# A Survey of Transfer Learning and Analysis of Various State-of-the-Art Algorithms

Connor Baumler, Nikhil Chakravarthy, Noah Crowley, Kha Dinh Luong, Toshiki Nazikian, Vimig Socrates

## I. SURVEY

### A. Introduction

The importance of data cannot be understated when developing learning systems. More specifically, when machine learning models are trained, often they use a human-curated class label to determine their accuracy in training and testing. However, the generation of this data is often difficult and expensive. As such, a new field of study has been developed that attempts to *transfer* knowledge from a particular problem to another. There are a number of different types of transfer learning, as well as different approaches depending on the settings provided. For instance, transfer learning can be conducted between different domains, tasks, and with or without labelled data in the source and target domains. The following review will describe a survey of the current literature, specifically focusing on 12 particular algorithms, 6 of which have been implemented and tested.

The six implemented algorithms are structural correspondence learning, self-taught clustering, self-taught learning, feature-space remapping, warm start, sample-transfer zero-shot learning.

### B. Inductive Transfer Learning

Inductive transfer learning is when the target and source task differ, but the target and source domains may or may not be the same. Additionally, the target domain labels should always be present. If the source domain has labels, then this becomes similar multi-task learning. If the source domain does not have labels, then it is a case of self-taught learning.

*1) Multi-task Learning Comparison:* Multi-task learning seeks to learn the target and source task simultaneously, whereas this case of inductive learning seeks to transfer the knowledge from the source task to the target task.

*2) Self-taught Learning Comparison:* For self-taught learning, there is no guarantee that the labels of the source and target domains occupy the same domain space since the source labels are unavailable. The transfer learning algorithm has to find features in the unlabeled source domain that can be used to help correctly label the target domain data.

### C. Transductive Transfer Learning

Contrary to inductive transfer learning, where target domain labels are always available, in transductive transfer learning, the target domain labels are missing. However, we ensure that we choose a source domain that does contain labels to transfer knowledge from. The lack of target domain labels makes this problem more complicated than the previous case. In transductive learning, the feature spaces could potentially be different between source and target ($X_S \neq X_T$). Conversely, the feature spaces could be the same, but the marginal conditional probabilities are different, (i.e. $X_S = X_T, P(X_S) \neq P(X_T)$).

### D. Unsupervised Transfer Learning

In the case of unsupervised transfer learning, there are no labels either in the source nor the target domains. As such, this setting for transfer learning is the most difficult, as well as the newest [16]. In this case, the algorithms commonly used in unsupervised learning, such as clustering, dimensionality reduction, and density estimation are adapted to include the transferred source domain to adapt it to the target domain. Generally, the learning tasks are the same as well. Two specific algorithms transferred dimensionality reduction and self-taught clustering are discussed in this paper.

## II. DESCRIPTION OF ALGORITHMS

### A. Multi-task Feature Learning[2]

Training a new model for a new learning task is time consuming. Many times, a new learning task is not much different from tasks we have already learned. The key idea behind this algorithm is that, given a set of similar learning tasks, we can learn them at the same time as a single training task. The algorithm accomplishes that by learning a common representation dictionary for all the common tasks' features.

Assuming there are $T$ learning tasks, each with an associated concept function $f_t$, then, assuming that the tasks share some common features, each function $f_t$ can be written as:

$$f_t(x) = \sum_{i=1}^{d} a_{it} h_i(x),\ t \in \mathbb{N}_T$$

where $d$ is the dimension of the examples, $h_i$ are the features, and $a_{it}$ are the regression parameters of the features. The main assumption is that each task function is a linear combination of the features, in which most of the features have zero regression parameters. As a result, the algorithm will learn a sparse dictionary of the features. The authors considered only linear features so each feature $h_i$, in turn, can be written as:

$$h_i(x) = \langle u_i, x \rangle$$

The authors then defined the dictionary learning task as a minimization problem of the following loss function:

$$\mathcal{E}(A, U) = \sum_{t=1}^{T} \sum_{i=1}^{m} L(y_{ti}, \langle a_t, U^T x_{ti} \rangle) + \gamma \|A\|_{2,1}^2$$

where $A$ is a $d \times T$ matrix of regression parameters $a_{it}$ and $U$ is a $d \times d$ matrix of entries $u_i$. However, the above problem is non-convex and the $2,1-norm$ of matrix $A$ is not continuous. If we let $w_t = \sum_i a_{it} u_i$, then the original function $f_t$ can be rewritten as:

$$f_t(x) = \langle w_t, x \rangle$$

With this result, after a series of algebraic transformation and proof, the authors were able to transform the initial non-convex minimization problem into an equivalent convex minimization problem of the function:

$$\mathcal{R}(W, D) = \sum_{t=1}^{T} \sum_{i=1}^{m} L(y_{ti}, \langle w_t, x_{ti} \rangle) + \gamma \sum_{t=1}^{T} \langle w_t, D^+ w_t \rangle$$

where $D$ is a positive semi-definite matrix of size $d$ and $D^+$ is the pseudo-inverse of $D$. Solving the above optimization is, however, non-trivial. Another major contribution of the paper was coming up with an algorithm to iteratively solve the problem. The algorithm is algebraically complicated that it is not suitable to explain in details in the context of a brief overview. We have actually attempted to re-implement the algorithm, however, our programming was not able to run as efficiently and correctly as we want and thus will not be included in this report.

### B. *Self-Taught Learning[17]*

The main assumption behind this Self-taught Learning (STL) is similar tasks share some common representations. Even tasks that are not apparently related may have some similar features. As the result, even the representations learned from a collection of disorganized unlabeled data may be useful for later supervised learning tasks.

To illustrate the idea, let us consider a classifier trying to learn the concept $Lion$. The classifier is expected to be able to learn features that have great correlation with $Lion$ such as $SharpTeeth$, $HorizontalBody$, $FourLegs$, and $OneTails$. Even though the learner has never seen a $Lion$, it is still able to extract these features if it has seen a lot of animals, some of which have $SharpTeeth$, $HorizontalBody$, $FourLegs$, and $OneTails$. When the learner sees a $Lion$, it can fit the $Lion$ into the feature space that it is familiar with. The important point is, when observing other animals, the learner need not know the labels of those animals, because the focus is not to identify what the animals are, but to identify the features.

The idea consists of 2 steps. In the first step, given an unlabeled dataset, or multiple datasets, the learner must, in an unsupervised manner, identify the important features within those examples. In the second steps, the learner must apply its knowledge about the features to learn an supervised learning problem.

The authors translated this idea into an algorithm that first learn a sparse dictionary of features from a set of unlabeled source domain data. After that, the target domain examples are represented using this dictionary via sparse encoding. The resulting encoding is used to represent the labeled examples when learning a classifier. The algorithm steps are as the followings.

*a) Representation learning step:* Learn a sparse dictionary of the unlabeled data by solving the optimization problem:

$$\text{minimize}_{b,a} \sum_i \|x_u^{(i)} - \sum_j a_j^{(i)} b_j\|_2^2 + \beta \|a^{(i)}\|_1$$
$$\text{s.t. } \|b_j\|_2 \leq 1,\ \forall j \in 1, \ldots s$$

where $x$ is an example and the subscript $u$ is for unlabeled, $b$ is a set of basis vectors and $a$ is a set of corresponding activations for the vectors. Each unlabeled example is therefore encoded as a linear combination of the learned basis vectors. This first step returns the sets of basis vectors and activations.

*b) Transferring of representation step:* In this step, the labeled data is transformed using the representation learned in step 1, i.e. , produce an encoding for each labeled example from the set of basis vectors and activations we obtained. To do this, the model solve the next optimization problem:

$$\hat{a}(x_l^{(i)}) = \arg\min_{a^i} \|x_l^{(i)} - \sum_j a_j^{(i)} b_j\|_2^2 + \beta \|a^{(i)}\|_1$$

where $x$ is an example and the subscript $l$ is for labeled. $\hat{a}(x_l^{(i)})$ is the set of activations for a particular example, and will be used to represent that example during later learning task.

*c) Learn a classifier :* In this step, the model learn a classifier for the labeled examples based on their new representations. Any classification algorithm can do the job. In our implementation, we chose SVM as our base learner for the algorithm.

## *C. Sample Transfer Zero-Shot Learning[11]*

*1) Zero-Shot Learning:* We read two zero-shot learning (ZSL) papers and implemented one *(STZSL, described below)*. ZSL is a sub-domain of transfer learning. In ZSL, there is an abundance of labelled data on a source domain from which an algorithm can learn to perform well on a source task. ZSL is then used to learn about a new target task by using knowledge about the relationship between the source task and the target task. As the authors point out, humans have the capability to take knowledge about one class of things, be given knowledge about that class and its relation to another class, and immediately have an idea of the second class. A solid example of this could be a child who understands what a baseball looks like and is then told that tennis balls are like baseballs, except they're fuzzy and usually green instead of white. Most likely, if a child were to remember this information, they would be able to immediately recognize the first tennis ball they encounter as a tennis ball. The purpose of ZSL algorithms is to replicate just this kind of learning ability.

It is important to note that ZSL itself is not limited to any one of the main domains of transfer learning. ZSL algorithms can be applied to a variety of settings.

*2) STZSL Description:* This particular algorithm is called "Sample Transfer Zero-Shot Learning" (STZSL), named as such since it tries to take samples from both the source domain and the target domain and assigns pseudo-labels to train on for the target task. The algorithm itself can be trained using almost any one-vs-all classifier, however the authors specifically used an SVM and even developed an extension of the standard linear kernel function for use with a LIBSVM classifier for use with this algorithm.

STZSL assumes a set of labelled source data, a set of source classes, a set of target classes, and some form of auxiliary data that can be used as relational information between the classes. The algorithm is capable of using unlabelled target data, but does not require it. This results in two distinct versions of the STZSL algorithm — with unlabelled target data and without, labelled as STZSL-Transductive and STZSL-Inductive, respectively.

The authors' use of the terms "inductive" and "transductive" does not appear to meet the descriptions of our seed paper, as the inductive version of the algorithm fails to meet the seed paper's requirement of some amount of labelled training data and the transductive version also seems to miss some key tenets of the seed paper's definitions as it is trying to learn to recognize target classes that are separate from the source classes, which would likely classify as a new target task. In order to avoid confusion, we will reduce the use of the terms "inductive" and "transductive". Instead, for the case wherein unlabelled target data are made available will simply be referred to as STZSL-T and for the case wherein no unlabelled target data are available will be referred to as STZSL-I, which are acronyms used by the authors.

Many ZSL algorithms that have been developed have used what the STZSL authors call a two-step recognition strategy. This two-step recognition strategy is defined as the steps of projecting a test sample into an intermediary space and then doing the classification on the intermediary space. The authors' argue that such a method of projection must lead to a loss of information. Instead, STZSL aims to carry out ZSL in a one-step process by transferring samples within the same feature space and training classifiers on that original feature space. The relationship information between the classes is used to determine which samples to transfer rather than as a mode of projection.

*3) Notation:* It is necessary to explain some of the most common notation used throughout this paper. If there are $k^s$ source classes, then the set of source classes is denoted $C^s = \{c_1^s, \ldots, c_{k^s}^s\}$. The number of labelled source samples is denoted $n^s$. The $i^{th}$ source sample is denoted $\mathbf{x}_i^s \in \mathbb{R}^d$, where $d$ is the dimensionality of the feature space. Similarly, the $i^{th}$

source sample's label is denoted $\mathbf{y}_i^s \in \{0,1\}^{k^s}$ such that $y_{ij}^s = 1$ if the sample $i$ belongs to class $c_j^s$.

Similar to the source data, if there are $k^t$ target classes, then the set of target classes is denoted $C^t = \{c_1^t, \ldots, c_{k^t}^t\}$. STZSL assumes that $C^s \cap C^t = \emptyset$. Also, $n^t$ denotes the number of target samples and target samples are denoted as $\mathbf{x}_i^t \in \mathbb{R}^d$. Note that all target samples are unlabelled, so $\mathbf{y}_i^t$ would be unknown.

Additionally, $\tilde{\mathbf{y}}_i^t$ may be used to denote a pseudo-label for a target sample $i$, and similarly $\tilde{\mathbf{y}}_i^s$ may be used to denote a pseudo-label for a source sample $i$. Often the term $\sigma$ will appear and this value will be set to the average Euclidean distance between all samples. Finally, $c(\mathbf{x_i^s})$ may be used to denote whatever class the $i^{th}$ source sample is labelled as and $\tilde{c}(\mathbf{x_i^t})$ may be used to denote the estimated class for the $i^{th}$ target sample.

*4) STZSL Proposed Framework:* The goal of STZSL is to take samples from one class, or an unknown class, and to transfer them into a pseudo label that belongs to the set of target classes. The framework for this algorithm depends on this key premise. There are two key parts of this algorithm. The first is to select source samples and, if available, target samples to be transferred to a target label. The second is to use a modified, or "robust", SVM that is trained using the transferred labels. The transfer selection uses a process of finding a transferability and then ranking based on similarity. The robust SVM also uses these transferability probabilities to modify the standard linear kernel function.

Like STZSL's ability to work with or without target data, the sample transfer step can be done with either a set of embedding vectors or with a class similarity matrix. Either of these forms of input are used as a type of prior knowledge to assist in transferring knowledge from the labelled source data to the target labels. Depending on which of the two forms of prior information is given, the sample transfer step will behave slightly differently. In either case, though, the goal of this step is to identify which samples are most transferable to the target class based on the prior information. This is called the transferability of a sample to a given target class. The intuition behind this is summarized well by the authors when they describe the idea of a picture of a man sitting on a horse. While humans are hardly similar to horses from a holistic external perspective, a picture of a man sitting on a horse could easily be classified as either a horse or a man. As such, it is easy to imagine that this image could be transferred to either label.

