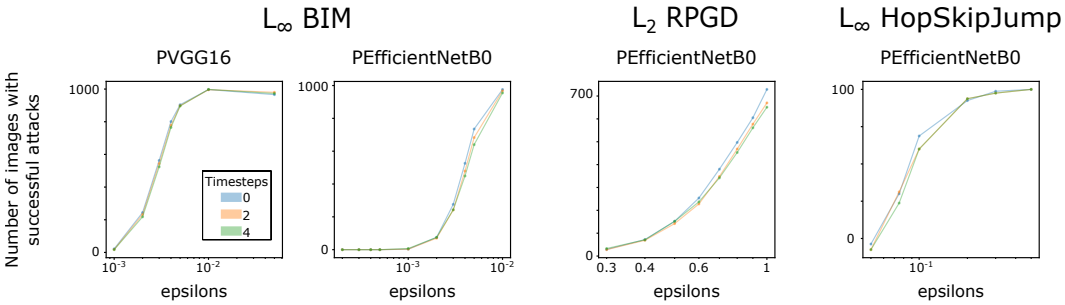


Figure S11: Adversarial Attacks with respect to epsilons. Here we show the number of successful attacks on 1000 (100 for HopSkipJump) images. Increasing the size of the epsilon leads to increase in the success rate of the attack as expected. As predictive coding timesteps increase, the curves shift slightly to the right, meaning that a slightly larger perturbation is required to fool the network. This robustness is more easily seen on Figure 4, where ϵ values are sampled near each curve's inflection point.

A.11 Absolute values of the plots shown in the main text

Noise Level	PVGG16		PEfficientNetB0	
	Accuracy at t=0	Accuracy at t=15	Accuracy at t=0	Accuracy at t=15
$\sigma = 0.00$	71.63	71.47	77.29	75.35
$\sigma = 0.50$	35.61	38.59	57.66	56.24
$\sigma = 0.75$	16.69	18.46	37.11	41.05
$\sigma = 1.00$	5.59	7.05	17.03	23.59

Table S4: Accuracy on gaussian noise-corrupted images. Here we show the accuracy obtained on images corrupted using gaussian noise (at t=0) as shown in figure 2a. All the values are calculated on the corrupted versions of the ImageNet validation dataset.

Noise Level	PVGG16		PEfficientNetB0	
	MSE at t=0	MSE at t=15	MSE at t=0	MSE at t=15
$\sigma = 0.00$	0.224	0.220	0.186	0.184
$\sigma = 0.25$	0.342	0.324	0.223	0.222
$\sigma = 0.50$	0.518	0.485	0.303	0.302
$\sigma = 0.75$	0.705	0.660	0.394	0.392
$\sigma = 1.00$	0.898	0.842	0.486	0.482
$\sigma = 2.00$	1.689	1.587	0.848	0.834

Table S5: MSE distances for reconstructions on noisy images. Here we show the MSE distances obtained between the noisy images corrupted using gaussian noises and the reconstructions made by the models as shown in Figure 2b.

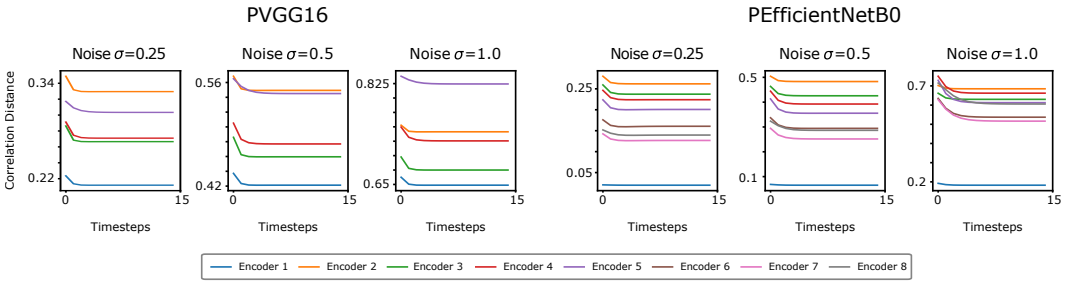


Figure S12: **Correlation distances for representations obtained on noisy images:** Here we show the absolute correlation distances obtained between clean and noisy representations as shown in Figure 2d in the main text.