Self-ensembling for domain adaptation

French, G.    Mackiewicz, M.
g.french@uea.ac.uk    m.mackiewicz@uea.ac.uk

Fisher, M.
mark.fisher@uea.ac.uk

June 19, 2017

Abstract

This paper explores the use of self-ensembling with random image augmentation [11] – a technique that has achieved impressive results in the area of semi-supervised learning – for visual domain adaptation problems. We modify the approach of Laine et al. to improve stability and ease of use. Our approach demonstrates state of the art results when performing adaptation between the following pairs of datasets: MNIST and USPS, CIFAR-10 and STL, SVHN and MNIST, Syn-Digits to SVHN and Syn-Signs to GTSRB. We also explore the use of richer data augmentation to solve the challenging MNIST to SVHN adaptation path.

1 Introduction

The state of the art performance of deep learning in computer vision tasks such as image classification [10, 8], object localisation [16] and image segmentation [2] comes at the cost of requiring large data sets with corresponding ground truth labels for training. Such data sets are often expensive to produce, owing to the cost the human labour required to produce the ground truth labels.

Semi-supervised learning is an active area of research that aims to reduce the quantity of ground truth labels required for training. It is aimed at common practical scenarios in which only a small subset of a large data set has corresponding ground truth labels.

Unsupervised domain adaptation is a closely related problem in which one attempts to transfer knowledge gained from a labeled source data set to a distinct unlabeled target data set, within the constraint that the objective must be the same for both datasets (e.g., digit classification). Domain adaptation offers the potential to train a model using labeled synthetic data – that is often abundantly available – and unlabeled real data.

Recent work [11, 18] has demonstrated the effectiveness of self-ensembling with random image augmentations to achieve impressive performance in semi-supervised learning benchmarks.
We present a modified variant of the approach of Laine et al. and demonstrate that it is well suited to domain adaptation. It is able to achieve state of the art performance in several benchmarks, namely between the MNIST–USPS, CIFAR-10–STL and SVHN–MNIST data set pairs and for the synthetic to real dataset pairings, namely Syn-Digits–SVHN and Syn-Signs–GTSRB. We also discuss how some of these benchmarks required the use of specific rich data augmentation to obtain good results.

2 Related work

2.1 Semi-supervised learning

2.1.1 Autoencoders

Ladder networks [15] are autoencoders that use lateral connections between the encoding and decoding stages to incorporate layer-by-layer reconstruction objectives and noise into an end-to-end model. They achieve impressive results on the MNIST and CIFAR-10 benchmarks. Maaløe et al. [12] develop a variational autoencoder based model that achieved state of the art results in the MNIST semi-supervised classification benchmarks.

2.1.2 Generative Adversarial Networks (GANs)

A GAN [6] consists of a generator model that learns to generate samples that match the distribution of a source data set while a discriminator model learns to distinguish real samples for generated ones. The generator is trained to generate samples that trick the discriminator into predicting that they are real.

Salimans et al. developed a GAN based semi-supervised classification model [19] in which the discriminator operates as an $N + 1$-way classifier, with the additional class representing generated samples. The discriminator is trained to maximise the predicted probability of the ground truth class for labeled samples, the fake class for generated samples and the sum of all ground truth class probabilities for unlabeled samples.

2.1.3 Adversarial training

Szegedy et al. [22] discovered that deep neural networks are vulnerable to misclassification of images that are subtly and (almost) imperceptibly modified such that they retain their original visual appearance while causing the network to mis-predict. The image is modified using gradient in order to increase the score of a specified class label. Goodfellow et al. [7] investigated and explained the properties of adversarial examples and used them to provide a form of regularization. Miyato et al. [13] introduced virtual adversarial training, in which the training procedure generates adversarial examples that attempt to fool the classifier on the fly and use them for regularization, achieving good semi-supervised classification results in the process.
2.1.4 Self-ensembling

Sajjadi et al.\cite{18} apply rich data augmentation during training and record the history of the network’s predictions for each training sample at each epoch. At epoch $k$ they minimise the sum of squared differences between the prediction for sample $i$ at epoch $k$ and all of the predictions recorded in the history for sample $i$ from epoch 0 to $k-1$. Sajjadi et al. refer to this as transformation/stability loss.

