Unifying Heterogeneous Classifiers with Distillation

Jayakorn Vongkulbhisal[1], Phongtharin Vinayavekhin[1], Marco Visentini-Scarzanella[2]
[1]IBM Research, Tokyo, Japan
[2]Amazon, Tokyo, Japan
jayakornv@ibm.com, pvmilk@jp.ibm.com, marcovs@amazon.com

Abstract

In this paper, we study the problem of unifying knowledge from a set of classifiers with different architectures and target classes into a single classifier, given only a generic set of unlabelled data. We call this problem Unifying Heterogeneous Classifiers (UHC). This problem is motivated by scenarios where data is collected from multiple sources, but the sources cannot share their data, e.g.*, due to privacy concerns, and only privately trained models can be shared. In addition, each source may not be able to gather data to train all classes due to data availability at each source, and may not be able to train the same classification model due to different computational resources. To tackle this problem, we propose a generalisation of knowledge distillation to merge HCs. We derive a probabilistic relation between the outputs of HCs and the probability over all classes. Based on this relation, we propose two classes of methods based on cross-entropy minimisation and matrix factorisation, which allow us to estimate soft labels over all classes from unlabelled samples and use them in lieu of ground truth labels to train a unified classifier. Our extensive experiments on ImageNet, LSUN, and Places365 datasets show that our approaches significantly outperform a naive extension of distillation and can achieve almost the same accuracy as classifiers that are trained in a centralised, supervised manner.*

1. Introduction

The success of machine learning in image classification tasks has been largely enabled by the availability of big datasets, such as ImageNet [32] and MS-COCO [25]. As the technology becomes more pervasive, data collection is transitioning towards more distributed settings where the data is sourced from multiple entities and then combined to train a classifier in a central node (Fig. 1a). However, in many cases, transfer of data between entities is not possible due to privacy concerns (*e.g.*, private photo albums or medical data) or bandwidth restrictions (*e.g.*, very large datasets), hampering the unification of knowledge from different sources. This has led to multiple works that propose to learn classifiers without directly sharing data, *e.g.*, distributed optimisation [4], consensus-based training [12], and federated learning [20]. However, these approaches generally require models trained by each entity to be the same both in terms of architecture and target classes.

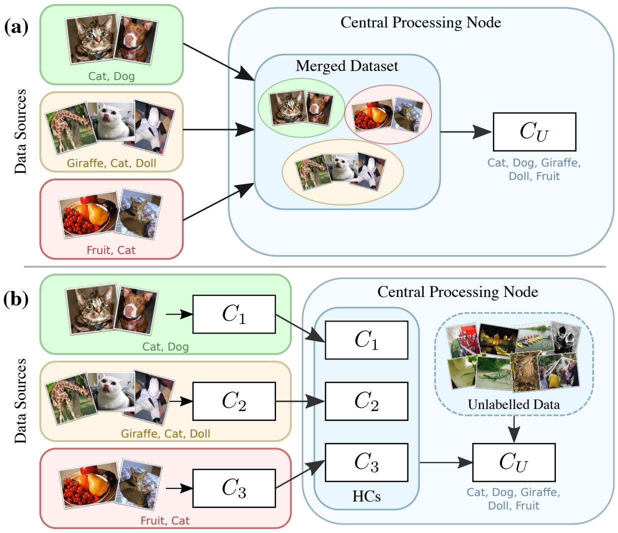

Figure 1. Unifying Heterogeneous Classifiers. (a) Common training approaches require transferring data from sources to a central processing node where a classifier is trained. (b) We propose to train a unified classifier from pre-trained classifiers from each source and an unlabelled set of generic data, thereby preserving privacy. The individual pre-trained classifiers may have different sets of target classes, hence the term *Heterogeneous Classifiers* (HCs).

In this paper, we aim to remove these limitations and propose a system for a more general scenario consisting of an ensemble of Heterogeneous Classifiers (HCs), as shown in Fig. 1b. We define a set of HCs as a set of classifiers which may have different architectures and, more importantly, may be trained to classify different sets of target classes. To combine the HCs, each entity only needs to forward their trained classifiers and class names to the central processing node, where all the HCs are unified into a single model that can classify all target classes of all input HCs. We refer to this problem as *Unifying Heterogeneous Classifiers* (UHC). UHC has practical applications for the cases when it is not possible to enforce every entity to (*i*) use the same model/architecture; (*ii*) collect sufficient training data for all classes; or (*iii*) send the data to the central node, due to computational, data availability, and confidentiality constraints.

To tackle UHC, we propose a generalisation of knowledge distillation [8, 17]. Knowledge distillation was originally proposed to compress multiple complex *teacher* models into a single simpler *student* one. However, distillation still assumes that the target classes of all teacher and student models are the same, whereas in this work we relax this limitation. To generalise distillation to UHC, we derive a probabilistic relationship connecting the outputs of HCs with that of the unified classifier. Based on this relationship, we propose two classes of methods, one based on cross-entropy minimisation and the other on matrix factorisation with missing entries, to estimate the probability over all classes of a given sample. After obtaining the probability, we can then use it to train the unified classifier. Our approach only requires unlabelled data to unify HCs, thus no labour is necessary to label any data at the central node. In addition, our approach can be applied to any classifiers which can be trained with soft labels, *e.g.*, neural networks, boosting classifiers, random forests, *etc*.

We evaluated our proposed approach extensively on ImageNet, LSUN, and Places365 datasets in a variety of settings and against a natural extension of the standard distillation. Through our experiments we show that our approach outperforms standard distillation and can achieve almost the same accuracy as the classifiers that were trained in a centralised, supervised manner.

2. Related Work

There exists a long history of research that aims to harness the power of multiple classifiers to boost classification result. The most well-known approaches are arguably ensemble methods [19, 23, 30] which combine the output of multiple classifiers to make a classification. Many techniques, such as voting and averaging [23], can merge prediction from trained classifiers, while some train the classifiers jointly as part of the technique, *e.g.*, boosting [13] and random forests [6]. These techniques have been successfully used in many applications, *e.g.*, multi-class classification [15], object detection [34, 27], tracking [1], *etc*. However, ensemble methods require storing and running all models for prediction, which may lead to scalability issues when complex models, *e.g.*, deep networks, are used. In addition, ensemble methods assume all base classifiers are trained to classify all classes, which is not suitable for the scenarios addressed by UHC.

