Source-Target-Source Classification Using Stacked Denoising Autoencoders

Chetak Kandaswamy[1,2,3](✉), Luís M. Silva[2,4], and Jaime S. Cardoso[3]

[1] Instituto de Investigação e Inovação em Saúde, Universidade do Porto, Porto, Portugal
chetak.kand@gmail.com
[2] INEB - Instituto de Engenharia Biomédica, Porto, Portugal
[3] INESC TEC and Faculdade de Engenharia, Universidade do Porto, Porto, Portugal
[4] Departamento de Matemática, Universidade de Aveiro, Porto, Portugal

Abstract. Deep Transfer Learning (DTL) emerged as a new paradigm in machine learning in which a deep model is trained on a source task and the knowledge acquired is then totally or partially transferred to help in solving a target task. Even though DTL offers a greater flexibility in extracting high-level features and enabling feature transference from a source to a target task, the DTL solution might get stuck at local minima leading to performance degradation-negative transference-, similar to what happens in the classical machine learning approach. In this paper, we propose the Source-Target-Source (STS) methodology to reduce the impact of negative transference, by iteratively switching between source and target tasks in the training process. The results show the effectiveness of such approach.

Keywords: Deep neural network · Transfer learning · Optimization

1 Introduction

Transfer learning is an approach in which the knowledge acquired by a machine trained to solve a task is applied with minor modifications to solve a new target task without having to follow the whole training procedure. It is anticipated that new tasks and concepts are learned more quickly and accurately by exploiting past knowledge. In the past, a variety of transfer learning tasks have been investigated, including lifelong learning [1], multi-task learning [2], cross-domain learning [3], self-taught learning [4], and *deep transfer learning* (DTL) [5,8] to name a few. We investigate the DTL approach proven to be successful in object recognition and image recognition problems using a layer-by-layer feature transference on large-scale data in either a supervised [5] or a unsupervised [6] setting.

C. Kandaswamy—This work was financed by FEDER funds through the *Programa Operacional Factores de Competitividade* COMPETE and by Portuguese funds through FCT Fundação para a Ciência e a Tecnologia in the framework of the project PTDC/EIA-EIA/119004/2010. We thank Faculdade de Engenharia, Universidade do Porto.

R. Paredes et al. (Eds.): IbPRIA 2015, LNCS 9117, pp. 39–47, 2015.
DOI: 10.1007/978-3-319-19390-8_5

All these methods have shown that there is a limitation on choosing the source problem that would offer good features to solve the new target problem. Even though the problem existed for more than a decade, very few viable solutions have appeared to deal with the problem of negative transference, that is when the knowledge leads to a lower performance on the target problem than the no-transference approach. In the case of neural networks, negative transference occurs due to two reasons: (1) specialization of higher layer neurons to the source problem [7] and (2) fragile co-adaptation of neurons is broken by splitting of transferred layer and randomly initialized layer leads to difficulty in optimization [8]. It is observed either of these two reasons may dominate, depending on whether features are transferred from the bottom, middle, or top of the network. It is observed that the bottom-layer features are standard [7,8] regardless of the cost function or dataset used, called as *general,* similarly the top-layer features depend greatly on the chosen dataset and task, called as *specific*. We may therefore pose the following questions:

- If feature transference is performed, should we transfer general or specific features?
- If feature transference is performed, can we avoid or minimize negative feature transference?

Generally problems change with environment, thus providing only a few training observations to solve the problem. In this paper, we analyse such questions and propose the Source-Target-Source (STS) approach and study the performance STS for two cases: (1) using a few number of examples, and (2) using complete data. As we will see, STS not only effectively reduces the issue of negative transference as well as improves performance over the positive transference situations.

2 Baseline, Transfer Learning and STS Approaches

Let's represent a dataset by a set of tuples $D = (x_n, y_n) \in X \times Y$, where X is the input space and Y is a set of labels. Assume that the n instances are drawn by a sampling process from the input space X with a certain probability distribution $P(X)$. The dataset is split into subsets of training, validation and test sets, $D = \{(x_n, y_n), (x_v, y_v), (x_m, y_m)\}$ drawn from the same distribution $P(X)$. We assume that the "source" dataset D_S with input space X_S and a set of labels Y_S is drawn from a distribution $P_S(X)$ and "target" dataset D_T with input space X_T and a set of labels Y_T is drawn from a distribution $P_T(X)$. Such $P_S(X)$ and $P_T(X)$ may be equal or different.

