Transfer Learning with Network Traffic Data for the Identification of Anomalous Traffic

Byron Barkhuizen
Télécom SudParis
for Submission
at https://github.com/byronbark/IOTProject
with the help of

Marwan Lazrag
PhD Student, Télécom SudParis

Gregory Blanc
Associate Professor, Télécom SudParis

Houda Jmila,
PhD Student,
Télécom SudParis

ABSTRACT

UPDATED—3 March 2020. Transfer learning is a developing field and framework in artificial intelligence to assist the development of new models, typically in the case of limited data or label availability, or to bootstrap the process for a new unknown environment. 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 brain learns a new task. It relies on the idea that there exists similarities or knowledge that can be transferred to perform a task in a target domain at a 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, packet capture, network traffic.

INTRODUCTION

Transfer learning is the attempt to establish 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'. A computer being able to do a task does not imply intelligence, however the ability of a computer to learn to do a new task does (Russel, 2002) and is a step towards true artificial intelligence. The existence or assumption that there is some similarity between the source and target is the theoretical basis for the application of transfer learning (S.J Pan, 2010). One consideration for the applicability of transfer learning is whether 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 in order to improve the model metrics in the target domain.

The ultimate goal for a transfer learning application is to reduce the reliance on large quantities of data collection which is traditionally the only method of obtaining highly performant machine learning models, and to boost the speed at which a new model in a target domain can be trained, as well as improving the accuracy achieved through these models in novel environments. There are an extremely high number of approaches to transfer learning, and the field is a long way from establishing a best practice approach for taking on this framework.

STATE OF THE ART

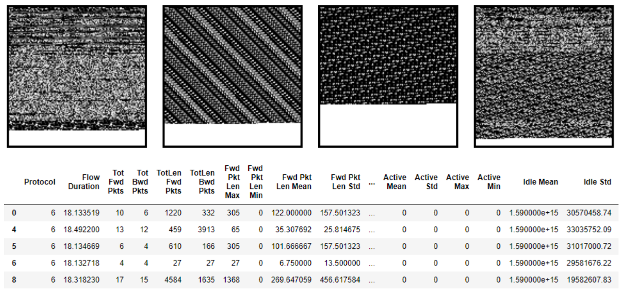
Currently the most approachable application of transfer learning involves re-using all or some parts of a pre-existing model. In the machine learning world the most fitting type of architecture for such an approach is a neural network. These models can be trained on almost anything and put into use in a new domain, however a transfer learning application with this approach will perform better if there exist some similarities between the source and target domain, namely the concept of a feature within each domain concerned. 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, with varying depth (eg. ResNet-50 corresponds to a model with 50 layers of neurons). This pre-trained convolutional neural network can exist of hundreds of layers, and the initial layers closest to the input learn some low-level features such as edges or gradients. The higher level feature layers are where the classification begins to take form, with these basic or atomic features being combined to identify an object in an image, this is not specific to images but it is the most understandable representation (S.J Pan, 2010). 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 low level features, while getting rid of higher-level features that may not exist in the target domain. These features, or profiles of 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 re-trained on the new classification task with some newly labeled data in the target domain. The core concept of this is that transfer learning can be achieved by maintaining some knowledge of the source domain to be reused 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. Often the task of transfer learning can lead to a negative transfer (Wang et. al, 2019). This is the case when the similarities between the source and target domain are non-existent or they are too unrelated for transfer learning to yield positive results. This can lead to wasted resources, in terms of time and performance.

CONCEPTS

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 addresses; these MAC addresses are suitable as our labels as they correspond to a device and are able to differentiate the devices for training models. 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. This step can be considered the precursor to model selection. The chosen representation of the domains needs to be the same and it will determine which models are viable to be used. Some data types will not work with certain models. An image can not be easily used in a regression model as a CSV file of time series information can be used. The data is normalized after they are split into test sets to not influence each other. This means the test sets and the training sets are normalized separately but by the same method. This shifts certain values into a range that is usable and will not make the model overly biased towards large or small values. At the end of the preprocessing we are left with m by m pixel images, and CSV files that represent 84 calculated time series features along with flow identifying features including source and destination addresses, port number and protocol numbers associated. For the purpose of all experimentation involved these features were not included as they could lead to overtraining on these features, and therefore low transferability to a new target environment. Many of these features can also be spoofed and will nullify the use of a machine learning model that considers them as features. For that reason part of the dataset cleaning process involves removing them, however the protocol number is used to filter only for TCP traffic so that only this type of traffic which is more uniform flow to flow is concerned. The images are PNG files and can be easily processed by the NumPy library within Python. Similarly the CSV files can be easily processed by these tools in order to be passed into a model.



