Neural Data Server: A Large-Scale Search Engine for Transfer Learning Data

Xi Yan[1,2*] David Acuna[1,2,3*] Sanja Fidler[1,2,3]
[1]University of Toronto [2]Vector Institute [3]NVIDIA
xi.yan@mail.utoronto.ca, {davidj, fidler}@cs.toronto.edu

Abstract

Transfer learning has proven to be a successful technique to train deep learning models in the domains where little training data is available. The dominant approach is to pretrain a model on a large generic dataset such as ImageNet and finetune its weights on the target domain. However, in the new era of an ever increasing number of massive datasets, selecting the relevant data for pretraining *is a critical issue. We introduce Neural Data Server (NDS), a large-scale search engine for finding the most useful transfer learning data to the target domain. NDS consists of a* dataserver *which indexes several large popular image datasets, and aims to recommend data to a* client*, an end-user with a target application with its own small labeled dataset. The* dataserver *represents large datasets with a much more compact mixture-of-experts model, and employs it to perform data search in a series of* dataserver-client *transactions at a low computational cost. We show the effectiveness of NDS in various transfer learning scenarios, demonstrating state-of-the-art performance on several target datasets and tasks such as image classification, object detection and instance segmentation. Neural Data Server is available as a web-service at* http://aidemos.cs.toronto.edu/nds/.

1. Introduction

In recent years, we have seen an explosive growth of the number and the variety of computer vision applications. These range from generic image classification tasks to surveillance, sports analytics, clothing recommendation, early disease detection, and to mapping, among others. Yet, we are only at the beginning of our exploration of what is possible to achieve with Deep Learning.

One of the critical components of the new age of computer vision applications is the need for labeled data. To achieve high performance, typically a massive amount of data needs to be used to train deep learning models. Transfer learning provides a promising approach to reduce the need for large-scale labeled data for each target application. In transfer learning, a neural network is pretrained [11, 24, 43] on existing large generic datasets and then fine-tuned in the target domain. While transfer learning is a well studied concept that has been proven successful in many applications [11, 24, 43], deciding which data to use for pretraining the model is an open research question that has received surprisingly little attention in the literature. We argue that this is a crucial problem to be answered in light of the ever increasing scale of the available datasets.

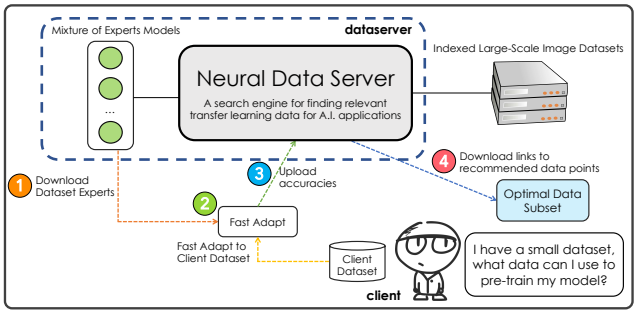

Figure 1: **Neural Data Server**: Search engine for finding relevant transfer learning data for the user's target domain. In NDS, a *dataserver* indexes several popular image datasets, represents them with a mixture-of-experts model, and uses *client*'s target data to determine most relevant samples. Note that NDS **indexes** available public datasets and **does not host** them. Data recommendation is done by providing **links** to relevant examples.

To emphasize our point, recent efforts on curating computer vision benchmarks[1] list over 400 public datasets, ranging from generic imagery, faces, fashion photos, to self-driving data. Furthermore, the dataset sizes are significantly increasing: the recently released OpenImages [31] contains 9M labeled images (600GB in size), and is 20 times larger compared to its predecessor MS-COCO [33] (330K images, 30GB). The video benchmark YouTube8m [1] (1.9B frames, 1.5TB), is 800× larger compared to Davis [9] (10k frames, 1.8GB), while the recently released autonomous driving dataset nuScenes [10] contains 100× the number of frames than KITTI [20] which was released in 2012.

It is evident that downloading and storing these datasets locally is already cumbersome and expensive. This is further amplified by the computational resources required for training neural networks on this massive amount of data.

*authors contributed equally

[1]Websites listing CV datasets: https://www.visualdata.io/, https://pytorch.org/docs/stable/torchvision/datasets.html, https://datasetsearch.research.google.com/

Figure 2: Examples of images from the *dataserver* (COCO+OpenImages) recommended to each *client* dataset by our Neural Data Server.

The latter is an even more pronounced issue in research, where the network architectures are continuously being developed and possibly many need to be tested. Furthermore, for commercial applications, data licensing may be another financial issue to consider. Recent works [23, 37] have also shown that there is not a "the more the better" relationship between the amount of pretraining data and the downstream task performance. Instead, they showed that selecting an appropriate subset of the data was important to achieve good performance on the target dataset.

In this paper, we introduce Neural Data Server (NDS), a large-scale search engine for finding the most useful transfer learning data to the target domain. One can imagine NDS as a web-service where a centralized server, referred to as the *dataserver*, recommends data to *client*s (Fig 1). A *client* is an end-user with an A.I. application in mind, and has a small set of labeled target data. We assume that each client is only interested in downloading a subset of the server-indexed data that is most relevant to the client's target domain, limited to the user-specified budget (maximum desired size). We further require the transaction between the *dataserver* and the *client* to be both computationally efficient and privacy-preserving. This means the client's data should not be visible to the server. We also aim to minimize the amount of *dataserver*'s online computation per client, as it may possibly serve many clients in parallel.

We index several popular image datasets and represent them using a mixture-of-experts (MoE) model, which we store on the *dataserver*. MoE is significantly smaller in size than the data itself, and is used to probe the usefulness of data in the *client*'s target domain. In particular, we determine the accuracy of each expert on the target dataset, and recommend data to the *client* based on these accuracies.

We experimentally show significant performance improvements on several downstream tasks and domains compared to baselines. Furthermore, we show that with only 20% of pretraining data, our method achieves comparable or better performance than pretraining on the entire *dataserver*-indexed datasets. We obtain significant improvements over ImageNet pretraining by downloading only 26 Gb of server's data in cases when training on the entire *dataserver* (538 Gb) would take weeks. Our Neural Data Server will be made available as a web-service with the aim of improving performance and reducing the development cost of the end-users' A.I. applications.

