Multi-Stage Transfer Learning with an Application to Selection Process

Andre Mendes[1] and **Julian Togelius**[2] and **Leandro dos Santos Coelho**[3]

Abstract. In multi-stage processes, decisions happen in an ordered sequence of stages. Many of them have the structure of dual funnel problem: as the sample size decreases from one stage to the other, the information increases. A related example is a selection process, where applicants apply for a position, prize or grant. In each stage, more applicants are evaluated and filtered out and from the remaining ones, more information is collected. In the last stage, decision-makers use all available information to make their final decision. To train a classifier for each stage becomes impracticable as they can underfit due to the low dimensionality in early stages or overfit due to the small sample size in the latter stages. In this work, we proposed a *Multi-StaGe Transfer Learning* (MSGTL) approach that uses knowledge from simple classifiers trained in early stages to improve the performance of classifiers in the latter stages. By transferring weights from simpler neural networks trained in larger datasets, we able to fine-tune more complex neural networks in the latter stages without overfitting due to the small sample size. We show that is possible to control the trade-off between conserving knowledge and fine-tuning using a simple probabilistic map. Experiments using real-world data show the efficacy of our approach as it outperforms other state-of-the-art methods for transfer learning and regularization.

Figure 1. Representation of a dual funnel problem. The left funnel shows the number of applicants decreasing, whereas the right funnel shows the amount of data (in terms of variables) increasing during the process. The top and down parts show the stages in the conversion and evaluation phase, respectively.
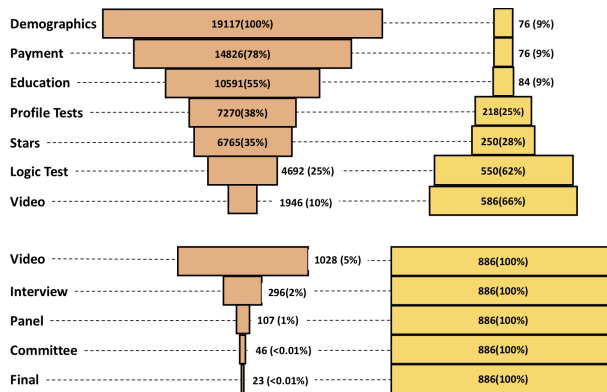
1 Introduction

In many practical applications including homeland security, medical diagnosis, and network intrusion detection, decision systems are composed of an ordered sequence of stages [39]. This can be referred to as a multi-stage process. To train classifiers to work in such processes, many aspects have to be observed, since they present different structures and the level of decision, data and sample size can significantly change in each stage.

One specific common example of such a process is a selection process. A selection process can be defined as the process of selection and shortlisting of the right candidates with the necessary qualifications and skill set to fill determined positions [25]. Although the majority of examples are related to job opportunities, selection processes are also used for fellowships, grants, and prizes.

The simplest type of selection process is comprised of one single stage where the evaluator has information about the applicant and makes a decision. However, the more elaborated the process is, the more stages are necessary to filter the applicants, as not everyone is given the same attention to save resources.

One of the hardest challenges for training classifiers for a multi-stage process can be defined as a dual funnel problem (see Figure 1

for details). During each stage of the conversion phase, the number of applicants decrease whereas the data about them increases, which means that the dataset grows in dimensionality.

The classifiers that are trained in initial stages have sufficiently large samples to generalize. However, there might not be useful information to differentiate between the best applicants, so the models have high bias and tend to underfit. In the final stages, there is plenty of information for each applicant, but the sample size is so low that classifiers trained only on this stage suffer from high variance and tend to overfit the data.

Here we address this problem using *Transfer Learning* (TL) [29, 41]. In TL, the goal is to use the knowledge obtained from solving a problem in one domain to solve a related problem in a different domain. We apply this methodology here to transfer knowledge learned in each stage to a subsequent stage to create better classifiers.

In our approach, called *Multi-StaGe Transfer Learning* (MSGTL), we propose a structure where simpler and more general *neural networks* (NN) are trained in the initial stages. Such NNs are more general because they were trained in a dataset with a large sample size but small dimensionality. In later stages, the weights from this NN will be transferred to more complex NNs trained in datasets with more features but a much smaller sample size. The general knowledge transferred from the early stages helps the more complex NN to prevent overfitting and achieve better performance than a classifier trained in the single-stage only. We show that our approach can overcome the common problems in transfer learning and classical

---

[1] New York University, USA, email: andre.mendes@nyu.edu
[2] New York University, USA, email: julian.togelius@nyu.edu
[3] Pontifical Catholic University of Parana and Federal University of Parana, Brazil, email: leandro.coelho@pucpr.br

challenges from real data such as covariate shift and class imbalance. The main contributions of this paper are:

1. We present a framework that allows knowledge to be transferred between NN structures in different stages in a multi-stage problem. Our approach can handle different types of unstructured (text) and structured data (tabular, categorical).
2. We introduce a hyperparameter to control the knowledge transferred during training in each stage. We show that our hyperparameter can be used to find an appropriate trade-off between keeping the knowledge to prevent overfitting and allowing fine-tuning to improve performance.
3. Our method improves on cascaded classifiers and other multi-stage structures by making full decisions in every stage with available data.
4. The efficacy of the proposed model is demonstrated by extensive longitudinal and cross-validated experiments on the real data from a selection process in 3 distinct years. We show that our model can outperform other state-of-the-art methods in transfer learning and regularization in the multi-stage case. Our method presents significant gains particularly in the late stages of the process with smaller sample sizes.

The remainder of this paper is organized as follows. Section 2 provides an overview of the related work in the field. Section 3 defines the problem and in Section 4, the proposed method is presented. Section 5 explains the experiments used to validate the method and the results are shown in Section 6. Finally, Section 7 presents the conclusion.

2 Related Work

The work presented here is most closely related to two subfields of machine learning, namely transfer learning [29, 41] and multi-stage classifiers [35] or decision cascades [33]. Additionally, our method is similar to some regularization techniques, such as dropout.

2.1 Transfer Learning

Transfer Learning (TL) focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. Humans do this process in many cases such as using knowledge from playing tennis to play table tennis.