	Protocol	Flow Duration	Tot Fwd Pkts	Tot Bwd Pkts	TotLen Fwd Pkts	TotLen Bwd Pkts	Fwd Pkt Len Max	Fwd Pkt Len Min	Fwd Pkt Len Mean	Fwd Pkt Len Std	...	Active Mean	Active Std	Active Max	Active Min	Idle Mean	Idle Std
0	6	18.133519	10	6	1220	332	305	0	122.000000	157.501323	...	0	0	0	0	1.590000e+15	30570458.74
4	6	18.492200	13	12	459	3913	65	0	35.307692	25.814675	...	0	0	0	0	1.590000e+15	33035752.09
5	6	18.134669	6	4	610	166	305	0	101.666667	157.501323	...	0	0	0	0	1.590000e+15	31017000.72
6	6	18.132718	4	4	27	27	27	0	6.750000	13.500000	...	0	0	0	0	1.590000e+15	29581676.22
8	6	18.318230	17	15	4584	1635	1368	0	269.647059	456.617584	...	0	0	0	0	1.590000e+15	19582607.83

Figure 1: Examples of data representation after preprocessing

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 after, before or during they have been represented. When they can be similarly represented (their features or their distributions are the same) then learning has occurred (Kouw et. al, 2018). In many cases this means that the size of the representation can become smaller, as a subset of features is chosen that is able to represent both domains, this is the case in TCA and similar methods.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 data poor target domain to achieve greater results. More specifically, one method of performing domain adaptation is discriminative feature alignment (DFA).

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. It is one of the most often used data sets to test various concepts within the artificial intelligence and machine learning community. It is also particularly useful for transfer learning due to the nature of the data sets. The task of classification of the digits remains the same however the digits can be written slightly differently therefore they can be considered as different domains. In this scenario the features within the images are identical (eg. edges, gradients) and it is their distributions that might differ. 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, that is high dimensional data that has been reduced to a 2d space in order to visualize a grouping between the features. 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 correspond to a different digit. The grouping before the adaptation does not exist in the target domain and therefore any attempt to classify in the target domain would achieve extremely poor results.

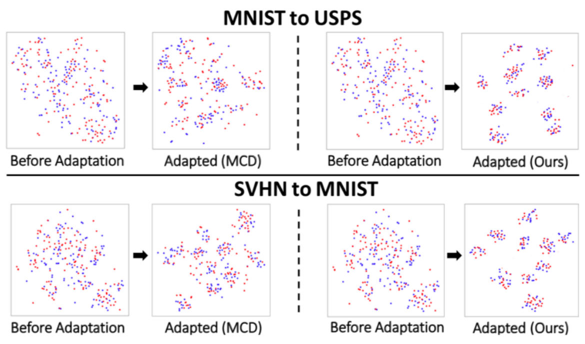


Figure 2: Dimensionality reduction of DFA

The application of this to our datasets 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, if we were able to visualize it, we will have clear class separations).
- 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 fully connected layer (all neurons output to all other neurons) 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 distance calculation such as Euclidean distance, we can determine whether a new data point belongs to this one class.

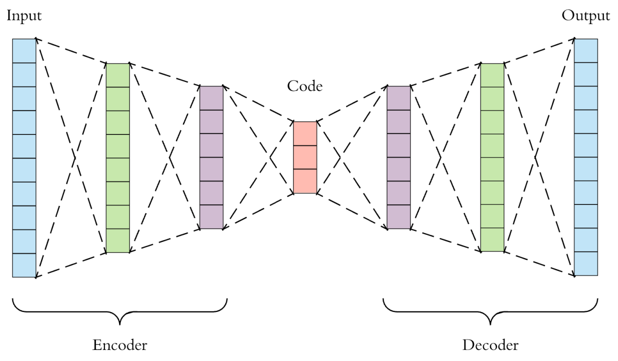


Figure 3: Autoencoder architecture

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.

