Transfer Learning with Network Traffic Data

Byron Barkhuizen
Télécom SudParis
for Submission
at
https://github.com/byronbark/IOTProject

ABSTRACT
UPDATED—18 December 2020. Transfer learning is a developing research field and framework in artificial intelligence to assist the development of new models, typically in the case of limited data or label availabilities. In other instances, it may simply be a desire to reduce the time for creating a new model, or an attempt to improve the accuracy of a model without a large new dataset being collected. The concept describes learning between a source and target domain or task in the same way that a human learns. It relies on the idea that there exists similarities or knowledge that can be transferred to perform a task in a target domain at higher level of accuracy. There are various ways of performing transfer learning and they are dependent on the type of data available and the differences in the source and target domain or tasks where learning is desired.

Keywords
Transfer Learning, Representation Learning, Domain Adaptation, Python, Network Traffic.

INTRODUCTION
Transfer learning is the establishing of a relationship between a source domain and a target domain, along with source tasks and target tasks. DARPA (Defense Advanced Research Projects Agency) defines the mission of transfer learning as 'the ability of a system to recognize and apply knowledge and skills learned in a previous task to novel tasks. The existence or assumption that there is some similarity between the source and target is the theoretical basis for the application of transfer learning (1). The features within a source or target domain have some probability distribution. When these features distributions are the same then the new dataset can essentially just be classified in the same way that the original dataset was. If the source and target dataset do not conform to the same probability distribution or the same features, then this cannot be done accurately. However, this does not mean that a model needs to be completely re-trained and re-classified on the new dataset, there theoretically can exist some knowledge that can be transferred between the two datasets.

The final goal for a transfer learning application is to reduce the need for large quantities of data collection to be necessary, to boost the speed at which a new model in a target domain can be trained, and the accuracy achieved through these models in novel environments.

STATE OF THE ART
Currently the simplest application of transfer learning involves re-using all or some parts of a pre-existing model. These models can be trained on almost anything, however a transfer learning application with this approach will perform better if there exist some similarities between the source and target domain. In this case the source domain would be the dataset that the initial model was trained on. A popular pre-trained model that is used is Google's ResNet which is trained on an enormous number of images for the task of classification. This pre-trained convolutional neural network exists of many layers, and the initial layers closest to the input learn some low-level features such as edges or gradients. These features are application agnostic, they exist on all images that we may use. For this reason, it is possible to 'cut' this model so we keep only this distribution of features, while getting rid of higher-level features that may not exist in the target domain. These features become more specific to the classes that the initial model was trained on and may be entirely unnecessary. This application of transfer learning is very easy to implement, however often it will not achieve a very high level of accuracy until it is retrained on the new classification task with some newly labeled data in the target domain.

The basis of current approaches works on the assumption that the training and some future data are either in different feature spaces or have different distributions. An extra consideration can also be made as to whether the task is the same or not.

EXPERIMENTS

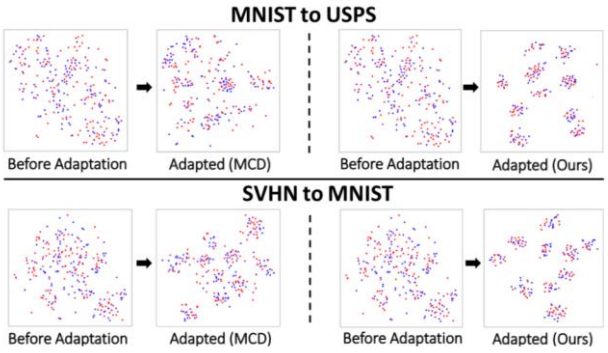
Data Preprocessing
The data that is available in our experiments are PCAP (Packet Capture) files collected over various time spans for many devices. The PCAP is separated with the SplitCap tool, as well as the CICFlowMeter tool. We are left with raw PCAP files that are separated by flows and MAC address; these MAC addresses are suitable as our labels. We can use some windows shell scripts and small Python programs to

process these PCAP files and create binary files along with CSV files to represent the activity of individual devices. The data is normalized after they are split into test sets to not influence each other. At the end of the preprocessing we are left with 28x28 pixel images, and CSV files that represent 84 calculated time series features. The images are PNG files and can be easily processed by the NumPy library within Python.

