Scalable Multitask Representation Learning for Scene Classification
Supplementary Material

Maksim Lapin, Bernt Schiele
Max Planck Institute for Informatics
{mlapin,schiele}@mpi-inf.mpg.de

Matthias Hein
Saarland University
hein@cs.uni-saarland.de

This supplementary material provides further details concerning the implementation of STL-SDCA and MTL-SDCA solvers, their runtime analysis, and visualization of selected prediction results. All source code and the scripts for running experiments are available at http://github.com/mlapin/cvpr14mtl.

1. Implementation Details

In this section we discuss certain implementation details of our STL-SDCA and MTL-SDCA solvers. We begin with some notation and then proceed with technical details for each solver.

Notation: Let $\{(x_i, y_{it}) : 1 \leq i \leq n, 1 \leq t \leq T\}$ be the input/output pairs of the multitask learning problem, where $x_i \in \mathbb{R}^d$, $y_{it} \in \{\pm 1\}$, T is the number of tasks, and n is the number of training examples per task. We assume that all tasks have the same training examples even though this can be easily generalized. The standard single task learning (STL) approach learns linear predictors w_t in the original space $\mathbb{R}^d$. In contrast, the proposed multitask learning (MTL) method learns a matrix U in $\mathbb{R}^{d \times k}$, which is used to map the original features x_i into a new representation z_i via $z_i = U^\top x_i$. The linear predictors w_t are then learned in the subspace $\mathbb{R}^k$.

Let X in $\mathbb{R}^{d \times n}$ be the matrix of stacked vectors x_i, Z in $\mathbb{R}^{k \times n}$ the matrix of stacked vectors z_i, Y in $\{\pm 1\}^{n \times T}$ the matrix of labels, and W in $\mathbb{R}^{\cdot \times T}$ the matrix of stacked predictors w_t (the dimensionality of w_t will be clear from the context). We define the following kernel matrices: $K = K_X = X^\top X$, $K_Z = Z^\top Z$, and $M = K_W = W^\top W$.

As mentioned in the paper, both solvers use precomputed kernel matrices and work with dual variables α_t in $\mathbb{R}^n$. We define A in $\mathbb{R}^{n \times T}$ as the matrix of stacked dual variables for all tasks.

STL-SDCA: The STL optimization problem for a task t is defined as follows:

$$\min_{w_t \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \max\left(0, 1 - y_{it} \langle w_t, x_i \rangle\right) + \frac{\lambda}{2} \|w_t\|_2^2,$$

where $\lambda > 0$ is the regularization parameter. This yields the following dual problem, see [1].

$$\max_{\alpha_t \in \mathbb{R}^n} \frac{1}{n} \sum_{i=1}^{n} y_{it}\alpha_{it} - \frac{\lambda}{2} \left\| \frac{1}{\lambda n} \sum_{i=1}^{n} \alpha_{it} x_i \right\|_2^2 \quad \text{subject to} \quad 0 \leq y_{it}\alpha_{it} \leq 1 \quad \text{for all} \quad i = 1, \ldots, n$$

At step s, a dual variable is updated via $\alpha_{it}^{(s)} = \alpha_{it}^{(s-1)} + \Delta\alpha_{it}$, where the update $\Delta\alpha_{it}$ can be computed as [1]:

$$\Delta\alpha_{it} = y_{it} \max\left(-y_{it}\alpha_{it}^{(s-1)}, \min\left(1 - y_{it}\alpha_{it}^{(s-1)}, \frac{1 - y_{it}\langle w_t, x_i\rangle}{\frac{1}{\lambda n}\|x_i\|_2^2}\right)\right). \tag{1}$$

Note that $\|x_i\|_2^2 = K_{ii} = K_X[i,i]$ and, since

$$w_t = \frac{1}{\lambda n} \sum_{i=1}^{n} \alpha_{it} x_i = \frac{1}{\lambda n} X\alpha_t, \quad W = \frac{1}{\lambda n} XA, \tag{2}$$

one obtains

$$\langle w_t, x_i \rangle = \frac{1}{\lambda n} \sum_{j=1}^{n} \alpha_{jt} \langle x_i, x_j \rangle = \frac{1}{\lambda n} K_i^\top \alpha_t = K_i^\top \tilde{\alpha}_t,$$

where $K_i = K_X[:, i]$ is the i-th column of K_X and we apply a change of variables $\tilde{\alpha}_{it} = \frac{1}{\lambda n} \alpha_{it}$. From now on, we always use the transformed $\tilde{\alpha}_{it}$ variables and drop the $\tilde{}$ notation for convenience.

The vector $K\alpha_t$ in $\mathbb{R}^n$ can be precomputed using the initial $\alpha_t^{(0)}$ and then updated whenever $\Delta\alpha_{it} \neq 0$ as follows:

$$K\alpha_t^{(s)} = K\alpha_t^{(s-1)} + \Delta\alpha_{it} K_i. \tag{3}$$

Note that this update, as well as the $K\alpha_t$, can be computed efficiently using BLAS routines `xAXPY` and `xGEMV`. Let

$$h = \frac{1 - y_{it} K_i^\top \alpha_t}{K_{ii}} \quad \text{and} \quad C = \frac{1}{\lambda n}, \tag{4}$$

where $K_i^\top \alpha_t$ is the i-th element of the precomputed vector $K\alpha_t$. Then $\alpha_{it}^{(s)}$ can be computed directly as follows:

$$\alpha_{it}^{(s)} = \begin{cases} \max\left(0, \min\left(C, \alpha_{it}^{(s-1)} + h\right)\right) & \text{if } y_{it} = +1, \\ \max\left(-C, \min\left(0, \alpha_{it}^{(s-1)} - h\right)\right) & \text{if } y_{it} = -1. \end{cases} \tag{5}$$

