Adaptive Neural Networks for Effective Inference

Tolga Bolukbasi Joseph Wang Ofer Dekel Venkatesh Saligrama

Proceedings of the 34th International Conference on Machine Learning, PMLR 2017

Presenter: Chao Jiang

Outline

Outline

Motivation

Motivation:

- Deep neural networks (DNNs) are among the most powerful and versatile machine learning techniques, achieving state-of-the-art accuracy in a variety of important applications.
- However, as the networks become more complex, computational cost of applying them to new examples also grows higher.
- Test-time cost, has increased rapidly for many tasks with ever-growing demands for improved performance in state-of-the-art systems

Performance versus Evaluation Complexity

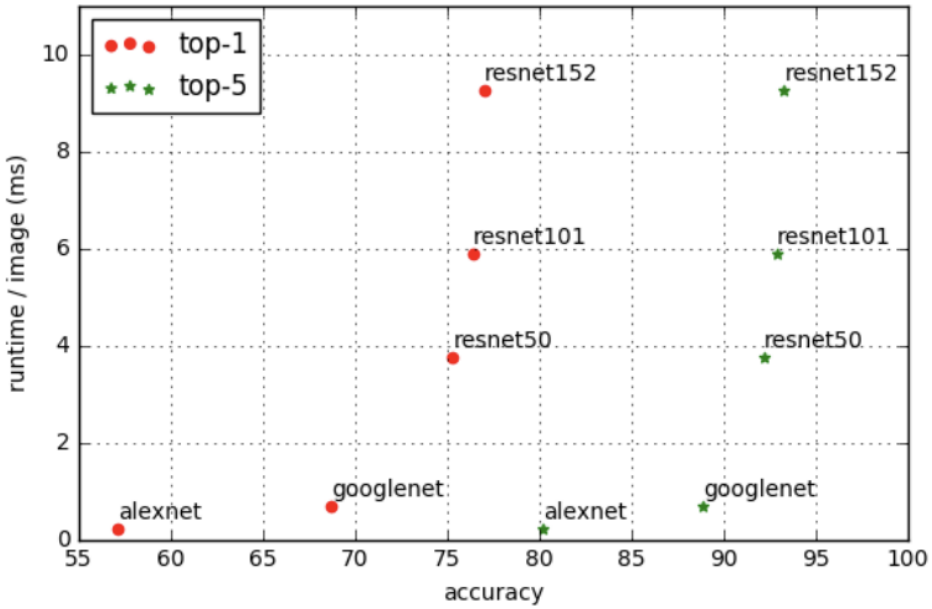


Figure 1. Performance versus evaluation complexity of the DNN architectures that won the ImageNet challenge over past several years. The model evaluation times increase exponentially with respect to the increase in accuracy.

- The Resnet152 architecture with 152 layers, realizes a substantial 4.4% accuracy gain in top-5 performance over GoogLeNet on the large-scale ImageNet dataset but is about 14X slower at test-time.

Facts

The facts about the data:

- Natural data is typically a mix of easy examples and difficult examples.
- The easy examples do not require the full power and complexity of a massive DNN.
- Namely, easy examples could be correctly classified by using either a simple network or the early layers in a complex network.

Outline

Problem Setting and Previous Solution

Problem Setting:

- Target: Shorten the test-time cost without losing much accuracy.

Previous Solution:

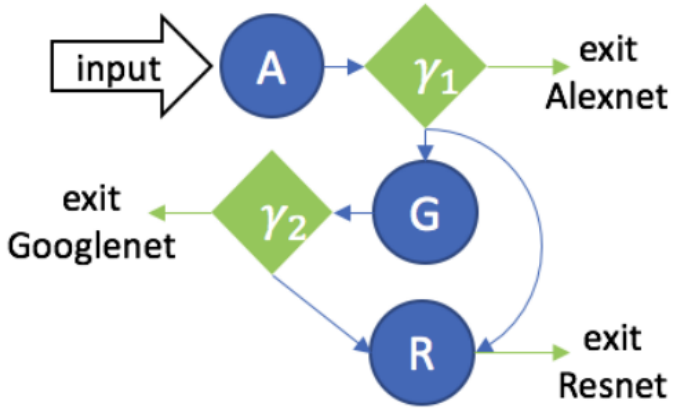
- Most of the work on this topic focuses on designing more efficient network topologies and on compressing pre-trained models using various techniques.

