Paper Review : "Risk Bounds for Transferring Representations With and Without Fine-Tuning"

Soundouss Messoudi SOUNDOUSS.MESSOUDI@DAUPHINE.EU
Paris Dauphine University

Abstract

The following report describes the work done for the ISI-10 Major project which objective is to review a research paper and reproduce its experiments. The paper chosen is a paper published in the International Conference on Machine Learning (ICML) 2017 called "Risk Bounds for Transferring Representations With and Without Fine-Tuning" by D. McNamara and M.F. Balcan. This paper proposes an approach to guarantee the usefulness of transferring a representation learned on a source task to a target one.

1. Introduction

Transferring representations is a widely used technique in Machine Learning. It consists of transferring a representation learned from a source task to a target task which has approximately the same intermediate representation. It is mainly used when the source task's data is abundant and the target task's data is scarce. For example, the representation learned from an image of a human face to predict age may also be useful for predicting gender. There are two approaches for transferring a representation. The first one is to consider the representation trained on the source task as fixed and use it directly for the target task. This approach is useful when there is an over-fitting problem for the target task, especially if its available data is few. The second one is called fine-tuning and consists of reducing the class of representations considered on the target task. This method is best used for a greater hypothesis class expressiveness.

The paper "Risk Bounds for Transferring Representations With and Without Fine-Tuning" by D. McNamara and M.F. Balcan was published in the International Conference on Machine Learning (ICML) in 2017. Its objective is to find a way to guarantee the usefulness of transferring a representation learned on a source task which is already known to other tasks without having to test it on these target tasks.

In this report, we will begin by presenting the chosen paper, its problem and main ideas. Then, we will present the experiments conducted to reproduce the research results and the difficulties encountered. We will also give a critical opinion of the paper based on the comparison between our results and those of the authors. We will finish by testing the practical effectiveness of transferring representation in a real world application using Datasets from Kaggle.

2. Paper presentation

The paper "Risk Bounds for Transferring Representations With and Without Fine-Tuning" by D. McNamara and M.F. Balcan published in ICML 2017 tackles a widely and successfully used technique in Deep Learning which is the transfer of representations. Its main objective is to find risk bounds for transferring representations from a source task to a target one with and without fine-tuning. Finding these risk bounds can guarantee the usefulness of transferring a representation from a source task which is already known to other tasks without having to actually test its performance on each of these target tasks.

The problem is defined as follows : Let X, Y and Z be sets known as the input, output and feature spaces respectively with Y = {-1, 1}. Let F be a class of representations where $f : X \to Z$ for $f \in F$. Let G be a class of specialized classifiers where $g : Z \to Y$ for $g \in G$. Let the hypothesis class $H := \{h : \exists f \in F, g \in G,$ such that $h = g \circ f\}$. Let $h_S, h_T : X \to$ Y be the labeling functions and P_S, P_T be the input distributions for source task S and target task T respectively. Let the risk of a hypothesis h on S and T be $R_S(h)$ and $R_T(h)$ respectively. Let $\hat{R}_S(h)$ and $\hat{R}_T(h)$ be the corresponding empirical risks. We have m_S labelled points for S and m_T labelled points for T. Let d_H be the VC dimension of H.

Before tackling this problem, the paper introduced the related work which was focused on domain adaptation and lifelong learning.

The work on domain adaptation has considered learning a hypothesis h on S and re-using it on T, bounding $R_T(h)$ using $R_S(h)$ which is measured with labeled source data and similarity between P_S and P_T which is measured with additional unlabeled target data. The results motivate a joint optimization using labeled source and unlabeled target data. This approach assumes the tasks become the same if their input distributions can be aligned. This seems to be a realistic setting given the broad use of available representations on the web, where f and g_T are learned separately and there is no joint optimization over source and target data.