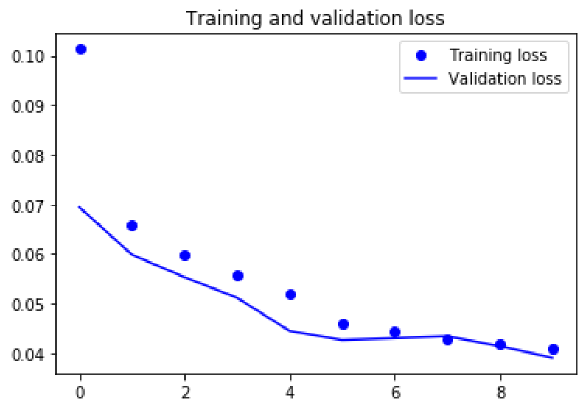


Figure 4: Performance of an autoencoder

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, or at least it is a developing idea, as we will explore in the following experiment.

EXPERIMENTATION

This section will expand upon some of the transfer learning approaches tried, including some interpretations of papers that exist on the subject. The general architecture for experimentation can be seen in figure 5.

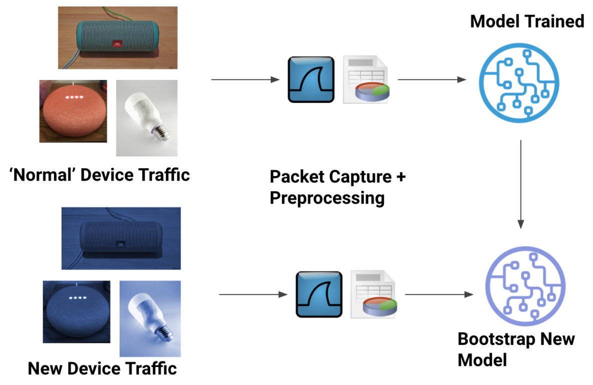


Figure 5: Flow of information for experimentation

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 not. The known class was represented by a 1 and everything else was a 0. The confusion matrix from this experiment can be seen below.

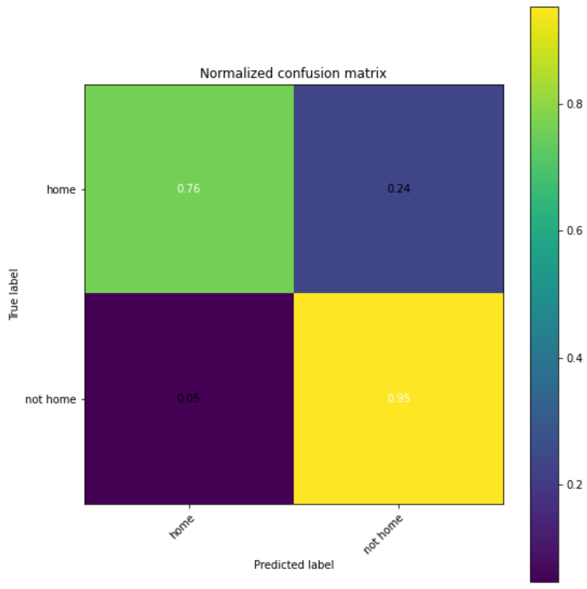


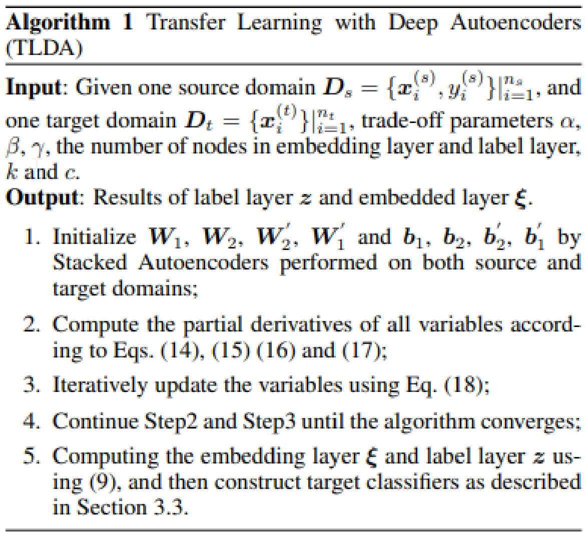
Figure 6: Confusion matrix shallow neural network

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. This model can also be used in concurrence with models where the weights are not adjusted and instead modify the data themselves (TrAdaBoost, TCA) as a classification stage.