Outline

Contributions

An adaptive early-exit strategy

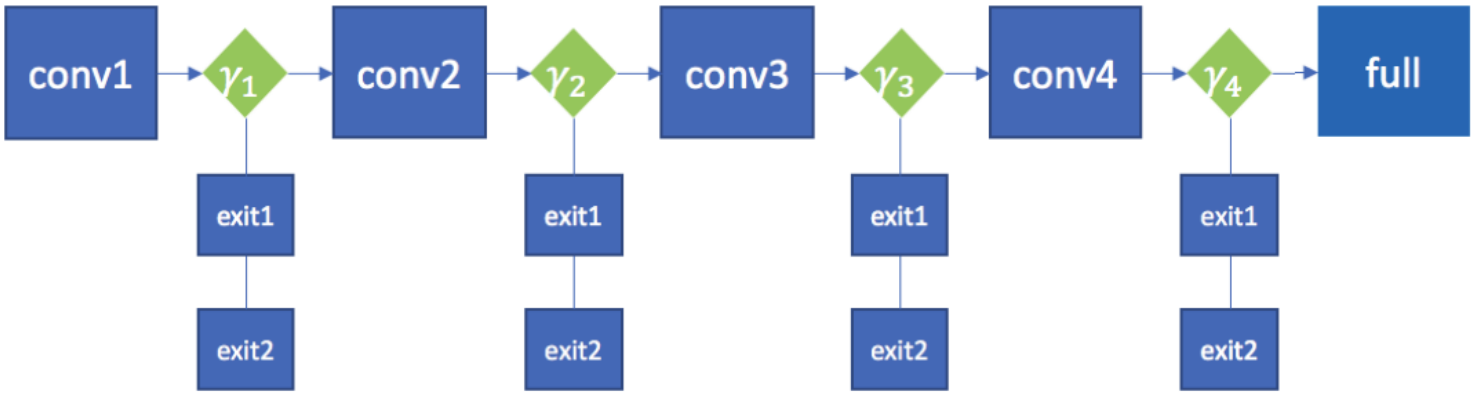
- Takes a set of pre-trained DNNs, each with a different cost/accuracy trade-off, and arranges them in a directed acyclic graph, with the the cheapest model first and the most expensive one last.
- They then train an exit policy at each node in the graph, which determines whether they should rely on the current models predictions or predict the most beneficial next branch to forward the example to.



Contributions

An adaptive early-exit strategy

- Allows easy examples to bypass some of the networks layers.
- Before each expensive neural network layer (e.g., convolutional layers), they train a policy that determines whether the current example should proceed to the next layer, or be diverted to a simple classifier for immediate classification.



Contributions

The whole structure overview

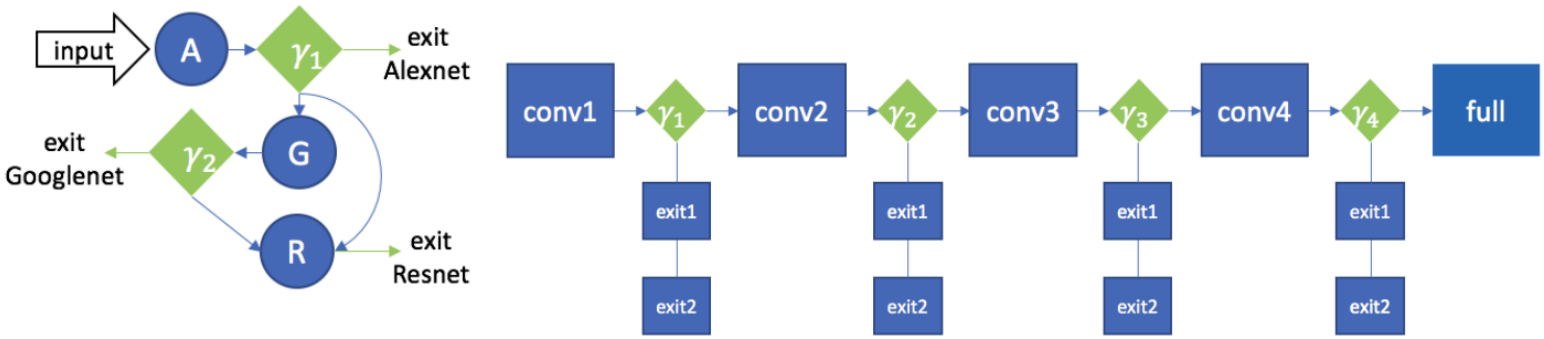


Figure 2. **(Left)** An example network selection system topology for networks Alexnet(A), GoogLeNet(G) and Resnet(R). Green γ blocks denote the selection policy. The policy evaluates Alexnet, receives confidence feedback and decides to jump directly to Resnet or send the sample to GoogLeNet->Resnet cascade. **(Right)** An example early exit system topology (based on Alexnet). The policy chooses one of the multiple exits available to it at each stage for feedback. If the sample is easy enough, the system sends it down to exit, otherwise it sends the sample to the next layer.