Laine et al.\cite{11} present two models; their II-model and their temporal model. The II-model passes each unlabeled sample through a classifier twice, each time with different dropout and data augmentation parameters. They minimise the mean of the squared difference in class probability predictions in addition to the standard categorical cross-entropy loss. Their temporal model is more similar to that of Sajjadi et al.\cite{18}, except that it maintains a per-sample moving average of the networks’ historical predictions. The unsupervised loss is the mean squared difference between the current prediction and the moving average, rather than the difference between two predictions in the current epoch only, as with the II-model. Their approach achieved state of the art results in the SVHN and CIFAR-10 semi-supervised classification benchmarks.

2.2 Domain adaptation

Ganin et al.\cite{4} use a bifurcated network that starts with feature extraction layers before splitting into two branches; a label classification branch and a domain classification branch that predicts the dataset from which a sample was drawn. A gradient reversal layer placed between the feature sub-network and the domain classification branch inverts the gradients propagating from the domain classifier to the feature sub-network, encouraging it to extract domain independent features. During training, samples from the source domain train both the label classification branch and the domain classification branch. Target domain samples train only the domain classification branch. An alternative and simpler implementation described in their appendix minimises the label cross-entropy loss in the feature and label classification layers, minimises the domain cross-entropy in the domain classification layers but maximises it in the feature layers.

The model of Ghifary et al.\cite{5} is an autoencoder that predicts class labels by connecting fully-connected classification layers to the latent representation generated by the encoder part of the network. The network is trained to classify samples from the source domain – for which ground truths are available – and reconstruct samples from both the source and the target domain. It is interesting contrast it with the model of Ganin et al.\cite{4}; both are bifurcated networks. Ganin et al. encourage domain independent features through the use of gradient reversal whereas Ghifary et al. use autoencoder reconstruction.

Sankaranarayanan et al.\cite{20} present a hybrid model that consists of an encoder and a conditional GAN. The encoder transforms samples from the source domain into a latent space. Given the latent representation and random noise,
the GAN’s generator learns to create a sample that matches the distribution of
the source data set. The GAN’s discriminator attempts to classify a sample as
real (from the source distribution) or fake. Finally, given a target sample, the
encoder is trained to produce a latent representation that induces the generator
to create an equivalent sample from the source distribution, thereby encouraging
the encoder to extract domain independent features.

The model of Tzeng et al.\cite{tzeng2017adversarial} operates along similar lines but is simpler.
A source classification network is trained to classify samples from the source
domain. A domain classification network is then trained to determine which do-
main a sample originated from, but with source samples passed through the fea-
ture extraction layers from the first network and target samples passed through
a new set of feature extraction layers for the target domain. These new feature
extraction layers are trained in order to trick the final domain classification
layers into predicting that the target samples come from the source data set,
while the final domain classification layers are trained to discriminate between
the two. Once complete, the label classifier trained in the first step is attached
to the target domain feature extraction layers.

Saito et al.\cite{Saito2017} use tri-training\cite{li2016tri}; feature extraction layers are used to
drive three classifier sub-networks. The first two are trained on samples from
the source domain, while a weight similarity penalty encourages them to learn
different weights. Pseudo-labels generated for target domain samples by these
source domain classifiers are used to train the final classifier to operate on the
target domain.

3 Method

Our approach is essentially that of the Π-model of Laine et al.\cite{laina2016object}, with a few
modifications designed to improved training stability and ease of use.

The structure of the Π-model is shown in Figure 1a. During training, the
network is evaluated twice for each input sample \(x_i\), resulting in predicted class
probability vectors \(\hat{z}_i\) and \(\tilde{z}_i\). Different data augmentation parameters, noise
and dropout masks are used for each evaluation.

The supervised loss is the standard cross-entropy loss. For unlabeled samples
for which no ground truth is available, the supervised loss is masked to 0.
The unsupervised loss is computed for both labeled and unlabeled samples.
It penalises the difference in class predictions for the same input sample that
result from the different augmentation, noise and drop-out parameters. It is
computed using the mean squared difference between the class predictions \(z_i\)
and \(\tilde{z}_i\).

