Towards Flexible Classification: Cost-Aware Online Query of Cascades and Operating Points

Brandyn White, Andrew Miller, Tom Yeh, and Larry S. Davis

University of Maryland: College Park

Abstract. We propose a method that uses off-the-shelf binary classifiers and combines them in real-time to achieve quality, time, and cost characteristics that satisfy user-provided constraints; moreover, it shares intermediate computation (e.g., features, classifiers, kernels) between classes to reduce time and cost. An important distinction between this approach and others is that the primary focus of this work is on improving *flexibility*, operation on a wide range of operating points, as opposed to *quality* or *speed*; however, we show that the proposed approach achieves performance comparable to the state-of-the-art on the scene recognition task for the SUN397[1] where we attain a mean AUC of 0.9625 compared to 0.9573[1], 0.8426 [2], 0.9227[3], and 0.9343(based on [4]). Additionally, we show that human annotators can be naturally integrated into the approach to produce hybrid human/algorithmic classifiers.

1 Introduction

As the field of computer vision advances, outside research fields, industry practitioners, and average developers will increasingly seek to use vision algorithms as 'black boxes' to solve problems that may arise with visual data. What traits should a vision algorithm have to accommodate these users? The most obvious is high *quality*, which is the focus of the majority of work in the field. However, the best performing methods are often not fast enough for many practical use cases, which necessitates a focus on *speed* (e.g., gpus, approximate methods). When a task needs to be both fast and perform with high quality, it is common to enlist humans to help (e.g., Amazon Mechanical Turk - so called "turkers"). Methods that use excessive computation, exotic hardware, large datasets, or humans bring up the practical issue of *cost*. Most approaches develop a solution with respect to one of these traits; however, to be useful as a 'black box' the ability to interpolate between these traits is essential. We refer to this requirement as *flexibility* and it is the focus of this work.

Our primary contribution is a method for producing multiple binary classification algorithms, in the form of rejection-chain cascades, in real-time that satisfy specified quality, time, and cost constraints. A user's specification of the quality, time, and cost constraints is referred to as a "query". The approach re-uses features, kernels, and classifier computations amongst the binary classifiers. Our method consists of three phases - a training phase, a query phase, and an execution phase. We start from the premise that resolving a query (i.e.,

constructing an appropriate cascade of binary classifiers) must be efficient compared to the time required to perform the actual vision computations specified by the query resolution. Moreover, in order to support batch computation we seek a method that can resolve queries using only the provided constraints, independent from the input testing data. To enable this we shift the majority of the computation (i.e., construction of rejection cascades for per-class recognition) to an offline training phase, enabling query resolution to involve only simple database look-ups. We focus on scene recognition as an initial task due to its wide applicability; however, our approach is applicable to a wide range of binary classification tasks in vision (e.g., localization, segmentation, image classification). For binary classification tasks, the primary metrics of interest are execution time and those derived from binary confusion matrices (e.g., precision, recall, accuracy). After a "training phase" (see below) each cascade is stored in a *cascade database* and is indexed by precision, recall, accuracy, F-1 (i.e., harmonic mean of precision and recall), time, and cost. By combining multiple classifiers for a class in a rejection chain cascade, we dramatically increase the number of operating points (see Section 4.3), which provides *flexibility* during query resolution. While cascade stages are often created using specific families of features (e.g., Haar) and classifiers (e.g., decision stumps), our approach allows for heterogeneous off-the-shelf features and classifiers to be used (see Section 3.1).

2 Related Work

This work is related to existing classification methods that trade-off quality vs. time and share computation between classes. We now compare our approach to these methods.

There have been a variety of computer vision approaches[5][6][7] for automatically tuning algorithm parameters to improve quality and/or speed. The majority of these approaches operate on classifiers. Brubaker *et al.*[8] provide a principled method for designing and training boosted cascades to meet target recall and false positive rates for face detection. Bourdev and Brandt [9] introduced the Soft Cascade for face detection, which allows individual cascade stages to reject without causing the overall classifier to reject. Their algorithm is 'calibrated' after training; given a target detection rate and execution time, it minimizes the false positive rate. They reported that the calibration process takes up to 5 minutes, which is unsuitable for online queries. Visentini *et al.*[10] proposed a scheme to tune asymboost cascades for face detection. In their scheme, a trade-off between accuracy and complexity is made for each cascade stage. Our method differs from these in that ours (1) weighs trade-offs in less than a second (enabling online selection), (2) supports fine-grained performance specification (i.e., precision, recall, accuracy, F-1, time, cost), (3) optimizes across multiple classes, and (4) operates on heterogeneous features and classifiers.

Sharing computation between classes is a popular approach taken by systems that operate on multiple classes. Computation that is commonly shared between classes includes classifiers[7], features[6][11], and kernels[12][13]. There

are two primary reasons to exploit sharing between classes (1) to reduce computation and (2) to operate given few training examples for a class[14]. Each classifier stage of our approach has explicit class independent computation. After training, it generates a representative set of classifier schematics that can be efficiently queried to enable sharing between classes to reduce computation; moreover, it allows for re-using classifiers between classes (e.g., outdoor/indoor for scene tasks) which enables operation when few training examples are provided. This sharing is performed such that user-provided quality, time, and cost constraints on the resulting classifiers are met.

