Comparison of Domain Adaptation performance's using different types of Classifiers

SVM and k-NN applied on SVHN, MNIST-M and SYNTH datasets

Filippo Nevi & Mario Libro

VR458510 & VR450927

Machine learning & artificial intelligence project

Computer Engineering for Robotics and Smart Industry
University of Verona
2021/2022

Contents

1 Introduction **2**
- 1.1 Objectives . . . 2
- 1.2 Motivation and Rationale . . . 2

2 State of the art **3**

3 Methodology **3**
- 3.1 Tools and libraries . . . 3
- 3.2 Datasets . . . 3
- 3.3 Feature extraction . . . 5
  - 3.3.1 CNN - Resnet34 . . . 5
  - 3.3.2 HOG - Histogram of Oriented Gradients . . . 5
- 3.4 Feature reduction . . . 6
  - 3.4.1 PCA - Principal Component Analysis . . . 6
  - 3.4.2 LDA - Linear Discriminant Analysis . . . 7
- 3.5 Classification . . . 7
  - 3.5.1 SVM - Support Vector Machine . . . 7
  - 3.5.2 k-Nearest Neighbours . . . 7

4 Experiment & Results **7**
- 4.1 Feature extraction with CNN . . . 8
  - 4.1.1 Performances (source = target) . . . 9
  - 4.1.2 Performances Domain Adaptation (source $\neq$ target) . . . 10
- 4.2 Feature extraction with HOG . . . 11
  - 4.2.1 Performances (source = target) . . . 11
  - 4.2.2 Performances Domain Adaptation (source $\neq$ target) . . . 12

5 Conclusions **12**

1 Introduction

1.1 Objectives

This project aims to provide a performance comparison between different combination of feature extractors, feature reductors and classifiers applying Domain Adpatation on different types of datasets (SVHN [9], MNIST-M [6] and SYNTH [5]) that contains images of digits extracted from real world and synthetic scenarios, allowing us to to explore the potential and the limits of Domain Adaptation applied using machine learning techniques.

The performance's analysis was performed between 72 different types of configurations, combining:

- 2x Features extractor (Resnet-34 and Histogram of Oriented Gradients);
- 2x Features reductor (Linear Discriminant Analysis and Principal Component Analysis);
- 2x Classifiers (Support Vector Machine and k-Nearest Neighbors);
- 3x Datasets (SVHN, MNIST-M and SYNTH).

1.2 Motivation and Rationale

Domain adaptation is a field of computer vision, where the goal is to train a model on a source dataset and get a good accuracy on the target dataset which is significantly different from the source.

Nowadays we could retrieve a lot of different datasets for different purposes, but if we need to get a model that perform very well for a very specific task, we need to collect and label a lot of data, which is a laborious and time consuming task. There are also some cases where is difficult (or impossible) to gather datasets and in this case with the help of different computer vision algorithms, we can generate large synthetic dataset and then train the model on the synthetic dataset (source) and test it on real-life data (target).

Obviously Domain Adaptation could be applied in many different domains, but we decided to focus our attention on the classification of digits.

The are many possible real scenarios where Domain Adaptation combined with classification of digits could be useful, for example:

- Street house numbers: can differ a lot according to the city in which they are located. Here we can use domain adaptation, as we can train the model on SVHN (source dataset) and test it on a dataset with house numbers with different shapes/colours (target dataset).
- Speed signs digits: each speed signs can vary according to the traffic laws in force in a particular country.

2 State of the art

The techniques used to classify digits are mainly two: Support Vector Machine [2] (in particular Support Vector Classification, which can perform multi-class classification) and Convolutional Neural Networks.

One of the main drawbacks of SVM is the need to have the complete dataset during training phase, it can't be reinforced afterwards. Therefore Domain Adaptation can't be applied to these kind of classifiers, but an interesting experiment will be to use different domains between training and testing sets: for example, the first phase will be done with SVHN and then the latter will be computed with MNIST-M.

On the other hand, Domain Adaptation can be performed on CNNs: the samples are taken from various sources and they are used for training the neural network in order to be able to generalize its knowledge about the problem and, ideally, being capable to classify samples from domains it has never interacted with.

Domain adaptation can be *supervised* or *unsupervised*: in the first case both the source and the target domains are labelled. In the second approach, the data from the source domain has labels, but the samples from the target domain are unlabelled. Furthermore, in some cases multiple source domains are used to train the classifier(s), but we decided to work with **supervised single-source** domain adaptation in our project.