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 try to learn a representation that is good for both domains. This step does not need to be performed at the same time, the training of the initial model in the source domain and the target domain can and should be trained at different times. 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. A stacked autoencoder refers to an autoencoder architecture where the intermediary layers which are normally considered bottleneck layers are separated into sparse neural networks like layers where all nodes are connected. This supplies us with the weights for some layers as an initial basis (Kouw, 2018). 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. It is a different method for calculating distances, but specifically concerned with the differences between distributions and therefore the distance between different distributions. More simply, we can calculate the distance between points after their reconstruction, or we can calculate the distance between the distributions of those points. The latter helps us to concurrently align two models to be more similar if we are able to measure it. The process outlined in the paper is shown in figure 7, 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 $\alpha$, $\beta$, $\gamma$, 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.
```

Figure 7: Process for applying TLDA

The results for applying TLDA were very promising. Despite the existing code being hard to manipulate we can easily use different data sets to test the efficacy of such an approach. The initial data is stored with their labels and we tell the model between which two data sets to perform the algorithms and attempt to classify the data. Within sub 20 epochs it was normally possible to get above 60% accuracy. The highest accuracy reached was 68%, however it still identified a lot of traffic incorrectly. This improves as we train on a much higher number of epochs but whether this is useful for the use case of transfer learning is a concern.

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 features differences between the concerned datasets. The ultimate goal is to be able to represent both data sets minimally, while improving or maintaining a level of accuracy.

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 (Pan et. al, 2010). 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 figure 8.

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

