ANYTIME NEURAL NETWORK: A VERSATILE TRADE-OFF BETWEEN COMPUTATION AND ACCURACY

Anonymous authors
Paper under double-blind review

ABSTRACT

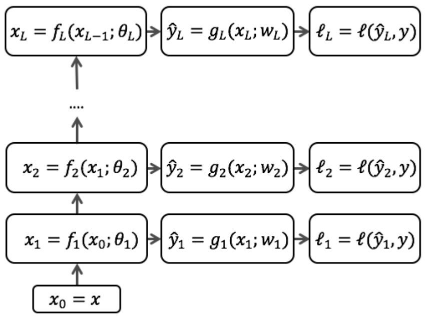
Anytime predictors first produce crude results quickly, and then continuously refine them until the test-time computational budget is depleted. Such predictors are used in real-time vision systems and streaming-data processing to efficiently utilize varying test-time budgets, and to reduce average prediction cost via early-exits. However, anytime prediction algorithms have difficulties utilizing the accurate predictions of deep neural networks (DNNs), because DNNs are often computationally expensive without competitive intermediate results. In this work, we propose to add auxiliary predictions in DNNs to generate anytime predictions, and optimize these predictions simultaneously by minimizing a carefully constructed weighted sum of losses, where the weights also oscillate during training. The proposed anytime neural networks (ANNs) produce reasonable anytime predictions without sacrificing the final performance or incurring noticeable extra computation. This enables us to assemble a sequence of exponentially deepening ANNs, and it achieves, both theoretically and practically, near-optimal anytime predictions at every budget after spending a constant fraction of extra cost. The proposed methods are shown to produce anytime predictions at the state-of-the-art level on visual recognition data-sets, including ILSVRC2012.

1 INTRODUCTION

An increasing number of machine learning applications now require fast and accurate responses. For examples, autonomous vehicles require real-time object detectors, and spam filters on email servers desire low latency. Furthermore, the computational budget limits can vary at test-time, e.g., autonomous vehicles need faster object detection at high speed, and servers have less time per sample during peak traffic. Hence, it is often difficult to balance the trade-off between computation and accuracy. One approach to such problems is through anytime predictors (Grass & Zilberstein, 1996; Yang et al., 2007), which output crude results first quickly, and then continuously refine them until the budgets are depleted. Such versatile predictors not only automatically utilize all the available budgets, but also enable systems to exit early on easy samples in order to focus computation on difficult ones. Using anytime predictions, Bolukbasi et al. (2017) speed up neural networks by three times, and (Mai et al., 2016) speed up a density-based clustering algorithm by orders of magnitude.

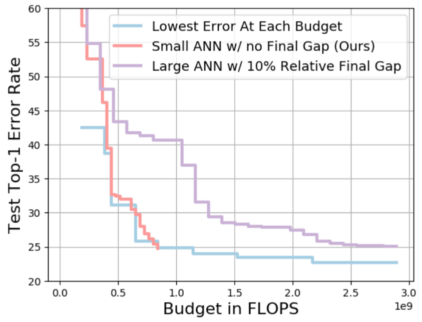
To produce competitive anytime predictions, an anytime predictor produces a sequence of predictions that are gradually more expensive and accurate, e.g., a wide array of works (Yang et al., 2007; Grubb & Bagnell, 2012; Chen et al., 2012; Xu et al.; Cai et al., 2015; Hu et al., 2016; Bolukbasi et al., 2017; Guan et al., 2017) have studied how to order the computation of features or predictions to form such a sequence. These meta-algorithms, however, rely on assembling existing features and predictors, and have difficulty utilizing the modern computationally intensive deep neural networks (DNNs), because DNNs typically greatly improve prediction performance at the cost of long chains of computation without intermediate outputs. Hence it would be ideal to have gradually improving predictions within a DNN so that we can trade off between computation and accuracy smoothly in order to adjust to, and utilize, any test-time computational budget.

To achieve competitive anytime predictions in DNNs, we propose to uniformly add auxiliary predictions and losses to intermediate layers of existing feed-forward networks to form *Anytime Neural Networks* (ANNs), as shown in Fig. 1a. Specifically, we desire the final prediction's performance to reach that of the original network. Subject to this constraint, we desire the anytime prediction performance after any cost B to be near the optimal at B, where we define the optimal at B as the



$x_L = f_L(x_{L-1}; \theta_L)$
$\hat{y}_L = g_L(x_L; w_L)$
$\ell_L = \ell(\hat{y}_L, y)$
$x_2 = f_2(x_1; \theta_2)$
$\hat{y}_2 = g_2(x_2; w_2)$
$\ell_2 = \ell(\hat{y}_2, y)$
$x_1 = f_1(x_0; \theta_1)$
$\hat{y}_1 = g_1(x_1; w_1)$
$\ell_1 = \ell(\hat{y}_1, y)$
$x_0 = x$

(a) Anytime Neural Network



(b) ANNs Should Have Low Final Gap

Figure 1: **(a)** Anytime neural networks contain auxiliary predictions and losses, $\hat{y}_i$ and ℓ_i, for intermediate feature unit f_i. In this work, each f_i is a sequence of residual building blocks; each $\hat{y}_i$ is from a global average pooling followed by a linear prediction; ℓ_i is the cross-entropy loss. **(b)** A previous ANN (purple) with just 10% relative final error rate gap from the optimal (blue) spends three times more computation to reach 25% error rate on CIFAR100 than the proposed ANN with no final gap (red).

performance of the network trained specifically to output at cost B. We highlight the importance of the low final performance gap, because previous works that utilize auxiliary predictions (Lee et al., 2015; Xie & Tu, 2015; Zamir et al., 2017) are using training strategies that may increase the final error rates relatively by $> 15\%$, which is significant, as shown in Fig. 1b, because the gap can cost anytime networks three times more computation than a regular DNN to reach the same accuracy.

We make the following contributions to achieve competitive anytime predictions from the auxiliary predictors. (1) We propose loss weighting schemes that utilize the high correlation among neighboring predictions, so that we maintain the accuracy of the final prediction ($\sim 2.5\%$ relative gap) and have competitive auxiliary predictions in the deeper layers. (2) We also propose to oscillate the loss weighting during training to further improve each individual anytime prediction by avoiding spurious solutions that are optimal for the sum of the losses but not individual ones. (3) Furthermore, thanks to these near-optimal late predictions, we assemble ANNs of exponentially increasing depths to perform near-optimally at every budget theoretically and practically.

2 Related Works

Prediction with test-time budget limits. Various works have studied how to predict efficiently to meet test-time budget limits by organizing the test-time computation of predictors and features. (Reyzin, 2011; Grubb & Bagnell, 2012; Hu et al., 2016) apply boosting and greedy methods to order feature and predictor computation. (Weinberger et al., 2009; Xu et al., 2012; 2013) sequentially generate features to empower the final predictor. (Karayev et al., 2012; Odena et al., 2017) form Markov Decision Processes for computation of weak predictors and features, and learn policies to order them. Cascade design (Viola & Jones, 2001; Lefakis & Fleuret, 2010; Chen et al., 2012; Xu et al.; Cai et al., 2015; Nan & Saligrama, 2017) speed up test-time prediction by early exiting on easy samples and focusing computation on difficult ones. However, these meta-algorithms are not easily compatible with complex and accurate predictors like CNNs, because the anytime predictions before the CNN computation are inaccurate, and there are no intermediate results during the long computation of the CNN. This work addresses this problem by learning competitive anytime predictions end-to-end within neural networks. These anytime predictions can also utilize gating functions from adaptive models (Bolukbasi et al., 2017) to reduce average test-time computation.

Computationally efficient neural networks. Many works have studied how to compress neural networks without sacrificing accuracy. Reducing redundant information in neural networks, (Li et al., 2017; Liu et al., 2017) prune network weights and connections; (Hubara et al., 2016; Rastegari et al., 2016; Iandola et al., 2016) quantize weights within networks to reduce computation and memory footprint. (Ba & Caruana, 2014; Hinton et al., 2014) transfer knowledge of deep networks into shallow ones by changing the training target of shallow networks. This work addresses the computational efficiency of deep networks from an orthogonal approach, where we, instead of making models small, train networks to be adaptive to varying budget limits. Furthermore, this work is compatible with the above-mentioned methods to produce potentially more efficient predictors.

Networks with auxiliary predictions. Auxiliary predictions have been used in neural networks for various purposes. (Lee et al., 2015; Szegedy et al., 2017; Zhao et al., 2017) regularize networks training using auxiliary predictions. (Bengio et al., 2009; Zamir et al., 2017) train auxiliary predictions to target label sets instead one-hot labels to improve final predictions. An increasing number of works (Xie & Tu, 2015; Larsson et al., 2017; Bolukbasi et al., 2017; Huang et al., 2017) now also appreciate high-quality auxiliary predictions, i.e., anytime predictions, for analyzing images multiple scales and speeding up average test-time predictions. We contribute to works on auxiliary predictions in two folds. First, we propose better-performing and easy-to-implement strategies to train all the auxiliary predictions end-to-end. In particular, the proposed methods leverage the correlation among neighboring auxiliary predictions to achieve near-optimal level of anytime predictions near the final outputs, where the previous methods suffer. Second, instead of competing against ensembles of networks as in the previous works, we show that assembling a sequence of exponentially deepening ANNs can greatly improve the cost-effectiveness of predictions.

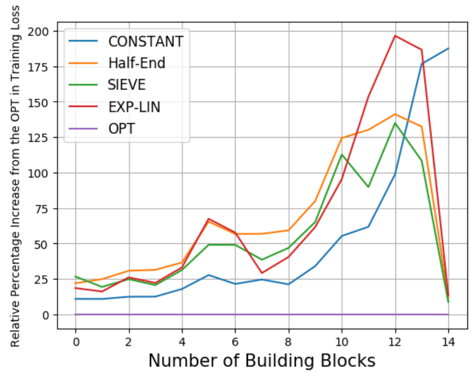
3 Training Anytime Neural Networks

As illustrated in Fig. 1a, given a sample $(x, y) \sim D$, the initial feature map x_0 is set to x, and the subsequent feature transformations $f_1, f_2, ..., f_L$ generate a sequence of intermediate features $x_i = f_i(x_{i-1}; \theta_i)$ for $i \geq 1$ using parameter θ_i. Each feature map x_i can then produce an auxiliary prediction $\hat{y}_i$ using a prediction layer g_i: $\hat{y}_i = g_i(x_i; w_i)$ with parameter w_i. Each auxiliary prediction $\hat{y}_i$ then incurs an expected loss $\ell_i := E_{(x,y)\sim D}[\ell(y, \hat{y}_i)]$. We call such a feed-forward network with auxiliary predictions and losses an *Anytime Neural Network (ANN)*. For instance, we base our ANNs on ResNets (He et al., 2016), so each f_i contains s number of residual building blocks (whose implementation details are deferred to the appendix), where s is the prediction period. Each g_i outputs the predicted label distribution $\hat{y}_i$ using a global pooling layer followed by a linear layer and a softmax; the loss function ℓ is the cross-entropy loss. During test-time, an ANN computes $\hat{y}_i$ sequentially until interruption, and outputs the latest available prediction. The interruption can be issued by users, or can be learned with additional computations as in (Bolukbasi et al., 2017).