Baseline (BL): Stacked Denoising Autoencoders (SDA) are multiple layer networks where each one is trained as a denoising autoencoders (dA) (see Fig. 1-BL). SDA training comprises of two stages: an unsupervised pre-training stage followed by a supervised fine-tuning stage. During pre-training (PT), the network is generated by stacking multiple dA one on top of each other thus learning *unsupervised features*, represented as a vector $U(w)$ of optimal weights and

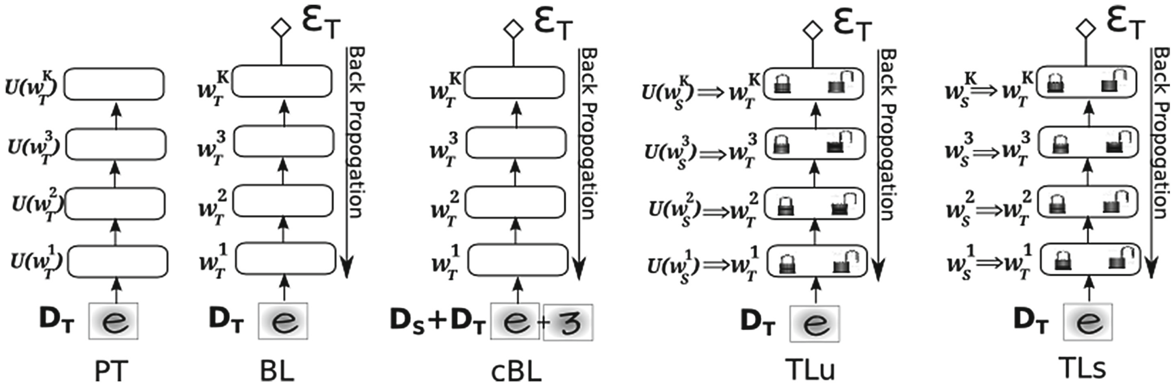

Fig. 1. A pictorial representation of approaches: Pre-training (PT), Baseline (BL), Combined Baseline (cBL), TL unsupervised (TLu),and TL supervised (TLs).

biases. Then a logistic regression layer is added on top and the whole network is fine-tuned (FT) in a supervised way. Thus learning *supervised features* $\mathbf{w} = (w^1, ..., w^K)$, where K is the number of layers.

Combine Baseline (cBL): Given D_S and D_T, a cBL classifier is any function $g(x)$ that is trained from a random combination of instances from $x_S \in X_S$ and $x_T \in X_T$ and then training the SDA to solve for target task Y_T.

Transfer Learning (TL): We first train the source network with source data D_S and Y_S and then copy its hidden layers to the target network. In case $Y_S \neq Y_T$, then we add a classifier layer randomly initialized. The network is trained towards the target task Y_T. We have a choice to fine-tune this entire network $\mathbf{w}_T$ as a multi-layer perceptron using back-propagation or *lock* a layer, meaning the transferred feature from source network $\mathbf{w_S^1} \Rightarrow \mathbf{w_T^1}$ do not change during the error propagation for the target task. This opens up several possible approaches to solve a problem as shown in the Fig. 1 TLu and TLs, where the layers are optionally locked or unlocked. This causes fragile co-adaptation of neurons between layers leading to optimization difficulties. The choice of whether or not to fine-tune the first layer of the target network or not depends on the size of the target dataset and number of parameters [6]. When the performance of the newly trained target network exceeds the performance of the baseline approach we have positive transference; otherwise we have negative transference.

Transfer Learning Unsupervised (TLu): We transfer the unsupervised features of the SDA model from the source to the target network, i.e., $U(\mathbf{w}_S) \Rightarrow \mathbf{w}_T$ as depicted in Fig. 1 TLu. Once the features are transferred to the target network, we add a logistic regression layer for the target task Y_T. Then we fine-tune the entire classifier like a regular multi-layer perceptron with back-propagation choosing to lock or unlock certain layer to solve the target task.

Transfer Learning Supervised (TLs): This is same as supervised layer based approach (SSDA) [7] where we train on the network with BL approach and then we transfer features from source to target network. By selecting to lock or unlock certain layer to solve the target task as illustrated in Fig. 1 TLs.