A sqeuence of papers

- Cost-sensitive learning by cost-proportionate example weighting
- Multiclass classification with filter trees
- Efficient learning by directed acyclic graph for resource constrained prediction

Importance-weighted binary classification problem (WBC)

Importance-weighted binary classification problem (WBC)

- Binary classification where each example has an associated weight specifying how important it is to predict its label correctly.
- (x, y, w), Defined by a distribution D on $X \times \{0, 1\} \times [0, \text{inf})$
- Loss function $E_{(x,y,w)\sim D}[w \infty(b(x) \neq y)]$, $\infty(\cdot)$ is 1 when the argument is true and 0 otherwise.

Cost-sensitive k-class classification problem (CSL)

Cost-sensitive k-class classification problem (CSL)

- $(w, \vec{c})$Defined by a distribution D over $X \times [o, \inf)^k$.
- The goal is to find a classifier $h : X \to \{1, \cdots, k\}$ minimizing the expected cost $e(h, D) = E_{(x,\vec{c})\sim D}[c_{h(x)}]$
- Here, $\vec{c} \in [o, \inf)^k$ gives the cost of each of the k choices for x.

Convert CSL to WBC - The Filter Tree Algorithm

Algorithm 1 The filter tree training algorithm

Filter-Train (cost-sensitive training set S, importance-weighted binary learner Learn)

1. Fix a binary tree T over the labels.
2. For each internal node n in the order from leaves to roots:
   (a) For each example $(x, c_1, ..., c_k) \in S$
      i. $S_n = \emptyset$
      ii. Let a and b be the two classes input to n (for internal nodes, these are the predictions of the left and the right subtrees on input x).
      iii. $S_n \leftarrow S_n \cup \{(x, \arg\min\{c_a, c_b\}, |c_a - c_b|)\}$
   (b) Let $\text{predict}_n = \text{Learn}(S_n)$
3. return $\{\text{predict}_n\}$

Algorithm 2 The filter tree testing algorithm

Filter-Test (classifiers $\{\text{predict}_n\}$, test example $x \in X$)

Output the label l such that every classifier on the path from leaf l to the root prefers l.

Convert CSL to WBC - The Filter Tree Algorithm

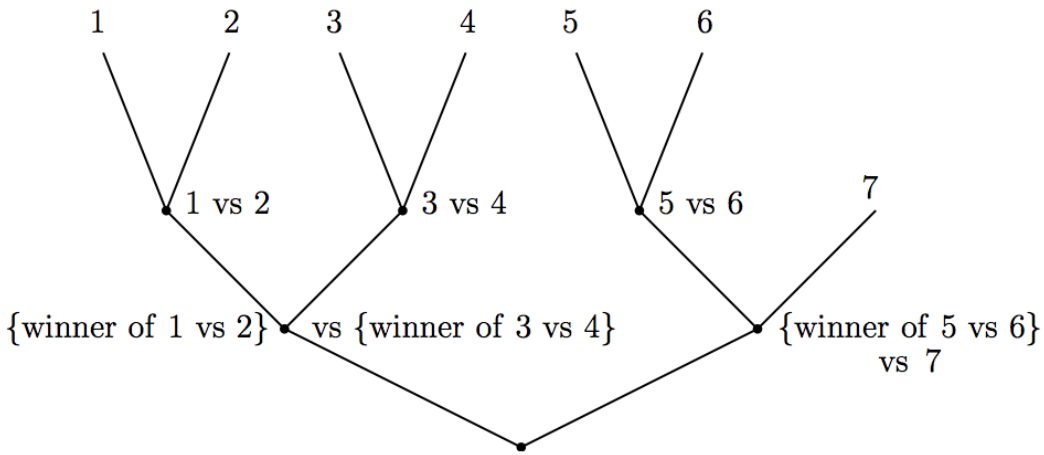


Figure 1: Filter Tree. Each node predicts whether the left or the right input label is more likely, conditioned on a given $x \in X$. The final output node predicts the best label for x.

- The training algorithm relies upon an importance weighted binary learning algorithm, which takes examples of the form (x, y, w), where x is a feature vector used for prediction, y is a binary label, and w is a positive real-valued importance.
- Importance-weighted binary classification can be further reduced to binary classification using the Costing reduction or other methods.