Following the definitions presented in [41], TL can be defined in two categories. The first one is called homogeneous transfer learning and it refers to problems where the feature spaces in both domains are the same. The main focus is on aligning data distributions between domains through domain adaptation. Previous attempts to solve this problem have achieved important results [28, 46], especially in computer vision [13] and natural language processing [6]. The second case is called heterogeneous transfer learning and the main problem is to align feature spaces between domains to achieve representation transformation. There are mainly two approaches used in this case: map X_T to X_S, which is referred to as asymmetric transformation, [10] or map both X_S, X_T to a common latent space, symmetric transformation [9].

Other methods use *deep neural networks* (DNN) that can better learn abstract representations of the data on different levels and generality [3, 44]. With this ability, DNNs have been used to achieve state-of-the-art results in many transfer learning problems [14, 32]. Additionally, adversarial learning has been successfully explored to

minimize cross-domain discrepancy. For example, a simple and efficient gradient reversal layer is proposed in [12] to promote the emergence of features that are discriminative for the main learning task on the source and indiscriminate concerning the difference in the stages. Other methods have extended this approach for the multi-domain case [21, 22].

Many methods rely on weight transfer, where a NN is pretrained on a source task and the weights of some of its layers are transferred to a second NN that is used for another task. The non-transferred weights of the NN are randomly initialized and a second training phase follows [2, 43].

2.2 Multi-Stage classification

Multi-stage and cascade classifiers share many similarities. However, an important difference between cascade [40] and multi-stage can be defined as the system architecture. Detection cascades make partial decisions, delaying a positive decision until the final stage. In contrast, multi-stage classifiers can deal with multi-class problems and can or have to make classification decisions at any stage [39]. In our case, our method needs to make a full decision in each stage, since there will only be more features in the next one. Figure 2(a) shows an example of a multi-stage architecture.

Cascade classifiers are an important part of a subarea of machine learning that researches the trade-off between learning and feature acquisition or learning with budget constraints [38, 39]. It has also been used for content classification [1] and detecting instructions and anomalies in networks [7]. However, the vast applications recently have been to unstructured data in computer vision, in areas such as object segmentation [24], scene understanding [31], and recognition [16]. We focus on structured data in tabular form with categorical attributes and unstructured data in text format.

The approaches in [30, 34, 45] explore the connection between deep models and multi-stage classifiers such that classifiers can be jointly optimized and they can cooperate across the stages. This structure is similar to our approach. However, our algorithm only has access to the information in a specific stage, whereas in the other methods, the entire dataset is available and a decision is made about which parts to use.

To avoid losing information from one stage to another, cascades can use soft classifiers, which collects the classification scores extracted by each stage of classifiers and then combines the classification scores for the final decision [8]. Similarly, we use the classification scores from the classifiers in each stage to inform the next NN in our structure. However, these classifiers are not soft and don't have a reject mechanism, so the scores come from the actual decisions in each stage.

2.3 Probabilistic Regularization

Regularization is an important part of training NN to improve generalization capabilities. Dropout [36] is a technique that improves the generalization of NN by preventing the co-adaptation of feature detectors. The working of dropout is based on the generation of a mask by utilizing Bernoulli and Normal distributions.

Following the success of Dropout, other methods have been explored different aspects, such as sampling dropout mask from different distributions [11, 19] or actively selecting nodes based on their strength in the NN [17]. Similar to these methods, our approach also uses a binary mask to control updates in the weights. However, a key difference is that we preserve the knowledge acquired in previous
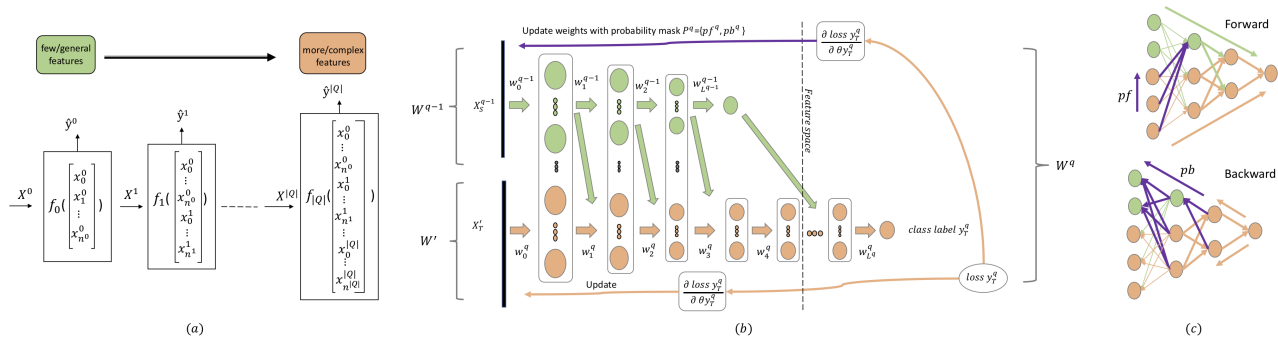
Figure 2. Representation of a multi-stage architecture (a) and the proposed approach for weight transfer (b-c). For every stage $q \in Q$, a set of features $X^q = X^{q-1}||X'$ is available (the symbol $||$ represents the column-wise concatenation of two matrices). In (a), a classifier f_q makes the decisions $\hat{y}^q$. Our proposed method is shown in (b), where the weights from the *neural network* (NN) structure in W^{q-1} (green) are transferred and together with the new weights in W' (orange), they form the new NN structure W^q (green+orange). The orange weights are initialized randomly and are updated during training. The green weights are initialized using W^{q-1} and their updates are controlled using binary masks defined by the probabilistic variable ρ. In (c), the top part shows the updates in the forward pass controlled by pf^q. The bottom part shows the backward pass controlled by pb^q. A purple weight will only be updated if its correspondent value in the binary mask is 1; otherwise it stays frozen.

states, by using the mask only in new nodes introduced in later stages. Additionally, our mask contains a forward and backward component that controls the active state of a node in both forward and backward passes through the NN.

