Sequential Model Specialization

Haichen Shen **Seungyeop Han** **Matthai Philipose**[†] **Arvind Krishnamurthy**

University of Washington, [†]Microsoft Research
{haichen, syhan, arvind}@cs.washington.edu, [†]matthaip@microsoft.com

Abstract

Recent advances have enabled high-capacity classifiers that can classify many classes with high accuracy across many settings. However, these classifiers are often slow, and many settings where they are applied have highly skewed data distributions that do not require their full complexity. We show how to combine bandit technology with cost-sensitive classification to detect and exploit these settings, a process we call *sequential model specialization*. When applied to recognizing faces in videos from YouTube, specialization yields end-to-end speedups of 1.9-3.8$\times$ relative to state-of-the-art convolutional neural networks, at competitive accuracy.

1 Introduction

The recent success of learning techniques that have the capability of absorbing extremely large data sets has yielded what may be termed *oracle* models. These are models that can classify accurately across many classes and a large variety of application settings. For instance, neural-network-based models for object, face and speech recognition are reaching the point where they can be used without re-training for general recognition tasks at competency levels equaling or exceeding that of humans. Oracle models, although generally applicable, tend to be computationally expensive to execute. On the other hand, many individual applications of these models only exercise a fraction of the complexity they are trained for. In this paper, we investigate the possibility of identifying settings where the underlying distribution is simpler than that addressed by the oracle, and using the oracle to derive simpler models that classify with equivalent accuracy but significantly lower execution latency in these settings, a process we call *model specialization*.

We view the oracle as classifying a stream of input data. If the elements of the stream are drawn independently from the general distribution that the oracle was trained on, we cannot, in general, do better than run the oracle for each element. However, in many common usages, the elements of the stream can be viewed as coming from an *abruptly-changing* [1], *skewed* non-stationary distribution. The underlying distribution remains fixed for relatively long stretches of time before switching, and moreover these fixed distributions are each significantly skewed toward small subsets of all classes addressed by the oracle. When processing video footage of the real world, for instance, depending on the scene (e.g., home, work, shop), a different small set of people may constitute a large fraction of the faces to be recognized; each such "scene" may be fixed for minutes to hours. In comparison, a state-of-the-art "oracle" face recognizer can recognize thousands of faces [5].

The above setting suggests the following scheme. Use the oracle to monitor class skew. If significant class skew exists, train and use a less expensive "specialized model" to classify for the duration of the skew. Switch back to the oracle when the underlying distribution switches, and repeat. We formalize this scheme into two loosely coupled problems that we together name the *sequential model specialization* problem. One problem is a non-stationary multi-arm bandit setup that switches between specialized models to minimize resource use, and the other a cost-sensitive classification problem to develop specialized models that have characteristics required by the bandit scheme. We show a cascaded model architecture that yields the cost-sensitive specialized model. These models are fast, accurate and fast to train. We also show how to adapt a windowed version of the Upper Confidence Bound (UCB) algorithm to perform switching. We justify our work empirically: we apply our framework to classifying video sequences using specialized convolutional neural networks that are 1.9-3.8x faster than state-of-the art baselines with competitive accuracy.

2 The sequential model specialization problem

We approach model specialization as a pair of loosely coupled problems. One is a sequential decision making problem to detect skews and select optimal models, and the other is a cost-sensitive classification problem to produce these models.

Let $x_1, x_2, \ldots, x_i, \ldots \in X = \mathbb{R}^n$ be a stream of values to be classified. Let $y_1, y_2, \ldots, y_i, \ldots \in Y = [1, \ldots, k]$ be the corresponding classification results. Let $\pi : \mathbb{I}^+ \to \mathbb{I}^+$ be a partition over the values. Associate the distribution T_j with partition j, so that each pair (x_i, y_i) is sampled from $T_{\pi(i)}$. Intuitively, π partitions, or segments, $\ldots, x_i, \ldots$ into a sequence of "epochs", where elements from each epoch j are drawn independently from the corresponding stationary distribution T_j. Thus, for the overall series, samples are drawn from an abruptly-changing, piece-wise stationary distribution. At test time, we assume neither results y_i nor partitions π are known.

Let $h^* : X \to Y$ be a classifier, designated the "oracle" classifier, trained on distribution T^*. Intuitively T^* is a mixture of all distributions comprising the oracle's input stream: $T^* = \sum_j T_j$. Let $R(h^*)$ be the resources (e.g., execution cycles), assumed invariant across X, needed to execute h^* on any $x \in X$. Executing h^* incurs a cost of $R(h^*)$ with probability 1. At test time, on each input x_i, we can always consult h^* at cost $R(h^*)$ to get a label y_i with accuracy a^* on average.