Outline

Notation

- Labeled example $(x, y) \in R^d \times \{1, \cdots, L\}$, where d is the dimension of the data and $\{1, \cdots, L\}$ is the set of classes representation.
- $X \times Y$: the distribution generating the examples.
- For a predicted label $\hat{y}$, they denote the loss $L(\hat{y}, y)$. In this paper, they foucus on indicator loss $L(\hat{y}, y) = \infty_{\hat{y}=y}$.

Notation

- Using AlexNet which is composed of 5 convolutional layers followed 3 fully connected layers as an example, during evaluation of the network, computing each convolutional layer takes more than 3 times longer than computing a fully connected layer
- they consider a system that allows an example to exit the network after each of the first 4 convolutional layers.
- $\hat{y}(x)$ dentoes the label predicted by the network for example x and assume that computing this prediction takes a constant time of T.
- $\sigma_k(x)$ denotes the output of the k-th convolutional layer for example x.
- t_k denotes the time it takes to compute this value (from the time that x is fed to the input layer).
- $\hat{y}_k(x)$ denotes the predicted label if they exit after the k-th layer.

Notation

- After computing the k-th convolutional layer, they introduce a decision function γ_k that determines whether the example should exit the network with a label of $\hat{y}_k(x)$ or proceed to the next layer for further evaluation.
- Input to γ_k is the output of the corresponding convolutional layer $\delta_k(x)$ and the value of $\gamma_k(\delta_k(x))$ is ether 1 or -1 (indicating an early exit).

Objective Function

Globally, our goal is to minimize the evaluation time of the network such that the error rate of the adaptive system is no more than some user-chosen value B greater than the full network:

$$\min_{\gamma_1,\ldots,\gamma_4} \mathbb{E}_{x\sim\mathcal{X}}\left[T_{\gamma_1,\ldots,\gamma_4}(x)\right]. \tag{1}$$

$$\text{s.t. } \mathbb{E}_{(x,y)\sim\mathcal{X}\times\mathcal{Y}}\left[\left(L(\hat{y}\gamma_1,\ldots,\gamma_4(x),y) - L(\hat{y}(x),y)\right)_+\right] \leq B$$

Here, $T_{\gamma_1,\ldots,\gamma_4}(x)$ is the prediction time for example x for the adaptive system, $\hat{y}\gamma_1,\ldots,\gamma_4(x)$ is the label predicted by the adaptive system for example x.

- The key idea is that, for each input, a policy must identify whether or not the future reward (expected future accuracy minus comp. loss) outweighs the current-stage accuracy.

Learning the decision function

Using γ_4 which determines if an example should exit after the fourth convolutional layer or whether it will be classified using the entire network as an example.
The time it takes to predict the label of example x depends on this decision and can be written as

$$T_4(x, \gamma_4) = \begin{cases} T + \tau(\gamma_4) & \text{if } \gamma_4(\sigma_4(x)) = 1 \\ t_4 + \tau(\gamma_4) & \text{otherwise} \end{cases}, \quad (2)$$

where $\tau(\gamma_4)$ is the computational time required to evaluate the function γ_4.

Learning the decision function

Their goal is to learn a system that trades-off the evaluation time and the induced error:

$$\underset{\gamma_4 \in \Gamma}{\operatorname{argmin}} \mathbb{E}_{x \sim \mathcal{X}} \left[T_4(x, \gamma_4) \right] + \lambda \mathbb{E}_{(x,y) \sim \mathcal{X} \times \mathcal{Y}} \Big[\Big(L\left(\hat{y}_4(x), y\right) - L\left(\hat{y}(x), y\right) \Big)_+ \mathbb{1}_{\gamma_4(\sigma_4(x))=-1} \Big] \tag{3}$$

where $(\cdot)_+$ is the function $(z)_+ = \max(z, 0)$ and $\lambda \in \mathbb{R}^+$ is a trade-off parameter that balances between evaluation time and error.

Learning the decision function

$$
\begin{aligned}
T_4\left(x, \gamma_4\right) = & \left(T + \tau(\gamma_4)\right) \mathbb{1}_{\gamma_4(\sigma_4(x))=1} \\
& + \left(t_4 + \tau(\gamma_4)\right) \mathbb{1}_{\gamma_4(\sigma_4(x))=-1} \\
= & T \mathbb{1}_{\gamma_4(\sigma_4(x))=1} + t_4 \mathbb{1}_{\gamma_4(\sigma_4(x))=-1} + \tau_4(\gamma_4)
\end{aligned}
$$

Substituting for $T_4(x, \gamma_4)$ allows us to reduce the problem to an importance weighted binary learning problem:

$$
\underset{\gamma_4 \in \Gamma}{\operatorname{argmin}} \mathbb{E}_{(x,y)\sim\mathcal{X}\times\mathcal{Y}} \left[C_4(x, y) \mathbb{1}_{\gamma_4(\sigma_4(x)) \neq \beta_4(x)}\right] + \tau(\gamma_4) \tag{4}
$$