The work on lifelong learning connects the past performance of a representation over many tasks to its expected future performance. In the previous work, from a representation $f \in F$ is constructed $G \circ f := \{g \circ f : g \in G\}$. These studies have provided bounds on the difference between the average empirical risk and the expected risk of the best hypothesis in $G \in f$ for a new task drawn from the distribution over tasks. All these bounds depend on known past performance on many tasks. Nonetheless, representations such as neural network weights or word embeddings are often learned in practice using only one source task, which is the setting considered in the paper.

The risk bounds in the paper were studied depending on the transfer of representations' approach :

- Representation fixed by source task : the trained representation on the source task is directly used on the target task without modifications.
- Representation fine-tuned on target task : the trained representation on the source task is slightly modified when transferred to the target task.

2.1. Representation Fixed by Source Task

We suppose that labeled source data is abundant, labeled target data is scarce, and we believe the tasks share a representation $\hat{f} \in F$ that we may extract by learning $\hat{g}_S \circ \hat{f} \in H$ on the source task S. After getting $\hat{f}$, we conduct empirical risk minimization over $G \circ \hat{f}$ on the target task T to get $\hat{g}_T \circ \hat{f}$.

In this case, we can find an upper-bound for $R_T(\hat{g}_T \circ \hat{f})$ by using the theorem above which states that we can improve the VC dimension-based bound for learning T from scratch by avoiding the generalization error of a hypothesis in H learned from m_T samples if the empirical risk $\hat{R}_S(\hat{g}_S \circ \hat{f})$ is a small constant, $m_S \gg m_T, d_H \gg d_G$ and $\omega(R) = O(R)$ with ω a function measuring a transferability property obtained from the problem setting.

The theorem is as follows :

Theorem 1. Let $\omega : R \to R$ be a non-decreasing function.

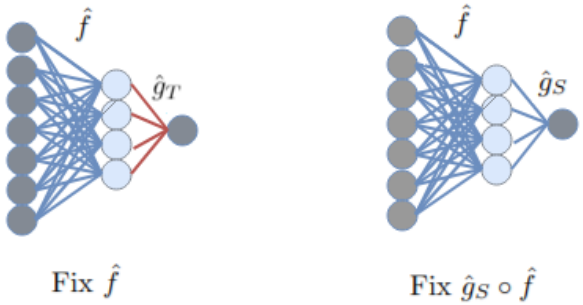
Suppose that $P_S, P_T, h_S, h_T, \hat{f}, G$ have the property that :

$$\forall \hat{g}_S \in G, \min_{g \in G} R_T(g \circ \hat{f}) \leq \omega(R_S(\hat{g}_S \circ \hat{f})).$$

Let $\hat{g}_T := argmin_{g \in G} \hat{R}_T(g \circ \hat{f})$. Then with probability at least $1 - \delta$ over pairs of training sets for tasks S and T, $R_T(\hat{g}_T \circ \hat{f}) \leq \omega(\hat{R}_S(\hat{g}_S \circ \hat{f}) + 2\sqrt{\frac{2d_H log(2em_S/d_H)+2log(8/\delta)}{m_S}}) + 4\sqrt{\frac{2d_G log(2em_T/d_G)+2log(8/\delta)}{m_T}}$.

An example of the representation fixed by Source task is a neural network with a single hidden layer where $\hat{f}$ corresponds to the lower-level weights learned on S and G corresponds to the upper-level weights that will be learned on T. Another possibility is to fix both upper-level and lover-level weights learned on S and use the neural network to predict T. Both techniques are illustrated in the figure 1. The paper shows that for this problem setting, $\hat{f}$ is also useful for T.

Figure 1. Neural network example with weights transferred and fixed from S.



