Rejection Ensembles with Online Calibration

Sebastian Buschjäger(✉)

The Lamarr Institute for Machine Learning and Artificial Intelligence,
TU Dortmund University, Dortmund, Germany
sebastian.buschjaeger@tu-dortmund.de

Abstract. As machine learning models become increasingly integrated into various applications, the need for resource-aware deployment strategies becomes paramount. One promising approach for optimizing resource consumption is rejection ensembles. Rejection ensembles combine a small model deployed to an edge device with a large model deployed in the cloud with a rejector tasked to determine the most suitable model for a given input. Due to its novelty, existing research predominantly focuses on ad-hoc ensemble design, lacking a thorough understanding of rejector optimization and deployment strategies. This paper addresses this research gap by presenting a theoretical investigation into rejection ensembles and proposing a novel algorithm for training and deploying rejectors based on these novel insights. We give precise conditions of when a good rejector can improve the ensemble's overall performance beyond the big model's performance and when a bad rejector can make the ensemble worse than the small model. Second, we show that even the perfect rejector can overuse its budget for using the big model during deployment. Based on these insights, we propose to ignore any budget constraints during training but introduce additional safeguards during deployment. Experimental evaluation on 8 different datasets from various domains demonstrates the efficacy of our novel rejection ensemble outperforming existing approaches. Moreover, compared to standalone large model inference, we highlight the energy efficiency gains during deployment on a Nvidia Jetson AGX board.

Keywords: Ensemble Learning · Learning with Rejection · Resource-aware Machine Learning

1 Introduction

In recent years, the pervasive integration of machine learning models into various applications has underscored the importance of resource-aware deployment. Most

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/978-3-031-70365-2_1.

A. Bifet et al. (Eds.): ECML PKDD 2024, LNAI 14946, pp. 3–20, 2024.
https://doi.org/10.1007/978-3-031-70365-2_1

famously, Deep Learning is one of the most resource-hungry technologies available, and therefore, a large body of the literature tries to improve the resource usage of Deep Learning (see [19] for a recent overview). Similarly, approaches for improving the resource usage of non-Deep Learning approaches, such as Random Forests [3,4] or graphical models [21], have also been discussed in the literature. Last, as ML models find their way into critical decision-making processes across diverse domains, there is a growing need for strategies that balance fast model application and the opportunity for human model inspection. This, again, leads to a resource-accuracy trade-off. For example, consider a medical scenario in which a machine-learning model autonomously diagnoses patients. Naturally, such a model will not always be correct, and human supervision and intervention are sometimes necessary. Hence, we have to balance human supervision and autonomous predictions during deployment.

One area of research reduces resource consumption through the fusion of small and large models into an ensemble coupled with a rejector (sometimes called a router) that determines the most suitable model for a given input [2,8, 12]. Such a rejection ensemble first applies the small model alongside the rejector and, if the rejector accepts the output of the small model, serves it. If the rejector rejects the small model's prediction, it also queries the big model (e.g., the doctor in the previous example) for additional help. Such an approach can reduce the overall resource consumption of the system if the small model is used most of the time while maintaining competitive predictive performance due to the big model.

The current literature on rejection ensembles mostly focuses on the ad-hoc design of rejection ensembles. The design and training of a good rejector is difficult and poorly understood. Moreover, while a budget is typically introduced during training to capture how often the big model can be queried, it is no longer considered during deployment. Hence, a deployed system might query the big model too often (i.e., overuse its budget) and – in the worst case – only query the big model if no additional safeguards are employed.

To address these issues, this paper presents the first theoretical investigation of learning rejection ensembles and derives practical insights from it. To this end, we propose a novel algorithm for training the rejector based on our theoretical insights and introduce a novel algorithm that ensures that the budget is always kept during deployment. More precisely, our contributions are as follows:

- **Theoretical Investigation:** We offer a thorough theoretical investigation of the impact of the rejector on the ensemble. We give precise conditions of when a good rejector can improve the ensemble's overall performance beyond the big model's performance and when a bad rejector can make the ensemble worse than the small model. Second, we show that even the perfect rejector can overuse its budget during deployment. Third, we give an example of when a rejector should not trust the outputs of the small and big models but learn its own decision boundary based on the input data.
- **Novel Algorithm:** Based on our theoretical investigation, we propose to ignore any budget constraints during training but introduce additional

safeguards during deployment. For training the rejector, we introduce a novel training algorithm based on so-called virtual labels capturing when to use the small and when to use the big model. For deployment, we introduce safeguards that essentially rank the rejector's output during deployment and ensure we always adhere to the prediction budget.
– **Experimental Evaluation:** We experimentally evaluate our proposed algorithm on 8 datasets from various domains and execute it on a Nvidia Jetson AGX board. We show that our novel rejection ensemble outperforms other ensembles while keeping the budget during deployment. Moreover, we highlight that these rejection ensembles use less energy during deployment compared to simply running the big model. The code for these experiments is available under https://github.com/sbuschjaeger/rewoc.

This paper is organized as the following: Sect. 2 introduces the notation and related work. Section 3 presents our main theoretical findings, whereas Sect. 4 translates these into a practical algorithm. Section 5 then discusses the experiments, whereas Sect. 6 concludes the paper.

2 Notation and Related Work

We consider a supervised classification setting in which training and test points are drawn i.i.d. according to some distribution $\mathcal{D}$ over the input space $\mathcal{X} \subseteq \mathbb{R}^d$ of d-dimensional feature vectors and labels $\mathcal{Y} = \{1, \dots, C\}$, where C is the number of classes. We are interested in a classifier triplet that we call *Rejection Ensemble*:

$$f(x) = (f_s, f_b, r)(x) = \begin{cases} f_s(x) & \text{if } r(x) = 0 \\ f_b(x) & \text{else} \end{cases} \tag{1}$$

Conceptually, $f_s \colon \mathcal{X} \to \mathcal{Y}$ is a small model whose predictions can be easily explained by a human and/or a model that does not use many resources, e.g. a Decision Tree. Similarly, $f_b \colon \mathcal{X} \to \mathcal{Y}$ is a big model whose predictions cannot be easily explained, and its execution might require many resources, such as e.g. a large neural network running in the cloud. Naturally, we want to use the small model as often as possible to make predictions explainable and the overall system more resource-efficient. At the same time, a small model might not be powerful enough to provide (good) predictions for certain inputs. Hence, we use a rejection function $r \colon \mathcal{X} \to \{0, 1\}$ that outputs 1 if we should reject the small model's prediction and use the big model instead. For training the triplet (f_s, f_b, r), we have given a (user-defined) budget[1] $p \in [0, 1]$, which defines how often we can query the big model. For example, a budget of $p = 1$ means we can always query the big model, a budget of $p = 0$ means we should never query