3 Problem Statement

A domain $\mathcal{D}$ is defined by a feature space $\mathcal{X}$ and a marginal probability distribution $P(X)$. Each x_i is the i-th feature vector (instance), n is the number of feature vectors and m is the number of samples in $X \in \mathbb{R}^{m \times n}$. We need a predictive function $f(\cdot)$ from the feature vector and label pairs $\{x_i, y_i\}$ where $x_i \in X$ and $y_i \in \mathcal{Y}$. From the definitions above, a domain $\mathcal{D} = \{\mathcal{X}, P(X)\}$ and a task $\mathcal{T} = \{\mathcal{Y}, f(\cdot)\}$ are defined. Therefore, $\mathcal{D}_S$ and $\mathcal{T}_S$ are the source domain and task, respectively. Similarly, $\mathcal{D}_T$ and $\mathcal{T}_T$ are the target and task respectively. We can use TL to improve the target predictive function $f_T(\cdot)$ by using the related information from $\mathcal{D}_S$ and $\mathcal{T}_S$.

Let's define Q as the stage vector and a single stage $q \in Q$. In a multi-stage process, we can define previous stages as the source domain $\mathcal{D}_S$, and the following stage as the target domain $\mathcal{D}_T$.

For each stage in a dual funnel problem, new data is received while the number of samples decreases, which means $X^{q-1} \neq X^q$, $m^{q-1} > m^q, n^{q-1} < n^q$. As m gets significantly smaller in absolute value and in comparison to n, classifiers trained on the specific stage data tend to overfit. With new features in a new stage, we also have a shift in the feature space, $\mathcal{X}_S \neq \mathcal{X}_T$. This requires that a domain adaptation so that the features in the source in the target can be in the same feature space.

The process also contains domain class imbalance, defined as $P_S(Y) \neq P_T(Y)$. This happens because the distribution between positives and negatives samples is different across source and target. This is a challenge for many transfer learning algorithms [42].

With all together, our method tries to solve the problem of transferring knowledge from one stage to the other in order to address overfitting caused by $m^{q-1} > m^q, n^{q-1} < n^q$, the shift in the feature space $\mathcal{X}_S \neq \mathcal{X}_T$, and the domain class imbalance $P_S(Y) \neq P_T(Y)$. Our method does not directly address domain class imbalance, but we use different penalties for misclassifications on positive samples to address the problem of class imbalance in each stage.

4 Multi-Stage Transfer Learning (MSGTL)

Our proposed method uses transfers weights to share knowledge between the stages. Figure 2 shows the main idea for our approach. To address overfitting, we propose a simple weight transfer mechanism to use knowledge from previous stages to train the NN in later stages. The weights trained in the initial stages are exposed to larger and simpler data. By transferring them to the later stages, we add additional regularization for the more complex structures.

In the first step, our method learns a simpler NN that is trained in a dataset with more samples and less complexity. Later, the weights from this NN will be used as initialization for a more complex NN trained in a dataset with fewer samples but more features. During the training of the second NN, the weights from the previous one are updated according to the respective probabilistic map. The knowledge shared between NNs allows the second to perform better in the more complex dataset without overfitting due to the smaller sample size.

For the shift in feature space, we augment our NN structure with a gradient reversal layer proposed in [12]. This method uses adversarial learning where a domain discriminator is trained to distinguish the source from the target and the feature representation is trained simultaneously to confuse the domain discriminator, decreasing the gaps between the feature spaces. To improve the discriminator, we add a weighted loss function where the weights represent the difference in the number of samples from source and target.

In our method, the NN is trained sequentially incorporating information from the entire process. It is important to note that our approach is different than an ensemble, in which a group of classifiers is trained separately, with no information shared, and the decisions are typically made based on a voting system. In our approach, the weights from previous stages are combined with new ones, affecting the batch update and sharing information in all layers. Our method also differs from other weight transferring approaches. Traditional techniques transfer the entire layer of the NN, usually the early ones with general attributes and fine-tune the last layers with the new data. In our method, we combine transferred and new weights in all layers and control the fine-tuning with a probabilistic variable.

4.1 Architecture

The components of the proposed method are:

- We redefine $\mathcal{X}$ and $\mathcal{Y}$ as vectors containing all input features and labels, respectively for the entire process.
- $q \in Q$ represents a single stage in the process with $|Q|$ stages.
- $X^q \in \mathbb{R}^{m^q \times n^q}$ is the feature matrix input and $Y^q \in \mathbb{R}^{m^q \times 1}$ are the labels for current stage q.
- h_l^q is the number of nodes in a given layer $l = \{0, 1, ..., L^q\}$.
- L^q represents the number of layers in the NN. To simplify our structure, we start the NN with n^q nodes and each following layer has half of the nodes compared to the previous one. Therefore, we have $h_l^q = 2 * h_{l+1}^q$ for $l = \{0, ..L^q\}$. We define a max number of layers ω to bound the NN in case the initial input feature vector is too large (analysis of the hyperparameters in section 6.3). Finally, we also define a hyperparameter γ for the number of nodes in the last activation layer before the final classification. Therefore:

$$L^q = \begin{cases} 3, & if \ n^q \leq \gamma \\ \omega, & if \ \lceil \log_2(\frac{n^q}{\gamma}) \rceil + 2 \geq \omega \\ \lceil \log_2(\frac{n^q}{\gamma}) \rceil + 2, & otherwise \end{cases} \quad (1)$$

- $W^q = \{w_0^q, w_1^q, ...w_{L^q}^q\}$ is a vector containing the NN structure for stage q.
- $w_l^q \in \mathbb{R}^{h_l^q \times h_{l+1}^q}$ is a single weight matrix that connects two consecutive layers in the NN.
- $P^q = \{p_0^q, p_1^q, ...p_{L^q}^q\}$ is a vector containing all binary masks corresponding to parameters in W^q.
- $p_l^q = \{pf_l, pb_l\}$ is formed by the forward mask pf_l, and the backward mask pb_l. Each component is a binary matrix with independent random variables drawn from a Bernoulli distribution with probability ρ of being 1.
- $\mathcal{W} = \{W^0, W^1, ..., W^{|Q|}\}$ is the final vector containing all the NN structures for the entire process.

Considering a standard neural network structure, our modifications with ρ and P can be result in a loss function similar to the case in Dropout [36] and it can be solved with standard error backpropagation method. Additionally, we address the problem with class imbalance using the balanced cross-entropy loss function for optimization defined as

$$\mathcal{L} = -\sum_{i=1}^{m} \beta y_i \log \hat{y}_i + (1 - \beta)(1 - y_i)\log(1 - \hat{y}_i), \quad (2)$$

where y represents a ground truth label, $\hat{y}$ is a predicted label and β is the fraction of the sample which is dominant in a dataset, $\beta = 1 - \frac{\sum_{y \in Y} y}{|Y|}$ when $y \in \{0, 1\}$. This modification helps the model to focus on learning positive labels, since the learning for the dominant negative class is reduced.