2.2. Representation Fine-Tuned on Target Task

In this case, we consider learning $\hat{g}_S \circ \hat{f}$ on S, and then using $\hat{f}$ and $R_S(\hat{g}_S \circ \hat{f})$ so as to find $\hat{f} \subseteq F$. Let $\tilde{h}_{g \circ f}$ be a distribution over H associated with $g \circ f$. We

propose learning T with the hypothesis class $\tilde{H}_{G\circ\hat{f}}$ and the prior $\tilde{h}_{\hat{g}_S\circ\hat{f}}$. Learning T from scratch we assume that we would instead use $\tilde{H}_{G\circ F}$ and some fixed prior $\tilde{h}_0 \in \tilde{H}_{G\circ F}$. Let $R_T(\tilde{h})$ and compute $\hat{R}_T(\tilde{h})$ on the training set distribution of T.

In the following theorem, we show that if $\hat{f}$ is 'small enough', we can apply a PAC-Bayes bound to the generalization error of hypotheses in $\tilde{H}_{G\circ\hat{f}}$ by using the empirical risk $\hat{R}_S(\hat{g}_S \circ \hat{f})$, the generalization error of a hypothesis in H learned from m_S points, and a weak dependence on m_T and a function ω measuring a transferability property. The theorem shows that if the empirical risk $\hat{R}_S(\hat{g}_S \circ \hat{f})$ is a small constant, $m_S \gg m_T$ and $\omega(R) = O(R)$, we improve on the PAC-Bayes bound for $\tilde{H}_{G\circ F}$ and $\tilde{h}_0$.

The theorem is as follows :

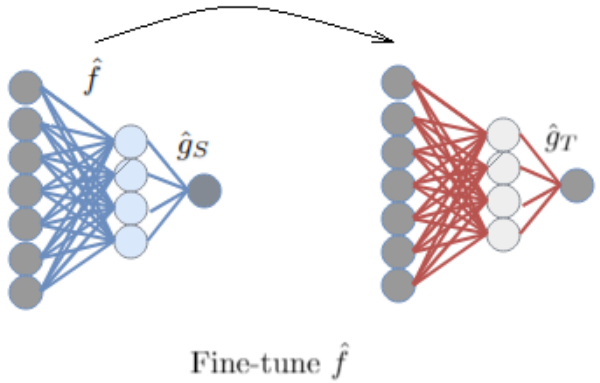
Theorem 2. Let $\omega : R \to R$ be a non-decreasing function.

Given $\hat{f} \in F$ and $\hat{R}_S(\hat{g}_S \circ \hat{f})$ estimated from S. If we can construct $\tilde{F}$ with the property $\forall \tilde{h} \in \tilde{H}_{G\circ\tilde{F}}, KL(\tilde{h}||\tilde{h}_{\hat{g}_S\circ\hat{f}}) \leq \omega(R_S(\hat{g}_S \circ \hat{f}))$. Then with probability at least $1-\delta$ we have :

$$\forall \tilde{h} \in \tilde{H}_{G\circ\tilde{F}}, R_T(\tilde{h}) \leq \hat{R}_T(\tilde{h}) + \sqrt{\frac{\omega(\hat{R}_S(\hat{g}_S\circ\hat{f}) + \sqrt{\frac{2d_H log(2em_S/d_H)+2log(8/\delta)}{m_S}}) + 2log(2m_T/\delta)}{2(m_T-1)}}$$

An example of the representation Fine-Tuned on Target task is a neural network with one hidden layer where the lower-level weight vectors $\hat{w}_i$ are learned on S. Then, when learning the task T, we only consider the lower-level weights near $\hat{w}_i$ corresponding to $\hat{f}$ as in the figure 2.