[1] Sometimes this is called the coverage, if there is no big model available and the small model abstains from a prediction.

it, and everything in between allows for some queries of the small and the big model. Given a loss $\ell : \mathbb{R}^C \times \mathcal{Y} \to \mathbb{R}_{\geq 0}$ our goal is to find a model such that

$$f^* = (f_s^*, f_b^*, r^*) = \arg\min_f \mathbb{E}_{\mathcal{D}}[\ell(f(x), y)] \text{ s.t. } \mathbb{E}_{\mathcal{D}}[r(x)] \leq p \tag{2}$$

Since $\mathcal{D}$ is typically unknown, we use a labeled training dataset $\mathcal{S} = \{(x_i, y_i)\}_{i=1}^m$ to approximate Eq. 2 with its empirical counterpart:

$$f^* = (f_s^*, f_b^*, r^*) = \arg\min_f \frac{1}{m} \sum_{i=1}^m \ell(f(x_i), y_i) \text{ s.t. } \frac{1}{m} \sum_{i=1}^m r(x_i) \leq p \tag{3}$$

For convenience, we further define the predictions of f_s, f_b, f as y_s, y_b, y_f and the corresponding confidences as c_s, c_b, c_f:

$$y_s(x) = \arg\max_{j=1,\dots,C} f_s(x)_j, \; y_b(x) = \arg\max_{j=1,\dots,C} f_b(x)_j, \; y_f(x) = \arg\max_{j=1,\dots,C} f(x)_j$$
$$c_s(x) = \max_{j=1,\dots,C} f_s(x)_j, \; c_b(x) = \max_{j=1,\dots,C} f_b(x)_j, \; c_f(x) = \max_{j=1,\dots,C} f(x)_j$$

2.1 Related Work

Several approaches in the literature focus on classification with a reject option. Arguably, the largest collection of works focuses on training a classifier tuple (f, r) where f is the prediction model and r is the rejector. The rejector can output a designated `REJECT` token, meaning that the model's prediction should be ignored. In this setting, the rejector is chosen such that f covers a certain percentage of the input space, and f is chosen to maximize the classification performance on the covered subspace. This way, a trade-off between the (likely better) performance of f on smaller subspaces and maximizing the coverage through r is introduced. The first works [5,6] in this area introduce a cost model (c.f. [23]) that balances the costs of rejection and its miss-classification. Subsequent theoretical works in this direction further refine this idea by developing new loss functions based on the hinge loss [1], introducing Bayes consistent loss functions [18], and studying the Rademacher complexity of such a classifier pair [7]. A second line of research is introduced in [22], sometimes called the bounded improvement model (c.f. [23]), which does not assign specific costs to rejection but views training a pair (f, r) as a min-max problem in which the goal is to maximize coverage while minimizing the error. A recent example is presented in [10], which introduces a novel 'selection with a guaranteed risk' algorithm that dynamically adjusts the bounds for confidence scoring of a pre-trained classifier. Similarly, SelectiveNet [11] is a neural network architecture that includes a reject option and is trained based on a convex combination of classification and coverage loss. Finally, some works in the literature also discuss the joint training of the model and the rejector without explicitly considering costs or coverage. For example, [15] studies the joint learning of the rejector and classification model by drawing inspiration from portfolio theory. Here, the authors introduce the

`REJECT` token as an additional class and a novel information-centric loss function that uses the `REJECT` token for better optimization. Last, Madras et al. study an edge case similar to our setting in [17], in which the small model can `PASS` an observation to a domain expert. However, their study focuses on fairness and does not introduce a budget constraint for optimization.

The framework presented in this paper extends prior work by using a classifier triplet (f_s, f_b, r) instead of a classifier tuple. Clearly, this generalization recovers the classification with rejection framework by setting the big model to output a constant reject value $f_b(x) = \texttt{REJECT}$ so that whenever $r(x) = 1$, we output the `REJECT` token. Training such a triplet is much more difficult because we cannot rely on the coverage as a guideline: When training a classifier with a reject option, we can essentially ignore parts of the input space by training the rejector r accordingly. However, when training a triplet, rejecting an observation means that the big model is tasked to provide a prediction. Hence, the rejector must take the shortcomings of the small and the big models into account and *only* transfer those samples to the big model when it can be sure that the big model will likely answer correctly to minimize the overall resource consumption. To our knowledge, only three articles in the literature utilize this more general framework. In [12], the authors propose a novel hybrid learning method in which a triplet is trained using a Frank-Wolfe-style algorithm. First, they start with a random rejector, which assigns training examples to a large and a small Deep Neural Network. After these two models are trained, the rejector is updated based on the overall performance, and the process is then repeated until convergence. Notably, the rejector only receives the outputs of the small model as input to minimize resource consumption. The resulting hybrid system achieves better ImageNet accuracy over inference latency and energy used. In [2,8], the authors deal with the problem of Human Activity Recognition by using a similar hybrid system that combines models from different model classes, i.e., a decision tree (DT) and a CNN. In [8], the authors introduce a multi-step learning process that first fits a decision tree on the entire task (i.e., all samples with all labels) and then iteratively merges the labels of samples that are too difficult in a common `FALLBACK` class. Finally, a DT is trained on all unchanged 'easy' samples and all difficult samples with the novel `FALLBACK` class assigned to them, whereas a CNN is trained on all available difficult samples with their original classes. While [8] shows the feasibility of this approach on a microcontroller unit (MCU), [2] goes one step further and deploys the DT model to the sensor directly by leveraging in-sensor computation and only executing the CNN on the MCU when necessary.

3 A Theoretical Investigation of Rejection

The adaption of Rejection Ensembles shows promising successes in practice [2,8, 12]. We extend this work by investigating the theoretical properties of Rejection Ensembles with a particular focus on the rejector r. To this end, we assume that the small model f_s and the big model f_b are already trained and given to us for deployment. We assume that f_b generally performs better than f_s, but

we do not have any special requirements towards f_s and f_b. In particular, our discussion here does not assume any special model class or training algorithm for the small and big models. Last, we assume that during deployment we receive data in batches $\mathcal{T} = \{x_1, \ldots, x_N\}$ of N data points for prediction and our task is to provide a labeled set $\{(x_1, f(x_1)), \ldots, (x_M, f(x_N))\}$. We highlight three theoretical insights about the rejector in this setting.