Gao and Koller[6] developed a multi-class classification system that actively evaluates which feature/classifier provides the most *value* given the outputs of previous stages. This is similar to ours in that it operates directly on classifier confidence values; however, our method differs in that the constraints are optimized independently from the input, where their method adds significant per-image computation (reported at 10ms for an image). Additionally, new classifiers can be added in our method without fully retraining. Gao and Koller previously presented an approach[7] that produces a multi-class classifier using a hierarchy of binary classifiers. Both of these approaches are similar to ours in that they explore the combination of heterogeneous classifiers, exploit sharing between classes (features[6] and classifiers[7]), and enable adjusting speed-accuracy trade-offs; however, ours differs in that (1) it allows for directly specifying desired classifier characteristics (e.g., accuracy, time, cost) where they use a trade-off parameter, (2) it produces binary classifiers that share computation between classes and theirs produces a multi-class classifier, and (3) our goal is to improve classifier *flexibility* by presenting a wide range of operating points, where theirs has a single operating point controlled by a trade-off.

Schwing *et al.*[15] proposed a multi-class classification approach that allows trading-off accuracy and speed without re-training. This is similar to our work in that the trade-off can be performed without re-training; however, they assume that each "expert" is equally knowledgeable and their trade-off selects the number of "experts" to consult, where our method selects from a heterogeneous set of classifiers with associated operating points.

3 Overview

We propose a method for selecting multiple binary classifiers in real-time given quality, time, and cost constraints. The binary classifiers produced by this method are rejection-chain cascades where the individual stages are heterogeneous, binary, and make their prediction independently, allowing each stage to be trained independently. This approach offers four advantages: (1) a large range of operating points (flexibility), (2) early rejection and reuse (speed), (3) combination of information sources (quality), and (4) straightforward analysis (simplicity). The cascade is not boosted because boosted stages can not be reordered at will, reducing expressivity and query performance. Fig. 1 provides an overview of the method and illustrates how the three datasets are used: training (train

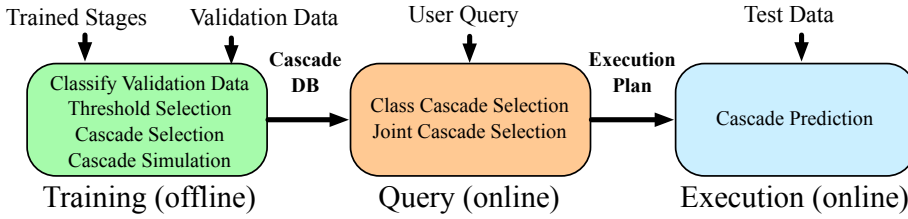


Fig. 1. Overview of our approach. The product of the training stage is a *cascade database* which is queried online during cascade selection. The cascade selection produces an *execution plan* that specifies which cascade to execute for each class. The order of execution is top-to-bottom and left-to-right.

cascade stages), validation (build cascade database), and testing (query cascade database). The query and execution phases are separated to emphasize that the execution plan is independent of the test data. Thresholds are selected for each cascade stage that characterize their performance over their operating range. We construct cascades using off-the-shelf cascade stages (see Section 3.1) with prediction confidence values for each image in the validation set. Each cascade's validation set predictions are then efficiently simulated over a range of thresholds. The result is a cascade database representing the joint performance of all cascades and thresholds. Each entry in the database is an operating point corresponding to a cascade and thresholds. We index the cascade database, allowing sub-second algorithm and operating point selection. Finally, we optimize constraints for multiple classes to minimize overall cost. Our primary contribution, then, is a method for generating multiple binary classifiers in real-time given desired quality, time, and cost constraints.

3.1 Cascade Stages

At a high level, a cascade stage S takes an image I and a set of classes $\mathcal{L}$ from which it produces a sparse confidence vector x^S. For the single class case in Fig. 2(a), this is a single image feature f and a single binary classifier c_1 which produces the confidence vector x^S as $x_1^S \leftarrow c_1(f(I))$. For the general form in Fig. 2(b), a single feature f is shared by many classifiers and x^S is produced as $\forall l \in \mathcal{L}, x_l^S \leftarrow c_l(f(I))$. We now provide a definition for a cascade stage S where I is an image and $\hat{\mathcal{L}}$ is the set of all supported classes (represented as positive integers).

1. The cascade stage is defined for all $\mathcal{L}$ where $\mathcal{L} \subseteq \hat{\mathcal{L}}$.
2. $x^S \leftarrow S(I, \mathcal{L})$ where S may be non-deterministic and x^S is a real-valued sparse vector with a value x_l^S defined if $l \in \mathcal{L}$ and undefined otherwise.
3. $\forall l, m \in \mathcal{L}$, confidence value x_l^S is independent of $m \in \mathcal{L}$ or $m \notin \mathcal{L}$ where $l \neq m$.
4. Larger values of x_l^S signify higher confidence that the input belongs to class l.

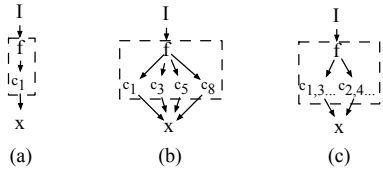


Fig. 2. The input to each cascade stage is an image I, and a subset of the available classes, $\mathcal{L} \subseteq \hat{\mathcal{L}}$. The cascade stage performs a class-independent computation f (e.g., a feature or a kernel), which may be shared between classes, followed by one or more class-specific computations (e.g., binary classifiers). The output is a vector x^S of confidence values for each class in $\mathcal{L}$. Example instantiations of our cascade stage model for a (a) single feature/classifier with $\mathcal{L} = \{1\}$, (b) single feature shared by four classifiers with $\mathcal{L} = \{1, 3, 5, 8\}$, and (c) all classes using the same classifier f with $\mathcal{L} = \{1, 2, \dots\}$ (used in Section 5). While only the classifiers needed (specified by $\mathcal{L}$) are shown, each stage is defined for all classes $\hat{\mathcal{L}}$. See text for details.