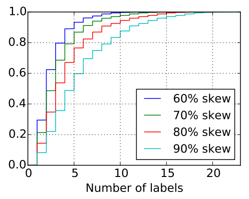
Alternately, we are free to consult one of many available "specialized" classifiers h_k instead. Intuitively, a non-stationary distribution T_j would likely have an associated specialized classifier that, on average, requires substantially fewer resources than the oracle on samples from that distribution. For simplicity, we assume for now that h_k agrees with h^* on all inputs x. Further, we require that for a given epoch j, the overhead $R(h_k)$ of executing h_k on input x_i can take one of two fixed values, (R_-, R_+) with probabilities $(p_{kj}, 1 - p_{kj})$. Intuitively, we expect that specialized models will have some inputs they "do well" on, and others they do poorly on, corresponding to lower versus higher resource costs. Finally, we require that $R_- \leq R(h^*) \leq R_+$ (specialized models whose best-case resource use exceed that of the oracle are useless), and that each specialized model h_k must have some epoch j where this holds with high probability p_{kj}, ideally with $R_- \ll R(h^*)$.

The overall problem is to select a model (whether specialized or oracle) to process each input x_i such that the aggregate resource use across all inputs is minimized. Ignoring details of how specialized models are obtained (and the cost of doing so), this problem can be framed as a piece-wise stationary multi-armed bandit problem [1]. Each classifier h_k (including the oracle, which can be regarded as h_0) is an arm of the bandit. The cost-distribution of arm k during the j-th stationary epoch is (R_-, R_+) with a priori unknown probabilities $(p_{kj}, 1 - p_{kj})$. Let h_j^* be the specialized model with lowest resource use for epoch j. If h_i is the classifier selected by our algorithm to process x_i, we wish to minimize the regret $\sum_i R(h_i) - R(h^*_{\pi(i)})$.

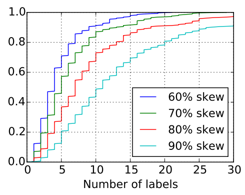
For regret to be low in absolute terms, the bandit scheme above requires that for any input distributions T_j it may encounter, it has access to specialized classifier h_k such that h_k has low overhead R_- with relatively high probability p_{kj}. Note, for instance that if all p_{kj} were zero, regret would be at least as much as using the oracle on every input, so that the specialized classifier would consume at least as many resources as the oracle. Further, the above setup requires that the specialized models are at least as accurate as the oracle over *any* input, and vary only in the resources required to process the input. Although in our solution below we are not able to meet this ideal, we are able to provide h_k's that match the oracle for specific epochs. This relaxation still allows useful solutions to the problem.

If the T_j were known offline, one could imagine training some form of cost-sensitive classifier that is especially cheap on a skewed distribution T_j with high probability. Note however, that the number of distributions to specialize over (i.e., the number of distributions T_j) is very large when the number of classes is large, even assuming techniques like quantization of distributions. Thus, in the case we address below, the T_j are inferred online by the bandit, and we need to *train* corresponding specialized classifiers h_k online. This brings up the challenging requirement that training overhead

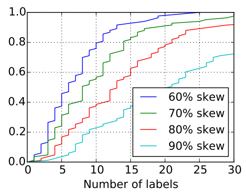
Figure 1: Samples of day-to-day video: movie clips (left), repair (mid), biking (right).



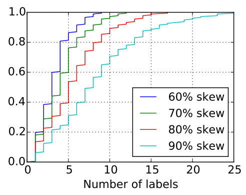
(a) 10s segments, objects



(b) 1 min seg., objects



(c) 3 min seg., objects



(d) 1 min seg., scenes

Figure 2: Temporal skew of classes in day-to-day video.

be minimized (since the bandit cannot provide access to the arm before it is done training) while ensuring that the resulting model is accurate as well. Further, if R_T is the training cost, the bandit scheme needs to assess a one-time cost of R_T to access the model for the first time.

Below, we provide a framework that heuristically combines an exploration/exploitation scheme with a novel cascaded classifier to find and exploit opportunities for sequential model specialization.

3 Temporal class skew in day-to-day video

Specialization depends on temporal skew in the distribution of classes presented to the classifier. In this section, we analyze the skew in videos of day-to-day life culled from YouTube. We assembled a set of 30 videos of length 3 minutes to 20 minutes from five classes of daily activities: socializing, home repair, biking around urban areas, cooking, and home tours. Figure 1 shows sample frames from these videos. We expect this kind of footage to come from a variety of sources such as movies, amateur productions of the kind that dominate YouTube and wearable videos.