3.1 Three Distinct Situations Can Occur When Training the Rejector

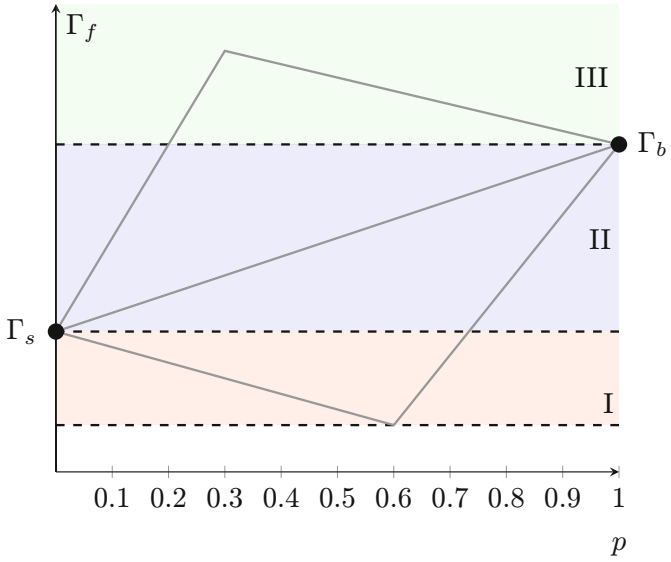

Fig. 1. Number of correctly classified samples over the rejection rate p for three archetypical examples. $\Gamma_{\{f,s,b\}}$ denotes the number of correctly classified samples of the ensemble and the small and big model respectively. The green area III marks improved performance in which the ensemble outperforms the big model while being more resource-efficient. The blue area II marks an accuracy-resource trade-off in which the ensemble underperforms compared to the big model but uses fewer resources. Finally, the red area I marks where the combined model performs worse than the small model and is more resource-hungry. The example curves in grey illustrate three different archetypical behaviors, although they highly depend on the specific models, data, and task. (Color figure online)

We identify three distinct situations that can occur when training a rejector: 1) A bad rejector can destroy the performance of both models by always choosing the wrong model for a given sample. For example, if the small model is correct, it might choose the big one instead, and if the big one is correct, it chooses the

small model. 2) If the big model is always better than the small model, i.e., $\forall x, y \sim \mathcal{D} : \ell(f_b(x), y) \leq \ell(f_s(x), y)$ then the Rejection Ensemble will never be better than the big model. However, the rejector can try to find situations in which the small model makes the same prediction ($y_s(x) = y_b(x)$) as the big model. Compared to f_b this will not increase the overall accuracy, but it decreases resource consumption while maintaining a comparable performance. 3) Both models complement each other through the rejector. Whenever f_s would be wrong, the rejector uses f_b instead, and whenever f_s is correct (and f_b might be wrong), f_s is used. This way, the overall accuracy might exceed the accuracy of f_b, resulting in a better *and* more resource-friendly ensemble. An illustration of these three cases is depicted in Fig. 1 and Theorem 1 formally establishes the conditions for each case.

Theorem 1. *Consider a binary classification problem with $C = 2$. Let (f_s, f_b, r) be a rejection ensemble with budget p and let $\mathcal{S} \sim \mathcal{D}^m$ be a sample with m data points. Define the following:*

$$\Gamma_s = \sum_{(x,y)\in\mathcal{S}} \mathbb{1}\{y_s(x) = y\}, N_{s\bar{b}} = \sum_{(x,y)\in\mathcal{S}} \mathbb{1}\{y_s(x) = y \neq y_b(x)\}$$
$$\Gamma_b = \sum_{(x,y)\in\mathcal{S}} \mathbb{1}\{y_b(x) = y\}, N_{\bar{s}b} = \sum_{(x,y)\in\mathcal{S}} \mathbb{1}\{y_b(x) = y \neq y_s(x)\}$$
$$\Gamma_f = \sum_{(x,y)\in\mathcal{S}} \mathbb{1}\{y_f(x) = y\}, N_{sb} = \sum_{(x,y)\in\mathcal{S}} \mathbb{1}\{y_s(x) = y_b(x) = y\}$$

Let $P = \lfloor p \cdot m \rfloor$, then the following holds:

1. *For all rejectors the following lower bound holds: $N_{sb} + \max\{N_{s\bar{b}} - P, 0\} \leq \Gamma_f$*
2. *If $\Gamma_b - P \geq \Gamma_s$, then Γ_s is a lower bound for Γ_f, i.e. $\Gamma_s \leq \Gamma_f$*
3. *If $\Gamma_b \leq \Gamma_s + P$, then Γ_b is a lower bound for Γ_f, i.e. $\Gamma_b \leq \Gamma_f$*

Proof. It holds that $\Gamma_s = N_{sb} + N_{s\bar{b}}$ and $\Gamma_b = N_{sb} + N_{\bar{s}b}$. Further, it holds that

$$\Gamma_f = N_{sb} + N_{s\bar{b}} + \min\{P, N_{\bar{s}b}\} = N_{sb} + N_{s\bar{b}} + \min\{P, \Gamma_b - N_{sb}\}$$

since we can only use the big model up to P times and have to revert to the small model otherwise. Given this, we proof each statement separately:

1. A worst-case rejector would always choose the big model when the small model is correct and vice versa. For those samples on which both models agree this is impossible, i.e., N_{sb} is a trivial lower bound for the performance of the worst rejector. Moreover, due to the budget, the rejector can only choose the big model up to P times, leading to
$$N_{sb} + \max\{N_{s\bar{b}} - P, 0\} \leq \Gamma_f$$
2. We want to show that Γ_s is a lower bound for Γ_f given $\Gamma_b - P \geq \Gamma_s$:
$$\Gamma_s \leq \Gamma_f \Leftrightarrow \Gamma_s < N_{sb} + N_{s\bar{b}} + \min\{P, \Gamma_b - N_{sb}\} = \Gamma_s + \min\{P, \Gamma_b - N_{sb}\}$$
$$\Leftrightarrow 0 \leq \min\{P, \Gamma_b - N_{sb}\}$$

Note that $0 \leq P$ is true by definition of P. Similarly, $0 \leq \Gamma_b - N_{sb} \Leftrightarrow N_{sb} \leq \Gamma_b$ is true due to the assumption $\Gamma_b - P \geq \Gamma_s$.