4.2 Step-by-Step training

The algorithm for our method is presented in Alg. 1. Let's define the general case with $start = 0$. In $q = 0$ a neural network receives input features and labels from X^0 and Y^0. The first layer of the NN receives a input feature vector from $X^0 = \{x_0^0, ..., x_{n0}^0\}$ and all other parameters, $\{w_0^0, ..., w_L^0\}$ are randomly initialized. The masks in P^0 are defined with all values equal to 1. We train the initial NN, creating the NN structure $W^0 = \{w_0^0, ..., w_L^0\}$.

In the following stage q, features and labels come from X^q and Y^q, respectively. We first initialize a new NN structure W^q with random weights. Then in line 10, we loop through the parameters in W^{q-1} updating the respective weights in W^q so that the pretrained

values in W^{q-1} are passed to W^q. In the same process, we also create the masks $P^q = \{pf_l^q, pb_l^q\}_{l=0}^{L^q}$, where pf_l^q and pb_l^q are the binary masks defined by $Bernoulli(\rho)$.

During training, all the weights that were initialized randomly are updated. As for the pretrained weights, they will be updated according to the binary masks pf_l^q and pb_l^q during the forward and backward pass respectively. If $\rho = 0$, all the pretrained weights are kept frozen. We call this *fixed configuration*. In the case of $\rho = 1$, all weights are updated. We call this *initialization configuration* since the pretrained weights are only initialized with the previous structure but they are fine-tuned during training (more details for ρ in 6.3).

After all the stages are completed, we have a vector $\mathcal{W}$ that contains a neural network for each stage. For inference, each NN is retrieved and used to make the predictions for the respective stage.

Algorithm 1 Multi-Stage Transfer Learning (MSGTL)

Input: $\mathcal{X}, \mathcal{Y}, \omega, Q, pu, \lambda, start, epochs$
Output: $\mathcal{W}$
1: $\mathcal{W} \leftarrow \{\}; q \leftarrow start$
2: $X^q, Y^q \leftarrow \mathcal{X}[q], \mathcal{Y}[q]$
3: $W^q \leftarrow Initialize(W, \gamma, \omega)$
4: $P^q \leftarrow Ones(W^q)$
5: $W^q \leftarrow Train(X^q, Y^q, W^q, P^q, \lambda, epochs)$
6: Add W^q in $\mathcal{W}$
7: **for** $q = start + 1$ to $|Q|$ **do**
8:    $X^q, Y^q \leftarrow \mathcal{X}[q], \mathcal{Y}[q]$
9:    $W^q \leftarrow Initialize(W, \gamma, \omega)$
10:    $W^q, P^q \leftarrow TransferWeights(W^{q-1}, W^q, \rho)$
11:    $W^q \leftarrow Train(X^q, Y^q, W^q, P^q, \lambda, epochs)$
12:    Add W^q in $\mathcal{W}$
13: **end for**=0

---

5 Experiments

To validate our method we perform a series of experiments using a real-world dataset from a multi-stage selection process. This section presents the datasets, feature preparation, performance metrics, validation methods and the algorithms that we use for a benchmark.

5.1 Datasets

Our dataset comes from an organization that select students for a fellowship. This selection process contains two phases, conversion and evaluation as shown in Figure 1 and Table 1. In conversion, the applicants apply for the fellowship and fill up general information in a web portal. For each new stage, the applicants enter new and more detailed information. In this phase, there is no interference from humans evaluators and the applicants can stop the process, or be filtered out by automatic filters or conditions. All the applicants that finish the first phase move to the second phase called evaluation. In this phase, the applicants are selected based on the criteria for each stage. Fewer applicants continue in the stages until the final selection.

With regards to data, the video stage marks the last part where data about the applicants is collected in a structured, objective and automatic form. In the following stages that require interviews, even though the applicants present more information, only the evaluators have access to it. All decisions in all stages are binary, meaning if the applicant was accepted or rejected.

Table 1. Stages in the multi-stage selection process

Conversion Phase	
Demographics	Provide information about the country, state, city, and age. The system performs automatic validations to see if they match the criteria for the process.
Payment	Pay the application fee or ask for a waiver. For a waiver, the applicant needs to write a justification.
Education	Provide information on the university, major and extra activities.
Profile Tests	Perform online tests to map profile characteristics such as interests, values, mindset, and personality.
Star	Write about life experiences that were meaningful following a STAR structure (S: situation, T: task, A: action, R: result).
Logic Tests	Respond to online tests to map levels in problem-solving involving logic puzzles.
Video Submission	Submit a 2-min long video, explaining why they deserve the fellowship. There are no restrictions on the content of the video and the quality of the recording.
Evaluation Phase	
Video Evaluation	Applicants are evaluated based on their entire profile submitted. This is the first stage where students are accepted or rejected by evaluators.
Interview	Applicants have the first direct contact with the evaluators. In this stage, the goal is to have a screening interview where the evaluators can go deeper and clarify questions about the applicant's profile.
Panel	Former fellows are evaluators in a panel section interview. The panel has 3 to 4 fellows that interview 5 to 6 applicants at the same time.
Committee	The selection team and some of the senior fellows discuss the results and select applicants to move to the final step. Applicants do not participate in this step.
Final	Applicants are interviewed by the direct board of the foundation. The interview structure is similar to the panel structure and the applicants are evaluated together. This is the final stage where the applicants will be selected for the fellowship.

5.2 Feature Preparation

In the first stages, all the data obtained from the applicants is structured in a tabular format. For categorical data, such as the state where they live, a standard one hot encode transformation is used to convert the data to a numerical form. For more complex datasets such as education, there is a feature engineering step to reduce the dimension size of the dataset. This includes, for example, creating groups for universities and excluding strong correlated features. For the datasets from profile or logic tests, all the information is already in continuous numerical form and we perform a standard normalization.