We sample one in three frames uniformly from these videos and apply state-of-the-art face (derived from [5]), scene [6] and object recognizers [4] to every sampled frame. Note that these "oracle" recognizers can recognize up to 200 faces[1], 205 scenes and 1000 objects respectively. For face recognition, we record the top-scoring label for each face detected, and for the others, we record only the top-scoring class on each frame. For object recognition in particular, this substantially undercounts objects in the scene; our count (and specialization) applies to applications that identify some distinctive subset of objects (e.g., all "handled" objects, such as the iron in the middle frame of Figure 1). We seek to compare these numbers to the number of distinct recognized faces, scenes and objects that dominate "epochs" of τ = 10 seconds, 1 minute and 3 minutes.

Figure 2 shows the results for object recognition and scene recognition. We partition the sequence of frames into segments of length τ and show one plot per segment length. Each line in the plot corresponds to percentage skew $s \in \{60, 70, 80, 90\}$. Each line in the plots shows the cumulative distribution representing the fraction of all segments where n labels comprised more than s percent of all labels in the segment. For instance, for 10-second segments (Figure 2(a)), typically roughly 100 frames, 5 objects comprised 90% of all objects in a segment 60% of the time (cyan line), whereas they comprise 60% of objects 90% of the time (dark blue line). In practice, detecting skews and training models to exploit them within 10 seconds is often challenging. As figures (b) and (c) show, the skew is less pronounced albeit still very significant for longer segments. For instance, in 90% of 3-minute segments, the top 15 objects comprise 90% of objects seen. The trend is similar with faces and scenes, with the skew significantly more pronounced, as is apparent from comparing figures (b) and (d); e.g. the cyan line in (d) dominates that in (b). We expect that if we ran a hand-detector

[1]Note that the classifier [5] recognizes 4000 distinct people with over 90% accuracy. However, their training set is not public. We have considerably less training data.

Task	Model	Accuracy	FLOPs	Weights	CPU latency (ms)	GPU latency (ms)
Object Recognition (1k classes)	Obj-VGG	68.4%	39.3G	144M	3120	36.7
	O1	48.9%	0.82G	34.1M	209.6 ($\times$14.9)	6.13 ($\times$6.0)
	O2	42.6%	0.43G	8.1M	105.9 ($\times$29.5)	4.81 ($\times$7.6)
Scene Recognition (205 classes)	Scene-VGG	58.1%	30.9G	135M	2535	30.2
	S1	48.9%	0.55G	11.7M	150.3 ($\times$16.9)	4.21 ($\times$7.2)
	S2	40.8%	0.43G	6.5M	139.2 ($\times$18.2)	3.63 ($\times$8.3)
Face Recognition (200 classes)	Face-NN	93.11%	225M	21.9M	71.2	6.6
	F1	83.13%	183M	0.58M	57.8 ($\times$1.2)	4.1 ($\times$1.6)
	F2	70.15%	39M	0.15M	6.2 ($\times$11.5)	1.6 ($\times$4.1)

Table 1: Oracle classifiers versus compact classifiers in top-1 accuracy (without oversample), number of FLOPs, total weights, and CPU and GPU execution time in feedforward. The execution time was measured as the feedforward time of a single image without batching by Caffe [2]. We ran the experiments on a Linux server with a 24-core CPU Intel Xeon E5-2620 and an NVIDIA K20c GPU.

and only tried to recognize objects in the hand (analogously to recognizing detected faces), the skew would be much sharper.

Specialized models must exploit skews such as these to deliver appreciable speedups over the oracle. Typically, they should be generated in much less than a minute, handle varying amounts of skew gracefully, and deliver substantial speedups when inputs belong to subsets of 20 classes or fewer out of a possible several hundred in the oracle.

4 Specializing Models

In order to exploit skews in the input, we cascade the expensive but comprehensive oracle model h^* with a (hopefully much) less expensive "compact" model h_c. h_c is designed so that if its input belongs to the frequent classes in the incoming distribution it will return early with the classification result, else it will invoke the oracle model. Thus if the skew dictates that n frequent classes, or *dominant classes*, comprise percentage p of the input, model execution will cost $R_- = R(h_c)$ (the overhead of just executing h_c) roughly p% of the time, and $R_+ = R(h_c) + R(h^*)$ (executing h_c and h^* sequentially) the rest of the time. When p is large, the lower cost R_- will be incurred with high probability. Note that this specialized model clearly satisfies the criteria from Section 2, as long as we can establish that h_c uses far fewer resources than the oracle, and that the accuracy of the cascade is comparable to that of the oracle. To establish the latter, we need a detailed understanding of h_c.