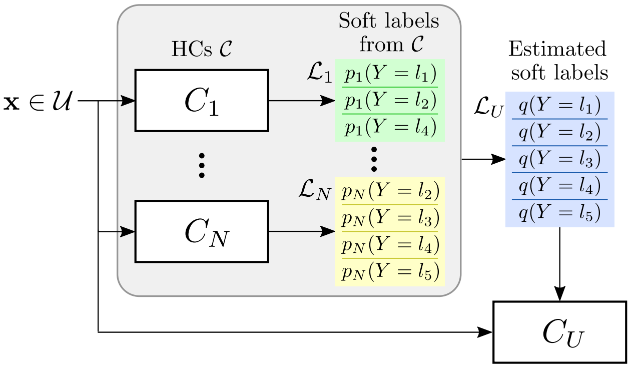

Figure 2. UHC problem and approach overview. An input image $\mathbf{x}$ is drawn from an unlabelled set $\mathcal{U}$ and input to a set of pre-trained classifiers $\{C_1, \cdots, C_N\}$, where each C_i returns soft label p_i over classes in $\mathcal{L}_i$. Here, the classes $\mathcal{L}_i$ may be different for each C_i. The goal of UHC is to train a classifier C_U that can classify all target classes in $\mathcal{L}_U$ using the prediction of C_i on $\mathbf{x} \in \mathcal{U}$ instead of labelled data. Our approach to UHC involves using p_i to estimate q, the soft label of $\mathbf{x}$ over all classes in $\mathcal{L}_U$, then using $\mathbf{x}$ and q to train C_U.

To the best of our knowledge, the closest class of methods to UHC is knowledge distillation [8, 17]. Distillation approaches operate by passing unlabelled data to a set of pretrained *teacher* models to obtain soft predictions, which are used to train a *student* model. Albeit originally conceived for compressing complex models into simpler ones by matching predictions, distillation has been further extended to, for instance, matching intermediate features [31], knowledge transfer between domains [14], combining knowledge using generative adversarial-based loss [35], *etc*. More related to UHC, Lopes *et al*. [26] propose to distill teacher models trained by different entities using their metadata rather than raw inputs. This allows the student model to be trained without any raw data transfer, thus preserving privacy while also not requiring any data collection from the central processing node. Still, no formulation of distillation can cope with the case where each teacher model has different target classes, which we tackle in this paper. We describe how distillation can be generalised to UHC in the next section.

3. Unifying Heterogeneous Classifiers (UHC)

We define the Unifying Heterogeneous Classifiers (UHC) problem in this paper as follows (see Fig. 2). Let $\mathcal{U}$ be an unlabelled set of images ("transfer set") and let $\mathcal{C} = \{C_i\}_{i=1}^N$ be a set of N Heterogeneous Classifiers (HCs), where each C_i is trained to predict the probability $p_i(Y = l_j)$ of an image belonging to class $l_j \in \mathcal{L}_i$. Given $\mathcal{U}$ and $\mathcal{C}$, the goal of this work is to learn a *uni-*

fied classifier C_U that estimates the probability $q(Y = l_j)$ of an input image belonging to the class $l_j \in \mathcal{L}_U$ where $\mathcal{L}_U = \bigcup_{i=1}^{N} \mathcal{L}_i = \{l_1, l_2, \ldots, l_L\}$. Note that C_i might be trained to classify different sets of classes, *i.e.*, we may have $\mathcal{L}_i \neq \mathcal{L}_j$ or even $|\mathcal{L}_i| \neq |\mathcal{L}_j|$ for $i \neq j$.

Our approach to tackle UHC involves three steps: (*i*) passing the image $\mathbf{x} \in \mathcal{U}$ to C_i to obtain $p_i, \forall i$, (*ii*) estimating q from $\{p_i\}_i$, then (*iii*) using the estimated q to train C_U in a supervised manner. We note that it is possible to combine (*ii*) and (*iii)* into a single step for neural networks (see Sec. 3.5.1), but this 3-step approach allows it to be applied to other classifiers, *e.g.*, boosting and random forests. To accomplish (*ii*), we derive probabilistic relationship between each p_i and q, which we leverage to estimate q via the following two proposed methods: cross-entropy minimisation and matrix factorisation. In the rest of this section, we first review standard distillation, showing why it cannot be applied to UHC. We then describe our approaches to estimate q from $\{p_i\}_i$. We provide a discussion on the computation cost in the supplementary material.

3.1. Review of Distillation

Overview Distillation [8, 17] is a class of algorithms used for compressing multiple trained models C_i into a single unified model C_U using a set of unlabelled data $\mathcal{U}$[1]. Referring to Fig. 2, standard distillation corresponds to the case where $\mathcal{L}_i = \mathcal{L}_j, \forall(i, j)$. The unified C_U is trained by minimising the cross-entropy between outputs of C_i and C_U as

$$J(q) = -\sum_i \sum_{l \in \mathcal{L}_U} p_i(Y = l) \log q(Y = l). \tag{1}$$

Essentially, the outputs of C_i are used as soft labels for the unlabelled $\mathcal{U}$ in training C_U. For neural networks, class probabilities are usually computed with softmax function:

$$p(Y = l) = \frac{\exp(z_l/T)}{\sum_{k \in \mathcal{L}_U} \exp(z_k/T)}, \tag{2}$$

where z_l is the logit for class l, and T denotes an adjustable temperature parameter. In [17], it was shown that minimising (1) when T is high is similar to minimising the ℓ_2 error between the logits of p and q, thereby relating the cross-entropy minimising to logit matching.

Issues The main issue with standard distillation stems from its inability to cope with the more general case of $\mathcal{L}_i \neq \mathcal{L}_j$. Mathematically, Eq. (1) assumes C_U and C_i's share the same set of classes. This is not true in our case since each C_i is trained to predict classes in $\mathcal{L}_i$, thus $p_i(Y = l)$ is undefined for $l \in \mathcal{L}_{-i}$[2]. A naive solution to this issue would be to simply set $p_i(Y = l) = 0$ for $l \in \mathcal{L}_{-i}$. However, this could incur serious errors, *e.g.*, one may set $p_i(Y = \text{cat})$ of a cat image to zero when C_i does not classify cats, which would be an improper supervision. We show that this approach does not provide good results in the experiments.