In the later stages of the process, such as stars and video, the data is unstructured. The stars are open answers in form of text. For the video, we only consider the content of the speech component. The speech is extracted and used as a text. To convert this unstructured data to numerical values, we create word embeddings using Word2Vec [26]. The goal is to assign high-dimensional vectors (embeddings) to words in a text corpus in a way that preserves their syntactic and semantic relationships. To train the embeddings, we use a pretrained model trained on the entire Wikipedia in Portuguese from Brazil and extracted a 300-dimensional vector representation for each word. We tested different methods to aggregate the words and the best method was standard average in the word vectors.

5.3 Validation and Metrics

For the experiments, we first perform cross-validation using each dataset for each year separately. For example, the dataset from applicants in 2017 is split into training, test, and validation. In the second batch, we combine pairs of years, such as 2017 and 2018, and perform the random cross-validation split in the aggregated data. Finally, we perform the same experiment using data from all years combined. The combinations result in 7 experiments using 10-fold cross-validation in each one of them for a total of 70 runs.

In the longitudinal experiments, we use a previous year as a training and test set and the following year as a validation set. For example, we split the dataset from 2017 and train and test, find the best model and validate its results using the dataset from 2018. The model, in this case, is trained in 2017 and has never seen any data in 2018. We repeat the process for 2017 and 2019, and 2018 and 2019. Finally, we also combine the datasets between 2017 and 2018 and validate the results in 2019. Therefore, we have 4 different experiments with 10-fold cross-validation resulting in a total of 40 runs.

Since our dataset has a funnel structure, it is common in this scenario to have an imbalanced dataset. Moreover, in a selection process, there is also a significant difference between misclassifying a weaker applicant as strong (false positive) then a misclassifying a stronger applicant as weak (false negative). Therefore, we choose to compare the models in terms of f1-score, which balances precision and recall, for the positive class.

5.4 Benchmark Methods

We compare the results with other established methods, following the choices for optimization and regularization defined in the experiments for each paper and for the selection of the hyperparameters, we conduct *importance-weighted cross-validation* (IWCV) [37].

Due to the size of the dataset, especially in the last stages, we expect that commonly used non-deep algorithms such as *Logistic Regression* (LR) [15] and *Support Vector Machines* (SVM) [4] would achieve better performance because tend to generalize better on smaller datasets. The trade-off hyperparameter C for both SVM and LR is $C = 0.1$. We also compare our method with ensemble approaches such as *Random Forest* (RF) [20] and *XBoost* (XB) [5] using trees with a maximum depth equals 8, shrinkage equals 0.1 and no column sub-sampling.

For domain adaptation methods, we use *Domain Adversarial Neural Network* (DANN) [12], *Transferable Adversarial Training (TAT)* [21], and *Transfer Kernel Learning (TKL)* [23] with an SVM as the base classifier. For DANN and TAT, we follow the hyperparameters used in [21] adopting Adam [18] with initial learning rate $\eta_0 = 10^4$ and a inverse-decay strategy [12] where the learning rate changes by $\eta_p = \frac{\eta_0}{(1+\omega_p)^\phi}$, $\omega = 10$, $\phi = 0.75$, and p is the progress ranging from 0 to 1. For TKL we use $\sigma = 0.1$, $\lambda = 10$, $p = 10$ and $\gamma = 10$. For each domain adaptation method, we perform training using two stages: the previous stage as the source domain and the current stage as the target domain.

For standard NN, we use regularization methods to counteract the tendency to overfit on small data. Specifically, we include the *Virtual Adversarial Training (NN-VAT)* [27] with hyperparameters $K = 1, \epsilon = 2$ and $\alpha = 1$ and *Dropout* (NN-DO) [36] with $p = 0.5$.
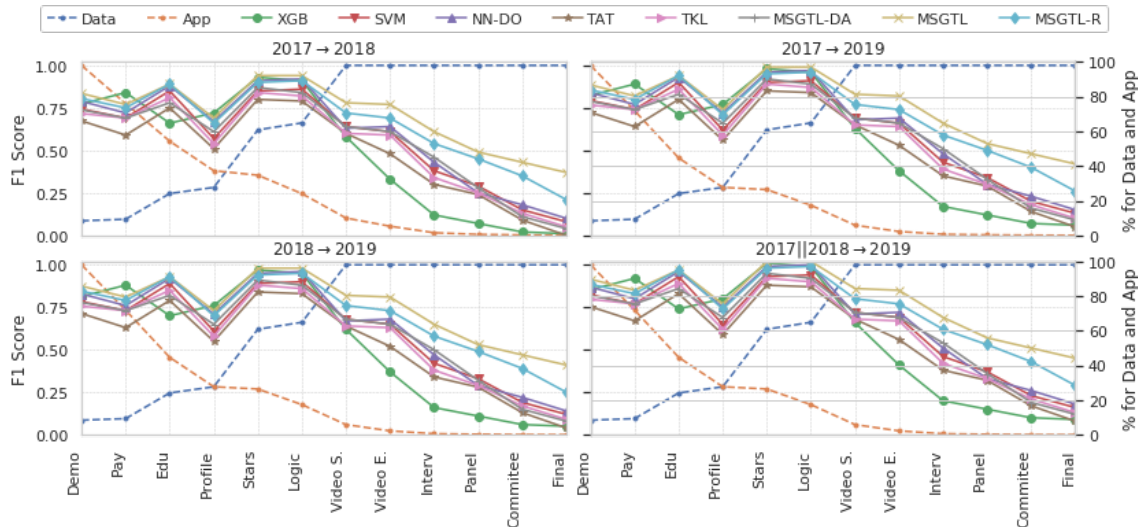
Figure 3. Results obtained in the longitudinal setup. For each new stage, the number of applicants drops while more data about them is obtained. Algorithms perform better while the number of samples is relatively high but the performance worsens drastically in the later stages. The percentage of applicants and data in each stage is shown on the y-axis. For algorithms, the y-axis represents the performance in terms of F-score for positive class.

As for the method proposed in this work, we include tree variations: MSGTL is the base configuration with transfer weights from one stage to the other following different values of ρ. MSGTL-DA is the configuration where we add a domain adaptation component using the reverse gradient layer following the structure proposed in TAT [21]. The final variation is MSGTL-R, which combines the method proposed here with an extra regularization term from DO. We use TAT and DO because they performed better than other methods such as DANN and VAT, respectively. The best results in our experiments were achieving using $\omega = 6, \gamma = 2$ and $\rho = 0.3$.