Domain Adaptation

Domain adaptation refers to bridging the gap between the differences in the source and target domains. Essentially, we want to minimize the differences in their representations. When they can be similarly represented (their features or their distributions are the same) then learning has occurred. The new representation can then be used to train an initial model where we have rich data (source domain), and this model can be used on the target domain to achieve accurate results. More specifically, one method of performing domain adaptation is discriminative feature alignment.

An application of this is explored in the GitHub repository. It uses 3 different image sets, the popular MNIST image dataset along with SVHN and USPS. In the following images we can see a reduced dimensionality representation of what domain adaptation attempts to achieve. The individual points represent different images in latent space. Once a new representation is learned (after adaptation) we are again able to represent them in the latent space and there are clear clusters that are formed. In the case of the image dataset they are clusters that will each represent a different digit. The grouping before the adaptation does not exist and therefore any attempt to classify in the target domain would achieve extremely poor results.



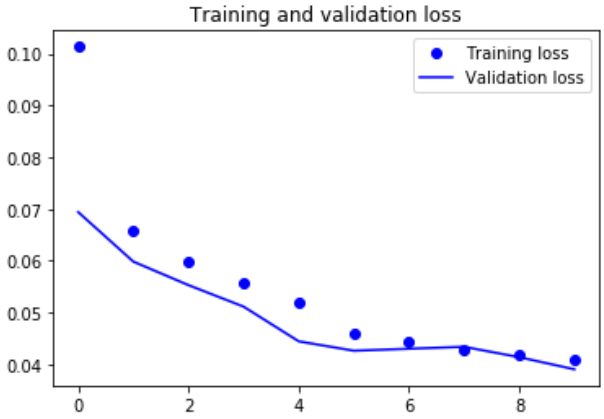
The application of this to our dataset as desired would be the following.

- Represent both images and CSV files as a NumPy array (pre-processing on CSV data) so that their initial representations are the same
- Learn best feature representation and modify both source and target data accordingly (such that on our latent space representation we will have one clear class).
- Train source model, test on target data on the new representation
- Testing can be done with any one class classifier

Autoencoder

Autoencoder is the most basic feature representation implementation. A single connected layer attempts to determine a minimum representation for some input data, such that it can reconstruct it from the fewest number of nodes possible. Through doing this it learns a representation of the data that belongs to a single class. If we then apply this model to some new data, alongside a conditional calculation such as Euclidean distance, we can determine whether a new data point belongs to this one class.

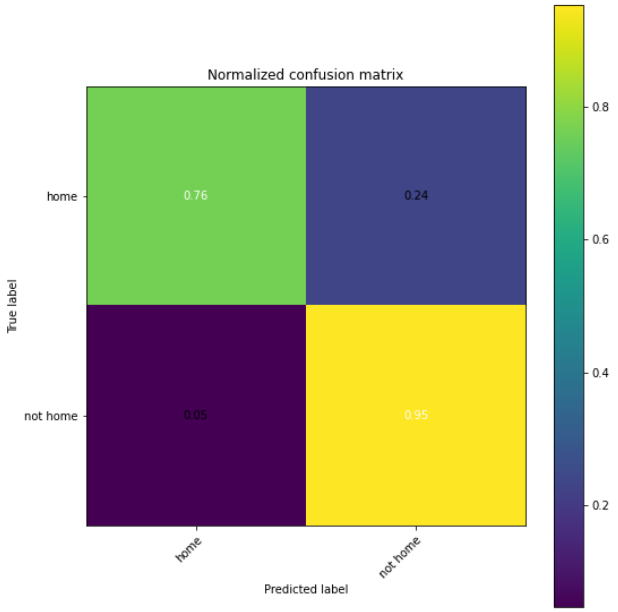
An implementation of this can be found in the GitHub repository and it presents a standard autoencoder architecture. With our CSV data the following loss data can be achieved on the target data. The loss represents the distance of the data points when used as the input in the initial model. A low loss means there is very little deviation and that it likely belongs to the 'home' class. In this instance the target domain data consisted only of 'home' data.