It is also worth mentioning that C_i in UHC is different from the *Specialised Classifiers* (SC) in [17]. While SCs are trained to specialise in classifying a subset of classes, they are also trained with data from other classes which are grouped together into a single *dustbin class*. This allows SCs to distinguish dustbin from their specialised classes, enabling student model to be trained with (1). Using the previous example, the cat image would be labelled as dustbin class, which is an appropriate supervision for SCs that do not classify cat. However, the presence of a dustbin class imposes a design constraint on the C_i's, as well as requiring the data source entities to collect large amounts of generic data to train it. Conversely, we remove these constraints in our formulation, and C_i's are trained without a dustbin class. Thus, given data from $\mathcal{L}_{-i}$, C_i will only provide p_i only over classes in $\mathcal{L}_i$, making it difficult to unify $\mathcal{C}$ with (1).

3.2. Relating outputs of HCs and unified classifier

To overcome the limitation of standard distillation, we need to relate the output p_i of each C_i to the probability q over $\mathcal{L}_U$. Since p_i is defined only in the subset $\mathcal{L}_i \subseteq \mathcal{L}_U$, we can consider $p_i(Y = l)$ as the probability q of $Y = l$ given that Y cannot be in $\mathcal{L}_{-i}$. This leads to the following derivation:

$$\begin{aligned}
p_i(Y = l) &= q(Y = l | Y \notin \mathcal{L}_{-i}) && (3)\\
&= q(Y = l | Y \in \mathcal{L}_i) && (4)\\
&= \frac{q(Y = l, Y \in \mathcal{L}_i)}{q(Y \in \mathcal{L}_i)} && (5)\\
&= \frac{q(Y = l)}{\sum_{k \in \mathcal{L}_i} q(Y = k)}. && (6)
\end{aligned}$$

We can see that $p_i(Y = l)$ is equivalent to $q(Y = l)$ normalised by the classes in $\mathcal{L}_i$. In the following sections, we describe two classes of methods that utilise this relationship for estimating q from $\{p_i\}_i$.

3.3. Method 1: Cross-entropy approach

Recall that the goal of (1) is to match q to p_i by minimising the cross-entropy between them. Based on the relation in (6), we generalise (1) to tackle UHC by matching $\frac{q(Y=l)}{\sum_{k \in \mathcal{L}_i} q(Y=k)}$ to $p_i(Y = k)$, resulting in:

$$J(q) = -\sum_i \sum_{l \in \mathcal{L}_i} p_i(Y = l) \log \hat{q}_i(Y = l), \tag{7}$$

where:

$$\hat{q}_i(Y = l) = \frac{q(Y = l)}{\sum_{k \in \mathcal{L}_i} q(Y = k)}. \tag{8}$$

[1] Labelled data can also be used in a supervised manner.

[2] We define $\mathcal{L}_{-i}$ as the set of classes in $\mathcal{L}_U$ but outside $\mathcal{L}_i$.

We can see that the difference between (1) and (7) lies in the normalisation of q. Specifically, the cross-entropy of each C_i (*i.e.*, the second summation) is computed between $p_i(Y = l)$ and $\hat{q}_i(Y = l)$ over the classes in $\mathcal{L}_i$. With this approach, we do not need to arbitrarily define values for $p_i(Y = l)$ whenever $l \in \mathcal{L}_{-i}$, thus not causing spurious supervision. We now outline optimality properties of (7).

Proposition 1 (Sufficient condition for optimality) Suppose there exists a probability $\bar{p}$ over $\mathcal{L}_U$, where $p_i(Y{=}l) = \frac{\bar{p}(Y=l)}{\sum_{k\in\mathcal{L}_i}\bar{p}(Y=k)}, \forall i$, then $q = \bar{p}$ is a global minimum of (7).

Sketch of proof Consider $\tilde{J}_i(\tilde{q}_i) = -\sum_{l\in\mathcal{L}_i} p_i(Y = l)\log\tilde{q}_i(Y = l)$ (Note $\tilde{J}_i$ is a function of $\tilde{q}_i$ whereas J is a function of q). $\tilde{J}_i(\tilde{q}_i)$ achieves its minimum when $\tilde{q}_i = p_i$, with the a value of $\tilde{J}_i(p_i)$. Thus, the minimum value of $\sum_i \tilde{J}_i(\tilde{q}_i)$ is $\sum_i \tilde{J}_i(p_i)$. This is a lower bound of (7), *i.e.*, $\sum_i \tilde{J}_i(p_i) \leq J(q), \forall q$. However, we can see that by setting $q = \bar{p}$, we achieve equality in the bound, *i.e.*, $\sum_i \tilde{J}_i(p_i) = J(\bar{p})$, and so $\bar{p}$ is a global minimum of (7). □

The above result establishes the form of a global minimum of (7), and that minimising (7) may obtain the true underlying probability $\bar{p}$ if it exists. However, there are cases where the global solution may not be unique. A simple example is when there are no shared classes between the HCs, *e.g.*, $N = 2$ with $\mathcal{L}_1 \cap \mathcal{L}_2 = \emptyset$. It may be possible to show uniqueness of the global solution in some cases depending on the structure of shared classes between $\mathcal{L}_i$'s, but we leave this as future work.

Optimisation Minimisation of (7) can be transformed into a geometric program (see supplementary material), which can then be converted to a convex problem and efficiently solved [3]. In short, we define $u_l \in \mathbb{R}$ for $l \in \mathcal{L}_U$ and replace $q(Y = l)$ with $\exp(u_l)$. Thus, (7) transforms to

$$\hat{J}(\{u_l\}_l) = -\sum_i \sum_{l\in\mathcal{L}_i} p_i(Y = l)\left(u_l - \log\left(\sum_{k\in\mathcal{L}_i}\exp(u_k)\right)\right), \tag{9}$$

which is convex in $\{u_l\}_l$ since it is a sum of scaled and log-sum-exps of $\{u_l\}_l$ [5]. We minimise it using gradient descent. Once the optimal $\{u_l\}_l$ is obtained, we transform it to q with the softmax function (2).