Laine et al. found that it was necessary to scale up the unsupervised loss
in a time dependent manner during training. Using unsupervised loss at full
strength from the start results in the network getting stuck in a degenerate
state that gives very poor classification performance. To address this, Laine et
al.weighted the unsupervised loss using a time dependent function that follows
a Gaussian curve from 0 to 1 during the first 80 epochs.
Figure 1: The network structures of the original Π-model and our model. Dashed lines in the Π-model indicate that ground truth labels – and therefore cross-entropy classification loss – are only available for labeled samples, that make up a small fraction of the total dataset.

Our implementation\footnote{Available at \url{http://github.com/Britefury/domain_adapt_self_ensemble}} was developed using Theano\footnote{The original implementation of [11] is available at \url{https://github.com/smlaine2/tempens}} and Lasagne\footnote{3}.  

3.1 Improving training stability

Our early experience with the Π-model lead us to conclude that masking the supervised loss to 0 for unlabeled samples can result in training instabilities that take the form of NaN values being generated, causing training to collapse. The original implementation\footnote{2} by Laine et al uses an implementation of the ADAM\footnote{9} optimisation algorithm that detects NaN values and sets them to 0 in order to avoid this problem.

We address this problem in our network by using the structure seen in Figure 1b. We split the network into separate supervised and unsupervised paths. The supervised path uses the standard cross-entropy loss and draws labeled samples \( x_{Li} \) and ground truths \( y_{Li} \) from the source (labeled) data set. The unsupervised path draws samples \( x_i \) from the concatenation of the source and target data sets. Each training mini-batch therefore consists of an equal number of samples for the supervised and unsupervised paths. In our experience this stabilises training at the cost of additional computational load.

3.2 Confidence thresholding

The Gaussian ramp-up function that scales the unsupervised loss in a time-dependent manner is necessary to prevent the network from getting stuck in a degenerate state during training. The appropriate length and speed of the ramp-up will be related to the rate at which the network learns, which will depend on the nature of the problem. As a consequence, this hyper-parameter is likely to be tricky to tune. We propose replacing it with confidence thresholding.
For each unlabeled sample $x_i$ we compute the confidence of the corresponding class prediction using $c_i = \max(z_i)$ and $\tilde{c}_i = \max(\tilde{z}_i)$. If neither $c_i$ or $\tilde{c}_i$ are above the confidence threshold (we find that a value of 0.99 works well) then the unsupervised loss for the sample $x_i$ is masked to 0. This causes the average weight of the unsupervised loss to scale according to the progress of the supervised component of the network.

3.3 Richer data augmentation

The augmentation used in [11] consisted of horizontal flips (for some experiments), integer valued translations and Gaussian noise. In addition, we apply random affine transformations whose matrices are generated using the matrix shown in (1), where $\mathcal{N}(0,0.1)$ denotes a real value drawn from a normal distribution with mean 0 and standard deviation 0.1. Our translations are real valued and drawn from the range $[-2,2]$.

$$\begin{bmatrix}
1 + \mathcal{N}(0,0.1) & \mathcal{N}(0,0.1) \\
\mathcal{N}(0,0.1) & 1 + \mathcal{N}(0,0.1)
\end{bmatrix}$$

(1)

3.4 Batch balancing

When learning to adapt from a large source data set to a smaller target data set we found that a slight improvement in target accuracy could be obtained by ensuring that mini-batches of samples used to compute the unsupervised loss ($x_i$ in Figure 1) draw half their samples from each of the source and target data sets. This is in contrast to the normal case where unlabeled samples will be drawn from the source and target data sets in proportion to their size.

4 Results

Our results can be seen in Table 1. The ‘Convsrc’ results show the baseline performance on the target domain test set of a standard neural network with no domain adaptation trained on the source domain training set. Similarly, ‘Convtgt’ results show the performance on the target domain test set of a standard network trained on the target domain training set, giving an estimate of the best achievable result. The baseline results were obtained using the same data augmentation scheme as was used for the domain adaptation experiments. The ‘Convsrc (aug)’, ‘SE (aug)’ and ‘Convtgt (aug)’ results demonstrate the effect of additional data augmentation used for the MNIST ↔ SVHN adaptation paths (discussed below).