This is not an application of transfer learning as no interaction between the source and target domain has occurred yet, we are simply transferring the model that was initially created. The concept of autoencoder could be extended to use transfer learning as we will explore in the next application.

Neural Network (Shallow)

This experiment involved looking at the binary classification of a single device coupled with the data of multiple other devices. It is like anomaly detection in that we are identifying whether it belongs to this known class or now. The known class was represented by a 1. The confusion matrix from this experiment can be seen below.



We can observe that this neural network is able to identify both classes to a high degree of accuracy. This is not an example of transfer learning; however we can achieve transfer learning by freezing some layers of the neural network and retraining the model on this data.

EXPERIMENTATION

This section will expand upon some of the transfer learning approaches tried, including some interpretations of papers that exist on the subject.

TLDA (Deep Autoencoder)

While a basic autoencoder attempts to learn a good representation of the input data such that it can reconstruct it with high accuracy, an autoencoder approach that seeks to apply transfer learning must learn a representation that is good for both domains. This can be achieved by using some new data in the target domain in concurrence with the data that already exists in the source domain.

We train a stacked autoencoder on both the source and target domain to learn good representations of both individually. This supplies us with the weights for some layers as an initial basis. This model can be created at any time and saved as the source model until some target data exists to train a target autoencoder. After these are initialized, we will calculate some partial derivatives (this is the basis of most deep learning models where a gradient is calculated) to iteratively update values that will be tested against KL-convergence conditions. KL-divergence is a method of evaluating the differences between two probability distributions, therefore when this condition is satisfied, we can assume some convergence between the two domains. The process outlined in the paper is shown below, and an initial Python implementation of this can be found on the GitHub repository.

Algorithm 1 Transfer Learning with Deep Autoencoders (TLDA)

Input: Given one source domain $\boldsymbol{D}_s = \{\boldsymbol{x}_i^{(s)}, y_i^{(s)}\}|_{i=1}^{n_s}$, and one target domain $\boldsymbol{D}_t = \{\boldsymbol{x}_i^{(t)}\}|_{i=1}^{n_t}$, trade-off parameters α, β, γ, the number of nodes in embedding layer and label layer, k and c.

Output: Results of label layer $\boldsymbol{z}$ and embedded layer $\boldsymbol{\xi}$.

1. Initialize $\boldsymbol{W}_1$, $\boldsymbol{W}_2$, $\boldsymbol{W}_2^{'}$, $\boldsymbol{W}_1^{'}$ and $\boldsymbol{b}_1$, $\boldsymbol{b}_2$, $\boldsymbol{b}_2^{'}$, $\boldsymbol{b}_1^{'}$ by Stacked Autoencoders performed on both source and target domains;
2. Compute the partial derivatives of all variables according to Eqs. (14), (15) (16) and (17);
3. Iteratively update the variables using Eq. (18);
4. Continue Step2 and Step3 until the algorithm converges;
5. Computing the embedding layer $\boldsymbol{\xi}$ and label layer $\boldsymbol{z}$ using (9), and then construct target classifiers as described in Section 3.3.

Discriminative Feature Alignment

Initially based on a paper to classify handwritten numbers from 0 to 9 this code is based on the paper by Jing Wang and is written in Python. The paper outlines some methods for the application of domain adaptation. From the survey of transfer learning we discovered that domain adaptation is the correct paradigm for our problem as the task (classification) is the same however the domains differ in their feature distributions. There are two approaches for a domain adaptation problem, and these are instance transfer and feature representation transfer.

The solution offered in the ‘TLwithDomainAdaptation’ folder uses an encoder and decoder approach to align features between two similar but different datasets. The paper outlines various algorithms to reach this target. The code written here is an adaptation of earlier work in domain adaptation.