*a) Transferability from embedding vectors:* If given a set of embedding vectors of dimensionality $q$, denoted $\mathbf{a}_c \in \mathbb{R}^q$, STZSL creates an embedding space where the transferability is computed. In order to do this, STZSL looks to find the optimal functions $\varphi$ and $\psi$ by the following function:

$$(\varphi, \psi) = \underset{(\varphi,\psi)}{\arg\min} \sum_{i=1}^{n^s} \mathcal{L}\big(\varphi(\mathbf{x}_i^s), \psi(\mathbf{a}_{c(\mathbf{x}_i^s)}\big) + \sum_{j=1}^{n^t} \alpha_j \mathcal{L}\big(\varphi(\mathbf{x}_j^t), \psi(\mathbf{a}_{\tilde{c}(\mathbf{x}_j^t)})\big) \tag{1}$$

Where $\mathcal{L}$ is a loss function and $\alpha_j$ is a weight for the $j^{th}$ target sample. At each step, $\alpha_j$ is set to the previous step's computed transferability for that example, and during the initial step $\alpha_j = 1 \ \forall j$. Note that for STZSL-I, where there are no target samples provided, the second summation is dropped entirely as $n^t = 0$; also for STZSL-I, there is only one iteration. The authors of this paper decided upon using a linear projection for $\varphi$, the identity function for $\psi$, and squared Euclidean distance for $\mathcal{L}$, and so we have followed suit.

In this paper, and in our implementation, $\mathbf{P}$ is defined as a linear projection matrix in $\mathbb{R}^{dxq}$ for $\varphi$ such that:

$$\mathbf{P} = (\mathbf{X}^T\mathbf{X} + \epsilon\mathbf{I}_d)^{-1}\mathbf{X}^T\mathbf{A} \tag{2}$$

Where $\mathbf{X} = [\mathbf{x}_1^s, \ldots, \mathbf{x}_{n^s}^s, \alpha_1\mathbf{x}_1^t, \ldots, \alpha_{n^t}\mathbf{x}_{n^t}^t]$, $\mathbf{A} = [\mathbf{a}_{c(\mathbf{x}_1^s)}, \ldots, \mathbf{a}_{c(\mathbf{x}_{n^s}^s)}, \alpha_1\mathbf{a}_{c(\mathbf{x}_1^t)}, \ldots, \mathbf{a}_{c(\alpha_{n^t}\mathbf{x}_{n^t}^t)}]$, and $\epsilon$ is set to a small real value such as $10^{-4}$. Then the transferability that sample $\mathbf{x}_i$, source or target, to target class $c_j^t$ is set to:

$$p_i^j = \mathcal{N}(\mathbf{x}_i\mathbf{P}|\mathbf{a}_{c_j^t}, \sigma^2\mathbf{I}) \tag{3}$$

*b) Transferability from class similarity graph:* If instead of embedding vectors a class similarity graph is given, then some base one-vs-all classifiers are generated for each class, denoted as $f^c$. For $c \in C^s$, $f^c$ is created by creating a one-vs-all standard linear SVM based on LIBSVM. In our implementation, Scikit Learn's SVC was used. For $c \in C^t$, $f^c$ is created using the robust SVM *(detailed later on)* that are trained in the initial *(and for STZSL-I, only)* step by finding a set of source samples that can be transferred to target class $c$ for each $c \in C^t$. Then each example $\mathbf{x}_i$ from the source or target samples is run through all the classifiers and given an output of $o_i^c = f^c(\mathbf{x}_i)$.

Using the outputs from each base classifier, the transferability of each sample to each target class is computed in two parts. First, $\tilde{p}_i^c$ is computed like so:

$$\tilde{p}_i^c = \frac{\exp o_i^c}{\sum_{c^{\prime}} \exp o_i^{c^{\prime}}} \tag{4}$$

Which is then used to compute the actual transferability of example $i$ to class $j$, $p_i^j$, like so:

$$p_i^j = \frac{1}{Z} \sum_c p_i^c g_{jc} \tag{5}$$

Where $Z$ is a regularization constant set such that $\sum_j p_i^j = 1$ and $g_{jc}$ is the entry in the similarity matrix that gives the similarity between class $c$ and the target class $c_j^t$.

*c) Transferring samples:* Using the transferability from either the embedding space or the class similarity graph, STZSL then ranks samples for each target class and picks the $m$ top-ranked samples for transfer. When transferring source samples, $m = m^s$; similarly, when transferring target samples, $m = m^t$. In our implementation, $m^s = \frac{n^s}{5}$ and $m^t = \frac{n^t}{k^t}$.

These transferability rankings values are assigned to a vector $\mathbf{r}_j \in \mathbb{R}^n$, where $n$ is either $n^s$ or $n^t$ depending on which set of samples are being ranked. To find $\mathbf{r}$, an iterative procedure is employed to solve the following optimization problem:

$$\mathbf{r} = \arg\min_{\mathbf{r}} \frac{\beta}{2} \mathbf{r}\mathbf{K}\mathbf{r}^T - \mathbf{r}\mathbf{p}^T \tag{6}$$

Where $\beta$ is a hyperparameter and such that:

$$\mathbf{r}\mathbf{1}_n^T = \rho,\ r_i \geq 0$$
$$K_{ij} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2)}{\sigma^2}$$
$$\mathbf{p} = [p_1^c, \ldots, p_n^c]$$

This iterative procedure is defined in **Algorithm 1** by the authors and is implemented in the function *_compute_transferability_ranking_scores* in *stzsl.py*. To summarize for the sake of brevity, the optimization is a QP problem, but to improve time efficiency the authors detailed an algorithm based on a Lagrangian multiplier.

*d) Robust SVM kernel function:* While the author's claim that STZSL can work with any supervised classifier, the SVM was chosen for its generalization ability. However, since SVMs can also be particularly disturbed by noisy data and transferred samples are likely to be especially noisy, the authors developed a slight change to the standard linear kernel matrix. The SVM using this kernel matrix is dubbed the "robust SVM." This new kernel function is defined as $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbb{E}_\epsilon[M_{ij}] < \mathbf{x}_i, \mathbf{x}_j >$.

$\mathbb{E}_\epsilon[M_{ij}]$ depends on $\theta_i$ which depends on $l_i^c$, which is defined to have the value 1 if sample $\mathbf{x}_i$ was transferred to the target class $c$ and 0 otherwise. Then $\theta_i$ is defined as:

$$\theta_i = \begin{cases} (1 = \exp(\delta p_i^c))^{-1}, & \text{if } l_i^c = 1 \\ 0, & \text{otherwise} \end{cases}$$

Where $\delta$ is set to a 5 as per the author's decision. Then $M_{ij}$ is defined as $(1 - 2\epsilon_i)^2$ where $\epsilon_i$ is a binary random variable with $p(\epsilon_i = 1) = \theta_i$ and $\epsilon_i = 1$ indicates that a sample $i$ was transferred. So:

$$\mathbb{E}_\epsilon[M_{ij}] = \begin{cases} 1, & \text{if } i = j \\ 1 - 2\theta_i - 2\theta_j + 4\theta_i\theta_j, & \text{otherwise} \end{cases} \tag{7}$$

After assigning pseudo labels to a set of samples, a one-vs-all robust SVM is trained for the target class. If in the STZSL-T setting, this is repeated in an iterative process. The authors do not define how to set the termination condition, so we decided to run the iteration five times for STZSL-T for runtime's sake.

### *D. Semi-Supervised Zero-Shot Classification with Label Representation[14]*

We also read about a second ZSL algorithm — Semi-Supervised Zero-Shot Classification with Label Representation Learning (ZSWLR for zero-shot with representation learning). This algorithm is developed in order to learn a max-margin, multi-class zero-shot classification model that is trained with both observed and unobserved labels.

*1) Notation:* All classes belong to the set $\mathcal{Y} = \{1, \ldots, K\}$. All observed labels belong to the first $K^\ell$ classes and all unobserved labels belong to the last $K^u$ classes s.t. $K = K^\ell + K^u$. All $t$ examples are row vectors in the matrix $\mathbf{X} \in \mathbb{R}^{t \times d}$ and each $i^{th}$ example's label is kept in an indicator row vector in the corresponding $i^{th}$ row of $\mathbf{Y} \in \{0,1\}^{t \times K}$. The first $t^\ell$ examples in $\mathbf{X}$ are assumed to be labelled in $\mathbf{Y}$ whereas the last $t^u$ examples are assumed to have unknown labels s.t. $t = t^\ell + t^u$. As such, $\mathbf{Y}^\ell$ is the matrix containing the first $t^\ell$ rows of $\mathbf{Y}$ and each row of $\mathbf{Y}^\ell$ is assumed to have a 1 in one component and zeroes elsewhere.

*2) ZSWLR Proposed Framework:* The ZSWLR algorithm is aimed at solving for the optimal values for $\mathbf{Y}^u$ according to the following formulation:

$$\underset{\mathbf{Y}^u \in \mathcal{Q}, \mathbf{W}}{\arg\min} \sum_{i=1}^{t} \mathcal{L}\big(f(\mathbf{X_i}, \mathbf{W}), \mathbf{Y_i}\big) + \frac{\alpha}{2}\|\mathbf{W}\|_F^2 + \frac{\rho}{2} tr(\mathbf{Y}^{u\intercal}\mathbf{L}^u\mathbf{Y}^u\big) \tag{8}$$

The authors define this s.t $f(\cdot,\cdot)$ is a prediction function using the model parameter matrix $\mathbf{W}$, $\mathcal{L}(\cdot,\cdot)$ is a convex loss function, $Q$ is the feasible set for $\mathbf{Y}^u$, and $L^u \in \mathbb{R} t_u \times t_u$ is a Laplacian matrix s.t. $L^u = diag(\mathbf{A1}) - \mathbf{A}$ for a similarity matrix $\mathbf{A} \in \mathbb{R}^{t_u \times t_u}$ and with $\mathbf{1}$ being a column vector with ones in every component.

Because this algorithm is in the class of ZSL, it does not have any labelled data for the target classes. To compensate, the algorithm uses embedding vectors in a $\sqsubseteq$ dimensional space. However, unlike ZSTSL and other ZSL algorithms, ZSWLR tries to combine learning the embedding matrix with training a classifier. They use a label embedding matrix $\mathbf{M} \in \mathbb{R}^{K \times \sqsubseteq}$ which leads to the following formulation:

$$\underset{\mathbf{Y}^u \in \mathcal{Q}, \mathbf{M}, \mathbf{W}}{\arg\min} \sum_{i=1}^{t} \mathcal{L}\big(f(\mathbf{X_i}, \mathbf{W}), \mathbf{Y_i}\mathbf{M}\big) + \frac{\alpha}{2}\|\mathbf{W}\|_F^2 + \frac{\beta}{2}\|\mathbf{M} - \mathbf{M}^0\|_F^2 + \frac{\rho}{2} tr(\mathbf{Y}^{u\intercal}\mathbf{L}^u\mathbf{Y}^u\big) \tag{9}$$

Where $\mathbf{M}^0$ is a provided prior for $\mathbf{M}$ or a matrix of zeroes if none was provided.

In addition to training a classifier and learning an embedding matrix, ZSWLR also implements a loss function that maximizes the margin. The reasoning behind this is that other common loss functions are not considered very ready to handle the semantic categories that ZSWLR was developed for. To do this, the authors added a multi-class hinge loss based function that scores examples across all classes like so:

$$s(\mathbf{x}^\intercal, \mathbf{1}_k^\intercal) = \mathbf{x}^\intercal \mathbf{W} \mathbf{M}^\intercal \mathbf{1}_k \tag{10}$$

Where $\mathbf{1}_k$ is a column vector such that it has all zeroes except in its $k^{th}$ entry, which is a one. This scoring function is then used for the loss function $\mathcal{L}(\cdot,\cdot)$:

$$\begin{aligned}&\mathcal{L}\big(f(\mathbf{X_i}, \mathbf{W}), \mathbf{Y_i}\mathbf{M}\big) \\ &= \underset{k \in \mathcal{Y}}{\arg\max} \big(1 - \mathbf{Y_k}\mathbf{1}_k + s(\mathbf{X_i}, \mathbf{1}_k^\intercal) - s(\mathbf{X_i}, \mathbf{Y_i})\big)_+\end{aligned} \tag{11}$$

Where the operator $(\cdot)_+ = max(\cdot, 0)$ so that the loss function is always non-negative.

This algorithm is further completed by defining the feasible space for $\mathbf{Y}^u$, $Q$. The first obvious constraint is that the first $K^\ell$ columns of $\mathbf{Y}^u$ should be zero, and also that the final $\mathbf{Y}^u$ should have one entry that is equal to one. However, the authors also add a term for class balance to avoid having classes that take too large of a portion of the unlabeled examples. This gives the authors the following final constraint set:

$$Q = \left\{ \begin{array}{c} \mathbf{Y}^u \in \{0,1\}^{t_u \times K}, \mathbf{Y}^u\mathbf{S} = \mathbf{0}, \mathbf{Y}^u\mathbf{1} = \mathbf{1}, \\ a\mathbf{1}^\intercal \leq \mathbf{1}^\intercal(\mathbf{Y}^u\bar{\mathbf{S}}) \leq b\mathbf{1}^\intercal \end{array} \right\}$$

Where $\mathbf{S}$ is a matrix such that $\mathbf{Y}^u\mathbf{S}$ gives the first $K^\ell$ columns of $\mathbf{Y}^u$ and $\bar{\mathbf{S}}$ is a matrix such that $\mathbf{Y}^u\bar{\mathbf{S}}$ gives the last $K^u$ columns of $\mathbf{Y}^u$, and $a$ and $b$ are just some scalar parameters to control class balancing.

Using all of this, the authors then describe how they used the convexity per matrix to iteratively train to solve the joint training problem.

### *E. SVD-based Alternating Structure Optimization[1]*

SVD-ASO looks at predictors for different problems to find an underlying predictor space which is analyzed to find common structures.