Based on (5), the update (1) can be shown to be 0, *i.e.* $\Delta\alpha_{it} = 0$, in the following two cases which typically hold for most of the data points after the first few epochs. Note that if the update is zero, then computation of (3) and (4) is avoided:

$$\Delta\alpha_{it} = 0 \quad \text{if } \left(\alpha_{it}^{(s-1)} = 0 \text{ and } h \leq 0\right) \text{ or } \left(\alpha_{it}^{(s-1)} = y_{it} C \text{ and } h \geq 0\right).$$

The intuition here is that if a point is not a support vector ($\alpha_{it}^{(s-1)} = 0$) and there is no loss on this example ($h \leq 0$), then there is no incentive for the data point to become a support vector. Similarly, if there is some non-negative loss ($h \geq 0$), but the point already exerts the maximum force ($\alpha_{it}^{(s-1)} = y_{it} C$), then it will not be updated.

Since $K_{ii} > 0$ (we skip examples with $K_{ii} = 0$), the conditions $h \gtreqless 0$ can be simplified and one obtains

$$\Delta\alpha_{it} = 0 \quad \text{if } \left(\alpha_{it}^{(s-1)} = 0 \text{ and } y_{it} K_i^\top \alpha_t \geq 1\right) \text{ or } \left(\alpha_{it}^{(s-1)} = y_{it} C \text{ and } y_{it} K_i^\top \alpha_t \leq 1\right). \tag{6}$$

U-SDCA: As discussed in the paper, MTL-SDCA alternates between learning predictors w_t via STL-SDCA on Z and learning the matrix U via an algorithm that we call U-SDCA.

Let W be fixed. The problem of learning a matrix U is formulated as follows:

$$\min_{U \in \mathbb{R}^{d \times k}} \frac{1}{nT} \sum_{i=1}^{n} \sum_{t=1}^{T} \max\left(0, 1 - y_{it} \langle w_t, U^\top x_i \rangle\right) + \frac{\mu}{2} \|U\|_F^2,$$

where $\mu > 0$ is the regularization parameter and $\|\cdot\|_F$ is the Frobenius norm. The corresponding dual problem is given below.

$$\max_{A \in \mathbb{R}^{n \times T}} \frac{1}{nT} \sum_{i,t} y_{it} \alpha_{it} - \frac{\mu}{2} \left\| \frac{1}{\mu n T} \sum_{i,t} \alpha_{it} x_i w_t^\top \right\|_F^2 \quad \text{subject to} \quad 0 \leq y_{it} \alpha_{it} \leq 1 \quad \text{for all} \quad i = 1, \ldots, n, \; t = 1, \ldots, T$$

Similarly to STL-SDCA, an update $\Delta\alpha_{it}$ can be computed as follows:

$$\Delta\alpha_{it} = y_{it} \max\left(-y_{it} \alpha_{it}^{(s-1)}, \min\left(1 - y_{it} \alpha_{it}^{(s-1)}, \frac{1 - y_{it} \langle w_t, U^\top x_i \rangle}{\frac{1}{\mu n T} \|x_i\|_2^2 \|w_t\|_2^2}\right)\right),$$

Note that $\|x_i\|_2^2 = K_{ii} = K_X[i, i]$, $\|w_t\|_2^2 = M_{tt} = K_W[t, t]$ and, since

$$U = \frac{1}{\mu n T} \sum_{i,t} \alpha_{it} x_i w_t^\top = \frac{1}{\mu n T} X A W^\top, \tag{7}$$

one obtains

$$\left\langle w_t, U^\top x_i \right\rangle = \frac{1}{\mu n T} \sum_{j,s} \alpha_{js} \left\langle x_i, x_j \right\rangle \left\langle w_s, w_t \right\rangle = \frac{1}{\mu n T} K_i^\top A M_t = K_i^\top \tilde{A} M_t,$$

where $K_i = K_X[:, i]$ is the i-th column of K_X, $M_t = K_W[:, t]$ is the t-th column of K_W and we apply a change of variables $\tilde{\alpha}_{it} = \frac{1}{\mu n T} \alpha_{it}$. As before, we always use the transformed $\tilde{\alpha}_{it}$ variables and drop the $\tilde{}$ notation.

Note that the matrix A now introduces coupling between all tasks and all examples, hence every non-zero update $\Delta\alpha_{it}$ affects all scores $\left\langle w_t, U^\top x_i \right\rangle$. We experimented with one approach where the whole matrix KAM is precomputed and then updated via rank-1 updates $\Delta\alpha_{it} x_i w_t^\top$ (using BLAS routine xGER). However, this strategy seemed inferior in terms of runtime when compared to the approach we present next (most likely due to less efficient memory access pattern).

Instead of sampling both i and t at every iteration, we proceed as follows. At each epoch, we iterate over all tasks in random order (task IDs are permuted at the beginning of the epoch) and precompute KAM_t for a given task t. Then we iterate over all examples in random order and update the *vector* KAM_t in $\mathbb{R}^n$ similar to (3):

$$KA^{(s)} M_t = KA^{(s-1)} M_t + \Delta\alpha_{it} K_i M_{tt}.$$

The formula (5) for $\alpha_{it}^{(s)}$ remains unchanged, while the h and C are now computed differently:

$$h = \frac{1 - y_{it} K_i^\top A M_t}{K_{ii} M_{tt}} \quad \text{and} \quad C = \frac{1}{\mu n T},$$

where $K_i^\top A M_t$ is the i-th element of the precomputed vector KAM_t. Similarly, the condition (6) becomes

$$\Delta\alpha_{it} = 0 \quad \text{if} \; \left(\alpha_{it}^{(s-1)} = 0 \text{ and } y_{it} K_i^\top A M_t \geq 1\right) \; \text{or} \; \left(\alpha_{it}^{(s-1)} = y_{it} C \text{ and } y_{it} K_i^\top A M_t \leq 1\right).$$