Algorithm 1. Pseudocode for STS

```
Initialize with trained features $D_T$:
Two datasets $D_S$ and $D_T$, with tasks $Y_S$ and $Y_T$ are drawn from $P_S$ and $P_T$ distributions.
Select $D_S$ dataset to train
baseline: train network $A$ as shown in the baseline approach
Set value to max cycles
list of max cycles errors to zero
for M in max cycles do
    transfer: transfer features from network $A$ to new network $B$ as shown in the transfer learning approach
    update errors list with best test error
    if cycle = odd number then
        STS M = test error for Dataset $D_S$
    else
        STS M = test error for Dataset $D_T$
    end if
    if error < avg(errors list) then
        BREAK
    end if
    Switch between dataset $D_S$ and $D_T$
end for
```

2.1 Source-Target-Source Framework

In this paper we propose a STS approach[1]. The main idea of transfer learning is that the knowledge (features) learnt in a source domain provide a good initialization for the learning task in a target problem, better than starting the learning in the target domain at random (likely to get stuck in a poor local optimum). In here we propose to iterate the learning between both domains. The intuition is that, like in typical metaheuristics in optimization (i.e. tabu search and simulated annealing), moving the learning from one domain to the other will 'shake' the current local optimal solution, allowing us to keep exploring the space of solutions (ideally, allowing us to reach a better solution in the process). Likewise the metaheuristics in optimization, we keep track of the solutions reached in each iteration, and the outputted solution is the best of all. The pseudo-code for the STS process is listed in Algorithm 1.

Evaluation: We are interested in measuring the improvement using the transferred features over random initialization. We use relative improvement as a measure for comparing the performance of baseline over transfer approach.

$$relative\ improvement = \frac{Baseline\ Avg.\ error\ rate - compared\ method\ Avg.\ error\ rate}{Baseline\ Avg.\ error\ rate}$$

3 Experimental Setup and Results

We evaluate the framework in two different settings for the character recognition task: We use the *MNIST* dataset P_L which has 60,000 training and 10,000 testing instances with labeled hand-written digits from 0 to 9. Additionally, the Chars74k dataset was modified to obtain *Lowercase* dataset P_{LC} labelled lowercase letters from a-to-z and, the *Uppercase* dataset P_{UC} labelled uppercase letters from A-to-Z. Both lowercase and uppercase dataset have 19,812 training and 6,604 testing instances.

[1] The naming 'source' and 'target' is some what misleading in our learning framework.

For object recognition tasks: We generated three shapes datasets each with 6,000 training and 14,000 testing instances. First, the *canonical* dataset P_{Sh1} have canonical objects, i.e., equilateral triangle, circle and square. Second, the *non-canonical* dataset P_{Sh2} have non-canonical objects, i.e., triangle, ellipse and rectangles. Finally, the *curve Vs. corner* dataset P_{Sh3} have objects shapes with a curved surface or a corner. All the datasets[2] used in our experiments have images with 28 x 28 pixels and a sample of each dataset are shown in Fig. 2.

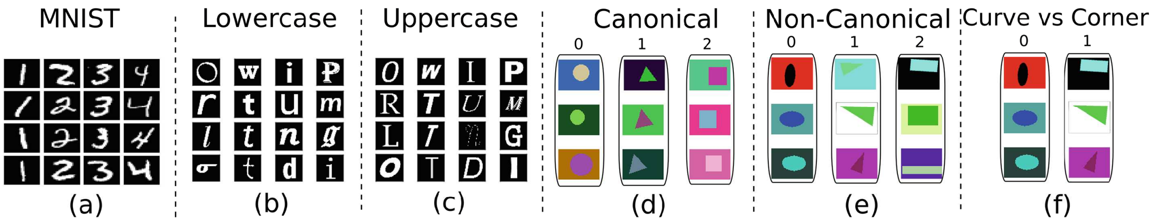

Fig. 2. Samples from character recognition tasks: (a) digits P_L, (b) lowercase P_{LC} and, (c) uppercase P_{UC}; Samples from shape recognition tasks: (d) Canonical P_{sh1}, (e) Non-Canonical P_{sh2} and (f) curve vs. corner P_{sh3}

Training Deep Neural Network: The network we used in character recognition experiments had three hidden layers with [576, 400, 256] units i.e., and the networks used in object recognition experiments also had three hidden layers with [100, 200, 300] units, in order of [bottom, middle, top] respectively. Both networks have an output layer appropriate to the number of classes being considered. All hidden layers were pre-trained as denoising autoencoders via gradient descent, using the cross-entropy cost and a learning rate of 0.001. Pre-training ran for a minimum of 50 epochs in the case of character recognition tasks, and for a minimum of 60 epoch when using object recognition tasks. The complete networks were fine-tuned via gradient descent, using the cross-entropy cost and a learning rate of 0.1. The fine-tuning ran until the validation error did not decrease below 0.1 % or until 1000 epochs for all tasks. Our code for experiments was based on the Theano library 6 and ran with the help of a GTX 770 GPU.

