Improving Anytime Prediction with Parallel Cascaded Networks and a Temporal-Difference Loss

Anonymous Author(s)
Affiliation
Address
email

Abstract

Although deep feedforward neural networks share some characteristics with the primate visual system, a key distinction is their dynamics. Deep nets typically operate in *serial* stages wherein each layer completes its computation before processing begins in subsequent layers. In contrast, biological systems have *cascaded* dynamics: information propagates from neurons at all layers in parallel but transmission occurs gradually over time, leading to speed-accuracy trade offs even in feedforward architectures. We explore the consequences of biologically inspired parallel hardware by constructing cascaded ResNets in which each residual block has propagation delays but all blocks update in parallel in a stateful manner. Because information transmitted through skip connections avoids delays, the functional depth of the architecture increases over time, yielding anytime predictions that improve with internal-processing time. We introduce a temporal-difference training loss that achieves a strictly superior speed-accuracy profile over standard losses and enables the cascaded architecture to outperform state-of-the-art anytime-prediction methods. The cascaded architecture has intriguing properties, including: it classifies typical instances more rapidly than atypical instances; it is more robust to both persistent and transient noise than is a conventional ResNet; and its time-varying output trace provides a signal that can be exploited to improve information processing and inference.

Since the earliest investigations of artificial neural nets, their design has been informed by biological neural nets [33]. Perhaps the most compelling example is the convolutional net for machine vision, which has adopted properties of primate cortical neuroanatomy including a hierarchical layered organization, local receptive fields, and spatial equivariance [11]. In this article, we investigate computational consequences of two fundamental properties of biological information processing systems that have not been considered in the design of deep neural nets. First, *the brain consists of massively parallel, dedicated hardware with neurons throughout the cortex updating continuously and simultaneously.* Second, *information transmission between neurons introduces time delays* [1]. As a result, unrefined and possibly incomplete neural state in one region is transmitted to the next region even as the state evolves; and even feedforward connectivity yields a speed-accuracy trade off in which the initial response to a static input occurs rapidly but can be inaccurate, with the output gradually improving over internal processing time. Following [32], we refer to such an architecture as *cascaded*.

Cascaded dynamics contrast sharply with the dynamics of standard feedforward nets, which operate in *serial* stages, each layer completing its computation before subsequent layers

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

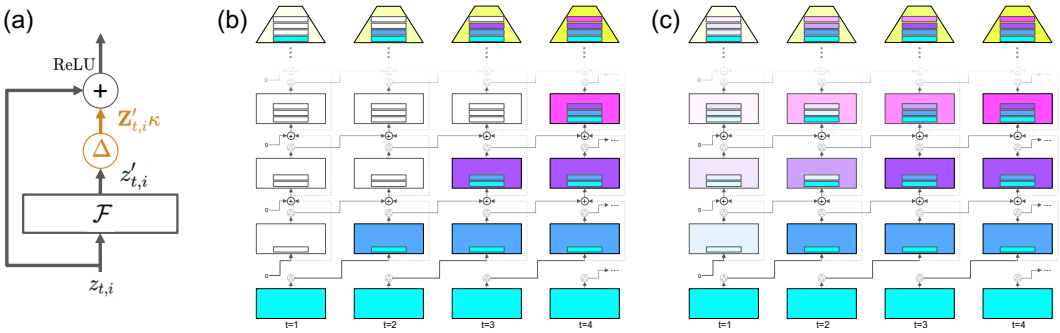

Figure 1: (a) ResNet building block, with additional delay component (Δ, in orange) that convolves a temporal kernel with the block output. Details in text. (b) Unrolling a standard *serial* ResNet in time (columns). Each rectangle is a ResNet block, which may consist of two or more convolutional layers. Blocks are updated sequentially. The color intensity (saturation) of a block indicates the extent to which a block has been completely activated. The narrow bars within each block signify the activation state of all blocks below that are contributing to the block's state (via skip connections). The yellow trapezoid at the top shows the layer information available at each time for classification. (c) Unrolling a *cascaded* ResNet. Note partial activations propagating through architecture due to parallel updates of all layers.

begin processing. Cascaded dynamics are also quite different than the dynamics of vision models with recurrent connections [e.g., 22, 24, 34, 42], which, given a static input, may iteratively update, but layer updates are still computed serially with each layer completing its computation and then feeding it immediately to the next layer (or back to itself). Fundamentally, our investigation asks: Supposing we take a step toward biological realism with massively parallel hardware and relatively slow inter-neuron communication, what are the computational benefits and consequences?

We construct cascaded networks by introducing propagation delays in deep feedforward nets provided with a static input. We treat the net as massively parallel such that all units across all layers are updated simultaneously and iteratively. We focus on the ResNet architecture [13] and we introduce a propagation delay into each residual block (Figure 1a). Because the skip connection permits faster transmission of more primitive perceptual representations, the functional depth of the resulting architecture increases over internal-processing time, yielding a trade off between processing speed and complexity of processing. Consequently, the architecture offers a natural, integral mechanism for *anytime prediction* [53]. Speed-accuracy trade offs are a fundamental characteristic of human information processing [21, 38] and human perception has been modeled with deep learning anytime prediction methods [26].