MTL-SDCA: We now describe the master problem of the MTL-SDCA algorithm. Recall that the joint problem is non-convex and we use an STL solution as the initial point, *i.e.* $U^{(0)} = W_{\text{STL}} = X A_{\text{STL}}$, where A_{STL} are the dual variables computed by STL-SDCA on X (using the kernel K_X). It follows that

$$Z = {U^{(0)}}^\top X = A_{\text{STL}}^\top K_X \quad \text{and} \quad K_Z = Z^\top Z.$$

MTL-SDCA takes Y, K_X, K_Z, λ, and μ as input and outputs A and K_W. Since $U = XAW^\top$, the test scores are given as:

$$W^\top Z_{\text{tst}} = W^\top U^\top X_{\text{tst}} = K_W A^\top K_{\text{tst}},$$

where $K_{\text{tst}} = X^\top X_{\text{tst}}$. The procedure is summarized in Algorithm 1.

Algorithm 1 MTL-SDCA

```
Input: labels $Y$, Gram matrices $K_X$ and $K_Z$, parameters $\lambda$ and $\mu$, stopping criterion $\epsilon$
Let: $A, A_{\text{old}}, B, B_{\text{old}} = \mathbf{0}$  // initialize U-SDCA and STL-SDCA dual variables
loop
    Update $B$ via STL-SDCA($Y$, $K_Z$, $\lambda$)
    Let $K_W = B^\top K_Z B$  // since $W = ZB$, see (2)
    Update $A$ via U-SDCA($Y$, $K_X$, $K_W$, $\mu$)
    if RMSE $((A, B) - (A_{\text{old}}, B_{\text{old}})) < \epsilon$ then
        break
    end if
    Let $K_Z = K_X A K_W A^\top K_X$  // since $Z = U^\top X$ and $U = XAW^\top$, see (7)
    Let $A_{\text{old}} = A$, $B_{\text{old}} = B$
end loop
return $A$, $K_W$
```

Note that both the STL-SDCA and U-SDCA solvers support warm restart, hence the dual variables A and B are actually updated rather than recomputed from scratch.

2. Runtime Analysis

To estimate the overhead of the proposed MTL method relative to the standard STL approach, we performed a single run of the full pipeline for the best performing setting (SIFT+LCS+PN+L2 features, Ntrain=50 examples per class), see Table 3 in the paper. We used a 32 core 64 bit 2.7 GHz Intel CPU machine with 256 GB of RAM. Both solvers were compiled by GCC version[1] 4.4 with the `-O3` option and linked the Intel MKL library version 11.1 for the vectorized BLAS subroutines.

Table 1 reports the elapsed wall-clock time for various steps of the complete pipeline excluding the model selection step. Note that most of the time (over 9 hours) is spent in the computation of image descriptors, where the Fisher Vector encoding step is the most expensive compared to SIFT and LCS feature extraction. MTL-SDCA training takes more time than STL-SDCA, but is faster than training an encoder, where most of the time is spent in SIFT and LCS extraction and in learning a visual words vocabulary. Moreover, the MTL time also includes the STL-SDCA training time since STL provides the initial point in our approach. For further details, see the script `matlab/analysis/resProfileTime.m`.

Routine	STL	MTL	MTL/STL
Prepare image encoder (fit a GMM for the Fisher Vector)	1.4 hours		–
Compute Fisher Vector image descriptors (train and test subsets)	9.3 hours		–
Compute train and test kernels	11.1 mins		–
Solve SDCA optimization problem (pure training time)	2.2 mins	24.9 mins	11.23
+ compute training kernels + compute MTL initial point	8.0 mins	32.9 mins	4.13
+ compute descriptors for training images	6.2 hours	6.7 hours	1.07

Table 1: Runtime comparison for the standard STL and the proposed MTL methods (**Wall-clock** time).

Table 2 reports a similar breakdown of the total runtime as measured by CPU time.

Routine	STL	MTL	MTL/STL
Prepare image encoder (fit a GMM for the Fisher Vector)	7.3 hours		–
Compute Fisher Vector image descriptors (train and test subsets)	3.3 days		–
Compute train and test kernels	2.8 hours		–
Solve SDCA optimization problem (pure training time)	15.8 mins	5.8 hours	22.00
+ compute training kernels + compute MTL initial point	1.7 hours	7.5 hours	4.43
+ compute descriptors for training images	2.0 days	2.3 days	1.12

Table 2: Runtime comparison for the standard STL and the proposed MTL methods (**CPU** time).

We conclude from the presented runtime comparison that the relative overhead of the proposed multitask learning method is rather small when other steps of the pipeline are taken into account (roughly a factor of 4 when all features are precomputed and about 12% otherwise).

3. Top-5 Predictions Modulo Human Confusions

We show that the proposed MTL method not only improves the top-K accuracy for varying K, as argued in the paper, but also tends to produce more "reasonable" confusions in the following sense. Let P^5_{STL} be a set of top-5 prediction results for the baseline STL method, similarly, let P^5_{MTL} be a set of top-5 predictions of the proposed MTL approach, finally, let P_{human} be a set of all classes that AMT workers confused with the given class (*i.e.* classes of all non-zeros in the corresponding row of the confusion matrix of "good workers"). Let

$$f_{\mathrm{STL}} = \left|P^5_{\mathrm{STL}} \cap P_{\mathrm{human}}\right|, \qquad\qquad f_{\mathrm{MTL}} = \left|P^5_{\mathrm{MTL}} \cap P_{\mathrm{human}}\right|,$$

where $|A|$ is the cardinality of a set A. Figure 1 shows the distribution of $\Delta_f = f_{\mathrm{MTL}} - f_{\mathrm{STL}}$ across all 10 splits of the SUN397 dataset. Note that $\Delta_f > 0$ for more test examples than $\Delta_f < 0$, which means there is a tendency for the MTL method to produce more "human" confusions.

