Online Learning to Accelerate Neural Network Inference with Traveling Classifiers

Ege Beyazit [1] and **Yi He** [2] and **Nian-Feng Tzeng** [3] and **Xindong Wu** [4]

Abstract. Deep neural networks trained on millions of instances can recognize a wide variety of patterns. It is common to use these pre-trained deep networks in applications where the domain specific training data is not readily available. Once a pre-trained network is deployed to such applications, some of the information contained in the network may be irrelevant due to the difference between the training set and the application data distributions. As a result, parts of the neural network become redundant and slow down inference. This redundancy is unknown until the model is deployed and input data is received. Therefore, it can only be identified and avoided in real-time. Existing works on neural network acceleration can not exploit such redundancy during offline training when the domain-specific datasets are unavailable. In this paper, we study online learning to accelerate neural network inference. We propose *traveling classifiers* that continuously learn from the activations of two consecutive network layers to accelerate inference in real-time. Traveling classifiers model class conditional probabilities to generate early predictions and bypass unnecessary computation of network layers. The classifiers also adaptively switch the layers they learn from by measuring the *feature space differences* between the activations. This traveling mechanism automatically adjusts the aggressiveness of the acceleration without sacrificing prediction accuracy. We demonstrate the performance of the proposed algorithm on the ImageNet dataset [10] using the state-of-the-art ResNet-50, ResNet-152 [18] and VGG-16 [38] architectures. Experiments demonstrate that our method significantly outperforms baseline approaches.

1 INTRODUCTION

Recent studies have shown that the use of very large models can result in overcomplete systems that carry redundant information [8]. This suggests the acceleration of neural networks while preserving the accuracy is possible [31]. Existing approaches on neural network acceleration either generate truly static models regardless of the input received in inference time, or are adaptive but involve offline retraining. Moreover, their computational complexities scale up with the depth of the reference neural network. When a large pre-trained model that contains information from millions of labeled instances and hundreds of categories is used for a specific application, a new source of redundancy emerges. This redundancy is caused by the difference between the distributions of the labeled training set and the input data provided by the application. To answer queries on traffic surveillance cameras in real time, for example, state-of-the-art pre-trained object detectors such as YOLO [34] are used. These query answering systems typically suffer from high latency at query time, since inference with such large architectures is computationally expensive. These object detectors are trained to recognize a wide variety of objects and most of these objects may be redundant for a traffic surveillance system. As another example, when ResNet-50 [18], a convolutional network that is capable of classifying instances from a thousand categories, is used to classify a stream of photographs shared on a social media platform, the pattern distribution provided by the stream is unknown but can be skewed toward the current trends. Therefore, an unknown subset of information from the reference network may be sufficient for the task. Moreover since the trends can change over time, different subsets of information may be sufficient at different times. Existing pruning and acceleration strategies do not address these issues.

[1] University of Louisiana at Lafayette, USA, ege93@louisiana.edu
[2] University of Louisiana at Lafayette, USA, yi.he1@louisiana.edu
[3] University of Louisiana at Lafayette, USA, nianfeng.tzeng@louisiana.edu
[4] Mininglamp Academy of Sciences, China, wuxindong@mininglamp.com

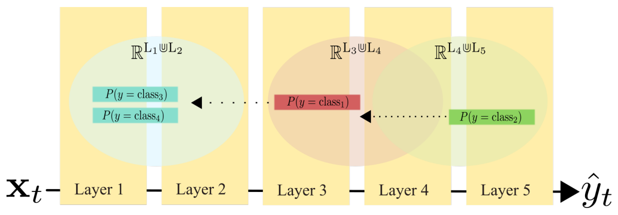

Figure 1. Proposed traveling classifier accelerator on a 5-layer neural network

To enable adaptive and real-time acceleration on neural network inference, we propose *online learning to accelerate with traveling classifiers*. Our proposed approach is motivated by two observations: (1) instances of the same class generate similar activations [1], and (2) different classes may require executions of different amounts of layers until their sufficiently discriminative features are extracted [40]. Our proposed algorithm initializes a *traveling classifier* for each class encountered by the neural network at inference time. Each classifier learns to model a class conditional probability only from a designated pair of neural network layers and automatically changes these layers for better acceleration, as shown in Figure 1. At each feedforward pass, the classifiers attempt to early classify the network activations. If the prediction of the most confident classifier is above a certain threshold, the feedforward pass is interrupted and the classifier output is used instead of the network output. Otherwise, the feedforward pass is completed and the network output is used to update the traveling classifiers. Each classifier incrementally measures the *feature space difference* between the two layers it learns from. Using the feature space differences, the classifiers dynamically switch the layers that feed them. This traveling mechanism lets the algorithm adjust the aggressiveness of the acceleration for each class separately, with respect to the input data distribution. The contributions of this paper are as follows:

- We propose traveling classifiers that learn to early classify network activations, while automatically switching the layers they learn from.
- We study the problem of online learning to accelerate neural network inference, proposing real-time, adaptive and efficient acceleration with traveling classifiers.
- We empirically validate the performance of the proposed method via the ImageNet dataset on state-of-the-art ResNet-50, ResNet-152 and VGG-16 architectures.

2 RELATED WORK

2.1 Online Learning

Online learning is performed in consecutive rounds. On each round, the learner receives a training example and makes a prediction according to the given task. Then the actual answer to the prediction task is revealed, causing the model suffer a potential loss. The learner aims to minimize the total loss over all rounds. Online learning algorithms can be grouped into two main categories: first-order [35, 42, 27] where the first derivative of the loss function is utilized, and second-order [7, 32, 16, 5] where both first and second-order information are used to update the model. There exists a family of online learning algorithms that focus on problems where the feature spaces change as the data stream in. [43] studied online classification where the data volume and number of features grow simultaneously over time. [3, 19] studied classification when feature spaces can arbitrarily change as the data stream in. [21] studied evolving feature spaces where new features are introduced to replace old ones. In this paper, we pose the problem of real-time acceleration as online learning by changing the feature spaces on-demand to balance accuracy and acceleration. Existing online learning methods can not efficiently learn in such setting because they do not provide any mechanism to switch feature spaces while learning. Finally, it is important to mention that our problem is different than online transfer learning [12] which focuses on building a new model in a given target domain.

2.2 Dynamic Neural Network Pruning

Pruning has been shown to improve generalization of neural networks [13], extensively studied as an acceleration and compression technique for large neural network architectures [9]. Some of the early and well known examples of neural network pruning algorithms are [26, 17, 23, 22, 36]. Several algorithms have been proposed to dynamically prune neural networks to accelerate inference. [15] pruned weights on-the-fly by using connection splicing and retraining. [28] pruned the network at runtime by training a recurrent neural network that models a Markov Decision process, using an encoder-decoder structure. [29] proposed a low-rank decomposition based global and dynamic pruning method that can recover removed filters, to improve the model accuracy. The dynamic pruning step they proposed needs to be followed by retraining. Even though the existing dynamic pruning methods work upon inference, their decision functions are trained offline. Therefore, they are not capable of adapting themselves to the inference task. Moreover, these methods introduce a large amount of memory and computational complexity due to neural network retraining. Hence, they are not appropriate for resource constrained systems in online setting.