3 Methodology

3.1 Tools and libraries

To develop our project we used Python and the following libraries:

- Numpy;
- PyTorch;
- SciPy;
- Scikit-learn;
- Pandas.

We started off by creating a notebook with Google Colab, but soon we realized that the resources needed for our project are many more than what the platform offers, and that it slowed the computation down by a large margin. Thus, we decided to save the code on GitHub and run the project locally with PyCharm.

3.2 Datasets

The datasets used for the project are MNIST-M [6], SVHN [9] and SYNTH [5].

MNIST-M is a variation of MNIST, the most known set used for digits: it contains images of handwritten numbers and it is split in 60000 samples for training, and 10000 samples for testing. The images are 28×28 with labels that range from 0 to 9, corresponding to the actual number they represent. The difference from the original version is that the images are colored with patches extracted from BSDS500: Berkeley Segmentation Dataset 500, a dataset in which edges are extracted from real images.

Figure 1: MNIST-M samples

SVHN consists of images taken from *Google Street View* representing house numbers. This means that, contrary to the previous dataset, its samples come from a real-world scenario. The dataset is made of 73257 images for training and 26032 for testing, and it can be downloaded in two formats: the first one contains the original images (uncropped, with different resolutions and rotations) in `.png`; whereas the second is MNIST-like with 32×32 images centered around one digit but with **distraction** digits at the sides. The latter format is provided in two `.mat` files. The samples in both the variations are labeled ranging from 1 to 10, with the value 10 representing the digit 0. Therefore we decided to change them with the same labels used in MNIST-M (from 0 to 9).

Figure 2: SVHN samples

Finally, **SYNTH** is a synthetic dataset generated using Windows™ fonts. It contains around 500000 images that represent numbers with one, two or three digits. The authors of this dataset have manually applied transformations to the digits that include: orientation, blur, background and stroke colors, positioning; these variations were performed to make the data similar to SVHN yet distinct, since it comes without the noise present in the background of SVHN samples.

Figure 3: SYNTH samples

3.3 Feature extraction

3.3.1 CNN - Resnet34

We started extracting features with a pre-trained model of Residual Neural Network [7] ResNet50 contained in the `torchvision` library, but we soon figured out it would be unfeasible due to the huge memory requirements, since it extracts 100352 features for each sample of the dataset. Consequently, ResNet34 was our next choice: the output of its final layer *that we selected for our purpose* is $512 \times 7 \times 7$; this means that it provides 25088 features per sample, which are more manageable in terms of computation. As mentioned before, we tweaked the network to suit it for our purpose. For example we removed the last two layers (AvgPool and FullConnected layer) used for classification, because we just need to take out the features computed by the network after the convolution and pooling steps (taking out the output of the last max-pooling layer). Moreover, we have only implemented the `forward` method to allow the data to be propagated to the end, without the need of `backward` propagation. Finally, we set the network to use the set of pre-computed weights provided by the library.

3.3.2 HOG - Histogram of Oriented Gradients

In addition of extracting features using a CNN, we utilized also Histogram of Oriented Gradients [3]: that is a popular feature descriptor that is often used to extract features from image data.

A feature descriptor is a representation of an image that simplifies the image by extracting useful information and throwing away unnecessary information. In the case of the HOG feature descriptor, the input image is of size 64 x 128 x 3 and the output feature vector is of length 3780.

The HOG descriptor focuses on the structure or the shape of an object providing the edge direction as well, which should apply very well for our use case. This is done by extracting the gradient and orientation (magnitude and direction) of the edges. The complete image is broken down into smaller regions and for each region, the gradients and orientation are calculated. Finally the HOG would generate a Histogram for each of these regions separately. The histograms are created using the gradients and orientations of the pixel values.

The process of feature extraction with HOG applied to our datasets, provides images that looks like this:

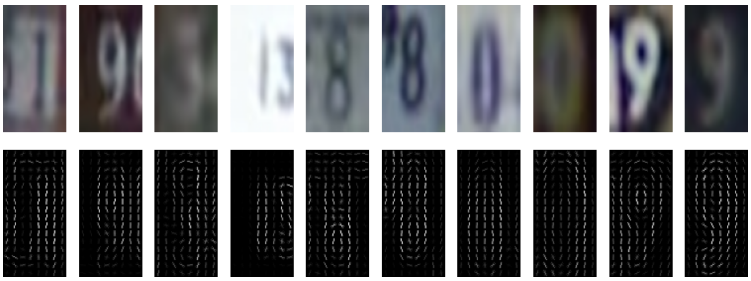

Figure 4: HOG applied to SVHN

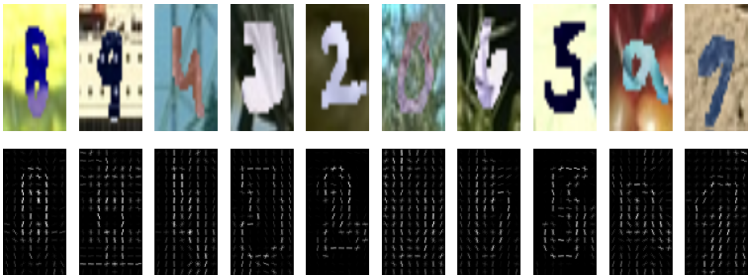

Figure 5: HOG applied to MNIST-M

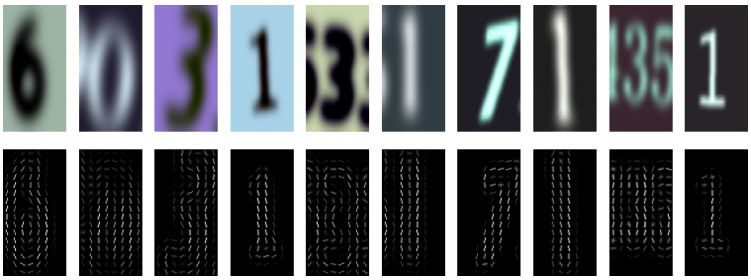

Figure 6: HOG applied to SYNTH

3.4 Feature reduction

To reduce the features, we utilized two methods provided by the `Scikit-learn` library: Principal Component Analysis [4] and Linear Discriminant Analysis [8].

3.4.1 PCA - Principal Component Analysis

PCA allows to reduce the number of features, while maintaining a certain amount of variance. This is done by calculating the covariance matrix and its eigenvalues. In this way, we can find out which have the largest dimension and their variance. In our project we made sure that the function returned the number of components needed in order to keep a variance greater than 90%, while the parameter `random_state` is the seed to have a constant selection of random data. This ensures that the results of the analysis are consistent between every execution.

3.4.2 LDA - Linear Discriminant Analysis

Linear Discriminant Analysis is very similar to PCA since both look for linear combinations of the features which best explain the data. The main difference is that the Linear discriminant analysis is a supervised dimensionality reduction and focuses on maximizing the distance between samples of different classes (inter-class) and minimize the intra-class distances between samples.

3.5 Classification

We modeled two classifiers: Support Vector Machine and k-Nearest Neighbours [1].

3.5.1 SVM - Support Vector Machine

SVMs are powerful models that allow to perform binary classification both with *linearly separable* data and *non-linearly separable* data. In the first case, it's simply a matter of finding the best hyperplane that separates the samples of the two classes. However having data that fit this condition is rare, therefore we need to either map the dataset to a different space in which a hyperplane exists (which is costly in terms of time, difficulty and computational power), or we can find an appropriate kernel function that implicitly implements the mapping during the calculation. This latter approach is the obvious way to go, thus we decided to work with:

- linear kernel $K(x, z) = \langle x, z \rangle$;
- radial basis function kernel $K(x, z) = \langle e^{-(\frac{||x-z||^2}{2\sigma^2})} \rangle$.

3.5.2 k-Nearest Neighbours

k-NN is a non-parametric technique that exploits the idea that the data can be separated with respect to a specific metric, and a new sample can be classified by counting the most recurrent class in its neighbourhood. The choice of how to separate the dataset is not always trivial, but the correct metric can lead to very precise classifiers. Another crucial step is finding the best number `k` of neighbours to be considered: choosing a small number may exclude some fundamental features of the samples, while a large number would make us lose the properties of the metrics we have chosen.

4 Experiment & Results

We decided to run the tests using 500 samples for each class, because higher quantities of data would have required an amount of computing resources unavailable to us. Thus, we worked with a total of 5000 samples for each dataset splitted in 4000 samples for the training set and 1000 for the testing set.

The results are divided in two groups: in the first one we extracted the features via CNN, instead in the second one we used HOG feature extractor. Hence, 36 different combinations between feature reductors, classifiers and datasets are obtained for each group. This allows us to analyze the performance of 72 different combinations.