[1] MATLAB R2013a requires GCC 4.4, see `http://www.mathworks.com/support/compilers/R2013a/index.html?sec=glnxa64`.

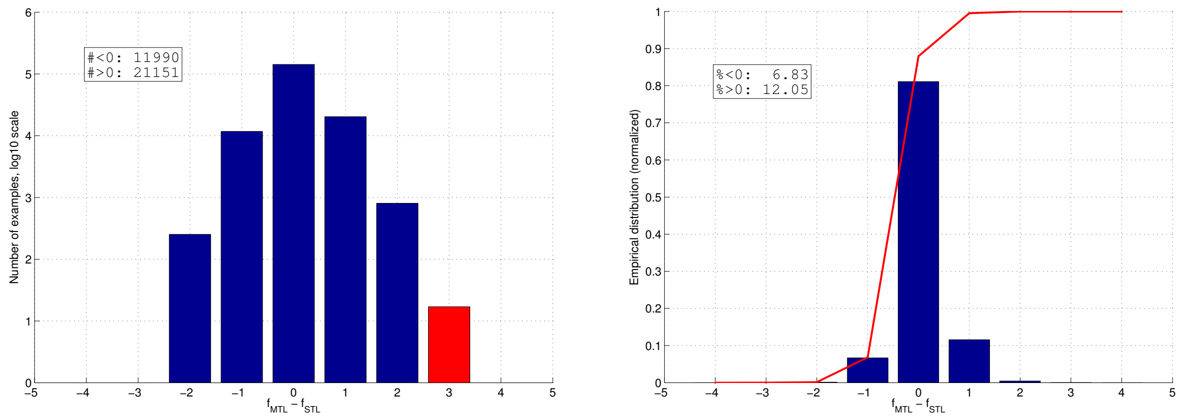

Figure 1: Comparison of STL and MTL top-5 prediction results modulo human confusions. For each method, the top-5 predictions are intersected with human predictions to count "reasonable" confusions, denoted $f_{\mathrm{STL/MTL}}$. The plots show the distribution of $f_{\mathrm{MTL}} - f_{\mathrm{STL}}$ over all 10 splits of SUN397. The highlighted (red) bar corresponds to "the best" MTL results, some of which are visualized in the following section. **Left:** counts on the $\log_{10}$ scale; **Right:** normalized distribution.

4. Selected Prediction Results on SUN397

We visualize top-5 predictions of STL and MTL methods on a few selected examples where MTL produces "more human" confusions (see Figure 1). Human performance is estimated based on the confusion matrix of "good workers" provided by Xiao *et al*. [2]. Classifiers are trained using the SIFT descriptor with the Hellinger kernel and Ntrain=20 images per class.

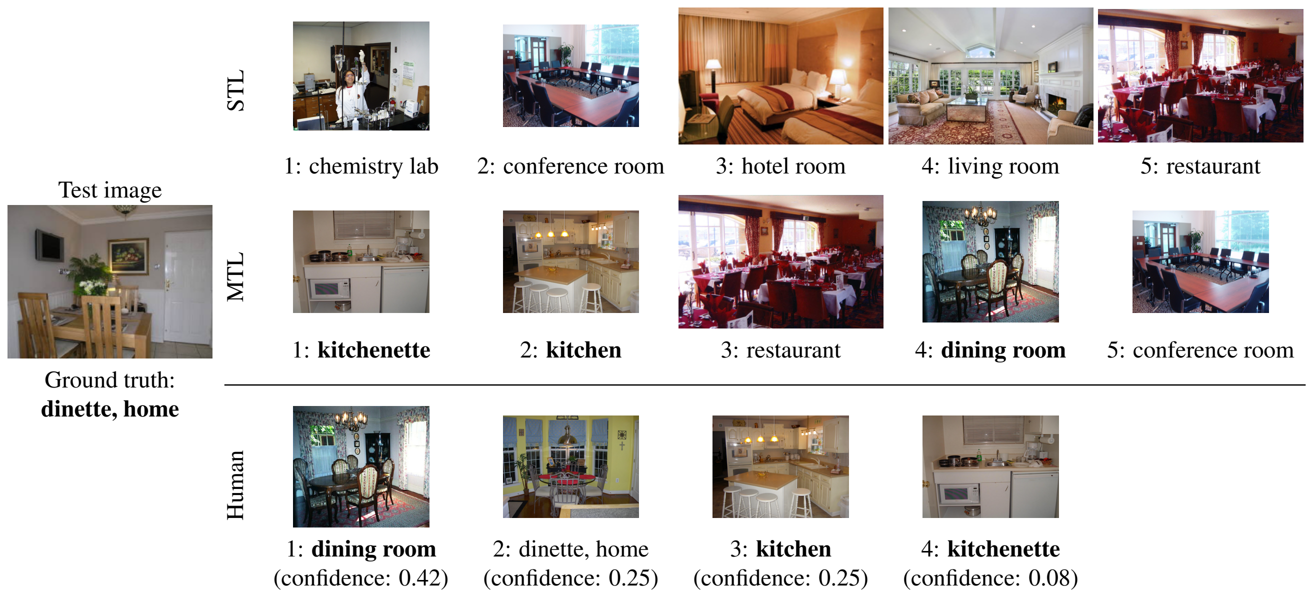

Figure 2: Multitask learning improves upon single task learning on the given test image. Top five predictions along with thumbnail images are shown for both approaches. **Top row:** single task learning results. **Middle row:** multitask learning results. **Bottom row:** confidence scores from the confusion matrix of "good workers".