4.1 Experiments

The data sets used in this work are described in Appendix A Table 2, our training procedure is described in Appendix B and our network architectures are described in Appendix C.
MNIST ↔ USPS. MNIST and USPS are both hand-written digit datasets. The USPS images were up-scaled 16 × 16 to 28 × 28 resolution. Batch balancing was used for the MNIST → USPS path. In both directions our approach demonstrates a significant improvement over prior art; particularly in the USPS → MNIST direction. This can be attributed to the similarity of the data sets to one another and the fact that the original algorithm of Laine et al. [11] exhibits strong performance in semi-supervised settings. Our MNIST → USPS performance is strong in comparison to prior work, but is less marked when contrasted to the baseline performance of a standard network trained on MNIST only. It is worth noting that our baseline performance in both directions is also stronger than that of prior art, most likely due to our richer data augmentation.

SVHN → MNIST. Google’s SVHN (Street View House Numbers) is a colour digits dataset of house number plates. The SVHN images were converted to grey-scale – as in [5] – and the MNIST images were padded to 32 × 32 resolution. Our approach is slightly better than [17] with default augmentation parameters and exhibits further improvements with the richer augmentation parameters designed for the MNIST → SVHN adaptation path discussed below.

MNIST → SVHN. This adaptation path is somewhat more challenging as MNIST digits are uniform in terms of size, aspect ratio and intensity range, where SVHN is far more varied. Successfully adapting from MNIST to SVHN required additional tuning. The default augmentation parameters presented in section 3.3 result in an accuracy score of around 20%; no better than a null classifier that predicts the most populous SVHN class (hence the null entry in Table 1). In order to achieve the performance figures shown, we standardised (zero mean, unit variance) the pixel intensities each image separately and added further data augmentation. Our additional augmentation consisted of: random scale in the [0.4, 2.5] range in the X-axis and [0.667, 1.5] range in the Y-axis; and

|               | USPS | MNIST | SVHN | MNIST | CIFAR | STL | Syn | Syn |
|---------------|------|-------|------|-------|-------|-----|-----|-----|
|               | 99.71 | 98.31 | 99.79 | 97.3  | 96.6  |     |     |     |
| Conv_{src}    | 99.71 | 98.31 | 99.79 | 97.3  | 96.6  |     |     |     |
| Conv_{src} (aug)b | 92.93 | 96.3  | 95.73 | 97.24 | 87.82 | 97.3 | 99.66 |
| RevGrad [4]   | 86.20 | 74.01 | 91.11 | 73.91 | 66.12 | 56.91| 91.09| 88.65|
| DCRN [5]      | 73.67 | 90.8  | 92.5  | 84.70 | 36.4  | 91.11| 73.91| 66.12 |
| G2A [20]      | 90.1  | 89.4  | 76.00 | –     | –     | –   | –   | –   |
| ADDA [23]     | –     | –     | –     | –     | –     | –   | –   | –   |
| ATT [17]      | –     | –     | –     | –     | –     | –   | –   | –   |
| SE (ours)     | 99.21 | 98.16 | 87.86 | null  | 76.25 | 64.21| 94.24| 98.35|
| SE (aug)b     | –     | –     | –     | –     | –     | –   | –   | –   |
| Conv_{tgt}    | 99.71 | 98.31 | 99.76 | 95.73 | 70.24 | 87.82| 97.3 | 99.66|
| Conv_{tgt} (aug)b | –     | –     | –     | –     | –     | –   | –   | –   |

Table 1: Classification accuracy

\(^a\) Combined results from [4] and [5], taking best result if more than one available.

\(^b\) Used additional augmentation.
random intensity augmentation that consists of scaling the value of all pixels by a value drawn from a uniform distribution with the range \([-1.5, 1.5]\) and offsetting them with values drawn from a uniform distribution with the range \([-0.5, 0.5]\). These augmentation parameters were chosen by manually inspecting their effect in order to cause the sizes, aspect ratios and intensity ranges to overlap. The unsupervised loss was scaled by a factor of 0.1 when we found that during training, the test set performance on the target set would degenerate to that of a null classifier. With the modifications described we achieved the strong performance seen in the ‘SE (aug)’ row in Table I significantly out-performing prior art. It is however worth noting that this rich augmentation scheme raises the baseline performance of a standard network (‘Conv\(_{src}\) (aug)’ row) to just above that of much of the prior art.