2.3 Conditional Computation and Early Prediction

Conditional computation methods utilize decision functions that depend on the current input. These methods decide which parts of the neural network need to be executed for a given instance. Several conditional computation methods use these decision functions to generate early predictions, before the network executes a full feedforward pass. Conditional computation methods such as [39, 2] use reinforcement learning strategies to learn optimal sequences of executions, given input instances. These methods rely on availability of the labeled training data and sufficient computational power. Moreover, they scale poorly if the reference neural network is very deep. Another set of literature introduces new neural network architectures, proposing joint training strategies for loss minimization and efficient execution [33, 6, 30]. These architectures typically involve early prediction mechanisms that adaptively calculate the sufficient amount of feature extraction. Joint training based approaches need training from scratch. Therefore, they cannot be used to accelerate pre-trained models. Existing works on early prediction assume that a labeled training dataset exists for the given application. As a result, due to the difference between the training set and the inference data, these works cannot exploit the redundancy. Unlike existing works, our method focuses on exploiting this difference and discarding redundant information of the reference network with respect to the inference task. Our method introduces negligible additional memory and computational complexity, and adapts to the inference task in real-time. Also different from the existing works such as [41, 14, 11], our proposed method is not strictly coupled with a specific network architecture. Our method can be jointly used with existing pruning and acceleration methods, since it does not involve any additional constraint to the neural network training or inference.

3 NOTATION AND THE PROBLEM SETTING

We study the problem of online learning to accelerate a neural network. On every round t, the network receives an instance $\mathbf{x}_t$, generates a sequence of activations $\mathbf{H}_t = \{\mathbf{h}_t^1 \in \mathbb{R}^{l_1}, ..., \mathbf{h}_t^D \in \mathbb{R}^{l_D}\}$, and outputs a prediction $\hat{y}_t$. Each activation $\mathbf{h}_t^i$ is generated by a different layer, and therefore belongs to a different feature space $\mathbb{R}^{l_i}$. Since every neural network activation is a function of its preceding activation, the set of feature spaces $\{\mathbb{R}^{l_1}, ..., \mathbb{R}^{l_D}\}$ are disjoint but correlated. Hence, we see each sequence of activations generated by the network as an instance from a data stream generated by this set of correlated feature spaces. To accelerate a neural network in real-time and adaptively, we learn a multi-class model $\mathbf{M}$ that maintains a classifier $\mathbf{M}[k]$ for each class k. Let l_k^c and l_k^p denote the pair of consecutive network layers that the classifier $\mathbf{M}[k]$ receives its input from. Let $\mathbb{R}_k^c$ and $\mathbb{R}_k^p$ be the feature spaces of the activations generated by the layers l_k^c and l_k^p respectively. We denote the feature space $\mathbb{R}_k^c$ as the *child* space and $\mathbb{R}_k^p$ as the *parent* space for class k. Then, each classifier $\mathbf{M}[k] : [\mathbf{M}[k]^c, \mathbf{M}[k]^p]$ takes a pair of activations $[\mathbf{h}^{l_k^c}, \mathbf{h}^{l_k^p}]$ generated by the layers $[l_k^c, l_k^p]$. For notational simplicity we refer to these activations as $[\mathbf{h}^{k_c}, \mathbf{h}^{k_p}]$. When $\mathbf{M}[k]$ classifies an activation with a certain confidence, the layers after l_k^p are bypassed and the prediction of $\mathbf{M}[k]$ is used as the output. To be able to learn efficiently and in real-time, each class probability is modeled using only the activations from two consecutive layers, instead of using the full activation set $\mathbf{H}$. On each round, every classifier learns in its parent feature space and teaches to its child feature space counterpart. If the difference between the parent and child feature spaces with respect to the class conditional probability is small, the classifier travels to the preceding layer: it sets the child feature space as the new parent and truncates the old parent's weights. The classifier also declares the preceding layer of the new parent as the child feature space and initializes new weights for it. This lets the model automatically adjust the amount of acceleration for each class, while maintaining its accuracy. Therefore, we refer to $\mathbf{M}[k]$ as a *traveling classifier*.

4 ONLINE LEARNING TO ACCELERATE

4.1 Learning Class Conditional Probabilities

The traveling classifier $\mathbf{M}[k]$ represents a log-linear model that learns a class conditional probability from the parent feature space $\mathbb{R}^p_k$ and teaches to its weights in the child feature space $\mathbb{R}^c_k$. $\mathbf{M}[k]$ measures the difference between the two feature spaces and seeks to switch the layers it learns from for further acceleration. On round t, we define the probability of class k in the parent feature space parameterized by a traveling classifier as $P(\hat{y}_t = k \mid \mathbf{h}_t^{k_p}; \mathbf{M}[k]^p_t) = \sigma(\mathbf{M}[k]^p_t \cdot \mathbf{h}_t^{k_p}) = \frac{1}{1+\exp(-\mathbf{M}[k]^p_t \cdot \mathbf{h}_t^{k_p})}$, where $\hat{y}_t$ is the output of the neural network after a full feedforward pass for the instance $\mathbf{x}_t$. The probability in the child feature space is defined similarly. We express the loss of a traveling classifier $\mathbf{M}[k]$ with respect to the network output $\hat{y}$ as the negative log likelihood $\ell(\hat{y}, \sigma(\mathbf{M}[k] \cdot \mathbf{h}^k)) = \hat{y} \log[\sigma(\mathbf{M}[k] \cdot \mathbf{h}^k)] - (1 - \hat{y}) \log[1 - \sigma(\mathbf{M}[k] \cdot \mathbf{h}^k)]$. Let $\bar{y}_t = P(\hat{y}_t = k \mid \mathbf{h}_t^{k_p}; \mathbf{M}[k]^p_t)$ and $\tilde{y}_t = P(\hat{y}_t = k \mid \mathbf{h}_t^{k_c}; \mathbf{M}[k]^c_t)$. Then, we formulate the problem of learning the traveling classifier $\mathbf{M}[k]_{t+1} : \left[\mathbf{M}[k]^c_{t+1}, \mathbf{M}[k]^p_{t+1}\right]$ as the minimization of the following pair of regularized objective functions:

$$\begin{aligned} C(\mathbf{M}[k]^c) &= \frac{1}{2} \|\mathbf{M}[k]^c_t - \mathbf{M}[k]^c\|^2 \\ &+ \frac{\lambda_1}{2}\Big(\sigma(\mathbf{M}[k]^c_t \cdot \mathbf{h}_t^{k_c}) - \sigma(\mathbf{M}[k]^p_t \cdot \mathbf{h}_t^{k_p})\Big)^2 \qquad (1) \\ &+ \lambda_2 \ell\Big(\hat{y}_t, \sigma(\mathbf{M}[k]^c_t \cdot \mathbf{h}_t^{k_c})\Big), \\ C(\mathbf{M}[k]^p) &= \frac{1}{2} \|\mathbf{M}[k]^p_t - \mathbf{M}[k]^p\|^2 \qquad (2) \\ &+ \lambda_3 \ell(\hat{y}_t, \sigma(\mathbf{M}[k]^p_t \cdot \mathbf{h}_t^{k_p})), \end{aligned}$$

where $\lambda_1, \lambda_2, \lambda_3 \geq 0$ are tradeoff hyperparameters. On every round t, the traveling classifier aims to satisfy multiple soft constraints in two feature spaces. In the parent feature space, the classifier minimizes the loss suffered, with the smallest change to the prototype vector $\mathbf{M}[k]^p_t$. In the child feature space, the classifier has an additional regularizer to minimize the difference between predictions made in the two feature spaces. With this additional regularizer, the classifier forces its child feature space weights to receive information from its parent space counterpart. If the child feature space is sufficiently representative to replace its parent, the traveling classifier declares the current child space as the new parent, travels to the preceding layer, therefore starts making earlier predictions for class k.