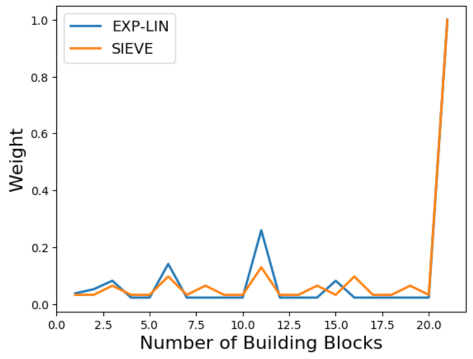
3.1 Joint Optimization of Losses via an Oscillating Weighted Sum

Let the parameters of the full ANN be $\theta = (\theta_1, w_1, ..., \theta_L, w_L)$. Let $\ell_i^* := \min_\theta \ell_i(\theta)$ be the optimal loss at the i^{th} prediction. The goal of training an ANN is to find a single θ^* such that $\ell_i(\theta^*) = \ell_i^*$ for all i, i.e., $\theta^* \in \cap_{i=1}^L \{x : x = \arg\min_\theta \ell_i(\theta)\}$. Such an ideal θ^* does not exist in general as this is a multi-objective optimization, and it often does not exist in practice. Hence, we instead minimize a weighted sum of the losses in hopes of approximating ℓ_i^* with $\ell_i(\theta)$, i.e., we solve $\min_\theta \sum_{i=1}^L B_i \ell_i(\theta)$, where B_i is the weight of the loss ℓ_i. We call the various choices of the B_i *weight schemes*. The most intuitive scheme is the CONST scheme, which set $B_i = 1$ for all i. This is a part of the formulation of a number of previous works (Lee et al., 2015; Xie & Tu, 2015; Huang et al., 2017). Other works have proposed LINEAR scheme, which set the weights to be linearly increasing (Zamir et al., 2017). However, as we will see in Sec. 4.2, optimizing the weighted sum to convergence using these intuitive schemes lead to about 12% relative increase in test error, in comparison to optimizing only the final loss. This is significant, as we explained in Fig. 1b.

Weight scheme. We now explain how we form the proposed weight scheme. We study on a ResNet of 15 residual building blocks for the CIFAR100 (Krizhevsky, 2009) data-set. To know the best possible performance at each depth i, we train a network whose weights are only on ℓ_i to convergence. We call the resulting performance OPT, and compare the training loss from CONST against OPT in Fig. 2a. We observe that within an ANN, the CONST scheme is substantially worse than OPT in deep layers. To remedy this, we set B_L to be half of the total weights, so that the final loss gradient can overcome the sum of all other loss gradients when all gradients have equal 2-norms. If we distribute the remaining half of the total weights evenly among the first $L - 1$ layers, we get the "Half-End" scheme in Fig. 2a, which reduces the final loss to be near the OPT, but increases the the early losses. Luckily, we also found that increasing weightings at a depth will improve the performance of a neighborhood. E.g., Half-End improves the penultimate layer, even its relative weighting is smaller in Half-End than in CONST. For another example, if we triple the weights at $L/4$, $3L/4$ and quadruple the weights at $L/2$ in the Half-End, the depths around these milestone depths also have reduced loss even their weights are unaffected, as shown in the SIEVE scheme in Fig. 2a. We believe the more uneven weight scheme wins here because neighboring layers share most of their parameters, and the well optimized prediction at the milestone layers make the sur-



(a) Relative increase in Training Loss vs. depths



(b) Proposed Weight Schemes

Figure 2: **(a)** We want high weight in the final layer, as the CONSTANT scheme is increasingly worse than the optimal at high depths. The more uneven SIEVE scheme outperforms Half-End, because in SIEVE, each high weight improves a neighborhood of losses. **(b)** The proposed weight schemes for $L = 21$.

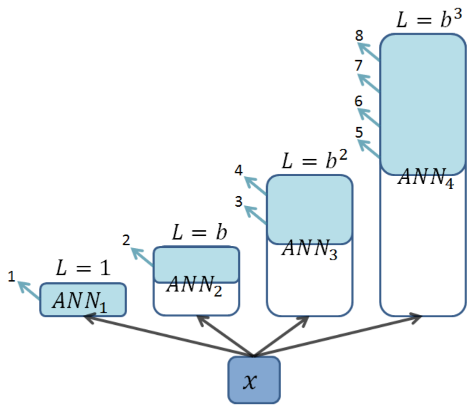


Figure 3: ANNs in an EANN are computed in order of depth; anytime results are reported if the current depth in the current ANN is higher than the depth of the previous ANN (light blue region), and the new results are better on validation sets.

rounding layers easier to optimize. To scale the ***SIEVE*** scheme to larger L, we form it iteratively: we first add to $B_{\lfloor \frac{L}{2} \rceil}$ one unit of weight, where $\lfloor \bullet \rceil$ means rounding. We then add one unit to each $B_{\lfloor \frac{kL}{4} \rceil}$ for $k = 1, 2, 3$, and then to each $B_{\lfloor \frac{kL}{8} \rceil}$ for $k = 1, 2, ..., 7$, and so on, until all predictors have non-zero weights. We finally normalize B_i so that $\sum_{i=1}^{L-1} B_i = 1$, and set $B_L = 1$. The values of B_1 to B_{L-1} resemble the length of markings on a ruler: large weights are few and apart, and are interleaved by small weights, as shown on Fig. 2b.

Alternating weights. We also propose to oscillate the training loss for each batch of samples, since minimizing $\sum_{i=1}^{L} B_i \ell_i(\theta)$ may lead to spurious solutions where the sum of the gradients is zero, $\sum_{i=1}^{L} B_i \nabla \ell_i(\theta) = 0$, but individual gradients $\nabla \ell_i(\theta)$ are not. To do so, we choose, in each iteration, a loss ℓ_i, and temporarily increase its weight, B_i, for the current iteration, by a constant τ of the total weights, $\tau \sum_{i=1}^{L} B_i$. We choose layer i with the probability proportional to the weight B_i in the weight scheme. We call this sampling strategy **Alternating ANN** (AANN), and will show experimentally in Sec. 4.1 that it improves anytime predictions at each budget limit. We also learn the sampling strategy via no-regret online learning, but this requires more hyper-parameters and has almost no performance gains in our experiments, as shown in the appendix. This finding echoes the recent results by Graves et al. (2017) that sampling according to B_i is a competitive baseline.

3.2 Sequence of Exponentially Deepening Anytime Neural Networks

Since we have half of the total weights at the final loss, we observe in Sec 4.2 that ANNs are much closer to the optimal in the second half of the network than in the first half. In order to improve the early predictions, we leverage the more accurate late predictions in ANNs, by forming a sequence of ANNs whose depths grow exponentially (**EANN**). As shown in Fig 3, the EANN sequentially computes the independently-trained ANNs whose depths grow exponentially in base $b = 2$. Then at interruption, the EANN outputs the better prediction between the final prediction of the latest finished ANN, and the latest anytime prediction of the currently-running ANN, where the comparison is determined on a validation set. Intuitively, if for some $b > 1$ each ANN is competitive after $\frac{1}{b}$ of its layers are computed, then we can recursively delegate the predictions in the early $\frac{1}{b}$ fraction of each network to a smaller network that has $\frac{1}{b}$ fraction of the cost. Then we only rely on the competitive predictions from the later $1 - \frac{1}{b}$ fraction of each ANN. Furthermore, because the ANN depths grow exponentially, we only pay a constant $C - 1$ fraction of extra costs to evaluate all the shallower ANNs first for some $C > 1$. Formally, the following proposition proves that EANN is competitive at any budget with a constant fraction of additional computational complexity.

Proposition 3.1. *Let $b > 1$. Assume for any L, any ANN of depth L has competitive anytime prediction at depth $i > \frac{L}{b}$ against the optimal of depth i. Then after B layers of computation, EANN produces anytime predictions that are competitive against the optimal of depth $\frac{B}{C}$ for some $C > 1$, such that $\sup_B C = 2 + \frac{1}{b-1}$, and C has expectation $E_{B\sim uniform(1,L)}[C] \leq 1 - \frac{1}{2b} + \frac{1+\ln(b)}{b-1}$.*

This proposition implies that if we multiply the allowed budget by C, we can build an anytime predictor that is competitive everywhere. Furthermore, the stronger each anytime model is, the larger b becomes, so that the worst case inflation rate, $\sup_B C$, shrinks to 2, and the average rate, $E[C]$, shrinks to 1. Moreover, if we have M number of parallel workers instead of one, we can speed up EANNs by computing ANNs in parallel in a first-in-first-out schedule, so that we effectively increase the constant b to b^M for computing C. It is also worth noting that if we form the sequence using regular networks instead of ANNs, then we will lose the ability to output frequently, as we only produce $\Theta(\log(B))$ intermediate predictions instead of the $\Theta(B)$ predictions in an EANN. We will further have a larger cost inflation, C, such that $\sup_B C \geq 4$ and $E[C] \geq 1.5 + \sqrt{2} \approx 2.91$, so that the average cost inflation is at least about 2.91. We defer the proofs to the appendix.

Weight scheme inspired by EANN. Inspired by the EANN, which achieves near-optimal guarantee by having near-optimal performance at exponentially spaced locations, we also consider the following weight scheme. The ***EXP-LIN*** weight scheme sets $B_{\lfloor \frac{L}{2^k} \rceil} = 2^{-k}$ for $k = 1, ..., \log_2(L)$, and then adds $1/L$ to each B_i. The idea of this scheme is to repeatedly apply the following idea: if a location is important, then its gradient should have enough weight to overcome the sum of the less important gradients when all the gradients have equal norms. This idea suggest we exponentially decrease the weight in the order of priority. Besides the exponentially spaced locations suggested by EANN, we also want to prioritize the layer at $\frac{3L}{4}$, because it is natural to prioritize the milestones at 1/4, 1/2, and 3/4 of the total costs, and the relative loss increase from the OPT at $\frac{3L}{4}$ is high for EXP-LIN as in Fig. 2a. We set $\frac{3L}{4}$ to have priority after $\frac{L}{2}$ and $\frac{L}{4}$. We again normalize the weights so that $\sum_{i=1}^{L-1} B_i = 1$, and set $B_L = 1$, as shown in Fig. 2b.

4 Experiments

We experiment with ANNs on CIFAR10, CIFAR100 (Krizhevsky, 2009), and SVHN (Netzer et al., 2011) [1]. We evaluate our ANN augmentations on 13 different ResNets (He et al., 2016). We refer to these models as (n, c), where n is the number of basic residual building blocks (He et al., 2016), each of which contains two BN-ReLU-3x3conv and a BN-ReLU-addition with the input, and c is the number of channels of the initial convolution. Each network contains $3n$ blocks or $6n + 2$ conv layers. The 13 models are in $\{7, 9, 13, 17, 25\} \times \{16, 32\}$ and $\{(9, 64), (9, 128), (13, 64)\}$. We set the prediction period $s = 1$, i.e., an anytime prediction is expected at every building block, unless specified otherwise. We defer a detailed study of the prediction period to the appendix. We optimize all networks with stochastic gradient descent with a momentum of 0.9. The learning rate starts at

[1] All three data-sets consist of 32x32 colored images. CIFAR10 and CIFAR100 have 10 and 100 classes, and each have 50000 training and 10000 testing images. The last 5000 training samples in CIFAR10 and CIFAR100 were held out as a validation set for hyper parameter tuning using the network $(5, 16)$; the same parameters are then used in other models for these three data-sets. We adopt the standard augmentation from Lee et al. (2015); He et al. (2016). SVHN contains around 600000 training and around 26032 testing images of numeric digits from the Google Street Views. We adopt the same pad-and-crop augmentations of CIFAR for SVHN.