**CIFAR-10 ↔ STL.** CIFAR-10 and STL are both 10-class image datasets. The STL images were down-scaled to 32\(\times\)32 resolution to match that of CIFAR-10. The ’frog’ class in CIFAR-10 and the ’monkey’ class in STL were removed as they have no equivalent in the other data set, resulting in a 9-class problem with 10% less samples in each data set. We obtained strong performance in the STL \(\rightarrow\) CIFAR-10 path, likely due to the similar nature of the data sets and the strong performance exhibited in semi-supervised settings as seen in the USPS \(\rightarrow\) MNIST path earlier. The CIFAR-10 \(\rightarrow\) STL results are more interesting; the baseline performance of a standard network trained on the CIFAR-10 source domain out performs that of a network trained on the STL target domain, most likely due to small size of the STL training set. Our self-ensembling results outpace both the baseline performance and the ‘theoretical maximum’ of ‘Conv\(_{tgt}\)’, lending further evidence the view of Sajjadi \textit{et al.}\[18]\) and Laine \textit{et al.}\[11]\) that self-ensembling acts as an effective regulariser.

**Syn-Digits \(\rightarrow\) SVHN.** The Syn-Digits dataset is a synthetic digit images data set designed by Ganin \textit{et al.}\[4]\) to be used as a source data set in domain adaptation experiments for the purpose of targeting SVHN. Other approaches have achieved good scores on this benchmark, beating the baseline by a significant margin. Our result improves on them, reducing the error rate from 6.9% to 5.76%.

**Syn-Signs \(\rightarrow\) GTSRB.** Syn-Signs is another synthetic images data set designed by Ganin \textit{et al.}\[4]\), but designed to target the 43-class GTSRB (German Traffic Signs Recognition Benchmark) \[21]\) data set. GTSRB is composed of images that vary in size and come with corresponding annotations that provide region of interest (bounding box around the sign) and ground truth classification. We extract the region of interest from each image and scale to a resolution of 40\(\times\)40 to match those of Syn-Signs. Our approach halves the best error rate of competing approaches. As seen in the MNIST \(\leftrightarrow\) USPS and CIFAR-10 \(\rightarrow\) STL results our baseline out-performs existing domain adaptation algorithms, most likely due to our use of data augmentation.
5 Conclusions and future work

We have presented a self-ensembling based algorithm for domain adaptation. Its success is not altogether surprising, given the excellent semi-supervised classification results seen by Laine et al. [11]. Our results support the view that it acts as an effective regulariser. The use of rich data augmentation is key to this approach. This is indicated by our results in the tricky MNIST to SVHN adaptation path; using default augmentation parameters results in a classifier that is no better than the null classifier for SVHN. By using richer augmentation parameters we are able to close this gap. We believe that the algorithm operates by the following two mechanisms. Firstly, by label propagation – as suggested in [13] – and secondly by eliminating irrelevant axes of variation. Penalising the network for generating different predictions for differently augmented versions of the same sample causes it to learn to suppress the effect of these variations, in spite of the fact that the effects of geometric transformations are non-trivial at the pixel level.

An obvious extension to our work would be to implement the temporal model of Laine et al. [11], which would likely lead to an improvement in performance. Furthermore, the gradient reversal model of Ganin et al. [4] could be integrated in order to assess its impact on performance.

References

[1] Rami Al-Rfou and et al. Theano: A Python framework for fast computation of mathematical expressions. arXiv e-prints, May 2016.

[2] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. CoRR, abs/1511.00561, 2015.

[3] S. Dieleman, J. Schlüter, and et al. Lasagne, August 2015.

[4] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In Proceedings of the 32nd International Conference on Machine Learning (ICML-15), pages 1180–1189, 2015.

[5] Muhammad Ghifary, W Bastiaan Kleijn, Mengjie Zhang, David Balduzzi, and Wen Li. Deep reconstruction-classification networks for unsupervised domain adaptation. In European Conference on Computer Vision, pages 597–613. Springer, 2016.

[6] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Advances in neural information processing systems, pages 2672–2680, 2014.

[7] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572, 2014.
[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[9] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. 2012.

[11] Samuli Laine and Timo Aila. Temporal ensembling for semi-supervised learning. In *International Conference on Learning Representations*, 2017.

[12] Lars Maaløe, Casper Kaæ Sønderby, Søren Kaæ Sønderby, and Ole Winther. Auxiliary deep generative models. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1445–1453, 2016.

[13] Takeru Miyato, Schi-ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *arXiv preprint arXiv:1704.03976*, 2017.

[14] Lutz Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 1998.