6 Results

In this section, we present the results obtained in the experiments described in section 5. We also present an analysis of different values of the hyperparameters and how they affect the final performance.

6.1 Longitudinal Validation

Table 2 shows the average results obtained for each stage in all experiments. For each one of the groups, we show the results for the best algorithm: standard (SVM), ensemble (XB), neural networks with dropout (NN-DO) and domain adaption (TAT, TKL). From these results we can make the following observations:

We see that (1) during the conversion phase, as the number of applicants is higher, all the algorithms can achieve good results. Ensemble algorithms such as XB can perform well especially due to large sample size and also the low dimensionality of the dataset. Even more complex algorithms such as NN-DO can prevent overfitting. However, during the Evaluation phase, there is a drastic drop in the number of applicants, reducing the sample size. Also, the dimensionality of the data increases significantly as the unstructured data from text and video is added to the dataset. Therefore, (2) in the evaluation stages, the performance of the standard algorithms (SVM, XB, TKL) drop, especially due to the complexities added to the dataset. The methods based on neural networks (NN-DO, TAT) that should

capture these new complexities don't have enough samples to generalize well and overfit the smaller datasets. Even NN structures with strong regularization (NN-DO) are not able to achieve good performance, because all the weights are trained in a single stage.

We also notice that (3) algorithms based on domain adaption approaches (TKL, TAT) do not perform well, as the knowledge passed from the previous stage is not enough to reduce the overfitting problem. TKL has a better performance than TAT due to it is a simpler structure. The results from MSGTL-DA are also lower than the other versions, which shows that the domain adaptation layer introduced increases the complexity of the final NN and hurts the performance in the evaluation phase. This suggests that the discrepancy between the feature spaces cannot be reduced by the proposed approach.

Finally, (4) our models MSGTL and MSGTL-R can outperform all other methods due to the aggressive regularization introduced by the weights from previous states. MSGTL performs slightly better than MSGTL-R, which suggests that the combination of weight transferring with the aggressive dropout can cause the model to underfit in some cases. Comparing all DA methods, we can see that the transfer weight introduced in MSGTL-DA can make the model achieve better results than TKL and TAT. Finally, MSGTL is able to improve the average results in the evaluation phase by 0.17 points (0.33→0.54, 63%) comparing to the second best method.

In Figure 3, we plot the results for each method in the four cases in the longitudinal experiment. We can see the general pattern where the algorithms perform well during the conversion phase but are not able to keep up performance in the evaluation phase. By using weights transferred from other stages, MSGTL methods can reduce the drop in performance in achieving better results in later stages.

6.2 Cross-validation

Table 2 also shows the results for all experiments using the cross-validation setup in parenthesis in each cell. As expected, all algorithms perform better in this scenario because the data comes from the same year, reducing the chance of covariate shift between the

Table 2. Average results for all methods in each stage using F1-Score for the positive class. Numbers in the left in each cell show results for the longitudinal setup while numbers in parenthesis represent results obtained in the cross-validation setup.

Steps	XGB	SVM	NN-DO	TAT	TKL	MSGTL-DA	MSGTL	MSGTL-R	**MEAN**
					Conversion Phase				
Demographics	**0.80 (0.81)**	0.77 (0.79)	**0.8 (0.82)**	0.70 (0.71)	0.73 (0.73)	0.75 (0.77)	**0.86 (0.89)**	**0.83 (0.85)**	0.78 (0.80)
Payment	**0.85 (0.85)**	0.7 (0.75)	0.73 (0.77)	0.6 (0.63)	0.7 (0.72)	0.7 (0.75)	**0.78 (0.83)**	0.76 (0.79)	0.73 (0.76)
Education	0.67 (0.7)	0.86 (0.87)	**0.89 (0.89)**	0.76 (0.79)	0.82 (0.85)	0.79 (0.79)	**0.9 (0.92)**	**0.9 (0.94)**	0.82 (0.85)
Profile	**0.73 (0.75)**	0.58 (0.61)	0.68 (0.73)	0.52 (0.54)	0.55 (0.55)	0.62 (0.67)	**0.7 (0.75)**	0.67 (0.69)	0.63 (0.66)
Star	**0.94 (0.97)**	0.86 (0.9)	**0.92 (0.94)**	0.81 (0.83)	0.85 (0.86)	0.88 (0.89)	**0.95 (0.97)**	**0.91 (0.92)**	0.89 (0.91)
Logic	**0.92 (0.93)**	0.87 (0.91)	**0.93 (0.96)**	0.8 (0.81)	0.83 (0.88)	0.85 (0.87)	**0.95 (0.97)**	**0.92 (0.96)**	0.88 (0.91)
Video	0.59 (0.63)	0.65 (0.67)	0.64 (0.64)	0.61 (0.66)	0.61 (0.63)	0.65 (0.67)	**0.79 (0.8)**	**0.73 (0.74)**	0.66 (0.68)
MEAN	**0.78 (0.81)**	0.75 (0.79)	**0.8 (0.82)**	0.68 (0.71)	0.73 (0.73)	0.75 (0.77)	**0.85 (0.88)**	**0.82 (0.84)**	
					Evaluation Phase				
Video	0.34 (0.42)	0.62 (0.69)	0.65 (0.72)	0.49 (0.54)	0.6 (0.67)	0.62 (0.7)	**0.78 (0.81)**	**0.7 (0.72)**	0.6 (0.66)
Interview	0.13 (0.31)	0.39 (0.48)	0.44 (0.49)	0.31 (0.38)	0.35 (0.41)	0.47 (0.49)	**0.62 (0.64)**	**0.55 (0.55)**	0.41 (0.47)
Panel	0.08 (0.15)	0.3 (0.31)	0.26 (0.31)	0.25 (0.3)	0.26 (0.33)	0.29 (0.3)	**0.44 (0.46)**	**0.41 (0.43)**	0.29 (0.32)
Committee	0.03 (0.18)	0.16 (0.24)	0.19 (0.19)	0.1 (0.18)	0.14 (0.15)	0.12 (0.16)	**0.44 (0.47)**	**0.36 (0.39)**	0.19 (0.24)
Final	0.02 (0.2)	0.09 (0.18)	0.11 (0.18)	0.01 (0.07)	0.06 (0.14)	0.05 (0.12)	**0.38 (0.43)**	**0.22 (0.23)**	0.1 (0.17)
MEAN	0.12 (0.25)	0.31 (0.38)	0.33 (0.38)	0.23 (0.3)	0.28 (0.34)	0.31 (0.35)	**0.54 (0.57)**	**0.45 (0.46)**	
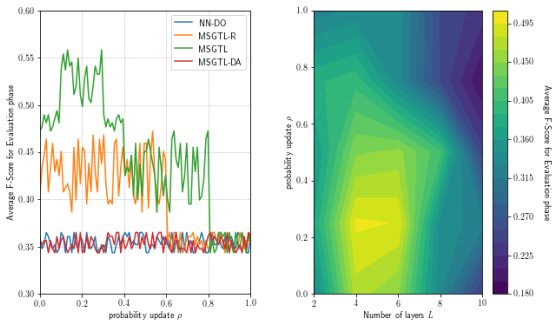
Figure 4. Results for hyperparameter sensitivity analysis. Left shows the performance with different values for ρ. Right shows results from changes in ρ and the number of layers L, defined by ω and γ.