Although we focus on the ResNet, our approach can be incorporated into any model with skip connections (e.g., Highway Nets [43], DenseNet [18], U-Net [39], Transformers [47]). The contrast between a *serial*, one-layer-at-a-time model and a *cascaded*, parallel-update model is illustrated in Figures 1b and 1c, respectively. To step through the operation of the cascaded model, at time 1, only the first residual block has received meaningful input, and the model prediction is therefore based only on this block's computation. At time 2, *all* higher residual blocks have received input from block 1, and the output is therefore based on *all* blocks' computations, though blocks 2 and above have deficient input. At each subsequent time, all blocks are receiving meaningful input, but it is not until time t that block t has reached its asymptotic output because its input does not stabilize until $t-1$. In essence, the cascaded model behaves like a WideResNet [51] on the first steps and then becomes a deep ResNet.

Our work makes the following key contributions.

- We demonstrate the superiority of the cascaded architecture to the serial (Figures 1b,c), indicating that parallelism can be exploited in a way that has not previously been studied.
- We propose and evaluate a novel training objective aimed at improving the predictions of anytime models. This *temporal-difference (TD) loss* [44] encourages the most accurate response as quickly as possible. TD training improves the performance of both cascaded and serial architectures. Although a rich literature exists aimed at reducing the number

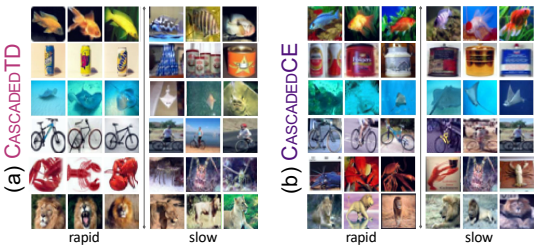

Figure 2: (a) Three CIFAR-100 instances categorized rapidly (left) and slowly (right) by a cascaded model trained using a TD loss. (b) same as (a) for a standard cross-entropy loss. When trained with the TD loss, the cascaded model stratifies instances by typicality, with rapid processing of prototypical views on a homogenenous background and slow processing of unusual and cluttered views.

of computational steps required to obtain an accurate answer [2, 3, 4, 5, 10, 14, 15, 16, 17, 19, 23, 28, 34, 36, 40, 46, 49, 52], all of this work uses a degenerate form of TD for training and our results suggest that these models can be improved using TD.

- The cascaded model trained with TD (CascadedTD) tends to respond most rapidly to prototypical exemplars, whereas training with the standard cross-entropy loss (CascadedCE) does not (Figure 2). We assess with three quantitative prototypicality measures, and we further show that CascadedTD rapidly homes in on the correct semantic family, whereas CascadedCE does not. These facts indicate that CascadedTD organizes knowledge differently across layers than does CascadedCE.
- We show that CascadedTD obtains a strictly superior speed-accuracy profile compared to previously proposed anytime prediction models, which are all based on a serial architecture.
- We demonstrate other virtues of CascadedTD: it is more robust to input noise, and its time-varying output trace provides useful signals for *meta-cognitive* processes—separately trained nets that make judgments about the cascaded architecture's accuracy.

Related Work

Prior research on cascaded models. From a psychological perspective, McClelland [32] characterizes human mental computation in terms of a hierarchy of leaky integrators that continually transmit partial information as it becomes available. We are aware of no work in deep learning on static image processing with cascaded models, but there exist two investigations focused on video sequence processing where the model state from the previous frame is used to efficiently process the next. Fischer et al. [10] present a *streaming rollout* framework for recurrent nets and they very briefly explore the temporal dynamics of cascaded models, showing benefits to early predictions. Carreira et al. [5] present a causal video understanding model that performs depth-parallel computation with the objective of improving video processing efficiency via the maximizing throughput, minimizing latency, and reducing clock cycles.

Recurrent nets for vision. Recurrent nets have been used in vision [e.g., 19, 22, 24, 42, 41], which adds a dimension of internal processing time for every external input (see also [12]). However, these models perform serial layerwise updates and therefore fundamentally differ in their operation from cascaded models.

Anytime prediction. Anytime prediction models [2, 8, 14, 15, 16, 17, 19, 23, 27, 28, 30, 34, 36, 40, 46, 48, 49, 52] assume serial operation of layers but allow for predictions to be made from intermediate layers of the architecture. In the simplest case, after t steps, t layers have been activated, and at each step, a prediction is made from the last activated layer [e.g., 15, 23]. Figure 1b illustrates a serial model that performs anytime prediction. Some of these models have intrinsic stopping criteria [e.g., 6]; others rely on selection of a stopping confidence threshold [e.g., 23, 46, 49].

Temporal-difference learning. TD learning has a rich history, mostly in the RL community for value function estimation. TD learning can be used for supervised training as well, and in fact the two previous works in deep learning using cascaded models [5, 10] perform supervised training to attain the correct model classification at each step. The form of training is equivalent to a degenerate case of temporal difference learning, TD(1), which we show to have inferior performance. A variety of non-cascaded models, both recurrent [19, 34, 52] and feedforward [2, 3, 4, 14, 15, 16, 17, 23, 28, 36, 40, 46, 49], aim to reduce the number of computational steps required to obtain an accurate output. All use TD(1) for

training. No previous research has explored the general formulation of TD for improving anytime prediction.