3.1 Transferring Specific Vs. Generic for STS Approach

In this experiment, we intentionally set adverse configurations for feature transference, to study the two main causes of negative feature transference. First, by transferring specific features on tasks that are different, $Y_S \neq Y_T$ we focus on feature specialization in tasks 1 to 4 as listed in Table 1. Second, by transferring generic features on distribution that are similar, we focus on splitting of

[2] We would like to acknowledge researchers making available their datasets, Center for Neural Science, New York University for MNIST; Microsoft Research India for Chars74k; and LISA labs, University of Montreal, Canada for BabyAI shapes.

Table 1. Comparison of percentage average error rate ($\overline{\varepsilon}$) for BL, cBL, TLu, TLs and STS approach for different ratios of target data (P_T) reusing source (P_S) distribution. Tasks 1 to 4 study *specific* feature transfer on character recognition problem and tasks 5 & 6 study *generic* feature transfer on object recognition problem.

	Approach			Ratio of total number of training samples							
		P_T	P_S	0.05	0.1	0.2	0.3	0.4	0.5	1	#
Characters Tasks are different	BL	P_L		6.4 (0.1)	4.7 (0.1)	3.3 (0.1)	2.7 (0.2)	2.3 (0.0)	2.3 (0.4)	1.5 (0.1)	
	TLu	P_L	P_{LC}	7.4 (0.2)	5.3 (0.1)	3.8 (0.2)	3.5 (0.7)	2.7 (0.2)	2.5 (0.2)	2.3 (0.0)	①
	TLs	P_L	P_{LC}	7.4 (0.1)	5.8 (0.2)	4.6 (0.2)	3.7 (0.0)	3.2 (0.2)	2.9 (0.1)	2.1 (0.1)	
	STS	P_L	P_{LC}	**2.6 (0.1)**	**2.1 (0.0)**	**2.0 (0.1)**	**1.9 (0.1)**	**1.8 (0.0)**	**1.7 (0.1)**	**1.5 (0.0)**	
	BL	P_L		6.4 (0.1)	4.7 (0.1)	3.3 (0.1)	2.7 (0.2)	2.3 (0.0)	2.3 (0.4)	1.5 (0.1)	
	TLu	P_L	P_{UC}	7.4 (0.3)	5.6 (0.6)	4.0 (0.2)	3.1 (0.1)	2.8 (0.2)	3.0 (0.5)	2.1 (0.3)	②
	TLs	P_L	P_{UC}	7.6 (0.3)	5.8 (0.2)	4.4 (0.2)	3.5 (0.0)	3.1 (0.0)	2.7 (0.0)	2.0 (0.1)	
	STS	P_L	P_{UC}	**2.4 (0.0)**	**2.2 (0.2)**	**1.9 (0.0)**	**1.7 (0.1)**	**1.7 (0.1)**	**1.6 (0.1)**	**1.5 (0.0)**	
	BL	P_{LC}		17.1 (0.1)	13.3 (0.2)	10.8 (0.1)	9.5 (0.1)	8.4 (0.1)	7.7 (0.6)	4.8 (0.1)	