Given a target supervised prediction problem, multiple other prediction problems are created from unlabeled data which are called auxiliary problems. Then, a structural parameter $\theta$ is learned by using joint empirical risk minimization on the auxiliary problems. For the purposes of this paper, joint empirical risk minimization can be defined as the optimization problem

$$[\hat{\theta}, \{\hat{f}_l\}] = \underset{\theta \in \Gamma, \{f_l \in \mathcal{H}_{l,\theta}\}}{\arg\min} \sum_{l=1}^{m} \frac{1}{n_l} \sum_{i=1}^{n_l} L(f_l(\mathbf{X}_i^l), Y_i^l$$

where $f_l$ are the predictors and $L(p, y)$ is a modification of Huber's robust loss function defined as

$$L(p,y) = \begin{cases} \max(0, 1 - py)^2 & \text{if } py \geq -1 \\ -4py & \text{otherwise} \end{cases}$$

Since $\hat{\theta}$ is leaned by the optimization

$$\hat{\theta} = \arg\min_{\theta \in \Gamma} \left[ r(\theta) + \sum_{l=1}^{m} \mathcal{O}_l(S_l, T_l, \theta) \right]$$

where $r(\theta)$ is a regularization parameter and $\mathcal{O}(S_l, T_l, \theta) = \alpha_l \sum_{j=1}^{\bar{n}_l} L(\hat{f}_{l,\theta}(\bar{\mathbf{X}}_j^l), \bar{Y}_j^l)$ where $\alpha_1 > 0$ are weighting parameters, we can then write our main optimization problem as

$$[\{\hat{\mathbf{w}}_l, \hat{\mathbf{v}}_l, \hat{\theta}\}] = \arg\min_{\{\mathbf{w}_l, \mathbf{v}_l\}, \theta} \Big[ r(\theta) + \sum_{l=1}^{m} \Big( g(\mathbf{w}_l, \mathbf{v}_l) + \frac{1}{n_l} \sum_{i=1}^{n_l} L(f_\theta(\mathbf{w}_l, \mathbf{v}_l; \mathbf{X}_i^l), Y_i^l) \Big) \Big]$$

The optimization is done on the unlabeled data that have been labeled with auxiliary class labels for each $l$ up to $m$ with a fixed $\theta$ to approximate $\hat{\mathbf{w}}_1$.

$$\hat{\mathbf{w}}_l = \arg\min_{\mathbf{w}_l} \left[ \frac{1}{n_l} \sum_{i=1}^{n_1} L(\mathbf{w}_l^T \mathbf{X}_i^l + (\mathbf{x}_l^T \theta) \mathbf{X}_i^l, Y_i^l) + \lambda_l \|\mathbf{w}_l\|_2^2 \right]$$

where the $\lambda$ are given constants. After each of these calculations, $\mathbf{u}_l$, which was originally set to 0 for all $l = 1, \ldots, m$, is set to $\hat{\mathbf{w}}_l + \theta^T \mathbf{v}_l$

The rows of the rows of $\theta$ become the first $h$ rows of $V_1^T$ from the SVD of $\mathbf{U} = [\sqrt{\lambda_1}\mathbf{u}_1, \ldots, \sqrt{\lambda_m}\mathbf{u}_m]$. With this new $\theta$, the process is repeated until it convergence.

Finally, a predictor for the target problem by empirical risk minimization on the original labeled data is learned using $\theta$.

It is also important to not the differences between SVD-ASO and PCA. While PCA is a dimensionality reduction in the data space $\mathcal{X}$, SVD-ASO is a does its dimensionality reduction on the predictor space. In addition, SVD-ASO looks for low-dimensional structures that have the highest predictive power while PCA's principle components of the input data in the data space will not necessarily have a good predictive power.

### *F. Structural Correspondence Learning[4]*

SCL is an extension of SVD-ASO. Its key idea of SCL is finding "pivot features" to model the correlation between features in the source and target domains. Given labled source data and unlabled from the target, SCL creates a predictor as follows:

1) Identify pivot features
   $m$ pivot features are selected that occur frequently in both domains while still being diverse enough to be able to estimate the covariance with non-pivot features.
2) Determine the Pivot Predictors
   From each of our $m$ pivot features, $m$ binary classification problems are formed asking if the pivot feature is in the given example. These classification problems are not concerned with the labels, so they can be trained on all of the data. So, for $l = 1$ to $m$, we calculate

$$\hat{\mathbf{w}}_l = \arg\min_{\mathbf{w}} \Big( \sum_j L(\mathbf{x} \cdot \mathbf{x}_j, p_l(\mathbf{x}_j)) + \lambda \|w\|^2 \Big)$$

   where $p_1(\mathbf{x})$ is the binary predictor problem for pivot feature $l$, and $L(p, y)$ is the modified Huber loss as discussed in II-E. Here, the weight vectors $\hat{\mathbf{w}}_1$, will encode the covariance of the non-pivot features with the pivot features.

   We make some observations about the pivot predictors. The $z^{th}$ feature's weight as given by the $l^{th}$ pivot predictor is positive when $z$ is positively correlated with $l$. Since pivot features that occur frequently in both domains are chosen as pivot features, it is expected that non-pivot features from both domains will be correlated with them. When two non-pivot features are correlated similarly with many of the same pivot features, they are highly correlated. Finally, it can be observed that $\hat{\mathbf{w}}_1$ is a linear projection of the original feature space onto $\mathbb{R}$
3) Find mapping $\theta$ from the SVD of $W$
   First, it is a good statistical and computational choice to compute a low-dimensional linear approximation of the pivot predictor space. Letting $W = UDV^T$ be a matrix with the pivot predictor weight vectors as columns, $U_{[1:h,:]}^T$, a matrix whose rows are the top left singular vectors of $W$, now equals $\theta$.

   It can be observed that the rows of $\theta$ are the principle pivot predictors which capture the variance of the pivot predictor space in $h$ dimensions as best as possible. Additionally, $\theta$ can also be observed to be a projection onto $\mathbb{R}^h$ from the original feature space, so $\theta\mathbf{x}$ is the desired mapping to the low-dimensional shared feature representation.
4) Train the predictor
   First, the new shared features $\theta\mathbf{x}_t$ are added to the original features $\mathbf{x}_t$. Then a standard discriminative learner is trained on the source data and returned.

For the purposes of this report, we used a decision tree as the "standard discriminative learner" since it deals with multi-class learning easily.

Another change we made was in the binary predictions in the second step. Blitzer et al. mention that the pivot feature itself should be removed from each example when making the prediction because it is totally predictive of the the presence of the pivot feature in the example. However, this caused problems with the dimensionality of the resulting weight vectors and, eventually, the $W$ matrix, so this instruction was followed, possible to the detriment of the collected results.

## *G. Self-Taught Clustering[6]*

In general, there has been limited work on unsupervised transfer learning. One of the reasons for this could be the relative difficulty of this task, as well as the relatively limited work in the general domain in unsupervised learning methods. However, there have been a few approaches in the past years. Specifically, one group has developed a co-clustering technique, while another has developed a dimensionality reduction technique that uses source and target domain data.

A common method to approach unsupervised learning tasks is through clustering, in which an algorithm attempts to group similar examples together to maximize the similarity within the group as well as minimize the similarity between elements in the group and those in other groups. Dai et al. have modified this concept to take advantage of a target and source domain to employ transfer learning. Much of the motivation for this method came from information theory[8] and more conventional self-taught learning[17]. The algorithm is therefore called self-taught clustering (STC) and **has been implemented in this paper**. As such, references to equations in this section will be made in the methods and results sections.

As mentioned previously, STC transfers a feature representation by finding a common feature representation that has relevant features in both the source and target domains. Due to the underlying assumption of common relevant features, having additional auxiliary unlabelled data will help develop this feature representation. Then, examples from the source and target domain are co-clustered using the common feature space. The clustering is done using an information theoretic algorithm to minimize the *loss in mutual information* before and after the co-clustering, much like decision trees in the c4.5 algorithm. This will be formalized below.

We can define two random variables (R.V.) X and Y as selecting from sample image sets $\{x_1, x_2, x_3..., x_m\}$ and $\{y_1, y_2, y_3..., y_n\}$ respectively. The $X$ variable is the target dataset, while the $Y$ variable is the auxiliary dataset, which will be used in the transfer learning. We also define a R.V. Z that draws from a set of common features $\{z_1, z_2, z_3..., z_k\}$.

We can therefore represent the joint probability distribution $P(X, Z)$ as shown in 12 and $P(Y, Z)$

$$P(X, Z) = \begin{bmatrix} 0.2 & 0.0 & 0.2 \\ 0.0 & 0.2 & 0.0 \\ 0.0 & 0.2 & 0.2 \end{bmatrix} \tag{12}$$

We use $0.2$ since we want the probability for all values, so we divide the total occurrences of our features in the images and divide our frequency matrix by this value. This calculation is then repeated for the auxiliary dataset to achieve $P(Y, Z)$.

We will then define cluster functions for each random variable, defined as $C_X : X \mapsto \widetilde{X}$, $C_Y : Y \mapsto \widetilde{Y}$, $C_Z : Z \mapsto \widetilde{Z}$. These functions map each example into a particular cluster. Using these values, we can define our objective function as

$$I(X, Z) - I(\widetilde{X}, \widetilde{Z}) \tag{13}$$

The function 13 minimizes loss in mutual information before and after co-clustering between the instances $X$ and features $Z$. We can define the two functions as follows (Figure 14).

$$I(X, Z) = \sum_{x \in X} \sum_{z \in Z} p(x, z) log \frac{p(x, z)}{p(x)p(z)} \tag{14}$$

The mutual information with respect to the two clusters $I(\widetilde{X}, \widetilde{Z})$ is associated with the joint probability with respect to the two clusters, one for image, and one for common features, as shown in figure 15.

$$p(\widetilde{x}, \widetilde{z}) = \sum_{x \in \widetilde{x}} \sum_{z \in \widetilde{z}} p(x, z) \tag{15}$$

We can also use this formula to calculate the $q(\widetilde{x}, \widetilde{z})$ in the same method.

If we employ the following clustering functions, $\widetilde{x}_1 = \{x_1, x_2\}, \widetilde{x}_2 = \{x3\}$ on $X$ and $\widetilde{z} = \{z_1, z_2\}, \widetilde{z}_2 = \{z_3\}$ on $Z$, we can calculate the joint probability of clusters $\widetilde{X}$ and $\widetilde{Z}$ off of equation 15. We sum over the rows and columns included within each row and cluster to get an $m \times n$ matrix where $m$ is the number of clusters $\widetilde{x}$ and n is the number of clusters $\widetilde{z}$. Doing so, we get,

$$P(\widetilde{X},\widetilde{Z}) = \begin{bmatrix} 0.4 & 0.2 \\ 0.2 & 0.2 \end{bmatrix} \tag{16}$$

Theoretically, we then get an objective function that can be formulated as

$$\mathcal{J} = I(X,Z) - I(\widetilde{X},\widetilde{Z}) + \lambda\Big[I(Y,Z) - I(\widetilde{Y},\widetilde{Z})\Big] \tag{17}$$

In the equation above, the $\lambda$ is a tradeoff parameter that determines how to balance the influence of the target dataset on the source dataset, which is a hyperparameter that is tested in the experiments below. The original group to implement this found that a hyperparameter of 1.0, or an equal influence of source and target data worked the best. We see from equation 17 that while the loss in information gain is calculated separated for both the source and target datasets, they are combined in the objective function that will be optimized through the STC algorithm.

One issue that we run into however, is that the STC objective function formulation as we have it is difficult to minimize, so we transform it as a difference in Kullback-Leibler divergence values, as shown following. We first define a few new probabilities.

$\widetilde{p}(X,Z)$ denotes the joint probability distribution of X and Z with respect to the co-clusters $(C_X, C_Z)$ as shown in equation 18.

$$\widetilde{p}(x,z) = p(\widetilde{x},\widetilde{z})\frac{p(x)}{p(\widetilde{x})}\frac{p(z)}{p(\widetilde{z})} \tag{18}$$

Given matrices 12 and 16 we get the following joint probability matrix in matrix 19

$$\widetilde{p}(X,Z) = \begin{bmatrix} 0.089 & 0.178 & 0.133 \\ 0.044 & 0.089 & 0.067 \\ 0.067 & 0.133 & 0.200 \end{bmatrix} \tag{19}$$

If we define the probability of an image as

$$p(x) = \sum_{z \in Z} p(x,z) \tag{20}$$

by the definition of marginalization. We also define the probability of an image cluster as

$$p(\widetilde{x}) = \sum_{x \in \widetilde{x}} p(x) \tag{21}$$

Using these two and the matrix in 16, we have all the components necessary to calculate this matrix. This calculation is repeated for $(Y, Z)$ in the equation

$$\widetilde{q}(y,z) = p(\widetilde{y},\widetilde{z})\frac{p(y)}{p(\widetilde{y})}\frac{p(z)}{p(\widetilde{z})} \tag{22}$$

We can rewrite the objective function in equation 17 as following

$$\begin{aligned} I(X,Z) - I(\widetilde{X},\widetilde{Z}) + \lambda\Big[I(Y,Z) - I(\widetilde{Y},\widetilde{Z})\Big] \\ = D(p(X,Z)||\widetilde{p}(X,Z)) + \lambda D(q(Y,Z)||\widetilde{q}(Y,Z)), \end{aligned} \tag{23}$$

where $D(\cdot||\cdot)$ is the KL divergence defined for conditional probabilities as

$$D(q(u \mid v) \mid\mid p(u \mid v)) = q(u \mid v)\log\frac{q(u \mid v)}{p(u \mid v)} \tag{24}$$

The authors show that the KL divergence for joint probabilities from our objective function can be written as KL divergences from conditional probabilities to make optimization easier in the algorithm.