2. Related Work

Transfer Learning. The success of deep learning and the difficulty of collecting large scale datasets has recently brought significant attention to the long existing history of transfer learning, cross-domain annotation and domain adaptation [39, 15, 4, 44, 3, 46]. Specifically in the context of neural networks, fine-tuning a pretrained model on a new dataset is the most common strategy for knowledge transfer.

Most literature in this domain analyzes the effect of pretraining on large-scale datasets [44, 34, 17] with respect to network architectures, network layers, and training tasks [49, 50]. Concurrent with our work, Achille *et al*. [2] proposes a framework for selecting the best pre-trained feature extractor for a new task from a collection of classifiers. In contrast, our work aims to identify the optimal set of data points for pre-training. Works most related to ours are [16, 37] which show that pretraining on only relevant examples is important to achieve good performance on fine-grained classification tasks. Specifically, in [16] the authors use a predefined similarity metric between the source and target categories in order to greedily select the most similar categories from the source dataset to be used for pretraining. [37], on the other hand, exploits a model pretrained on the source domain to obtain pseudolabels of the target images, and uses these to re-weight the source examples.

Unlike ours, [16, 37] are limited to classification tasks, and do not easily scale to a constantly growing datacenter (the model needs to be retrained each time a new dataset is added). Thus, their approach does not naturally handle our scenario in which indexed datasets have diverse sets of tasks and labels, and where the number of indexed datasets may grow over time.

Federated Learning. [35, 8] introduce a distributed ML

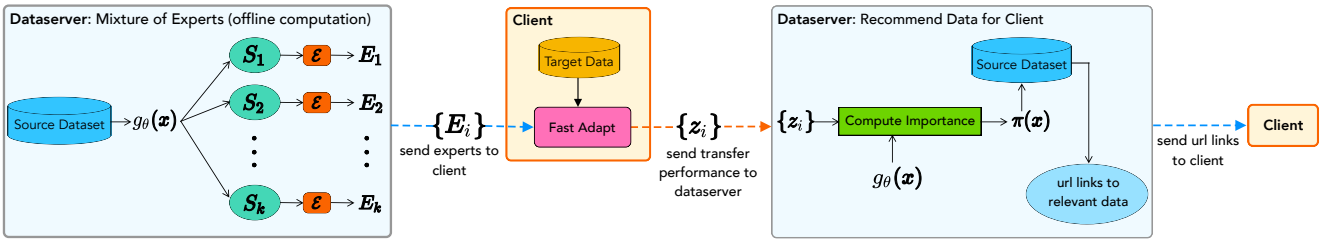

Figure 3: Overview of Neural Data Server. NDS consists of a *dataserver* that represents indexed datasets using a mixture-of-experts model. Experts are sent to *client* in order to compute accuracies in the *client*'s target domain. These accuracies are then used by *dataserver* to recommend relevant data samples.

Algorithm 1 Dataserver's Modules

Require representation learning alg. $\mathcal{E}$, number of experts K
$g_\theta \leftarrow$ HARDGATING$(\mathcal{S}, K)$ ▷ Sec 3.2: partition $\mathcal{S}$ into local subsets to obtain gating
$\mathcal{S}_i := \{x \in \mathcal{S} | g_{\theta,i}(x) = 1\}$
procedure MOE$(\{\mathcal{S}_i\}, \mathcal{E}, K)$:
  For $i = 1, ..., K$
    Run $\mathcal{E}$ on $\mathcal{S}_i$ to obtain expert e_{θ_i}
  return $\{e_{\theta_i}\}$
procedure OUTPUTDATA$(\{S_i\}, z,$ budget):
  $w \leftarrow \text{softmax}(z, T = 0.1)$
  $\pi(x) = \sum_{i=1}^{K} w_i \, g_{\theta,i}(x) \frac{1}{|S_i|}$
  Sample $\mathcal{S}_*$ from $\mathcal{S}$ at rate according to $[\pi_{x_1}, ..., \pi_{x_n}]$
  return $\mathcal{S}_*|_{\text{budget}}$

Algorithm 2 Overview of our Neural Data Server

Input: $\mathcal{S}$ (source), $\mathcal{T}$ (target), b (desired budget of data)
  $\{e_{\theta_i}\} \leftarrow$ MOE$(\mathcal{D}_\mathcal{S}, \mathcal{E}, K)$
  $z \leftarrow$ FASTADAPT$(\mathcal{T}, \{e_{\theta_i}\})$
  $\mathcal{S}_* \leftarrow$ OUTPUTDATA$(\mathcal{S}, z, b)$
  return $\mathcal{S}_*$
Output: $\mathcal{S}_* \in \mathcal{S}$ to download

Algorithm 3 Client's Module

procedure FASTADAPT$(\mathcal{D}_\mathcal{T}, \{e_{\theta_i}\})$:
  For $i = 1, ..., K$
    $z_i \leftarrow$ PERFORMANCE/FINET$(e_{\theta_i}, \mathcal{T})$ ▷ Sec 3.3.1
  return z

approach with the goal of training a centralized model on decentralized data over a large number of client devices, (*i.e.*, mobile phones). Our work shares a similar idea of restricting the visibility of data in a client-server model. However, in our case the representation of data is centralized (*dataserver*) and the clients exploit the transfer learning scenario for their own (decentralized) models.

Active and Curriculum Learning. In active learning [42] one searches over unlabeled data to find optimal samples to be labeled by an oracle, while in curriculum learning [7] subsets of data of increasing difficulty are sought for during training. In both scenarios, data search is performed at each iteration of training a particular model. Search is typically done by running inference on the data samples with the current snapshot of the model and selecting the examples based on uncertainty-based metrics. Our scenario differs in that we do not have the luxury of running inference with the *client*'s model on the massive amount of indexed data as this would induce a prohibitive computational overhead on the *dataserver* per *client*. Moreover, we do not assume *dataserver* to have access to the *client*'s model: this would entail the *client*s to share their inference code which many users may not be willing to do.

Learning to Generate Synthetic Images. Related to NDS are also [41, 28, 47, 36]. These approaches aim to bridge the synthetic vs real imagery gap by optimizing/searching over the set of parameters of a surrogate function that interfaces with a synthesizer.

In NDS, the search has to be done over massive (non-parametric) datasets and further, the target data cannot be sent to the server side. Our method is also significantly more computationally efficient.