[15] Antti Rasmus, Mathias Berglund, Mikko Honkala, Harri Valpola, and Tapani Raiko. Semi-supervised learning with ladder networks. In *Advances in Neural Information Processing Systems*, pages 3546–3554, 2015.

[16] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems 28*, pages 91–99. 2015.

[17] Kuniaki Saito, Yoshitaka Ushiku, and Tatsuya Harada. Asymmetric tri-training for unsupervised domain adaptation. *arXiv preprint arXiv:1702.08340*, 2017.

[18] Mehdi Sajjadi, Mehran Javanmardi, and Tolga Tasdizen. Regularization with stochastic transformations and perturbations for deep semi-supervised learning. In *Advances in Neural Information Processing Systems*, pages 1163–1171, 2016.

[19] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2226–2234, 2016.

[20] Swami Sankaranarayanan, Yogesh Balaji, Carlos D Castillo, and Rama Chellappa. Generate to adapt: Aligning domains using generative adversarial networks. *arXiv preprint arXiv:1704.01705*, 2017.
[21] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In IEEE International Joint Conference on Neural Networks, pages 1453–1460, 2011.

[22] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In International Conference on Learning Representations, 2014.

[23] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. arXiv preprint arXiv:1702.05464, 2017.

[24] Zhi-Hua Zhou and Ming Li. Tri-training: Exploiting unlabeled data using three classifiers. IEEE Transactions on knowledge and Data Engineering, 17(11):1529–1541, 2005.

A  Data sets

The data sets used in this paper are described in Table 2.

|       | # train | # test | # classes | Resolution | Channels |
|-------|---------|--------|-----------|------------|----------|
| USPS  | 7,291   | 2,007  | 10        | 16 × 16    | Mono     |
| MNIST | 60,000  | 10,000 | 10        | 28 × 28    | Mono     |
| SVHN  | 73,257  | 26,032 | 10        | 32 × 32    | RGB      |
| CIFAR-10 | 50,000 | 10,000 | 10        | 32 × 32    | RGB      |
| STL   | 5,000   | 8,000  | 10        | 96 × 96    | RGB      |
| Syn-Digits | 479,400 | 9,553 | 10        | 32 × 32    | RGB      |
| Syn-Signs | 100,000 | –    | 43        | 40 × 40    | RGB      |
| GTSRB | 32,209  | 12,630 | 43        | varies     | RGB      |

a Available from [http://statweb.stanford.edu/~tibs/ElemStatLearn/datasets/zip.train.gz](http://statweb.stanford.edu/~tibs/ElemStatLearn/datasets/zip.train.gz) and [http://statweb.stanford.edu/~tibs/ElemStatLearn/datasets/zip.test.gz](http://statweb.stanford.edu/~tibs/ElemStatLearn/datasets/zip.test.gz)

b Available from [http://ai.stanford.edu/~acoates/stl10/](http://ai.stanford.edu/~acoates/stl10/)

c Available from Ganin’s website at [http://yaroslav.ganin.net/](http://yaroslav.ganin.net/)

Table 2: Data sets

B  Training

B.1  Training procedure

Our networks were trained for 200 epochs. We used the Adam gradient descent algorithm with a learning rate of 0.001. We trained using mini-batches composed of 64 labeled samples and 64 unlabeled samples. A complete pass over the entire unlabeled data set – using samples from both the source and target data sets – was considered to be one epoch.
The best target set error rate was chosen using a form of early stopping [14]; after each epoch the performance of the network was measured for the source and target test sets. The final score was the target test set performance at the epoch at which the best source test set performance was obtained; the source test set was used in the place of a validation score.

The same procedure was used to train our baseline networks. It is worth noting that the theoretical maximum performance baselines in the ‘Conv\text{tgt}’ rows in Table 1 are the best test set score achieved – at any epoch – during training. Choosing the best test set performance in this fashion is not considered to be particularly good machine learning practice, however we do not consider this to be problematic in this case as we are using these numbers as a theoretical upper bound for the purpose of comparison to our approach, rather than as results themselves.

### B.2 Batch balancing

We used batch balancing when the target data set was significantly smaller than the source data set. Given that batch balancing draws half a mini-batch of samples (32) from both the source and target data set, this raises the issue of whether an epoch is considered to be a complete iteration over all of the samples from the larger sources set or the smaller target set. We used both options in different circumstances; when targeting small data sets such as USPS we considered a complete iteration over the larger source set to be an epochs. When adapting from the large Syn-Digits (~480k) data set to the moderately sized SVHN (~73k), a complete iteration over Syn-Digits would result in very long training times, so we use the size of the smaller SVHN as a complete epoch.