Method	1/4	1/2	3/4	1
CONST	43.59	64.10	61.54	64.10
LINEAR	38.46	53.85	51.28	61.54
EXP-LIN	28.21	46.15	64.10	64.10
SIEVE	43.59	38.46	66.67	66.67

(a) Percentage of the 39 Experiments where AANN outperforms ANN using each weight scheme

Method	1/4	1/2	3/4	1
CONST	14.82	15.25	18.19	18.40
LINEAR	27.30	12.92	12.60	12.87
EXP-LIN	31.11	22.05	12.63	**2.74**
SIEVE	34.63	24.62	16.21	**2.29**

(b) The mean of relative percentage increase in error from OPT by each weight scheme (0 means OPT)

Table 1: **(a)** Percentages of experiments that oscillating training loss outperforms targeting a static loss. **(b)** The averages of relative increases in error rates of various AANNs from OPT in percentages.

0.1, and is divided by 10 after $\frac{1}{2}$ and $\frac{3}{4}$ of all epochs. We train for 300 epochs on CIFAR and 60 epochs on SVHN. We evaluate using single-crop.

To quantitatively evaluate each training technique, we first train the optimal (**OPT**) predictions at 1/4, 1/2, 3/4 and 1 of the total depth for each network, where the optimal at depth i is from training an ANN specifically to predict $\hat{y}_i$. Then we can compute for each ANN the relative increase in top-1 error rate at these four milestone depths. The ANN error rate is bi-linearly interpolated if the ANN doesn't predict at these milestones. We compare the proposed weight schemes against the popular CONST scheme ($B_i = 1$ for all i), which is used in (Lee et al., 2015; Xie & Tu, 2015; Huang et al., 2017) for its simplicity, and LINEAR scheme, which is used by Zamir et al. (2017) and increases the weight linearly from 0.25 at B_1 to 1 at B_L.

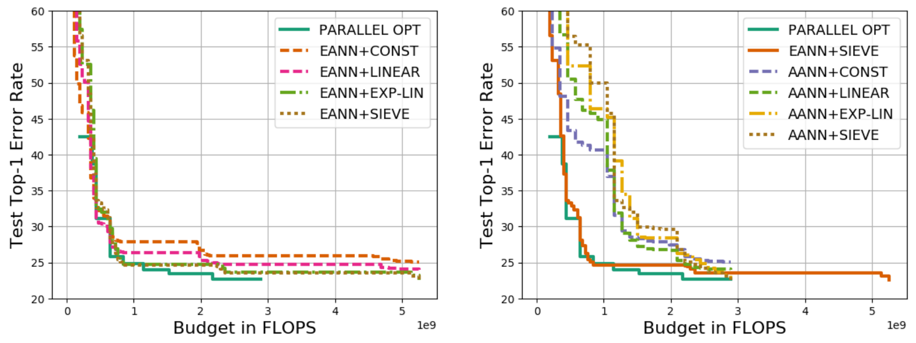
4.1 AANN: A General Technique to Improve All Predictions

We showcase AANN as a useful technique independent of the weight schemes. As explained in Sec. 3, AANN samples a layer i with the probability proportional to B_i in each iteration, and increases the weight B_i by $\tau \sum_{i=1}^{L} B_i$ for the iteration. We choose $\tau = 0.5$ from the set $\{0.25, 0.5, 1.0, 2.0\}$ by experiments on the validation network. We experiment on all 39 settings to compare AANNs against ANNs. Table 1a presents the percentage of the 39 settings such that AANNs outperform ANNs. Treating the experimental settings as 39 independent coin flips, we apply hypothesis testing to understand these percentages. For each entry in Table 1a, let X be the probability listed in the entry and $p \in [0, 1]$ be the probability that AANN outperforms ANN using the associated weight scheme at the associated milestone cost. Let the null hypothesis H_0 be "$p \leq 0.5$" and the alternative hypothesis H_1 be "$p > 0.5$". Using Hoeffding's inequality[2], we have $Pr(X|H_0) \leq 0.05$ if $X \geq 0.598$, and $Pr(X|H_0) \leq 0.001$ if $X \geq 0.65$. Then since almost all $X > 0.6$ in Table 1a for the 3/4 and 1 milestones, we conclude that sampling according to B_i to form AANNs allows us to improve predictions in the second half of ANNs independent of weight schemes. This also suggests that optimizing a static weighted sum of anytime losses hinders the optimization of the late predictions more than that of the early predictions. From now on we assume all ANNs are trained with AANN.

4.2 Weighting Scheme Comparisons

Table 1b presents the average relative performance gap from the OPT by various weight schemes. We observe that the CONST scheme makes $15 \sim 18\%$ more errors than the OPT, and the relative gap widens at deeper depths. The LINEAR scheme also has a 13% relative gap in the final prediction. In CIFAR100, such a relative final error translates to $3 \sim 4\%$ absolute error, which means the final performances of ANNs with CONST scheme are often equivalent to those of ResNets of $< \frac{1}{3}$ of the cost, e.g., on CIFAR100, $(9, 128)$ achieves 22.01% final error using CONST and 21.25% using LINEAR, but the four-times-cheaper $(9, 64)$ achieves 21.14% and 20.85% using EXP-LIN and SIEVE. Sec 4.3 will further show that these final performance gaps are highly detrimental to the cost-efficiency of EANNs. These two intuitive weight schemes fail at the final prediction, possibly because the final prediction is more difficult to train than the early ones, but it only was assigned with a weighting on the order of $\frac{1}{L}$. On the other hand, the proposed SIEVE and EXP-LIN schemes both enjoy very small relative performance gap from the OPT at the final prediction ($\sim 2.5\%$). Furthermore, we also found ANNs with the proposed schemes can outperform the OPT at the final predictions of expensive models, such as $(9, 64)$, $(9, 128)$, where OPT has trouble converging without dropouts(Zagoruyko & Komodakis, 2016). This is possibly because the regularization effect of the anytime predictions, used by Lee et al. (2015); Szegedy et al. (2017); Zhao et al. (2017). The

[2]Using Hoeffding's inequality on the 39 experiments, we know $Pr(X|H_0) \leq Pr(X \geq p(\epsilon + 1)) \leq \exp(-78\epsilon^2)$. Under the hypothesis H_0, for each probability X Table 1a, we set $\epsilon = 2X - 1 \leq \frac{X}{p} - 1$.



(a) CIFAR100 EANNs vs. OPT+ (b) CIFAR100 EANN vs. AANNs

Figure 4: **(a)** Performance of EANN on CIFAR100. The proposed SIEVE and EXP-LIN schemes achieve much better performance than CONST and LINEAR due to low final performance gaps. **(b)** EANNs are so much more efficient than AANNs that the early predictions of CONST and LINEAR are irrelevant.

optimal-level final performances of the proposed schemes, however, cost higher error rates in the first half of layers. Since EXP-LIN is designed to have higher weights at depth 2^k for each k, it outperforms SIEVE in the early layers.

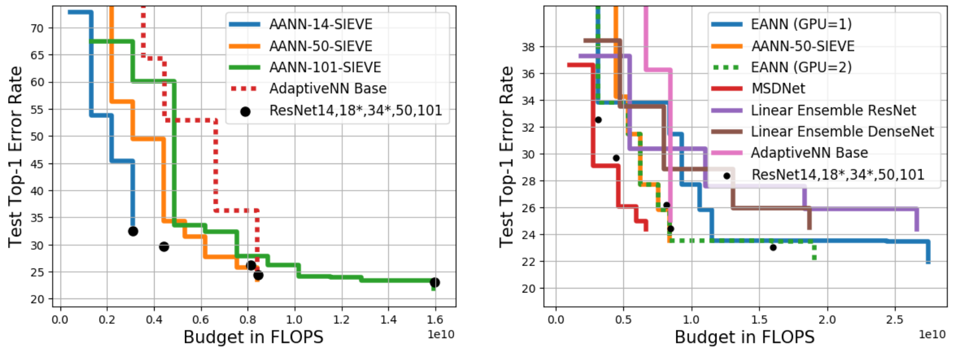
4.3 EANN: Closing Early Performance Gaps

In this section, we leverage the near-optimal final predictions ANNs with SIEVE and EXP-LIN schemes to close their early performance gap from the OPT using EANN as described in Sec. 3.2. We train individually ANN models that have $c = 32$ and $n = 7, 13, 25$ in order to form an EANN with exponential base $b \approx 2$. By proposition 3.1, $E[C] \approx 2.44$ for $b = 2$, and the EANN should compete against the optimal with $n = (7 + 13 + 25)/2.44 \approx 20$. We instead compare this EANN against a more challenging **OPT+**, where we first collect the four OPT for each of model that also has $c = 32$, and then for each depth i, we set the error rate of OPT+ at i to be the minimum among the collection that also has depth no greater than i. Hence, OPT+ equates to running all the OPT in parallel and outputting the best at each cost, whereas the EANN computes its AANNs sequentially. As we noted in Sec. 3.2, parallelism in EANN effectively exponentiate the base b with the number of parallel workers, and we will discuss it in Sec. 4.4.

Fig. 4 compares EANNs with various weight schemes against the OPT+. Without any parallelism, EANNs with SIEVE and EXP-LIN schemes are shown in Fig. 4a to be near OPT+ at every budget, closing the early-mid performance gaps of AANNs. AANNs are worse than EANNs in the early cost (Fig. 4b) because the early predictions of a large AANN (n=25) is less accurate than the final predictions of a small AANN (n=7). Then since the early AANNs are exponentially cheaper, EANNs only pay a constant fraction of extra FLOPS to compute them first. Fig. 4a also shows that when combined with EANNs, the popular CONST and LINEAR schemes have clearly higher error rate for most budgets, because these schemes fail to reach near-optimal final predictions in small AANNs. For instance, to achieve the error rate 25%, CONST and LINEAR require two to six times more computation than SIEVE and EXP-LIN do. Fig. 4b further shows the better early-mid AANN predictions of CONST and LINEAR than SIEVE and EXP-LIN to be irrelevant, since the EANN+SIEVE is much more cost-efficient than the AANNs. These comparisons suggest that it is imperative to have near-optimal final predictions in ANNs, because large ANNs with final performance gaps are outperformed by small ANNs without gaps, like the early AANNs in the EANN+SIEVE. The latter are also much more suitable for the cost-efficient EANNs.

4.4 Evaluation on ILSVRC2012

To showcase the practical use of ANNs, we train ResNet+AANNs with SIEVE scheme on ILSVRC2012 (Russakovsky et al., 2015), which is a visual recognition data-set that contains around 1.2 million natural images and 50000 validation images for 1000 classes. We report the top-1 error rates on the validation set using a single-crop of size 224x224. ResNet50 and ResNet101 are designed by He et al. (2016), and they have 16 and 33 residual "bottleneck" building blocks, each of which contains a BN-ReLU-interleaved sequence of 1x1conv-3x3conv-1x1conv and a summation with the input. We set the prediction period to $s = 2$ and $s = 3$ blocks for these two networks. To



(a) AANN on ILSVRC2012 (b) EANN on ILSVRC2012

Figure 5: **(a)** ILSVRC2012 performance of individual ANNs that are used in the EANN. AANN-14 finishes at 33.50% error, AANN-50 at 23.53, and AANN-101 at 21.91. Using CONST, the final error of AANN-50 and AANN-101 are 25.14% and 23.62%, more than 1.6% higher absolutely than using SIEVE. The ResNet dots mark the errors and costs of four ResNets designed by He et al. (2016). 18* and 34* do not use bottlenecks, and hence, are more expensive for their depths. **(b)** EANN is easily parallelizable, and running AANNs of an EANN in a FIFO manner with 2 workers (orange) can reduce the time to get accurate results (blue).

form an EANN we also design ResNet14, which contains 4 residual bottleneck blocks (one block for each scale). We set the period to $s = 1$. AANNs based on these ResNet models are trained using SGD with momentum 0.9 for 120 epochs with batch size 256. The learning rate starts at 0.05, and is divided by 10 at epoch 60, 90 and 105. Since each anytime prediction only costs a global pooling followed by a linear product, anytime predictions in total cost $< 0.1\%$ of the total FLOPS.