	TLu	P_{LC}	P_L	17.1 (0.6)	13.8 (0.6)	10.9 (0.2)	9.2 (0.4)	8.2 (0.4)	7.2 (0.2)	**4.7 (0.2)**	③
	TLs	P_{LC}	P_L	18.9 (0.2)	14.6 (0.8)	11.3 (0.2)	9.6 (0.2)	8.7 (0.4)	7.5 (0.2)	5.3 (0.3)	
	STS	P_{LC}	P_L	**12.3 (0.3)**	**9.7 (0.0)**	**8.5 (0.6)**	**7.2 (0.4)**	**6.7 (0.2)**	**6.0 (0.1)**	5.0 (0.2)	
	BL	P_{UC}		16.2 (0.2)	12.9 (0.2)	10.4 (0.2)	9.1 (0.1)	8.5 (0.7)	7.3 (0.5)	4.9 (0.2)	
	TLu	P_{UC}	P_L	15.9 (0.3)	13.2 (0.4)	10.8 (0.3)	9.1 (0.3)	8.0 (0.1)	7.4 (0.3)	**4.6 (0.1)**	④
	TLs	P_{UC}	P_L	16.5 (0.3)	13.6 (0.5)	10.8 (0.2)	9.2 (0.2)	8.5 (0.2)	7.4 (0.1)	5.0 (0.2)	
	STS	P_{UC}	P_L	**10.8 (0.4)**	**9.1 (0.1)**	**7.8 (0.2)**	**6.8 (0.1)**	**6.6 (0.1)**	**6.1 (0.1)**	4.7 (0.1)	
Objects Tasks are similar	BL	P_{Sh2}		37.9 (10.2)	36.6 (4.8)	25.1 (3.6)	16.9 (9.6)	14.7 (7.8)	11.9 (7.1)	**4.2 (2.3)**	
	cBL	P_{Sh2}	P_{Sh1}	28.7 (6.3)	13.6 (2.2)	12.6 (10.4)	9.9 (8.0)	6.6 (3.0)	13.0 (8.4)	10.6 (6.7)	⑤
	TLs	P_{Sh2}	P_{Sh1}	32.3 (2.3)	32.0 (3.3)	30.7 (4.1)	26.9 (1.7)	26.4 (1.9)	27.0 (1.3)	24.0 (0.3)	
	STS	P_{Sh2}	P_{Sh1}	**7.7 (2.6)**	**6.2 (2.4)**	**5.9 (3.5)**	**5.4 (3.1)**	**5.3 (2.6)**	**5.0 (3.0)**	5.2 (2.2)	
	BL	P_{Sh2}		37.9 (10.2)	36.6 (4.8)	25.1 (3.6)	16.9 (9.6)	14.7 (7.8)	11.9 (7.1)	**4.2 (2.3)**	
	cBL	P_{Sh2}	P_{Sh3}	31.0 (1.8)	30.5 (8.8)	18.4 (11.3)	20.0 (11.2)	5.6 (1.7)	12.4 (7.6)	8.9 (6.6)	⑥
	TLs	P_{Sh2}	P_{Sh3}	25.0 (3.3)	20.7 (1.8)	18.4 (2.0)	18.4 (1.1)	16.8 (1.8)	17.2 (1.7)	15.5 (2.5)	
	STS	P_{Sh2}	P_{Sh3}	**6.1 (2.3)**	**5.9 (2.7)**	**5.8 (2.6)**	**4.9 (2.1)**	**5.0 (2.2)**	**5.7 (3.0)**	5.6 (2.6)	

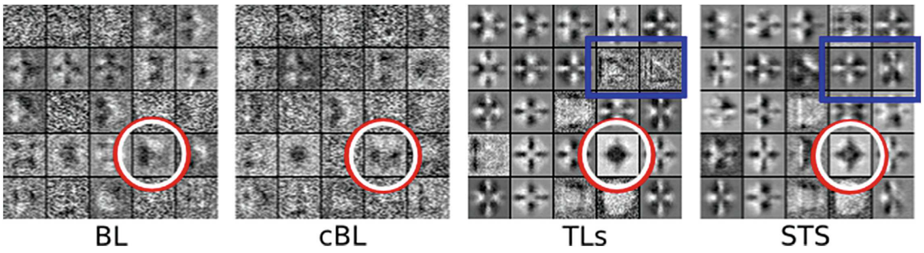

Fig. 3. Feature samples from first layer of non-canonical object recognition task. We observe the transition of same features becoming more distinct, from BL towards STS approach are marked in red circle and from TLs towards STS marked in blue box (Color figure online).

co-adapted neurons between layers in tasks 5 & 6 in Table 1. Here we study the effects of negative feature transference problems with few training samples.