The code includes various python files for data processing, creating training and testing sets including their proper labeling. The code also contains a setup for the encoder and decoder model with specifications for all layers

Transfer Component Analysis

Transfer component analysis, or TCA, is another feature representation method for performing transfer learning. This transfer learning is well suited to handle both image and textual data as it functions like an autoencoder with many features. The theory behind this method of transfer learning involves techniques to achieve dimensionality reduction. When only a source domain is involved this is essentially a technique that is the same as an autoencoder based classifier. However, the transfer of knowledge comes into play when there are multiple domains involved, and there is a lack of labels in the target domain.

The exact steps taken performing TCA on some datasets involve reducing the statistical distribution differences between the concerned datasets.

The first method to do this is called the Maximum Mean Discrepancy (MMD) which is a method for computing the overall differences in the means of features. This method is not as sensitive to feature size which makes it applicable to transfer learning for images. Typical approaches such as the KL divergence for calculating differences In distributions are extremely difficult to perform when the number of features are too high. From the paper the goal or cost function that we are seeking to minimize can be demonstrated within the following image:

Algorithm 1: TCA

Input: Source domain data set $\mathcal{D}_S = \{(x_{S_i}, y_{src_i})\}_{i=1}^{n_1}$, and target domain data set $\mathcal{D}_T = \{x_{T_j}\}_{j=1}^{n_2}$.

Output: Transformation matrix W.

1: Construct kernel matrix K from $\{x_{S_i}\}_{i=1}^{n_1}$ and $\{x_{T_j}\}_{j=1}^{n_2}$ based on (2), matrix L from (3), and centering matrix H.

2: (Unsupervised TCA) Eigendecompose the matrix $(KLK + \mu I)^{-1}KHK$ and select the m leading eigenvectors to construct the transformation matrix W.

3: (Semisupervised TCA) Eigendecompose matrix $(K(L + \lambda)\mathcal{L}K + \mu I)^{-1}KH\widetilde{K}_{yy}HK$ and select the m leading eigenvectors to construct the transformation matrix W.

4: **return** transformation matrix W.

or:

$$\mathrm{MMD}(X_S^*, X_T^*) = \mathrm{Tr}((\mathbf{KWW}^\top\mathbf{K})\mathbf{L}) = \mathrm{Tr}(\mathbf{W}^\top\mathbf{KLKW})\,.$$

To reduce the differences between two distributions, it is very intuitive to seek a shared feature representation across domains, but it is not a simple process. Such a representation is intended to mitigate the shift occurred between the source and target datasets. To this end, we present the TCA technique (in its unsupervised form, i.e. not requiring any labeled target data) designed to extract meaningful transfer components from the original data belonging to different but related domains. The purpose of algorithms in a DA setting is to find a mapping function φ, practically a transformation matrix W, whose aim is to preserved the main properties of the two distributions while also reducing the distances between them. The core mechanisms to achieve this is by performing eigen decomposition and statistical analysis of the datasets iteratively until the function previously mentioned is minimized.

The results obtained for this method of transfer learning is early, however through using various mixes of data sizes and domains for IoT devices the detection of google home has been as high as 80%. This result is promising and merits further exploration with the transfer component analysis paradigm.

[5] F. Zhuang, X. Cheng, P. Luo, S.J. Pan, and Q. He, "Supervised representation learning with double encoding-layer autoencoder for transfer learning," ACM Trans. Intell. Syst. Technol., vol. 9, no. 2, pp. 1–17, Jan. 2018.

REFERENCES

[1] S.J. Pan and Q. Yang, "A survey on transfer learning," IEEE Trans. Knowl. Data Eng., vol. 22, no. 10, pp. 1345–1359, Oct. 2010.

[2] Z. Wang, Z. Dai, B. Poczos, and J. Carbonell, "Characterizing and ′ avoiding negative transfer," in Proc. IEEE Conference on Computer Vision and Pattern Recognition, Long Beach, Jun. 2019, pp. 11293– 11302.

[3] S.J. Pan, I.W. Tsang, J.T. Kwok, and Q. Yang, "Domain adaptation via transfer component analysis,"IEEE Trans. Neural Netw., vol. 22, no. 2, pp. 199–210, Feb. 2011.

[4] C. Chen, Z. Chen, B. Jiang, and X. Jin, "Joint domain alignment and discriminative feature learning for unsupervised deep domain adaptation," in Proc. 33rd AAAI