Deep Cascaded Networks

Many modern deep architectures—including ResNet [13], Highway Nets [43], DenseNet [18], U-Net [39]—incorporate skip connections that bypass strictly layered feedforward connectivity, analogous to the architecture of visual cortex [9]. Under the biological assumption that signals transmitted through a neural layer are delayed relative to signals that bypass the layer, we construct a cascaded model using ResNets by introducing a novel computational component that delays the transmission of signals from the output of each computational layer, denoted Δ in Figure 1a. Because these delays extend processing in time, the hidden states require a time index. The input to ResNet block i at time t is denoted $\boldsymbol{z}_{t,i}$. The block transforms this input via the residual transform, yielding $\boldsymbol{z}'_{t,i} = \mathcal{F}(\boldsymbol{z}_{t,i})$. We conceive of Δ as a tapped delay-line memory of the transform history, $\boldsymbol{Z}'_{t,i} = [\boldsymbol{z}'_{t,i}\ \boldsymbol{z}'_{t-1,i}\ \ldots\ \boldsymbol{z}'_{1,i}]$, which is convolved with a temporal kernel $\boldsymbol{\kappa}$ to produce the block output

$$\boldsymbol{z}_{t,i+1} = \mathrm{ReLU}\left(\boldsymbol{z}_{t,i} + \boldsymbol{Z}'_{t,i}\boldsymbol{\kappa}\right). \tag{1}$$

The kernel $\boldsymbol{\kappa} = [1\ 0\ 0 \ldots 0]$ recovers the standard ResNet in which communication between layers is instantaneous. We consider two kernels to introduce time delays. With $\boldsymbol{\kappa} = [0\ \ 1\ \ 0\ \ 0\ \ \ldots 0]$, a discrete one-step delay is introduced (*OSD* for short). With $\boldsymbol{\kappa} = (1-\alpha)[1\ \ \alpha\ \ \alpha^2\ \ \alpha^3 \ldots]$, we obtain exponentially weighted smoothing (*EWS* for short), with larger $\alpha \in [0, 1)$ producing slower transmission times. Note that both of these special kernels have efficient implementations: the OSD kernel with a one-element queue, the EWS kernel with incremental update and a finite (one-step) state vector,

$$\boldsymbol{Z}'_{t,i}\boldsymbol{\kappa} = \alpha \boldsymbol{Z}'_{t-1,i}\boldsymbol{\kappa} + (1-\alpha)\boldsymbol{z}'_{t,i}.$$

We use the OSD kernel for training all models. Modifications of batch norm were required to do time-step-conditional normalization. All experiments use a ResNet-18, which has 8 residual blocks and hence 8 time delays. (We have done experiments with larger ResNets and the additional compute did not affect qualitative properties.) We also add a time delay to the output of the model's first convolutional layer. Consequently, with the OSD kernel, the cascaded model requires 9 updates for the output to reach asymptote. The cascaded and serial models with the same weights will necessarily produce identical asymptotic outputs.

To obtain a finer temporal granularity at evaluation, some simulations switch to the EWS kernel with $\alpha = 0.9$. Temporal dynamics are qualitatively similar for OSD and EWS, but EWS allows us to better distinguish individual examples in terms of their fine-grain timing. EWS with $\alpha = 0.9$ requires about 70 steps for the output to asymptote. The choice of α over a had no impact on our findings, as long as it slowed transmission.

Training Cascaded Networks with TD(λ)

To allow for anytime prediction, we include an output head following each of the T residual blocks in both the serial and cascaded models (see Figure 1b,c, respectively). The output heads may share weights or have separate weights. To encourage correct outputs sooner, we use temporal difference (TD) learning [44] over the output sequence. Readers may associate TD methods with reinforcement learning because TD methods have traditionally been used to predict future rewards. However, TD methods are fundamentally designed for supervised learning. We use TD to predict a future outcome—the correct classification of an image—from a sequence of successively more informative states—the information flowing through the ResNet at each internal time step.

TD(λ) specifies a target output y_t at each time $t \in \{1, ...T\}$ based on the model's actual output $\hat{y}_{t+i}$ at future times $t+i$ for $i > 0$, and the eventual outcome or true target, y_{true}:

$$y_t = (1-\lambda)\left[\sum_{i=1}^{T-t} \lambda^{i-1}\hat{y}_{t+i}\right] + \lambda^{T-t} y_{\text{true}}, \tag{2}$$

where $\lambda \in [0, 1]$ is a free parameter that essentially specifies the time horizon for prediction. TD(1) predicts the eventual outcome at each step; TD(0) predicts the model's output at the

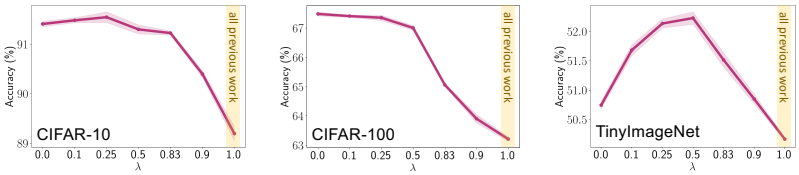

Figure 3: Effect of TD hyperparameter λ on CascadedTD test accuracy for three data sets. $\lambda = 1$ corresponds to the training methodology of all past research on anytime prediction, which is inferior to any $\lambda < 1$ for all data sets. Shaded error bands—hard to see on most curves—indicate ± 1 SEM, corrected to remove performance variance due to weight initialization and unrelated to λ [31].

next step (and the eventual outcome at the final step). Given target y_t and actual output $\hat{y}_t$, we specify a cross-entropy loss, $\mathcal{L} = \sum_{t=0}^{T} H(y_t, \hat{y}_t)$, where $H(p, q)$ is the cross-entropy. Note that y_t must be treated as a constant, not as a differentiable variable, via a `stop_gradient` (for TensorFlow or Jax) or `requires_grad=False` (for PyTorch). Although Equation 2 requires knowledge of all subsequent network states, the beauty of TD methods is that this loss can be computed incrementally (see Appendix). The edge cases TD(0) and TD(1) have particularly trivial implementations. Past research has always used TD(1) for specifying intermediate targets, but we will show that TD(1) is suboptimal because the model is penalized for being unable to classify correctly at the earliest steps.