References

[1] S. Shalev-Shwartz and T. Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *JMLR*, 14:567–599, 2013. 1

[2] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba. SUN database: Large-scale scene recognition from abbey to zoo. In *CVPR*, 2010. 5

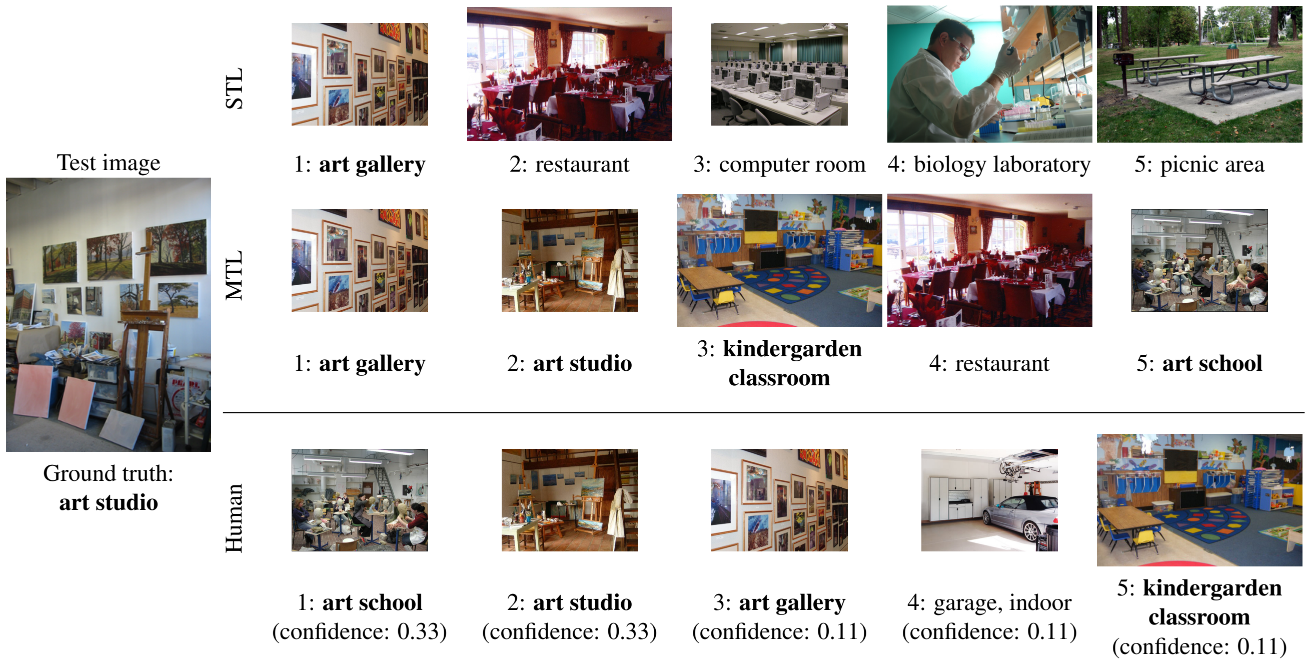
Test image
Ground truth: art studio
STL
1: art gallery
2: restaurant
3: computer room
4: biology laboratory
5: picnic area
MTL
1: art gallery
2: art studio
3: kindergarden classroom
4: restaurant
5: art school
Human
1: art school (confidence: 0.33)
2: art studio (confidence: 0.33)
3: art gallery (confidence: 0.11)
4: garage, indoor (confidence: 0.11)
5: kindergarden classroom (confidence: 0.11)

Figure 3: Multitask learning improves upon single task learning on the given test image. Top five predictions along with thumbnail images are shown for both approaches. **Top row:** single task learning results. **Middle row:** multitask learning results. **Bottom row:** confidence scores from the confusion matrix of "good workers".