We assume $\lambda_1 = \lambda_2 = \lambda_3 = \eta$ for simplicity, and set the derivatives of above objective functions to zero for solving $\mathbf{M}[k]_{t+1}$, arriving at the following update rules:

$$\begin{aligned} \mathbf{M}[k]_{t+1} : \left[\mathbf{M}[k]^c_{t+1}, \mathbf{M}[k]^p_{t+1}\right] = \\ \Big[\mathbf{M}[k]^c_t + \eta \mathbf{h}_t^{k_c}(\hat{y} - \tilde{y}(1 + (\bar{y} - \tilde{y})(1 - \tilde{y}))), \\ \mathbf{M}[k]^p_t + \eta \mathbf{h}_t^{k_p}(\hat{y} - \bar{y})\Big]. \qquad (3) \end{aligned}$$

Note that when the predictions from the two feature spaces are the same, the two update rules are also the same. Therefore, the classifier directly minimizes the negative likelihood function ℓ in both of the feature spaces. If the predictions are different, the classifier suffers from an extra loss in the child feature space because of the constraint $(\tilde{y}_t - \bar{y}_t)^2$. If the two feature spaces are similarly representative of the class conditional probability for k, this constraint helps the child feature space weights to converge to the probability distribution modeled by the parent feature space weights. If the spaces are not similar enough, the extra loss suffered helps the traveling classifier to early identify and approximate the *feature space difference*. This approximation is used to output a layer switching decision.

4.2 Adaptive Acceleration by Traveling

The traveling classifier $\mathbf{M}[k]$ learns to model the conditional probability of class k given the activations from feature spaces $\mathbb{R}^c_k$ and $\mathbb{R}^p_k$. On each round, it translates information from $\mathbb{R}^p_k$ to $\mathbb{R}^c_k$ while making predictions in $\mathbb{R}^p_k$. If the two feature spaces are similarly representative of the class conditional probability for k, the traveling classifier can directly learn and make predictions in $\mathbb{R}^c_k$ instead of $\mathbb{R}^p_k$, without sacrificing too much accuracy. This accelerates the network inference which can now be made without executing the network layers after l^p_k. In this section, we first introduce the *feature space difference* for a traveling classifier. Then, we design a decision mechanism that helps the traveling classifier to switch layers automatically if the two feature spaces are similar enough. We start by defining $R_t(\tilde{y}) = \frac{1}{2}(\tilde{y}_t - \bar{y}_t)^2 + \ell(\hat{y}_t, \tilde{y}_t)$ and $R_t(\bar{y}) = \ell(\hat{y}_t, \bar{y}_t)$ as the regularization received in the feature spaces $\mathbb{R}^c_k$ and $\mathbb{R}^p_k$ on round t respectively. Let $\tilde{y}^*_t = P(k \mid \mathbf{h}_t^{k_c}; \mathbf{M}^*[k]^c)$ and $\bar{y}^*_t = P(k \mid \mathbf{h}_t^{k_p}; \mathbf{M}^*[k]^p)$ be the outputs generated by the unknown optimal traveling classifier weights in the two feature spaces. We define the feature space difference for the traveling classifier $\mathbf{M}[k]$ as follows:

$$\mathbb{E}[\Delta^*_k] = \frac{1}{T} \sum_{t=1}^{T} \mid R_t(\tilde{y}^*_t) - R_t(\bar{y}^*_t) \mid . \qquad (4)$$

Therefore, the feature space difference is the expected value of the absolute regularization difference distribution Δ^*_k. It is important to note that measuring the difference based on the functions $R_t(\tilde{y})$ and $R_t(\bar{y})$ is supported by the objective functions given in Equations (1) and (2) because the traveling classifier receives an extra loss in the child feature space if the predictions are different. As a result, the total regularization difference increases. If the outputs generated in the two feature spaces are similar even though their predictions are wrong, they receive similar regularizations, and hence the difference is small.

The proposed feature space difference metric is not immediately practical due to two reasons. First, since the optimal weight matrix $\mathbf{M}^*$ is unknown, it is not possible to calculate the regularizations received on each round. Second, because an upper bound to T is not available, the number of samples that needs to be observed to calculate $\mathbb{E}[\Delta^*_k]$ is unknown. We address these problems with the following two theorems respectively.

Theorem 1. *Let $[\mathcal{K}^c_k, \mathcal{K}^p_k]$ be a pair of bounded convex and compact sets in the Euclidian spaces defined by their corresponding layers. Let $[D^c_k, D^p_k]$ be the upper bounds on the diameters of the sets $[\mathcal{K}^c_k, \mathcal{K}^p_k]$. Let $[G^c_k, G^p_k]$ be the upper bounds on the norm of the regularizer gradients $[\nabla R_t(\tilde{y}), \nabla R_t(\bar{y})]$ with respect to $\mathbf{M}^*$. Let the step sizes in the two feature spaces $[\eta^c_t, \eta^p_t]$ on round t be*

$[D^c_k/G^c_k\sqrt{t}, D^p_k/G^p_k\sqrt{t}]$. *After observing T instances, the total regularization suffered by the optimal traveling classifier $\mathbf{M}^*[k]$ in the parent and child feature spaces is bounded as follows:*

$$\sum_{t=1}^{T} R_t(\tilde{y}^*_t) \le \sum_{t=1}^{T} R_t(\tilde{y}_t) + G^c_k D^c_k\sqrt{T},$$
$$\sum_{t=1}^{T} R_t(\bar{y}^*_t) \le \sum_{t=1}^{T} R_t(\bar{y}_t) + G^p_k D^p_k\sqrt{T}.$$

Proof of Theorem 1. Using the functions $R_t(\tilde{y})$ and $R_t(\bar{y})$, we can rewrite the update rules derived for a traveling classifier as $\mathbf{M}[k]^c_{t+1} = \mathbf{M}[k]^c_t + \eta^c_t\nabla R_t(\tilde{y})$ and $\mathbf{M}[k]^p_{t+1} = \mathbf{M}[k]^p_t + \eta^p_t\nabla R_t(\bar{y})$. Note that in this form, the functions $R_t(\tilde{y})$ and $R_t(\bar{y})$ act as the loss functions of a pair of online gradient descent update rules. These functions are convex in the vector $\mathbf{M}[k]$ [24]. Hence to prove Theorem 1, we borrow the concept of *regret* from Online Convex Optimization framework [37] and use the following lemma.