Results

TD(λ) Training

We conducted a sweep over hyperparameter λ to determine its effect on asymptotic accuracy of CascadedTD. Figure 3 shows results from five replications of CascadedTD on CIFAR-100, CIFAR-10, and TinyImageNet. The hyperparameter has a systematic effect, consistent with classic studies with linear models [45, Chapter 12]. Importantly, $\lambda = 1$, which is the implicit choice of *every* previous anytime-prediction model [2, 8, 14, 15, 16, 17, 19, 23, 27, 28, 30, 34, 36, 40, 46, 48, 49, 52], obtains the poorest performance for all three data sets, significantly worse than $\lambda \approx .5$. The essential explanation is that larger λ penalize the network for behavior it does not have the capability to achieve: obtaining the asymptotic prediction at the earliest time steps. To paraphrase the classic illustration of TD from Sutton [44], if the task is predicting the weather on December 31, no model can predict as accurately on December 1 as on December 30. Selecting $\lambda < 1$ shortens the prediction horizon; $\lambda = 0$ corresponds with requiring a prediction only of the weather on the next day.

In the rest of the article, we report results for CascadedTD with $\lambda = 0$, or *TD(0)*. Although TD(0) is not optimal for all data sets, it is strictly superior to TD(1), has a trivial implementation (no need for eligibility traces), and does not risk overfitting to our test set.

Anytime Prediction and Speed-Accuracy Trade Offs

Given a static input, an anytime prediction model attempts to obtain the best classification possible as quickly as possible. Anytime prediction can be performed by both serial and cascaded models. Both yield predictions at each time slice, as depicted by the yellow trapezoids in Figures 1b,c, which denote model readout. Critical to anytime prediction is deciding when to terminate processing and initiate a response [23, 6, 46]. Following [23] and [46], we assume that processing terminates when the confidence (probability) for the most likely class rises above threshold θ. For any θ, one can measure the mean stopping time and the mean accuracy for all instances in a test set. By sweeping $\theta \in [0, 1]$ and plotting mean accuracy as a function of mean stopping time, one obtains a *speed-accuracy trade off* curve. Figure 4 shows curves for models we'll describe next.

To evaluate the cascaded model, we compare to a recent state-of-the-art method, the *Shallow-Deep Network (SDN)* [23]. The SDN has the serial architecture depicted in Figure 1b. One critical design decision was whether to have separate read-out heads at each step (MultiHead) or a shared read-out head (SingleHead), i.e., whether weights are separate or shared. From the perspective of the cascaded model, which considers the vertical columns of Figure 1c to be copies of a network unrolled in time, the SingleHead approach is natural. The SDN, as a serial model, chose the MultiHead approach. We tested all four logical

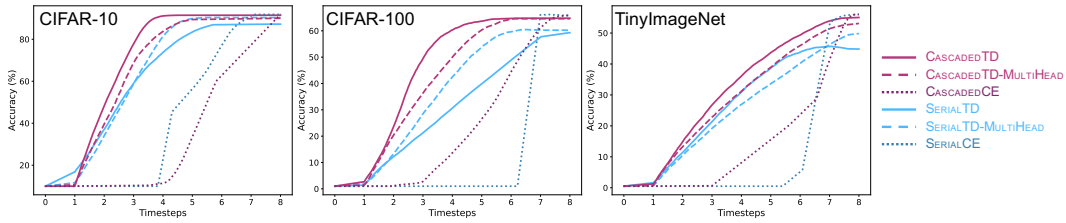

Figure 4: Speed accuracy trade off for three data sets and six models, obtained by varying a stopping threshold and measuring mean latency and mean accuracy. CascadedTD is our parallel anytime prediction model; SerialTD-MultiHead is the state-of-the-art method, SDN [23].

combinations of {SerialTD, CascadedTD} × {SingleHead, MultiHead}. We use SerialTD and CascadedTD as shorthand for the SingleHead variants, and append MultiHead to the model name for that version. Additionally, we consider SerialCE and CascadedCE, which are SingleHead variants trained with the standard cross entropy loss that penalizes only asymptotic accuracy and does not explicitly attempt to obtain a speeded response.

Let's step through key observations from Figure 4, which shows speed-accuracy trade offs for the six models on three data sets. First, our canonical cascaded model, CascadedTD, obtains better anytime prediction than SerialTD-MultiHead (i.e., the architecture of SDN). CascadedTD also achieves higher asymptotic accuracy; its accuracy matches that of CascadedCE, a ResNet trained in the standard manner. Thus, cascaded models can exploit parallelism to obtain computational benefits in speeded perception without costs in accuracy.[1] Second, while MultiHead is superior to SingleHeadfor serial models, the reverse is true for cascaded models. This finding is consistent with the cascaded architecture's perspective on anytime prediction as unrolled iterative estimation, rather than, as cast in SDN, as distinct read out heads from different layers of the network. Third, models trained with TD outperform models trained with standard cross-entropy loss. Training for speeded responses reorganizes knowledge in the network so that earlier layers are more effective in classifying instances. We now turn to better understanding what this reorganization entails.