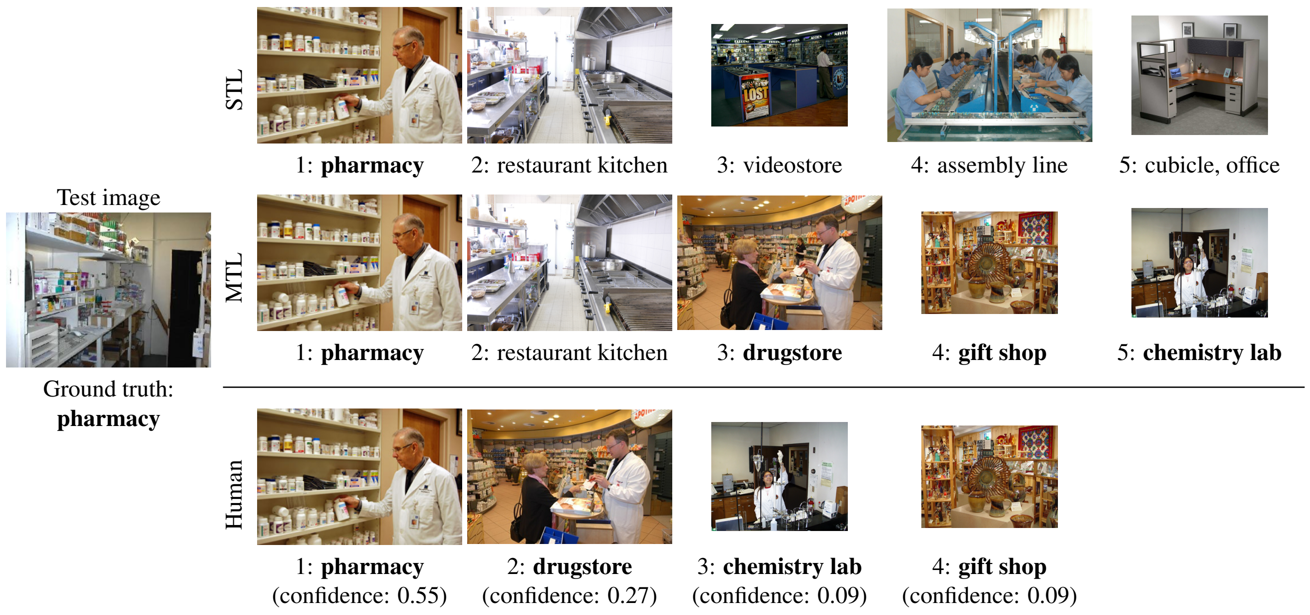

Figure 4: Multitask learning improves upon single task learning on the given test image. Top five predictions along with thumbnail images are shown for both approaches. **Top row:** single task learning results. **Middle row:** multitask learning results. **Bottom row:** confidence scores from the confusion matrix of "good workers".

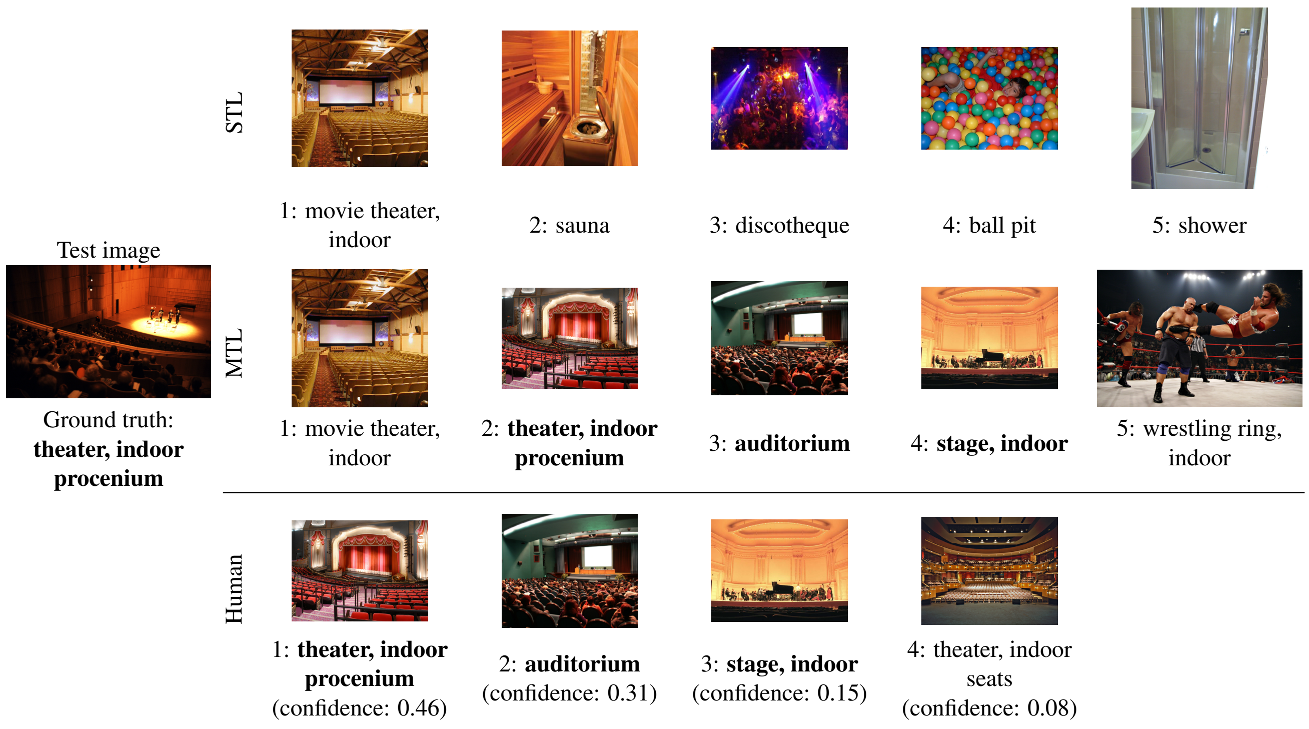
STL
MTL
Human
Test image
Ground truth:
theater, indoor procenium
1: movie theater, indoor
2: sauna
3: discotheque
4: ball pit
5: shower
1: movie theater, indoor
2: **theater, indoor procenium**
3: **auditorium**
4: **stage, indoor**
5: wrestling ring, indoor
1: **theater, indoor procenium** (confidence: 0.46)
2: **auditorium** (confidence: 0.31)
3: **stage, indoor** (confidence: 0.15)
4: theater, indoor seats (confidence: 0.08)

Figure 5: Multitask learning improves upon single task learning on the given test image. Top five predictions along with thumbnail images are shown for both approaches. **Top row:** single task learning results. **Middle row:** multitask learning results. **Bottom row:** confidence scores from the confusion matrix of "good workers".