Lemma 1. *The traveling classifier $\mathbf{M}[k]$ regularized by $R_t(\tilde{y})$ and $R_t(\bar{y})$ attains the following regret bounds in the child and parent feature spaces, relative to the output distribution:*

$$\text{regret}^{k_c}_T \le G^c_k D^c_k\sqrt{T} \quad \text{and} \quad \text{regret}^{k_p}_T \le G^p_k D^p_k\sqrt{T}. \qquad (5)$$

Proof of Lemma 1. We start by proving the lemma for the child feature space. By convexity of $R_t(\tilde{y})$, the following inequality holds:

$$R_t(\tilde{y}_t) - R_t(\tilde{y}^*_t) \le (\nabla R_t(\tilde{y}))^T(\mathbf{M}[k]^c_t - \mathbf{M}^*[k]^c). \qquad (6)$$

Therefore by summing over T, we can upper bound $\text{regret}^{k_c}_T$ with the right side of Equation (6). We proceed to upper bound the right side using the update rule $\mathbf{M}[k]^c_{t+1} = \mathbf{M}[k]^c_t + \eta^c_t\nabla R_t(\tilde{y})$ we previously defined, and derive the following equation.

$$\|\mathbf{M}[k]^c_{t+1} - \mathbf{M}^*[k]^c\|^2 = \|(\mathbf{M}[k]^c_t - \mathbf{M}^*[k]^c) - \eta^c_t\nabla R_t(\tilde{y})\|^2 =$$
$$\|\mathbf{M}[k]^c_t - \mathbf{M}^*[k]^c\|^2 - 2\eta^c_t(\nabla R_t(\tilde{y}))^T(\mathbf{M}[k]^c_t - \mathbf{M}^*[k]^c)$$
$$+ (\eta^c_t)^2\,\|\nabla R_t(\tilde{y})\|^2.$$

After rearranging, we get

$$\begin{aligned}
&(\nabla R_t(\tilde{y}))^T(\mathbf{M}[k]^c_t - \mathbf{M}^*[k]^c) = \\
&\frac{\|\mathbf{M}[k]^c_t - \mathbf{M}^*[k]^c\|^2 - \|\mathbf{M}[k]^c_{t+1} - \mathbf{M}^*[k]^c\|^2}{2\eta^c_t} \\
&+ \frac{\eta^c_t\,\|\nabla R_t(\tilde{y})\|^2}{2} \\
&\le \frac{\|\mathbf{M}[k]^c_t - \mathbf{M}^*[k]^c\|^2}{2\eta^c_t} - \frac{\|\mathbf{M}[k]^c_t - \mathbf{M}^*[k]^c\|^2}{2\eta^c_{t-1}} \\
&+ \frac{\eta^c_t\,\|\nabla R_t(\tilde{y})\|^2}{2}.
\end{aligned}$$

We sum the expression above over T, and upper bound the regret by the following inequality:

$$\begin{aligned}
\text{regret}^{k_c}_T &= \sum_{t=1}^{T} R_t(\tilde{y})(\mathbf{M}[k]^c_t) - R_t(\tilde{y})(\mathbf{M}^*[k]^c) \\
&\le \sum_{t=1}^{T} \frac{\|\mathbf{M}[k]^c_t - \mathbf{M}^*[k]^c\|^2}{2\eta^c_t} \\
&- \frac{\|\mathbf{M}[k]^c_t - \mathbf{M}^*[k]^c\|^2}{2\eta^c_{t-1}} + \sum_{t=1}^{T} \frac{\eta^c_t\,\|\nabla R_t(\tilde{y})\|^2}{2}.
\end{aligned}$$

After plugging D^c_k and G^c_k in, we use *telescoping series* over η^c_t. Finally, we conclude the proof of Lemma 1 as follows:

$$\begin{aligned}
\text{regret}^{k_c}_T &\le \frac{(G^c_k D^c_k)^2}{2}\sum_{t=1}^{T}\eta^c_t + \frac{1}{\eta^c_t} - \frac{1}{\eta^c_{t-1}} \\
&\le G^c_k D^c_k\sqrt{T}
\end{aligned}$$

Since $R_t(\bar{y})$ is also a convex function, the same sequence of operations can be followed to derive the bound for $\text{regret}^{k_p}_T$ by using η^p_t, D^p_k and G^p_k. Plugging the definition of regret in the inequality above and rearranging concludes the proof of Theorem 1. This theorem suggests that instead of using the difference between regularization received by the optimal weights in Equation (4), we can upper bound the difference. To address the second problem of the proposed difference metric, we use the following theorem.

Theorem 2. *Assume that the regularization difference distribution Δ^*_k takes values in [0,1]. Then the following inequality holds for $\delta \in [0,1]$:*

$$P\Big[\big(\mathbb{E}[\Delta^*_k] - \frac{1}{n}\sum_{t=1}^{n} \mid R_t(\tilde{y}_t) - R_t(\bar{y}_t)\mid\big) \le$$
$$\frac{1}{\sqrt{2n}}(\sqrt{\log(2/\delta)} + 2\sqrt{2}(G^c_k D^c_k + G^p_k D^p_k))\Big] \ge 1 - \delta. \qquad (7)$$

Proof of Theorem 2. Let $\Delta^*_k[1], ..., \Delta^*_k[n]$ be *i.i.d.* samples from the difference distribution Δ^*_k. Then, Hoeffding's inequality [20] states that $P(|\frac{1}{n}\sum_{t=1}^{n}\Delta^*_k[t] - \mathbb{E}[\Delta^*_k]| > \epsilon) \le 1 - \delta$ where $\epsilon = \sqrt{\frac{1}{2n}\log 2/\delta}$. By Theorem 1 we have,

$$\sum_{t=1}^{n} \mid R_t(\tilde{y}^*_t) - R_t(\bar{y}^*_t)\mid \le \sum_{t=1}^{n} \mid R_t(\tilde{y}_t) - R_t(\bar{y}_t)\mid +$$
$$(G^c_k D^c_k + G^p_k D^p_k)\sqrt{n}. \qquad (8)$$

Combining Theorem 1 and Hoeffding's inequality, we derive the following equation:

$$P(\frac{1}{n}\sum_{t=1}^{n} \mid R_t(\tilde{y}_t) - R_t(\bar{y}_t)\mid + \frac{(G^c_k D^c_k + G^p_k D^p_k)}{\sqrt{n}}$$
$$- \mathbb{E}[\Delta^*_k] < \epsilon) = 1 - \delta. \qquad (9)$$

Plugging ϵ in and rearranging the equation above, we conclude the proof of Theorem 2. This theorem utilizes Hoeffding's inequality and the upper bounds provided by Theorem 1 to decide if two feature spaces are similarly representative. Note that Hoeffding's inequality applies to arbitrary stationary distributions and $\{(\bar{y}_t, \tilde{y}_t)\}_{0<t<T}$ is not guaranteed to be stationary due to the updates received by the traveling classifier on each round. Theorem 2 uses the fact that $\{(\bar{y}_t, \tilde{y}_t)\}_{0<t<T}$ can be calculated on each round and be used with Theorem 1 to upper bound the stationary output distribution $\{(\bar{y}^*_t, \tilde{y}^*_t)\}_{0<t<T}$ generated by the optimal weights. The final layer switching mechanism for the traveling classifier $\mathbf{M}[k]$ is shown in Algorithm 1. When the condition given in Theorem 2 is satisfied with a margin of hyperparameter th_s, the classifier travels to the preceding layer by declaring the child space as the new parent and initializing a new child prototype, as listed on lines 2-9.