4.1 Feature extraction with CNN

In our first tests we extracted features with CNN and reduced them both with PCA and LDA.

For SVM we tried to use the RBF kernel function and the linear one, but we decided to use RBF kernel because the performance were approximately the same.

The same thing happened with k-NN, since we tried different number of neighbours ($k = [10, 20, 50, 70, 90, 110, 130]$), but the performance were approximately the same (see Figure 7), therefore we decided to set $k = 5$ and obtain a little speedup on the computation phase.

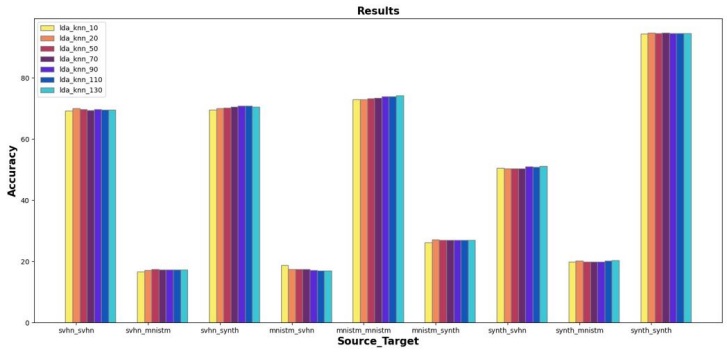

Figure 7: LDA + k-NN accuracy

For the sake of brevity we decided to collapse the means and standard deviations in two tables (Table 1 and Table 2), since the values only move by a slight margin when k changes.

Metrics	**Accuracy**	**Precision**	**Recall**
Mean	79.26	0.79	0.79
Std	11.25	0.11	0.11

Table 1: Source = target

Accuracy	**Precision**	**Recall**
33.75	0.35	0.34
20.26	0.21	0.21

Table 2: Source $\neq$ target

The Figure 8 shows the performances (based on accuracy) reached on the 36 different combinations using CNN as feature extractor.

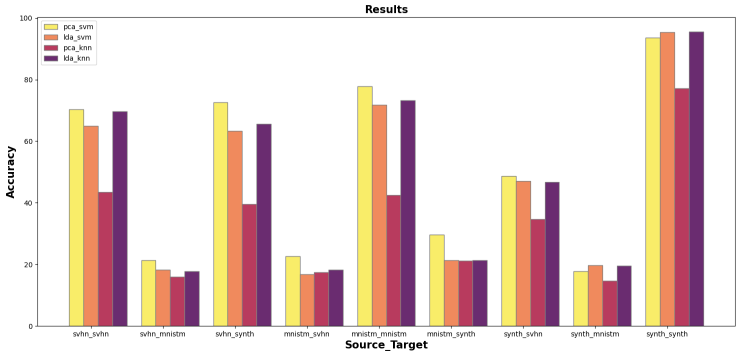

Figure 8: Accuracy graphs with features extracted on CNN

4.1.1 Performances (source = target)

The next four tables report the **average** results when the source domain is the same as the target domain:

Metrics	Accuracy	Precision	Recall
Mean	80.60	0.81	0.81
Std	11.85	0.12	0.12

Table 3: PCA + SVM

Metrics	Accuracy	Precision	Recall
Mean	77.37	0.78	0.77
Std	15.99	0.16	0.16

Table 4: LDA + SVM

Metrics	Accuracy	Precision	Recall
Mean	54.40	0.56	0.54
Std	19.75	0.19	0.20

Table 5: PCA + k-NN

Metrics	Accuracy	Precision	Recall
Mean	79.43	0.80	0.80
Std	14.03	0.14	0.14

Table 6: LDA + k-NN

Additionally, here are some confusion matrices of these cases:

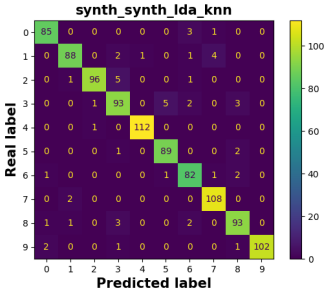

Figure 9: LDA + k-NN on SYNTH

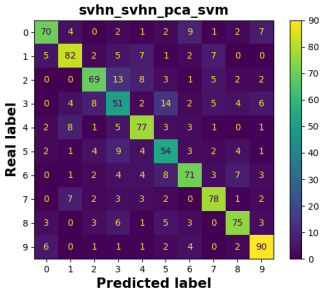

Figure 10: PCA + SVM on SVHN

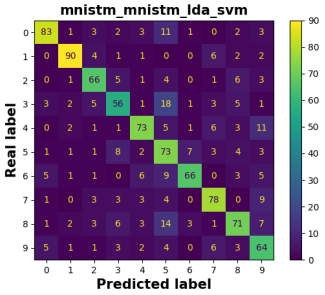

Figure 11: LDA + SVM on MNIST-M

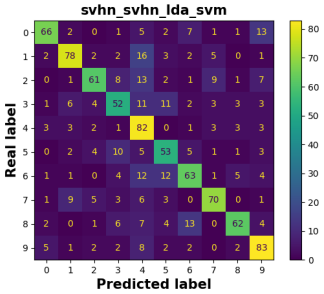

Figure 12: LDA + SVM on SVHN

4.1.2 Performances Domain Adaptation (source ≠ target)

Next tables represent the **average** results when the source domain is different from the target domain.

Metrics	Accuracy	Precision	Recall
Mean	38.87	0.42	0.39
Std	26.14	0.25	0.26

Table 7: PCA + SVM

Metrics	Accuracy	Precision	Recall
Mean	31.10	0.33	0.31
Std	19.46	0.21	0.20

Table 8: LDA + SVM

Metrics	Accuracy	Precision	Recall
Mean	23.88	0.28	0.24
Std	10.55	0.13	0.11

Table 9: PCA + k-NN

Metrics	Accuracy	Precision	Recall
Mean	31.57	0.33	0.32
Std	20.02	0.21	0.20

Table 10: LDA + k-NN

Finally, these are some confusion matrices of this latter case:

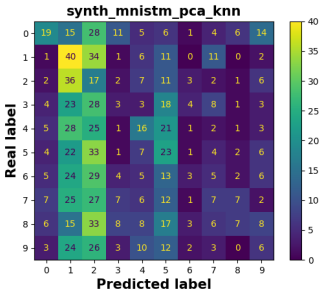

Figure 13: SYNTH → MNIST-M

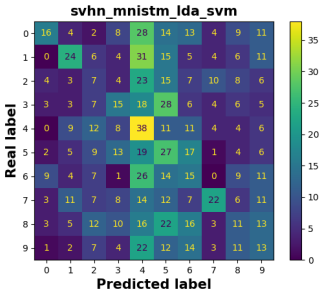

Figure 14: SVHN → MNIST-M

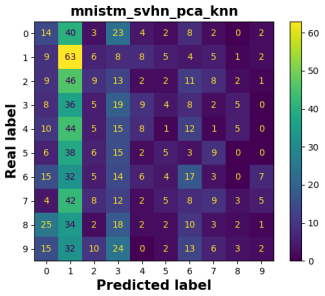

Figure 15: MNIST-M → SVHN

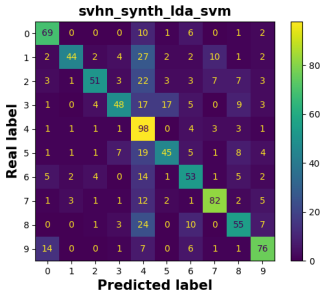

Figure 16: SVHN → SYNTH

4.2 Feature extraction with HOG

The following picture shows the performances (based on accuracy) reached on the 36 different combinations using HOG as feature extractor.

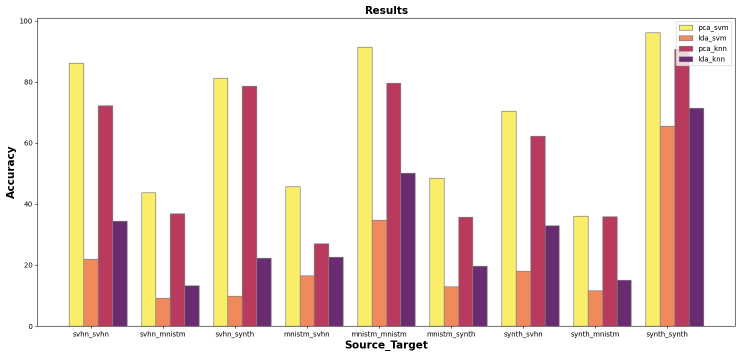

Figure 17: HOG accuracy

4.2.1 Performances (source = target)

The next four tables report the **average** results when the source domain is the same as the target domain:

Metrics	Accuracy	Precision	Recall
Mean	91.20	0.91	0.91
Std	5.00	0.05	0.05

Table 11: PCA + SVM

Metrics	Accuracy	Precision	Recall
Mean	40.74	0.55	0.41
Std	22.47	0.17	0.23

Table 12: LDA + SVM

Metrics	Accuracy	Precision	Recall
Mean	80.87	0.84	0.81
Std	9.36	0.08	0.10

Table 13: PCA + k-NN

Metrics	Accuracy	Precision	Recall
Mean	52.00	0.56	0.52
Std	18.62	0.16	0.19

Table 14: LDA + k-NN

4.2.2 Performances Domain Adaptation (source $\neq$ target)

Next tables represent the **average** results when the source domain is different from the target domain. We divided the **average** results in four tables:

Metrics	Accuracy	Precision	Recall
Mean	54.28	0.60	0.55
Std	17.50	0.14	0.17

Table 15: PCA + SVM

Metrics	Accuracy	Precision	Recall
Mean	13.02	0.20	0.13
Std	3.59	0.09	0.04

Table 16: LDA + SVM

Metrics	Accuracy	Precision	Recall
Mean	46.07	0.52	0.46
Std	19.93	0.19	0.20

Table 17: PCA + k-NN

Metrics	Accuracy	Precision	Recall
Mean	20.97	0.32	0.21
Std	6.97	0.09	0.07

Table 18: LDA + k-NN

5 Conclusions

We have found out that performing Domain Adaptation with SVM and k-NN on the chosen datasets is feasible but it will mainly give poor results more than halving the average accuracy.

The first noticeable aspect is that the worst performance obtained is when the dataset used for source or target is MNIST-M: this is coherent with our expectations because samples belonging to SVHM and SYNTH share a similar structure. This means that it is possible that the samples of SVHN and SYNTH share some of their major features, that could be a consequence of the transformations applied to the data during the creation phase of SYNTH.

Even though the other combinations (SVHN and SYNTH) gives better results, reaching more than 80% accuracy in some single cases, the average accuracy still can't be satisfactory. There is a chance that using higher numbers of samples during the training phase could improve the performances.

As we can see from some random misclassified samples in Figures 18, 19 and 20, we admit that some digits would be pretty hard to classify also for a human being. Thus, the quality of the samples in the dataset plays a big role on the performances reached.

Figure 18: Misclassified samples with SVHN

Figure 19: Misclassified samples with MNIST-M

Figure 20: Misclassified samples with SYNTH

We noticed also some differences between the use of the two Feature Extractor, in particular:

- the best average accuracy (54%) was reached using HOG as feature extractor combined with SVMP and PCA. This behaviour (HOG and SVM working very well together) was also mentioned in the literature, and so is coherent with our expectation;
- there is a drastic drop of performances when LDA is used in combination with HOG;
- there is a drop of performances when k-NN is used in combination with HOG compared to CNN;
- from the point of view of classifiers we noticed that extracting features with CNN, SVM performs better combined with PCA, while k-NN works better when combined with LDA.

A possible future work is to analyze the performance introducing a CNN as a classifier, and maybe trying other methods to extract features since we noticed that they play an important role in the resulting performances.

References

[1] Altman, N.S.: An introduction to kernel and nearest-neighbor nonparametric regression. The American Statistician **46**(3), 175–185 (1992). https://doi.org/10.1080/00031305.1992.10475879, `https://www.tandfonline.com/doi/abs/10.1080/00031305.1992.10475879`

[2] Cortes, C., Vapnik, V.: Support-vector networks. Machine learning **20**(3), 273–297 (1995)

[3] Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. In: 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05). vol. 1, pp. 886–893 vol. 1 (2005). https://doi.org/10.1109/CVPR.2005.177

[4] F.R.S., K.P.: Liii. on lines and planes of closest fit to systems of points in space. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science **2**(11), 559–572 (1901). https://doi.org/10.1080/14786440109462720

[5] Ganin, Y., Lempitsky, V.S.: Unsupervised domain adaptation by backpropagation. In: ICML (2015)

[6] Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., Marchand, M., Lempitsky, V.: Domain-adversarial training of neural networks (05 2015)

[7] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 770–778 (2016). https://doi.org/10.1109/CVPR.2016.90

[8] Li, S.Z., Jain, A. (eds.): LDA (Linear Discriminant Analysis), pp. 899–899. Springer US, Boston, MA (2009)

[9] Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A.: Reading digits in natural images with unsupervised feature learning. NIPS (01 2011)