Learning the decision function

where $\beta_4(x)$ and $C_4(x, y)$ are the optimal decision and cost at stage 4 for the example (x, y) defined:

$$\beta_4(x) = \begin{cases} -1 & \text{if } T > \left(t_4 + \lambda\left(L\left(\hat{y}_4(x), y\right) - L\left(\hat{y}(x), y\right)\right)_+\right) \\ 1 & \text{otherwise} \end{cases}$$

and

$$C_4(x, y) = \left|T - t_4 - \lambda\left(L\left(\hat{y}_4(x), y\right) - L\left(\hat{y}(x), y\right)\right)_+\right| .$$

Learning the decision function

For a general early exit system, we recursively define the future time, $T_k(x, \gamma_k)$, and the future predicted label, $\tilde{y}_k(x, \gamma_k)$, as

$$T_k(x, \gamma_k) = \begin{cases} T + \tau(\gamma_k) & \text{if } \gamma_k(\sigma_k(x)) = 1, k = K \\ T_{k+1}(x, \gamma_k & \text{if } \gamma_k(\sigma_k(x)) = 1, k < K \\ \quad +1) + \tau(\gamma_k) & \\ t_k + \tau(\gamma_k) & \text{otherwise} \end{cases}$$

and

$$\tilde{y}_k(x, \gamma_k) = \begin{cases} \hat{y}(x) & \text{if } k = K + 1 \\ \hat{y}(x) & \text{if } k = K \\ & \quad \text{and } \gamma_k(\sigma_k(x)) = 1 \\ \tilde{y}_{k+1}(x, \gamma_{k+1}) & \text{if } k < K \\ & \quad \text{and } \gamma_k(\sigma_k(x)) = -1 \\ \hat{y}_k(x) & \text{otherwise} \end{cases}.$$

I think this should be 1

Learning the decision function

For a system with K early exit functions, the k-th early exit function can be trained by solving the supervised learning problem:

$$\underset{\gamma_k \in \Gamma}{\operatorname{argmin}} \, \mathbb{E}_{(x,y)\sim\mathcal{X}\times\mathcal{Y}} \left[C_k(x,y) \mathbb{1}_{\gamma_k(x) \neq \beta_k(\sigma_k(x))} \right] + \tau(\gamma_k), \tag{5}$$

where optimal decision and cost $\beta_k(x)$ and $C_k(x,y)$ can be defined:

$$\beta_k(x) = \begin{cases} -1 & \text{if } k < K \text{ and } T_{k+1}(x, \gamma_{k+1}) \geq t_k + \\ & \lambda \left(L\left(\hat{y}_k(x), y\right) - L\left(\tilde{y}_{k+1}(x), y\right) \right)_+ \\ -1 & \text{if } k = K \text{ and } T \geq t_k + \\ & \lambda \left(L\left(\hat{y}_k(x), y\right) - L\left(\tilde{y}_{k+1}(x), y\right) \right)_+ \\ 1 & \text{otherwise} \end{cases}$$

$$C_k(x,y) = \begin{cases} \left| T_{k+1}(x, \gamma_{k+1}) - t_k \right| & \text{if } k < K \\ \left| -\lambda \Big(L\left(\hat{y}_k(x), y\right) - L\left(\tilde{y}_{k+1}(x), y\right) \Big)_+ \right| & \\ \left| T - t_k \right| & \text{otherwise} \\ \left| -\lambda \left(L\left(\hat{y}_k(x), y\right) - L\left(\hat{y}(x), y\right) \right)_+ \right| & \end{cases}.$$

Outline

Learning the decision function

They seek to exploit the fact:

- Many examples are correctly classified by relatively efficient networks such as alexnet and googlenet.
- Only a small fraction of examples are correctly classified by computationally expensive networks such as resnet 152 and incorrectly classified by googlenet and alexnet.

Notation

- N_1, N_2, N_3: three pre-trained networks
- For an example x, denote the predictions for the networks as $N_1(x), N_2(x), N_3(x)$.
- $\tau(N_1), \tau(N_2), \tau(N_3)$ denotes the evaluation times for each of the networks.

Notation

- $k_1 : |X| \rightarrow \{N_1, N_2, N_3\}$ is applied after evaluation of N_1 to determine if the classification decision from N_1 should be returned or if network N_2 or network N_3 should be evaluated for the example.
- For examples that are evaluated on N_2, $k_2 : |X| \rightarrow \{N_2, N_3\}$ determines if the classification decision from N_2 should be returned or if network N_3 should be evaluated.