3. We want to show that Γ_b is a lower bound for Γ_f given $\Gamma_b < \Gamma_s + P$:

$$\Gamma_b \leq \Gamma_f \Leftrightarrow \Gamma_b < N_{sb} + N_{s\bar{b}} + \min\{p, \Gamma_b - N_{sb}\} = \Gamma_s + \min\{P, \Gamma_b - N_{sb}\}$$

We check both min− cases individually:

$$\Gamma_b \leq \Gamma_s + \Gamma_b - N_{sb} \Leftrightarrow N_{sb} \leq \Gamma_s$$
$$\Gamma_b \leq \Gamma_s + P \Leftrightarrow \Gamma_b - P \leq \Gamma_s$$

The first case $N_{sb} \leq \Gamma_s$ always holds by definition of N_{sb} and Γ_s, whereas the second case $\Gamma_b - P \leq \Gamma_s$ holds by assumption. Hence, if $\Gamma_b - P \leq \Gamma_s$ holds, then $\Gamma_b \leq \Gamma_f$. □

Theorem 1 shows that both a good rejector and well-trained models are crucial for good performance. Surprisingly, it also implies that the small model is much more important for the overall performance because it will be queried most of the time. The big model only needs to perform well on those $P = \lfloor p \cdot m \rfloor$ data points on which the small model underperforms and hence has a much smaller impact overall. Therefore, a good rejector should always favor the small model as much as possible and carefully pick those P samples.

3.2 Even a Perfect Rejector Will Overuse Its Budget

Recall that the rejector in [2,8,12] is trained to pick the big model at most p times on average during training. Unfortunately, there is no guarantee that the rejector will satisfy this constraint during deployment without further safeguards. Consider a deployed model (f_s, f_b, r), and a given batch $\mathcal{T}$ of N samples that should be classified. Since during training $\mathbb{E}_{\mathcal{S}}[r(x)] \leq p$ we hope that during deployment it also holds that $\mathbb{E}_{\mathcal{T}}[r(x)] \leq p$ but clearly we can construct corner cases in which this is not true. More critically, even if the estimation of $\mathbb{E}_{\mathcal{D}}[r(x)] = \mathbb{E}_{\mathcal{S}}[r(x)]$ during training is perfect and we have the perfect rejector with $\mathbb{E}_{\mathcal{D}}[r(x)] \leq p$, then there is still a non-zero chance to find a sample $\mathcal{T}$ that forces us to overuse the big model breaking our budget p, i.e., a sample with $\mathbb{E}_{\mathcal{T}}[r(x)] > p$. Theorem 2 formalizes this insight. Its proof utilizes the fact that $\mathbb{E}_{\mathcal{T} \sim \mathcal{D}^N}[r(x)]$ is normally distributed around $\mathbb{E}_{\mathcal{D}}[r(x)]$ due to the central limit theorem. Conversely, for $N < \infty$, the standard deviation of this normal distribution is larger than 0 so that there is a non-zero chance to find values above the mean $\mathbb{E}_{\mathcal{D}}[r(x)]$.

Theorem 2. *Assume we have given the Bayes optimal classifier* $(f_s^*, f_b^*, r^*) = \arg\min_f \mathbb{E}_{\mathcal{D}}[\ell(f(x), y)]$ *s.t.* $\mathbb{E}_{\mathcal{D}}[r(x)] \leq p$ *and* $\mathbb{E}_{\mathcal{D}}[r(x)] \in (0, 1)$. *Then there exists a sample* $\mathcal{T} \sim \mathcal{D}^N$ *such that* $\mathbb{E}_{\mathcal{T}}[r(x)] = \frac{1}{N} \sum_{i=1}^{N} r(x_i) > p$.

Proof. Consider a sample $\mathcal{T} \sim \mathcal{D}^N$ and the empirical mean $\widehat{p} = \frac{1}{N}\sum_{i=1}^{N} r(x_i)$. For a sufficiently large N the empirical mean $\widehat{p}$ is normal distributed due to the central limit theorem with $\widehat{p} \sim \mathcal{N}(\mu, \sigma)$ where

$$\mu = \mathbb{E}_{\mathcal{D}}[r(x)]$$
$$\sigma = \frac{\mathbb{V}_{\mathcal{D}}[r(x)]}{\sqrt{N}} = \frac{\mu(1-\mu)}{\sqrt{N}}$$

Let Φ denote the CDF of a Gaussian distribution. Then there is a non-zero probability $P(\widehat{p} > p) = \Phi(\widehat{p}) > 0$, given $M < +\infty$ and $\mu(1-\mu) \neq 0$ which holds due to the assumption that $\mu = \mathbb{E}_{\mathcal{D}}[r(x)] \in (0, 1)$. Hence, there exists a sample $\mathcal{T}$ such that $r(x)$ is queried more often than p. □

3.3 A Rejector Should Not Trust f_s and f_b

Arguably, the most straightforward rejector that always adheres to the budget p only selects the big model up to $P = \lfloor N \cdot p \rfloor$ times during deployment. In this case, we do not necessarily need to train a rejector, as we could simply trust the small model's outputs to determine if it is in doubt about a sample or not. More formally, we query the small model for all N data points, sort them according to a confidence score (e.g., the model's uncertainty), and then select those P data points with the smallest scores to be predicted by the big model. The pseudocode for this algorithm is depicted in Algorithm 3.1, where $\mathcal{T}$ is the current batch and p is the budget. For general applicability (we will re-visit Algorithm 3.1 in the next section), we explicitly include the rejector r as a parameter. However, note that using the confidence scores of the small model as rejector means we set $r = f_s$ in the parameters, i.e., use `confidence_thresholding`$(f_s, f_b, f_s, \mathcal{T}, p)$ so that no rejector r is necessary.

Implicitly, this algorithm trusts that the small model can express its confidence accurately. Unfortunately, this is not guaranteed without further assumptions on f_s, and we argue that any rejector that blindly trusts the output probabilities of a model can be fooled. And indeed, we can easily construct a simple counter-example in which the big model is a decision tree of depth 2, which is overly confident (but wrong), and the small model is a decision tree of depth 1, which is uncertain (but correct) for some samples. Theorem 3 formally established this argument and shows that – in the worst case – the lower bound in Theorem 2 can be realized.