This gives us $D(p(Z|x)||\widetilde{p}(Z|\widetilde{x}))$. They then also show that minimizing this value for a single $x$ minimizes the function globally as well. Thus, we can iteratively choose the best cluster per example $x$ to minimize the objective function monotonically. This gives us the following cluster functions for example sets $X$, $Y$, and common feature set $Z$.

$$C_X(x) = \arg\min_{\widetilde{x} \in \widetilde{X}} D(p(Z|x)||\widetilde{p}(Z|\widetilde{x})) \tag{25}$$

$$C_Y(y) = \arg\min_{\widetilde{y} \in \widetilde{Y}} D(q(Z|y)||\widetilde{q}(Z|\widetilde{y})) \tag{26}$$

$$\begin{aligned} C_Z(z) = \arg\min_{\widetilde{z} \in \widetilde{Z}} D(p(X|z)||\widetilde{p}(X|\widetilde{z})) \\ + \lambda D(q(Y|z)||\widetilde{q}(Y|\widetilde{z})) \end{aligned} \tag{27}$$

We can show the expanded form of one of these equations to be

$$\begin{aligned} C_X(x) = \arg\min_{\widetilde{x} \in \widetilde{X}} \frac{p(x,z)}{p(x)} * \\ log\big(p(x,z)p(\widetilde{x})p(\widetilde{z})\big) - log\big(p(\widetilde{x},\widetilde{z})p(z))\big) \end{aligned} \tag{28}$$

Keep in mind that this requires that all of these clusters are kept constant. Due to no details given by the author, in our testing of this algorithm, we kept these values constant based on the initialization values.

We can see from equation 27 that although we determine the clustering functions normally, the common feature space $Z$ and its clustering function act as the bridge between our auxiliary and target datasets.

Based on these three functions, and the conditional probabilities we calculated earlier, we are able to write out the pseudocode for our algorithm.

**Self-Taught Clustering Pseudocode**

*Input:* The target unlabeled dataset $X$; an auxiliary dataset $Y$, the shared feature space $Z$ for both $X$ and $Y$, three initial clustering functions $C_X^{(0)}(x)$, $C_Y^{(0)}(y)$, $C_Z^{(0)}(z)$, the number of iterations $T$

*Output:* The final clustering function $C_X^{(T)}(x)$ on target data $X$.

*Procedure:*

```
1) Initialize $p(X, Z)$ and $q(Y, Z)$ based on the data
2) Initialize $\widetilde{p}^{(0)}(X, Z)$ based on $p(X, Z)$ and using
   the clustering functions for $X$ and $Z$ to calculate
   a matrix similar to equation 16
3) Initialize $\widetilde{q}^{(0)}(Y, Z)$ based on $q(Y, Z)$ and its
   corresponding clustering functions
4) for t $\leftarrow$ 1, ..., $T$ do
5) Update $C_X^{(t)}(x)$ using $p$ and $\widetilde{p}^{(t-1)}(X, Z)$
6) Update $C_Y^{(t)}(y)$ using $q$ and $\widetilde{q}^{(t-1)}(Y, Z)$
7) Update $C_Z^{(t)}(z)$ using $p$, $q$, $\widetilde{p}^{(t-1)}(X, Z)$, and
   $\widetilde{q}^{(t-1)}(Y, Z)$
8) Update $\widetilde{p}^{(t)}(X, Z)$ based on $p(X, Z)$, $C_X^{(t)}(x)$,
   and $C_Z^{(t)}(z)$
9) Update $\widetilde{q}^{(t)}(Y, Z)$ based on $q(Y, Z)$, $C_Y^{(t)}(y)$, and
   $C_Z^{(t)}(z)$
```

As can be seen from the above pseudocode, we basically repeatedly calculate objective function-minimized new clusters based on our joint probabilities and then use them to update our joint probabilities until convergence. In order to create their initial feature set, they used SIFT keypoint descriptors and the visual "bag of words" representation [9], explained in more detail in the methods section.

Dai et al. conducted their experiments on the Caltech-256 image corpus, and conducted a comparative evaluation. In general, they created binary clustering tasks to discriminate between two categories, and used the other 254 categories to train. However, they did also test a 3-way and a 5-way clustering task. They compared their algorithm against conventional co-clustering and a few other common clustering algorithms, showing significant decrease in entropy (their metric of choice), defined in their paper. It essentially calculates the purity of each cluster and then aggregates them for the entire algorithm as a weighted sum. In general, the group showed that they achieved an average entropy value of $0.610$ which was significantly less than the next best algorithm at $0.865$. They also found that in tested the tradeoff parameters $\lambda$ they found the entropy to drop until they hit $1.0$. Based on theoretical and empirical validation, Dai et al. found that STC works effectively in transferring knowledge in the unsupervised setting from a source to target domain.

### *H. Transferred Discriminative Analysis[18]*

Another method that has been studied to transfer knowledge was originally cited as an unsupervised model in [16]. However, in reviewing the paper, it could also be considered a transductive model, due to the allowance of target labels in the source domain. This model is known as transferred discriminative analysis (TDA). In general, the concept behind is the paper is to extract discriminative information from a labelled dataset and use that information, along with unsupervised discriminative dimensionality reduction to adjust the results of the final clustering of data. The reason that this is not classified as semi-supervised learning is because the classes of the labelled and unlabelled data do not have to be the same, so there is no direct transmission of knowledge. This is also the reason that this algorithm was no implemented in this study. *Because of the same classes across all three of our chosen datasets, our task would be the special case in TDA that makes it semi-supervised learning*, as opposed to unsupervised learning.

In order to find the discriminative structure of data, we can define three different scatter matrices given a set of labelled data $\mathbf{X}_L = (\mathbf{x}_1, \mathbf{x}_2 ... \mathbf{x}n)$ from $C$ classes as the within-class scatter matrix $\mathbf{S}_w$, the between-class scatter matrix $\mathbf{S}_b$, and the total-scatter matrix $\mathbf{S}_t$. These functions can be described below as

$$\mathbf{S}_w = \sum_{j=1}^{C} \sum_{i=1}^{l_j} (x_i - m_j)(x_i - m_j)^T = XL_wX^T \tag{29}$$

$$\mathbf{S}_b = \sum_{j=1}^{C} l_j (m_j - m)(m_j - m)^T = XL_bX^T \tag{30}$$

$$\mathbf{S}_t = \sum_{j=1}^{C} (x_i - m)(x_i - m)^T = \mathbf{S}_b + \mathbf{S}_w \quad XL_tX^T \tag{31}$$

In the above equation, $x_i$ and $m_j$ are both matrices and $\mathbf{L_w}$, $\mathbf{L_b}$, and $\mathbf{L_t}$ are all the graph Laplacians with definitions in the paper.

In the TDA algorithm, we modify the above definitions of the between-scatter and total-scatter matrices to include a relationship between the labelled and unlabelled data. We calculate the between-class scatter by including both the source and the target data. This can be defined as

$$\mathbf{S_b} = \mathbf{S_{bl}} + \widetilde{\mathbf{S}}_{\mathbf{bu}} = \sum_{i=1}^{C} l_i m_i m_i^T + \sum_{j=1}^{C} l_j \widetilde{m}_j \widetilde{m}_j^T \quad (32)$$

In this equation, the first part is the between-cluster scatter of the labeled data. We then wanted to include the between-cluster scatter of the unlabeled clusters of the data as defined by $\widetilde{m}_j$.

To modify the total-scatter matrix, we get We can use a modified Graph Laplacian to describe the structure of the unlabeled data. We can define an adjacency matrix $M$ of a neighborhood graph as

$$\mathbf{M}_i j = \begin{cases} e^{-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2}} & if(\mathbf{x_i}, \mathbf{x}_j) \in E \\ 0 & otherwise \end{cases} \quad (33)$$

We use this to define our graph Laplacian as

$$\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}} \mathbf{M} \mathbf{D}^{-\frac{1}{2}} \quad (34)$$

We can combine equation 34 and 31 for the total-scatter matrix to get the following equation with a zero sample mean,

$$(\mathbf{S}_t + \lambda \mathbf{X} \mathbf{L} \mathbf{X}^{\mathbf{T}}) = \mathbf{X}(\mathbf{I} + \lambda \mathbf{L})\mathbf{X}^{\mathbf{T}} \quad (35)$$

The last two things we need to define for the objective function of TDA is a dummy indicator matrix $\mathbf{H_u}$ that represents the clustering structured of the unlabeled data that we will generate and a matrix to represent the projection direction of the dimensionality reduction, known as $\mathbf{W}$. Using all these values, we get the following target of TDA as

$$\begin{aligned} \arg\max_{\mathbf{W}, \widetilde{\mathbf{H}}_{\mathbf{u}}} \textbf{trace}((\mathbf{W^T}(\mathbf{S_t} + \lambda \mathbf{X} \mathbf{L} \mathbf{X^T})\mathbf{W})^{-1} \\ (\mathbf{W^T}(\mathbf{S_{bl}} + \widetilde{\mathbf{S}}_{\mathbf{bu}}(\widetilde{\mathbf{H}}_{\mathbf{u}}))\mathbf{W})) \end{aligned} \quad (36)$$

Instead of optimizing this function directly, we will iteratively estimate the discriminative structure of the unlabeled data and then use the projected data in the lower dimension to revise the clustering result, until convergence. The clustering operation used in the paper is K-means clustering.

The algorithm uses the addition of the labeled data in the maximization of the traces by ensuring that the projection of the labelled and unlabelled data into a lower-dimensionality subspace retains the discriminative structure of the labeled data. Therefore, if this structure is lost, the objective function is decreased, forcing the unlabeled data to cluster in a similar discriminative structure to the labeled data. As we calculate the most discriminative projection direction, we optimize the clusters of the unlabeled data, with knowledge from the source domain.

Using this understanding, we can define the algorithm pseudocode as below,

*Input:* The input labelled data $\mathbf{D}_L = \{\mathbf{X_L}, \mathbf{Y_L}\}$ which belongs to C classes, and the unlabelled data $\mathbf{D_U} = \{\mathbf{X_U}\}$ with number of classes K

*Output:* The final clustering on the unlabeled data $\widetilde{\mathbf{H}}_{\mathbf{u}}$, and the projection matrix $\mathbf{W_{tda}}$

*Procedure:*

1) Find an optimal subspace with dimensionality $m = C + K - 1$ using eigenvalue decomposition for the objection function in equation 36
2) Cluster the unlabeled data in the subspace found in step 1. We do this by fixing projection direction amtrix $\mathbf{W}$ and using a clustering method to get the optimal indicator matrix $\widetilde{\mathbf{H}}_{\mathbf{u}}$ for the unlabeled data. The equation to do this is $max_{(\widetilde{\mathbf{H}}_{\mathbf{u}})} \widetilde{\mathbf{S}}_{\mathbf{bu}} \widetilde{\mathbf{H}}_{\mathbf{u}}$.
3) Go to step 2. Repeat until convergence, which is determined by the cluster indicator matrix $\widetilde{\mathbf{H}}_{\mathbf{u}}$ not changing between two iterators.

Wang et al. then kernalized the objective function to achieve kernel TDA using the Representer theorem. To test their algorithm, the authors two different facial recognition databases, AT&T and Yale, cited in their paper. In the paper, the authors only demonstrated their results as compared to three other facial recognition algorithms, *Eigenface*, *LaplacianFace*, and *DisKmeans*.

They show accuracy scores over a number of different $C$ and $K$ values, which are the labeled and unlabeled classes. In general, they show that the TDA algorithm is able to achieve between $0.92 - 1.0$ accuracy, as compared to the next best algorithm *DisKmeans* which only achieves, on average, $0.89$ in accuracy. This remains consistent as the number of classes is varied, although at low Ks the improvement by the TDA algorithm is more significant over the other algorithms than at higher Ks. They also tested unconstrained face recognition using the Labeled Faces in the Wild (LFW) database. Using this database, they got equally improved accuracy of between $0.94 - 0.99$, while *DisKmeans* only performed at $0.89 - 0.96$.

In summary, this algorithm has the ability to effectively discover the true structure of unlabeled data using some source labeled data (in a different subspace) using only a few iterations. Moreover, the TDA algorithm learns both a clustering of the unlabeled data and a discriminative model of the unlabeled and labeled samples in a lower-dimensionality subspace than the one initially provided.

## *I. Translated Learning[5]*

Unlike most algorithms, Translated Learning deals with situations where there is no correspondence between the source and target feature spaces. This is an example of an algorithm that requires the use of an oracle, which translates the training data into a target feature space so that learning can be performed in a single feature space. This way one can reduce the amount of time and effort of labelling.

Because the source and target feature spaces are different, a translator must be created from co-occurrence data, which can be represented as any of the following joint probabilities between the source and target data:

$$p(y_t, y_s), p(y_t, x_s), p(x_t, y_s), p(x_t, x_s)$$

Wenyuan et al. estimate the translator by estimating the feature-level co-occurrence probability (Equation), expressing the translator as:

$$p(y_t|y_s) = p(y_t, y_s) / \int_{X_t} p(y_t', y_s) dy_t'$$

Such that $X_t$ represents the target instance space. Next the authors estimate a hypothesis from the labeled co-occurrence data.

*Risk Minimization Framework:* In order to estimate an optimal hypothesis, a risk minimization framework is used to measure the risk for classifying the target data correctly. Next, the class label that minimizes the risk is chosen. The authors formalize the expected loss function as:

$$R(c, x_t) \equiv L(r-1|c, x_t) - \int_{\Theta_C} \int_{\Theta_{X_t}} L(\theta_C, \theta_{X_t}, r-1)$$
$$p(\theta_C|C) p(\theta_{X_t}|x_t) d\theta_{X_t} d\theta_C$$

Where $r = 1$ means that the label of $x_t = c$. Estimation: Because $\theta C$ and $\theta(X_t)$ can increase exponentially in size, the authors use the Kullback-Leibler divergence to find the distance between the two models, and assuming there is no prior difference between classes, the authors rewrite the risk as:

$$R(c, x_t) \propto \Delta(\hat{\theta}_C, \hat{\theta}_{X_t})$$

## *J. Feature Space Remapping (FSR)[10]*

Cook et al. propose a novel method for transferring knowledge between domains with different feature spaces that does not require a translator oracle. FSR uses labeled data in the target domain to map to the source domain. It also operates under the assumption that related features between source and target datasets have similar distributions in values. The authors propose both a informed and uninformed version of this algorithm, but for the sake of simplicity I will only discuss the informed version, as it is the one implemented in this project.