Organization of Knowledge in TD-Trained Cascaded Model

Having examined the response profile of our models over an evaluation set, we now turn to analyzing the response to individual instances. Specifically, we ask about the time course of reaching a classification decision. We define the *selection latency* for an instance to be the minimum number of steps required to reach a confidence threshold on one class and maintain that level going forward, i.e., $\min\{t \mid [\exists j \mid \hat{y}_{t',j} > \theta \ \forall \ t' \geq t]\}$, where $\hat{y}$ is the model output, j is an index over classes, and θ is the threshold. The selection latency does not specify whether or not the chosen class is correct. We picked a threshold $\theta = 0.83$ for CascadedTD such that only $\sim 10\%$ of the test examples failed to reach threshold; results that follow are robust to this selection.

What determines an instance's selection latency? Figure 2 presents instances that have the lowest and highest latency—labeled 'rapid' and 'slow', respectively. For CascadedTD (Figure 2a), notice the homogeneity of the rapid images: the objects are viewed from a canonical perspective and lie against a solid background with no clutter in the image; in contrast, the slow images are more varied, both in the object's instantiation in the image and the background complexity. Turning to CascadedCE (Figure 2b), instances do not appear to stratify by prototypicality. In the rest of this section, we formalize the notion of prototypicality with three measures and compute the correlation of each measure with selection latency for the cascaded model trained with a TD loss (CascadedTD) and with the standard cross-entropy loss (CascadedCE). Our three measures are as follows.

- *Centrality.* We compute the cosine distance of an instance's embedding (the penultimate layer activation) and the target-class weight vector. The larger this quantity, the better aligned the two vectors are. Because the weight vector will tend to point near the center of class instances, the cosine distance is a measure of instance centrality.

[1] We used $\lambda = 1$ to train SerialTD and SerialTD-MultiHead, as was done for *all* previously proposed serial models. Could SDN and other serial models be improved with $\lambda < 1$? In the Appendix, we show that the serial model with $\lambda = 0$ still does not perform as well as CascadedTD.

Table 1: Spearman rank correlations between three measures of instance prototypicality and selection latency. Larger coefficient indicates faster responses for more prototypical instances.

Measure	Spearman's ρ	
	CascadedCE	CascadedTD
centrality	0.140	0.352
C-score	0.326	0.489
human consistency	0.153	0.142

- *C-score.* Jiang et al. [20] describe an instance-based measure of statistical regularity called the C-score. The C-score is an empirical estimate of the probability that a network will generalize correctly to an instance if it is held out from the training set. It reflects statistical regularity in that an instance similar to many other instances in the training set should have a high C-score.
- *Human labeling consistency.* Peterson et al. [37] collected human labels on images. Most images are consistently labeled, but some are ambiguous. The negative entropy of the response distribution indicates inter-human labeling agreement. Presumably consistently labeled instances are more prototypical.

All three measures are available only for the CIFAR-10 training set. Consequently, we ran 10-fold cross validation on the training set, assessing the correlation based on the held out images in each fold. To obtain a granular selection latency, we use the EWS kernel.

Table 1 presents the correlation—Spearman's ρ—between the three prototypicality measures and negative selection latency for CascadedCE and CascadedTD. A positive coefficient indicates shorter latency for prototypical instances. The coefficient is reliably positive ($p < .001$) for each of the three typicality measures and both models. However, CascadedTD obtains reliably higher correlations on two of the three measures than CascadedCE ($p < .001$). They are not significantly different on the consistency measure ($p = .29$). Thus, by these quantitative scores, the TD training procedure leads to better stratification of instances by typicality, in line with the qualitative results presented in Figure 2. Why does TD training distinguish instances based on prototypicality? Intuitively, a prototypical instance shares features with many other class instances. Because these features are frequent in the data set, the TD loss focuses on rapidly classifying instances with those features.

Beyond investigating the time course of fine-grain classification, we also examined coarse-grain classification. Forming twenty superclasses from the 100 fine-grain classes of CIFAR-100, as specified in [25], we examined the probability of correct coarse-grain classification conditional on incorrect fine-grain classification. Zamir et al. [52] refer to this probability as *taxonomic compliance*, which reflects information being transmitted about coarse category even when the specific class cannot be determined. As shown in Figure 5, taxonomic compliance rises faster for CascadedTD than for CascadedCE. Whereas chance compliance is .05, CascadedTD achieves a compliance probability of .35 after 2 steps. CascadedCE requires 8 steps to achieve the same performance. TD training pushes instances to the correct semantic neighborhood sooner, even when not to the correct class label. This result further supports the reorganization of knowledge for more robust behavior.

Robustness to Input Noise

In previous simulations, we have assumed the input was static while internal processing took place. Now we consider static inputs with time-varying noise. Figure 6 shows four types of lossy noise we consider on CIFAR-10 images. The noise types are: (1) *Focus*: a 16×16

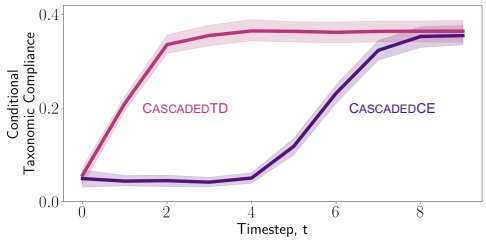

Figure 5: CascadedTD performs coarse-grain classification before fine-train classification, as assessed by a measure of conditional taxonomic compliance [52]. Graphs are based on 5 runs of each model with different random initializations. Shaded error bands indicate ±1 SEM, corrected to remove performance variance due to initial seed [31].