In Fig. 5a, we observe that the AANN with the SIEVE scheme on each of ResNet14,50,101 can achieve the optimal level of performance at the final layer, and the error rates are near-optimal in the later half of the depths. AANN50 also outperforms significantly the underlying anytime ResNet50 used by Adaptive Networks (Bolukbasi et al., 2017). Hence, combining their early stopping policies with the proposed AANNs could drastically improve the cost-efficiency. We also note that ANNs with deeper final depths have better final performances but worse early ones, so there is a trade-off between the final and early accuracy. In Fig. 5b, the EANN using these three AANNs achieves the optimal performance at a constant fraction of additional cost, as suggested by Proposition 3.1. In addition, by running AANNs in parallel in a FIFO manner on two worker GPUs, we can speed up the EANN significantly as the exponential base b effectively becomes b^2. We also compare against an ensemble of ResNets whose depths grow linearly from 10 to 50. Linear ensembles are used by Huang et al. (2017) and Zamir et al. (2017) as baselines for anytime predictors. We observe in Fig. 5b that both AANN101 and the EANN outperform this baseline. Moreover, thanks to the ANN outputs, EANN predicts more frequently and more accurately than the linear ensemble, even though the EANN evaluate large networks exponentially earlier. Finally, we also show the performance of the recent MSDNet (Huang et al., 2017), which is slightly faster, but produces 0.8% more errors than AANN50. We also note that the proposed alternating weights, and the exponential ensembles are not restricted to ResNet based ANNs, so it will be interesting future work to see whether the proposed techniques are beneficial to ANNs based on other networks.

5 Conclusion and Discussion

In this work, we propose weighting techniques to achieve anytime predictions in deep neural networks without degrading the final performance. Such anytime predictors can be combined using an ensemble technique to achieve near-optimal level of performance at all budgets in theory and in practice. These versatile and near-optimal anytime predictions allow applications to automatically adjust to varying test-time budget limits, and we can use cascade early-exit policies with the proposed methods to reduce average prediction time. For future work, ANNs can possibly benefit from more complex anytime predictors than the simple linear predictors on the globally pooled features. EANNs can also be formed based on both the computational cost and accuracy of ANNs. This can potentially be addressed by adaptive models (Bolukbasi et al., 2017), predictor cascades (Viola & Jones, 2001; Chen et al., 2012; Cai et al., 2015; Guan et al., 2017), and anytime model selection (Grubb & Bagnell, 2012).

REFERENCES

Abadi, Martín and et al., Ashish Agarwal. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/.

Auer, Peter, Cesa-Bianchi, Nicolo, Freund, Yoav, and Schapire, Robert E. The nonstochastic multiarmed bandit problem. In *SIAM Journal on Computing*, 2002.

Ba, L. J. and Caruana, R. Do deep nets really need to be deep? In *Proceedings of NIPS*, 2014.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, 2009.

Bolukbasi, Tolga, Wang, Joseph, Dekel, Ofer, and Saligrama, Venkatesh. Adaptive neural networks for fast test-time prediction. In *ICML*, 2017.

Cai, Zhaowei, Saberian, Mohammad J., and Vasconcelos, Nuno. Learning Complexity-Aware Cascades for Deep Pedestrian Detection. In *International Conference on Computer Vision (ICCV)*, 2015.

Chen, Minmin, Weinberger, Kilian Q., Chapelle, Olivier, Kedem, Dor, and Xu, Zhixiang. Classifier Cascade for Minimizing Feature Evaluation Cost. In *AISTATS*, 2012.

Grass, Joshua and Zilberstein, Shlomo. Anytime Algorithm Development Tools. *SIGART Bulletin*, 1996.

Graves, Alex, Bellemare, Marc G., Menick, Jacob, Munos, Remi, and Kavukcuoglu, Koray. Automated curriculum learning for neural networks. In *ICML*, 2017.

Grubb, Alexander and Bagnell, J. Andrew. SpeedBoost: Anytime Prediction with Uniform Near-Optimality. In *AISTATS*, 2012.

Guan, Jiaqi, Liu, Yang, Liu, Qiang, and Peng, Jian. Energy-efficient amortized inference with cascaded deep classifiers. In *arxiv preprint, arxiv.org/abs/1710.03368*, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Computer Vision and Pattern Recognition (CVPR)*, 2016.

Hinton, Geoffrey, Vinyals, Oriol, and Dean, Jeff. Distilling the knowledge in a neural network. In *Deep Learning and Representation Learning Workshop, NIPS*, 2014.

Hu, Hanzhang, Grubb, Alexander, Hebert, Martial, and Bagnell, J. Andrew. Efficient feature group sequencing for anytime linear prediction. In *UAI*, 2016.

Huang, G., Chen, D., Li, T., Wu, F., van der Maaten, L., and Weinberger, K. Q. Multi-scale dense convolutional networks for efficient prediction. In *arxiv preprint: 1703.09844*, 2017.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. Binarized neural networks. In *NIPS*, 2016.

Iandola, Forrest N., Han, Song, Moskewicz, Matthew W., Ashraf, Khalid, Dally, William J., and Keutzer, Kurt. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size. In *arxiv preprint: 1602.07360*, 2016.

Karayev, Sergey, Baumgartner, Tobias, Fritz, Mario, and Darrell, Trevor. Timely Object Recognition. In *Conference and Workshop on Neural Information Processing Systems (NIPS)*, 2012.

Krizhevsky, Alex. Learning multiple layers of features from tiny images. Technical report, 2009.

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

Lee, Chen-Yu, Xie, Saining, Gallagher, Patrick W., Zhang, Zhengyou, and Tu, Zhuowen. Deeply-supervised nets. In *AISTATS*, 2015.

Lefakis, Leonidas and Fleuret, Francois. Joint Cascade Optimization Using a Product of Boosted Classifiers. In *Advances in Neural Information Processing Systems (NIPS)*, 2010.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets. In *ICLR*, 2017.

Littlestone, N. and Warmuth, M.K. The weighted majority algorithm. *Information and Computation*, 1994.

Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., and Zhang, C. Learning efficient convolutional networks through network slimming. In *arxiv preprint:1708.06519*, 2017.

Mai, Son T., Assent, Ira, and Storgaard, Martin. Anydbc: An efficient anytime density-based clustering algorithm for very large complex datasets. In *KDD*, 2016.

Nan, Feng and Saligrama, Venkatesh. Dynamic model selection for prediction under a budget. In *NIPS*, 2017.

Netzer, Yuval, Wang, Tao, Coates, Adam, Bissacco, Alessandro, Wu, Bo, and Ng, Andrew Y. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

Odena, A., Lawson, D., and Olah, C. Changing model behavior at test-time using reinforcement. In *Arxive preprint: 1702.07780*, 2017.

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016.

Reyzin, Lev. Boosting on a budget: Sampling for feature-efficient prediction. In *the 28th International Conference on Machine Learning (ICML)*, 2011.

Russakovsky, Olga, Deng, Jia, Su, Hao, Krause, Jonathan, Satheesh, Sanjeev, Ma, Sean, Huang, Zhiheng, Karpathy, Andrej, Khosla, Aditya, Bernstein, Michael, Berg, Alexander C., and Fei-Fei, Li. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 2015.

Shalev-Shwartz, Shai and Wexler, Yonatan. Minimizing the maximal loss: How and why. In *International Conference on Machine Learning (ICML)*, 2016.

Szegedy, Christian, Ioffe, Sergey, Vanhoucke, Vincent, and Alemi, Alex. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, 2017.

Viola, Paul A. and Jones, Michael J. Rapid Object Detection using a Boosted Cascade of Simple Features. In *Computer Vision and Pattern Recognition (CVPR)*, 2001.

Weinberger, K.Q., Dasgupta, A., Langford, J., Smola, A., and Attenberg, J. Feature Hashing for Large Scale Multitask Learning. In *ICML*, 2009.

Xie, Saining and Tu, Zhuowen. Holistically-nested edge detection. In *ICCV*, 2015.

Xu, Z., Kusner, M. J., Weinberger, K. Q., Chen, M., and Chapelle, O. Classifier cascades and trees for minimizing feature evaluation cost. *Journal of Machine Learning Research*.

Xu, Z., Weinberger, K., and Chapelle, O. The Greedy Miser: Learning under Test-time Budgets. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, 2012.

Xu, Z., Kusner, M., Huang, G., and Weinberger, K. Q. Anytime Representation Learning. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, 2013.

Yang, Y., Webb, G., Korb, K., and Ting, K. Classifying under computational resource constraints: Anytime classification using probabilistic estimators. *Mach. Learn.*, 2007.

Zagoruyko, Sergey and Komodakis, Nikos. Wide residual networks. In *British Machine Vision Conference (BMVC)*, 2016.

Zamir, Amir R., Wu, Te-Lin, Sun, Lin, Shen, William, Malik, Jitendra, and Savarese, Silvio. Feedback networks. In *Computer Vision and Pattern Recognition (CVPR)*, 2017.

Zhao, Hengshuang, Shi, Jianping, Qi, Xiaojuan, Wang, Xiaogang, and Jia, Jiaya. Pyramid scene parsing network. In *Computer Vision and Pattern Recognition (CVPR)*, 2017.