FSR begins by computing meta-features for both the source and target features, in order to later compare them and compute similarity scores between features:

$$E(x|c) = \frac{1}{n_c} \sum_{i=1}^{n} x_i$$

Where E represents the expected value of a feature given a class label. In the MNIST data there are ten candidate labels ranging from 0 to 10, so each feature has ten meta-features

The average similarities between each source and target meta-feature are computed and used as similarity scores. The average similarity of a source and target feature is given by:

$$S_{x_y} = \frac{1}{N} \Omega(m_x^i, m_y^i)$$

N represents the number of meta-features per feature. For MNIST classification, $\{0-9\}$ represent ten possible labels for each entry in the target data, so $N = 10$.

Because the meta-features are all positive, the normalized equation becomes:

$$\Omega(m_x^i, m_y^i) = 1 - \frac{|m_x^i - m_y^i|}{max(m_x^i, m_y^i \forall x \in D_s \forall y \in D_t)}$$

Omega represents the normalized similarity between two meta-features. By summing the normalized differences for each meta-feature of each pair of source-target features, we can obtain the similarity score for that pair. Next, FSR maps each target feature to a source feature by selecting the source feature with the highest similarity score, producing a many-to-one mapping from target to source.

Once the features with maximal scores are selected, FSR applies the mapping to the target data so that a hypothesis trained on the source data can classify it. If more than one target feature maps to a source feature, an aggregation protocol must be applied to the feature values, such as selecting the maximum, average, or total. The authors postulate that using this mapping can be interpreted as a greedy approximation for a loss function of the classifier.

### K. CATE Estimators and Warm Start[12]

### L. Unsupervised Representation Learning for Domain Adaptation[3]

*1) Context:* This paper seeks to show that unsupervised pre-training can be used to expand the domain a learning algorithm can perform well on. The paper set up a challenge to tackle with this approach. The challenge was that the test and validation sets are selected from classes that are sparse in the training set. In addition, the labeling of these sets are sparse, and the training set contains some labeled examples of classes that are not in the test or validation sets. In order to make a representation for the test set example, a transformation from the input vectors into a new space should be learned, where this transformation has the goal of capturing the critical variation factors present in the distribution of the training set. Then, that transformation is applied to the test set, and a classifier is then trained on a subset of the test set. Performance is measured on the rest of the test set [3].

The conditions and constraints for this scenario are as follows:

- The training set distribution is different from the test set distribution. This means it may not be possible to transfer knowledge from the training set to the test set.
- Few labels are present in the test set, making the the ability of the classifier to generalize sensitive to the architecture of the representation chosen.
- Labels of examples in the test set are completely missing from the training set, meaning the transformation may optimize for the classes present in the training set, which may not carry over to the classes in the test set.

*2) Domain Adaptation and Transductive Specialization:* The conditions described in the previous section set up a transfer learning issue of domain adaptation, where the distribution of the source domain is vastly different from the target domain. Bengio et al. used a *transductive strategy* to solve these issues. What they did was train the topmost level of the unsupervised feature-learning hierarchy on the test set, while using the larger and more diverse training set to obtain the general, abstract features for the possible variations across classes. In order to identify the features that are useful in the test distribution, features must be selected out the extracted features from the training set. This was done by determining the feature variation in the test set, and selecting the features from the extracted set of features from the training set for which there was high variation in the test set. This can be done with Principle Component Analysis (PCA) on the last level of the classifier, over the trained test examples [3].

According to the seed paper, this transfer learning strategy falls under both transductive and unsupervised transfer learning. Because this method is domain adaptation, it can be considered transfer learning. However, since neither the training set or the test set are fully labeled, as well as the use of unsupervised pre-training and dimensionality reduction, this method falls under unsupervised transfer learning.

## III. EXPERIMENTS

### A. Datasets

The datasets that we used in our experiments are MNIST [13], Fashion MNIST (FMNIST) [19], and house numbers (HNUM) [15].

MNIST consists of grayscale small cropped images of handwritten numbers. Similarly, FMNIST consists of grayscale small cropped images of hand-drawn clothing pieces. The house numbers data set consists of real-world images of digits taken from the sides of houses, and the actual digits are centered on the image.

We particularly chose these datasets because each of them has something similar to another, which is essential for transfer learning (of similar tasks). For instance, MNIST and Fashion MNIST are represented in the same format and the house numbers and MNIST both consist images of numbers.

### B. Self-Taught Learning

Datasets: house numbers, MNIST, Fashion MNIST

Pairs of datasets were used to test the algorithm, each took turn being the source domain and the target domain. When learning the classifier for each experiment, we applied parameter selection for the

SVM classifier using the following ranges of values:

Kernel : [linear, polynomial, rbf]

Degree : [2,3] , for polynomial kernel

C : [0.001, 0.01, 0.1, 0.5, 1, 2, 3] , the trade-off constant

Gamma : [0.001, 0.01, 0.1, 0.5, 1, 2, 3], the kernel coefficient for rbf and polynomial

The reported result of each experiment is the result obtained from the best set of parameters. In each experiment, a set of 1000 unlabeled data was randomly generated to learn the dictionary. A training set of size 2500 and a test set of size 500 of labeled data were randomly generated with equal amount of examples from each of the 10 classes. Cross-validation with 5 folds was applied.

*a) Baseline results:* First, we collected baseline results when learning each dataset without transfer learning. The performance collected from the best SVM model for each dataset is shown on $Table$ I.

TABLE I
BASELINE PERFORMANCE

| Dataset | MNIST | Fashion MNIST | House Numbers |
|---|---|---|---|
| Accuracy | $91.1 \pm 0.3$ | $81.7 \pm 0.4$ | $33.9 \pm 0.5$ |

*b) Self-taught Learning:* We collected the results of pairwise experiments, as described earlier in this section. The performance of the best SVM model in each experiment is shown in $Table$ II (The unlabeled data is in the left column).

TABLE II
SELF-TAUGHT LEARNING RESULTS

| | Mnist | Fashion Mnist | House Numbers |
|---|---|---|---|
| Mnist | | $80.7 \pm 0.8$ | $40.5 \pm 1.2$ |
| Fashion Mnist | $91.4 \pm 0.2$ | | $42.4 \pm 1.6$ |
| House Numbers | $86.9 \pm 0.7$ | $82.0 \pm 0.6$ | |

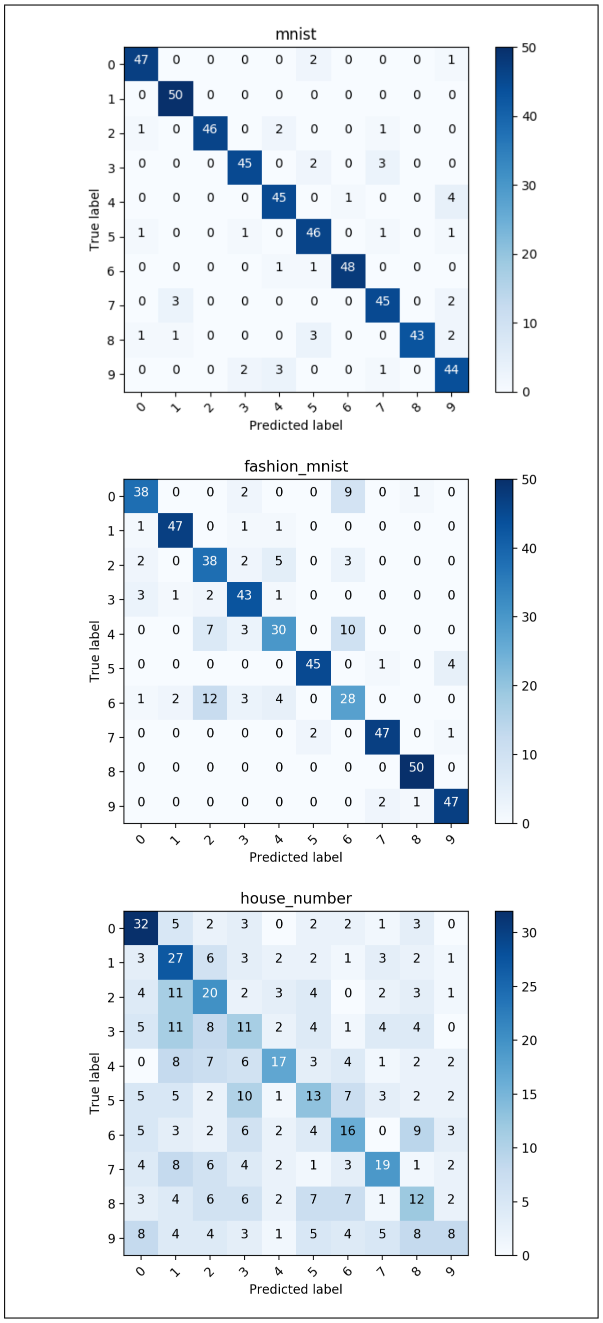

Fig. 1. Baseline Confusion Matrices

The experiment results above show that choosing an appropriate dataset to learn the dictionary has a positive effect on the classification performance on the target domain. To be specific, using MNIST or house numbers as the source domain does not improve the classification performance of Fashion MNIST. Using Fashion house numbers as the source domain has negative effect on the classification of MNIST. However, the classification performance of house numbers improve by about 6% when using MNIST as the source domain and by about 8% when using Fashion MNIST as the source domain.

*c) Varying the amount of source domain data:* With the unlabeled - labeled pair the yielded that largest improvement, Fashion MNIST - house numbers,

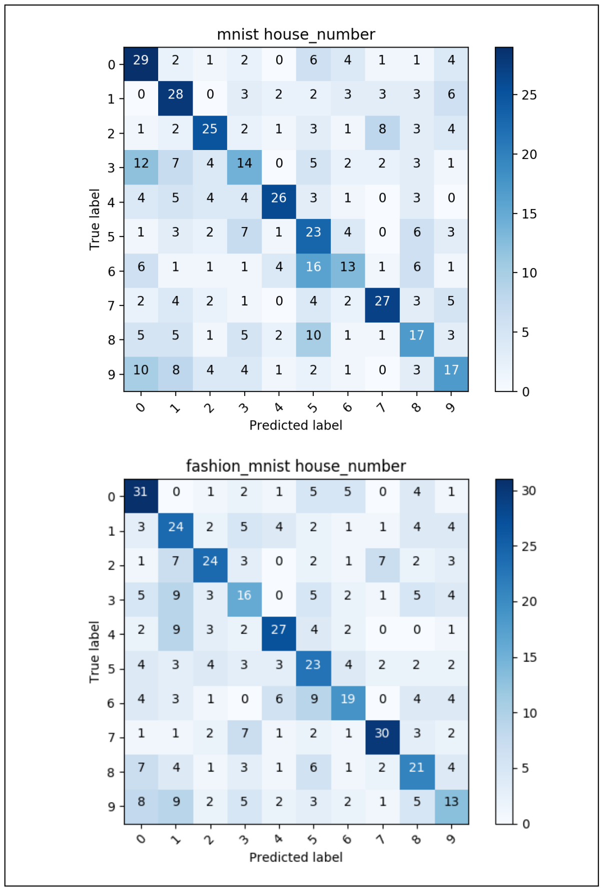

Fig. 2. Self-taught Learning Performance on house numbers

we conducted another experiment to see the effect of increasing the amount of unlabeled data on the classification performance.

Using the same setup as before, this time, we varied the amount of unlabeled data (200, 500, 1000 - the dictionary size was kept the same). The results are shown in *Table* III.

TABLE III

AMOUNT OF UNLABELED DATA VERSUS PERFORMANCE

| Unlabeled data size | 200 | 500 | 1000 |
|---|---|---|---|
| Performance | $45.2 \pm 0.7$ | $44.8 \pm 1.0$ | $42.4 \pm 1.6$ |

Interestingly, smaller amount of unlabeled data yielded better classification result on the target domain. This might be due to overfitting when the dictionary started picking up the irrelevant details from the unlabeled data set. To be more specific, when there was more unlabeled data, the dictionary became less sparse as the learner started including features from the unlabeled domain that is not relevant to the target domain.

*d) Interpretation of the results:* The obtained results show that self-taught learning did not improve the classification performance on the MNIST and Fashion MNIST datasets. Using house numbers as the unlabeled data for classifying MNIST actually hurt the performance.

However, there was a significant improvement of at least 6% on the house numbers dataset.

A possible explanation for these observations might be came up by looking at the complexity of the classification tasks, where:

$$\text{MNIST} < \text{Fashion MNIST} < \text{House Numbers}$$

A human learns by picking up simple concepts first and then applying those concepts to learn more complex concepts. That intuition may holds in this case. Applying a more complex representation to a less complex concept may not improve the overall performance (house numbers $\rightarrow$ Fashion MNIST) or even hurt the overall performance (house numbers $\rightarrow$ MNIST). In the other hand, applying a less complex representation to a more complex concept may improve the performance (MNIST $\rightarrow$ house numbers and Fashion MNIST $\rightarrow$ house numbers).

Another thing that we can think about is what the algorithm is actually doing. Consider 2 learning tasks: classification of *Shoes* and classification of *Boat*. We know that *Shoes* and *Boat* have similarities in term of their shapes, and learning the representation of *Shoes* beforehand may be beneficial for learning the *Boat* classifier. However, only some features of the *Shoes* are actually useful for learning *Boat* (the general oval shape); other features such as shoe laces are not relevant. For that reason, we would want to pick only the relevant features. The self-taught learning algorithm that we evaluated, in the contrary, pick all the features. Essentially, the algorithm first learns all the useful features for classifying *Shoes* and then fit the *Boat* into the shoe representation. As the result, when learning to classify *Boat*, the algorithm may not learn the boat concept, but the things that look like shoes concept.