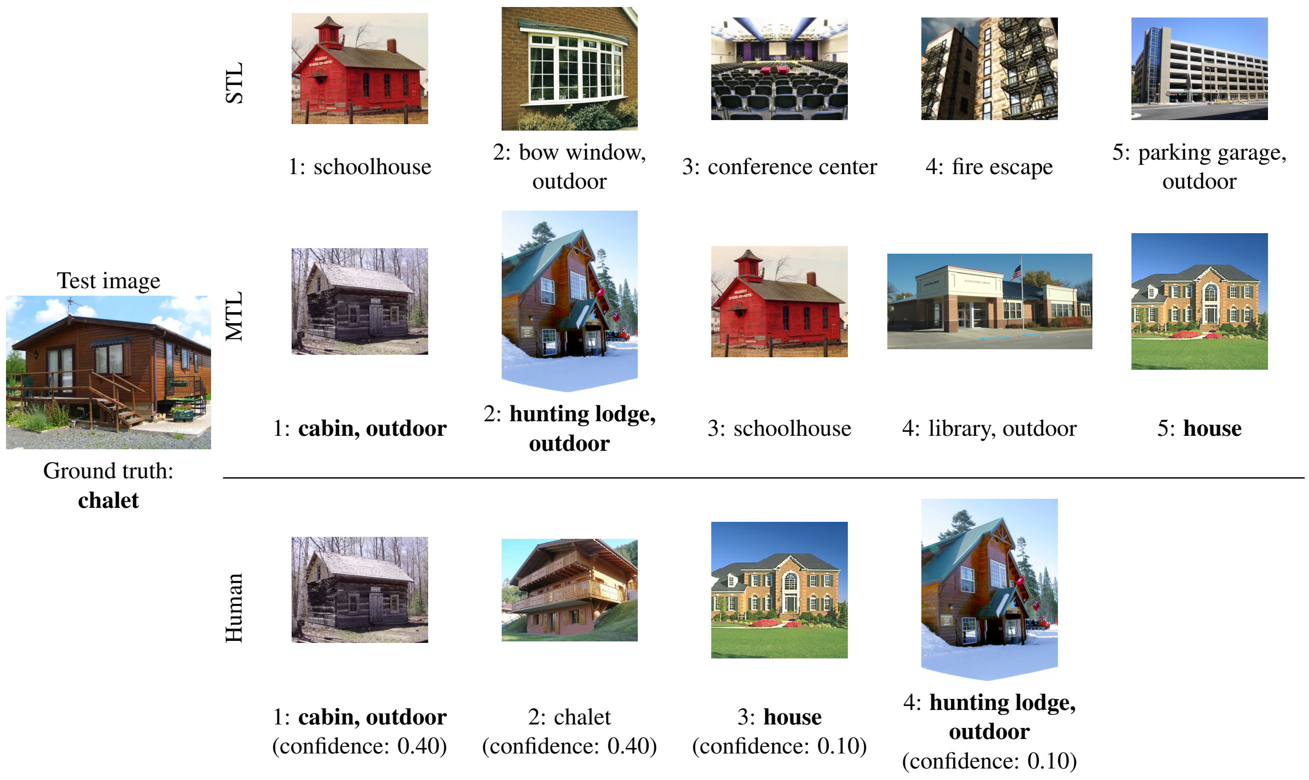
STL
MTL
Human
Test image
Ground truth:
chalet
1: schoolhouse
2: bow window, outdoor
3: conference center
4: fire escape
5: parking garage, outdoor
1: **cabin, outdoor**
2: **hunting lodge, outdoor**
3: schoolhouse
4: library, outdoor
5: **house**
1: **cabin, outdoor** (confidence: 0.40)
2: chalet (confidence: 0.40)
3: **house** (confidence: 0.10)
4: **hunting lodge, outdoor** (confidence: 0.10)

Figure 6: Multitask learning improves upon single task learning on the given test image. Top five predictions along with thumbnail images are shown for both approaches. **Top row:** single task learning results. **Middle row:** multitask learning results. **Bottom row:** confidence scores from the confusion matrix of "good workers".

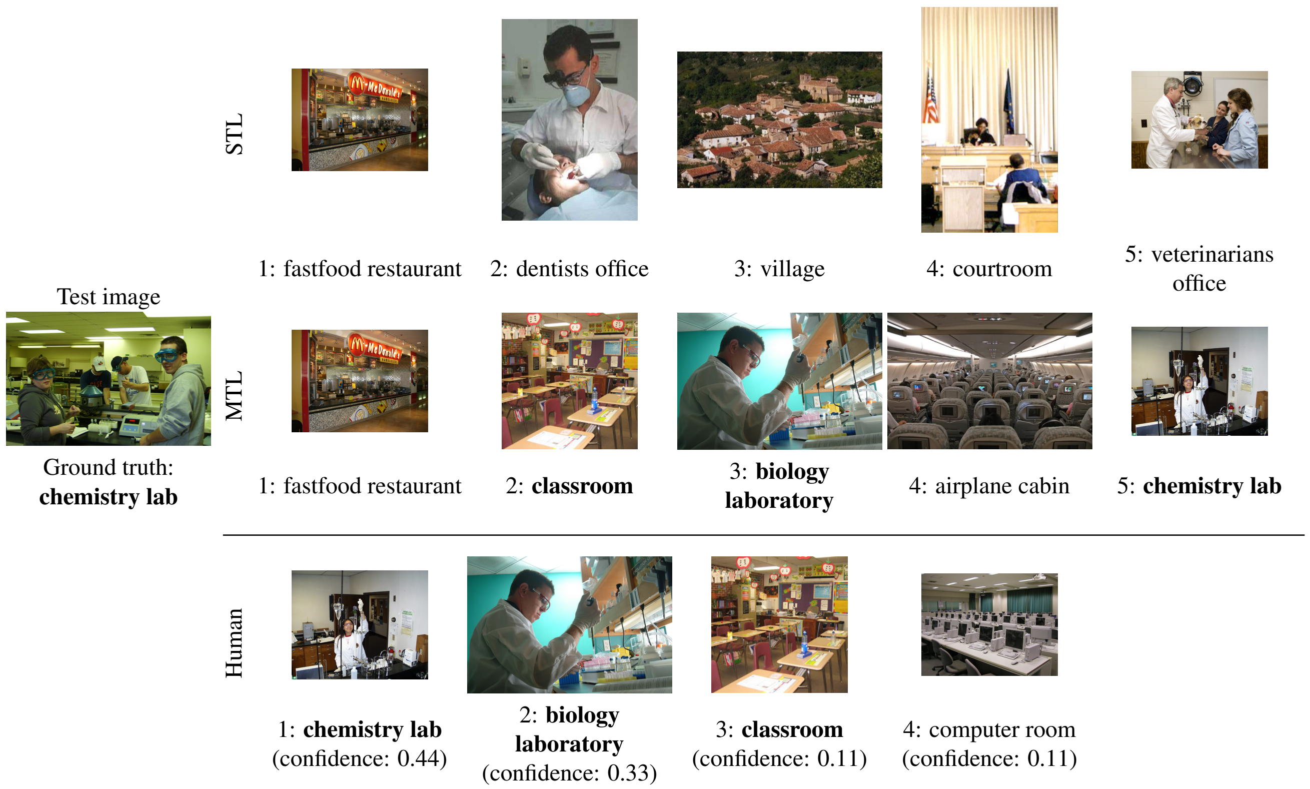
STL
1: fastfood restaurant
2: dentists office
3: village
4: courtroom
5: veterinarians office
Test image
Ground truth: **chemistry lab**
MTL
1: fastfood restaurant
2: **classroom**
3: **biology laboratory**
4: airplane cabin
5: **chemistry lab**
Human
1: **chemistry lab** (confidence: 0.44)
2: **biology laboratory** (confidence: 0.33)
3: **classroom** (confidence: 0.11)
4: computer room (confidence: 0.11)