4.3 Online Acceleration with Traveling Classifiers

In this section, we first extend the traveling classifier to multi-class setting and derive the update rules for the proposed accelerator. Then,

Algorithm 1: Traveling Process

```
Input : $n_k$: Number of instances seen by $\mathbf{M}[k]$.
        $th_s$ : Layer switching threshold.
        $[G^c_k, G^p_k]$: Grad. upper bounds in two spaces.
        $[D^c_k, D^p_k]$: Diameters of two spaces.
        $\delta$: Probability threshold.
        $\sum \Delta_k$: $\sum_{t=1}^{n_k} \mid R_t(\tilde{y}_t) - R_t(\bar{y}_t) \mid$ for $\mathbf{M}[k]$.
1 $\epsilon =$
    $(\sqrt{\log(2/\delta)} + 2\sqrt{2}(G^c_k D^c_k + G^p_k D^p_k))/\sqrt{2n} + (\sum \Delta_k)/n_k$.
2 if $\epsilon < th_s$ then
3     Set child as new parent: $\mathbf{M}[k]^p = \mathbf{M}[k]^c$
4     Initialize the new child: $\mathbf{M}[k]^c \in R^{l^c_k - 1}$
5     Update parent layer: $l^p_k = l^p_k - 1$
6     Update child layer: $l^c_k = l^c_k - 1$
7     Reset regularizer sum: $\sum \Delta_k = 0$
8     Reset number of instances: $n_k = 0$
9 end
```

we propose an accelerated feedforward pass algorithm that uses traveling classifiers, and discuss the algorithm's running time and memory complexity.

Multi-class Extension. Let $\mathbf{M}$ be the multi-class model that contains a traveling classifier $\mathbf{M}[k]$ for each class k encountered by the neural network. On any round t, $\mathbf{M}$ may have traveling classifiers that expect outputs from different neural network layers due to the layer switching mechanism. When layer l is executed on round t, $\{\mathbf{M}[k]_t\}_{k|l_k=l}$ denotes the set of traveling classifiers compatible with the output of l. The prediction of the *multi-class traveling classifier (MTC)* is calculated as $\hat{k}_t = \underset{k}{\operatorname{argmax}}\{\sigma(\mathbf{M}[k]^p_t \cdot \mathbf{h}^l_t)\}_{k|l^p_k=l}$. On every round t a feedforward pass is interrupted or completed, and an output is generated either by MTC or the neural network. This output $o_t \in \{\hat{k}_t, \hat{y}_t\}$ is used as supervision to update the model. Let $\mathbf{M}'_t = \{k \neq o_t : \sigma(\mathbf{M}[k]^p_t \cdot \mathbf{h}^{k_p}_t) \geq \sigma(\mathbf{M}[o_t]^p_t \cdot \mathbf{h}^{(o_t)_p}_t)\}$ be the prototype set that contains the classifiers making an incorrect prediction with a higher confidence than the prototype of o_t. To update the multi-class model, we use a simple approach by only modifying $\mathbf{M}'_t \cup \mathbf{M}[o_t]_t$ when $\mathbf{M}'_t \neq \emptyset$. This helps to minimize the computational cost and keep the overall change to all classes balanced. On round t, MTC is updated as follows:

$$\begin{aligned}\mathbf{M}[k]_{t+1} = [\mathbf{M}[k]^c_{t+1}, \mathbf{M}[k]^p_{t+1}] = \\ [\mathbf{M}[k]^c_t + \tau_t \eta^c_t \mathbf{h}^{k_c}_t(\hat{y} - \tilde{y}(1 + (\bar{y} - \tilde{y})(1 - \tilde{y}))), \\ \mathbf{M}[k]^p_t + \tau_t \eta^p_t \mathbf{h}^{k_p}_t(\hat{y} - \bar{y})]\end{aligned} \quad (10)$$

where

$$\tau_t = \begin{cases} 1/|\mathbf{M}'_t|, & \text{if } k \in \mathbf{M}'_t \\ 1, & \text{if } k = o_t \\ 0, & \text{otherwise.} \end{cases} \quad (11)$$

Algorithm and Complexity Analysis. The proposed accelerated feedforward pass is shown in Algorithm 2. MTC is initialized to the empty set on round $t = 1$, hence it can not generate any prediction and lets the network complete its feedforward pass. As the neural network f outputs predictions $f.output$, MTC initializes a traveling classifier that learns from the last layer of the network for each class, listed on lines 16-20. For each layer, if MTC generates a confident activation, the feedforward pass is interrupted and MTC output $\hat{k}_t$ is used as the final prediction, shown on lines 6-12. MTC weights are

Algorithm 2: Accelerated Feedforward Pass

```
Input : $f$: Trained neural network.
        $\mathbf{x}_t$: Neural network input.
        $th_s$ : Layer switching threshold.
        $th_b$ : Bypass threshold.
        $\delta$: Probability threshold.
        $D^p, D^c$: Diameter bounds.
        $G^p, G^c$: Gradient bounds.
        MTC: Multi-class clf.
1  Set initial activation: $f[0](\mathbf{x}_t) = \mathbf{x}_t$.
2  Set $\eta^p_t = D^p/G^p\sqrt{t}$, $\eta^c_t = D^c/G^c\sqrt{t}$
3  for $l$ in $f.layers - 1$ do
4      Generate act. pair: $\mathbf{h}^l_t, \mathbf{h}^{l+1}_t = f[l](\mathbf{h}^{l-1}_t), f[l+1](\mathbf{h}^l_t)$.
5      Get prediction: $\hat{k}_t = \underset{k}{\operatorname{argmax}}\{\sigma(\mathbf{M}[k]^p_t \cdot \mathbf{h}^l_t)\}_{k|l^p_k=l}$.
6      if confidence($\hat{k}_t$) > $th_b$ then
7          Interrupt feedforward pass.
8          Set output: $o_t = \hat{k}_t$.
9          Update MTC with $\eta^p_t, \eta^c_t, o_t$ using Eq. (10), (11).
10         Call Algorithm 1 to attempt travel.
11         Update $n_k$ and $\sum \Delta_k$ for each traveling clf.
12         return $o_t$
13     end
14 end
15 Set output $o_t = f.output$.
16 if $o_t$ not in MTC then
17     Initialize new traveling clf. for $o_t$: $\mathbf{M}[o_t] \in \mathbb{R}^l$.
18     Set variables for new child and parent:
19     $l^p_{o_t} = l$, $l^c_{o_t} = l - 1$, and $n_{o_t} = 1$.
20 end
21 Update MTC with $\eta^p_t, \eta^c_t, o_t$ using Eq. (10), (11).
22 Update $n_k$ and $\sum \Delta_k$ for each traveling clf.
23 return $o_t$.
```

updated based on the final prediction generated on each round, shown on the lines 9 and 21. After generating predictions, MTC checks if any of its prototypes is eligible to switch layers, shown on line 10. If MTC generates an activation that is not confident enough, the next layer of the network is executed. If MTC cannot generate a confident activation for any of the network layers, the network output is used as the final prediction, shown on line 15. Let the number of different classes encountered by the neural network on round t be $|C_t|$, and the maximum neural network activation size be $|h|_{max}$. On every round t, the proposed algorithm introduces $O(|C_t||h|_{max})$ running time and memory complexity. If no early prediction occurs, the additional overhead of using the accelerated feedforward pass per round is only linear in $|h|_{max}$ and $|C_t|$. Note that unlike most of the existing works, additional complexity introduced by MTC does not depend on the neural network depth.