To be more concrete, we use state of the art convolutional neural networks (CNNs) for oracles. In particular, we use the 19-layer VGG Net [4] (denoted as Obj-VGG) as our baseline model, or the complete classifier, for object recognition; the VGG Net 16-layer version for scene recognition [6] (Scene-VGG); and a derivative of the DeepFace network [5] for face recognition (Face-NN). We do not use the original DeepFace model because we have much less training data than it was designed for, so that DeepFace-architecture-based "oracle" has low accuracy. This substitution downplays the effectiveness of specialization, since DeepFace is far more resource hungry than Face-NN.

The compact models are also CNNs. For these, we use architectures derived from the corresponding oracles by systematically (but manually) removing layers, decreasing kernel sizes, increasing kernel strides, and reducing the size of fully-connected layers. The end result are architectures (O[1|2] for objects, S[1|2] for scenes and F[1|2] for faces) that use noticeably less resources (Table 1), but also yield significantly lower average accuracy when trained and validated on unskewed data, i.e., the standard training and validation sets. For instance, O1 requires roughly 91$\times$ fewer FLOPs to execute than VGG19, but achieves roughly 60% of its accuracy. However, in our approach, we train these architectures to classify *skewed* distributions observed during execution, and their performance on skewed distributions is the critical measure. In particular, to generate a specialized model, we create a new training dataset with the data from the n dominant classes of the original data, and a randomly chosen subset from the remaining classes with label "other" such that the dominant classes comprise p percent of the new data set. We train the compact architecture with this new dataset.

Figure 3 shows how compact models trained on skewed data and cascaded with their oracles perform on validation data of different skews. Figure 3(a) analyzes the case where $n = 10$, for various combinations of training and validation skews for model O1. Recall from Table 1 that O1 uses roughly 45$\times$ fewer FLOPs while delivering only 70% of its accuracy on unskewed inputs. However,

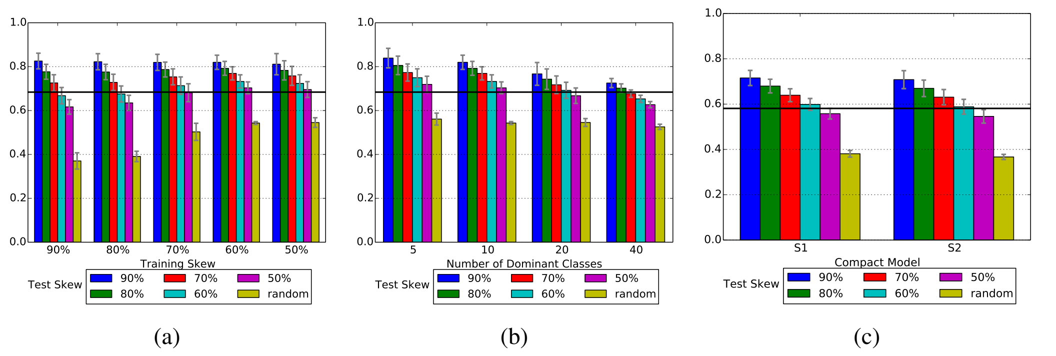


Figure 3: (a) Accuracy of compact model O1 trained by various skews and tested by various skews for 10 dominant classes; (b) Accuracy of O1 trained by 60% skew and tested by various skews for different number of dominant classes; (c) Accuracy of scene classifiers trained by 70% skew and 10 dominant classes. All experiments repeated 10x with randomly selected dominant sets.

when training and testing is on skewed inputs, the numbers are much more favorable. When O1 is trained on p=90% skewed data with n=10 dominant classes, it delivers over 84% accuracy on average (the left-most dark-blue bar). This is significantly *higher* than the oracle's average of 68% (top-1 accuracy), denoted by the horizontal black line. An important question is whether, when testing skews deviate from that used for training (typically as new classes enter or old classes leave the dominant set), performance falls catastrophically. As the remaining bars in the left-most cluster of Figure 3(a) show, performance in fact degrades gracefully. Figure 3(b) and (c) show that these effects are quite robust. In (b), where n is varied for O1 with p=60% shows, specialized cascades retain accuracy for quite large n. And in (c), which reports on similar measurements on compact models S1 and S2 for scene recognition shows, these trends carry to scene models. (Similarly for face, not shown.)