Algorithm 3.1: Confidence Thresholding.

Function `confidence_thresholding`$(f_s,\ f_b,\ r,\ \mathcal{T},\ p)$**:**
  $s \leftarrow (c_r(x_1), \ldots, c_r(x_N))$
  `// Confidences of` r
  $s, x \leftarrow \text{sorted}(s, x)$ `// Sort confidences ascending`
  $P \leftarrow \lfloor N \cdot p \rfloor$ `// Set cutoff`
  return $\{(x_i, y_s(x_i)\mathbb{1}\{c_r(x_i) > s_P\} + y_b(x_i)\mathbb{1}\{c_r(x_i) \leq s_P\})\}_{i=1}^{N}$

Theorem 3. *Given two models f_s, f_b, a batch $\mathcal{T}$ of N data points, and a budget $p \in [0,1]$, then confidence thresholding of the small model f_s, i.e., using* `confidence_thresholding`$(f_s, f_b, f_s, \mathcal{T}, p)$*, can have a performance that matches the lower bound 1 in Theorem 1, i.e.*

$$\Gamma_f = N_{sb} + \max\{N_{s\bar{b}} - P, 0\}$$

Proof. We give an example situation with two decision trees. Consider a one-dimensional example $x \in [0, 10]$ with two classes $y \in \{0, 1\}$ that are assigned by the following rule:

$$y = \begin{cases} 0 & \text{if } x \leq 5 \\ 1 & \text{if } 5 < x \leq 7.5 \\ 0 & \text{if } 7.5 < x \end{cases}$$

We have gathered the following training data: $\mathcal{S} = \{(3, 0.2), (3.5, 0.2), (4, 0.2), (6, 0.2), (6.5, 0.2), (8, 0.2), (8.5, 0.2), (9, 0.2), (9.5, 0.2)\}$. We trained two trees f_b and f_s as depicted in Fig. 2 using the CART algorithm. Clearly, f_b performs better most of the time than f_s. However, for any $x \in (7.25, 7.5)$, it is very confident in its prediction ($f_b(x) = 1$) and f_s is comparably unconfident ($f_s = 4/6$). Now consider a batch of N data points that fall exactly in the region of $x \in (7.25, 7.5)$. Here, we should always pick the small model because – although unconfident – it is correct. However, `confidence_thresholding` picks at least $P = \lfloor Np \rfloor$ predictions from the big model, leading to a performance of $\Gamma_f = N_{sb} + N_{s\bar{b}} - P$.

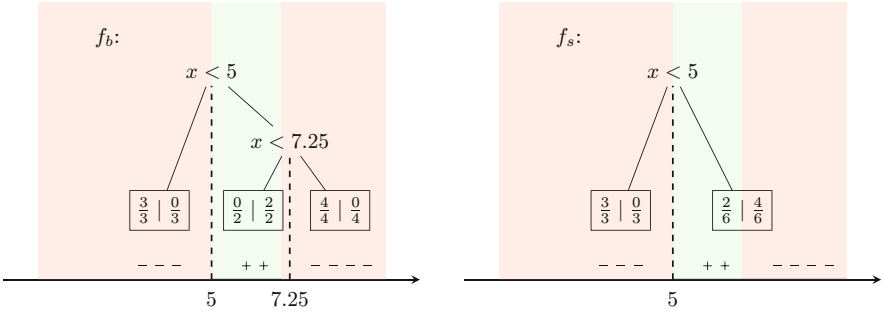

Fig. 2. Two decision trees trained on the sample $\mathcal{S} = \{(3, 0.2), (3.5, 0.2), (4, 0.2), (6, 0.2), (6.5, 0.2), (8, 0.2), (8.5, 0.2), (9, 0.2), (9.5, 0.2)\}$. The color-shaded area marks the correct class, and the samples are depicted as '-' and '+'. Each leaf node shows the empirical class probabilities for both classes, where the left entry represents the negative '-' class and the right entry represents the positive class '+'.

□

4 Training a Rejector for a Rejection Ensemble

The previous section discussed the theoretical properties of a (good) rejector, which we summarize as follows: First, a good rejector should not (blindly) use the confidence scores of the small model (c.f. Theorem 3). Hence, we argue against training the rejector on the outputs of the small model (i.e., $f_s(x)$) but propose using the input data x or intermediate transformations such as, e.g., embeddings of a Neural Network derived from f_s directly. Second, a rejector will likely overuse its budget during deployment. Hence, if the budget is a hard constraint, we must employ additional safeguards (c.f. Theorem 2). Therefore, we argue that we can simplify the training of r by ignoring any budget constraints during training, but we add additional safeguards during deployment that handle these constraints. Third, a rejector can improve the overall performance over f_b if f_s performs well and is sufficiently different from f_b. Hence, the training of r should favor the small model in all cases where it is correct and only use f_b if f_s is incorrect (c.f. Theorem 1). To this end, we propose Algorithm 4.1 for training a rejection ensemble. It receives the models f_s and f_b and applies them to the given training set $\mathcal{S}$. Then, it generates virtual labels that are 1 if both models disagree and the big model is correct. Otherwise, it assigns the label 0 to a sample. Finally, the rejector is trained on the original observations (or intermediate representations if available) with the new labels.

Algorithm 4.1: Training of Rejection Ensembles via virtual labels.

Function `fit`(f_s, f_b, $\mathcal{S}$)**:**
  $\mathcal{V} = \emptyset$ `// Virtual Labels for` r
  for $i = 1, \ldots, m$ **do**
    if $y_s(x_i) = y_b(x_i)$ **then**
      $v_i \leftarrow 0$ `// Both models agree. Pick the small model`
    else
      if $y_b(x_i) = y_i$ **then**
        $v_i \leftarrow 1$ `// Big model is correct, pick it.`
      else
        $v_i \leftarrow 0$ `// Big model is wrong. Use the small model.`
      end if
    end if
    $\mathcal{V} \leftarrow \mathcal{V} \cup \{(x_i, v_i)\}$
  end for
  `// Train` r `by minimizing` ℓ `over` $\mathcal{V}$. f_s, f_b `do not change`
  $r \leftarrow \arg\min_r \frac{1}{m} \sum_{(x,v)\in\mathcal{V}} \ell(r(x), v)$
  return (f_s, f_b, r)

Ensuring that the rejector satisfies the budget constraint during deployment is more challenging. For clarity, we now assume that the rejector outputs a confidence score, i.e., it is a function $r\colon \mathcal{X} \to [0, 1]$. Then we can use Algorithm 3.1 to

sort the confidence scores of the rejector in ascending order and only use the big model up to $\lfloor Np \rfloor$ times, i.e. using `confidence_thresholding`$(f_s, f_b, r, \mathcal{T}, p)$. Notably, this algorithm now assumes that $r(x)$ reflects the propensity of the rejector to favor the big model instead of trusting f_s. We argue that this is a more favorable scenario, as we can focus our energy entirely on training a good rejector instead of training three models at once.