Table 1. Hyperparameters adopted by the proposed accelerator

Architecture	th_s	th_b	D
ResNet-50	0.1	0.9	1
ResNet-152	0.1	0.8	1
VGG-16	0.1	0.9	0.5

5 EXPERIMENTS AND RESULTS

5.1 Experiment Setting

We evaluate the performance of the proposed acceleration algorithm using ResNet-50, ResNet-152 [18] and VGG-16 [38] trained on ImageNet dataset [25]. We train each neural network using their best parameter settings and report their Top-1 accuracy in our experiment results. To simulate the redundancy introduced by the difference between the training set and application data distributions, we randomly sample subsets of instances from 100 out of 1000 of the classes available in the ImageNet test set. Then, we use Gaussian noise and random flips to increase the number of instances of our inference task. We shuffle the sampled instances on each round, and report the average performance over 10 rounds. We tune the th_b, th_s and D parameters using grid search and set them as shown in Table 1. Note that we use D to set both D^c and D^p, since both diameters belong to the same neural network. We set the probability threshold δ to 10^{-10}.

Setting the Gradient Upper Bounds. We provide a theorem to upper bound the gradients of the functions $R_t(\tilde{y}) = R_t(\sigma(\mathbf{M}[k]^c_t \cdot \mathbf{h}^{k_c}_t))$ and $R_t(\bar{y}) = R_t(\sigma(\mathbf{M}[k]^p_t \cdot \mathbf{h}^{k_p}_t))$. We use these bounds to set the values for the hyperparameters G^c and G^p. Let $\mathbf{x}$ belong to a bounded convex set $\mathcal{K}$, and D be the diameter upper bound of the set $\mathcal{K}$. Gradient of a function $f(\mathbf{x})$ can be written as $\nabla_{\mathbf{x}} f(\mathbf{x}) = c\mathbf{x}$ where c is the proportionality constant decided by the function [4]. If we find an upper bound for c such that $c \leq L$, we can also upper bound the gradient norm as $\|\nabla_{\mathbf{x}} f(\mathbf{x})\| \leq LD$. We start by providing the definition of Lipschitz continuity.

Definition 1. *Let $L > 0$ be an upper bound on the (sub)derivative of a function f over a bounded convex set $\mathcal{K}$ such that, $|\frac{\partial}{\partial \mathbf{x}} f(\mathbf{x})| \leq L$ for all $\mathbf{x} \in \mathcal{K}$. Existence of such bound implies that f is L-Lipschitz.*

Note that Lipschitzness does not imply any bound on gradients, but implies a bound on the proportionality constant c. This suggests that if we show the functions $R_t(\tilde{y})$ and $R_t(\bar{y})$ are Lipschitz continuous and derive their corresponding Lipschitz variables, we can use them with D to upper bound the gradients. Since $\tilde{y}$ and $\bar{y}$ are generated by the function $\sigma(\mathbf{x}) = 1/(1+e^{-\mathbf{x}})$, the inequality $0 \leq \tilde{y}, \bar{y} \leq 1$ holds. Also note that, $\hat{y} \in \{0,1\}$ for each $\mathbf{M}[k]$ of $\mathbf{M}$. As a result, to bound the gradients of the functions $R_t(\tilde{y})$ and $R_t(\bar{y})$, it is sufficient to prove that they are locally Lipschitz in $[0,1]$.

Theorem 3. *The functions $R_t(\sigma(\mathbf{M}[k]^c_t \cdot \mathbf{h}^{k_c}_t))$ and $R_t(\sigma(\mathbf{M}[k]^p_t \cdot \mathbf{h}^{k_p}_t))$ are 1-Lipschitz in $[0,1]$, and their gradients with respect to their corresponding classifier weights $\mathbf{M}[k]^c_t$ and $\mathbf{M}[k]^p_t$ are bounded as follows:*

$$\left\| \frac{\partial R_t(\tilde{y})}{\partial \mathbf{M}[k]^c_t} \right\| \leq 2, \left\| \frac{\partial R_t(\bar{y})}{\partial \mathbf{M}[k]^p_t} \right\| \leq 1. \tag{12}$$

Proof of Theorem 3. We start by taking the partial derivatives of both functions as follows:

$$\frac{\partial R_t(\tilde{y})}{\partial \mathbf{M}[k]^c_t} = \hat{y} - \tilde{y}(1 + (\bar{y} - \tilde{y})(1 - \tilde{y}))), \tag{13}$$

$$\frac{\partial R_t(\bar{y})}{\partial \mathbf{M}[k]^p_t} = \hat{y} - \bar{y}. \tag{14}$$

It is trivial to see that for $\hat{y}, \bar{y} \in [0,1]$, the inequality $\partial R_t(\bar{y}) \leq 1$ holds. Therefore the function $R_t(\bar{y})$ is 1-Lipschitz in $[0,1]$. To prove Lipschitzness of the function $R_t(\tilde{y})$, we use the fact that it is the sum of two locally Lipschitz functions $\frac{1}{2}(\tilde{y}_t - \bar{y}_t)^2$ and $\ell(\hat{y}_t, \tilde{y}_t)$ [4]. Then, we use the following lemma.

Lemma 2. *Assume $f(\tilde{y}) = \frac{1}{2}(\tilde{y}_t - \bar{y}_t)^2$, $g(\tilde{y}) = \ell(\hat{y}_t, \tilde{y}_t)$ are locally Lipschitz functions in the region $[0,1]$ with the constants L_f and L_g. Then the sum $f(\tilde{y}) + g(\tilde{y})$ is also locally Lipschitz with the constant L_{sum}, where $L_{sum} \leq L_f + L_g$.*

Proof of Lemma 2. By using Definition 1 we prove the lemma for all $\tilde{y}_1, \tilde{y}_2 \in [0,1]$ as follows.

$$\begin{aligned}
&|f(\tilde{y}_1) + g(\tilde{y}_1) - f(\tilde{y}_2) - g(\tilde{y}_2)| \\
&\leq |f(\tilde{y}_1) - f(\tilde{y}_2)| + |g(\tilde{y}_1) - g(\tilde{y}_2)| \\
&\leq L_f |\tilde{y}_1 - \tilde{y}_2| + L_g |\tilde{y}_1 - \tilde{y}_2| \\
&= (L_f + L_g)|\tilde{y}_1 - \tilde{y}_2|.
\end{aligned}$$

Note that $\frac{1}{2}(\tilde{y}_t - \bar{y}_t)^2$ and $\ell(\hat{y}_t, \tilde{y}_t)$ are both 1-Lipschitz in $[0,1]$. Then according to Lemma 2, the function $R_t(\tilde{y})$ is $\tilde{L}$-Lipschitz where $\tilde{L} \leq 2$. This concludes the proof of Theorem 3.

Handling Convolutional Layers. Convolutional layers generate feature maps of different dimensions based on input and receptive field sizes, stride of convolution and padding. Also, the number of feature maps generated by each convolutional layer is typically decided by the number of kernels contained in the corresponding layer. Each kernel ideally extracts a different feature, and combinations of these features are used to output predictions. Motivated by this, we employ a simple mapping strategy from the set of kernels generated by a convolutional layer to a vector. Let $\Phi^l = \{\phi^l_1, ..., \phi^l_M\}$ be the set of feature maps generated by the l^{th} layer and $\phi^l_m[i]$ be the i^{th} element of ϕ^l_m. We define the activation vector for layer l as $\mathbf{h}^l = \left[\frac{1}{|\phi_1|}\sum_{i=1}^{|\phi_1|}\phi_1[i], ..., \frac{1}{|\phi_M|}\sum_{i=1}^{|\phi_M|}\phi_M[i]\right]$. Note that this strategy approximates the *likelihood* of the feature maps, when each value of a feature map is assumed to be independent of each other.