3. Neural Data Server

Neural Data Server (NDS) is a search engine that aims to recommend transfer learning data. NDS consists of a *dataserver* which has access to a massive source dataset(s), and aims to suggest most relevant data samples to a *client*. A *client* is an end-user who wants a budget-constrained amount of data to improve the performance of her/his model in the target domain in a transfer learning scenario. We note that the *dataserver* does not host the data, and thus its recommendations are to be provided as a list of urls to data samples hosted by the original datasets' providers.

The *dataserver*'s indexed datasets may or may not be completely labeled, and the types of labels (*e.g.*, segmentation masks, detection boxes) across data samples may vary. The *client*'s target dataset is considered to only have a small set of labeled examples, where further the type of labels may or may not be the same as the labels in the *dataserver*'s dataset(s). The main challenge lies in requiring the *dataserver-client* transactions to have low computational overhead. As in any search engine that serves information to possibly numerous users, we want the online computation performed by the *dataserver* to be minimal. Thus we defer most of the computation to be performed on the *client*'s side, while still aiming for this process to be fast. Furthermore, the transactions should ideally be privacy preserving for the *client*, *i.e.*, the client's data nor the model's architecture are accessible, since the *client* may have sensitive information such as hospital records or secret tech. In NDS, we represent the *dataserver*'s data using a mixture-of-experts (MoE) trained on a self-supervised task. MoE

naturally partition the indexed datasets into different subsets and produce classifiers whose weights encode the representation of each of these subsets. The experts are trained offline and hosted on the *dataserver* for online transactions with the clients. In particular, the experts are sent to each client and used as a proxy to determine the importance of *dataserver*'s data samples for the *client*'s target domain.

To compute importance, the experts are fast-adapted on the client's dataset, and their accuracy is computed on a simple self-supervised task. We experimentally validate that the accuracy of each adapted expert indicates the usefulness of the data partition used to train the expert. The *dataserver* then uses these accuracies to construct the final list of data samples that are relevant for the *client*. Figure 3 provides an illustration while Algorithm 2 summarizes our NDS.

In Section 3.1 we formalize our problem. In Section 3.2 we describe how we train our mixture-of-experts model and analyze the different choices of representation learning algorithms for the experts (*dataserver* side). In Section 3.3.1 we propose how to exploit the experts' performance in the *client*'s target domain for data selection.

3.1. Problem Definition

Let $\mathbb{X}$ denote the input space (images in this paper), and $\mathbb{Y}_a$ a set of labels for a given task a. Generally, we will assume that multiple tasks are available, each associated with a different set of labels, and denote these by $\mathbb{Y}$. Consider also two different distributions over $\mathbb{X} \times \mathbb{Y}$, called the source domain $\mathcal{D}_s$ and target domain $\mathcal{D}_t$. Let $\mathcal{S}$ (*dataserver*) and $\mathcal{T}$ (*client*) be two sample sets drawn i.i.d from $\mathcal{D}_s$ and $\mathcal{D}_t$, respectively. We assume that $|\mathcal{S}| \gg |\mathcal{T}|$.

Our problem then relies on finding the subset $\mathcal{S}_* \in \mathcal{P}(S)$, where $\mathcal{P}(S)$ is the power set of $\mathcal{S}$, such that $\mathcal{S}_* \cup \mathcal{T}$ minimizes the risk of a model h on the target domain:

$$\mathcal{S}_* = \underset{\hat{\mathcal{S}} \in \mathcal{P}(S)}{\arg\min}\ \mathbb{E}_{(\boldsymbol{x}, \hat{\boldsymbol{y}}) \sim \mathcal{D}_t}[\mathcal{L}(h_{\hat{\mathcal{S}} \cup \mathcal{T}}(\boldsymbol{x}), \hat{\boldsymbol{y}})] \qquad (1)$$

Here, $h_{\hat{\mathcal{S}} \cup \mathcal{T}}$ indicates that h is trained on the union of data $\hat{\mathcal{S}}$ and $\mathcal{T}$. Intuitively, we are trying to find the subset of data from $\mathcal{S}$ that helps to improve the performance of the model on the target dataset. However, what makes our problem particularly challenging and unique is that we are restricting the visibility of the data between the *dataserver* and the *client*.

This means that fetching the whole sample set $\mathcal{S}$ is prohibitive for the client, as it is uploading its own dataset to the server. We tackle this problem by representing the *dataserver*'s indexed dataset(s) with a set of classifiers that are agnostic of the client (Section 3.2), and use these to optimize equation 1 on the *client*'s side (Section 3.3.1).

3.2. Dataserver

We now discuss our representation of the *dataserver*'s indexed datasets. This representation is pre-computed offline and stored on the *dataserver*.

3.2.1 Dataset Representation with Mixture-of-Experts

We represent the *dataserver*'s data $\mathcal{S}$ using the mixture-of-experts model [27]. In MoE, one makes a prediction as:

$$\boldsymbol{y}(\boldsymbol{x}) = \sum_{i=1}^{K} g_{\theta,i}(\boldsymbol{x}) e_{\theta_i}(\boldsymbol{x}) \qquad (2)$$

Here, $\boldsymbol{g}_\theta$ denotes a gating function ($\sum_{i=1}^{K} g_{\theta,i}(.) = 1$), e_{θ_i} denotes the i-th expert model with learnable weights θ_i, $\boldsymbol{x}$ an input image, and K corresponds to the number of experts. One can think of the gating function as softly assigning data points to each of the experts, which try to make the best guess on their assigned data points.

The MoE model is trained by using maximum-likelihood estimation (MLE) on an objective $\mathcal{L}$:

$$\theta = \arg\min_{\theta} \mathbb{E}_{(\boldsymbol{x}, \hat{\boldsymbol{y}}) \sim \mathcal{S}}[\mathcal{L}(\boldsymbol{y}(\boldsymbol{x}), \hat{\boldsymbol{y}})] \qquad (3)$$

We discuss the choices for the objective $\mathcal{L}$ in Sec 3.2.2, dealing with the fact that the labels across the source datasets may be defined for different tasks.

While the MoE objective allows end-to-end training, the computational cost of doing so on a massive dataset is extremely high, particularly when K is considerably large (we need to backpropagate gradients to every expert on every training example). A straightforward way to alleviate this issue is to associate each expert with a local cluster defined by a hard gating, as in [26, 22]. In practice, we define a gating function g that partitions the dataset into mutually exclusive subsets $\mathcal{S}_i$, and train one expert per subset. This makes training easy to parallelize as each expert is trained independently on its subset of data. Furthermore, this allows for new datasets to be easily added to the *dataserver* by training additional experts on them, and adding these to *dataserver*. This avoids re-training MoE over the full indexed set of datasets.