### B.3 Experimental conditions

Experimental conditions are shown in Table 3.

| Experiment          | Batch balancing | Sample standardisation | Augmentation                      |
|---------------------|-----------------|------------------------|-----------------------------------|
| USPS → MNIST        |                 |                        | Affine, translation               |
| MNIST → USPS        | ✓               |                        | Affine, translation               |
| SVHN → MNIST        |                 |                        | Affine, translation               |
| SVHN → MNIST (aug)  |                 | ✓                      | Affine, translation, scale, intensity |
| MNIST → SVHN (aug)  |                 | ✓                      | Affine, translation, scale, intensity |
| CIFAR-10 → STL      | ✓               |                        | Affine, translation, horizontal flip |
| STL → CIFAR-10      |                 |                        | Affine, translation               |
| Syn-Digits → SVHN   | ✓               |                        | Affine, translation               |
| Syn-Signs → GTSRB   |                 | ✓                      | Affine, translation               |

Table 3: Experimental conditions
C  Network architectures

Our network architectures are shown in Tables 4 - 8.

| Description                                      | Shape          |
|--------------------------------------------------|----------------|
| 28 × 28 Mono image                               | 28 × 28 × 1    |
| Conv 5 × 5 × 32, batch norm                      | 24 × 24 × 32   |
| Max-pool, 2x2                                    | 12 × 12 × 32   |
| Conv 3 × 3 × 64, batch norm                      | 10 × 10 × 64   |
| Conv 3 × 3 × 64, batch norm                      | 8 × 8 × 64     |
| Max-pool, 2x2                                    | 4 × 4 × 64     |
| Dropout, 50%                                     | 4 × 4 × 64     |
| Fully connected, 256 units                        | 256            |
| Fully connected, 10 units, softmax               | 10             |

Table 4: MNIST ↔ USPS architecture

| Description                                      | Shape          |
|--------------------------------------------------|----------------|
| 32 × 32 Mono image                               | 32 × 32 × 1    |
| Conv 3 × 3 × 48, pad 1, leaky ReLU 0.1, batch norm | 32 × 32 × 48   |
| Conv 3 × 3 × 48, pad 1, leaky ReLU 0.1, batch norm | 32 × 32 × 48   |
| Conv 3 × 3 × 48, pad 1, leaky ReLU 0.1, batch norm | 32 × 32 × 48   |
| Max-pool, 2x2                                    | 16 × 16 × 48   |
| Dropout, 50%                                     | 16 × 16 × 48   |
| Conv 3 × 3 × 96, pad 1, leaky ReLU 0.1, batch norm | 16 × 16 × 96   |
| Conv 3 × 3 × 96, pad 1, leaky ReLU 0.1, batch norm | 16 × 16 × 96   |
| Conv 3 × 3 × 96, pad 1, leaky ReLU 0.1, batch norm | 16 × 16 × 96   |
| Max-pool, 2x2                                    | 8 × 8 × 96     |
| Dropout, 50%                                     | 8 × 8 × 96     |
| Conv 3 × 3 × 192, pad 0, leaky ReLU 0.1, batch norm | 6 × 6 × 192   |
| Conv 1 × 1 × 192, leaky ReLU 0.1, batch norm     | 6 × 6 × 192    |
| Conv 1 × 1 × 192, leaky ReLU 0.1, batch norm     | 6 × 6 × 192    |
| Global pooling layer                             | 1 × 1 × 192    |
| Fully connected, 10 units, softmax               | 10             |