Supplementary Material for "Anytime Neural Network via Joint Optimization of Auxiliary Losses"

A Sketch of Proof of Proposition 3.1

Proof. For each budget consumed x, we compute the cost x' of the optimal that EANN is competitive against. The goal is then to analyze the ratio $C = \frac{x}{x'}$. The first ANN in EANN has depth 1. The optimal and the result of EANN are the same. Now assume EANN is on depth z of ANN number $n+1$ for $n \geq 0$, which has depth b^n.
(Case 1) For $z \leq b^{n-1}$, EANN reuse the result from the end of ANN number n. The cost spent is $x = z + \sum_{i=0}^{n-1} b^i = z + \frac{b^n - 1}{b-1}$. The optimal we compete has cost of the last ANN, which is b^{n-1} The ratio satisfies:

$$C = x/x' = \frac{z}{b^{n-1}} + 1 + \frac{1}{b-1} - \frac{1}{b^{n-1}(b-1)} \leq 2 + \frac{1}{b-1} + \frac{1}{b^{n-1}(b-1)} \xrightarrow{n\to\infty} 2 + \frac{1}{b-1}.$$

Furthermore, since C increases with z,

$$E_{z\sim Uniform(0,b^{n-1})}[C] \leq b^{1-n} \int_0^{b^{n-1}} zb^{1-n} + 1 + \frac{1}{b-1}\, dz = 1.5 + \frac{1}{b-1}.$$

(Case 2) For $b^{n-1} < z \leq b^n$, EANN outputs anytime results from ANN number $n+1$ at depth z. The cost is still $x = z + \frac{b^n-1}{b-1}$. The optimal competitor has cost $x' = z$. Hence the ratio is

$$C = x/x' = 1 + \frac{b^n - 1}{z(b-1)} \leq 2 + \frac{1}{b-1} + \frac{1}{b^{n-1}(b-1)} \xrightarrow{n\to\infty} 2 + \frac{1}{b-1}.$$

Furthermore, since C decreases with z,

$$E_{z\sim Uniform(b^{n-1},b^n)}[C] \leq (b-1)^{-1} b^{1-n} \left[(2 + \frac{1}{b-1}) + \int_{b^{n-1}}^{b^n} 2 + \frac{1}{b-1} + \frac{b^n - 1}{z(b-1)}\, dz \right]$$
$$\xrightarrow{n\to\infty} 1 + \frac{b \ln b}{(b-1)^2}$$

Finally, since case 1 and case 2 happen with probability $\frac{1}{b}$ and $(1 - \frac{1}{b})$, we have

$$\sup_B C = 2 + \frac{1}{b-1} \quad \text{and} \quad E_{B\sim Uniform(0,L)}[C] \leq 1 - \frac{1}{2b} + \frac{1}{b-1} + \frac{\ln b}{b-1}.$$

We also note that with large b, $\sup_B C \to 2$ and $E[C] \to 1$ from above. □

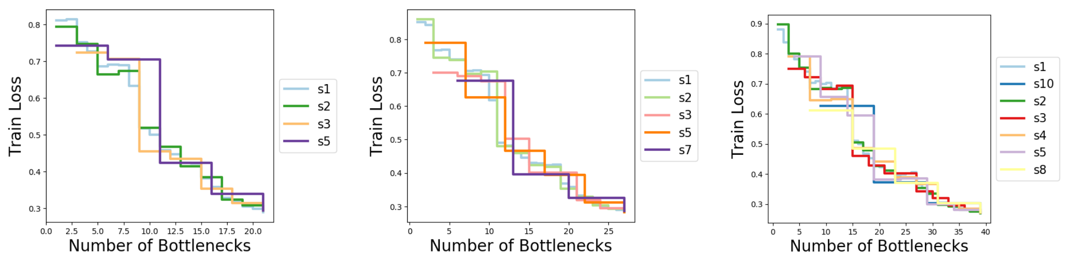
If we form a sequence of regular networks that grow exponentially in depth instead of ANN, then the worst case happen right before a new prediction is produced. Hence the ratio between the consumed budget and the cost of the optimal that the current anytime prediction can compete, C, right before the number $n+1$ network is completed, is

$$\frac{\sum_{i=1}^n b^i}{b^{n-1}} \xrightarrow{n\to\infty} \frac{b^2}{b-1} = 2 + (b-1) + \frac{1}{b-1} \geq 4.$$

Note that $(b-1) + \frac{1}{b-1} \geq 2$ and the inequality is tight at $b = 2$. Hence we know $\sup_B C$ is at least 4. Furthermore, the expected value of C, assume B is uniformly sampled such that the interruption happens on the $(n+1)^{th}$ network, is:

$$E[C] = \frac{1}{b^n} \int_0^{b^n} \frac{x + \frac{b^n-1}{b-1}}{b^{n-1}}\, dx \xrightarrow{n\to\infty} 1.5 + \frac{b-1}{2} + \frac{1}{b-1} \geq 1.5 + \sqrt{2} \approx 2.91.$$

The inequality is tight at $b = 1 + \sqrt{2}$. With large n, since almost all budgets are consumed by the last few networks, we know the overall expectation $E_{B\sim Uniform(0,L)}[C]$ approaches $1.5 + \frac{b-1}{2} + \frac{1}{b-1}$, which is at least $1.5 + \sqrt{2}$.



(a) n=7 (b) n=9 (c) n=13

Figure 6: Test error versus cost curves with various periods on CIFAR100. Steps drop at new predictions.

B Wall-Clock Speed of ANNs

For reference, on a single GTX 1080 Ti, our implementation of ResNet+AANNs using Tensorflow (Abadi & et al., 2015) has the following FLOPS cost and test-time wall-clock speed on 224x224 images: 3.1e9 FLOPS and 690 FPS for ResNet14, 8.4e9 FLOPS and 310 FPS for ResNet50, and 15.9e9 FLOPS and 190 FPS for ResNet101.

C ResNet Residual Blocks Details

For CIFAR10, CIFAR100, and SVHN, we first apply a 3x3 convolution of c output channels with no stride on the input image, where c is specified by the model name (n, c). The initial conv layer is followed by three groups of residual blocks and each group has n basic residual building blocks. A residual building block applies a transformation $\phi_i(x_i)$ on the input feature map x_i, and then add the transformation and the input together to form the next feature map, i.e., $x_{i+1} = x_i + \phi_i(x_i)$. In a basic residual block with pre-activation, ϕ_i is BN-ReLU-3x3conv-BN-ReLU-3x3conv-BN. The two conv have the same channel size as the input within a group, and both have a stride of 1. In between two groups, the second conv of the previous residual block produces twice as many channels but use a stride of 2. The input of this block, i.e., x_{n-1} or x_{2n-1}, are also first average pooled and padded with zero channels before added to the second convolution result.

For ILSVRC, we first, apply a 7x7conv with 64 output channels and a stride of 2 on the input 224x224 image. Then we apply BN-ReLU and max-pooling with kernel size 3. In a bottleneck residual block (for ILSVRC), ϕ_i is BN-ReLU-1x1conv-BN-ReLU-3x3conv-BN-ReLU-1x1conv-BN. The first bottleneck residual block have its two conv to have 64 output channels, and the third conv has 64x4=256 channels. For other bottleneck block not at scale transition, the first 1x1 conv outputs has 1/4 of the channel size as its input; the 3x3conv maintains the same channel size as its input; the second 1x1 conv outputs four times the channel size as its input. At the scale transition, the first 1x1 conv outputs 1/2 of the channel size as its input; the other two conv have the same output channel size multiplier relative to their inputs as before; the 3x3 conv, however, uses a stride of 2. The input of the transition residual blocks are again first pooled and padded with zero channels before adding to the result of ϕ_i.

D Frequency of Anytime Prediction

We note that not every residual bottleneck unit is required to compute an anytime output, and it is possible that allowing more layers to be computed without interruption can lead to better predictions. This section studies the relationship between frequency and performance of anytime predictions. We expect that a model with more frequent anytime predictions to perform worse, because optimizing the extra auxiliary losses limits the freedom of the network. Additionally, with more anytime outputs, the average weight of anytime losses decreases, assuming the total loss weight is a constant. Then the less relative weight a layer has, the less optimized the layer will be in general. Following (Zamir et al., 2017), we call an ANN that makes an anytime prediction every i units of feature transformations (residual bottlenecks) as *stack-i* (i.e., the prediction period is i). The layers that predict bottlenecks whose index can be written as $L - ci$, where c is a non-negative integer, so that the last layer always predict. Fig. 6 and Table 2 demonstrate the performances of ANNs on CIFAR100 with different stack-i on networks with $n = 7, 9, 13$. In all cases all anytime predictions are able to achieve near-optimal final predictions due to the final large weights from the sieve scheme. From

Metric	s1	s2	s3	s5
1/4	68.58	**66.48**	72.35	74.29
1/2	49.94	51.81	**45.48**	70.51
3/4	35.78	38.33	35.32	**33.80**
1	**28.86**	29.31	29.28	29.30

(a) Anytime error rates for n=7

Metric	s1	s2	s3	s5	s7
1/4	74.03	73.86	68.97	79.06	**67.70**
1/2	46.02	46.00	50.26	46.55	**39.59**
3/4	35.79	35.21	40.12	39.37	**32.50**
1	28.44	28.36	28.74	**28.23**	28.55

(b) Anytime error rates for n=9

Metric	s1	s2	s3	s4	s5	s8	s10
1/4	69.92	68.25	68.27	64.45	65.63	**61.12**	62.65
1/2	42.87	42.70	42.74	44.04	38.18	48.36	**37.19**
3/4	33.28	33.46	34.17	36.59	**29.95**	37.10	30.23
1	26.96	**26.93**	27.27	27.19	27.29	27.20	27.11

(c) Anytime error rates for n=13

Table 2: Tables of anytime performance error rates for various prediction period s on various models ($n = 7, 9$ and 13) on CIFAR100. The best of each metric (each row) is in bold.

the slightly larger network $n = 9$ and $n = 13$, we observe the general trend of improvement of anytime performances in all metrics as the period s increases. However, such increase is not without a limit. For instance, when $n = 7$, if have a relatively large period, $s = 5$, then it is possible that a milestone cost ($\frac{1}{2}$ in this case) is right before a new prediction, so that the anytime performance at such a milestone suffers. In applications, we should choose large period s as long as it does not cause much waste of budgets right before milestones.

E Full Table for AANN with various weight schemes

Table 3 displays the performance of AANNs with each weight schemes on the 13 models and 3 data-sets listed in Sec. 4. All models have $s = 1$, except for $(25, 16)$ and $(25, 32)$, which have $s = 3$.

F No-regret Online learners for Choosing Layers to Amplify

F.1 Learn to Choose Layers via Min-max Optimization

This section consider to replace the heuristic and static strategy in AANN with strategies that are learned during training. Inspired by Shalev-Shwartz & Wexler (2016), we choose B_i to amplify in each iteration by applying no-regret online learner on the maximization of the following min-max optimization:

$$\min_{\theta} \max_{v \in \Delta^L} \sum_{i=1}^{L} v_i \mathcal{L}_i(\theta), \tag{1}$$

where Δ^L is the L-dimensional probability simplex, and $\mathcal{L}_i(\theta)$ is a heuristic loss objective of choosing layer i at parameter θ. Note that $\mathcal{L}$ may not be the same as the loss of layer i, ℓ_i. Intuitively, this min-max optimization can be considered as a two player zero-sum game, where the max player chooses layer i using v to generate the maximal loss, and the min player updates θ to reduce the chosen loss. As suggested by Shalev-Shwartz & Wexler (2016), applying no-regret online learner on the maximization w.r.t. v leads to a no-regret strategy of choosing which layers against any static strategies. There are multiple options for no-regret online learners. For instance, EXP3(Auer et al., 2002) and Random Weighted Majority (RWM)(Littlestone & Warmuth, 1994) both exponential gradient ascend on variable v to find layers that consistently results in high losses. Ideally, these no-regret online learners will discover high loss layer for optimization to focus on, so that eventually all loss $\mathcal{L}_i$ will be equal. We also consider another heuristic sampling strategy called *AVGL*: we sample layer i with probability proportional to the exponential average of $\mathcal{L}_i$.