In our work, we use two simple partitioning schemes to determine the gating: (1) superclass partition, and (2) unsupervised partition. For superclass partition (1), we represent each class c in the source dataset as the mean of the image features f_c for category c, and perform k-means clustering over $\{f_c\}$. This gives a partitioning where each cluster is a superclass containing a subset of similar categories. This partitioning scheme only applies to datasets with class supervision. For unsupervised partitioning (2), we partition the source dataset using k-means clustering on the image features. In both cases, the image features are obtained from a pretrained neural network (*i.e.*, features extracted from the penultimate layer of a network pre-trained on ImageNet).

3.2.2 Training the Experts

We discuss two different scenarios to train the experts. In the simplified scenario, the tasks defined for both the *dataserver*'s and *client*'s datasets are the same, *e.g.*, classification. In this case, we simply train a classifier for the

task for each subset of the data in $\mathcal{S}$. We next discuss a more challenging case where the tasks across datasets differ.

Ideally, we would like to learn a representation that can generalize to a variety of downstream tasks and can therefore be used in a task agnostic fashion. To this end, we use a self-supervised method to train the MoE. In self-supervision, one leverages a simple surrogate task that can be used to learn a meaningful representation.

Furthermore, this does not require any labels to train the experts which means that the *dataserver*'s dataset may or may not be labeled beforehand. This is useful if the client desires to obtain raw data and label the relevant subset on its own. To be specific, we select classifying image rotation as the task for self-supervision as in [21], which showed this to be a simple yet powerful proxy for representation learning. Formally, given an image $\boldsymbol{x}$, we define its corresponding self-supervised label $\boldsymbol{y}$ by performing a set of geometric transformations $\{r(\boldsymbol{x}, j)\}_{j=0}^{3}$ on $\boldsymbol{x}$, where r is an image rotation operator, and j defines a particular rotation by one of the predefined angles, $\{0, 90, 180, 270\}$. We then minimize the following learning objective for the experts:

$$\mathcal{L}(\theta_i) = -\sum_{\boldsymbol{x}\in\mathcal{S}_i}\sum_{j=0}^{3}\log e_{\theta_i}(r(\boldsymbol{x}, j))_j \tag{4}$$

Here, index j in $e(.)_j$ denotes the output value for class j.

3.3. Dataserver-Client Transactions

In this section, we describe the transactions between the *dataserver* and *client* that determines the relevant subset of the server's data. The *client* first downloads the experts in order to measure their performance on the *client*'s dataset. If the tasks are similar, we perform a quick adaptation of the experts on the *client*'s side. Otherwise, we evaluate the performance of the experts on the *client*'s data using the surrogate task (i.e image rotation) (Section 3.3.1). The performance of each expert is sent back to the *dataserver*, which uses this information as a proxy to determine which data points are relevant to the *client* (Section 3.3.2). We describe these steps in more detail in the following subsections.

3.3.1 FastAdapt to a Target Dataset (on Client)

Single Task on Server and Client: We first discuss the case where the dataset task is the same for both the *client* and the *dataserver*, e.g., classification. While the task may be the same, the label set may not be (classes may differ across domains). An intuitive way to adapt the experts is to remove their classification head that was trained on the server, and learn a small decoder network on top of the experts's penultimate representations on the client's dataset, as in [50]. For classification tasks, we learn a simple linear layer on top of each pre-trained expert's representation for a few epochs. We then evaluate the target's task performance on a held-out validation set using the adapted experts. We denote the accuracy for each adapted expert $\hat{e}_{\theta_i}$ as z_i.

Diverse Tasks on Server and Client: To generalize to unseen tasks and be further able to handle cases where the labels are not available on the *client*'s side, we propose to evaluate the performance of the common self-supervised task used to train the experts on the *dataserver*'s data. Intuitively, if the expert performs well on the self-supervised task on the target dataset, then the data it was trained on is likely relevant for the *client*. Specifically, we use the self-supervised experts trained to learn image rotation, and evaluate the proxy task performance (accuracy) of predicting image rotation angles on the target images:

$$z_i = \frac{1}{4|\mathcal{T}|}\sum_{\boldsymbol{x}\in\mathcal{T}}\sum_{j=0}^{3}\mathbb{1}\big(\arg\max_k[e_{\theta_i}(\boldsymbol{r}(\boldsymbol{x}, j))_k] == j\big) \tag{5}$$

Here, index k in $e(.)_k$ denotes the output value for class k.

Note that in this case we do not adapt the experts on the target dataset (we only perform inference).

3.3.2 Data Selection (on Dataserver)

We now aim to assign a weighting to each of the data points in the source domain $\mathcal{S}$ to reflect how well the source data contributed to the transfer learning performance. The accuracies $\boldsymbol{z}$ from the client's FastAdapt step are normalized to $[0, 1]$ and fed into a *softmax* function with temperature $T = 0.1$. These are then used as importance weights w_i for estimating how relevant is the representation learned by a particular expert for the target task's performance. We leverage this information to weigh the individual data points $\boldsymbol{x}$. More specifically, each source data $\boldsymbol{x}$ is assigned a probabilistic weighting:

$$\pi(\boldsymbol{x}) = \sum_{i=1}^{K} w_i\, g_{\theta,i}(\boldsymbol{x})\frac{1}{|S_i|} \tag{6}$$

Here, $|S_i|$ represents the size of the subset that an expert e_{θ_i} was trained on. Intuitively, we are weighting the set of images associated to the i-th expert and uniformly sampling from it. We construct our dataset by sampling examples from $\mathcal{S}$ at a rate according to $\boldsymbol{\pi} = [\pi_{\boldsymbol{x}_1}, \pi_{\boldsymbol{x}_2}, \ldots, \pi_{\boldsymbol{x}_n}]^T$.