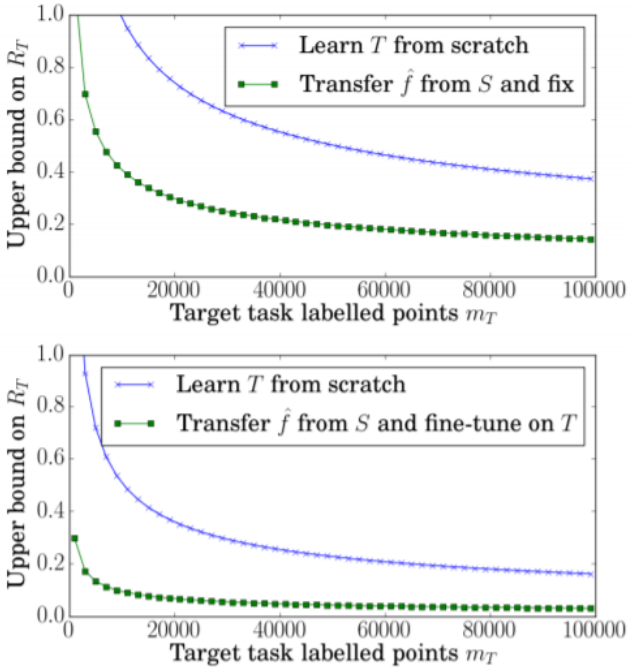
Figure 2. Neural network example with weights transferred from S and fine-tuned on T.



2.3. Applications

To demonstrate the usefulness of the risk bounds, the paper presents experiments with a comparison of risk bounds compared to learning the task T from scratch, without fine-tuning and with fine-tuning. The results show that the risk bounds are tighter with transferring representations with and without fine-tuning in comparison with learning T from scratch. The figure 3 shows the bound values depending on the transfer of representations approach (Representation fixed by source task or Representation fine-tuned on target task) and using the bounds from the theorems described above.

Figure 3. A comparison of risk bounds compared to learning T from scratch, without fine-tuning (top) and with fine-tuning (bottom) from the paper.



In addition, the authors propose a modified loss function which performs better in practice. Thus, instead of using a loss function with the sum of training set log loss and L2 regularization on the weights as follows :

$$\sum_{i=1}^{m}[-y_i \log(\hat{y}_i) - (1-y_i)\log(1-\hat{y}_i)] + \frac{\lambda}{2}\sum_{j=1}^{l}||W^{(j)}||_2^2$$

The regularization penalty was replaced by :

$$\sum_{j=1}^{l}[\frac{\lambda_1(j)}{2}||W^{(j)} - \hat{W}^{(j)}||_2^2] + \sum_{j=1}^{l}[\frac{\lambda_2(j)}{2}||W^{(j)}||_2^2]$$

where y_i and $\hat{y}_i$ are the label and prediction respectively for the i^{th} training point, $W^{(j)}$ the j^{th} weight matrix, $\hat{W}^{(j)}$ its estimate from S (excluding weights for bias units in both cases) in a fully-connected feedforward network with l layers of weights, $||.||_2$ the entry-wise 2 norm and $\lambda_1(.)$ a decreasing function, while $\lambda_2(.)$ controls standard L2 regularization.

This modified regularization penalizes the weights that are very far from the weights learned on S.

2.4. Experiments

In this research paper, the authors chose to experiment on two datasets for image and text classification. The first one is the MNIST dataset that contains many examples of black and white images of handwritten digits classified into 10 classes representing the digits from 0 to 9. The second one is the NEWSGROUPS dataset which is a collection of newsgroup documents partitioned across 20 different classes according to subject matter such as space, religion, etc.

In both cases, and to control the similarity between the intermediate representations of the source and the target tasks, the binary label classes (+ and –) for S and T are constructed as follows:

The original classes of the dataset (10 classes for MNIST and 20 classes for NEWSGROUPS) are randomly and evenly partitioned into two sets S_+ and S_-. Then, a subset of S_+ is randomly selected to construct T_+ and the same goes for T_- based on S_-. The remaining subsets which were not selected from S_+ (respectively S_-) are then added to the opposite class in T_- (respectively T_+), such that $\gamma = \frac{|S_+ \cap T_+|}{|S_+|}$. For example, for the MNIST dataset, the original classes are [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]. Let's suppose that the partitions that were randomly chosen are S_+= [0, 1, 2, 3, 4] and S_- = [5, 6, 7, 8, 9]. With $\gamma = \frac{3}{5} = 0.6$, we can get a subset from S_+ which is [0, 1, 2] added to T_+ and a subset from S_- which is [5, 6, 7] added to T_-. Then, the remaining subsets are added such that we get T+ = [0, 1, 2, 8, 9] and T_- = [5, 6, 7, 3, 4]. Thus, for $0 < \gamma < 1$, the tasks are non-identical and for $\gamma = 1$, S and T are the same task.