3.4. Method 2: Matrix factorisation approaches

Our second class of approaches is based on low-rank matrix factorisation with missing entries. Indeed, it is possible to cast UHC as a problem of filling an incomplete matrix of soft labels. During the last decade, low-rank matrix completion and factorisation [10, 11] have been successfully used in various applications, *e.g.*, structure from motion [18] and recommender systems [21]. It has also been used for multilabel classification in a transductive setting [9]. Here, we will describe how we can use matrix factorisation to recover soft labels q from $\{p_i\}_i$.

3.4.1 Matrix factorisation in probability space

Consider a matrix $\mathbf{P} \in [0, 1]^{L\times N}$ where we set P_{li} (the element in row l and column i) to $p_i(Y = l)$ if $l \in \mathcal{L}_i$ and zero otherwise. This matrix $\mathbf{P}$ is similar to the *decision profile matrix* in ensemble methods [23], but here we fill in 0 for the classes that C_i's cannot predict. To account for these missing predictions, we define $\mathbf{M} \in \{0, 1\}^{L\times N}$ as a mask matrix where $\mathbf{M}_{li}$ is 1 if $l \in \mathcal{L}_i$ and zero otherwise. Using the relation between p_i and q in (6), we can see that $\mathbf{P}$ can be factorised into a masked product of vectors as:

$$\mathbf{M} \odot \mathbf{P} = \mathbf{M} \odot (\mathbf{u}\mathbf{v}^\top), \tag{10}$$

$$\mathbf{u} = \begin{bmatrix} q(Y = l_1) \\ \vdots \\ q(Y = l_m) \end{bmatrix}, \mathbf{v} = \begin{bmatrix} \frac{1}{\sum_{l\in\mathcal{L}_1} q(Y=l)} \\ \vdots \\ \frac{1}{\sum_{l\in\mathcal{L}_N} q(Y=l)} \end{bmatrix}, \tag{11}$$

where $\odot$ is the Hadamard product. Here, $\mathbf{u}$ is the vector containing q, and each element in $\mathbf{v}$ contains the normalisation factor for each C_i. In this form, we can estimate the probability vector $\mathbf{u}$ by solving the following rank-1 matrix completion problem:

$$\underset{\mathbf{u},\mathbf{v}}{\text{minimise}} \quad \|\mathbf{M} \odot (\mathbf{P} - \mathbf{u}\mathbf{v}^\top)\|_F^2 \tag{12}$$

$$\text{subject to} \quad \mathbf{u}^\top \mathbf{1}_L = 1 \tag{13}$$

$$\mathbf{v} \geq \mathbf{0}_N, \mathbf{u} \geq \mathbf{0}_L, \tag{14}$$

where $\|\cdot\|_F$ denotes Frobenius norm, and $\mathbf{0}_k$ and $\mathbf{1}_k$ denote vectors of zeros and ones of size k. Here, the constraints ensure that $\mathbf{u}$ is a probability vector and that $\mathbf{v}$ remains non-negative so that the sign of probability in $\mathbf{u}$ is not flipped. This formulation can be regarded as a non-negative matrix factorisation problem [24], which we solve using Alternating Least Squares (ALS) [2] where we normalise $\mathbf{u}$ to sum to 1 in each iteration[3]. Due to gauge freedom [7], this normalisation in $\mathbf{u}$ does not affect the cost function.

3.4.2 Matrix factorisation in logit space

In Sec. 3.1, we discussed the relationship between minimising cross-entropy and logit matching under ℓ_2 distance. In this section, we consider applying matrix factorisation in logit space and show that our formulation is a generalisation of logit matching between C_i and C_U.

Let z_l^i be the given logit output of class l of C_i[4], and u_l be that of C_U to be estimated. Consider a matrix $\mathbf{Z} \in \mathbb{R}^{L\times N}$ where $Z_{li} = z_l^i$ if $l \in \mathcal{L}_i$ and zero otherwise. We

[3] We note there are more effective algorithms for matrix factorisation than ALS [7, 29, 11]. Here, we use ALS due to ease of implementation.

[4] For algorithms besides neural networks, we can obtain logits from probability via $z_l^i = \log p_i(Y = l)$.

can formulate the problem of estimating the vector of logits $\mathbf{u} \in \mathbb{R}^L$ as :

$$\underset{\mathbf{u},\mathbf{v},\mathbf{c}}{\text{minimise}} \; \|\mathbf{M} \odot (\mathbf{Z} - \mathbf{u}\mathbf{v}^\top - \mathbf{1}_L\mathbf{c}^\top)\|_F^2 + \lambda(\|\mathbf{u}\|_2^2 + \|\mathbf{v}\|_2^2) \tag{15}$$

$$\text{subject to } \mathbf{v} \geq \mathbf{0}_N, \tag{16}$$

where $\mathbf{c} \in \mathbb{R}^N$ deals with shift in logits[5], and $\lambda \in \mathbb{R}$ is a hyperparameter controlling regularisation [7]. Here, optimising $\mathbf{v} \in \mathbb{R}^N$ is akin to optimising the temperature of logits [17] from each source classifier, and we constrained it to be nonnegative to prevent the logit sign flip, which could affect the probability.

Relation to logit matching The optimisation in (15) has three variables. Since $\mathbf{c}$ is unconstrained, we derive its closed form solution and remove it from the formulation. This transforms (15) into:

$$\underset{\mathbf{u},\mathbf{v}}{\text{minimise}} \; \sum_{i=1}^{N} \left\|\mathcal{P}_{|\mathcal{L}_i|}\left([\mathbf{z}_i - \mathbf{u}v_i]_{\mathcal{L}_i}\right)\right\|_2^2 + \lambda(\|\mathbf{u}\|_2^2 + \|\mathbf{v}\|_2^2) \tag{17}$$

$$\text{subject to } \mathbf{v} \geq \mathbf{0}_N, \tag{18}$$

where $\mathbf{z}_i$ is the i^{th} column of $\mathbf{Z}$; $[\mathbf{x}]_{\mathcal{L}_i}$ selects the elements of $\mathbf{x}$ which are indexed in $\mathcal{L}_i$; and $\mathcal{P}_k(\mathbf{x}) = (\mathbf{I}_k - \frac{1}{k}\mathbf{1}_k\mathbf{1}_k^\top)\mathbf{x}$ is the orthogonal projector that removes the mean from the vector $\mathbf{x} \in \mathbb{R}^k$. This transformation simplifies (15) to contain only $\mathbf{u}$ and $\mathbf{v}$. We can see that this formulation minimises the ℓ_2 distance between logits, but instead of considering all classes in $\mathcal{L}_U$, each term in the summation considers only the classes in $\mathcal{L}_i$. In addition, (17) also includes regularisation and optimises for scaling in $\mathbf{v}$. Thus, we can say that (15) is a generalisation of logit matching for UHC.