In that sense, it is reasonable for the overall performance to go down when we try to fit a more complex concept into a less complex concept.

*e) Further results to compare with other transfer learning approaches:* We prepared another set of experiments to get results that we can used to compare with other algorithms that we implemented. Pairwise experiments with 200 unlabeled examples, 600 labeled training examples, 3-fold cross validation, and parameter selection were conducted. For each experiment, the average performance of 3 folds and the performance for each class in each fold were returned. We used the result with the highest performance for each dataset for comparison with other algorithms.

### C. Sample Transfer Zero-Shot Learning

*a) Experiment set-up:* For STZSL, we ran a total of eight experiments for each data set. Each experiment was a change in one of three different parameters of the algorithm, but followed the same approach. For each experiment, 600 samples were pulled at random with replacement from the data set file and were then run on three-fold cross validation.

Within each fold, there were $\binom{M}{2}$ separate iterations of the STZSL algorithm where $M$ is the number of classes in the data set *(all three data sets had 10 classes)* as all pairs of two classes, $c_i$ and $c_j$, were used as the "unlabeled" classes. The individual samples that were classified as either $c_i$ or $c_j$ were put into a target samples list while all other samples were put into a source samples list and paired with their labels. For half the experiments, STZSL was given the target samples list so as to run STZSL-T and the other half were run without that information so as to run STZSL-I.

Following the source paper's experimental design, the embedding vectors were created as just the average vector for each class from the training data for the fold, and half of the experiments were given these. The other half were given the class similarity matrices, which were calculated following the original paper's experimental design as well. Each entry $g_{ij}$ of the class similarity matrix is just defined as the magnitude of the Euclidean distance between the embedding vectors $i$ and $j$.

Finally, the algorithm itself has an important hyperparameter $\beta$ that affects how much similarity between transferred samples is allowed. Half of the experiments used $\beta = 0$, which means that similarity between examples is ignored. The other half of the experiments used $\beta = 50$, a value that lied within the range of values that the original paper's experiment used for this parameter.

Altogether, these three different parameters each with two states being used resulted in three distinct experimental conditions for each data set. This provided a fair amount of empirical data to compare between the three data sets and various conditions. Some other parameters were set to constants, such as the number of iterations in the STZSL-T repetition with 10 total iterations, chosen for the sake of run times.

*b) Experimental setting:* Experiments were run in parallel on separate processes on Noah's computer which features a 4.5GHz quad-core processor. The experiments took roughly 11 hours in total to run while going in parallel.

| TL Type | PKT | $\beta$ | Acc. | Runtime |
|---|---|---|---|---|
| I | E | 0 | 68.82% | 4.728$s$ |
| I | E | 50 | 74.07% | 4.727$s$ |
| I | S | 0 | 67.61% | 7.246$s$ |
| I | S | 50 | 74.13% | 10.188$s$ |
| T | E | 0 | 68.49% | 16.328$s$ |
| T | E | 50 | 74.03% | 16.340$s$ |
| T | S | 0 | 67.56% | 79.043$s$ |
| T | S | 50 | 74.12% | 104.799$s$ |

TABLE IV
TL TYPE IS THE TRANSFER LEARNING TYPE, WHERE STZSL-I HAS THE VALUE I AND STZSL-T HAS THE VALUE T. PKT MEANS "PRIOR KNOWLEDGE TYPE", AND E STANDS FOR EMBEDDING VECTORS AND S FOR CLASS SIMILARITIES MATRIX.

*c) Results:* As can be seen in Table IV, the runtime increases monotonically with the Table's y-axis, reaching an average runtime of well over one minute per iteration. However, the increasing runtimes do not show a strong relationship with the accuracies. Instead, the variable that seems to most strongly correllate to accuracy is $\beta$. The quickest average runtime with $\beta = 50$ also happened to be the quickest average runtime overall at $4.727$ seconds, and it gave an average accuracy of $74.07\%$ on all the data sets, only an insignificant $0.06\%$ lower than the most accurate experimental setting. This indicates that the other variables would be better optimized for time by using the STZSL-I with embedding vectors.

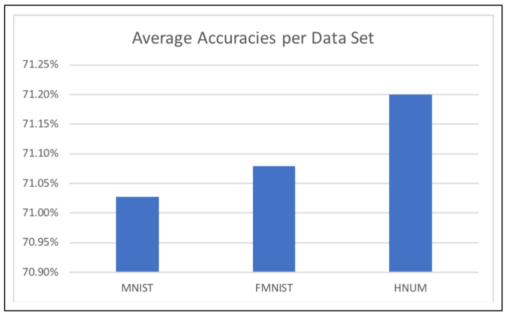

Fig. 3. Each data set's overall average accuracy rate on all ten classes, across all experiments, within each class

As is seen here, the difference between the three data sets is relatively small at $0.17\%$, but the house numbers data set performed best with STZSL by the slight margin.

| Data Set | Class | Acc. |
|---|---|---|
| MNIST | 9 | $73.51\%$ |
| FMNIST | 9 | $72.77\%$ |
| HNUM | 8 | $73.70\%$ |

TABLE V
CLASS WITH BEST AVERAGE ACCURACY ACROSS ALL EXPERIMENTS PER DATA SET

| Data Set | Class | Acc. |
|---|---|---|
| MNIST | 1 | $66.29\%$ |
| FMNIST | 1 | $67.74\%$ |
| HNUM | 1 | $67.83\%$ |

TABLE VI
CLASS WITH WORST AVERAGE ACCURACY ACROSS ALL EXPERIMENTS PER DATA SET

As seen in Tables V and VI, the best accuracies for each data set all lie close to each other, as do the worst. The difference between the worst class for accuracy and the best leads to the possibility that those classes are more transferable and produced less strict classifiers, though it also could be a sign of over-fitting.

Across all experimental settings and all data sets, the algorithm took an average of 30.425 seconds and achieved an average of $71.10\%$ accuracy, an accuracy far better than the $50\%$ chance that would be expected when using random chance since the algorithm trains individual one-vs-all binary classifiers for each class.

## D. Structural Correspondence Learning

*a) Baseline Results:* To get our baseline results, we run SCL on the MNIST, fashion MNIST, and house numbers datasets on 600 examples and 3-fold cross validation with hyperparameters are discussed in section III-D.0.b. We use one dataset and have a source domain of labeled examples and a target domain of unlabeled examples. We get the following results:

TABLE VII
SCL BASELINE PERFORMANCE

| Dataset | MNIST | Fashion MNIST | House Numbers |
|---|---|---|---|
| Accuracy | $0.535 \pm 0.037$ | $0.695 \pm 0.083$ | $0.675 \pm 0.080$ |

Looking at the accuracies per fold, we can say with $95\%$ confidence that SCL performs the same on the fashion MNIST and house numbers datasets, but we can say that SCL performs differently on MNIST than on fashion MNIST and house numbers at the $95\%$ confidence level.

*b) Hyperparameters:* As discussed in section II-F, $h$ is the the dimensionality of our low-rank representation. Ando and Zhang[1] found that there was no significant change in results with $h$ ranging from 20 to 100. They set their $h$ to 50 while Blitzer et al[4] set theirs to 25. Testing on $h = 25, 35, 50$ on the fashion mnist dataset, we also find no difference in accuracy at the $95\%$ confidence level. We choose in all other tests to set our default $h$ to 25.

As discussed in section II-E, $\lambda$ is the square regularization of Huber's robust loss function. Ando and Zhang[1] set their $\lambda$ to $10^{-4}$. Testing on $\lambda = 10^{-3}, 10^{-4}, 10^{-5}$ on the fashion mnist dataset, we find that $\lambda = 10^{-4}$ yields the shortest runtime while having do difference in accuracy from $\lambda = 10^{-3}, 10^{-5}$ at the $95\%$ confidence level. We choose in all other tests to set our default $\lambda$ to $10^{-4}$

Our final hyperparameter $m$, the number of pivot features. Ando and Zhang[1] used any feature that was present at least 50 times in the source and target domain as pivot features. Because our data was continuous, this approach wouldn't have worked. Testing on $m = 1, 2, 3, 4, 5$ on the house numbers dataset, we find no difference in accuracy at the $95\%$ confidence level. However, we find a very strong correlation between runtime and number of pivot features (as can be seen in figure 4) and thus decide to use 1 pivot features for all other tests.

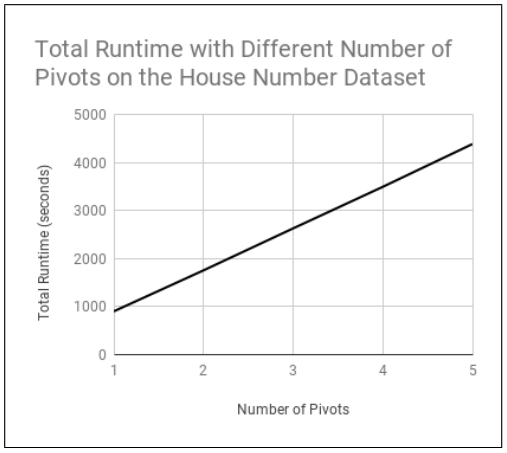

Fig. 4. Runtime from running SCL on $m = 1, 2, 3, 4, 5$ pivot features on the house numbers dataset

*c) Other analyses:* First, we analyzed the source versus target accuracy on the three datasets.

TABLE VIII
SCL SOURCE AND TARGET ACCURACIES

| Dataset | Source Accu. | Target Accu. |
|---|---|---|
| MNIST | $0.655 \pm 0.031$ | $0.535 \pm 0.037$ |
| Fashion MNIST | $0.568 \pm 0.055$ | $0.695 \pm 0.083$ |
| house numbers | $0.603 \pm 0.049$ | $0.675 \pm 0.080$ |

Looking at the accuracies per fold, we find no difference in the accuracy of our classifier trained by SCL when run on the house numbers source or target domain at the $95\%$ confidence level. We do find a significant difference of the accuracies from the other two datasets at the $95\%$ confidence level. In other words, Using SCL, we find that the classifier trained in the source domain does as well or better in the target domain than the source when using the fashion MNIST and house numbers datasets which is a good but somewhat surprising result.

We also analyze the change in target domain accuracy on a classifier using the SCL pivots and one trained on the data as given.

TABLE IX
SCL TARGET ACCURACIES WITH AND WITHOUT PIVOTS

| Dataset | Accu. w/ Pivots | Accu. w/o Pivots |
|---|---|---|
| MNIST | $0.535 \pm 0.037$ | $0.530 \pm 0.042$ |
| Fashion MNIST | $0.695 \pm 0.083$ | $0.710 \pm 0.069$ |
| House Numbers | $0.675 \pm 0.080$ | $0.710 \pm 0.043$ |

Looking at the accuracies per fold, we find no difference between using and not using pivots on any of the datasets at the $95\%$ confidence level. This casts some doubt on the efficacy of SCL, as one would expect that using the pivots would increase accuracy significantly. No comparison like this one was made by Ando and Zhang[1], who instead opted only to compare target domain accuracies with other algorithms.

## E. Self-Taught Clustering

We implemented self-taught clustering to determine its relative validity when compared to other transfer learning algorithms found through a literature survey. Similar to Dai et al., we used SIFT keypoint descriptors to determine the common feature space between the various datasets. These were identified using openCV(`https://opencv.org/`), a robust and well-validated computer vision library implementing many of the state-of-the-art models. Keypoints are important curves or edges in images and are generally found to be corners or boundaries between objects. An example can be seen in image 5.

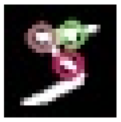

Fig. 5. Example of a number from MNIST with keypoints identified

All the descriptors for a both the source and the target datasets are clustered through *sklearn*'s KMeans clustering in function *build_codebook*. We then create a frequency histogram divided by auxiliary and target data source. This creates our common feature space. Our intial joint probabilities from equation 12 are created in *create_initial_image_feature_joint_probs*. We then create the initial clustering functions using these probabilities. The joint probability by cluster in equation 16 is created in *get_joint_prob_by_cluster*. And finally, we find the matrix in equation 19 in *get_joint_prob_co_cluster*. With this we have all our initialized values and we can run self-taught clustering, which goes through our pseudocode presented in section II-G.

In general, we evaluated runtimes and accuracies by class. We are not able to cluster our algorithm without the auxiliary data, as this would default to normal clustering, as opposed to co-clustering so no analysis of the improvement of the model given the source data to learn concepts in the target data was conducted. We recorded the accuracy of the model clustered on each dataset, and augmented with MNIST, except for the MNIST data set of course, which was augmented with the Fashion-MNIST dataset, to give the best chance of success. Accuracies are reported in table III-E.

The runtimes for these are as reported in table III-E

We then evaluated the two hyperparamters necessary for the STC algorithm. These are the tradeoff parameter $\lambda$ and the number of feature clusters to be used intially by KMeans. Here, we report the findings for scaling the $\lambda$ between $0.5, 1.0, 1.5$ and the number of clusters between $50, 100, 150, 200, 300$.