5 Experiments

We now experimentally validate our findings in Theorem 1 and Theorem 2. Further, we show that training via virtual labels and online calibration outperforms existing methods. To do so, we perform two sets of experiments on the datasets listed in Table 1. The first experiment uses Deep Learning models evaluated on CIFAR100 and ImageNet, whereas the second experiment uses decision trees evaluated on several UCI datasets. We compare four different methods: For our baseline, we follow the established approaches of [2,8,12] by using confidence scores for training the rejector. More formally, we apply the small model to the training data and sort it according to the confidence score of the small model. Then, we assign a 1-label to the $\lfloor Np \rfloor$ examples with the smallest scores, whereas the remaining samples receive a 0-label. Finally, we train the rejector with these new labels. Note that, by construction, this approach satisfies the budget for the training data if the rejector is sufficiently accurate. We call this approach *confidence labels*. As a variation of this baseline, we combine a rejector trained via confidence labels with Algorithm 3.1, i.e., with confidence calibration, and call this method *confidence calibrated*. Third, we use Algorithm 4.1 to train the rejector and call this approach *virtual labels*, and finally, we combine Algorithm 4.1 and Algorithm 3.1 into *virtual labels calibrated*. The code for these experiments is available under https://github.com/sbuschjaeger/rewoc. Additional ablation studies on the UCI datasets can be found in the full version of the paper available in the code repository.

Table 1. Datasets used for the experiments.

Dataset	# Samples	Dimensionality	# Classes
ImageNet [24]	50 000	$3 \times 224 \times 224$	1 000
CIFAR 100 [14]	10 000	$3 \times 32 \times 32$	100
Anuran [13]	7 195	22	10
Covertype [13]	581 012	54	7
EEG [13]	14 980	14	2
Elec [13]	45 312	14	2
Gas Drift [13]	13 910	128	6
Weather [13]	18 159	8	2

As mentioned earlier, we are interested in an energy-efficient deployment in real-world scenarios. To measure the energy improvement under real-world circumstances, we perform all experiments on an Nvidia Jetson Orin AGX board. The Nvidia AGX board is a high-performance system-on-module (SoM) tailored for AI applications and marketed explicitly for model deployment. It offers 12 ARM CPU cores, 2048 CUDA cores, and 64 tensor cores combined with 64 GB of main memory. Its maximum power usage is 50 W, although we measured significantly less than that during our experiments. In total, *all* experiments can be run in roughly under 6 hours on the AGX, and we estimate a total energy consumption of around 540 kJ on this platform for all experiments, which equates to around 0.0444 KG CO2 given an average European energy mix. We are interested in the following questions:

1. Out of the four methods, which method performs the best overall?
2. Can a Rejection Ensemble improve over the performance of the big model?
3. How severely will the budget be overused if no calibration is done?
4. Does a Rejection Ensemble use less energy than the big model f_b?

5.1 Experiments with Deep Learning Models

For the Deep Learning experiments, we use the following setup: For the ImageNet experiment, we employed ShuffleNetV2 x0.5 [16] as the small model and Efficientnet-B4 [25] as the big model[2]. To evaluate the rejection ensemble's performance, we conducted a 5-fold cross-validation over the validation dataset of ImageNet, i.e., in each fold, we used one part of the validation dataset to train the rejector and the other part to test the ensemble. For CIFAR100, we also use ShuffleNetV2 x0.5 [16] as the small model, while RepVGG-A2 [9] acted as the big model[3]. Similar to the ImageNet experiment, we conducted a 5-fold cross-validation over the test dataset of CIFAR100. We tested different rejectors during pre-experiments but could not find meaningful differences. Hence, we use a Logistic Regression as the rejector trained via scikit-learn [20] trained on intermediate representations of the small model. In all experiments, we use $N = 32$ as the batch size during deployment and vary $p \in \{0, 0.1, \ldots, 1.0\}$.

Figure 3 shows the results for CIFAR100 (left column) and ImageNet (right column). As expected, the accuracy improves with increasing budget for all methods except *virtual labels without calibration*. Here, the method chooses always to use the small model, so it does not increase its performance. *Virtual labels calibrated* seems to be the best method, offering the highest accuracy, although it does not outperform the big model. Looking at the power consumption, we see a similar trend: As expected, with increasing usage of the big model, the power consumption increases but never exceeds the power consumption of the big model. However, on Imagenet, a notable plateau is visible for $p > 0.5$, in which the energy consumption is already close to the big model. Second, we see

[2] Obtained from https://pytorch.org/vision/stable/models.html.
[3] Obtained from https://github.com/chenyaofo/pytorch-cifar-models.

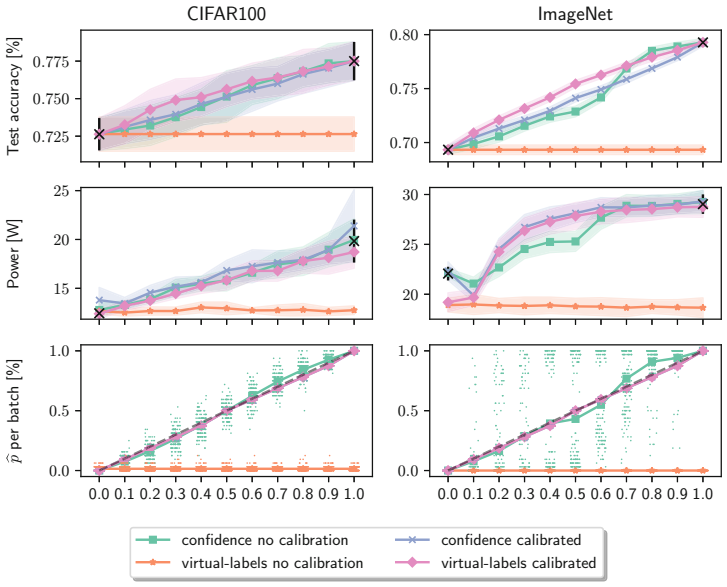

Fig. 3. Experimental results on CIFAR100 (left column) and ImageNet (right column). The first row shows the test accuracies, and the second row shows the energy consumption. The standard deviation is computed over the 5 cross-validation folds, and the crosses mark the small (left cross) and big model's (right cross) accuracy and power consumption. The third row shows how often the big model is queried for each batch $\widehat{p} = \frac{1}{N}\sum_{x,y\in\mathcal{T}} r(x)$, with one marker representing one batch. In all plots, the x-axis represents the given budget p.