Optimisation While (17) has fewer parameters than (15), it is more complicated to optimise as the elements in $\mathbf{u}$ are entangled due to the projector. Instead, we solve (15) using ALS over $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{c}$. Here, there is no constraint on $\mathbf{u}$, so we do not normalise it as in Sec. 3.4.1.

Alternative approach: Setting $\mathbf{v}$ as a constant While setting $\mathbf{v}$ as a variable allows (15) to handle different scalings of logits, it also introduces cumbersome issues. Specifically, the gauge freedom in $\mathbf{u}\mathbf{v}^\top$ may lead to arbitrary scaling in $\mathbf{u}$ and $\mathbf{v}$, *i.e.*, $\mathbf{u}\mathbf{v}^\top = (\mathbf{u}/\alpha)(\alpha\mathbf{v}^\top)$ for $\alpha \neq 0$. Also, while the regularisers help prevent the norms of $\mathbf{u}$ and $\mathbf{v}$ to become too large, it is difficult to set a single λ that works well for all data in $\mathcal{U}$. To combat these issues, we propose another formulation of (15) where we fix $\mathbf{v} = \mathbf{1}_N$. With $\mathbf{v}$ fixed, we do not require to regularise $\mathbf{u}$ since its scale is determined by $\mathbf{Z}$. In addition, the new formulation is convex and can be solved to global optimality. We solve this alternative formulation with gradient descent.

3.5. Extensions

In Secs. 3.3 and 3.4, we have described methods for estimating q from $\{p_i\}$ then using q as the soft label for training C_U. In this section, we discuss two possible extensions applicable to all the methods: (*i*) direct backpropagation for neural networks and (*ii*) fixing imbalance in soft labels.

3.5.1 Direct backpropagation for neural networks

Suppose the unified classifier C_U is a neural network. While it possible to use q to train C_U in a supervised manner, we could also consider an alternative where we directly backpropagate the loss without having to estimate q first. In the case of cross-entropy (Sec. 3.3), we can think of q as the probability output from C_U, through which we can directly backpropagate the loss. In the case of matrix factorisation (Sec. 3.4), we could consider $\mathbf{u}$ as the vector of probability (Sec. 3.4.1) or logit (Sec. 3.4.2) outputs from C_U. Once $\mathbf{u}$ is obtained from C_U, we plug it in each formulation, solve for other variables (*e.g.*, $\mathbf{v}$ and $\mathbf{c}$) with $\mathbf{u}$ fixed, then backpropagate the loss via $\mathbf{u}$. Directly backpropagating the loss merges the steps of estimating q and using it to train C_U into a single step.

3.5.2 Balancing soft labels

All the methods we have discussed are based on individual samples: we estimate q from $\{p_i\}$ of a single $\mathbf{x}$ from the transfer set $\mathcal{U}$ and use it to train C_U. However, we observe that the set of estimated q's from the whole $\mathcal{U}$ could be imbalanced. That is, the estimated q's may be biased towards certain classes more than others. To counter this effect, we apply the common technique of weighting the cross-entropy loss while training C_U [28]. The weight of each class l is computed as the inverse of the mean of $q(Y = l)$ over all data from $\mathcal{U}$.

4. Experiments

In this section, we perform experiments to compare different methods for solving UHC. The main experiments on ImageNet, LSUN, and Places365 datasets are described in Sec. 4.1, while sensitivity analysis is described in Sec. 4.2.

We use the following abbreviations to denote the methods. **SD** for the naive extension of Standard Distillation (Sec. 3.1) [17]; **CE-X** for Cross-Entropy methods (Sec. 3.3); **MF-P-X** for Matrix Factorization in Probability space (Sec. 3.4.1); and **MF-LU-X** and **MF-LF-X** for Matrix Factorization in Logit space with Unfixed and Fixed $\mathbf{v}$ (Sec. 3.4.2), *resp.* The suffix '**X**' is replaced with '**E**' if we estimate q first before using it as soft label to train C_U; with '**BP**' if we perform direct backpropagation from the loss function (Sec. 3.5.1); and with '**BS**' if we estimate and balance the soft labels q before training C_U (Sec. 3.5.2).

[5] Recall that a shift in logit values has no effect on the probability output, but we need to account for the different shifts from the C_i's to cast it as matrix factorisation.

Table 1. HC configurations for the main experiment

Dataset	#Classes in $\mathcal{L}_U$ (L)	#HCs (N)	#Classes for each HC	
			Random	Compl. overlap.
ImageNet	20-50	10-20	5-15	$= L$
LSUN	5-10	3-7	2-5	$= L$
Places365	20-50	10-20	5-15	$= L$

In addition to the mentioned methods, we also include **SD-BS** as the SD method with balanced soft labels, and **SPV** as the method trained directly in a supervised fashion with all training data of all C_i's as a benchmark. For MF-LU-X methods, we used $\lambda = 0.01$. All methods use temperature $T = 3$ to smooth the soft labels and logits (See (2) and [17]).

4.1. Experiment on large image datasets

In this section, we describe our experiment on ImageNet, LSUN, and Places365 datasets. First, we describe the experiment protocols, providing details on the datasets, architectures used as C_i and C_U, and the configurations of C_i. Then, we discuss the results.

4.1.1 Experiment protocols

Datasets We use three datasets for this experiment. (*i*) ImageNet (ILSVRC2012) [32], consisting of 1k classes with ~700 to 1300 training and 50 validation images per class, as well as 100k unlabelled test images. In our experiments, the training images are used as training data for the C_i's, the unlabelled test images as $\mathcal{U}$, and the validation images as our test set to evaluate the accuracy. (*ii*) LSUN [36], consisting of 10 classes with ~100k to 3M training and 300 validation images per class with 10k unlabelled test images. Here, we randomly sample a set of 1k training images per class to train the C_i's, a second randomly sampled set of 20k images per class also from the training data is used as $\mathcal{U}$, and the validation data is used as our test set. (*iii*) Places365 [37], consisting of 365 classes with ~3k to 5k training and 100 validation images per class, as well as ~329k unlabelled test images. We follow the same usage as in ImageNet, but with 100k samples from the unlabelled test images as $\mathcal{U}$. We preprocess all images by centre cropping and scaling to 64×64 pixels.