As we can see generally from the results, the STC classifier did not perform with high accuracy, regardless of hyperparamter tuning or choice of dataset. There could be a number of reasons for this. One such reason is that the images used in the original Caltech-256 dataset were image files. However, do to the fact that

TABLE X
STR ACCURACIES BY CATEGORY

| Category # | Fashion | MNIST | HouseNum |
|---|---|---|---|
| 1 | 0.015 | 0.023 | 0.010 |
| 2 | 0.020 | 0.015 | 0.010 |
| 3 | 0.007 | 0.023 | 0.010 |
| 4 | 0.015 | 0.015 | 0.027 |
| 5 | 0.015 | 0.018 | 0.020 |
| 6 | 0.007 | 0.008 | 0.020 |
| 7 | 0.017 | 0.010 | 0.015 |
| 8 | 0.017 | 0.013 | 0.010 |
| 9 | 0.025 | 0.010 | 0.008 |
| 10 | 0.013 | 0.008 | 0.012 |

TABLE XI
STR RUNTIMES BY DATASET IN SECONDS

| Fashion | MNIST | HouseNum |
|---|---|---|
| 899 (48) | 1015 (195) | 817 (7) |

most of our images were black and white, the selection of the SIFT descriptors could be very limited, with little knowledge. For example, if we select keypoints from black and white images, we generally end up with edges and curves that might look very similar, and even be in the same location, simply by chance. Additionally, do to the low resolution of the images, the SIFT descriptors, as seen in image 5 would have little information in the neighborhood of the keypoint to encode into the descriptor. All these problems are accentuated when we implement the KMeans algorithm to create the codebook, and eventually the STC algorithm as well.

As we look at the change in accuracy as we increase the number of clusters, we see that there is a slight peak at 100, after which we lose accuracy. Using this

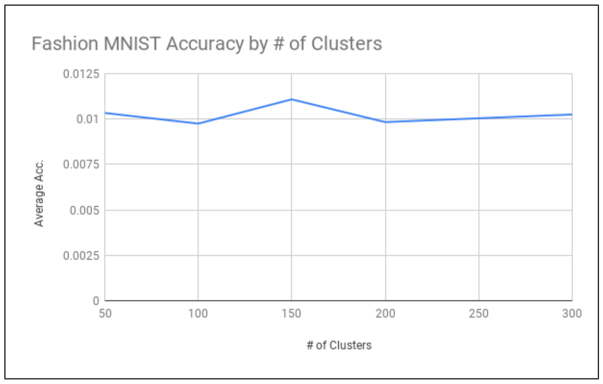

Fig. 6. Average (across categories) cluster accuracy for Fashion MNIST while changing number of feature clusters

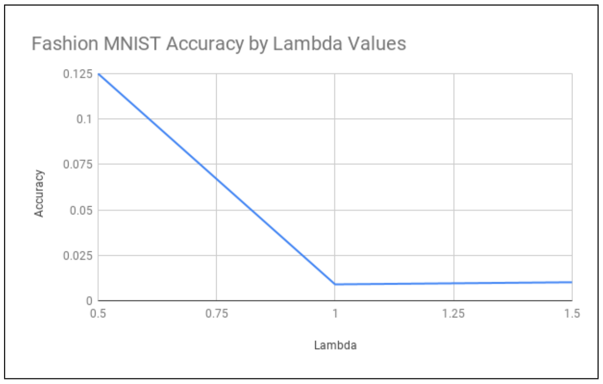

Fig. 7. Average (across categories) cluster accuracy for Fashion MNIST while changing tradeoff lambda hyperparamter

result, the data was run with the optimized 100 cluster value. Moreover, beyond this point, it is possible that the clusters compartmentalize the data too much, and allow for redundant clusters, reducing accuracy when these are used to calculate coclusters. Finally, we see a steep drop in accuracy as we change the lambda parameter. This is possible to the fact that we want less information from the auxiliary dataset, rather than the source dataset. Thus, the best hyperparameter was 0.5. This could be due to the fact that we used wildly different datasets, which could potentially not have any underlying distribution of keypoints, and so our algorithm loses accuracy when trying to cluster both datasets together. In general, these results contradict the original paper's results, and more analysis is needed to determine the differences. The authors were contacted to compare implementation details, and this paper's authors are awaiting correspondence.

## F. Feature Space Remapping

Datasets: House number, MNIST, Fashion MNIST Tests were run on pairs of datasets to test FSM both when the source and target feature spaces were the same and different. 3-fold stratified cross validation was run on the training sets to produce an average accuracy. The training set consisted of 600 entries sampled with replacement, while the test set consisted of 200 entries from a different dataset with bootstrap sampling. GridSearchCV was used to find parameters for an SVM classifier, which were selected from the following hyperparameters:

Kernel : (linear, polynomial, rbf)
Degree : (2,3) , for polynomial kernel
C : (0.01, 0.1, 1, 3) , the trade-off constant

Gamma : (0.001, 0.01, 0.1, 0.5, 1, 2, 3)

The best performing parameters were selected on the source dataset and then used for testing. Results for pairwise experiments were collected as shown in Table XII.

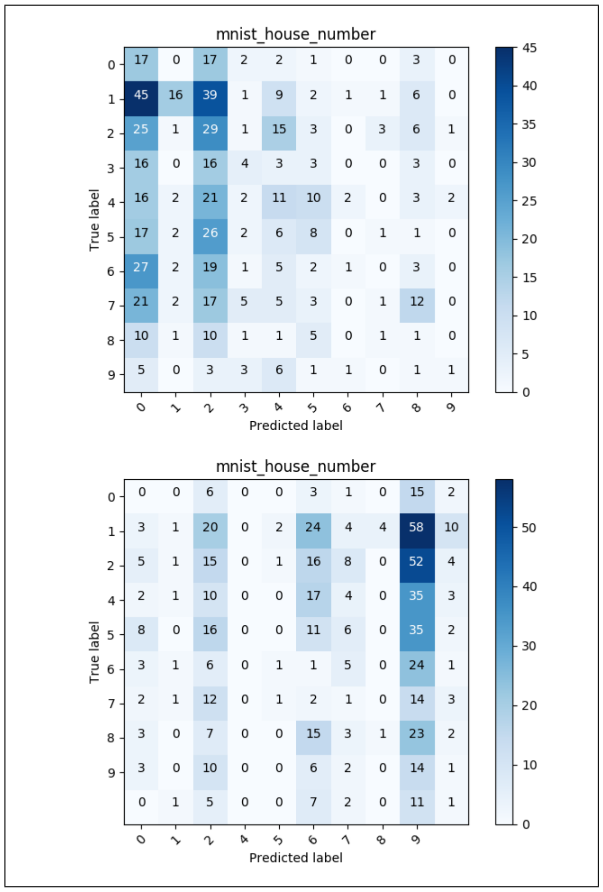

Fig. 8. Feature Space Remapped Learning Performance on House Number

TABLE XII
SVM Accuracy with FSR

| | Mnist | Fashion Mnist | House Number |
|---|---|---|---|
| Mnist | | $10.0 \pm 1.5$ | $14.8 \pm 4.36$ |
| Fashion Mnist | $10.0 \pm 1.5$ | | $5.67 \pm 1.56$ |
| House Number | $10.1 \pm 0.7$ | $6.7 \pm 1.4$ | |

*Results:* Comparing the FSR accuracies of Table XII to the baseline, FSR greatly reduced the performance of the classifier across all datasets. Also, the average accuracy of the algorithm did not differ greatly with respect to the size of the training set.

FSR is an algorithm that is able to use a small amount of data in the source to draw relations to the target domain. However, the performance of SVM using FSR is far less than the baseline. This can be attributed to multiple factors, such as negative transfer, or a poor method of calculating meta-features. Furthermore, the authors suggest the use of $\max()$ as the aggregation protocol for combining image target features that map to the same source feature. Further experimentation found that the $\max()$ aggregation protocol resulted in greater accuracy with SVM. Therefore, we can discount the aggregation protocol as the source of its poor performance. Upon examining the confusion matrices, it becomes apparent that the mapping causes a loss of information such that the classifier leans heavily toward classifying most entries as either $0$ or $2$. Similarly in the comparison of mnist to house number, the classifier classifies most entries as $8$ (shown as $9$ in the figure) and has a high correlation between $1$ and all other labels. This can be attributed to the disparity in complexities between the datasets. Mnist features grayscaled and centered digits, whereas housing number contains multiple numbers and colors per entry. This may cause similarity scores that do not produce helpful mappings. It is also possible that the formula for computing meta-features using only expected values does not capture the complexity of the datasets, causing faulty mappings.

Interestingly, the accuracy values in Table XII resulted in similar accuracy scores and standard deviations as the entry across its diagonal. This can be attributed to the calculation for the similarity scores between source and target features. Notice that the average similarity function sums the difference between two meta-features regardless of the order of $x$ and $y$. Given that the original datasets were sampled with replacement, it would be natural for significant overlap in samples to exist between the source and target datasets. These two factors may cause the similarity scores between features to be similar across multiple tests, causing the same mappings to be generated regardless of what order the datasets are in.

## G. Warm Start (WS)

In the Discussions section, the accuracy stated for warm start was for an MLP trained over a treatment set of size 600, with initial weight matrix $\theta_c$, where $\theta_c$ refers to the weight matrix of the neural network trained over the control (untreated) data. The experiments we performed for warm start specifically use three separate networks. The first network, which we will refer to as $\pi_c$ (control-trained network), was trained over the control data with a randomized initial weight matrix. The finalized weight matrix of this network will be referred to as $\theta_c$. The second network, $pi_w$ (warm-start network trained over treatment), was trained over the treatment data with an initial weight matrix equal to $\theta_c$. The finalized weight matrix of this network will be referred to as $\theta_w$. The last network, which we will refer to as $\pi_t$ (treatment-trained network), was trained over the treatment data with a randomized initial weight matrix. The finalized weight matrix of this network will be referred to as $\theta_t$.

For each data set MNIST, Fashion MNIST, and House Numbers, we performed 10 experiments. In each experiment, 2400 labeled examples were obtained from the relevant data set for training, as well as 400 labeled examples for testing. The training and test sets are then split into two sets of 1200 and 200 examples respectively. The first of these sets are the control training and test sets, and the others are the treatment training and test sets. The data sets comprise of $28x28$ images strung out into a vector of length 784, with 10 possible class labels for each example. The treatment we applied to the data to obtain a treatment set was a wrap-around shift of size 100 to the image vector.

Then $\pi_c$ was randomly initialized and trained over the control set, $\pi_w$ was initialized with $\theta_c$ and trained over the treatment set, and $\pi_t$ was randomly initialized and trained over the treatment set. To calculate accuracies, each network was then tested on both the control and treatment test sets. The accuracies for each network over each set are then averaged over the 10 experiments. We also kept track of the number of iterations each network took to finalize its weights. These values are displayed in the tables below.

TABLE XIII

MNIST Performance Metrics (Nearest $1000_{th}$)

| | $\pi_c$ | $\pi_w$ | $\pi_t$ |
|---|---|---|---|
| Control | 0.795 | 0.825 | 0.815 |
| Treatment | 0.805 | 0.860 | 0.835 |
| Iterations | 2150 | 1081 | 2309 |

*1) Iteration Comparison:* For the MNIST and Fashion MNIST experiments, the average iterations for $\pi_c$ and $\pi_t$ were very comparable. This makes sense because the two networks are both starting randomized initial $\theta$'s, and are both mapping the domain of data of the same size (treated and control still occupy the same space in memory) to the target labels. However,

TABLE XIV

Fashion MNIST Performance Metrics (Nearest $1000_{th}$)

| | $\pi_c$ | $\pi_w$ | $\pi_t$ |
|---|---|---|---|
| Control | 0.795 | 0.810 | 0.785 |
| Treatment | 0.840 | 0.835 | 0.79 |
| Iterations | 2003 | 1188 | 2004 |

TABLE XV

House Numbers Performance Metrics (Nearest $1000_{th}$)

| | $\pi_c$ | $\pi_w$ | $\pi_t$ |
|---|---|---|---|
| Control | 0.260 | 0.372 | 0.312 |
| Treatment | 0.273 | 0.360 | 0.305 |
| Iterations | 1428 | 1984 | 2100 |

$\pi_w$ needed significantly less iterations on average to converge to the final weight matrix.

For the House Numbers experiments, $\pi_c$ had significantly less iterations than $\pi_w$ and $\pi_t$. The iterations for $\pi_w$ and $\pi_t$ were not significantly different. This goes against the iteration data from MNIST and Fashion MNIST. This could be due to the difference in domains causing the shift treatment we applied to change the House Numbers data significantly more than it changed the MNIST and Fashion MNIST data. Therefore, even though $\pi_w$ was initialized with $\theta_c$, the drastic change in data meant that $\theta_c$ was not significantly closer to the final weight matrix $\theta_w$ as compared to a randomized initial weight matrix.

From this, we can conclude that $\pi_w$ will not use significantly more iterations to converge to $\theta_w$ than $\pi_t$ will use to converge to $\theta_t$. Therefore, warm start has the potential to significantly reduce learning time for neural networks.

*2) Accuracy Comparison:* For this section, we will be analyzing the accuracies of $\pi_w$, $\pi_c$, and $\pi_t$, and comparing them to one another. The CATE estimation for this will be the differences in accuracies between $\pi_w$ and $\pi_c$, and $\pi_w$ and $\pi_t$. The goal is to minimize $(\pi_t - \pi_c) - (\pi_w - \pi_c)$ for the control test sets and $(\pi_c - \pi_t) - (\pi_w - \pi_t)$ for the treatment test sets. If this is true, it means $\pi_w$ improved performance over the set relative to $\pi_i$ (where $i = c$ if working with the control set and $i = t$ if working with the treatment set) as compared to how $\pi_i$ performed relative to $\pi_{i'}$ (where $i'$ refers to $c$ if $i = t$ and refers to $t$ when $i = c$). However, this does not take into account the interesting cases where the networks do not perform as expected on the control and treatment sets. For example, $\pi_c$ performing better than $\pi_t$on the treatment set means that it is actually better for $\pi_w$ to be closer to $\pi_c$ for the treatment set than it is for $\pi_w$ to be closer to $\pi_t$.