Finally, we note that since skews are only evident at test-time, specialized models must be trained extremely fast (ideally a few seconds at most). We use two techniques to accomplish this. First, before we begin processing any inputs, we train all model architectures on the full, unskewed datasets of their oracles. At test time, when the skew n, p and the identity of dominant classes is available, we only retrain the top (fully connected and softmax) layers of the compact model. The lower layers, being "feature calculation" layers do not need to change with skew. Second, as a pre-processing step, we run all inputs in the training dataset through the lower feature-calculation layers, so that when re-training the top layers at test time, we can avoid doing so. This combination of techniques allows us to re-train the specialized model in roughly 4s for F1 and F2 and 14s for O1/O2, many orders of magnitude faster than fully re-training these models. We fully expect that a combination of selecting training examples judiciously and using faster-to-train top layers (e.g., boosted ensembles or linear SVMs) can get this overhead well under a second.

5 Selecting Specialized Models

Recall from Section 2 that given specialized models with the right characteristics, we need to solve a piecewise-stationary multi-arm bandit problem. Each arm of this problem is a specialized classifier (i.e., a DNN architecture trained with skew n, p), the cost distribution is simply the distribution over the execution overhead of the specialized classifiers, and the regret is the average difference in cost between the selected classifier and the optimal one for the current skew.

In the stationary setting, a standard algorithm for solving the bandit problem is Upper-Confidence Bound (UCB) algorithm [3], which at each step selects the arm with the highest upper confidence bound constructed from past observed rewards, typically after a relatively long "exploration" phase to gather support for calculating the confidence bound. To address non-stationarity, a standard approach [1] is to restrict the context-search to a fixed-sized window (say of length w).

Our algorithm (Algorithm 1) is heavily influenced by these approaches but contains some heuristically motivated differences. We pick the arm with the lowest resource use as long as its estimated accuracy based on past observed rewards over the current window compares well with that of the oracle (Line

Algorithm 1 Non-stationary bandit algorithm for sequential model selection

```
    Exploration Phase
 1: $y_t \leftarrow h^*(x_t)$ [Use the oracle classifier]
 2: $\hat{y}_t \leftarrow \hat{y}_{t-1} \oplus [y_t]$ [Update context]
 3: Among all classifiers with sufficient support, find specialized classifier $h_t$ and window size $w$ with lowest resource use $R(h_t)$ such that estimated $a_{h_t}(\hat{y}_t, w) > a^* + \varepsilon$ where $w \in W$ and $w_{min} \le w \le |\hat{y}_t|$. [See Equation 1 for estimating $a_{h_t}$; $W$ is a small discrete set of window sizes]
 4: If $h_t$ is found go to Exploitation Phase with $h_t, w$.

    Exploitation Phase with $h_k, w$
 5: $y_t \leftarrow h_k(x_t)$
 6: $\hat{y}_t \leftarrow \hat{y}_{t-1} \oplus [y_t]$.
 7: Compute the estimated accuracy $a_{h_k}(\hat{y}_t, w)$.
 8: $q_{bad} \leftarrow$ fraction of time steps in the last window $w$ that $a_{h_k}(\hat{y}_{t'}, w) < a^* + \varepsilon$
 9: if $q_{bad} > q_{threshold}$ then
10:     $\ell \leftarrow$ # the time steps in the Exploitation Phase.
11:     Increase $w_{min}$ to the next value in $W$ if $\ell < w \cdot r$; otherwise reset $w_{min}$.
12:     Reset $\hat{y}_t \leftarrow \emptyset$ and go to Exploration Phase.
```

3), instead of using the highest upper confidence bound constructed from past observed rewards. The upper confidence bound is hard to compute due to the complexity of our problem. Besides, it is acceptable that the specialized classifier selected by the algorithm is not the best one but has lower latency with equivalent accuracy to the oracle. Also, note that we do not exit the exploitation phase periodically, but only when the accuracy is no longer comparable to the oracle (Line 9-12). Because pulling an arm for the first time in our case has a quite high cost (the training overhead), we greedily exploit the specialized classifier as much as possible during exploration.

A central challenge in the algorithm is that the number of arms is prohibitively high, up to 2^N, where N is the total number of classes classified by the oracle. It is not practical to evaluate all arms at every time step during the exploration. We need a way to limit the number of arms to be considered. One intuition is that the specialized classifiers can achieve equivalent accuracy only when the input distribution is skewed towards a dominant subset. If the input distribution is skewed, these skewed classes should appear more frequently in the classification results, even though the classifier is not 100% accurate. Therefore, we use a minimum support number c to keep the classes that appear at least c times in the context $\hat{y}_t$, denoted by the outstanding class set. In Line 3 the algorithm only considers the specialized classifiers of which all dominant classes S_i are contained in the outstanding class set, i.e., they have "sufficient" support. The minimum support number c increases with the increase of the window size w. See supplementary material for the analysis of the relationship between minimum support number and window size w.