3.4. Relation to Domain Adaptation

If we assume that the client and server tasks are the same then our problem can be interpreted as domain adaptation in each of the subset $\hat{\mathcal{S}} \in \mathcal{P}(S)$ and the following generalization bound from [5] can be used:

$$\varepsilon_{\mathcal{T}}(h) < \varepsilon_{\hat{\mathcal{S}}}(h) + \frac{1}{2}d_{\mathcal{H}\Delta\mathcal{H}}(\hat{\mathcal{S}}, \mathcal{T}) \tag{7}$$

where ε represents the risk of a hypothesis function $h \in \mathcal{H}$ and $d_{\mathcal{H}\Delta\mathcal{H}}$ is the $\mathcal{H}\Delta\mathcal{H}$ divergence [5], which relies on the capacity of $\mathcal{H}$ to distinguish between data points from $\hat{\mathcal{S}}$ and $\mathcal{T}$, respectively.

Let us further assume that the risk of the hypothesis function h on any subset $\hat{\mathcal{S}}$ is similar such that: $\varepsilon_{\hat{\mathcal{S}}}(h) \approx \varepsilon(h) \quad \forall\hat{\mathcal{S}} \in \mathcal{P}(\mathcal{S})$. Under this assumption, minimizing

equation 1 is equivalent to finding the subset $\mathcal{S}_*$ that minimizes the divergence with respect to $\mathcal{T}$. Formally,

$$\mathcal{S}_* = \arg\min_{\hat{\mathcal{S}}} d_{\mathcal{H}\Delta\mathcal{H}(\hat{\mathcal{S}},\mathcal{T})} \tag{8}$$

In practice, it is hard to compute $d_{\mathcal{H}\Delta\mathcal{H}}$ and this is often approximated by a *proxy $\mathcal{A}$-distance* [6, 12, 19]. A classifier that discriminates between the two domains and whose risk ε is used to approximate the second part of the equation 7.

$$\hat{d}_{\mathcal{H}} \approx \hat{d}_{\mathcal{A}} \approx 2(1 - 2\varepsilon) \tag{9}$$

Note that doing so would require having access to $\mathcal{S}$ and $\mathcal{T}$ in at least one of the two sides (i.e to train the new discriminative classifier) and this is prohibitive in our scenario. In our case, we compute the domain confusion between $\hat{\mathcal{S}}$ and $\mathcal{T}$ by evaluating the performance of expert e_i on the target domain. We argue that this proxy task performance (or error rate) is an appropriate proxy distance that serves the same purpose but does not violate the data visibility condition. Intuitively, if the features learned on the subset cannot be discriminated from features on the target domain, the domain confusion is maximized. We empirically show the correlation between the domain classifier and our proposed proxy task performance in our experiments.

4. Experiments

We perform experiments in the tasks of classification, detection, and instance segmentation. We experiment with 3 datasets on the sever side and 7 on the client side.

4.1. Support for Diverse Clients and Tasks

In this section, we provide an extensive evaluation of our approach on three different client's scenarios: autonomous driving, fashion and general scenes. In each of them, the client's goal is to improve the performance of its downstream task (*i.e.*, object detection or instance segmentation) by pretraining in a budget-constrained amount of data. Here, the *dataserver* is the same and indexes the massive OpenImages [31] and MS-COCO [33] datasets. Specifically, our server dataset can be seen as the union of COCO and OpenImages [31, 33] (approx 538 GB) represented in the weights of the self-supervised trained experts (2 GB).

Autonomous Driving: Here, we use Cityscapes [14] as the client's dataset, which contains 5000 finely annotated images divided into 2975 training and 500 validation images. Eight object classes are provided with per-instance annotation. In practice, this simulates the scenario of a client that wants to crunch its performance numbers by pretraining on some data. This scenario is ubiquitous among state-of-the-art instance and semantic segmentation approaches on the Cityscapes leaderboard [45, 24, 52].

Fashion: We use the ModaNet dataset [51] to simulate a client that wants to improve its models' performance in the task of object detection of fashion related objects. ModaNet is a large-scale street fashion dataset consisting of 13 classes of objects and $55,176$ annotated images. Since the effectiveness of pre-training diminishes with the size of the dataset [23], we create a small version of the dataset for our experiments. This constitutes of 1000 training and 1000 validation images that are randomly selected but keeping the same class distribution of the original dataset. We call it miniModaNet in our experiments.

General Scenes: We use PASCAL VOC object detection [18] as the client's dataset for this scenario. The task in this case is object detection on 20 object classes. We use the `trainval2007` set containing 5011 images for training and evaluate on `test2007` containing 4962 images.

Evaluation: We use Intersection-Over-Union (IoU) to measure client's performance in its downstream task. Specifically, we follow the MS-COCO evaluation style and compute IoU at three different thresholds: a) 0.50, b) 0.75, c) an average of ten thresholds $(.5 : .05 : .95)$. The same evaluation style is used for both, object detection and instance segmentation. Notice however that in the case of instance segmentation, the overlap is based on segmented regions.

Baselines: In this regime, we compare our approach vs no pretraining, uniform sampling, and pretraining on the whole server dataset (*i.e.*, MS-COCO). In all cases, we initialize with ImageNet pretrained weights as they are widely available and this has become a common practice.

Implementation Details: Client. We use Mask-RCNN [24] with a ResNet50-FRN backbone detection head as the client's network. After obtaining a subset of $\mathcal{S}$, the client pre-trains a network on the selected subset and uses the pre-trained model as initialization for fine-tuning using the client (target) dataset. For object detection, we pre-train with a 681 class (80 class from COCO, 601 class from OpenImages) detection head using bounding box labels. For instance segmentation, we pre-train with 80 class (for COCO) or 350 class (for OpenImages) detection head using object mask labels. **Server.** For all self-supervised experts, we use *ResNet18* [25], and train our models to predict image rotations. MS-COCO and OpenImages are partitioned into $K = 6$ and $K = 50$ experts, respectively.

4.1.1 Qualitative and Quantitative Results

Object Detection: Table 1 reports the average precision at various IoU of the client's network pre-trained using data selected using different budgets and methods. First, we see that a general trend of pre-training the network on sampled detection data helps performance when fine-tuning on smaller client detection datasets compared to fine-tuning the network from ImageNet initialization. By pre-training on 90K images from COCO+OpenImages, we observe a 1-5% gain in AP at 0.5 IoU across all 3 client (target) datasets. This result is consistent with [32] which suggests that a pre-training task other than classification is beneficial for improving transfer performance on localization tasks. Next,

Pretrain Server Data (COCO + OpenImages)			Client Dataset								
Sampled Data Size		Method	PASCAL-VOC2007			miniModaNet			Cityscapes		
File Size	# Images		AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}
ImageNet Initialization			44.30	73.66	46.44	33.40	57.98	35.00	34.94	59.86	35.69
26GB / 538GB	90K (5%)	Uniform Sampling	47.61	76.88	51.95	35.64	58.40	39.09	36.49	61.88	36.36
		NDS	**48.36**	**76.91**	**52.53**	**38.84**	**61.23**	**43.86**	**38.46**	**63.79**	**39.59**
54GB / 538GB	180K (10%)	Uniform Sampling	48.05	77.17	52.04	35.78	58.50	39.71	36.41	61.22	37.17
		NDS	**50.28**	**78.61**	**55.47**	**38.97**	**61.32**	**42.93**	**40.07**	**65.85**	**41.14**

Table 1: Results for object detection on the 3 client datasets. Scores are measured in %.