F.2 Maximization Objective

Now we explain our choice of heuristic loss objective $\mathcal{L}_i$ so that minimization of Eq. 1 leads to low loss ℓ_i. Ideally, we would like to set $\mathcal{L}_i = \max(\frac{\ell_i - \ell_i^*}{\ell_i^*}, 0)$ to measure the relative difference

	CIFAR10				CIFAR100				SVHN			
Method	1/4	1/2	3/4	1	1/4	1/2	3/4	1	1/4	1/2	3/4	1
7-16 OPT	17.52	9.91	7.17	6.54	56.82	37.63	31.63	29.49	4.48	2.59	2.08	1.81
7-16 CONST	21.36	12.24	10.77	9.30	64.19	41.35	34.83	32.36	5.02	2.98	2.59	2.25
7-16 LINEAR	25.64	11.47	9.78	8.52	66.39	43.22	34.54	31.92	5.45	2.88	2.36	2.13
7-16 EXP-LIN	26.25	14.34	9.93	7.24	69.39	47.52	38.29	30.16	5.91	3.45	2.61	1.90
7-16 SIEVE	24.69	12.52	9.58	7.47	69.56	47.16	38.10	29.92	-	-	-	-
7-32 OPT	12.16	7.46	5.82	5.26	42.52	31.11	25.81	24.80	3.57	2.09	1.79	1.84
7-32 CONST	13.27	8.82	7.78	6.94	46.37	33.32	29.98	27.95	3.33	2.20	2.03	2.06
7-32 LINEAR	15.85	8.84	7.59	6.75	51.73	32.57	28.54	26.36	-	-	-	-
7-32 EXP-LIN	17.16	8.62	7.11	5.99	55.18	36.19	30.10	24.71	4.29	2.60	2.11	1.83
7-32 SIEVE	-	-	-	-	54.95	35.52	29.64	24.57	4.01	2.77	2.22	1.96
9-16 OPT	15.95	9.35	7.35	6.13	53.16	36.55	31.35	28.15	4.09	2.27	2.09	1.85
9-16 CONST	20.38	11.07	9.77	8.89	62.55	40.94	33.83	31.69	4.41	2.85	2.48	2.24
9-16 LINEAR	23.72	10.55	8.82	8.32	65.98	40.63	33.06	30.89	4.90	2.57	2.15	2.04
9-16 EXP-LIN	24.82	12.28	9.46	7.38	66.06	44.75	35.19	29.53	4.75	3.08	2.21	1.82
9-16 SIEVE	25.85	11.88	9.15	7.29	68.63	45.25	37.08	29.22	5.05	3.39	2.64	2.02
9-32 OPT	11.36	6.76	5.78	5.44	40.09	29.62	25.47	23.63	3.22	1.92	1.87	1.71
9-32 CONST	12.64	7.81	7.45	6.75	45.61	31.46	28.52	27.15	3.22	2.23	2.15	2.14
9-32 LINEAR	14.29	7.55	6.73	6.24	48.82	31.39	27.38	25.70	3.43	2.17	1.90	1.89
9-32 EXP-LIN	15.61	8.35	6.51	5.61	50.27	33.66	27.57	23.74	3.32	2.35	2.00	1.84
9-32 SIEVE	16.10	8.55	6.79	5.64	53.02	35.20	29.02	24.26	3.52	2.46	2.22	1.86
9-64 OPT	8.76	5.51	5.00	5.41	33.23	24.64	22.25	21.84	2.38	1.84	1.84	1.81
9-64 CONST	9.68	6.72	6.31	5.84	35.23	26.61	24.73	23.81	2.66	1.87	1.85	1.82
9-64 LINEAR	9.73	6.30	5.69	5.35	38.97	26.79	24.23	23.25	-	-	-	-
9-64 EXP-LIN	10.73	6.38	5.49	4.96	40.55	27.63	23.89	21.46	2.92	2.05	1.80	1.74
9-64 SIEVE	12.45	6.43	6.00	5.23	41.86	27.91	24.75	21.13	2.87	2.08	1.76	1.64
9-128 OPT	7.06	5.32	4.84	5.73	27.71	22.17	21.14	22.07	-	1.92	1.79	1.70
9-128 CONST	7.26	5.39	5.05	4.77	30.25	24.31	22.77	22.01	2.38	1.91	1.92	1.89
9-128 LINEAR	8.24	5.34	4.94	4.50	32.63	23.36	21.64	21.25	2.53	1.97	1.96	1.91
9-128 EXP-LIN	9.31	6.24	4.78	4.41	34.25	24.00	21.53	19.77	2.59	1.93	1.76	1.71
9-128 SIEVE	8.48	5.79	4.96	4.08	34.32	24.55	22.18	19.82	2.67	2.10	1.91	1.72
13-16 OPT	14.15	8.33	6.40	5.67	51.16	35.05	29.99	26.20	3.61	2.01	1.97	1.78
13-16 CONST	18.62	10.13	9.07	8.80	61.56	37.11	31.98	30.68	4.07	2.52	2.17	2.10
13-16 LINEAR	22.44	9.78	8.15	7.70	65.90	37.94	31.55	29.91	4.55	2.52	2.09	2.05
13-16 EXP-LIN	22.76	10.45	8.05	6.82	64.96	42.69	33.01	27.71	4.31	2.65	2.13	1.92
13-16 SIEVE	26.47	11.04	8.40	6.51	66.44	43.55	34.56	28.08	4.59	2.85	2.41	1.99
13-32 OPT	9.75	6.27	5.23	4.96	38.65	28.11	24.01	23.46	2.85	1.86	1.68	1.74
13-32 CONST	11.19	7.72	7.03	6.78	44.99	29.92	26.71	25.96	2.88	1.98	1.85	1.84
13-32 LINEAR	13.73	7.62	6.63	6.43	48.83	30.00	25.87	24.96	3.17	2.12	1.99	2.00
13-32 EXP-LIN	15.00	7.66	6.21	5.45	51.24	34.07	27.57	23.67	3.26	2.17	1.83	1.73
13-32 SIEVE	15.86	7.81	6.47	5.50	53.64	33.58	27.79	23.57	3.17	2.33	1.97	1.82
13-64 OPT	7.59	5.28	4.58	4.71	31.80	23.55	21.94	21.13	2.37	1.72	1.71	1.76
13-64 CONST	8.67	6.45	5.86	5.68	35.09	26.27	24.03	23.60	2.57	1.90	1.87	1.92
13-64 LINEAR	9.20	5.88	5.22	5.16	36.24	25.77	23.60	23.13	2.64	1.83	1.81	1.81
13-64 EXP-LIN	10.15	6.26	5.34	4.73	39.92	26.52	23.65	21.28	-	-	-	-
13-64 SIEVE	11.27	6.65	5.49	4.95	40.91	27.76	23.77	20.99	2.77	2.13	1.73	1.69
17-16 OPT	13.12	7.51	5.98	5.71	49.73	33.78	28.54	26.66	2.83	1.82	1.83	1.81
17-16 CONST	16.47	9.76	8.46	8.38	59.81	36.54	30.83	29.90	3.87	2.28	1.97	1.98
17-16 LINEAR	19.59	9.82	8.37	8.17	64.21	37.12	30.72	28.72	3.92	2.23	1.94	1.95
17-16 EXP-LIN	22.02	9.99	7.58	6.56	64.05	40.88	32.41	28.03	4.15	2.55	1.99	1.95
17-16 SIEVE	24.32	10.16	7.46	6.68	66.40	44.55	34.45	27.27	4.28	2.86	2.13	1.84
17-32 OPT	9.47	5.63	5.11	4.74	37.57	27.03	23.93	22.80	2.55	1.76	1.81	1.76
17-32 CONST	11.24	7.57	6.90	6.71	45.02	29.49	26.88	25.87	2.94	2.05	1.97	1.99
17-32 LINEAR	12.11	7.03	6.08	5.91	47.76	29.15	25.51	24.66	3.00	1.93	1.78	1.81
17-32 EXP-LIN	14.30	7.66	5.99	5.27	48.83	31.06	25.65	23.16	3.11	2.11	1.89	1.88
17-32 SIEVE	-	-	-	-	51.99	33.27	27.12	23.14	-	-	-	-
25-16 OPT	12.63	6.79	5.70	5.80	47.36	33.46	27.31	25.51	3.00	1.82	1.82	1.68
25-16 CONST	15.44	9.41	8.14	8.02	55.50	35.28	31.08	29.24	3.62	2.21	1.91	1.94
25-16 LINEAR	17.08	8.27	7.36	7.22	62.86	35.22	29.47	28.26	3.85	2.16	1.96	1.96
25-16 EXP-LIN	21.07	9.27	7.13	6.43	63.72	40.47	30.98	26.32	4.09	2.33	1.83	1.73
25-16 SIEVE	22.41	9.34	7.01	6.37	65.41	37.90	30.51	26.07	4.30	2.46	1.81	1.77
25-32 OPT	8.24	5.42	5.00	5.17	36.28	25.93	22.66	22.93	2.31	1.66	1.70	1.77
25-32 CONST	10.71	6.91	6.19	6.12	41.07	28.44	26.10	25.10	2.68	1.97	1.93	1.98
25-32 LINEAR	12.69	6.71	6.10	5.99	46.01	27.69	24.92	24.02	3.03	1.83	1.76	1.76
25-32 EXP-LIN	14.46	7.83	5.77	5.11	50.45	29.82	26.09	22.60	3.03	2.25	1.91	1.77
25-32 SIEVE	14.74	6.94	5.74	5.06	53.64	30.99	25.67	22.54	3.11	2.05	1.79	1.81

Table 3: Full table for AANNs of various weight schemes on various network models for data-sets SVHN, CIFAR10, and CIFAR100.

Method	1/4	1/2	3/4	1
EXP3	31.58	48.68	71.05	47.37
Milstone	69.44	47.22	50.00	15.28
RWM	38.24	47.06	61.76	52.94
AVGL	39.47	53.95	55.26	27.63
AANN	66.67	67.95	76.92	55.13

Table 4: Percentages of the total experiments that the listed methods are better than the vanilla ANN.

of ℓ_i from its optimal ℓ_i^*. In fact Lee et al. (2015) apply intermediate loss of this very form, and set the ℓ_i^* to be a hyper parameter to tune. However, in general tuning this parameter is difficult, and we cannot compute ℓ_i^* efficiently without training a model specifically for each i. To avoid the dependency on ℓ_i^*, we consider the relative loss difference between neighboring layers, and use the following min-max optimization to select losses ℓ_i to optimize,

$$\min_{\theta} \max_{v \in \Delta^L} \sum_{i=1}^{L-1} v_i \frac{\ell_i(\theta) - \ell_{i+1}(\theta)}{\ell_i(\theta)} + \eta v_L \max_{i=1,\ldots,L-1} \frac{\ell_i(\theta) - \ell_{i+1}(\theta)}{\ell_i(\theta)}, \tag{2}$$

where η is a constant chosen by cross-validation. Intuitively, for $i < L$, $\mathcal{L}_i(\theta) = \frac{\ell_i(\theta) - \ell_{i+1}(\theta)}{\ell_i(\theta)}$ represents the relative reduction in loss from layer i to $i+1$, and if this value is high, layer i is performing much worse than its successor while their structures are relative the same. This suggests layer i could be improved significantly with more optimization focus. The final loss objective is set to be $\mathcal{L}_L = \eta \max_{i=1,\ldots,L-1} \frac{\ell_i(\theta) - \ell_{i+1}(\theta)}{\ell_i(\theta)}$, a constant fraction of the maximal $\mathcal{L}_i$ of the previous $L-1$ layers, because the final layer does not have a successor and we desire a relative high probability of choosing the final layer.