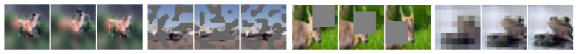

Figure 6: Lossy noise types. From left to right: *Focus*, *Perlin*, *Occlusion*, *Resolution*.

foveated patch randomly placed within the image, where regions outside of the patch are Gaussian blurred; (2) *Perlin*: gradient noise randomly applied to 40% of image pixels; (3) *Occlusion*: a 16×16 occluding patch randomly placed within the image; (4) *Resolution*: random downsampling by factors of $0\times$, $2\times$, or $4\times$ via average pooling followed by k-nearest upsampling to recover the original dimensionality of 32×32.

For each noise type, CascadedCE and CascadedTD models are trained with the corresponding image transformation type as a data augmentation. Training details can be found in the Appendix. Because the external environment changes more rapidly than any snapshot of the environment can be processed, cascaded models will necessarily integrate signals from multiple snapshots. To determine whether signal integration is beneficial for noise suppression, we compared to a serial model that is allowed to fully process each snapshot, which for the architecture requires $9\times$ as many sequential updates as the cascaded model. We therefore refer to the model as SerialCE$\times$9; it has the same weights as CascadedCE.

Two noise variants are applied to test images: *persistent*, in which a noise sample is drawn independently at each update step, and *transient*, in which the model reaches its asymptotic output on a noise-free input, input is corrupted by noise samples for a variable number of steps, and then the noise-free is presented input until the model returns to its previous asymptotic output. For persistent noise, we assess with asymptotic accuracy; for transient noise, we assess with a measure of *drop in integrated performance* over the course of the episode, which indicates how quickly the model recovers from noise perturbation (smaller is better). Simulation details in the Appendix.

Table 2 indicates that CascadedTD obtains a degree of robustness to persistent and transient noise not matched by the alternative models. Although SerialCE$\times$9 performs a full inference pass on each noise perturbation, the stateful nature of CascadedTD allows it to smooth out noise via slow integration. Although CascadedCE shares the same architecture as CascadedTD, TD training is required to orchestrate the integration of content-specific perceptual information. This experiment indicates that laggy information processing in the cascaded model can be advantageous in a noisy environment. In the Appendix, we note that the benefit for CascadedTD applies only to lossy noise, not translations and rotations.

Meta-cognitive Inference

In this section, we consider the hypothesis that temporally intermediate outputs from cascaded networks can provide additional signals to improve performance. We term this *metacognition*, by reference to human abilities to reason about our reasoning processes.

The temporal trace of output from CascadedTD is provided to a separate classifier, MetaCog, which is *discriminatively* trained for OOD detection. MetaCog is a fully connected feedforward net with a 256-unit hidden layer and a sigmoidal output unit for binary prediction: 1 or 0 for in- or out-of-distribution instances, respectively. CIFAR-10's validation set serves as the in-distribution training examples, whereas the validation sets of TinyImageNet, LSUN, and SVHN serve as OOD training examples; see details in Appendix. The CascadedTD output is represented in one of four ways as input to MetaCog: (1) the confidence of the

Table 2: Experiments on persistent and transient input noise. Highlight indicates best performance.

	Persistent Noise *Asymptotic Accuracy (%)*			**Transient Noise** *Drop in Integrated Performance*		
Noise	SerialCE$\times$9	CascadedCE	CascadedTD	SerialCE$\times$9	CascadedCE	CascadedTD
Focus	84.27 ± 0.06	83.75 ± 0.10	87.31 ± 0.04	0.62 ± 0.04	0.66 ± 0.05	0.00 ± 0.01
Occlusion	86.26 ± 0.08	82.73 ± 0.09	89.76 ± 0.05	7.70 ± 0.55	8.25 ± 0.72	0.93 ± 0.15
Perlin	85.18 ± 0.03	84.56 ± 0.05	87.67 ± 0.08	0.86 ± 0.06	0.87 ± 0.07	0.00 ± 0.01
Resolution	84.53 ± 0.07	85.40 ± 0.07	88.19 ± 0.10	0.81 ± 0.06	0.53 ± 0.05	0.18 ± 0.02

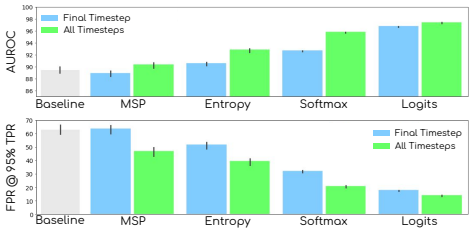

Figure 7: The output of CascadedTD over time provides a signal for significantly improving OOD detection over just using its asymptotic output. Larger AUROC and smaller FPR are better. All output representations benefit from the temporal trajectory. Error bars reflect ± 1 SEM corrected to remove comparison-unrelated variance [31]. Baseline comes directly from the final max softmax prediction.

most probable class, known as the *max softmax prediction* (*MSP*), (2) entropy of the class posterior distribution, (3) the class posterior distribution, and (4) the logit representation of the posterior. We investigate whether feeding the output of all time steps to MetaCog leads to improved prediction relative to feeding only the final asymptotic output. Only the latter information is available in a standard feedforward net.