distribution in training and test and validation sets.

Comparing the methods, the results are similar to the longitudinal case, and the proposed method MSGTL is able to achieve the highest score in both conversion and evaluation phases. However, the gain in the cross-validation setup is 0.16 (0.38→0.54, 42%) higher than the second place, which is lower than what was achieved in the longitudinal setup, 0.17. This suggests that our method can achieve better gains in case of a covariate shift comparing to other methods.

6.3 Hyperparameter Sensitivity

Nodes in the final layer γ and max number of layers ω: These two hyperparameters together control the shape of a neural network. Our proposed NN used a simple approach where each following layer has half of the nodes of the previous one until the last layer with γ number of nodes. Since the first layer has n nodes from the input features in X, if n is significantly large, we have a structure with many layers. We suspect that in such a structure, the effect of the transfer weights from previous stages is decreased while it also becomes more complex, making it more susceptible to overfitting. This assumption is verified in Figure 4, where we can see a decrease in performance as the number of layers increase.

Probability of update ρ: This hyperparameter controls the probabilities of updates from previous NNs. In the *initialization configu-*

ration, $\rho = 1$ and P contains masks with all values equal to 1. The weights from previous NNs are only used as initialization. The results in Figure 4 show that the NN converges to similar weights as if they were initialized randomly, performing similarly to a standard neural network. That shows that just initialization does not guarantee knowledge transfer from previous stages. For $\rho = 0$, P contains masks with all values equal to 0. In this *fixed stage*, the knowledge transfer is guaranteed and the results are improved. However, there is a possibility of partial underfitting since many weights updates are restricted. Hence, setting ρ between the two extremes achieved the best results. From many experiments, the best value is at $\rho = 0.3$. In this case, knowledge is still transferred from the previous stage, but the NN can better fine-tune to the new data without overfitting.

7 Conclusion

In this paper, we presented a *Multi-Stage Transfer Learning (MS-GTL)* approach to enhance the transferability of knowledge from different neural network (NN) structures. Each NN is trained in sequential stages in a multi-stage process. The method consists of transfer knowledge from a stage with many samples and low-dimensionality to a stage with high-dimensionality but much fewer samples. We confirm that previous weights are more general and they help more complex NNs to learn specific weights in the later stages of process. We perform an empirical evaluation using a dataset from 3 years with a dual funnel structure. We compare the results with other state-of-the-art algorithms for domain adaptation and regularization and the results show that our approach was able to achieve significant improvements, especially in later stages. We also show that allowing some level of updates in the transferred weights using a probabilistic hyperparameter improves the performance in our method. Future work includes investigating the approach in other multi-stage situations in different domains and processes with a small sample size.

Selection processes are a sensitive topic. It is possible that training on existing outcomes of such processes is likely to reproduce the biases exhibited by those who did the selection in the first place. We have not investigated deep enough how our method would either increase or reduce such bias. In future work, we also plan to use explainability methods to understand how the knowledge transfer mechanism affects different classes and groups.

REFERENCES

[1] César Alfaro, Javier Cano-Montero, Javier Gómez, Javier M Moguerza, and Felipe Ortega, 'A multi-stage method for content classification and opinion mining on weblog comments', *Annals of Operations Research*, **236**(1), 197–213, (2016).

[2] H. Azizpour, A.S. Razavian, J. Sullivan, A. Maki, and S. Carlsson, 'From generic to specific deep representations for visual recognition,', in *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 36–45.

[3] Yoshua Bengio, Aaron Courville, and Pascal Vincent, 'Representation learning: A review and new perspectives', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **35**(8), 1798–1828, (2013).

[4] Chih-Chung Chang and Chih-Jen Lin, 'Libsvm: A library for support vector machines', *ACM Transactions on Intelligent Systems and Technology (TIST)*, **2**(3), 27, (2011).

[5] Tianqi Chen and Carlos Guestrin, 'Xgboost: A scalable tree boosting system', in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794. ACM, (2016).

[6] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa, 'Natural language processing (almost) from scratch', *Journal of Machine Learning Research*, **12**(Aug), 2493–2537, (2011).

[7] Luigi Pietro Cordella and Carlo Sansone, 'A multi-stage classification system for detecting intrusions in computer networks', *Pattern analysis and applications*, **10**(2), 83–100, (2007).

[8] Piotr Dollár, Ron Appel, and Wolf Kienzle, 'Crosstalk cascades for frame-rate pedestrian detection', in *European Conference on Computer Vision*, pp. 645–659. Springer, (2012).

[9] Lixin Duan, Dong Xu, and Ivor Tsang, 'Learning with augmented features for heterogeneous domain adaptation', *arXiv preprint arXiv:1206.4660*, (2012).

[10] Kyle D Feuz and Diane J Cook, 'Transfer learning across feature-rich heterogeneous feature spaces via feature-space remapping (fsr)', *ACM Transactions on Intelligent Systems and Technology (TIST)*, **6**(1), 3, (2015).

[11] Yarin Gal and Zoubin Ghahramani, 'Dropout as a bayesian approximation: Representing model uncertainty in deep learning', in *international conference on machine learning*, pp. 1050–1059, (2016).

[12] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky, 'Domain-adversarial training of neural networks', *The Journal of Machine Learning Research*, **17**(1), 2096–2030, (2016).