The second challenge is how to estimate the accuracy of a specialized classifier based on the context $\hat{y}_t$. Consider a specialized classifier h_i with dominant classes S_i and skew p. The average accuracy of h_i over the skewed input can be written as:

$$a_{h_i} = p \cdot a_{in} + p \cdot e_{in \to out} \cdot a^* + (1-p) \cdot a_{out} \cdot a^*, \tag{1}$$

where a^* is the accuracy of oracle h^*, a_{in} is the accuracy of h_i over dominant classes, $e_{in \to out}$ is the fraction of dominant inputs that h_i classifies as non-dominant ones, and a_{out} is the fraction of non-dominant inputs that h_i classifies as non-dominant (note that these inputs will be cascaded to the oracle). We observed that the values of parameters a_{in}, $e_{in \to out}$, a_{out} are stable for a given $|S_i|$ (standard deviation is usually within 0.05). Thus we pre-compute these parameters when $n = |S_i| = 5, 10, 20, 30$, averaging over 10 iterations, and use linear interpolation for the rest of n's. Now we only need to know how to obtain the skew of dominant classes p. Notice that we can compute the empirical skew $\hat{p}$ in the context $\hat{y}_t$ by $\hat{p} = \sum_{1 \le j \le |\hat{y}_t|} \mathbf{1}_{y_j \in S_i} / |\hat{y}_t|$. However, the empirical skew $\hat{p}$ differs from the real skew p because the classifier makes mistakes in the classification. Using $|S_i| = n$ and assuming the confusion matrix is uniform, then the observed skew $\hat{p}$ during the exploration phase is:

$$\hat{p} = p \cdot a^* + p(1-a^*)\frac{n-1}{N-1} + (1-p)(1-a^*)\frac{n}{N-1} \tag{2}$$

The first term in Equation 2 indicates the input belongs to S_i and the classification is correct. The second term means that the input belongs to S_i, the classification is wrong, but it is confused to other classes in S_i. The third term assumes the input doesn't belong to S_i and is confused to classes in S_i. Given that we already know $\hat{p}$, we can derive the real skew p from Equation 2. Similarly the observed skew $\hat{p}$ during the exploitation phase is:

$$\hat{p} = p(1 - e_{in \to out}) + p \cdot e_{in \to out} \left(a^* + (1 - a^*) \frac{n-1}{N-1} \right) + (1-p)(1 - a_{out}) + (1-p) a_{out} (1 - a^*) \frac{n}{N-1} \tag{3}$$

Finally, in order to adapt to the non-stationary distributions, we follow [1] in applying a sliding window as mentioned above. However, a fixed window size w does not work well for our problem. If the window size is too small, it may miss some dominant classes in the window when the size of dominant set is large. If the window size is too large, it will take longer to detect the distribution changes and spend more time in exploration, which reduces the chance for exploitation. So, instead, we adaptively explore window size and specialized classifier together during the exploration phase and choose the pair that has the lowest resource use $R(h_t)$ with comparable accuracy for exploitation (Line 3). We limit window sizes to the set $W = \{30, 60, 90, 120, 150, 180\}$ to reduce overhead.

We added three knobs to the algorithm that can be tuned to optimize the real system. First, we introduced ε in the comparison between the estimated accuracy of the specialized classifier and a^* to adjust the overall accuracy (Lines 3 and 8). ε can be negative if the system can tolerate some accuracy loss; otherwise it can be positive to be more conservative. Second, we slacken the criterion that exits the exploitation phase (Line 9). We allow the estimated accuracy to be lower than $a^* + \varepsilon$ in a fraction $q_{threshold}$ of time steps in the last window. This avoids premature exits from exploitation due to jitter in the distribution in last window. Third (Line 11), we use w_{min} and r to limit the minimum size of w in the exploration. If the time steps in the exploitation phase is fewer than $w \cdot r$, the algorithm increases the w_{min} to the next value in W. When exploitation exits prematurely, it is often because of dominant classes that cannot be identified due to the small window, or because the sampling distribution is not good enough due to insufficient samples. Thus, increasing w_{min} mitigates the oscillation between exploration and exploitation.

6 Evaluation

We have implemented the specializer and the decision engine with a classification runtime based on Caffe [2]. The system can be fed with videos to produce classification results by recognizing frames, as well as with images with timestamps for trace-driven evaluation. For our evaluation, we conducted experiments with both synthetically generated data and real video.