Pretrain. Sel. Method		Target Dataset				
		Stanf. Dogs	Stanf. Cars	Oxford-IIIT Pets	Flowers 102	CUB200 Birds
0%	Random Init.	23.66	18.60	32.35	48.02	25.06
100%	Entire Dataset	64.66	52.92	79.12	84.14	56.99
20%	Uniform Sample	52.84	42.26	71.11	79.87	48.62
	NDS (SP+TS)	72.21	44.40	81.41	81.75	54.00
	NDS (SP+SS)	73.46	44.53	82.04	81.62	54.75
	NDS (UP+SS)	66.97	44.15	79.20	80.74	52.66
40%	Uniform Sample	59.43	47.18	75.96	82.58	52.74
	NDS (SP+TS)	68.66	50.67	80.76	83.31	58.84
	NDS (SP+SS)	69.97	51.40	81.52	83.27	57.25
	NDS (UP+SS)	67.16	49.52	79.69	83.51	57.44

Table 4: Ablation experiments on gating and expert training. SP=Superclass Partition, UP=Unsupervised Partition, TS=Task-Specific experts (experts trained on classif. labels), and SS=Self-Supervised experts (experts trained to predict image rotation).

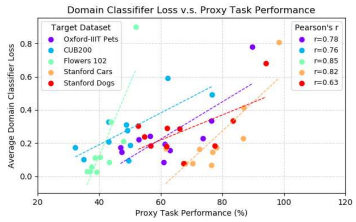

Figure 4: Relationship between domain classifier and proxy task performance on subsets $\hat{\mathcal{S}}$.

Data	Method	Oxford-IIIT Pet	CUB200 Birds
20%	Uniform Samp.	71.1	48.6
	KNN + [16]	74.4	51.6
	[37]	81.3	54.3
	NDS	**82.0**	**54.8**
40%	Uniform Samp.	76.0	52.7
	KNN + [16]	78.1	56.1
	[37]	81.0	**57.4**
	NDS	**81.5**	57.3
Entire ImageNet		79.1	57.0

Table 5: Transfer learning performance on classification datasets comparing data selection methods.

Data (# Images)	Method	AP^{bb}	AP^{bb}_{50}	AP	AP_{50}
0	ImageNet Initial.	36.2	62.3	32.0	57.6
23K	Uniform Sampling	38.1	64.9	34.3	60.0
	NDS	40.7	66.0	36.1	61.0
47K	Uniform Sampling	39.8	65.5	34.4	60.0
	NDS	**42.2**	**68.1**	**36.7**	**62.3**
59K	Uniform Sampling	39.5	64.9	34.9	60.4
	NDS	41.7	66.6	**36.7**	61.9
118K	Full COCO	41.8	66.5	36.5	**62.3**

Table 2: Transfer learning results for instance segmentation with Mask R-CNN on Cityscapes by selecting images from COCO.

Data (# Images)	Method	AP^{bb}	AP^{bb}_{50}	AP	AP_{50}
0	ImageNet Initial.	36.2	62.3	32.0	57.6
118K	Uniform Sampling	37.5	62.5	32.8	57.2
	NDS	**39.9**	**65.1**	**35.1**	**59.8**
200K	Uniform Sampling	37.8	63.1	32.9	57.8
	NDS	**40.7**	**65.8**	**36.1**	**61.2**

Table 3: Transfer learning results for instance segmentation with Mask R-CNN on Cityscapes by selecting images from OpenImages.

we see that under the same budget of 90K/180K images from the server, pre-training with data selected by NDS outperforms the baseline which uses images randomly sampled from $\mathcal{S}$ for all client datasets.

Instance Segmentation: Table 2 reports the instance segmentation performance by sampling 23K, 47K, and 59K images from COCO for pre-training on Cityscapes. We can see that pre-training using subsets selected by NDS is 2-3% better than the uniform sampling baseline. Furthermore, using 40% (47K/118K), or 50% (59K/118K) images from COCO yields comparable (or better) performance to using the entire 100% (118K) of data. Table 3 shows the results of sampling 118K, 200K images from OpenImages dataset as our server dataset.

Qualitative Results: Figure 6 shows qualitative results on MINIMODANET from detectors pre-trained from Imagenet, uniformly sampled images from $\mathcal{S}$, and images sampled using NDS. In the cases shown, the network pre-trained using the data recommended by NDS shows better localization ability, and is able to make more accurate predictions.

4.2. Support for Diverse Clients Same Task

For completeness, and in order to compare to stronger baselines that are limited to classification tasks, we also quantitatively evaluate the performance of NDS in the same-client-same-task regime. In this case, the task is set to be classification and the server indexes the Downsampled ImageNet [13] dataset. This a variant of ImageNet [17] resized to 32×32. In this case, we use $K = 10$ experts.

Client's Datasets: We experiment with several small classification datasets. Specifically, we use Stanford Dogs [29], Stanford Cars [30], Oxford-IIIT Pets [40], Flowers 102 [38], and CUB200 Birds [48] as client datasets.

Implementation Details: We use ResNet18 [25] as our client's network architecture, and an input size of 32×32 during training. Once subsets of server data are selected, we pre-train on the selected subset and evaluate the performance by fine-tuning on the client (target) datasets.