HC configurations We test the proposed methods under two configurations of HCs (see summary in Table 1). (*i*) Random classes. For ImageNet and Places365, in each trial, we sample 20 to 50 classes as $\mathcal{L}_U$ and train 10 to 20 C_i's where each is trained to classify 5 to 15 classes. For LSUN, in each trial, we sample 5 to 10 classes as $\mathcal{L}_U$ and train 3 to 7 C_i's where each is trained to classify 2 to 5 classes. We use this configuration as the main test for when C_i's classify different sets of classes. (*ii*) Completely overlapping classes. Here, we use the same configurations as in (*i*) except all C_i's are trained to classify all classes in $\mathcal{L}_U$. This case is used to test our proposed methods under the common configurations where all C_i and C_U share the same classes. Under both configurations, $\mathcal{U}$ consist of a much wider set of classes than $\mathcal{L}_U$. In other words, a large portion of the images in $\mathcal{U}$ does not fall under any of the classes in $\mathcal{L}_U$.

Models Each C_i is randomly selected from one of the following four architectures with ImageNet pre-trained weights: AlexNet [22], VGG16 [33], ResNet18, and ResNet34 [16]. For AlexNet and VGG16, we fix the weights of their feature extractor portion, replace their `fc` layers with two `fc` layers with 256 hidden nodes (with BatchNorm and ReLU), and train the `fc` layers with their training data. Similarly in ResNet models, we replace their `fc` layers with two `fc` layers with 256 hidden nodes as above. In addition, we also fine-tune the last residual block. As for C_U, we use two models, VGG16 and ResNet34, with similar settings as above.

For all datasets and configurations, we train each C_i with 50 to 200 samples per class; no sample is shared between any C_i in the same trial. These C_i's together with $\mathcal{U}$ are then used to train C_U. We train all models for 20 epochs with SGD optimiser (step sizes of 0.1 and 0.01[6] for first and latter 10 epochs with momentum 0.9). To control the variation in results, in each trial we initialise instances of C_U's from the same architecture using the same weights and we train them using the same batch order. In each trial, we evaluate the C_U's of all methods on the test data from all classes in $\mathcal{L}_U$. We run 50 trials for each dataset, model, and HC configuration combination. The results are reported in the next section.

4.1.2 Results

Table 2 shows the results for this experiment. Each column shows the average accuracy of each method under each experiment setting, where the best performing method is shown in underlined bold. To test statistical significance, we choose Wilcoxon signed-rank test over standard deviation to cater for the vastly different settings (*e.g.*, model architectures, number of classes and HCs, *etc.*) across trials. We run the test between the best performing method in each experiment and the rest. Methods where the performance is not statistically significantly different from the best method at $\alpha = 0.01$ are shown in bold.

First, let us observe the result for the *random classes* case which addresses the main scenario of this paper, *i.e.*, when each HC is trained to classify different sets of classes. We can make the following observations.

All proposed methods perform significantly better than SD. We can see that all methods in (A), (B), and (C)

[6] For MF-P-BP, we use $150\times$ the rates as its loss has a smaller scale.

Table 2. Average accuracy of UHC methods over different combinations of HC configurations, datasets, and unified classifier models. (**Underline bold**: Best method. **Bold**: Methods which are not statistically significantly different from the best method.)

Methods	Random Classes						Completely Overlapping Classes					
	ImageNet		LSUN		Places365		ImageNet		LSUN		Places365	
	VGG16	ResNet34	VGG16	ResNet34	VGG16	ResNet34	VGG16	ResNet34	VGG16	ResNet34	VGG16	ResNet34
SPV (Benchmark)	.7212	.6953	.6664	.6760	.5525	.5870	.7345	.7490	.6769	.7017	.5960	.6460
SD	.5543	.5562	.5310	.5350	.4390	.4564	**.7275**	**.7292**	.7004	**.7041**	**.6163**	**.6402**
(A) *Estimate q methods*												
CE-E	**.6911**	**.6852**	.6483	**.6445**	**.5484**	**.5643**	**.7276**	**.7290**	.7002	.7036	**.6162**	**.6406**
MF-P-E	.6819	.6747	.6443	.6406	.5349	.5488	**.7280**	**.7297**	**.7012**	**.7052**	**.6167**	**.6406**
MF-LV-E	.6660	.6609	.6348	.6330	.5199	.5414	.7231	.7242	**.7031**	**.7043**	.6129	.6374
MF-LF-E	.6886	**.6833**	.6490	**.6458**	.5441	.5609	.7265	**.7279**	**.7015**	**.7057**	**.6161**	**.6397**
(B) *Backprop. methods*												
CE-BP	**.6902**	**.6869**	**.6520**	.6439	.5466	**.5669**	**.7275**	**.7288**	.7003	**.7040**	**.6161**	**.6400**
MF-P-BP	**.6945**	**.6872**	.6480	**.6417**	**.5471**	.5609	**.7277**	**.7287**	.6999	**.7019**	.6146	.6384
MF-LV-BP	.6889	**.6847**	**.6495**	.6389	.5467	**.5681**	.7229	.7225	.7001	**.7046**	.6113	.6369
MF-LF-BP	.6842	**.6840**	**.6523**	**.6445**	.5383	**.5624**	.7239	.7252	**.7020**	.7034	.6104	.6366
(C) *Balanced soft labels*												
SD-BS	.6629	.6574	.6343	.6345	.5283	.5433	.7217	.7214	.6979	.7017	.6094	.6320
CE-BS	**.6928**	**.6856**	.6513	**.6464**	**.5548**	**.5687**	.7215	.7213	.6979	.7018	.6094	.6323
MF-P-BS	.6851	.6756	.6474	**.6450**	.5455	.5546	.7243	.7252	.6996	.7041	.6124	.6355
MF-LV-BS	.6772	.6682	.6388	.6357	.5346	.5497	.7168	.7173	.7014	.7028	.6063	.6301
MF-LF-BS	**.6935**	**.6865**	**.6549**	**.6485**	**.5544**	**.5692**	.7210	.7215	.6998	.7035	.6101	.6330

of Table 2 outperform SD by a large margin of 9-15%. This shows that simply setting probability of undefined classes in each HC to 0 may significantly deteriorate the accuracy. On the other hand, our proposed methods achieve significantly better results and almost reach the same accuracy as SPV with a gap of 1-4%. This suggests the soft labels from HCs can be used for unsupervised training at a little expense of accuracy, even though $\mathcal{U}$ contains a significant proportion of images that are not part of the target classes. Still, there are several factors that may affect the capability of C_U from reaching the accuracy of SPV, *e.g.*, accuracy of C_i, their architectures, *etc*. We look at some of these in the sensitivity analysis section.