TABLE XVI
MNIST CATE

| | $(\pi_t - \pi_c) - (\pi_w - \pi_c)$ | $(\pi_c - \pi_t) - (\pi_w - \pi_t)$ |
|---|---|---|
| Control | $-0.01$ | |
| Treatment | | $-0.055$ |

*a) MNIST:* For the MNIST data set, we can also see that in all cases, $\pi_w$ performed relatively better on both control and treatment, since all the values in the table are negative. In fact, $\pi_w$ outperformed both $\pi_c$ and $\pi_t$ on both the control and treatment test sets. Interestingly, $\pi_c$ performed worse on the control set than $\pi_t$, as well as $\pi_c$ performing worse on the control set that it did on the treatment set. However, since $\pi_w$ is more accurate than $\pi_t$ over the control set and is more accurate than $\pi_c$ over the treatment set, we can say that for MNIST, $\pi_w$ has transferred the knowledge over the control set to the treatment set successfully.

TABLE XVII
FASHION MNIST CATE

| | $(\pi_t - \pi_c) - (\pi_w - \pi_c)$ | $(\pi_c - \pi_t) - (\pi_w - \pi_t)$ |
|---|---|---|
| Control | $-0.025$ | |
| Treatment | | 0.005 |

*b) Fashion MNIST:* For the Fashion MNIST data set, $\pi_w$ outperformed both $\pi_c$ and $\pi_t$ on the control set. However, $\pi_c$ performed the best over the treatment set, and did better on the treatment set than it did on the control set, which should not happen. For the control set, we can say $\pi_w$ definitely performed better relative to $\pi_t$. For the treatment set, while $\pi_c$ still performed the best, $\pi_w$ outperformed $\pi_t$, so using warm start transfer learning still cannot hurt performance as compared to training a new network from scratch.

TABLE XVIII
HOUSE NUMBERS CATE

| | $(\pi_t - \pi_c) - (\pi_w - \pi_c)$ | $(\pi_c - \pi_t) - (\pi_w - \pi_t)$ |
|---|---|---|
| Control | $-0.06$ | |
| Treatment | | $-0.087$ |

*c) House Numbers:* For the House Numbers data set, it should be noted that the networks overall performed poorly, with accuracies significantly under acceptable margins. We still compared performances of $\pi_w$, $\pi_c$, and $\pi_t$, but the conclusions drawn from House Numbers does not carry the same weight in our overall conclusions as the conclusions from MNIST and Fashion MNIST.

For the House Numbers data set, $\pi_w$ outperformed both $\pi_c$ and $\pi_t$ on both the control and treatment test sets. We again see unexpected accuracies, as $\pi_c$ performed better on the treatment set than it did on the control set, and $\pi_t$ performed better on the control set than the treatment set, while also performing better than $\pi_c$ on the control set. However, since $\pi_w$ definitively did better than both $\pi_c$ and $\pi_t$ on both sets, we can say that it has successfully transferred knowledge over the control set to the treatment set successfully.

*d) Overall Conclusions for Warm Start:* While our networks did not always perform as expected on the data sets, we can still conclude that using warm start has a strong likelihood of reducing network learning time if the difference between the control and treatment domains is not drastic, and can transfer knowledge from the control domain to the treatment domain since it has comparable accuracy to $\pi_t$ over the treatment sets . An interesting note is that $\pi_w$ was always fairly accurate (relatively more accurate for House Numbers) over both the control domain and the treatment domain. From this, we can conclude that utilizing warm start for neural networks can possibly result in the network performing well on both the control and treatment domains, instead of just one of the domains. However, this is not a solid conclusion since $\pi_c$ and $\pi_t$ still performed comparable to one another on both the control and treatment domains.

## IV. DISCUSSION

To compare our three algorithms, we ran them on the house numbers, fashion MNIST, and MNIST datasets using 600 examples and 3-fold cross validation. The results were as follows:

TABLE XIX
ACCURACIES FOR ALL ALGORITHMS

| Algorithim | House Numbers | MNIST | Fashion MNIST |
|---|---|---|---|
| STL | 0.242 | 0.828 | 0.745 |
| STZSL | 0.712 | 0.710 | 0.710 |
| SCL | 0.675 | 0.535 | 0.695 |
| STC | 0.009 | 0.009 | 0.010 |
| FSR | 0.022 | 0.012 | 0.103 |
| WS | 0.221 | 0.753 | 0.730 |

To analyze the data, we follow the process outlined by Demšar[7] starting by ranking the classifiers returned by our algorithms on all three datasets and calculating their average ranks. We then perform a Friedman test on the average ranks to check if the they are significantly different than the average rank. With $p = .05$, we conclude that the average ranks are significantly different.

We continue by doing a Nemenyi test. At $p = .05$, we find that no classifiers are critically different from each other. In contrast, at $p = .1$ (which with 6 algorithms and 3 datasets has a critical difference of 3.95), we find that STL and STZL are critically different than FSR and STC.

We then do a Bonferroni-Dunn test using STC as our control. With $p = .05$ we have a critical difference of 3.93 leading us to conclude that STL and STZL lie outside the critical difference range of STC.

These results are summarized in the critical difference diagrams in Figure 9.

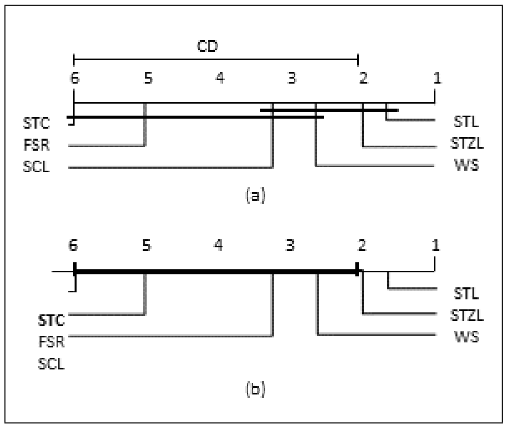

Fig. 9. (a) Comparison of all classifiers against each other with the Nemenyi test. Groups of classifiers that are not significantly different (at $p = 0.10$) are connected. (b) Comparison of STC against the others with the Bonferroni-Dunn test. All classifiers with ranks outside the marked interval are significantly different ($p < 0.05$) from the control.

We further compare the algorithms to STC using corresponding statistics and $p$ values. Our standard error was $SE$. Our results are shown in Table XX where $R_i$ refers to the average rank of the classifier returned by algorithm $i$, $\alpha$ is .05, and $p$ is calculated from the $z$ score using two tails on the stardard normal curve.

Unfortunately, all the step-up and step-down procedures (Holm, Hochberg, and Hommell) were unable to further elaborate on the results from the previous tests.

Overall, we conclude that STL and STZL are critically different than FSR and STC while the experimental data is not sufficient to reach any conclusion regarding SCL and WS. To better detect differences in the algorithms, we recommend further testing on more datasets to decrease the critical difference for both tests.

TABLE XX
STATISTICS FOR STEP-UP AND STEP-DOWN PROCEDURES

| $i$ | Classifier | $z = \frac{R_0 - R_i}{SE}$ | $p$ | $\frac{\alpha}{i}$ |
|---|---|---|---|---|
| 1 | STL | 2.837 | 0.0045 | 0.01 |
| 2 | STZL | 2.619 | 0.0088 | 0.0125 |
| 3 | WS | 2.1821 | 0.0291 | 0.0167 |
| 4 | SCL | 1.7457 | 0.0809 | 0.025 |
| 5 | FSR | 0.6547 | 0.5127 | 0.05 |

## V. SUMMARY

As a whole, our team covered the three major areas of transfer learning as per the seed paper [16]. A variety of algorithms blurred the lines between the three categories, such as STZSL from section II-C having STZSL-I and STZSL-T, neither of which perfectly met the criterion for inductive or transductive as per [16].

Throughout our experiments, it was clear that the accuracy of these algorithms could attain very impressive rates with a small sample sizes. While these data sets were relatively simple, the research that we read suggests that transfer learning algorithms can be used on highly complex tasks and domains. The papers that we read spanned more than a decade and showed a wide variety of applications and perspectives, including the use of a large set of possible classifiers.

In general, there is significant improvement to still be made in transfer learning. As mentioned previously, due to the high difficulty and cost of developing curated gold-standard training data, the ability to transfer knowledge from already developed datasets to new ones would significantly increase the ability of machine learning methods to solve more challenging problems. One significant component to transfer learning that wasn't evaluated in this paper requires further study. The concept of negative transfer is the ability of auxiliary data to negatively impact the learning of ML models. This can happen when the assumption that the source and target data domains are not related in some way, particularly for a specific learning algorithm. If these assumptions do not hold, the auxiliary data simply introduces noise into the system, reducing performance. We must first better further study transferability to better understand negative transfer. We must also better define the metrics to measure similarity between domains and tasks.

Another interesting potential future direction of transfer learning is the ability to transfer knowledge between feature spaces. For instance, transferring the knowledge we have about image categorization into text classification has very interesting applications to multi-model ML, such as multi-model querying and information retrieval. In general, many of the more complicated tasks to more completely understand the world around us would greatly benefit from additional study in transfer learning.

## VI. Contributions

Each member of the team read two papers and implemented an algorithm from one of them. Each person also wrote a section dedicated to each paper they read and described their experiments for the algorithm that they implemented.

### A. Connor Baumler

Connor read the papers “A framework for learning predictive structures from multiple tasks and unlabeled data”[1] and “Domain adaptation with structural correspondence learning”[4] and discussed them in sections II-E and II-F. He implemented SCL and tested it in section III-D. He wrote the get folds method that most members of the group used in their cross validation. He did the statistical testing for section IV including tests such as the Friedman, Nemenyi, and Bonferroni-Dunn tests and creating the critical difference diagrams. To do this, he read “Statistical comparisons of classifiers over multiple data sets”[7].

### B. Nikhil Chakravarthy

Nikhil implemented the Warm-Start algorithm for a neural network, as well as read the papers “Transfer Learning for Estimating Causal Effects using Neural Networks”[12] and “Deep Learning of Representations for Unsupervised and Transfer Learning”. Nikhil ran experiments on each of the three datasets with k-fold stratified cross validation on each.

### C. Noah Crowley

Noah read the papers for “Sample Transfer Zero-Shot Learning”[11] (STZSL) and “Semi-Supervised Zero-Shot Classification with Label Representation”[14] (ZSWLR) Learning. He implemented STZSL and ran 24 experiments each with three fold stratified cross validation.

### D. Kha Dinh Luong

Kha Dinh wrote the functions used to get the data for all three data sets that all team members used.

Kha Dinh read the papers for “Multitask Feature Learning”[2] and “Self-taught Learning”[17]. He implemented “Self-taught Learning” and conducted experiments to determined the baseline performance on each of the 3 datasets, the performance of self-taught learning using 1 dataset as the source domain and another dataset as the target domain (6 pairs), and the effect of varying the amount of source domain data on the classification performance in the target domain.

### E. Toshiki Nazikian

Toshiki Nazikian read the papers for “Translated Learning”[5] and “Feature Space Remapping”[10]. He implemented Feature Space Remapping, and conducted experiments to compare the baseline performance to his algorithm. Each of the six experiments was run using 3-fold stratified cross validation.

### F. Vimig Socrates

Vimig Socrates generally focused his implementation and deep analysis in unsupervised transfer learning. Therefore, he read a papers “Self-taught Clustering” and “Transferred Dimensionality Reduction”. He implemented Self-taught Clustering and conducted experiments varying the various hyperparameters (e.g. number of clusters, impact of source data on target data) on all three datasets using 3-fold stratified cross validation. External libraries were used to develop features, but core algorithm and evaluation was written from scratch.

## References

[1] Rie Kubota Ando and Tong Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6(Nov):1817–1853, 2005.

[2] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Multi-task feature learning. In *Advances in neural information processing systems*, pages 41–48, 2007.

[3] Yoshua Bengio. Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pages 17–36, 2012.

[4] John Blitzer, Ryan McDonald, and Fernando Pereira. Domain adaptation with structural correspondence learning. In *Proceedings of the 2006 conference on empirical methods in natural language processing*, pages 120–128. Association for Computational Linguistics, 2006.

[5] Chen Y. Xue G. R. Yang Q. & Yu Y. Dai, W. Translated learning: Transfer learning across different feature spaces. *In Advances in neural information processing systems*, pages 3537–360, 2009.

[6] Wenyuan Dai, Qiang Yang, Gui-Rong Xue, and Yong Yu. Self-taught clustering. In *Proceedings of the 25th international conference on Machine learning*, pages 200–207. ACM, 2008.

[7] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan):1–30, 2006.

[8] Inderjit S Dhillon, Subramanyam Mallela, and Dharmendra S Modha. Information-theoretic co-clustering. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 89–98. ACM, 2003.

[9] Li Fei-Fei and Pietro Perona. A bayesian hierarchical model for learning natural scene categories. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 524–531. IEEE, 2005.

[10] Cook D. J. Feuz, K. D. Transfer learning across feature-rich heterogeneous feature spaces via feature-space remapping (fsr). *ACM Transactions on Intelligent Systems and Technology*, 6(1), 2017.

[11] Yuchen Guo, Guiguang Ding, Jungong Han, and Yue Gao. Zero-shot learning with transferred samples. *IEEE Transactions on Image Processing*, 26(7):3277–3290, 2017.

[12] Sören R Künzel, Bradly C Stadie, Nikita Vemuri, Varsha Ramakrishnan, Jasjeet S Sekhon, and Pieter Abbeel. Transfer learning for estimating causal effects using neural networks. *arXiv preprint arXiv:1808.07804*, 2018.

[13] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[14] Xin Li, Yuhong Guo, and Dale Schuurmans. Semi-supervised zero-shot classification with label representation learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4211–4219, 2015.

[15] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 5, 2011.

[16] Sinno Jialin Pan, Qiang Yang, et al. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[17] Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer, and Andrew Y Ng. Self-taught learning: transfer learning from unlabeled data. In *Proceedings of the 24th international conference on Machine learning*, pages 759–766. ACM, 2007.

[18] Zheng Wang, Yangqiu Song, and Changshui Zhang. Transferred dimensionality reduction. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 550–565. Springer, 2008.

[19] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.