F.3 No-regret Online Learning Algorithms

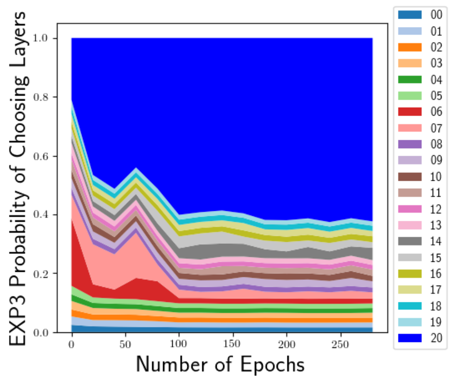
As suggested by Shalev-Shwartz & Wexler (2016), we let the max player use EXP3, a no-regret online learning algorithm, to update v in the min-max optimization in Eq. 1 as follows. In each iteration, we first sample a loss ℓ_i using EXP3 according to the probability distribution $p_i = (1-\gamma)v_i + \frac{\gamma}{L}$, where γ is a hyper parameter of EXP3 that represents the probability that the algorithm should explore a loss at random. Then the chosen ℓ_i is added to a fixed sum objective $\sum_{j=1}^L B_j \ell_j$ to form the total loss $\ell_{total} = \sum_{j=1}^L B_j \ell_j + \tau \sum_{j=1}^L B_j \ell_i$, where τ is a hyper parameter. The extra weight of ℓ_i is $\tau \sum_{j=1}^L B_j$, a fraction of the total weight of the sum objective, so that the chosen ℓ_i has significant influence on the total loss. After computing ℓ_{total}, we next update network parameter θ gradient of ℓ_{total}. Finally, we apply EXP3 update rules to update v_i to be $v_i exp(\frac{\gamma \mathcal{L}_i(\theta)}{L p_i})$ and normalize vector v onto the probability simplex Δ^L. If we use Random Weighted Majority (RWM) (Littlestone & Warmuth, 1994) instead of EXP3, the procedure is almost identical: the only difference is that in each iteration we update all v_j, $j = 1, ..., L$, by exponential gradient descent using the signal $\mathcal{L}_j$.

F.4 Hyper Parameter Set-up

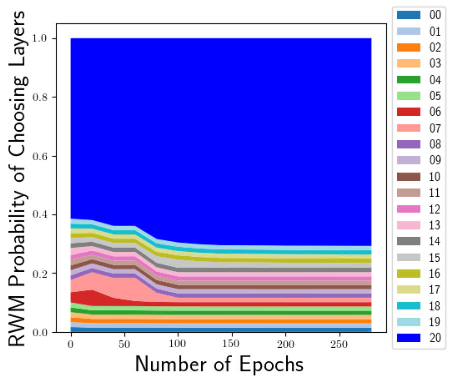
There are three hyper parameters for applying EXP3 or RWM to alternating optimization: a probability γ for the online learner to explore uniformly at random, a ratio η between the reward for the final layer and the maximum reward among previous layers, and the additional weight of the chosen layer as $\tau \sum_{i=1}^L B_i$. We run grid search on CIFAR10 and CIFAR100 validation network with γ from $\{0.1, 0.2, 0.3, 0.4\}$, η from $\{0.7, 0.8, 0.9, 1.0\}$, and τ from $\{2, 1, 0.5, 0.25\}$, and then sort the settings by average error and final error. We choose the setting that first appears on both ranking: $\gamma = 0.3$, $\eta = 0.8$, and $\tau = 0.5$. A true optimal parameter setting may be hard to determine due to the random nature of SGD and the randomized strategy itself, but luckily we found on validation set that the performance is not overly sensitive to parameters. We set the momentum constant in the exponential gradient average of the AVGL strategy to be 0.9

F.5 Experimental Results of Various Sampling Strategies

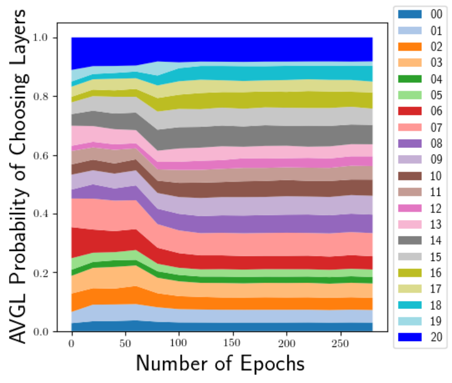
In Fig. 7, we plot the probability that each bottleneck unit i is chosen by various strategies on model with $n = 7$, and $c = 32$. The overall behavior of the probabilities are similar across models, even



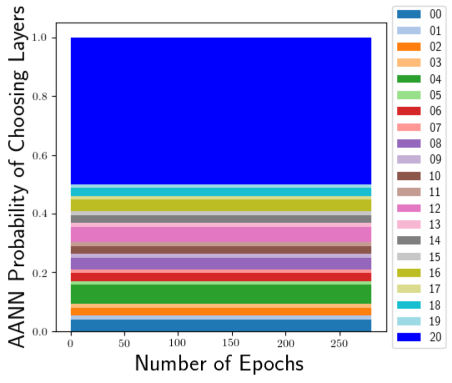
(a) EXP3 strategy over epochs



(b) RWM strategy over epochs



(c) AVGL strategy over epochs



(d) AANN strategy over epochs

Figure 7: Probability of each layer being chosen versus number of iterations using sampling strategy (a) EXP3, (b) RWM, (c) AVGL, and (d) AANN. (The stack plots are more visible with colors)

the performance ranking are not the same. We plot the stack graph versus the training epochs, so that we can view how the strategy evolve over time. We found that both no-regret algorithms, EXP3 and RWM, learned to pick the final layer with more than half of the probability. The remaining probability are roughly split evenly among layers. AVGL learns to put more weight on the final layer than the other layers as well, but since AVGL update its probabilities with gradient descent instead of exponentiated gradient descent in EXP3 and RWM, AVGL does not select the final layer with too high a probability. Moreover, AVGL learns to focus layers around layer 7 and 14, which are around the transition layers that subsample the feature maps and double the channel sizes via 1x1 convolutions. AANN based on sieve weight scheme naturally focus half of the weights on the final layer, and it does not spread the remaining weights evenly. Instead, we see intermittent large weights and small weights, so that every layer is somewhat focused on due to its correlation with its neighbors, and no neighborhood other than the final layer is overly focused on.

Table 4 lists for each sampling strategy and each anytime evaluation metric the percentage of the 39 experiments in Table 5 such that the corresponding strategy is better than the vanilla ANN in the corresponding metric. Missing experiments are ignored. We see that RWM degrades performance in every metric. EXP3 is only able to improve at the 3/4 milestone, and degrades performance at all other milestones. AVGL is able to somewhat improve performance at 1/2 and 3/4 milestones. However, this is at the cost of the final prediction performance, even though we note the difference is usually small. In contrast, AANN is able to improve performances in every metric, except in the final prediction, which cannot be improved because ANN is already at the optimal level in this metric. Overall, all of dynamic strategies, EXP3, RWM and AVGL are not as effective as the proposed simple AANN strategy that samples layers according to the normalized static weights, B_i. Since each of these dynamic strategies has extra parameters to tune, and the chosen heuristic loss objective $\mathcal{L}$ is probably sub-optimal, our experiments cannot fully prove that they are not helpful for improving performance of ANN. However, we can conclude that even if they can be helpful, they require extensive parameter tuning.

G Full Table for Evaluating Sampling Strategies

Table 5 lists the 13 different models that we experiment with sampling strategies on. The table also contains the performance of the sampling strategy *milestone*, which sample evenly at 1/4, 1/2, 3/4, and 1 of the total depths.