Goal

- Their goal is to learn the functions k_1 and k_2 that minimize the average evaluation time subject to a constraint on the average loss induced by adaptive network selection.

Learn k_2

They first learn k_2 to trade-off between the average evaluation time and induced error

$$\min_{\kappa_2 \in \Gamma} \mathbb{E}_{x \sim \mathcal{X}} \left[\tau(N_3) \mathbb{1}_{\kappa_2(x)=N_3} \right] + \tau(\kappa_2)$$
$$+ \lambda \mathbb{E}_{(x,y) \sim \mathcal{X} \times \mathcal{Y}} \Bigg[\Big(L\left(N_2(x), y\right)$$
$$- L\left(N_3(x), y\right) \Big)_+ \mathbb{1}_{\kappa_2(x)=N_2} \Bigg], \qquad (6)$$

where $\lambda \in \mathbb{R}^+$ is a trade-off parameter.

Learn k_2

This problem can be posed as an importance weighted supervised learning problem

$$\min_{\kappa_2 \in \Gamma} \mathbb{E}_{(x,y)\sim\mathcal{X}\times\mathcal{Y}} \left[W_2(x,y) \mathbb{1}_{\kappa_2(x) \neq \theta_2(x)} \right] + \tau(\kappa_2), \quad (7)$$

where $\theta_2(x)$ and $W_2(x,y)$ are the cost and optimal decision at stage 4 for the example/label pair (x,y) defined:

$$\theta_2(x) = \begin{cases} N_2 & \text{if } \tau(N_3) > \lambda \left(L\left(N_3(x), y\right) - L\left(N_2(x), y\right) \right)_+ \\ N_3 & \text{otherwise} \end{cases}$$

and

$$W_2(x,y) = \left| \tau(N_3) - \lambda \left(L\left(N_2(x), y\right) - L\left(N_3(x), y\right) \right)_+ \right|.$$

Learn k_1

- Once k_2 has been trained, the training times for examples that pass through N_2 and are routed by k_2 can be defined
  $T_{K_2(x)} = \tau(N_2) + \tau(k_2) + \tau(N_3)\infty_{k_2(x)=N_3}$
- they train and fix the last decision function, k_2, then train the earlier function, k_1. As before, they seek to trade-off between evaluation time and error:

$$\min_{\kappa_1 \in \Gamma} \mathbb{E}_{x\sim\mathcal{X}} \left[\tau(N_3)\mathbb{1}_{\kappa_1(x)=N_3} + \tau(N_2)\mathbb{1}_{\kappa_1=N_2}\right] + \tau(\kappa_1) +$$

$$\lambda \mathbb{E}_{(x,y)\sim\mathcal{X}\times\mathcal{Y}} \Bigg[\left(L\left(N_2(x), y\right) - L\left(N_3(x), y\right)\right)_+ \mathbb{1}_{\kappa_1(x)=N_2}$$

$$+ \left(L\left(N_1(x), y\right) - L\left(N_3(x), y\right)\right)_+ \mathbb{1}_{\kappa_1(x)=N_1} \Bigg] \qquad (8)$$

Learn k_1

This can be reduced to a cost sensitive learning problem:

$$\min_{\kappa_1 \in \Gamma} \mathbb{E}_{(x,y)\sim\mathcal{X}\times\mathcal{Y}} \Big[R_3(x,y)\mathbb{1}_{\kappa_1(x)=N_3} + R_2(x,y)\mathbb{1}_{\kappa_1(x)=N_2} + R_1(x,y)\mathbb{1}_{\kappa_1(x)=N_1} \Big] + \tau(\kappa_1), \qquad (9)$$

where the costs are defined:

$$R_1(x,y) = \left(L(N_1(x),y) - L(N_3(x),y)\right)_+$$
$$R_2(x,y) = \left(L(N_2(x),y) - L(N_3(x),y)\right)_+ + \tau(N_2)$$
$$R_3(x,y) = \tau(N_3).$$

Learn k_1

Algorithm 1 Adaptive Network Learning Pseudocode

Input: Data: $(x_i, y_i)_{i=1}^n$,
Models $\mathcal{S}$, routes, E, model costs $\tau(.)$),
while $\exists$ untrained π **do**
(**1**) Choose the deepest policy decision j, s.t. all downstream policies are trained
for example $i \in \{1, \ldots, n\}$ **do**
(**2**) Construct the weight vector $\vec{w}_i$ of costs per action from Eqn. (7).
end for
I think there should be CSL
(**3**) $\pi_j \leftarrow$Learn clf.$((x_1, \vec{w}_1), \ldots, (x_n, \vec{w}_n))$
(**4**) Evaluate π_j and update route costs to model j:
$C(x_i, y_i, s_n, s_j) \leftarrow \vec{w}_i^j\ (\pi_j(x_i)) + C(x_i, y_i, s_n, s_j)$
end while
(**5**) Prune models the policy does not route any example to from the collection
Output: Policy functions, $\pi_1, \ldots, \pi_K$