5.2 Experiments with State-of-the-art Neural Networks

In this section, we demonstrate the performance of the proposed online acceleration algorithm on three state-of-the-art deep neural network architectures. We evaluate the proposed acceleration algorithm in terms of average accumulating prediction error rate and the amount of layers executed, as the inference data stream across the neural network. The first row of Figure 2 shows the accumulating neural network usage of the proposed accelerator MTC. Several observations can be made from this row. First, the network usage is initially high because the proposed accelerator cannot output sufficiently confident predictions to trigger an interrupt to the feedforward pass. As more instances received, the accelerator is trained using the network output. The trained accelerator starts generating confident early predictions and network usage decreases. Second, the network usage does not always decrease smoothly. This is because the model encounters instances from classes that have not been seen by MTC before. Therefore MTC needs to let the network execute a few feedforward passes, initialize a new prototype for the recently encountered class and train it before outputting confident early predictions. Also note that as the neural networks get deeper, the network usage ratio decreases because there exists more redundancy to be exploited. While exploiting this redundancy, our proposed accelerator does not introduce additional complexity as the networks get deeper. From the second row of Figure 2 it can be seen that the accelerator converges after receiving one half of the instances. After convergence, the accelerator achieves comparable error rates to the original networks. This observation can also be validated by comparing the average numbers of prediction errors and network layers executed, as listed in Table 2.

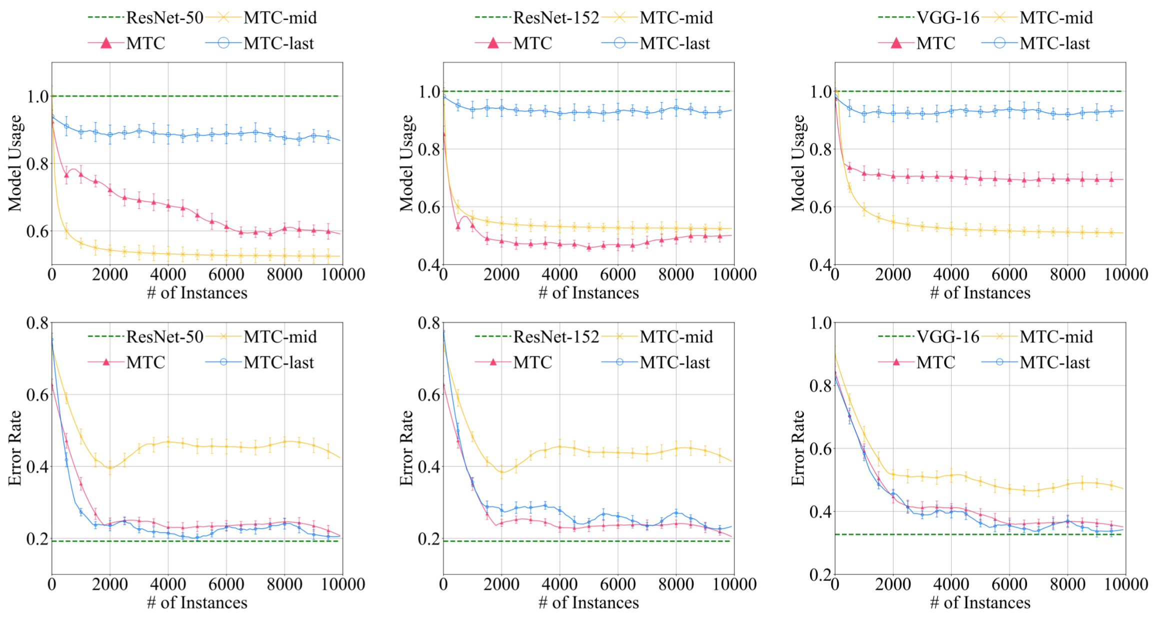

Figure 2. Experiments comparing error rate and network usage on the three network architectures

Table 2. Accumulating amounts of mispredictions and network usage

Error	**ResNet-50**	**ResNet-152**	**VGG-16**
No Acc.	1923	1930	3269
MTC	1981 ± 46	1998 ± 43	3340 ± 92
MTC-mid	4268 ± 66	4160 ± 67	1568 ± 114
MTC-last	1952 ± 24	2115 ± 15	3272 ± 86
Net Use	**ResNet-50**	**ResNet-152**	**VGG-16**
No Acc.	500K	1.52M	160K
MTC	295K $\pm$ 428	761K $\pm$ 216	111K $\pm$ 244
MTC-mid	262K $\pm$ 125	653K $\pm$ 294	81.6K $\pm$ 415
MTC-last	439K $\pm$ 27	1.39M $\pm$ 281	147K $\pm$ 328

5.3 Comparison with Baseline Accelerators

Existing methods on neural network inference acceleration require offline training of the accelerator models. These methods need multiple epochs (20 - 100) for convergence. As a result, it is not possible for these methods to exploit the existing distribution difference in runtime. Hence, it is unfair to compare these methods with our proposed algorithm in online setting. In this section, we introduce two baseline accelerators that can be trained online and compare the performance of the proposed algorithm with these baseline methods. These methods are designed to demonstrate that our proposed algorithm is able to balance the accuracy and acceleration of a neural network while learning in online setting. The baseline methods use the same update rules and thresholds as the proposed model MTC, but they do not have the capability of traveling among the network layers for adaptive acceleration. Note that these two baseline methods have the same asymptotic complexity as the proposed MTC.

Baseline 1: MTC-mid. This method is designed to learn from the middle layer of each neural network only. Therefore, it provides approximately $2\times$ acceleration by interrupting the feedforward pass in the middle.

Baseline 2: MTC-last. This method is designed to learn from the last layer of each neural network. Therefore, it represents the maximum accuracy that can be attained by the linear acceleration scheme in online setting. MTC-last can only interrupt the feedforward pass right before the last layer of the network is executed.

From Figure 2, it can be observed that the accelerator MTC-mid exhibits around 50% of neural network usage. This is because MTC-mid models all the class probabilities by using only the middle neural network layers, and outputs a bypass decision after the execution of these layers. It can be seen that MTC-mid sacrifices a lot in terms of prediction accuracy compared to the other methods we introduced. On the other hand, the accelerator MTC-last achieves a very similar accuracy to the original neural networks it accelerated, since MTC-last basically approximates the last fully connected layers of these networks using a linear model. Also note that, because MTC-last is located at the last layers of the networks, it cannot achieve high acceleration rates. From Figure 2 and Table 2, it can be observed that the proposed traveling classifier based acceleration algorithm MTC can automatically balance the prediction accuracy and acceleration of the feedforward pass with the help of its layer switching mechanism. This mechanism allows the proposed accelerator to model the conditional probabilities separately for each class, adapting the number of layers to be executed before interrupting the feedforward pass based on the class characteristics.

6 CONCLUSION

In this paper, we have explored online learning to accelerate neural network inference. We propose traveling classifiers that adaptively change the layers they learn from and output early predictions for a given network. For adaptive layer switching, we introduce the feature space difference for a traveling classifier and provide an algorithm. We also propose an accelerated feedforward pass algorithm that utilizes the traveling classifiers and analyze its running time complexity. Finally, we demonstrate the proposed algorithm's performance on state-of-the-art network architectures, compare it with two baseline online accelerators and discuss its effectiveness.

ACKNOWLEDGEMENTS

This research is supported by the US National Science Foundation (NSF) under grants 1652107 and 1763620.