	SVHN					CIFAR10					CIFAR100				
Method	1/4	1/2	3/4	1	AE	1/4	1/2	3/4	1	AE	1/4	1/2	3/4	1	AE
OPT(n=25,c=16)	3.00	1.82	1.82	1.68		12.63	6.79	5.70	5.80		47.36	33.46	27.31	25.51	
ANN(n=25,c=16)	5.98	2.79	2.06	1.87	6.11	22.13	8.46	6.66	6.28	14.58	64.03	38.17	29.00	25.30	45.33
EXP3(n=25,c=16)	5.89	2.77	2.17	1.91	4.54	21.81	9.07	7.22	6.36	15.79	62.96	37.96	28.68	26.07	45.39
Milstone(n=25,c=16)	5.20	2.62	2.13	1.94	6.70	19.42	8.80	6.52	6.04	14.27	63.60	40.36	30.51	27.29	47.64
RWM(n=25,c=16)	5.92	2.74	2.29	2.02	5.23	19.42	8.47	6.52	6.08	14.52	63.49	36.18	28.20	25.94	45.11
AVGL(n=25,c=16)	5.55	2.72	2.25	2.11	5.48						64.88	37.06	29.46	25.97	45.62
AANN(n=25,c=16)	5.10	2.34	2.02	1.92	5.13	21.48	8.12	6.66	6.18	14.29	66.08	36.71	28.32	25.46	45.15
OPT(n=25,c=32)	2.31	1.66	1.70	1.77		8.24	5.42	5.00	5.17		36.28	25.93	22.66	22.93	
ANN(n=25,c=32)	3.70	2.11	1.81	1.77	3.41	14.44	8.05	6.77	5.33	11.23	51.59	31.16	25.81	22.96	37.76
EXP3(n=25,c=32)	3.64	1.85	1.71	1.66	2.87						44.85	28.21	23.94	22.40	35.71
Milstone(n=25,c=32)	3.35	2.04	1.71	1.70	3.31	14.04	8.12	6.00	5.70	11.12	53.63	33.40	25.92	23.97	40.04
RWM(n=25,c=32)	3.65	2.14	1.93	1.83	3.11	11.05	6.52	5.34	5.23	9.75	46.22	27.97	23.55	21.98	35.89
AVGL(n=25,c=32)	4.19	2.39	1.95	1.78	3.55	13.26	7.80	5.86	5.51	10.74	53.76	31.70	24.50	22.84	38.72
AANN(n=25,c=32)	3.55	1.94	1.84	1.71	3.26	13.73	6.31	4.93	4.65	9.84	48.72	28.05	23.46	22.00	35.96
OPT(n=17,c=16)	2.83	1.82	1.83	1.81		13.12	7.42	5.98	5.71		49.73	33.78	28.54	26.66	
ANN(n=17,c=16)	5.23	2.64	2.07	1.98	7.78	26.56	10.62	7.61	6.12	17.17	70.80	39.83	32.20	26.59	49.49
EXP3(n=17,c=16)	7.52	3.20	2.26	2.06	8.49	22.05	9.78	6.98	6.24	16.11	72.54	41.30	30.75	26.20	49.99
Milstone(n=17,c=16)	6.13	2.88	2.20	2.06	8.62	25.21	11.43	7.81	7.14	18.52	63.71	41.26	30.96	27.90	49.33
RWM(n=17,c=16)	8.43	3.15	2.28	1.93	8.74	27.60	10.45	7.38	6.01	17.60	76.08	48.69	32.57	27.03	53.18
AVGL(n=17,c=16)	6.56	3.44	2.32	2.11	8.03	28.74	10.20	7.50	6.36	18.81	71.20	40.40	30.14	26.75	49.74
AANN(n=17,c=16)	5.32	2.89	2.01	1.87	8.61	24.50	9.20	6.71	6.23	16.19	67.85	42.36	30.58	26.74	49.27
OPT(n=17,c=32)	2.55	1.76	1.81	1.71		9.47	5.63	5.11	4.73		37.57	27.03	23.93	22.80	
ANN(n=17,c=32)	4.56	2.61	2.04	1.84	5.88	18.04	7.23	5.98	5.33	12.41	47.53	32.03	27.70	23.92	38.89
EXP3(n=17,c=32)	5.19	2.51	2.00	1.81	6.27	20.20	8.52	5.86	5.19	14.95	58.11	34.55	26.06	22.93	42.96
Milstone(n=17,c=32)	4.06	2.45	2.05	1.86	6.06	15.48	7.39	6.25	5.60	13.59	53.64	32.59	26.29	24.08	41.98
RWM(n=17,c=32)						14.48	6.97	5.44	5.09	12.00	53.65	30.37	23.96	22.58	39.73
AVGL(n=17,c=32)	5.11	2.76	2.17	1.97	5.47	15.29	7.80	6.28	5.41	12.08	53.24	32.53	25.68	23.30	40.38
AANN(n=17,c=32)	3.80	2.32	1.89	1.80	6.23	14.43	6.77	5.34	5.00	11.50	53.33	31.17	24.00	22.29	39.95
OPT(n=13,c=16)	3.61	2.01	1.97	1.84		14.15	8.33	6.40	6.17		51.16	35.05	29.99	27.04	
ANN(n=13,c=16)	6.42	2.88	2.48	1.92	9.35	25.11	11.76	7.86	6.30	18.18	69.97	41.99	31.98	26.94	49.42
EXP3(n=13,c=16)	7.15	3.12	2.32	2.10	7.69	26.90	10.26	7.69	6.58	18.17	69.50	44.28	31.37	27.13	50.06
Milstone(n=13,c=16)	5.92	3.09	2.30	2.11	9.06	24.25	10.47	8.08	6.85	17.99	68.96	40.12	30.88	27.66	49.99
RWM(n=13,c=16)	7.23	3.40	2.57	2.12	8.20	25.58	11.74	7.79	6.43	18.01	71.35	42.62	31.40	27.40	50.44
AVGL(n=13,c=16)	7.28	3.44	2.39	2.05	8.46	27.26	10.63	7.53	6.78	17.99	69.08	42.59	31.29	27.61	49.59
AANN(n=13,c=16)	5.72	3.15	2.16	2.01	8.62	25.38	10.11	7.55	6.40	17.64	67.84	44.35	31.41	26.84	49.72
OPT(n=13,c=32)	2.85	1.86	1.68	1.67		9.75	6.27	5.23	4.96		38.65	28.11	24.01	23.29	
ANN(n=13,c=32)	4.28	2.75	2.06	1.82	6.79	15.47	7.92	6.23	5.40	12.92	50.62	34.82	26.56	23.60	39.49
EXP3(n=13,c=32)	4.74	2.62	2.26	1.89	5.42	20.97	7.58	5.96	5.35	13.85	56.78	33.96	26.15	23.37	41.67
Milstone(n=13,c=32)	4.10	2.68	2.17	2.01	7.12	14.37	8.52	6.76	6.25	12.96	51.71	34.30	27.11	24.46	41.59
RWM(n=13,c=32)	4.92	2.79	2.18	1.94	5.49	19.90	9.91	6.54	5.62	14.16	55.16	32.03	25.16	22.55	40.53
AVGL(n=13,c=32)	4.27	2.55	2.07	1.86	5.40	15.89	8.21	6.57	5.91	12.59	56.93	33.39	27.35	23.71	41.55
AANN(n=13,c=32)	4.05	2.43	1.93	1.77	7.45	15.77	7.01	5.57	5.37	12.70	53.95	31.61	25.19	23.03	40.38
OPT(n=13,c=64)	2.37	1.72	1.71	1.76		7.59	5.28	4.58	4.71		31.80	23.55	21.94	21.13	
ANN(n=13,c=64)	3.28	2.06	1.79	1.71	5.40	12.33	6.68	6.03	5.12	11.11	42.84	27.47	23.14	20.95	34.70
EXP3(n=13,c=64)	3.45	2.45	1.86	1.77	4.03	10.57	5.69	4.90	4.69	9.81	39.56	24.67	21.80	20.29	32.53
Milstone(n=13,c=64)	3.51	2.46	2.11	1.81	5.54	12.17	7.06	6.47	5.10	10.60	40.93	29.54	24.58	22.21	35.29
RWM(n=13,c=64)	3.14	1.92	1.78	1.69	3.52	11.34	5.77	4.62	4.37	10.06	42.32	25.57	21.94	20.55	33.63
AVGL(n=13,c=64)	3.58	2.11	2.02	1.84	4.60	14.84	6.68	5.77	4.93	10.62	39.60	28.40	24.30	21.67	34.00
AANN(n=13,c=64)	3.28	2.19	1.67	1.61	5.45	11.43	6.19	5.09	4.78	10.31	41.07	25.94	21.98	20.58	33.42
OPT(n=9,c=16)	4.09	2.27	2.09	1.85		15.95	9.35	7.35	6.13		53.16	36.55	31.35	28.15	
ANN(n=9,c=16)	6.66	3.42	2.40	1.94	10.26	27.90	11.36	7.73	6.70	18.77	70.05	45.69	33.24	28.08	51.53
EXP3(n=9,c=16)	7.33	4.14	2.70	2.15	9.00	31.39	11.33	7.84	6.38	19.50	71.70	45.19	32.27	28.87	51.30
Milstone(n=9,c=16)	5.92	3.33	2.31	1.94	10.30	29.25	12.10	8.12	6.98	19.88	67.03	42.73	31.99	29.02	50.73
RWM(n=9,c=16)	8.13	3.83	2.56	2.18	10.04	30.79	12.35	7.99	6.53	20.14	72.23	45.43	33.00	27.98	51.62
AVGL(n=9,c=16)	6.63	3.47	2.49	2.07	8.49	29.44	11.05	7.88	7.05	18.32	70.07	43.35	32.05	28.76	50.66
AANN(n=9,c=16)	6.96	3.44	2.34	2.04	9.40	27.45	11.95	8.08	7.24	18.51	69.79	46.35	33.65	28.35	51.14
OPT(n=9,c=32)	3.22	1.92	1.87	1.71		11.36	6.76	5.78	5.44		40.09	29.62	25.47	23.63	
ANN(n=9,c=32)	4.43	2.89	1.96	1.72	6.94	17.02	8.40	6.24	5.73	13.17	58.55	34.40	27.61	24.05	42.11
EXP3(n=9,c=32)	4.77	2.89	2.08	1.89	5.60	17.99	8.60	6.80	5.45	14.45	58.66	34.01	26.02	23.04	41.96
Milstone(n=9,c=32)	4.62	2.57	2.01	1.87	7.80	17.03	8.75	6.75	5.93	13.77	54.68	33.70	26.88	23.96	42.19
RWM(n=9,c=32)	4.61	3.10	1.99	1.91	5.57	18.21	8.63	6.91	5.65	14.83	61.39	35.22	26.61	24.11	43.03
AVGL(n=9,c=32)	4.65	2.83	2.12	1.87	6.28	20.07	7.67	6.49	5.66	13.60	57.40	33.84	26.07	23.85	41.38
AANN(n=9,c=32)	4.28	2.65	1.99	1.83	7.27	18.30	8.20	6.15	5.45	13.72	54.20	33.86	26.14	23.48	41.10
OPT(n=9,c=64)	2.38	1.84	1.84	1.81		8.76	5.51	5.00	5.41		33.23	24.64	22.25	21.84	
ANN(n=9,c=64)	3.80	2.33	2.00	1.80	6.43	13.35	7.02	6.10	4.97	11.64	43.02	30.58	26.18	22.58	35.65
EXP3(n=9,c=64)	4.08	2.55	1.97	1.82	4.68	14.42	7.50	6.02	5.04	11.32	49.18	28.89	23.16	21.20	36.44
Milstone(n=9,c=64)	3.62	2.37	1.90	1.81	6.45	13.14	7.66	5.90	5.61	11.75	37.66	28.86	24.12	22.37	34.50
RWM(n=9,c=64)	3.53	2.17	1.82	1.74	3.91						42.78	26.66	22.40	21.08	34.26
AVGL(n=9,c=64)	3.70	2.37	1.89	1.77	5.22	10.55	6.45	5.37	4.87	10.09	44.91	29.35	23.65	21.88	36.37
AANN(n=9,c=64)	3.61	2.04	1.68	1.67	6.13	11.44	6.89	5.16	5.04	9.93	43.06	27.15	22.50	21.11	34.84
OPT(n=9,c=128)	2.38	1.92	1.79	1.70		7.06	5.32	4.84	5.33		27.71	22.17	21.14	22.07	
ANN(n=9,c=128)	3.11	2.05	1.76	1.73	5.26	9.10	5.50	4.77	4.53	8.57	34.97	26.14	23.74	20.25	31.99
EXP3(n=9,c=128)	2.92	1.83	1.71	1.68	3.37	8.32	5.53	4.60	4.32	7.92	34.38	23.07	20.29	19.94	29.75
Milstone(n=9,c=128)															
RWM(n=9,c=128)						9.12	5.55	4.52	4.45	7.99					
AVGL(n=9,c=128)	3.10	2.18	1.70	1.68	4.27	9.09	5.84	4.74	4.77	8.78	40.94	25.15	22.13	20.72	31.84
AANN(n=9,c=128)	3.11	2.30	1.76	1.73	5.25	9.97	5.53	4.59	4.41	8.57	34.18	24.55	21.70	20.45	30.00
OPT(n=7,c=16)	4.48	2.59	2.08	1.81		17.52	9.91	7.17	6.54		56.82	37.63	31.63	29.49	
ANN(n=7,c=16)	6.97	3.50	2.62	1.92	10.60	28.77	12.01	8.88	7.18	19.32	69.56	46.45	36.02	29.13	51.79
EXP3(n=7,c=16)	7.76	4.14	2.85	2.19	9.19	29.49	12.39	8.55	6.93	19.71	70.74	47.85	35.52	29.46	52.62
Milstone(n=7,c=16)	6.52	3.45	2.54	2.12	10.72	29.17	11.59	8.78	7.56	19.68	68.68	44.33	33.53	29.84	51.51
RWM(n=7,c=16)	8.04	3.53	2.77	2.09	9.15	30.68	12.83	8.67	7.08	19.82	71.60	46.60	35.27	29.41	52.83
AVGL(n=7,c=16)	7.04	3.68	2.79	2.09	9.30	29.71	11.55	8.18	6.79	18.93	68.91	46.00	34.10	29.47	51.29
AANN(n=7,c=16)	7.19	3.50	2.72	2.04	11.19	27.84	12.12	9.34	6.90	19.30	68.37	46.61	36.01	29.35	51.84
OPT(n=7,c=32)	3.57	2.09	1.79	1.84		12.16	7.46	5.82	5.26		42.52	31.11	25.81	24.80	
ANN(n=7,c=32)	4.66	2.90	2.18	1.85	7.71	17.69	8.58	7.02	5.26	13.43	54.18	34.81	28.09	23.96	41.77
EXP3(n=7,c=32)	5.02	3.15	2.35	1.91	6.30	18.67	8.96	6.81	5.58	14.57	58.87	36.40	27.99	24.62	43.54
Milstone(n=7,c=32)	4.76	2.88	2.25	1.95	8.32	18.59	8.92	6.81	6.33	15.09	53.84	33.23	27.56	23.97	41.48
RWM(n=7,c=32)	4.65	2.79	2.29	2.08	5.90	19.68	9.23	6.91	5.61	14.46					
AVGL(n=7,c=32)	4.62	2.52	2.23	1.85	6.51	16.63	8.21	6.64	5.96	13.65	55.84	34.22	27.76	24.70	41.62
AANN(n=7,c=32)	4.41	2.74	2.29	1.93	8.52	17.23	8.35	6.74	5.63	14.22	54.19	34.66	28.12	24.72	41.75

Table 5: Full table for evaluation of ANN and AANN along with other sampling strategies.