Figure 7: Multitask learning improves upon single task learning on the given test image. Top five predictions along with thumbnail images are shown for both approaches. **Top row:** single task learning results. **Middle row:** multitask learning results. **Bottom row:** confidence scores from the confusion matrix of "good workers".

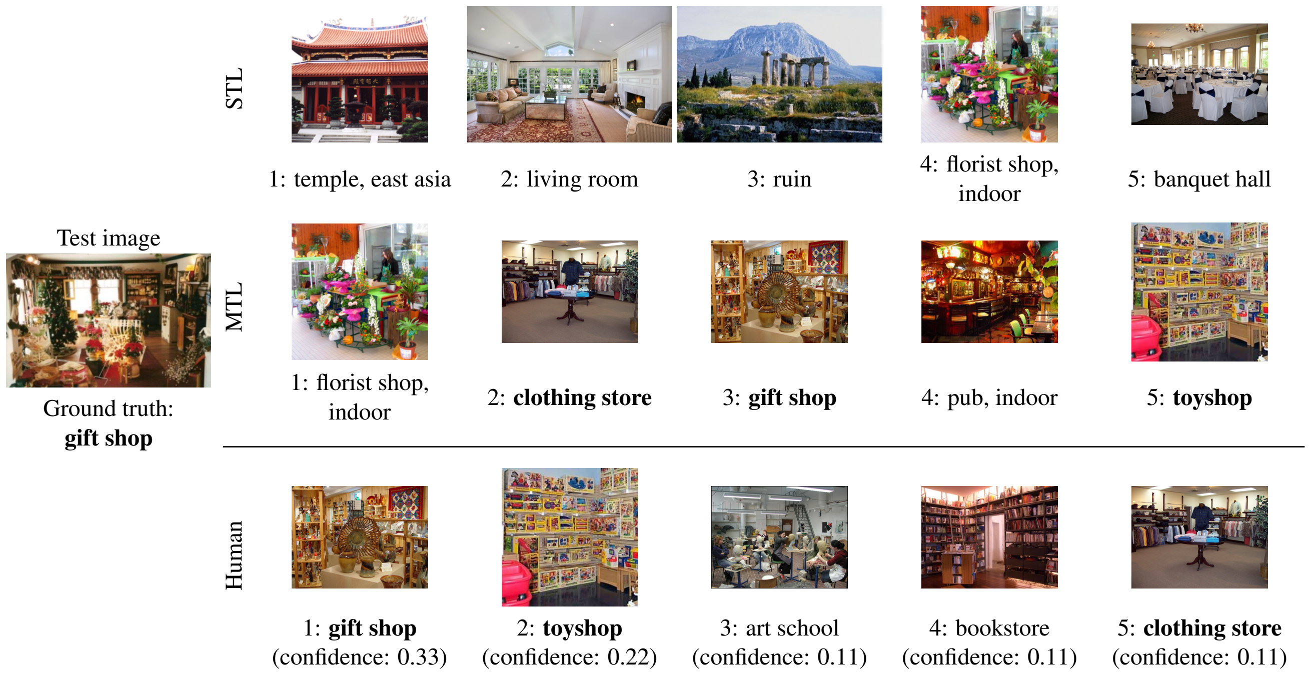
STL
1: temple, east asia
2: living room
3: ruin
4: florist shop, indoor
5: banquet hall
Test image
Ground truth: **gift shop**
MTL
1: florist shop, indoor
2: **clothing store**
3: **gift shop**
4: pub, indoor
5: **toyshop**
Human
1: **gift shop** (confidence: 0.33)
2: **toyshop** (confidence: 0.22)
3: art school (confidence: 0.11)
4: bookstore (confidence: 0.11)
5: **clothing store** (confidence: 0.11)

Figure 8: Multitask learning improves upon single task learning on the given test image. Top five predictions along with thumbnail images are shown for both approaches. **Top row:** single task learning results. **Middle row:** multitask learning results. **Bottom row:** confidence scores from the confusion matrix of "good workers".