Following [29], we assess OOD performance with *AUROC*, the area under the ROC curve, and *FPR @ 95% TPR*, the false positive rate at 95% true positive rate. A baseline metric is computed directly from the final max softmax predictions of the CascadedTD model, whereas the other metrics are based on the MetaCog model output. Figure 7 indicates that the temporal output trajectory of CascadedTD provides a valuable signal for OOD detection. Our goal here was not to propose a method for OOD detection, but merely to demonstrate that in principle, there is information about the input that is conveyed by the cascaded model's dynamics but that is not available in a traditional classifier's output.

Discussion

We investigated a neglected biologically-motivated architecture in which transmission delays are the bottleneck in neural information processing, not the number of neurons that can update in parallel. We proposed a temporal-difference (TD) loss that yields improved speed-accuracy trade offs. We showed that this model beats the state-of-the-art anytime prediction method, partly because of the TD loss and partly because of the model dynamics. The cascaded model has many distinctive properties, including: it classifies prototypical instances more rapidly than outliers; it performs coarse-to-fine semantic processing in which general semantic categories are rapidly inferred even if specific class labels are not; it is able to moderate time-varying input noise; and the temporal trace of the model's output provides an additional signal that can be exploited to improve information processing beyond that provided by the asymptotic model output. Of course, these interesting properties come at a computational cost when parallel hardware is simulated on existing compute infrastructure. We see three directions in which cascaded nets have particular potential.

- For neuroscientists using deep nets as a model of human vision, cascaded nets are a better approximation to the dynamics of the neural hardware. The properties we investigate—neurons operate in parallel, neurons are stateful, and neurons are slow to transmit information—seem likely to have a critical impact on the nature of cortical computing. As one simple illustration, cortical feedback processes are often posited to be critical for explaining difference in processing efficiency of visual stimuli [e.g., 22, 42]. We have shown that these difference might be partly explained by feedforward cascaded dynamics.
- For hardware researchers, cascaded networks are a possible direction for the future design of AI hardware. It is a direction quite unlike modern GPUs and TPUs, one that exploits massively parallel albeit slow and possibly noisy information processing. Our success in showing strong performance from cascaded models, as well as a training procedure to obtain quick and accurate responses, should encourage researchers in this direction.
- For AI research in anytime prediction, we've shown that existing models can be improved with a TD(λ) loss; all past research has adopted $\lambda = 1$, which we show to be inferior to $\lambda < 1$. For researchers who care little about cascaded models per se, cascaded models offer an intriguing method to train serial feedforward models. One can take a serial feedforward model, turn it into a cascaded model for training with TD methods, and then run it in serial mode. We've shown that TD training can improve asymptotic model accuracy while still providing anytime predictions due to inductive biases it imposes on the organization of representations.

References

[1] W. Bialek and F. Rieke. Reliability and information transmission in spiking neurons. *Trends in Neurosciences*, 15(11):428–434, 1992.

[2] T. Bolukbasi, J. Wang, O. Dekel, and V. Saligrama. Adaptive neural networks for efficient inference. In *International Conference on Machine Learning*, pages 527–536. PMLR, 2017.

[3] A. Bulat, J. Kossaifi, G. Tzimiropoulos, and M. Pantic. Toward fast and accurate human pose estimation via soft-gated skip connections. *arXiv preprint arXiv:2002.11098*, 2020.

[4] A. Bulat and G. Tzimiropoulos. How far are we from solving the 2D & 3D face alignment problem? In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1021–1030, 2017.

[5] J. Carreira, V. Patraucean, L. Mazare, A. Zisserman, and S. Osindero. Massively parallel video networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 649–666, 2018.

[6] X. Chen, H. Dai, Y. Li, X. Gao, and L. Song. Learning to stop while learning to predict. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1520–1530. PMLR, 13–18 Jul 2020.

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[8] M. Elbayad, J. Gu, E. Grave, and M. Auli. Depth-adaptive transformer. In *International Conference on Learning Representations*, 2020.

[9] D. J. Felleman and D. C. Van Essen. Distributed hierarchical processing in the primate cerebral cortex. In *Cereb cortex*. Citeseer, 1991.

[10] V. Fischer, J. Köhler, and T. Pfeil. The streaming rollout of deep networks-towards fully model-parallel execution. In *Advances in Neural Information Processing Systems*, pages 4039–4050, 2018.

[11] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

[12] A. Graves. Adaptive computation time for recurrent neural networks. *CoRR*, abs/1603.08983, 2016.

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[14] H. Hu, D. Dey, M. Hebert, and J. A. Bagnell. Anytime neural network: a versatile trade-off between computation and accuracy, 2018.

[15] H. Hu, D. Dey, M. Hebert, and J. A. Bagnell. Learning anytime predictions in neural networks via adaptive loss balancing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3812–3821, 2019.

[16] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Weinberger. Multi-scale dense networks for resource efficient image classification. In *International Conference on Learning Representations*, 2018.

[17] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger. Multi-scale dense networks for resource efficient image classification. *arXiv preprint arXiv:1703.09844*, 2017.

[18] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[19] M. Iuzzolino, Y. Singer, and M. C. Mozer. Convolutional bipartite attractor networks. *arXiv preprint arXiv:1906.03504*, 2019.