that *confidence no calibration* seems to use less energy for some budget constraints. While we are not entirely sure why this is the case, we assume that our implementation of `confidence_thresholding` is sub-optimal[4] and we expect a more evolved implementation to have a better energy consumption. Looking at the relative use of the big model (third row), we see a mixed picture: First, we see that *virtual-labels no calibration* does not use the big model at all as expected from its test accuracy. Second, we see that all other methods increase their usage of the big model with a growing budget. Please note that the plots for *confidence calibrated* and *virtual-labels calibrated* overlap due to the calibration step here, so it is difficult to distinguish them in these plots. Both methods do not overuse their budget and choose the big model close to as often as the budget allows. For better interpretability, the gray line depicts the maximum usage allowed of the big model for a given budget: Anything above the gray line means we are overusing the budget, whereas everything below means we could have picked the big model more often. Most interestingly, we see that *confidence no calibration* does not overuse its budget on average, but there are many batches (a single

[4] Due to sorting, data needs to be transferred between the CPU and GPU.

marker represents one batch) in which the budget is not kept. In particular, on the ImageNet dataset, there are batches in which the big model is used for almost all data points, although the budget is close to zero. We conclude that *virtual-labels calibrated* is overall the best approach: It always keeps the budget while having a better test accuracy than the other methods with similar power consumption.

5.2 Experiments with Decision Trees

For the decision tree experiments, we use the following setup: Theorem 1 suggests that if the performance of the small model is close to the performance of the big model, then a Rejection Ensemble can improve its accuracy over the big model. To test this hypothesis, we use a small decision tree of depth three as the big model and a decision stump as the small model trained via scikit-learn [20]. Contrary to before, we now perform a 5-fold cross-validation and use the training data to train the initial small and big models. Then, we further split the testing data in each cross-validation run 50:50 into the training set for the rejector and the actual test set. Similar to before, we tested different rejectors during pre-experiments but could not find meaningful differences. Hence, we use a Logistic Regression as the rejector in all experiments, now trained on the original raw data. Similar to before, we use $N = 32$ as the batch size during deployment and vary $p \in \{0, 0.1, \ldots, 1.0\}$. For space reasons, we now focus on classification accuracy and refer interested readers to the full version of the paper for additional results.

Figure 4 shows the test accuracies for our UCI experiment. We highlight three observations here: First, *virtual labels no calibration* and *confidence calibrated* both do not seem to respect the budget at all as they have nearly a constant usage of the big model. Second, on the gast-drift, covertype, and weather datasets, we see an increase in the performance of the ensemble over the big model. Most notably, on the gas-drift dataset, we see an increase of nearly 10% points in accuracy. Third, on the weather dataset, we also see a decrease in performance below the small model when using *confidence no calibration*. We conclude that our analysis in Theorem 1 is correct and that the three different scenarios can occur in practical settings. Overall, we conclude that *virtual-labels calibrated* is the best method as it seems to offer the best accuracy over different budgets compared to the other methods.

5.3 Conclusion from the Experiments

Indeed, a Rejection Ensemble can improve its performance over the big model if the small and the big models have similar performances as predicted by Theorem 1. In many scenarios, even when the rejector keeps the budget on average, there are batches on which the big model is over- or underutilized, as implied by Theorem 2. Overall, we find that *virtual-labels calibrated* seems to be the best method, as it achieves the best accuracies while keeping the budget in all scenarios. Last, we also find that, while Rejection Ensembles always use less

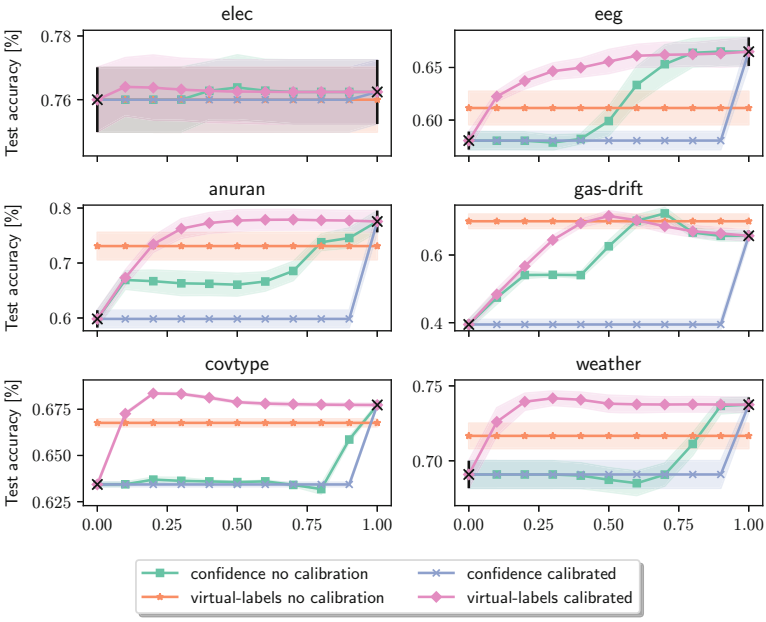

Fig. 4. Experimental results on the UCI datasets. All plots show the test accuracies over different budgets p. The standard deviation is computed over the 5 cross-validation folds, and the crosses mark the small (left cross) and big model's (right cross) accuracy.

energy than simply using the big model, there is some room for more improved implementations of the calibration step.