Comparison to data selection methods: Cui *et al.* [16] and Ngiam *et al.* [37] recently proposed data selection methods for improving transfer learning for classification tasks. In this restricted regime, we can compare to these methods. Specifically, we compare our NDS with [37], where they sample data based on the probability over

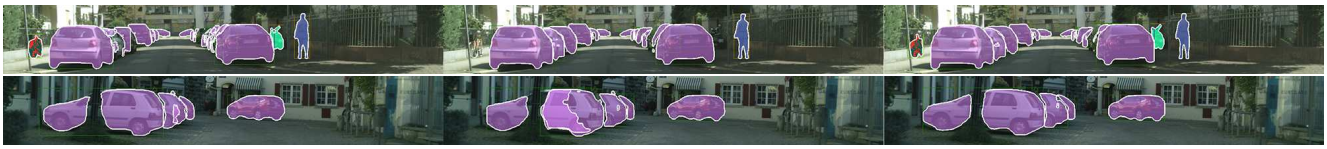

Figure 5: Instance segmentation results on Cityscapes using network pre-trained from ImageNet initialization (**left**), 47K images uniformly sampled (**middle**), and 47K images from NDS (**right**). Notice that the output segmentations generally look cleaner when training on NDS-recommended data.

.

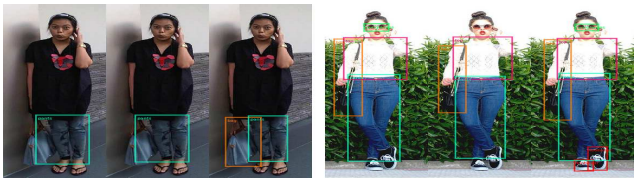

Figure 6: Object detection results in miniModaNet using network pre-trained from ImageNet initialization (**left**), 90K images uniformly sampled (**middle**), and 90K images sampled using NDS (**right**). A score threshold of 0.6 is used to display these images.

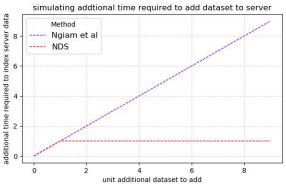

Figure 7: Simulating an incrementally growing *dataserver*, and the time required to "train" a model to represent the server. We NDS compare to the baseline of [37] (which is limited to classification tasks).

source dataset classes computed by pseudo-labeling the target dataset with a classifier trained on the source dataset. We also create a baseline KNN by adapting Cui *et al.*'s method [16]. Here, we sample from the most similar categories measured by the mean features of categories between the client and server data. We emphasize that the previous two approaches are limited to the classification task, and cannot handle diverse tasks. Furthermore, they do not scale to datasets beyond classification, and [37] does not scale to a growing *dataserver*. Our approach achieves comparable results to [37], and can be additionally applied to source datasets with no classification labels such as MS-COCO, or even datasets which are not labeled.

4.3. Ablation Experiments

Domain Confusion: To see how well the performance of the proxy task reflects the domain confusion, we perform an experiment comparing the proxy task performance and $\hat{d}_{\mathcal{A}}(\hat{\mathcal{S}}, \mathcal{T})$. To estimate $\hat{d}_{\mathcal{A}}$, we follow the same idea from [6, 12, 19] and for each subset $\hat{\mathcal{S}}$, we estimate the domain confusion. Figure 4 shows the domain confusion vs the proxy task performance using several classification datasets as the target (client) domain. In this plot, the highest average loss corresponds to the subset with the highest domain confusion (*i.e.*, $\mathcal{S}_i$ that is the most indistinguishable from the target domain). Notice that this correlates with the expert that gives the highest proxy task performance.

Ablation on gating and expert training: In Table 4, we compare different instantiations of our approach on five client classification datasets. For all instantiations, pre-training on our selected subset significantly outperforms the pre-training on a randomly selected subset of the same size. Our result in Table 4 shows that under the same super-class partition, the subsets obtained through sampling according to the transferability measured by self-supervised experts (SP+SS) yield a similar downstream performance compared to sampling according to the transferability measured by the task-specific experts (SP+TS). This suggests that self-supervised training for the experts can successfully be used as a proxy to decide which data points from the source dataset are most useful for the target dataset.

Scalability: Fig 7 analyzes the (simulated) required training time of the server as a new dataset is being incrementally added to it. We simulate a comparison between [37] (which needs to retrain the model on all datasets each time a dataset is added, and thus scales linearly) and NDS (where expert training is only ran on the additional dataset).

Limitations and Discussion: A limitation in our method is that the annotation quality/statistics in the *dataserver* datasets is not considered. This is shown in our instance segmentation experiment where the gains from pre-training on images sampled from OpenImages is smaller than pre-training on MS-COCO. This is likely due to the fact that MS-COCO has on average $\sim$7 instance annotations per image while OpenImages contains many images without mask annotations or at most $\sim$2 instance annotations per image. OpenImages has further been labeled semi-automatically and thus in many cases the annotations are noisy.

5. Conclusion

In this work, we propose a novel method that aims to optimally select subsets of data from a large dataserver given a particular target client. In particular, we represent the server's data with a mixture of experts trained on a simple self-supervised task. These are then used as a proxy to determine the most important subset of the data that the server should send to the client. We experimentally show that our method is general and can be applied to any pre-training and fine-tuning scheme and that our approach even handles the case where no labeled data is available (only raw data). We hope that our work opens a more effective way of performing transfer learning in the era of massive datasets. In the future, we aim to increase the capability of NDS to also support other modalities such as 3D, text and speech.

Acknowledgments: The authors acknowledge partial support by NSERC. SF acknowledges the Canada CIFAR AI Chair award at Vector Institute. We thanks Relu Patrascu for his continuous infrastructure support. We also thank Amlan Kar, Huan Ling and Jun Gao for early discussions, and Tianshi Cao and Jonah Philion for feedback in the manuscript.