[20] Z. Jiang, C. Zhang, K. Talwar, and M. C. Mozer. Characterizing structural regularities of labeled data in overparameterized models. In *Proceedings of the International Conference on Machine Learning*, 2021.

[21] M. Jones, S. Kinoshita, and M. C. Mozer. Optimal response initiation: Why recent experience matters. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, volume 21. Curran Associates, Inc., 2009.

[22] K. Kar, J. Kubilius, K. Schmidt, E. B. Issa, and J. J. DiCarlo. Evidence that recurrent circuits are critical to the ventral stream's execution of core object recognition behavior. *Nature Neuroscience*, 04/2019 2019.

[23] Y. Kaya, S. Hong, and T. Dumitras. Shallow-deep networks: Understanding and mitigating network overthinking. In *International Conference on Machine Learning*, pages 3301–3310. PMLR, 2019.

[24] N. Kriegeskorte. Deep neural networks: A new framework for modeling biological vision and brain information processing. *Annual Review of Vision Science*, 1(1):417–446, 2015.

[25] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Computer Science, University of Toronto, 2009.

[26] O. Kumbhar, E. Sizikova, N. J. Majaj, and D. G. Pelli. Anytime prediction as a model of human reaction time. *CoRR*, abs/2011.12859, 2020.

[27] G. Larsson, M. Maire, and G. Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. In *International Conference on Learning Representations*, 2017.

[28] H. Lee and J. Shin. Anytime neural prediction via slicing networks vertically. *arXiv preprint arXiv:1807.02609*, 2018.

[29] S. Liang, Y. Li, and R. Srikant. Enhancing the reliability of out-of-distribution image detection in neural networks. *arXiv preprint arXiv:1706.02690*, 2017.

[30] E. S. Marquez, J. S. Hare, and M. Niranjan. Deep cascade learning. *IEEE transactions on neural networks and learning systems*, 29(11):5475–5485, 2018.

[31] M. E. Masson and G. R. Loftus. Using confidence intervals for graphically based data interpretation. *Canadian Journal of Experimental Psychology/Revue canadienne de psychologie expérimentale*, 57(3):203, 2003.

[32] J. L. McClelland. On the time relations of mental processes: an examination of systems of processes in cascade. *Psychological review*, 86(4):287, 1979.

[33] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.

[34] L. McIntosh, N. Maheswaranathan, D. Sussillo, and J. Shlens. Recurrent segmentation for variable computational budgets. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1648–1657, 2018.

[35] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[36] A. Newell, K. Yang, and J. Deng. Stacked hourglass networks for human pose estimation. In *European conference on computer vision*, pages 483–499. Springer, 2016.

[37] J. C. Peterson, R. M. Battleday, T. L. Griffiths, and O. Russakovsky. Human uncertainty makes classification more robust. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9617–9626, 2019.

[38] R. Ratcliff and G. McKoon. The diffusion decision model: theory and data for two-choice decision tasks. *Neural Computation*, 20(4):873—922, 2008.

[39] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[40] S. Scardapane, M. Scarpiniti, E. Baccarelli, and A. Uncini. Why should we add early exits to neural networks? *Cognitive Computation*, 12(5):954–966, 2020.

[41] C. J. Spoerer, T. C. Kietzmann, J. Mehrer, I. Charest, and N. Kriegeskorte. Recurrent neural networks can explain flexible trading of speed and accuracy in biological vision. *bioRxiv*, 2020.

[42] C. J. Spoerer, P. McClure, and N. Kriegeskorte. Recurrent convolutional neural networks: A better model of biological object recognition. *Frontiers in Psychology*, 8:1551, 2017.

[43] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[44] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[45] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[46] S. Teerapittayanon, B. McDanel, and H.-T. Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469. IEEE, 2016.

[47] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[48] X. Wang, Y. Luo, D. Crankshaw, A. Tumanov, F. Yu, and J. E. Gonzalez. Idk cascades: Fast deep learning by learning not to overthink. *arXiv preprint arXiv:1706.00885*, 2017.

[49] L. Yang, Y. Han, X. Chen, S. Song, J. Dai, and G. Huang. Resolution adaptive networks for efficient inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2369–2378, 2020.

[50] F. Yu, A. Seff, Y. Zhang, S. Song, T. Funkhouser, and J. Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

[51] S. Zagoruyko and N. Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[52] A. R. Zamir, T.-L. Wu, L. Sun, W. B. Shen, B. E. Shi, J. Malik, and S. Savarese. Feedback networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1308–1317, 2017.

[53] S. Zilberstein. Using anytime algorithms in intelligent systems. *AI magazine*, 17(3):73–73, 1996.

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] In results and discussion sections
   (c) Did you discuss any potential negative societal impacts of your work? [No] We cannot conceive of negative impacts beyond the generic concerns about ML/AI models.
   (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? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] Derivations of the fairly trivial theory underlying our model is presented in the supplementary materials.
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] Details of experimental procedure are in supplemental materials. We will provide a git repository with code once the paper is accepted.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] In supplemental materials.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Where applicable, except for speed-accuracy curves where the error bars were small and the figure was sufficiently complex that error bars would make it harder to interpret figure.
   (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)? [No] We did describe our computing infrastructure in the supplementary materials.
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? [N/A] We do not cite the authors of the three data sets we used, CIFAR-10, CIFAR-100, and TinyImageNet
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (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]