6.1 Synthetic experiments

It is hard to find labeled video data for recognition tasks. In order to study diverse settings, we therefore synthesized a time-series of images picked from standard large validation sets of CNNs we use. To generate the synthetic data, we first define for each segment the number of dominant classes, the skew, and the duration in minutes. For each segment, we assume that images appear at a fixed interval (1/6s) and that each image is picked from the testing set based on the proportion defining the segment. For an example of a segment with 5 dominant classes and 90% skew, we pre-select 5 classes and pick an image with 90% probability from the dominant classes and an image with 10% probability from the other classes at each time of image arrival over 5 minutes duration. Representatives of non-dominant classes are picked in a uniform random way. We also generate traces with two consecutive segments with different sets of dominant classes and skews to study the effect of moving from one context to the other.

Table 2 shows the average accuracies and average latencies for the recognition tasks with and without the specializer enabled. The results are averaged over 5 iterations for each experiment, and the latency shown is per-image. The specializer was configured to use the compact classifiers O1 for objects, S1 for scenes, and F2 for face recognition from Table 1. (We chose F2 for face recognition as the speedup with F1 against the original model is insignificant.) As shown in the table, when the specializer is enabled, we could get higher or comparable accuracy with reduced per-image latency compared to the case when the specializer is disabled. For the single segment case, the GPU latency speedup per-image was 1.5× to 2.3×, 1.4× to 2.3×, and 1.2× to 1.9×, for object, scene, and face, respectively. Note that there is no accuracy or latency changes in the (random) segment as the specializer is not triggered, and the results for the combined segments were about the middle point of the results for the two single segments.

	Object				Scene					Face			
Segments	disabled		enabled		disabled		enabled		Segments	disabled		enabled	
	acc(%)	lat(ms)	acc(%)	lat(ms)	acc(%)	lat(ms)	acc(%)	lat(ms)		acc(%)	lat(ms)	acc(%)	lat(ms)
(n=5,p=.8)	68.46	37.07	77.28	16.31	57.64	30.49	65.80	13.33	(n=5,p=.8)	91.80	6.39	90.00	5.51
(n=10,p=.8)	66.13	37.14	74.11	21.85	57.17	30.49	59.71	17.20	(n=5,p=.9)	92.83	6.37	90.38	3.31
(n=10,p=.9)	68.29	37.09	76.05	17.57	59.12	30.47	63.81	16.13	(n=10,p=.9)	93.34	6.38	91.47	5.05
(n=15,p=.9)	66.64	37.16	69.57	24.41	57.79	30.48	58.68	21.40					
(random)	66.72	38.67	66.72	39.50	59.13	30.52	59.13	30.43	(random)	92.99	6.39	92.99	6.32
(n=10,p=.9) +(random)	66.52	39.20	70.62	28.68	56.96	30.42	58.46	22.65	(n=5,p=.9) +(random)	92.64	6.29	91.33	4.70
(n=15,p=.9) +(n=5,p=.8)	68.09	37.08	73.03	20.40	61.12	30.42	65.63	16.28	(n=10,p=.9) +(n=5,p=.8)	93.43	6.35	91.14	5.14

Table 2: Average accuracy and GPU latency of recognition over a segment. For the segment column, each parenthesis indicates a segment of 5 minutes with the number of dominant classes and the skew.

Videos	disabled		enabled	
	accuracy (%)	latency (ms)	accuracy (%)	latency (ms)
Friends	93.39	5.12	96.73	2.08
Ocean's Eleven / Twelve	87.82	5.11	92.85	2.39
Good Will Hunting	86.48	5.28	100	1.39
The Departed	85.64	5.16	92.43	2.73
Ellen Show	93.36	5.01	99.48	2.00

Table 3: Accuracy and GPU latency on 5 videos with the specializer enabled or disabled.

6.2 Video experiments

We also evaluated our system for face recognition with real video clips. We collected clips from 4 movies and an interview and manually labeled the faces in the videos. For each video, we collected two to three clips and concatenated them to form a single video in order to have more distribution changes. The lengths of concatenated videos range from 3 minutes to 6 minutes. We downsampled the videos to 6 fps and cropped out the faces from every frame for labeling. We found the accuracy of our trained face neural networks on faces in the videos is much lower than the accuracy on our validation data. This is because the faces we collected from web pages do not have an extensive coverage of faces with different angles and perspectives, and faces with different expressions. Therefore, we added the faces of actors and actresses from other video clips (different ones from the testing video clips) to the training data, and applied affine transformation to the faces in the videos to have a better handle on the variation seen in the videos. We used F2 as the compact classifier in the evaluation.