MF-LF-BS performs well in all cases. We can see that different algorithms perform best under different settings, but MF-LF-BS always performs best or has no statistical difference from the best methods. This suggests MF-LF-BS could be the best method for solving UHC. At the same time, CE methods offer a good trade-off between high accuracy and ease of implementation, which makes them a good alternative for the UHC problem.

Besides these main points, we also note the following small but consistent trends.

Balancing soft labels helps improve accuracy. While the improvement may be marginal (less than 1.5%), we can see that 'BS' methods in (C) consistently outperform their 'E' counterparts in (A). Surprisingly, SD-BS, which is SD with balanced soft labels, also significantly improved over SD by more than 10%. These results indicate that it is a good practice to use balanced soft labels to solve UHC. Note that while SD-BS received significant boost, it still generally underperforms compared to CE and MF methods, suggesting that it is important to incorporate the relation between $\{p_i\}$ and q into training.

Nonconvex losses perform better with 'BP'. Methods with suffixes 'E' and 'BS' in (A) and (C) are based on estimating q before training C_U, while 'BP' in (B) directly perform backpropagation from the loss function. As seen in Sec. 3, the losses of CE and MF-LF are convex in their variables while MF-P and MF-LV are nonconvex. Here, we observe a small but interesting effect that methods with nonconvex losses perform better with 'BP'. We speculate that this is due to errors in the estimation of q trickling down to the training of C_U if the two steps are separated. Conversely in 'BP', where the two steps are merged into a single step, such issue might be avoided. More research would be needed to confirm this speculation. For convex losses (CE and MF-LF), we find no significant patterns between 'E' in (A) and 'BP' in (B).

Next, we discuss the *completely overlapping case*.

All methods perform rather well. We can see that all methods, including SD, achieve about the same accuracy (within ~1% range). This shows that our proposed methods can also perform well in the common cases of all C_i's being trained to classify all classes and corroborates the claim that our proposed methods are generalisations of distillation.

Not balancing soft labels performs better. We note that balancing soft labels tends to slightly deteriorate the accuracy. This is the opposite result from the random classes case. Here, even SD-BS which receive an accuracy boost in the random classes case also performs worse than its counterpart SD. This suggests not balancing soft labels may be a

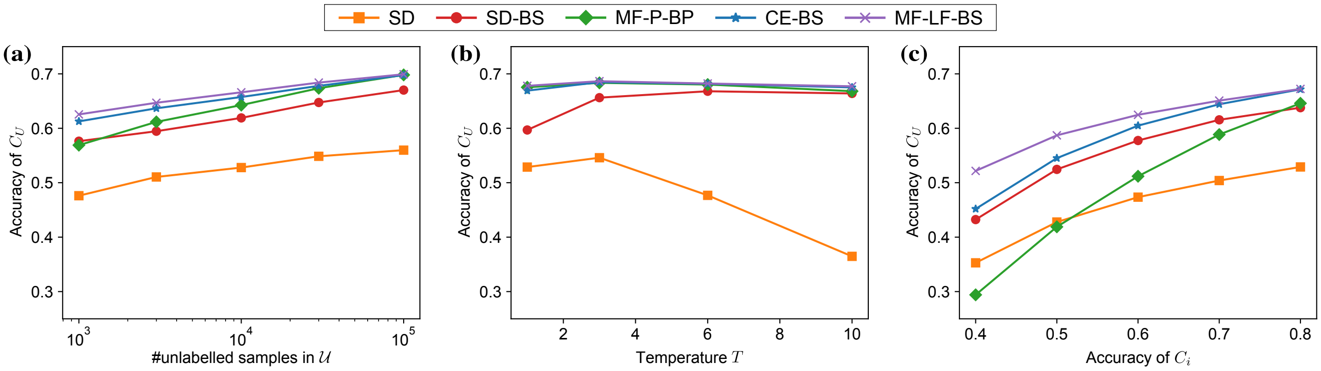

Figure 3. Sensitivity analysis results. (a) Size of unlabelled set. (b) Temperature. (c) Accuracy of HCs.

better option for overlapping classes case.

Distillation may outperform its supervised counterparts. For LSUN and Places365 datasets, we see that many times distillation methods performs better than SPV. Especially for the case of VGG16, we see SPV consistently perform worse than other methods by 1 to 3% in most of the trials. This shows that it is possible that distillation-based methods may outperform their supervised counterparts.

4.2. Sensitivity Analysis

In this section, we perform three sets of sensitivity analysis on the effect of size of the transfer set, temperature parameter T, and accuracy of HCs. We use the same settings as the ImageNet random classes experiment in the previous section with VGG16 as C_U. We run 50 trials for each test. We evaluate the following five methods as the representative set of SD and top performing methods from previous section: SD, SD-BS, MF-P-BP, MF-LF-BS, and CE-BS.

Size of transfer set We use this test to evaluate the effect of the number of unlabelled samples in the transfer set $\mathcal{U}$. We vary the number of samples from 10^3 to 10^5. The result is shown in Fig. 3a. As expected, we can see that all methods deteriorate as the size of transfer set decreases. In this test, MF-P-BP is the most affected by the decrease as its accuracy drops fastest. Still, all other methods perform better than SD in the whole test range, illustrating the robustness to transfer sets with different sizes.