Observe that #3 ensures that the computation of x_l is independent from other classes in $\mathcal{L}$. Cascade stages are given the set of classes $\mathcal{L} \subseteq \hat{\mathcal{L}}$ (#1 above) to allow for prediction on only the classes required as an optimization. A significant advantage of this approach is that it allows for a wide variety of cascade stages. In the previous example f was described as an image feature; however, it could also correspond to a kernel matrix that is shared amongst the classifiers (used in Section 7). While our approach produces binary classifiers, the classifiers, represented by c, can themselves be multi-class classifiers. For example, a cascade stage S using a k-nearest neighbor classifier can output the # of nearest neighbors as x_l^S for $l \in \mathcal{L}$.

In Section 5 we show that a human on Amazon Mechanical Turk (AMT) can be modeled as shown in Fig. 2(c) where all classes are scenes and belong to either outdoor (odd) or indoor (even). In this example, the 'turker' is represented as f and outputs a scalar confidence value for the outdoor class. The $c_{1,3\dots}$ is an identity function, it outputs the confidence value ($x_l^S \leftarrow f(I)$), and $c_{2,4\dots}$ negates the confidence value ($x_l^S \leftarrow -f(I)$). This shows that a classifier (i.e., a turker represented by f) can be used by multiple classes with the c's merely orienting the confidence values. 'Turkers' can produce different results for the same image; however, this is supported as the cascade stages can be non-deterministic (#2 above).

In the previous examples, we have used f to represent an image computation that is performed for all classes (e.g., image feature, kernel matrix) and c to represent a computation that is specific to each class (e.g., classifier). We then define $\tau_{S,f}$ to represent the time to compute f for a cascade stage S and τ_{S,c_l} as the time to compute c for class l. Additionally, we denote $\lambda_{S,f}$ and λ_{S,c_l} to represent the monetary cost (e.g., computational resources, AMT) for class-independent and class-dependent computation respectively. In Section 6 we employ these quantities to query the cascade database by time and cost; moreover, we show how multiple queries can be satisfied such that their overall cost is minimized.

4 Training Phase

Given cascade stages $\mathcal{S}$ (see Section 3.1) trained on the training set, we predict each class $l \in \hat{\mathcal{L}}$ using each cascade stage $S \in \mathcal{S}$ producing a confidence vector x^S for each of the N validation images $\boldsymbol{x}^S \leftarrow [S(I_1, \hat{\mathcal{L}}), S(I_2, \hat{\mathcal{L}}), \ldots, S(I_N, \hat{\mathcal{L}})]$ where the confidence matrix $\boldsymbol{x}^S$ has $|\hat{\mathcal{L}}|$ rows and N columns. We illustrate our approach on the scene recognition task, where each image belongs to exactly one class and the groundtruth $\boldsymbol{g}$ is a matrix of the same shape as $\boldsymbol{x}^S$ with values $\forall i \in \{1 \ldots N\}, \boldsymbol{g}_{l,i} \in \{-1, 1\}$. For each cascade stage S the threshold selection (Section 4.1) produces a set of thresholds $\mathcal{T}_l^S$ that are representative of that cascade stage's performance. The cascade selection (Section 4.2) produces $\mathcal{C}_l \subseteq \mathcal{P}(\mathcal{S})$ where $\mathcal{P}(\mathcal{S})$ is the powerset of the set of cascade stages $\mathcal{S}$. Finally the cascade simulation (Section 4.3) *simulates* each cascade $C \in \mathcal{C}_l$ using the thresholds selected $\mathcal{T}_l^S$ where $S \in C$. The simulation is an efficient method of evaluating the performance of the cascades and produces operating points (i.e., confusion matrices, times, and costs) that are then stored in the cascade database. This training phase operates on each class l independently and, for notational convenience, we let $\boldsymbol{g} = \boldsymbol{g}_l$, $\boldsymbol{x}^S = \boldsymbol{x}_l^S$, $\mathcal{T}^S = \mathcal{T}_l^S$, and $\mathcal{C} = \mathcal{C}_l$.

4.1 Threshold Selection

Given a trained cascade stage S and a validation set, our task is to find a set of thresholds $\mathcal{T}^S$ that compactly represents its operating points. The threshold selection occurs independently for each class and cascade stage S. We wish to minimize the number of thresholds $|\mathcal{T}^S|$ to reduce the cascade simulation complexity (see Section 4.3), where a confidence value $\boldsymbol{x}_i^S$ is positive if $\boldsymbol{x}_i^S \geq t$ where $t \in \mathcal{T}^S$. The initial set is $\mathcal{T}_u^S = \mathcal{X}^S \cup \{\infty\}$, where $\mathcal{X}^S$ is the set of confidence values of $\boldsymbol{x}_S$. Including infinity ensures that at least one point has no false positives. This set is *sufficient* as any other threshold produces a redundant partition; however, it is not *necessary* as thresholds which produce "worse" confusion matrices may be present (by Definition 3). We seek a subset of $\mathcal{T}_u^S$ that is both necessary and sufficient.