[13] Boqing Gong, Yuan Shi, Fei Sha, and Kristen Grauman, 'Geodesic flow kernel for unsupervised domain adaptation', in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2066–2073. IEEE, (2012).

[14] Yanming Guo, Yu Liu, Ard Oerlemans, Songyang Lao, Song Wu, and Michael S Lew, 'Deep learning for visual understanding: A review', *Neurocomputing*, **187**, 27–48, (2016).

[15] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant, *Applied logistic regression*, volume 398, John Wiley & Sons, 2013.

[16] Kevin Jarrett, Koray Kavukcuoglu, Yann LeCun, et al., 'What is the best multi-stage architecture for object recognition?', in *2009 IEEE 12th International Conference on Computer Vision*, pp. 2146–2153. IEEE, (2009).

[17] Rohit Keshari, Richa Singh, and Mayank Vatsa, 'Guided dropout', in *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 4065–4072, (2019).

[18] Diederik P Kingma and Jimmy Ba, 'Adam: A method for stochastic optimization', *arXiv preprint arXiv:1412.6980*, (2014).

[19] Durk P Kingma, Tim Salimans, and Max Welling, 'Variational dropout and the local reparameterization trick', in *Advances in Neural Information Processing Systems*, pp. 2575–2583, (2015).

[20] Andy Liaw, Matthew Wiener, et al., 'Classification and regression by randomforest', *R news*, **2**(3), 18–22, (2002).

[21] Hong Liu, Mingsheng Long, Jianmin Wang, and Michael Jordan, 'Transferable adversarial training: A general approach to adapting deep classifiers', in *International Conference on Machine Learning*, pp. 4013–4022, (2019).

[22] Mingsheng Long, Zhangjie Cao, Jianmin Wang, and Michael I Jordan, 'Conditional adversarial domain adaptation', in *Advances in Neural Information Processing Systems*, pp. 1645–1655, (2018).

[23] Mingsheng Long, Jianmin Wang, Jiaguang Sun, and S Yu Philip, 'Domain invariant transfer kernel learning', *IEEE Transactions on Knowledge and Data Engineering*, **27**(6), 1519–1532, (2014).

[24] Ping Luo, Xiaogang Wang, and Xiaoou Tang, 'Hierarchical face parsing via deep learning', in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2480–2487. IEEE, (2012).

[25] Peter V Marsden, 'The hiring process: recruitment methods', *American Behavioral Scientist*, **37**(7), 979–991, (1994).

[26] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean, 'Efficient estimation of word representations in vector space', *arXiv preprint arXiv:1301.3781*, (2013).

[27] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii, 'Virtual adversarial training: a regularization method for supervised and semi-supervised learning', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **41**(8), 1979–1993, (2018).

[28] Sinno Jialin Pan, Ivor W Tsang, James T Kwok, and Qiang Yang, 'Domain adaptation via transfer component analysis', *IEEE Transactions on Neural Networks*, **22**(2), 199–210, (2010).

[29] S.J. Pan and Q. Yang, 'A survey on transfer learning,', *Knowledge and Data Engineering, IEEE Transactions on*, **22**(10), 1345–1359.

[30] Tangquan Qi, Yong Xu, and Haibin Ling, 'Tourism scene classification based on multi-stage transfer learning model', *Neural Computing and Applications*, **31**(8), 4341–4352, (2019).

[31] M Ranzato, Joshua Susskind, Volodymyr Mnih, and Geoffrey Hinton, 'On deep generative models with applications to recognition', (2011).

[32] Artem Rozantsev, Mathieu Salzmann, and Pascal Fua, 'Beyond sharing weights for deep domain adaptation', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **41**(4), 801–814, (2018).

[33] Mohammad J Saberian and Nuno Vasconcelos, 'Boosting classifier cascades', in *Proceedings of the 23rd International Conference on Neural Information Processing Systems-Volume 2*, pp. 2047–2055. Curran Associates Inc., (2010).

[34] Mohammad Sabokrou, Mohsen Fayyaz, Mahmood Fathy, and Reinhard Klette, 'Deep-cascade: Cascading 3d deep neural networks for fast anomaly detection and localization in crowded scenes', *IEEE Transactions on Image Processing*, **26**(4), 1992–2004, (2017).

[35] Ted E Senator, 'Multi-stage classification', in *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pp. 8–pp. IEEE, (2005).

[36] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, 'Dropout: a simple way to prevent neural networks from overfitting', *The Journal of Machine Learning Research*, **15**(1), 1929–1958, (2014).

[37] Masashi Sugiyama, Matthias Krauledat, and Klaus-Robert MÃžller, 'Covariate shift adaptation by importance weighted cross validation', *Journal of Machine Learning Research*, **8**(May), 985–1005, (2007).

[38] Kirill Trapeznikov and Venkatesh Saligrama, 'Supervised sequential classification under budget constraints', in *Artificial Intelligence and Statistics*, pp. 581–589, (2013).

[39] Kirill Trapeznikov, Venkatesh Saligrama, and David Castañón, 'Multi-stage classifier design', in *Asian Conference on Machine Learning*, pp. 459–474, (2012).

[40] Paul Viola, Michael Jones, et al., 'Robust real-time object detection', *International Journal of Computer Vision*, **4**(34-47), 4, (2001).

[41] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang, 'A survey of transfer learning', *Journal of Big data*, **3**(1), 9, (2016).

[42] Karl R Weiss and Taghi M Khoshgoftaar, 'Investigating transfer learners for robustness to domain class imbalance', in *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 207–213. IEEE, (2016).

[43] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, 'How transferable are features in deep neural networks?', in *Advances in Neural Information Processing Systems 27 (Proceedings of NIPS)*, volume 27, p. 1–9.

[44] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson, 'How transferable are features in deep neural networks?', in *Advances in Neural Information Processing Systems*, pp. 3320–3328, (2014).

[45] Xingyu Zeng, Wanli Ouyang, and Xiaogang Wang, 'Multi-stage contextual deep learning for pedestrian detection', in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 121–128, (2013).

[46] Kun Zhang, Bernhard Schölkopf, Krikamol Muandet, and Zhikun Wang, 'Domain adaptation under target and conditional shift', in *International Conference on Machine Learning*, pp. 819–827, (2013).