First, we study the effects of transferring *specific features* on character recognition problem. In Table 1 for TLu and TLs approach, tasks 1 & 2 shown negative transference for classifying handwritten digits P_L by reusing source network P_{LC} and P_{UC}. Tasks 3 & 4 show positive transference for classifying either P_{LC} and

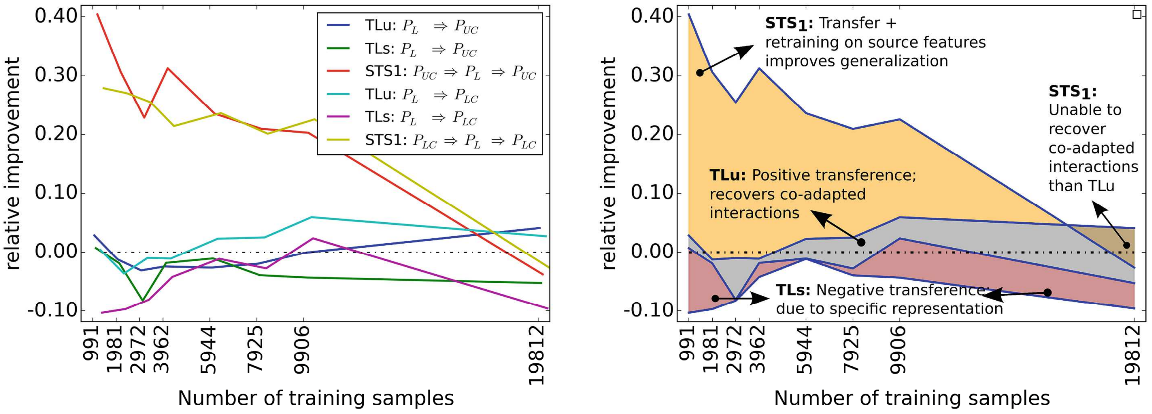

Fig. 4. (**Left:**) Relative improvement over baseline approach for character recognition tasks 3 & 4 as listed in Table 1; (**Right:**) Relative improvement for the tasks on the left, the regions are enclosed to observe relative improvement between two different approaches. We observe negative transference for **TLs** (supervised) approach as it gets stuck at local solution space of specialized features. **TLu** (unsupervised) approach easily recovers the fragile co-adapted neurons as the unsupervised features are not target specific. Also TLu improves over the baseline for complete training data. **STS** approach as intended shake the current local optimal solution, thus overcoming the specialized features of source network unlike TLs approach. The STS shows performance improvement, but unable to recover the fragile co-adapted neurons thus using complete target data, had lower performance than TLu and baseline.

P_{UC} by reusing source network P_L training on complete data. Using STS we observe for tasks 1 to 4 outperforms other approaches for few target samples. In tasks 1 & 2 on STS outperforms BL approaches with a relative improvement of ≈59 % and in tasks 3 & 4 on STS shows ≈30 % improvement for 0.05 % of target data. Figure 4. illustrates the relative improvement performance of BL, TLu, TLs and STS approaches for the tasks 1 & 2.

Second, tasks 5 & 6 analyse the effects of transferring *generic features* on object recognition problem as shown in Table 1. Intuitively canonical objects belong to a subset of non-canonical objects (equilateral triangles belong to a subset of triangles), thus $P_{Sh1} \in P_{Sh2}$ and also number of categories to classify in source and target tasks are equal $Y_S = Y_T$. Thus the only change are due to splitting of co-adapted neurons between the layers while fine-tuning. As we have forced to *lock* the bottom layer, making the optimization harder. cBL, TLu and TLs approaches show negative transference as intended. As solving non-canonical objects is more difficult than solving canonical objects [7]. Using STS approach we observe a relative improvement of ≈81 % for the same task using 0.05 % of total training data, and baseline approach performs better when using complete training data. Figure 3 shows the non-canonical task features for BL, cBL, TL and STS approach using 0.05 % of total training data.

To solve for complete target data using STS, we implement repeating several cycles of STS (see Algorithm 1) till a certain stop criteria is reached. we observe significant improvements using STS over both positive and negative transferred features using TLs as listed in Table 2.

Table 2. Comparison of positive vs. negative transference using *complete target data* and retraining all layers; Performance is measured using percent average test error ($\overline{\varepsilon}$) with 10 repetitions; TLs shows **positive** transference for classifying MNIST P_L reusing Lowercase P_{LC} same as Task 1. And **negative** transference for classifying P_{LC} reusing P_L, same as Task 3. In both cases iteratively repeating **STS** outperforms both BL and TLs approaches.