References

[1] Sami Abu-El-Haija, Nisarg Kothari, Joonseok Lee, Apostol Natsev, George Toderici, Balakrishnan Varadarajan, and Sudheendra Vijayanarasimhan. Youtube-8m: A large-scale video classification benchmark. *ArXiv*, abs/1609.08675, 2016. 1

[2] Alessandro Achille, Michael Lam, Rahul Tewari, Avinash Ravichandran, Subhransu Maji, Charless C. Fowlkes, Stefano Soatto, and Pietro Perona. Task2vec: Task embedding for meta-learning. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6429–6438, 2019. 2

[3] David Acuna, Amlan Kar, and Sanja Fidler. Devil is in the edges: Learning semantic boundaries from noisy annotations. In *CVPR*, 2019. 2

[4] David Acuna, Huan Ling, Amlan Kar, and Sanja Fidler. Efficient interactive annotation of segmentation datasets with polygon-rnn++. In *CVPR*, 2018. 2

[5] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine Learning*, 79:151–175, 2009. 5

[6] Shai Ben-David, John Blitzer, Koby Crammer, and Fernando Pereira. Analysis of representations for domain adaptation. In B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *NeurIPS*, pages 137–144. MIT Press, 2007. 6, 8

[7] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009. 3

[8] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *ACM Conf. on Computer and Communications Security*, 2017. 2

[9] Sergi Caelles, Alberto Montes, Kevis-Kokitsi Maninis, Yuhua Chen, Luc Van Gool, Federico Perazzi, and Jordi Pont-Tuset. The 2018 davis challenge on video object segmentation. *ArXiv*, abs/1803.00557, 2018. 1

[10] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. *ArXiv*, abs/1903.11027, 2019. 1

[11] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *CoRR*, abs/1606.00915, 2016. 1

[12] Minmin Chen, Kilian Q Weinberger, Zhixiang Xu, and Fei Sha. Marginalizing stacked linear denoising autoencoders. *Journal of Machine Learning Research*, 16(1):3849–3875, 2015. 6, 8

[13] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets, 2017. 7

[14] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. *CVPR*, pages 3213–3223, 2016. 6

[15] Gabriela Csurka. Domain adaptation for visual applications: A comprehensive survey. *arXiv preprint arXiv:1702.05374*, 2017. 2

[16] Yin Cui, Yang Song, Chen Sun, Andrew Howard, and Serge J. Belongie. Large scale fine-grained categorization and domain-specific transfer learning. *CVPR*, pages 4109–4118, 2018. 2, 7, 8

[17] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*, 2009. 2, 7

[18] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John M. Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88:303–338, 2009. 6

[19] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor S. Lempitsky. Domain-adversarial training of neural networks. *J. Mach. Learn. Res.*, 17:59:1–59:35, 2015. 6, 8

[20] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *CVPR*, 2012. 1

[21] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. In *ICLR*, 2018. 5

[22] Sam Gross, Marc'Aurelio Ranzato, and Arthur Szlam. Hard mixtures of experts for large scale weakly supervised vision. In *CVPR*, pages 6865–6873, 2017. 4

[23] Kaiming He, Ross B. Girshick, and Piotr Dollár. Rethinking imagenet pre-training. *CoRR*, abs/1811.08883, 2018. 2, 6

[24] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask r-cnn. *ICCV*, pages 2980–2988, 2017. 1, 6

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CVPR*, pages 770–778, 2015. 6, 7

[26] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *ArXiv*, abs/1503.02531, 2015. 4

[27] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3:79–87, 1991. 4

[28] Amlan Kar, Aayush Prakash, Ming-Yu Liu, Eric Cameracci, Justin Yuan, Matt Rusiniak, David Acuna, Antonio Torralba, and Sanja Fidler. Meta-sim: Learning to generate synthetic datasets. In *ICCV*, 2019. 3

[29] Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Li Fei-Fei. Novel dataset for fine-grained image categorization. In *CVPR Workshop on Fine-Grained Visual Categorization*, Colorado Springs, CO, June 2011. 7

[30] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013. 7

[31] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Tom Duerig, and Vittorio Ferrari. The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. *arXiv:1811.00982*, 2018. 1, 6

[32] Hengduo Li, Bharat Singh, Mahyar Najibi, Zuxuan Wu, and Larry S. Davis. An analysis of pre-training on object detection. *ArXiv*, abs/1904.05871, 2019. 6

[33] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014. 1, 6

[34] Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. Exploring the limits of weakly supervised pretraining. In *ECCV*, pages 181–196, 2018. 2

[35] H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. Federated learning of deep networks using model averaging. *ArXiv*, abs/1602.05629, 2016. 2

[36] Bhairav Mehta, Manfred Diaz, Florian Golemo, Christopher J Pal, and Liam Paull. Active domain randomization. *arXiv preprint arXiv:1904.04762*, 2019. 3

[37] Jiquan Ngiam, Daiyi Peng, Vijay Vasudevan, Simon Kornblith, Quoc V. Le, and Ruoming Pang. Domain adaptive transfer learning with specialist models, 2018. 2, 7, 8

[38] M-E. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. In *Proc. of the Indian Conference on Computer Vision, Graphics and Image Processing*, Dec 2008. 7

[39] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Trans. on knowledge and data engineering*, 22(10):1345–1359, 2009. 2

[40] O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. V. Jawahar. Cats and dogs. In *CVPR*, 2012. 7

[41] Nataniel Ruiz, Samuel Schulter, and Manmohan Chandraker. Learning to simulate. *arXiv preprint arXiv:1810.02513*, 2018. 3

[42] Burr Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009. 3

[43] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *PAMI*, 39(4):640–651, Apr. 2017. 1

[44] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *ICCV*, pages 843–852, 2017. 2

[45] Towaki Takikawa, David Acuna, Varun Jampani, and Sanja Fidler. Gated-scnn: Gated shape cnns for semantic segmentation. *ArXiv*, abs/1907.05740, 2019. 6

[46] Jonathan Tremblay, Aayush Prakash, David Acuna, Mark Brophy, Varun Jampani, Cem Anil, Thang To, Eric Cameracci, Shaad Boochoon, and Stan Birchfield. Training deep networks with synthetic data: Bridging the reality gap by domain randomization. In *CVPR Workshop*, pages 969–977, 2018. 2

[47] Shashank Tripathi, Siddhartha Chandra, Amit Agrawal, Ambrish Tyagi, James M Rehg, and Visesh Chari. Learning to generate synthetic data via compositing. In *CVPR*, pages 461–470, 2019. 3

[48] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011. 7

[49] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *NeurIPS*, 2014. 2

[50] Amir Roshan Zamir, Alexander Sax, William B. Shen, Leonidas J. Guibas, Jagannath Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. *CVPR*, pages 3712–3722, 2018. 2, 5

[51] Shuai Zheng, Fan Yang, M. Hadi Kiapour, and Robinson Piramuthu. Modanet: A large-scale street fashion dataset with polygon annotations. In *ACM Multimedia*, 2018. 6

[52] Yi Zhu, Karan Sapra, Fitsum A. Reda, Kevin J. Shih, Shawn D. Newsam, Andrew Tao, and Bryan Catanzaro. Improving semantic segmentation via video propagation and label relaxation. In *CVPR*, 2018. 6