Table 3 shows the average accuracies and average latencies for 5 videos. The results show that the specialized models consistently achieved higher accuracy with end-to-end decreases in GPU processing latency of 1.9× to 3.8×. We found that our system performed better in both accuracy and latency on real videos than on the synthetic traces. We attribute the gain to two reasons: (a) the real videos have higher skew and fewer dominant classes than the configurations used in the synthetic data, especially for the video clips from Good Will Hunting and Ellen Show, which only have two persons much of the time; (b) the added face data from other videos and affine transformation helps the specialized models maintain high accuracy.

7 Conclusion

We introduce the *sequential model specialization* problem to address the recent emergence of accurate, broadly applicable, but resource-hungry classifiers. The problem recognizes that classification inputs can often be seen as coming from a piecewise stationary distribution, and that for many of the pieces, relatively lean "specialized" classifiers may be adequate. We show how to combine a novel cascaded classifier design with bandit technology to address the problem. Our solution speeds up face recognition on YouTube videos by 1.9-3.8× with little loss in accuracy relative to a modern convolutional neural network.

References

[1] Aurélien Garivier and Eric Moulines. On upper-confidence bound policies for non-stationary bandit problems. In *Proceedings of the 22nd International Conference on Algorithmic Learning Theory (ALT)*, 2011.

[2] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia (MM)*, 2014.

[3] Tze Leung Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22, 1985.

[4] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[5] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. DeepFace: Closing the Gap to Human-Level Performance in Face Verification. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[6] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. In *Proceedings of the Twenty-eighth Annual Conference on Neural Information Processing Systems (NIPS)*, 2014.

index	N	a	n	p	w	c	p_{in}	p_{out}
1	1000	0.68	5	0.9	30	2	0.896	6.52E-5
2	1000	0.68	10	0.9	30	2	0.558	6.53E-5
3	1000	0.68	10	0.9	60	2	0.891	2.64E-4
4	1000	0.68	10	0.7	60	2	0.789	4.80E-4
5	1000	0.68	10	0.7	90	2	0.933	1.08E-3
6	205	0.58	10	0.9	90	2	0.959	0.019
7	205	0.58	10	0.9	90	3	0.872	1.31E-3
8	205	0.58	0	N/A	90	3	N/A	9.29E-3

Table 4: Probability p_{in} and p_{out} under various N, a, n, p and w, c. In row 8, $k = 0$ and $p =$ N/A indicates that the distribution has no skew and is uniform across all N classes.

Supplementary Material

We use a simple model to analyze how to choose the minimum support number c for different window sizes w in order to limit the number of specialized models to be evaluated in Algorithm 1 during exploration. Suppose N is the number of total classes classified by the oracle h^* and that the accuracy of h^* is a^*. If the classification result by the oracle is wrong, the probability that the oracle classifies the input to each of the other classes is assumed to be equivalent. Assume the input sequences are drawn independently from a skewed distribution T which has n dominant classes with skew p. Denote the dominant set as $\mathcal{S}$ and non-dominant set as $\mathcal{O}$. Among dominant or non-dominant set, the probability of each class is assumed to be the same. Based on these definitions, we can compute the probability of any single class being classified. Similar to Equation 1 in the paper, consider one dominant class ℓ, the probability that it will be classified by the oracle is:

$$p(\ell \in \mathcal{S}) = \frac{1}{n} \cdot \hat{p} = \frac{1}{n}\left(p \cdot a^* + p(1-a^*)\frac{n-1}{N-1} + (1-p)(1-a^*)\frac{n}{N-1}\right) \tag{4}$$

And we can compute the probability of a non-dominant class ℓ' being classified by:

$$p(\ell' \in \mathcal{O}) = \frac{1}{N-n}\left((1-p)a^* + (1-p)(1-a^*)\frac{N-n-1}{N-1} + p(1-a^*)\frac{N-n}{N-1}\right) \tag{5}$$

From Equation 4 and 5, we can then calculate the probability of a dominant class or a non-dominant class that is classified at least c times in the window w by treating it as a binomial distribution. Intuitively, we hope the probability of a dominant class that is classified at least c times (p_{in}) be as high as possible, while the probability of any non-dominant class (p_{out}) be as low as possible. We considered different combinations of N, a^*, n, p under different w and c settings, and computed p_{in} and p_{out}. Table 4 shows how p_{in} and p_{out} changes under different settings. From the table, we can tell that with an increase in the number of dominant classes and a decrease in skew, we need to increase the size of window to have a higher probability that the dominant classes can be detected in the window. However, when the window size is larger, we also need to increase the minimum support number c (Row 6 and 7) to limit the probability that a non-dominant class appears c times. In addition, when there is no skew in the distribution (Row 8), the minimum support number c is also effective at filtering out most of the non-dominant classes. In practice, we set $c = 2$ for $w < 90$ and $c = 3$ for $w \geq 90$.