```
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 $W$.
```

Figure 8: Steps involved in TCA

To reduce the differences between two datasets, 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 domain shift caused 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 transferable components from the original data belonging to different but related domains. The purpose of algorithms in a domain adaptation setting is to find a mapping function φ, practically a transformation matrix W, whose aim is to preserve 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 code to perform TCA is located on the github, however there are many issues concerning this approach. Experiments that were performed were not feasible with the resources available, and while it may be possible with more powerful machines the question is whether this is desirable. The scenario of identifying malicious IoT traffic involves rapid deployment of a model in a new environment that can adapt, and TCA may present a big time constraint in this case.

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 71% on very small feature sizes.

TrAdaBoost

TrAdaBoost is an extension of a common boosting algorithm called AdaBoost. Boosting algorithms deal with data scarcity and it is a method that does not deal with modification of the weights in a model or the data itself. AdaBoost deals with a single domain and attempts to increase the number of usable data points that won't bias the model it will be used to train in. The architecture can be seen in figure 9.

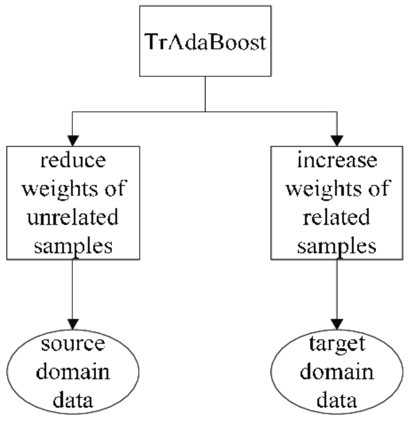

TrAdaBoost
reduce weights of unrelated samples
increase weights of related samples
source domain data
target domain data

Figure 9: Steps involved in TrAdaBoost

TrAdaBoost extends the AdaBoost concept to a source and target domain, into the realm of transfer learning. It involves the use of a classifier in order to identify whether a certain point belongs to the source or target domain. TrAdaBoost defines the source as those points with a label 0 and the target as label 1 (this could be any other label, it's just to differentiate the two domains). As the classifier correctly classifies a point into one of the two domains the weights of those points are adjusted according to their importance to the classification of the target domain. Ultimately we end up with data from the source domain that can be used to help classify target domain data points. This approach overcomes speed of training, data scarcity and lack of labels.

The experiments performed in TrAdaBoost involved the use of labelled source data points as 'home' and various other devices that would be considered as one class together. This is because we want to maintain the concept of anomaly detection which usually deals with just two classes (also called one class classification, the anomalous class). When used with a neural network as the modelling step, and TrAdaBoost to achieve a new data set to be used within the neural network the accuracy was 59.7% at its highest point. This was done by using half as many data points in the target domain as the source domain in order to simulate data scarcity. As the number of data points available to use in the target domain go up so did the accuracy.

DISCUSSIONS

In the figure 10 below we see a summary of the results achieved across experiments performed.

Type	Model	Device	F-1 Score**
No TL	OC-SVM	Google Home	0.3321
No TL	Autoencoder	Various	0.256
SoTA* TL	TrAdaBoost	Google Home	0.597
SoTA TL	TLDA	Google Home	0.646

Figure 10: Results from experiments

Without any attempt at transfer learning between the two datasets the results were very poor. The one class SVM performed the best, which is as expected since it is a core model within anomaly detection. The SVM is able to learn a good representation of the anomalous class if we concern ourselves with only the source domain, however when this same SVM is applied on a new similar situation (classifying the same device) in the target domain it is not able to perform well. The accuracy drops drastically, typically from 65-70% down to 33% on average.

The autoencoder performed worse again, reaching an accuracy as low as 25.6%. The autoencoder also did not perform very well on its own in just the source domain, however this may be improved with further changes to the architecture used and the training time. Autoencoders remain a good method for anomaly detection, particularly where the different classes possess the ability to be separated.

TrAdaBoost and TLDA were the two best models involving transfer learning. TrAdaBoost allowed a classifier to reach up to 60% between the source and target domains. TrAdaBoost is a less complex model to implement and understand and specifically tackles the issue of data scarcity more than any other model involved, however the performance is dependent on the classifier that it is coupled on so it is hard to evaluate its effectiveness without first having a good classifier. TLDA is an ensemble method in that it does not rely on anything beyond the preprocessing steps to determine how well it performs. TLDA reached up to 65% when transfer between the source and target domain of google home devices. TLDA was extremely flexible when it came to the use of source and target domain and it was even able to remain performant when the source domain was a set of MNIST handwriting data for a target domain of traffic images.

FUTURE WORK

In the future of work on transfer learning to identify anomalous traffic from network traffic data there are some important preliminary steps that need to be taken. Due to the nature of studying within transfer learning and the variance of the approaches that can be taken and how they can be implemented it is important to establish a solid foundation. This foundation consists firstly of a good representation of the data, which is important for any machine learning task. This should be done in collaboration with an expert in the identification of devices on networks and a study into which features are important to be represented, and which could present issues. While it is easy to say that the traffic flow should be represented as an image, or as a CSV file it is not easy to say whether these representations offer enough uniqueness between different devices. A more concrete understanding of what identifies a device would serve to make an earlier decision on which representation to use and therefore which subset of models to explore.

Along with this point there is a need for access to resources and data in as close to real time as possible to assist the development phase of any model with up to date data that can present further insights into the nature of the data. This could be assisted by collaboration with other researchers in the field that are training similar models in order to explore what is working and what is not, because there is a lack of openly available frameworks for this problem, or any similar problem.

Furthermore, the evolution of transfer learning means that there are very few shared phases between each approach. Across just the 3 model's experiments presented there were 3 unique input formats for the models where code existed. For models that are all programmed in Python there should be less variability and therefore more opportunity to modify and experiment with inputs into a model. This presents a need for a more uniform phase of data preprocessing, or a focus on fewer number of transfer learning models in future experimentation. TLDA was one of the earliest proposed models in the project, it was complex to conceptualize what was being done through the transformation of deep autoencoders and even more complex to tailor to our own needs however it was the most suitable for the task as identified (Chen et. al, 2012) and ultimately the most successful for the scenario. Despite this it was skipped earlier on to focus on various other approaches that turned out to not be useful.

As far as experiments go there is a lot of opportunity to perform more trials with different parameters for the input such as different number of features and different size data sets to suit different proposed scenarios and test the applicability in production. This would be greatly benefited by more uniform steps in the earlier phases.

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

[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.

[6] Kouw, W. M., & Loog, M. (2018). An introduction to domain adaptation and transfer learning. *arXiv preprint arXiv:1812.11806*.

[7] Russell, S., & Norvig, P. (2002). Artificial intelligence: a modern approach.

[8] Pan, S. J., Tsang, I. W., Kwok, J. T., & Yang, Q. (2010). Domain adaptation via transfer component analysis. IEEE Transactions on Neural Networks, 22(2), 199-210.

[9] 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.