REFERENCES

[1] Babajide O Ayinde, Tamer Inanc, and Jacek M Zurada, 'Redundant feature pruning for accelerated inference in deep neural networks', *Neural Networks*, (2019).

[2] Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup, 'Conditional computation in neural networks for faster models', *arXiv preprint arXiv:1511.06297*, (2015).

[3] Ege Beyazit, Jeevithan Alagurajah, and Xindong Wu, 'Online learning from data streams with varying feature spaces', in *Thirty-Third AAAI Conference on Artificial Intelligence*, (2019).

[4] Stephen Boyd and Lieven Vandenberghe, *Convex optimization*, Cambridge university press, 2004.

[5] Daniele Calandriello, Alessandro Lazaric, and Michal Valko, 'Efficient second-order online kernel learning with adaptive embedding', in *Advances in Neural Information Processing Systems*, pp. 6140–6150, (2017).

[6] Víctor Campos, Brendan Jou, Xavier Giró-i Nieto, Jordi Torres, and Shih-Fu Chang, 'Skip rnn: Learning to skip state updates in recurrent neural networks', *arXiv preprint arXiv:1708.06834*, (2017).

[7] Nicolo Cesa-Bianchi, Alex Conconi, and Claudio Gentile, 'A second-order perceptron algorithm', *SIAM Journal on Computing*, **34**(3), 640–668, (2005).

[8] Soravit Changpinyo, Mark Sandler, and Andrey Zhmoginov, 'The power of sparsity in convolutional neural networks', *arXiv preprint arXiv:1702.06257*, (2017).

[9] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang, 'A survey of model compression and acceleration for deep neural networks', *arXiv preprint arXiv:1710.09282*, (2017).

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei, 'Imagenet: A large-scale hierarchical image database', in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, (2009).

[11] Michael Figurnov, Maxwell D Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry Vetrov, and Ruslan Salakhutdinov, 'Spatially adaptive computation time for residual networks', in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1039–1048, (2017).

[12] Liang Ge, Jing Gao, and Aidong Zhang, 'Oms-tl: a framework of online multiple source transfer learning', in *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pp. 2423–2428. ACM, (2013).

[13] C Lee Giles and Christian W Omlin, 'Pruning recurrent neural networks for improved generalization performance', *IEEE transactions on neural networks*, **5**(5), 848–851, (1994).

[14] Alex Graves, 'Adaptive computation time for recurrent neural networks', *arXiv preprint arXiv:1603.08983*, (2016).

[15] Yiwen Guo, Anbang Yao, and Yurong Chen, 'Dynamic network surgery for efficient dnns', in *Advances In Neural Information Processing Systems*, pp. 1379–1387, (2016).

[16] Shuji Hao, Peilin Zhao, Jing Lu, Steven CH Hoi, Chunyan Miao, and Chi Zhang, 'Soal: Second-order online active learning', in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 931–936. IEEE, (2016).

[17] Babak Hassibi, David G Stork, and Gregory J Wolff, 'Optimal brain surgeon and general network pruning', in *IEEE international conference on neural networks*, pp. 293–299. IEEE, (1993).

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, 'Deep residual learning for image recognition', in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, (2016).

[19] Yi He, Baijun Wu, Di Wu, Ege Beyazit, Sheng Chen, and Xindong Wu, 'Learning with capricious data streams: A generative approach', in *IJCAI*, (2019).

[20] Wassily Hoeffding, 'Probability inequalities for sums of bounded random variables', in *The Collected Works of Wassily Hoeffding*, 409–426, Springer, (1994).

[21] Bo-Jian Hou, Lijun Zhang, and Zhi-Hua Zhou, 'Learning with feature evolvable streams', in *Advances in Neural Information Processing Systems*, pp. 1417–1427, (2017).

[22] Guang-Bin Huang, Paramasivan Saratchandran, and Narasimhan Sundararajan, 'A generalized growing and pruning rbf (ggap-rbf) neural network for function approximation', *IEEE Transactions on Neural Networks*, **16**(1), 57–67, (2005).

[23] Ehud D Karnin, 'A simple procedure for pruning back-propagation trained neural networks', *IEEE transactions on neural networks*, **1**(2), 239–242, (1990).

[24] Michał Kempka, Wojciech Kotłowski, and Manfred K Warmuth, 'Adaptive scale-invariant online algorithms for learning linear models', *arXiv preprint arXiv:1902.07528*, (2019).

[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, 'Imagenet classification with deep convolutional neural networks', in *Advances in neural information processing systems*, pp. 1097–1105, (2012).

[26] Yann LeCun, John S Denker, and Sara A Solla, 'Optimal brain damage', in *Advances in neural information processing systems*, pp. 598–605, (1990).

[27] Yi Li and Philip M Long, 'The relaxed online maximum margin algorithm', in *Advances in neural information processing systems*, pp. 498–504, (2000).

[28] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou, 'Runtime neural pruning', in *Advances in Neural Information Processing Systems*, pp. 2181–2191, (2017).

[29] Shaohui Lin, Rongrong Ji, Yuchao Li, Yongjian Wu, Feiyue Huang, and Baochang Zhang, 'Accelerating convolutional networks via global & dynamic filter pruning.', in *IJCAI*, pp. 2425–2432, (2018).

[30] Lanlan Liu and Jia Deng, 'Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution', in *Thirty-Second AAAI Conference on Artificial Intelligence*, (2018).

[31] Christos Louizos, Karen Ullrich, and Max Welling, 'Bayesian compression for deep learning', in *Advances in Neural Information Processing Systems*, pp. 3288–3298, (2017).

[32] Haipeng Luo, Alekh Agarwal, Nicolo Cesa-Bianchi, and John Langford, 'Efficient second order online learning by sketching', in *Advances in Neural Information Processing Systems*, pp. 902–910, (2016).

[33] Mason McGill and Pietro Perona, 'Deciding how to decide: Dynamic routing in artificial neural networks', in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2363–2372. JMLR. org, (2017).

[34] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi, 'You only look once: Unified, real-time object detection', in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, (2016).

[35] Frank Rosenblatt, 'The perceptron: A probabilistic model for information storage and organization in the brain.', *Psychological Review*, **65**(6), 386, (1958).

[36] Rudy Setiono, 'A penalty-function approach for pruning feedforward neural networks', *Neural computation*, **9**(1), 185–204, (1997).

[37] Shai Shalev-Shwartz et al., 'Online learning and online convex optimization', *Foundations and Trends® in Machine Learning*, **4**(2), 107–194, (2012).

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

[39] Yu-Chuan Su and Kristen Grauman, 'Leaving some stones unturned: dynamic feature prioritization for activity detection in streaming video', in *European Conference on Computer Vision*, pp. 783–800. Springer, (2016).

[40] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung, 'Branchynet: Fast inference via early exiting from deep neural networks', in *2016 23rd International Conference on Pattern Recognition (ICPR)*, pp. 2464–2469. IEEE, (2016).

[41] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris, 'Blockdrop: Dynamic inference paths in residual networks', in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8817–8826, (2018).

[42] Yiming Ying and Massimiliano Pontil, 'Online gradient descent learning algorithms', *Foundations of Computational Mathematics*, **8**(5), 561–596, (2008).

[43] Qin Zhang, Peng Zhang, Guodong Long, Wei Ding, Chengqi Zhang, and Xindong Wu, 'Online learning from trapezoidal data streams', *IEEE Transactions on Knowledge and Data Engineering*, **28**(10), 2709–2723, (2016).