Learn k_1 (From Previous paper)

A single iteration of Algorithm 1 proceeds as follows:

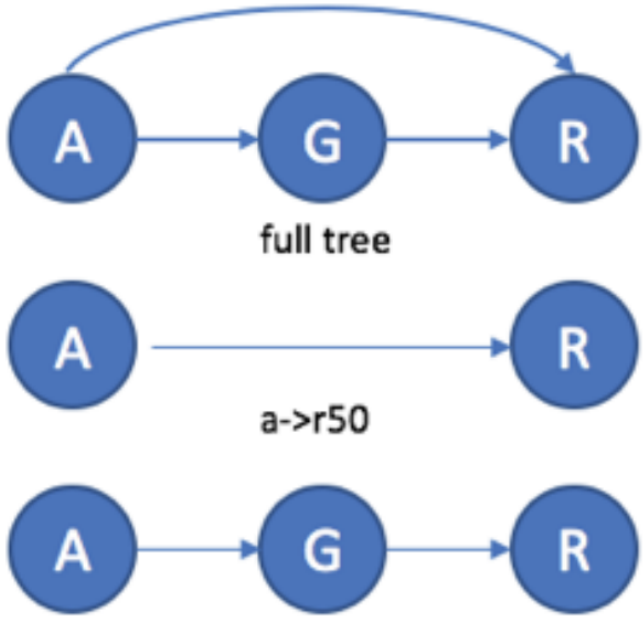
1. A node j is chosen whose outgoing edges connect only to leaf nodes.
2. For each example, the costs associated with each connected leaf node are found.
3. The policy ϕ_j is trained on the entire set of training data according to these costs by solving a CSL problem.
4. The costs associated with taking the action ϕ_j are computed for each example, and the costs of moving to state j are updated.
5. Outgoing edges from node j are removed (making it a leaf node).
6. disconnected nodes (that were previously connected to node j) are removed.

The algorithm iterates through these steps until all edges have been removed.

Outline

Network Selection

- Task: Imagenet 2012 classification
- Baselines, they compare against a uniform policy and a myopic policy which learns a single threshold based on model confidence.
- Report performance from different system topologies.
- The soft oracle has access to classification labels and sends each example to the fastest model that correctly classifies the example.



top5@	policy	myopic	uniform
@1	2.8X	1.9X	1.45X
@5	~ 8X	~ 5X	~ 6X

Network Selection

They sweep the cost trade-off parameter in the range 0.0 to 0.1 to achieve different budget points.