Temperature In this test, we vary the temperature T used for smoothing the probability $\{p_i\}$ (see (2) or [17]) before using them to estimate q or train C_U. The values evaluated are $T = 1, 3, 6,$ and 10. The result is shown in Fig. 3b. We can see that the accuracies of SD and SD-BS drop significantly when T is set to high and low values, *resp*. On the other hand, the other three methods are less affected by different values of T.

HCs' accuracies In this test, we evaluate the robustness of UHC methods against varying accuracy of C_i. The test protocol is as follows. In each trial, we vary the accuracy of all C_i's to 40-80%, obtain p_i from the C_i's, and use them to perform UHC. To vary the accuracy of each C_i, we take 50 samples per class from training data as the adjustment set, completely train each C_i from the remaining training data, then inject increasing Gaussian noise into the last `fc` layer until its accuracy on the adjustment set drops to the desired value. If the initial accuracy of C_i is below the desired value then we simply use the initial C_i. The result of this evaluation is shown in Fig. 3c. We can see that the accuracy of all methods increase as the C_i's perform better, illustrating that the accuracy of C_i is an important factor for the performance of UHC methods. We can also see that MF-P-BP is most affected by low accuracy of C_i while MF-LF-BS is the most robust.

Based on the sensitivity analysis, we see that MF-LF-BS is the most robust method against the number of samples in the transfer set, temperature, and accuracy of the HCs. This result provides further evidence that MF-LF-BS should be the suggested method for solving UHC. We provide the complete sensitivity plots with all methods in the supplementary material.

5. Conclusion

In this paper, we formalise the problem of unifying knowledge from heterogeneous classifiers (HCs) using only unlabelled data. We proposed cross-entropy minimisation and matrix factorisation methods for estimating soft labels of the unlabelled data from the output of HCs based on a derived probabilistic relationship. We also proposed two extensions to directly backpropagate the loss for neural networks and to balance estimated soft labels. Our extensive experiments on ImageNet, LSUN, and Places365 show that our proposed methods significantly outperformed a naive extension of knowledge distillation. The result together with additional three sensitivity analysis suggest that an approach based on matrix factorization in logit space with balanced soft labels is the most robust approach to unify HCs into a single classfier.

References

[1] Shai Avidan. Ensemble tracking. *IEEE TPAMI*, 29(2):261–271, 2007. 2

[2] Michael W. Berry, Murray Browne, Amy N. Langville, Paul V. Pauca, and Robert J. Plemmons. Algorithms and applications for approximate nonnegative matrix factorization. *Computational statistics & data analysis*, 52(1):155–173, 2007. 4

[3] Stephen Boyd, Seung-Jean Kim, Lieven Vandenberghe, and Arash Hassibi. A tutorial on geometric programming. *Optimization and engineering*, 8(1):67, 2007. 4

[4] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, Jan. 2011. 1

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

[6] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. 2

[7] Aeron M. Buchanan and Andrew W. Fitzgibbon. Damped newton algorithms for matrix factorization with missing data. In *CVPR*, 2005. 4, 5

[8] Cristian Bucila, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *ACM SIGKDD*, pages 535–541, 2006. 2, 3

[9] Ricardo S. Cabral, Fernando De la Torre, João P. Costeira, and Alexandre Bernardino. Matrix completion for multi-label image classification. In *NIPS*, 2011. 4

[10] Emmanuel J. Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational mathematics*, 9(6):717, 2009. 4

[11] Alessio Del Bue, Joao Xavier, Lourdes Agapito, and Marco Paladini. Bilinear modeling via augmented lagrange multipliers (BALM). *IEEE TPAMI*, 34(8):1496–1508, 2012. 4

[12] Pedro A. Forero, Alfonso Cano, and Georgios B. Giannakis. Consensus-based distributed support vector machines. *JMLR*, 11:1663–1707, Aug. 2010. 1

[13] Yoav Freund and Robert Schapire. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999. 2

[14] Saurabh Gupta, Judy Hoffman, and Jitendra Malik. Cross modal distillation for supervision transfer. In *CVPR*, 2016. 2

[15] Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou. Multi-class adaboost. *Statistics and its Interface*, 2(3):349–360, 2009. 2

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 6

[17] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015. 2, 3, 5, 6, 8

[18] Qifa Ke and Takeo Kanade. Robust l1 norm factorization in the presence of outliers and missing data by alternative convex programming. In *CVPR*, 2005. 4

[19] Josef Kittler, Mohamad Hatef, Robert P. W. Duin, and Jiri Matas. On combining classifiers. *IEEE TPAMI*, 20(3):226–239, 1998. 2

[20] Jakub Konečný, Brendan H. McMahan, Felix X. Yu, Peter Richtarik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. In *NIPS Workshop on Private Multi-Party Machine Learning*, 2016. 1

[21] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009. 4

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 6

[23] Ludmila I. Kuncheva. *Combining pattern classifiers: methods and algorithms*. John Wiley & Sons, 2004. 2, 4

[24] Daniel D Lee and Sebastian H. Seung. Algorithms for non-negative matrix factorization. In *NIPS*, 2001. 4

[25] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014. 1

[26] Raphael Gontijo Lopes, Stefano Fenu, and Thad Starner. Data-free knowledge distillation for deep neural networks. In *NIPS workshop on learning with limited labeled data*, 2017. 2

[27] Tomasz Malisiewicz, Abhinav Gupta, and Alexei A Efros. Ensemble of exemplar-SVMs for object detection and beyond. In *ICCV*, 2011. 2

[28] Andrew Ng. *Machine Learning Yearning*, chapter 39, page 76. deeplearning.ai, 2018. 5

[29] Takayuki Okatani, Takahiro Yoshida, and Koichiro Deguchi. Efficient algorithm for low-rank matrix factorization with missing components and performance comparison of latest algorithms. In *ICCV*, 2011. 4

[30] Robi Polikar. Ensemble based systems in decision making. *IEEE Circuits and Systems Magazine*, 6(3):21–45. 2

[31] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. FitNets: Hints for thin deep nets. In *ICLR*, 2015. 2

[32] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015. 1, 6

[33] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 6

[34] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR*, 2001. 2

[35] Zheng Xu, Yen-Chang Hsu, and Jiawei Huang. Learning loss for knowledge distillation with conditional adversarial networks. In *ICLR workshop*, 2017. 2

[36] Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. LSUN: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015. 6

[37] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 million image database for scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 40(6):1452–1464, 2018. 6