Given a confidence value $\boldsymbol{x}_i^S$ and ground truth label $\boldsymbol{g}_i$ for each image i in the validation set, we sort them ascending by confidence value with positive ground truth instances listed before negative ones for the same confidence values. The resulting vectors $\bar{\boldsymbol{x}}^S$ and $\bar{\boldsymbol{g}}^S$ can be partitioned (Definition 1), representing the positive and negative predictions made at that threshold level. Observe that the secondary sorting eliminates 'overestimated' confusion matrices that can result from naïve generation of confusion matrices with the same confidence value but different ground truth polarities[16]. The method described in Section 4.1.1 will implicitly remove 'underestimated' confusion matrices. As this process is independent of the stage S, we let $\bar{\boldsymbol{g}} = \bar{\boldsymbol{g}}^S$, $\bar{\boldsymbol{x}} = \bar{\boldsymbol{x}}^S$, and $\mathcal{X} = \mathcal{X}^S$. When operating on the vector $\bar{\boldsymbol{g}}$, $\bar{\boldsymbol{g}}_i$ represents the ground truth polarity at position i, with $\bar{\boldsymbol{x}}_i$ as its associated confidence value and $\bar{\boldsymbol{g}}_{i-1}$ as its neighbor in the descending direction.

4.1.1 Exact Thresholds

Our goal is to find the minimum number of thresholds $\mathcal{T}_e$ required to exactly represent the performance characteristics of the cascade stage. We show that using the selected thresholds for each stage there is no degradation in the overall cascade performance. A binary confusion matrix captures the cascade stage's performance at a specific threshold and we define a partial order over confusion matrices.

Definition 1. *$M_{t,\bar{\boldsymbol{x}},\bar{\boldsymbol{g}}}$ is the confusion matrix computed at threshold t (i.e., all $\geq t$ are positive), sequence of confidence values $\bar{\boldsymbol{x}}$, and ground-truth $\bar{\boldsymbol{g}}$.* [1]

Definition 2. *A weak partial order over the set of binary confusion matrices*

$$M_t \leq M_{t'} \equiv (TP_t \leq TP_{t'}) \wedge (TN_t \leq TN_{t'})$$

where TP_t and TN_t refer respectively to the number of true positives and true negatives from confusion matrix M_t. This relation describes 'better' performance: a confusion matrix $M_{t'}$ is better or the same as M_t if $M_{t'}$ makes no fewer correct predictions and no more errors. Note that the total number of positive and negative ground truth values is constant (i.e., $TP + FN = P$ and $TN + FP = N$) as they are created by partitioning a fixed sequence $\bar{\boldsymbol{x}}$.

Definition 3. *A strict partial order over the set of binary confusion matrices is*

$$M_t < M_{t'} \equiv (M_t \leq M_{t'}) \wedge (M_t \neq M_{t'})$$

We seek a subset of the thresholds such that: no threshold outside of the set is $>$ (sufficient condition) and no threshold in the set is $<$ another in the set (necessary condition). We now specify a predicate $Keep$ to select the thresholds $\mathcal{T}_e = \{\bar{\boldsymbol{x}}_i \in \mathcal{X} : Keep(\bar{\boldsymbol{x}}_i)\}$ and show that it produces a necessary and sufficient subset.

Definition 4. *$Keep(\bar{\boldsymbol{x}}_i)$ is true when $\bar{\boldsymbol{g}}_{i-1}$ is not positive and $\bar{\boldsymbol{g}}_i$ is not negative, where possible values are positive (i.e., $IsPos(\bar{\boldsymbol{g}}_i)$), negative (i.e., $IsNeg(\bar{\boldsymbol{g}}_i)$), or nil (i.e., $IsNil(\bar{\boldsymbol{g}}_i)$). Nil values occur past either end of the sequence.*

$$Keep(\bar{\boldsymbol{x}}_i) \equiv \neg IsPos(\bar{\boldsymbol{g}}_{i-1}) \wedge \neg IsNeg(\bar{\boldsymbol{g}}_i)$$

We now show how this simple rule produces a subset of points that meet our conditions.

Theorem 1. *For each threshold we either 1.) Keep it and it is not $<$ any other or 2.) Not keep it and there exists a threshold that is $>$ than it that we do keep*

$$\forall \bar{\boldsymbol{x}}_i \in \mathcal{X}, Keep(\bar{\boldsymbol{x}}_i) \Rightarrow \forall \bar{\boldsymbol{x}}_j \in \mathcal{X}, \neg(M_{\bar{\boldsymbol{x}}_i} < M_{\bar{\boldsymbol{x}}_j})$$
$$\forall \bar{\boldsymbol{x}}_i \in \mathcal{X}, \neg Keep(\bar{\boldsymbol{x}}_i) \Rightarrow \exists \bar{\boldsymbol{x}}_k \in \mathcal{X}, Keep(\bar{\boldsymbol{x}}_k) \wedge M_{\bar{\boldsymbol{x}}_i} < M_{\bar{\boldsymbol{x}}_k}$$

Proof. **See Supplementary Material** □

[1] $M_t \equiv M_{t,\bar{\boldsymbol{x}},\bar{\boldsymbol{g}}}$ when $\bar{\boldsymbol{x}}$ and $\bar{\boldsymbol{g}}$ are clear from the context.

In a rejection chain cascade, previous cascade stages can reject arbitrary inputs, resulting in a reduced set of confidence values $\tilde{\mathcal{X}}$ being considered which remains sorted and $\tilde{\mathcal{X}} \subseteq \mathcal{X}$. Theorem 1 shows that we can produce a necessary and sufficient set of thresholds. We now extend this to show that they are sufficient for the cascade overall.