Table 5: SVHN → MNIST architecture
Table 6: MNIST → SVHN architecture

| Description                                           | Shape            |
|-------------------------------------------------------|------------------|
| 32 × 32 Mono image                                    | 32 × 32 × 1      |
| Conv 3 × 3 × 32, pad 1, batch norm                    | 32 × 32 × 32     |
| Conv 3 × 3 × 32, pad 1, batch norm                    | 32 × 32 × 32     |
| Max-pool, 2x2                                         | 16 × 16 × 32     |
| Conv 3 × 3 × 64, pad 1, batch norm                    | 16 × 16 × 64     |
| Conv 3 × 3 × 64, pad 1, batch norm                    | 16 × 16 × 64     |
| Conv 3 × 3 × 64, pad 1, batch norm                    | 16 × 16 × 64     |
| Max-pool, 2x2                                         | 8 × 8 × 64       |
| Conv 3 × 3 × 128, pad 1, batch norm                   | 8 × 8 × 128      |
| Conv 3 × 3 × 128, pad 1, batch norm                   | 8 × 8 × 128      |
| Conv 3 × 3 × 128, pad 1, batch norm                   | 8 × 8 × 128      |
| Max-pool, 2x2                                         | 4 × 4 × 128      |
| Global pooling layer                                  | 1 × 1 × 128      |
| Dropout, 50%                                          | 1 × 1 × 128      |
| Fully connected, 128 units                            | 128              |
| Fully connected, 10 units, softmax                    | 10               |

Table 7: CIFAR-10 ↔ STL and Syn-Digits → SVHN architecture

| Description                                           | Shape            |
|-------------------------------------------------------|------------------|
| 32 × 32 RGB image                                     | 32 × 32 × 3      |
| Conv 3 × 3 × 128, pad 1, leaky ReLU 0.1, batch norm   | 32 × 32 × 128    |
| Conv 3 × 3 × 128, pad 1, leaky ReLU 0.1, batch norm   | 32 × 32 × 128    |
| Conv 3 × 3 × 128, pad 1, leaky ReLU 0.1, batch norm   | 32 × 32 × 128    |
| Max-pool, 2x2                                         | 16 × 16 × 128    |
| Dropout, 50%                                          | 16 × 16 × 128    |
| Conv 3 × 3 × 256, pad 1, leaky ReLU 0.1, batch norm   | 16 × 16 × 256    |
| Conv 3 × 3 × 256, pad 1, leaky ReLU 0.1, batch norm   | 16 × 16 × 256    |
| Conv 3 × 3 × 256, pad 1, leaky ReLU 0.1, batch norm   | 16 × 16 × 256    |
| Max-pool, 2x2                                         | 8 × 8 × 256      |
| Dropout, 50%                                          | 8 × 8 × 256      |
| Conv 3 × 3 × 512, pad 0, leaky ReLU 0.1, batch norm   | 6 × 6 × 512      |
| Conv 1 × 1 × 256, leaky ReLU 0.1, batch norm          | 6 × 6 × 256      |
| Conv 1 × 1 × 128, leaky ReLU 0.1, batch norm          | 6 × 6 × 128      |
| Global pooling layer                                  | 1 × 1 × 128      |
| Fully connected, 10 units, softmax                    | 10               |
| Description                                      | Shape  |
|-------------------------------------------------|--------|
| 40 × 40 RGB image                               | 40 × 40 × 3 |
| Conv 3 × 3 × 96, pad 1, batch norm              | 40 × 40 × 96 |
| Conv 3 × 3 × 96, pad 1, batch norm              | 40 × 40 × 96 |
| Conv 3 × 3 × 96, pad 1, batch norm              | 40 × 40 × 96 |
| Max-pool, 2x2                                   | 20 × 20 × 96 |
| Dropout, 50%                                    | 20 × 20 × 96 |
| Conv 3 × 3 × 192, pad 1, batch norm             | 20 × 20 × 192 |
| Conv 3 × 3 × 192, pad 1, batch norm             | 20 × 20 × 192 |
| Conv 3 × 3 × 192, pad 1, batch norm             | 20 × 20 × 192 |
| Max-pool, 2x2                                   | 10 × 10 × 192 |
| Dropout, 50%                                    | 10 × 10 × 192 |
| Conv 3 × 3 × 384, pad 1, batch norm             | 10 × 10 × 384 |
| Conv 3 × 3 × 384, pad 1, batch norm             | 10 × 10 × 384 |
| Conv 3 × 3 × 384, pad 1, batch norm             | 10 × 10 × 384 |
| Max-pool, 2x2                                   | 5 × 5 × 384 |
| Dropout, 50%                                    | 5 × 5 × 384 |
| Global pooling layer                            | 1 × 1 × 384 |
| Fully connected, 43 units, softmax              | 43      |

Table 8: Syn-signs → GTSRB architecture