6 Conclusion

In this paper, we have presented a comprehensive investigation into rejection ensembles, addressing the crucial challenge of resource-aware deployment in machine learning systems. By leveraging a combination of small and large models with a rejector, Rejection Ensembles offer a promising approach to reducing resource consumption while maintaining competitive predictive performance. Our theoretical investigation has shed light on key aspects of rejector optimization and deployment, providing precise conditions under which rejection ensembles can outperform standalone models. Furthermore, we have proposed a novel algorithm for training and deploying rejectors based on theoretical insights that adhere to a given budget during deployment in all cases.

In addition to our theoretical investigation, we experimentally evaluated our novel algorithm. Our experimental evaluation on multiple datasets across various domains, executed on an Nvidia Jetson AGX board, has demonstrated the efficacy of our proposed rejection ensemble approach. In particular, we showed

that our theoretical investigation in Theorem 1 and Theorem 2 capture the real-world characteristics of Rejection Ensembles. Moreover, we also showed that our novel algorithm performs better than existing approaches.

Looking ahead, future research may focus on refining and extending our theoretical framework to the real-time setting in which we have given a single sample for which we immediately need to provide a prediction (i.e., $N = 1$). This scenario is somewhat ill-defined at the moment because it is unclear what the budget should encapsulate here, i.e., under what time horizon should we consider the budget? Once a precise definition is available, we can adapt our proposed calibration algorithm to the more challenging online scenario.

Acknowledgements. This research has partly been funded by the Federal Ministry of Education and Research of Germany and the state of North-Rhine Westphalia as part of the Lamarr-Institute for Machine Learning and Artificial Intelligence.

References

1. Bartlett, P.L., Wegkamp, M.H.: Classification with a reject option using a hinge loss. J. Mach. Learn. Res. **9**, 1823–1840 (2008)
2. Brehler, M., Camphausen, L.: Combining decision tree and convolutional neural network for energy efficient on-device activity recognition. In: 2023 IEEE 16th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), pp. 179–185 (2023)
3. Buschjäger, S., Morik, K.: Joint leaf-refinement and ensemble pruning through l_1 regularization. Data Min. Knowl. Discov. **37**(3), 1230–1261 (2023)
4. Chen, K.H., et al.: Efficient realization of decision trees for real-time inference. ACM Trans. Embed. Comput. Syst. (2021)
5. Chow, C.: On optimum recognition error and reject tradeoff. IEEE Trans. Inf. Theory **16**(1), 41–46 (1970)
6. Chow, C.K.: An optimum character recognition system using decision functions. IRE Trans. Electron. Comput. **6**(4), 247–254 (1957)
7. Cortes, C., DeSalvo, G., Mohri, M.: Learning with rejection. In: Ortner, R., Simon, H.U., Zilles, S. (eds.) ALT 2016. LNCS (LNAI), vol. 9925, pp. 67–82. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46379-7_5
8. Daghero, F., Pagliari, D.J., Poncino, M.: Two-stage human activity recognition on microcontrollers with decision trees and CNNs. In: 2022 17th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME), pp. 173–176 (2022)
9. Ding, X., Zhang, X., Ma, N., Han, J., Ding, G., Sun, J.: RepVGG: making VGG-style convnets great again. In: IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2021, Virtual, 19–25 June 2021, pp. 13733–13742. Computer Vision Foundation/IEEE (2021)
10. Geifman, Y., El-Yaniv, R.: Selective classification for deep neural networks. In: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems, pp. 4878–4887 (2017)
11. Geifman, Y., El-Yaniv, R.: Selectivenet: a deep neural network with an integrated reject option. In: Proceedings of the 36th International Conference on Machine Learning. ICML 2019, vol. 97, pp. 2151–2159. PMLR (2019)

12. Kag, A., Fedorov, I., Gangrade, A., Whatmough, P.N., Saligrama, V.: Efficient edge inference by selective query. In: The Eleventh International Conference on Learning Representations. ICLR 2023, Kigali, Rwanda, 1–5 May 2023 (2023)
13. Kelly, M., Longjohn, R., Nottingham, K.: UCI machine learning repository. https://archive.ics.uci.edu/
14. Krizhevsky, A.: Cifar-10 and cifar-100 datasets. https://www.cs.toronto.edu/~kriz/cifar.html
15. Liu, Z., Wang, Z., Liang, P.P., Salakhutdinov, R., Morency, L., Ueda, M.: Deep gamblers: learning to abstain with portfolio theory. In: Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019. NeurIPS 2019, pp. 10622–10632 (2019)
16. Ma, N., Zhang, X., Zheng, H.-T., Sun, J.: ShuffleNet V2: practical guidelines for efficient CNN architecture design. In: Ferrari, V., Hebert, M., Sminchisescu, C., Weiss, Y. (eds.) ECCV 2018, Part XIV. LNCS, vol. 11218, pp. 122–138. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01264-9_8
17. Madras, D., Pitassi, T., Zemel, R.S.: Predict responsibly: improving fairness and accuracy by learning to defer. In: Bengio, S., Wallach, H.M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018. NeurIPS 2018, 3–8 December 2018, Montréal, Canada, pp. 6150–6160 (2018). https://proceedings.neurips.cc/paper/2018/hash/09d37c08f7b129e96277388757530c72-Abstract.html
18. Mao, A., Mohri, M., Zhong, Y.: Theoretically grounded loss functions and algorithms for score-based multi-class abstention. CoRR abs/2310.14770 (2023)
19. Menghani, G.: Efficient deep learning: a survey on making deep learning models smaller, faster, and better. ACM Comput. Surv. **55**(12), 259:1–259:37 (2023)
20. Pedregosa, F., et al.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
21. Piatkowski, N., Lee, S., Morik, K.: Integer undirected graphical models for resource-constrained systems. Neurocomputing **173**, 9–23 (2016)
22. Pietraszek, T.: Optimizing abstaining classifiers using ROC analysis. In: Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, 7–11 August 2005. ACM International Conference Proceeding Series, vol. 119, pp. 665–672. ACM (2005)
23. Pugnana, A., Ruggieri, S.: A model-agnostic heuristics for selective classification. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 37, no. 8, pp. 9461–9469 (2023). https://doi.org/10.1609/aaai.v37i8.26133
24. Stanford Vision Lab, S.U.: Imagenet. https://www.image-net.org/
25. Tan, M., Le, Q.V.: Efficientnet: rethinking model scaling for convolutional neural networks. In: Proceedings of the 36th International Conference on Machine Learning. ICML 2019, vol. 97, pp. 6105–6114. PMLR (2019)