Theorem 2. *Let* $K = \{\bar{\boldsymbol{x}}_k \in \mathcal{X} : Keep(\bar{\boldsymbol{x}}_k)\}$ *and* $\tilde{K} = \{\tilde{\boldsymbol{x}}_{\tilde{k}} \in \tilde{\mathcal{X}} : Keep(\tilde{\boldsymbol{x}}_{\tilde{k}})\}$. *For each threshold in* $\tilde{K}$ *there exists a threshold in* K *such that they have the same confusion matrices when applied to* $\tilde{\boldsymbol{x}}$ *and* $\tilde{\boldsymbol{g}}$ *(corresponding to* $\tilde{\mathcal{X}}$*).*

$$\forall \tilde{\boldsymbol{x}}_{\tilde{k}} \in \tilde{K} \exists \bar{\boldsymbol{x}}_k \in K : M_{\bar{\boldsymbol{x}}_k, \tilde{\boldsymbol{x}}, \tilde{\boldsymbol{g}}} = M_{\tilde{\boldsymbol{x}}_{\tilde{k}}, \tilde{\boldsymbol{x}}, \tilde{\boldsymbol{g}}}$$

Proof. **See Supplementary Material** □

4.1.2 Bounded approximate thresholds

We can choose to relax our exact representation by allowing for a bounded difference between the complete set of confusion matrices and a subset that represents them. We choose to specify the bounds in terms of maximum absolute precision and recall difference z which results in $O(1)$ thresholds. We seek to find the minimum subset of confusion matrices such that for all confusion matrices M_{t_e} there exists at least one in the subset that is within the specific bounds M_{t_a}, where *Bounds* is a reflexive and symmetric relation $Bounds(t_e, t_a)$. *Bounds* can be any relation with these properties derived from binary confusion matrices.

A set of thresholds $\mathcal{T}_a$ is a sufficient approximation if for all of the initial thresholds $\mathcal{T}_e$ (from Section 4.1.1) there is a value in $\mathcal{T}_a$ where *Bounds* is true.

$$SufApx(\mathcal{T}_a) \equiv \forall t_e \in \mathcal{T}_e \exists t_a \in \mathcal{T}_a : Bounds(t_e, t_a)$$

A set of thresholds $\mathcal{T}_a$ is a necessary approximation if there is no subset of $\mathcal{T}_e$ that is smaller and sufficient.

$$NecApx(\mathcal{T}_a) \equiv \nexists \mathcal{T}_{a'} \subseteq \mathcal{T}_e : SufApx(\mathcal{T}_{a'}) \wedge |\mathcal{T}_{a'}| < |\mathcal{T}_a|$$

We can represent this problem as an undirected graph, where nodes are confusion matrices and an edge is placed between all nodes where *Bounds* is true. Our goal is then to find the minimal subset of nodes such that each node in the original graph is connected to some node in the subset. This is the well known dominating set problem which is NP-hard. We use an efficient greedy algorithm to generate the dominating set; moreover, it produces comparable dominating sets to the exact algorithm in practice. The greedy algorithm produces a sufficient but possibly larger than necessary $\mathcal{T}_a$.

4.2 Cascade Selection

Given a set of cascade stages $\mathcal{S}$ for a class i we wish to generate a set of cascades $\mathcal{C}$. However, evaluating all combinations of cascade stages is intractable if the number of available cascade stages is large. A cascade C is defined as $C \subseteq \mathcal{S}$ and

is unordered as the final cascade decision is not affected by the stage ordering. We seek to identify a set $\mathcal{C}$ that provides comparable performance to $\mathcal{P}(\mathcal{S})$ which is the powerset of $\mathcal{S}$ and $\mathcal{C} \subseteq \mathcal{P}(\mathcal{S})$. A simple approach is to limit the length of the cascades as $\mathcal{C}_\ell = \mathcal{P}_{\ell+1}(\mathcal{S})$ resulting in $\forall C \in \mathcal{C}_\ell, |C| \leq \ell$. However, there are $\sum_{l \in 1 \ldots \ell} \binom{|\mathcal{S}|}{l}$ such cascades, which becomes prohibitive when longer cascades or more cascade stages are required.

An important piece of information we have not yet considered is the information gained by adding another cascade stage to a given cascade. This concept was explored recently in [6] where the value added by additional classification is computed. While their method computed this value during classification, we seek to identify a similar quantity beforehand. When comparing the confidence vectors for two stages, if the confidence values are correlated they will tend to produce little gain when combined in a cascade. This is because the stages will reject a similar set of images for any given operating point, making the additional stages ineffective. However, a randomly performing classifier would produce uncorrelated predictions, but would result in the cascade having no additional discriminative power. We seek a method that satisfies two properties (1) the confidence values between cascade stages are uncorrelated and (2) cascade stages are significantly better than random.

We use Spearman's rank correlation coefficient $\rho = \frac{6 \sum d_i^2}{N(N^2-1)}$ (where d_i is the difference of ranks for image i for the N validation images) which compares the correlation of the rank indices. This measure is robust to any monotonic transformation of the confidence values and naturally fits our cascade stage model without additional assumptions. The ranks for tied values are replaced with the the mean rank for that value. The cascades $\mathcal{C}$ are selected starting from each $S \in \mathcal{S}$ and greedily adding stages that minimize $|\rho|$ among all cascade elements. An integer valued parameter α specifies how many different stages to select for each level, providing a natural trade-off between cascades processed and time.