The paper then compares between the accuracy averaged over 10 trials on the target task T of four options:

- **Base :** learn T from scratch.
- **Fine-Tune $\hat{f}$:** train $\hat{f}$ on S, transfer and fine-tune $\hat{f}$ and train g on T using the defined regularization.
- **Fix $\hat{f}$:** train and fix $\hat{f}$ learned on S, transfer it and train g on T.
- **Fix $\hat{g}_S \circ \hat{f}$:** train and fix $\hat{g}_S \circ \hat{f}$ on S and transfer it on T.

The authors used for both datasets $\lambda_1(1) = \lambda_2(2) = \lambda := 1$, $\lambda_1(2) = \lambda_2(1) = 0$, $m_T = 500$ and the sigmoid activation function. For MNIST dataset, they used raw pixel intensities, a 784 x 50 x 1 network and m_S = 50000. For NEWSGROUPS dataset, they used TF-IDF weighted counts of most frequent words, a 2000 x 50 x 1 network and m_S = 15000. They also used conjugate gradient optimization with 200 iterations.

2.5. Results

The results of the experiments on MNIST and NEWSGROUPS datasets with different values of γ and according to the four options described previously are shown in table 1.

Table 1. Results of the original paper

Technique	Mnist, γ =			Newsgroups, γ =		
	0.6	0.8	1	0.6	0.8	1
Base	88.4	87.9	87.9	62.6	63.2	66.1
Fine-tune $\hat{f}$	91.9	93.9	95.4	62.3	72.3	83.3
Fix $\hat{f}$	87.5	92.3	97.3	52.2	69.6	83.3
Fix $\hat{g}_S \circ \hat{f}$	67.4	85.6	98.1	55.5	70.7	83.6

The results of this research paper show that when S and T are non-identical, FINE-TUNE $\hat{f}$ is the best especially on MNIST. When the tasks are identical, FIX $\hat{g}_S \circ \hat{f}$ is the strongest. Moreover, it seems that FIX $\hat{f}$ outperforms BASE when the tasks are similar and outperforms FIX $\hat{g}_S \circ \hat{f}$ when the tasks are non-identical on MNIST dataset. However, FIX $\hat{g}_S \circ \hat{f}$ and FIX $\hat{f}$ have nearly the same performance on NEWSGROUPS dataset and so there is no proof of usefulness. Finally, the paper states that when transferring representations is useful, fine-tuning using the regularization penalty proposed in the paper improves the performance compared to using the L2 regularization.

In conclusion, the paper treated how transferring representations can be useful both with and without fine-tuning and proposed a new way of regularization for neural networks which penalizes being far from source task weights and is stricter for lower-level weights. The authors also showed the usefulness of this technique with experiments on image and text classification.

3. Reproducing the research experiments

After reading, analyzing and understanding the research paper "Risk Bounds for Transferring Representations With and Without Fine-Tuning" on which this project is based, the most important part of the work was to reproduce the experiments so as to see whether the statements made in the paper are true or not. In this section, we will detail all the steps followed in order to reproduce the experiments. Also, the findings of our experiments will be presented, along with the difficulties and a comparison with the research paper results.

3.1. Experiments

To reproduce the same experiments, we chose to work with the most used language in Machine Learning which is Python because it is a general purpose language that is easy to read and has a lot of available libraries which make building Deep Learning models straightforward.