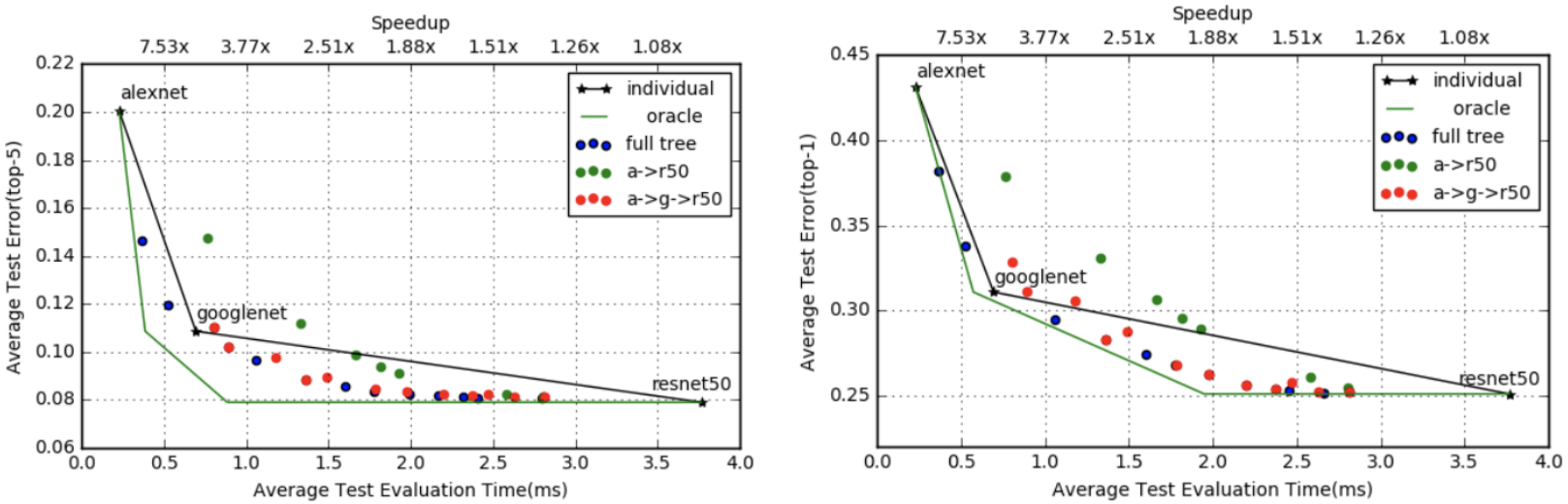


Figure 3. Performance of network selection policy on Imagenet (Left: top-5 error Right: top-1 error). Our full adaptive system (denoted with blue dots) significantly outperforms any individual network for almost all budget regions and is close to the performance of the oracle. The performances are reported on the validation set of ImageNet dataset.

Outline

Network Early Exits

Network	policy top-5	uniform top-5
GoogLeNet@1	9%	2%
GoogLeNet@2	22%	9%
GoogLeNet@5	33%	20%
Resnet50@1	8%	1%
Resnet50@2	18%	12%
Resnet50@5	22%	10%

Table 1. Early exit performances at different accuracy/budget trade-offs for different networks. @x denotes x loss from full model accuracy and reported numbers are percentage speed-ups.

Network Selection

- Accuracy gains per evaluation time for different layers.
- Interestingly, the accuracy gain per time is more linear within the same architecture compared to different network architectures.
- This explains why the adaptive policy works better for network selection compared to early exits.

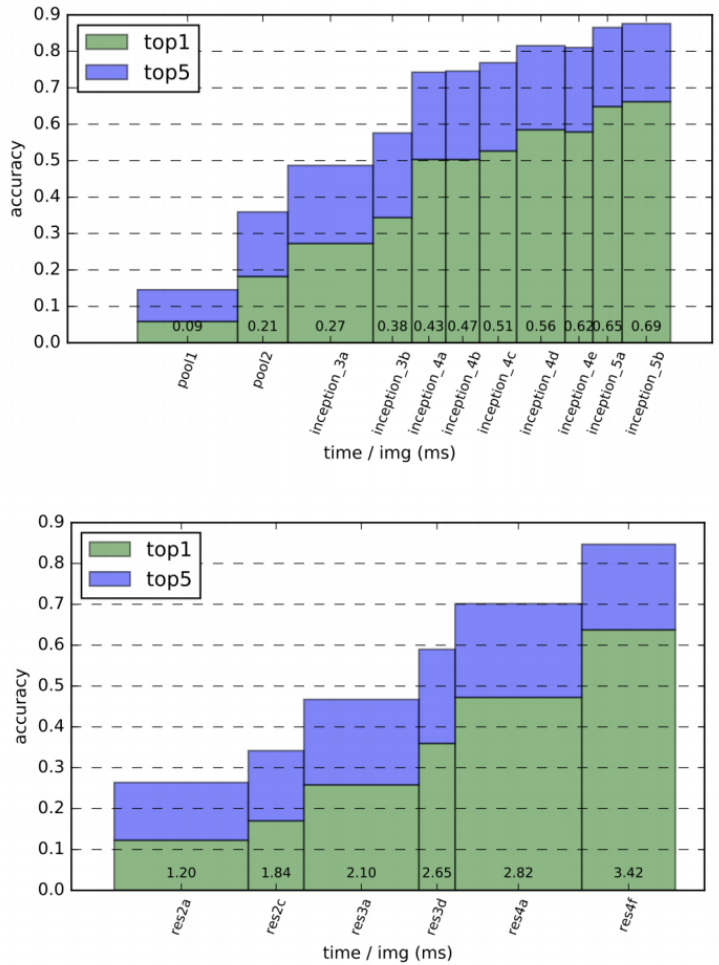


Figure 5. The plots show the accuracy gains at different layers for early exits for networks GoogLeNet (top) and Resnet50 (bottom).

Summary

- They proposed two different schemes to adaptively trade off model accuracy with model evaluation time for deep neural networks.
- They posed a global objective for learning an adaptive early exit or network selection policy and solved it by reducing the policy learning problem to a layer-by-layer weighted binary classification problem.
- They demonstrated that significant gains in computational time is possible through their novel policy with negligible loss in accuracy on ImageNet image recognition dataset.