While the previous method ensures that a cascade's stages are uncorrelated, it doesn't prevent poorly performing cascade stages from being incorporated. We cannot directly compute the accuracy of a stage as we only have its confidence values and associated ground truth; however, better confidence values will be positively correlated with the groundtruth $\boldsymbol{g}_i$ where $\boldsymbol{g}_i \in \{-1, 1\}$. As above, we compute ρ between the confidence values and the ground truth. An integer valued parameter β specifies how many cascade stages to consider in the previous approach, where $\mathcal{S}_\beta \subseteq \mathcal{S}$ and $|\mathcal{S}_\beta| = \beta$. This represents the trade-off between robustness to poor performing cascade stages and accuracy compared to $\mathcal{P}(\mathcal{S})$.

4.3 Cascade Simulation

Given a set of cascades $\mathcal{C}$ (see Section 4.2) for a class with an associated set of thresholds $\mathcal{T}^S$ (see Section 4.1) where $S \in C$ and $C \in \mathcal{C}$, the goal is to efficiently compute a *cascade database* that contains the union of all cascades in $\mathcal{C}$ that can be formed over all of their thresholds. This process is a simulation as we are not computing the cascade performance directly; rather, we find the confi-

dence values for all stages independently and exactly compute their combined performance. The output for each set of thresholds for a cascade includes a binary confusion matrix, stage names, stage thresholds, and % of inputs computed by each stage (used to compute time and cost). In order to ensure sub-second query processing time, we perform the majority of the computation offline in cascade simulation; however, unless care is taken the task can quickly become intractable. We avoid this by reducing the number of thresholds maintained for each cascade stage (Section 4.1) and by efficiently simulating the cascades.

The naïve approach would apply the validation set to every possible chain and its corresponding set of thresholds. There are a combinatorial number of operations in the length of the cascade if every possible ordering of a single cascade (e.g., $A \rightarrow B \rightarrow C$, $C \rightarrow B \rightarrow A$) is considered; however, the order of the cascade stages does not effect the resulting binary confusion matrix, although it does affect the running time. The running time for a stage is completely determined by its own execution time and the percent of inputs that reach it[17]. We order the stages by running time (i.e., $\tau_{S,f} + \tau_{S,c_l}$ for class l), and, in practice, we have found this to produce a near optimal ordering. By observing that the rejection decisions for the cascades are highly redundant, we can avoid recomputing all subsets of stages. For example, for a cascade $A \rightarrow B \rightarrow C$, we also compute A and $A \rightarrow B$ at the same time. There may be cascades with similar prefixes (e.g., $A \rightarrow B \rightarrow C$, $A \rightarrow B \rightarrow D$) that perform redundant computation. We can extend our solution by generating trees rooted at their initial stage, where stages with the same parent are all computed simultaneously.[2]

The computational complexity of this method is the sum of the complexity for each cascade from a leaf to a root node. The complexity of each path is bounded by $O(N \prod_{S \in C} |\mathcal{T}^S|)$ where N is the number of validation inputs (e.g., images for image classification). However, this worst case performance doesn't occur in practice as the number of validation inputs N reduces from stage to stage, exactly as they do when passing through a rejection chain cascade. For the worst case to occur, it would require each stage to not reject any inputs; however, in Section 4.1 we select the thresholds in a way that precludes this. In Section 4.1.2 we showed how $|\mathcal{T}^S|$ can be $O(1)$ with bounded representation error. With this constant number of thresholds t, the cost of each path is bounded by $O(Nt^k)$ where k is the cascade length. As before, this does not take into account the number of points rejected by previous stages; we have found cascade lengths of $\ell = 5$ to be tractable on a single core machine (see Section 7).

5 Human Classifiers

As the quality of arbitrary classification tasks is often lower than necessary for practical use, human input is commonly used to augment algorithmic approaches. One difficulty is that confidence values are needed to provide control over the resulting classifier's operating characteristics. A potential solution to

[2] Psuedo-code for the simulation algorithm is available in the supplementary material.

this problem is to have the turkers provide an explicit confidence; however, this would be more effort than the task itself requires. By observing that human errors occur as a function of response time $P(Time)$ and that some users are significantly worse than others $P(User)$, we are able to estimate the probability that a turker is correct (see Fig. 3) which is our 'confidence'. Here we assume that $Time$ and $User$ are independent $P(Time, User) = P(Time)P(User)$.

Turkers were shown a picture from the SUN397 dataset and asked if it is of an indoor or outdoor scene. This was performed on the 39.7K images in the first train/test split of the 397 classes. Each turker was paid 5 cents per 100 images annotated and the mean and median of their response times was 3.3 and 2.4 sec. respectively. The SUN397 dataset has annotations for the 397 classes in a three level hierarchy with the top level consisting of three classes "indoor", "outdoor, man-made", and "outdoor, natural". We combine the two outdoor classes to produce the groundtruth labels which resulted in an accuracy of 93% with no voting or filtering applied. We integrate the turker as shown in Fig. 2(c) and described in Section 3.1.

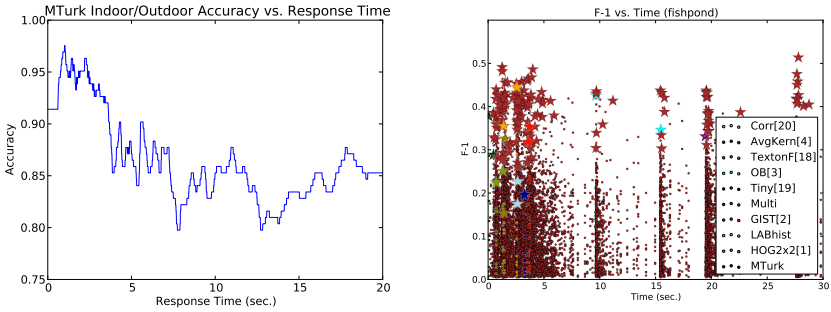
6 Query Phase

The training phase produces a cascade database which includes, for each cascade C (see Section 4.3), a binary confusion matrix, stage names, stage thresholds, and % of inputs computed $P_{C,S}$. The time is computed as $\sum_{S \in C}(\tau_{S,f} + \tau_{S,c_l})P_{C,S}$ for a class l (similarly for cost). We construct indexes for precision, recall, accuracy, F-1, time, and cost that index into the simulated cascade operating points. The database implementation is outside the scope of this work; however, for the experiments conducted we used a binary search-based algorithm that produced a median response time for a single constraint of 1.1ms (see Section 7).