	Iterative STS	**-ve transference**			**+ve transference**		
		D_B	D_A	$\overline{\varepsilon}$	D_B	D_A	$\overline{\varepsilon}$
BL	D_A	P_{LC}	P_L	1.7 (0.3)	P_L	P_{LC}	4.9 (0.2)
TLs	$D_B \Rightarrow D_A$	P_{LC}	P_L	1.9 (0.2)	P_L	P_{LC}	4.5 (0.2)
STS1	$D_A \Rightarrow D_B \Rightarrow D_A$	P_{LC}	P_L	1.6 (0.1)	P_L	P_{LC}	4.9 (0.1)
STS2	$D_B \Rightarrow D_A \Rightarrow D_B \Rightarrow D_A$	P_{LC}	P_L	1.9 (0.1)	P_L	P_{LC}	**4.4 (0.2)**
STS3	$D_A \Rightarrow D_B \Rightarrow D_A \Rightarrow D_B \Rightarrow D_A$	P_{LC}	P_L	1.6 (0.1)	P_L	P_{LC}	4.9 (0.1)
STS4	$D_B \Rightarrow D_A \Rightarrow D_B \Rightarrow D_A \Rightarrow D_B \Rightarrow D_A$	P_{LC}	P_L	1.9 (0.1)	P_L	P_{LC}	4.5 (0.2)
STS5	$D_A \Rightarrow D_B \Rightarrow D_A \Rightarrow D_B \Rightarrow D_A \Rightarrow D_B \Rightarrow D_A$	P_{LC}	P_L	**1.5 (0.1)**	P_L	P_{LC}	5.0 (0.1)

4 Conclusions and Discussion

Our experiments with the character and object recognition tasks show a deep neural network learns new task more quickly and accurately using transfer learning approach. Unfortunately, they are unreliable for different source and target distribution, because sometime they lead to negative feature transference. The STS algorithm was designed to avoid negative transfer, by recovering fragile co-adapted interactions of neurons between the layers. We make several contributions as listed: 1. The STS approach outperform both baseline and transfer learning approaches. 2. We observe TLu and TLs approach for transferring generic features on distribution that are similar and transferring specific features on tasks that are different to study the impact of splitting of co-adapted neurons. 3. Finally, using the cyclic STS approach reduced the transferability gap between the source and the target tasks. We summarize that the STS outperforms both the baseline and the transfer learning approaches.

Even though the cyclic STS reduced the transferability gap between the source and the target tasks. A pattern is observed when the initial transference was negative. In negative transference case of cyclic STS, we observe odd cycles perform better than even cycles. Iteratively switching the training between the source and the target did not sufficiently perturb the solution out of local minima to a new solution space. We would like to explore this issue by repeating the transference several times and train the network to jump to a new solution space to obtain good generalization. Also exploring the possibility of using multiple source problem to obtain diverse and generic features.

References

1. Thrun, S.: Learning to learn: introduction. In: Learning To Learn (1996)
2. Caruana, R.: Multitask learning. Mach. Learn. **28**(1), 41–75 (1997)

3. Daumé III, H., Marcu, D.: Domain adaptation for statistical classifiers. J. Artif. Intell. Res. (JAIR) **26**, 101–126 (2006)
4. Raina, R., Battle, A., Lee, H., Packer, B., Ng, A.Y.: Self-taught learning: transfer learning from unlabeled data. In: ACM Conference on Proceedings (ICML) (2007)
5. Ciresan, D., Meier, U., Schmidhuber, J.: Multi-column deep neural networks for image classification. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE (2012)
6. Kandaswamy, C., Silva, L.M., Alexandre, L.A., de Sá, J.M.: Improving deep neural network performance by reusing features trained with transductive transference. In: Wermter, S., Weber, C., Duch, W., Honkela, T., Koprinkova-Hristova, P., Magg, S., Palm, G., Villa, A.E.P. (eds.) ICANN 2014. LNCS, vol. 8681, pp. 265–272. Springer, Heidelberg (2014)
7. Kandaswamy, C., Silva, L.M., Alexandre, L.A., Sousa, R. Santos, J.M., de Sá, J.M.: Improving transfer learning accuracy by reusing stacked denoising autoencoders. In: IEEE Conference on SMC. IEEE (2014)
8. Yosinski, J., et al.: How transferable are features in deep neural networks? In: Advances in Neural Information Processing Systems (2014)