Given a set of constraint satisfying cascades for several classes, we only need to select a single cascade for each class to produce valid results. The primary method of saving time and cost is by exploiting reuse between classes that is independent of the classes themselves $\tau_{S,f}$ and $\lambda_{S,f}$ (e.g., feature computation, shared classifiers). The classification model used (see Section 3.1) allows for modeling these properties; however, exactly minimizing cost on novel data is not possible as we can only estimate how many inputs are rejected after each cascade stage by observing how they perform on the validation set. A straightforward approach to this problem is for each class to independently select the cascade with minimum cost. While this does not explicitly enforce reuse, in practice a significant amount of computation is shared between the classes.

7 Experiments

We show experimental results on the SUN397[1] scene recognition dataset which consists of 397 classes and 39,700 images split into training and testing sets (19,850 each) (using partition #1). Fig. 4(a) summarizes the features used along with their recognition rate for multi-class classification to enable comparison



(a) (b)

Fig. 3. (a) Relationship between a turker's response time and their accuracy for that response. This suggests that a quick response means the question is easier and even with additional time harder questions are likely to be answered wrong. (b) Scatter plot of F-1 values vs time for a single class. The best values for each unique cascade are shown as stars. 'Multi' are cascades longer than one stage. This figure is best viewed in color.

with other methods. We use linear and histogram intersection kernels (HIK) for all of the features. LABhist uses the CIE L*a*b* colorspace with 4 bins for L and 11 bins for a and b. TextonForest uses the method proposed in [18] (trained on the MSRC dataset) to compute spatial pyramids on the maximum label mask. Both texton and integral tree types are used with one and three trees respectively. As proposed in [19], the TinyImage feature is a reasonable baseline for scene recognition tasks. The image is resized to 32x32 in the CIE L*a*b* colorspace. The ObjectBank (OB)[3] feature is a powerful method that pools object detector predictions to produce a highly discriminative feature. Spatial pyramids (used by HOG2x2, LABhist, and Texton) are of scales 1x1, 2x2, 4x4, and 8x8. We find using a spatial pyramid significantly improves performance when used with HIK at marginal additional computational cost.

We use 20% of the training set as a validation set to learn the cascade database with the classifiers trained on the other 80%. As we are evaluating binary classification performance, we use the mean area under the curve over all classes (mean AUC). To gain further insight we compare the number of classes with AUC better, same, or worse for each result compared to [1] (rounded to two decimal places before comparison, as [1] did). The result from [1] was trained on the entire training set Fig. 4(b)(#0); however, our proposed method, with classifiers trained on 20% less data, produces a significant gain using similar features, classifiers, and kernels Fig. 4(b)(#7). It is clear that this gain is from the cascade design as the mean AUC using just the features Fig. 4(b)(#6) is significantly less. Moreover, we are able to calculate an upper bound for our method by identifying the optimal cascades and thresholds on the test set Fig. 4(b)(#4-5); the proposed approach has near ideal performance.

Name	Dim.	Rec. Rate Lin.	HIK
$\sum_F$Kernels[4]	-	22.5	29.0
OB[3]	44604	18.7	25.1
HOG2x2[1]	6321	17.2	26.5
GIST[2]	960	10.5	12.7
LABhist	41140	3.6	10.4
Texton[18]	1785	3.1	6.5
AutoCorr.[20]	256	4.8	5.9
TinyImage[19]	3072	2.1	5.0

(a)

#	Method	Mean AUC	# AUC [1] >	=	<	Sim. (sec.)	Mean # $\|\mathcal{T}\|$	$\|\mathcal{C}\|$	$\|DB\|$
0	SUN397[1]	0.9573	0	397	0	-	-	-	-
1	GIST[2]	0.8426	1	0	396	-	-	-	-
2	OB[3]	0.9227	15	26	356	-	-	-	-
3	$\sum_F$Kernels[4]	0.9343	23	28	346	-	-	-	-
4	U.B. $\ell = 1$	0.9540	91	117	189	0.021	40	16	639
5	U.B. $\ell = 2$	0.9685	216	138	43	2.148	40	136	117K
6	$\mathcal{T}_e$,1,D	0.9492	59	94	244	0.023	41	16	651
7	$\mathcal{T}_e$,2,D	**0.9625**	**157**	130	110	2.439	41	136	123K
8	$\mathcal{T}_e$,2,S_2	0.9559	105	112	180	0.157	41	19	8135
9	$\mathcal{T}_{a,.05}$,1,D	0.9381	25	40	332	0.010	13	16	201
10	$\mathcal{T}_{a,.05}$,2,D	0.9491	64	78	255	0.235	13	136	12K
11	$\mathcal{T}_{a,.05}$,2,S_2	0.9423	34	58	305	0.024	13	18	1095
12	$\mathcal{T}_{a,.05}$,3,S_2	0.9442	42	63	292	0.156	13	28	6296
13	$\mathcal{T}_e$,2,D	0.9649	175	135	87	2.629	41	153	146K
14	$\mathcal{T}_{a,.05}$,2,D	0.9504	69	87	241	0.276	13	153	14K
15	$\mathcal{T}_{a,.1}$,5,S_1	0.9250	5	19	373	0.103	8	21	1761

(b)

Fig. 4. (a) Features used along with their recognition rates for linear and HIK kernels. (b) We compare against existing methods (#0-3), the upper bound on our method (#4-5) (see text), and (#6-15) variations of our method in the form (threshold method, max cascade length ℓ, cascade selection method). The threshold selection methods (see Section 4.1) are $\mathcal{T}_e$ and $\mathcal{T}_{a,z}$ (with z as p/r relation bound). The cascade selection methods (see Section 4.2) are dense (D) and sparse (S_α) with β as half of the provided stages. The only runs that make use of mechanical turk are (#13-15). The best run without turkers is bold.

To evaluate query performance, for all classes and for constraints of length 1 to 6 (i.e., p, r, t, a, F-1, cost) we compute 100 random example queries with a cap of 100 returned cascades using the cascade database produced by Fig. 4(b)(#14). The median query times (in ascending order from 1-6) are: 1.1, 3.5, 1.0, .1, .09, and .07 all in milliseconds. The query time decreases with length as the additional constraints reduce the search space dramatically. This is fast compared to the feature/classifier computation and is only performed once per execution, not per image as in[6]. Computation for run-times shown is performed on a single core of a 2.2GHz Xeon processor.

Fig. 6 shows the relation between time and F-1 for the 'fishpond' class on Fig. 4(b)(#14). Note that this method is responsible for the 'multi' values; moreover, this shows that the proposed method (1) produces better performance than the base classifiers it is provided and (2) dramatically increases the *flexibility* as seen by the diverse operating points. Note that this figure is a 2D projection of the *cascade database* for this class, additional projections can be found in the supplementary material.

8 Conclusion

We propose a method that focuses on improving *flexibility* while achieving state-of-the-art results on the SUN397 dataset compared against standard methods

and an upper bound on the method's performance. Our method efficiently generates a cascade database that can be queried with user provided constraints. Additionally, we demonstrate that human annotators can be naturally incorporated into this model.[3]

References

1. Xiao, J., Hays, J., Ehinger, K., Oliva, A., Torralba, A.: Sun database: Large-scale scene recognition from abbey to zoo. In: CVPR. (2010) 3485–3492
2. Oliva, Aude, Torralba, Antonio: Modeling the Shape of the Scene: A Holistic Representation of the Spatial Envelope. IJCV (2001) 145–175
3. Li, L., Su, H., Xing, E., Fei-Fei, L.: Object bank: A high-level image representation for scene classification and semantic feature sparsification. NIPS (2010)
4. Gehler, P., Nowozin, S.: On feature combination for multiclass object classification. In: ICCV. (2009) 221–228
5. Angelova, A., Matthies, L., Helmick, D., Perona, P.: Fast terrain classification using variable-length representation for autonomous navigation. In: CVPR. (2007) 1–8
6. Gao, T., Koller, D.: Active classification based on value of classifier. In: NIPS. (2011)
7. Gao, T., Koller, D.: Discriminative learning of relaxed hierarchy for large-scale visual recognition. In: ICCV. (2011)
8. Brubaker, S., Wu, J., Sun, J., Mullin, M., Rehg, J.: On the design of cascades of boosted ensembles for face detection. IJCV (2008) 65–86
9. Bourdev, L., Brandt, J.: Robust Object Detection via Soft Cascade. In: CVPR. (2005) 236–243
10. Visentini, I., Micheloni, C., Foresti, G.: Tuning asymboost cascades improves face detection. In: ICIP. (2007)
11. Torralba, A., Murphy, K., Freeman, W.: Sharing features: efficient boosting procedures for multiclass object detection. In: CVPR. (2004) 762
12. Platt, J., Cristianini, N., Shawe-Taylor, J.: Large margin dags for multiclass classification. NIPS (2000) 547–553
13. Jiang, W., Chang, S., Loui, A.: Kernel sharing with joint boosting for multi-class concept detection. In: CVPR. (2007) 1–8
14. Palatucci, M., Pomerleau, D., Hinton, G., Mitchell, T.: Zero-shot learning with semantic output codes. In: NIPS. (2009) 1410–1418
15. Schwing, A., Zach, C., Zheng, Y., Pollefeys, M.: Adaptive random forest: How many "experts" to ask before making a decision? In: CVPR. (2011) 1377–1384
16. Fawcett, T.: An introduction to ROC analysis. Pattern recognition letters (2006) 861–874
17. Mccane, B., Novins, K., Albert, M., Kaelbling, P.: Optimizing Cascade Classifiers. JMLR (2008)
18. Shotton, J., Johnson, M., Cipolla, R.: Semantic texton forests for image categorization and segmentation. In: CVPR. (2008) 1–8
19. Torralba, A., Fergus, R., Freeman, W. T.: 80 Million Tiny Images: A Large Data Set for Nonparametric Object and Scene Recognition. PAMI (2008) 1958–1970
20. Huang, Jing, Kumar, S. Ravi, Mitra, Mandar, Zhu, Wei J., Zabih, Ramin: Image Indexing Using Color Correlograms. In: CVPR. (1997) 762

[3] For additional results, proofs, psuedo-code, and figures please refer to the supplementary material. Source code will be provided upon publication.