The first step was to prepare the data on which the experiments will be conducted. We worked on the same datasets used in the paper which are the MNIST dataset for handwritten digit identification and the NEWSGROUPS dataset for newsgroup documents' classification. To do so, we imported data for both datasets from the library scikit-learn.

The second step was to generate the partitions S_+, S_-, T_+ and T_- to get the new labels for the source and target tasks. This operation is very important since it guarantees the fact that we have a similar intermediate representation between the source task S and the target task T which we will later try to transfer from S to T. We constructed these partitions with the same way described in the paper and depending on the values of γ (0.6, 0.8 or 1) as follows:

Firstly, we used the algorithm 1 "Source Binary Classes" below to create the partitions S_+ and S_- depending on the number of the original classes (10 for MNIST and 20 for NEWSGROUPS). This was done by shuffling an array composed of all the original classes so as to obtain the source binary classes array that has the first half as S_+ and the second half as S_-.

Algorithm 1 Source Binary Classes

Input: size Num
Initialize $SrcClass = array(range(Num))$.
Shuffle $SrcClass$
Return $SrcClass$

Secondly, we used the algorithm 2 "Target Binary Classes" to get the subsets of S_+ and S_- depending on the source binary classes array obtained previously, the number of the original classes and the values of γ (0.6, 0.8 or 1). The partitions S_+ and S_- are shuffled and then only a subset is chosen based on the value of γ to get the subsets of T_+ and T_-. We will then switch the binary classes for the original classes in the subsets that weren't selected from S_+ and S_-. Therefore, if an original class was in S_+ but wasn't selected to be part of T_+, it will be added to T_-. The same goes for original classes not selected from S_- and added to T_+. This way, we guarantee that $\gamma = \frac{|S_+ \cap T_+|}{|S_+|}$. We thus obtain the target binary classes array that has the first half as T_+ and the second half as T_-.

Algorithm 2 Target Binary Classes

Input: array $SrcClass$, size Num, float $Gamma$
Initialize $ZeroClass = array()$
Initialize $OneClass = array()$
Initialize $TrgClass = array()$
for i **in** $SrcClass[0 : (Num/2)]$ **do**
  Append i to $ZeroClass$
end for
for i **in** $SrcClass[(Num/2) : Num]$ **do**
  Append i to $OneClass$
end for
Shuffle $ZeroClass$
Shuffle $OneClass$
for i **in** $ZeroClass[0 : (Num/2 * Gamma)]$ **do**
  Append i to $TrgClass$
end for
for i **in** $OneClass[(Num/2 * Gamma) :]$ **do**
  Append i to $TrgClass$
end for
for i **in** $OneClass[0 : (Num/2 * Gamma)]$ **do**
  Append i to $TrgClass$
end for
for i **in** $ZeroClass[(Num/2 * Gamma) :]$ **do**
  Append i to $TrgClass$
end for
Return $TrgClass$

Finally, we used the algorithm 3 "Binary Outputs" to obtain the new binary labels instead of the original ones in the outputs depending on the partitions S_+ and S_- obtained by the "Source Binary Classes" algorithm and to which one they belong for the source task. The same algorithm 3 "Binary Outputs" is also used to obtain the new binary labels instead of the original ones in the outputs depending on the partitions T_+ and T_- obtained by the "Target Binary Classes" algorithm and to which one they belong for the target

task. Thus, if the original class belongs to the first half of the binary classes array, this class is replaced by the number 0 (corresponding to the label "+" in practice). In the same way, if the original class belongs to the second half of the binary classes array, this class is replaced by the number 1 (corresponding to the label "–" in practice). As a result, we get the binary outputs as labeled by the source task S and the target task T.

Algorithm 3 Binary Outputs

Input: data $Outputs$, array $Class$, size Num
Initialize $BinaryOutputs = array()$
Initialize $Length = len(Outputs)$
for i **in** $range(Length)$ **do**
  if $Outputs[i]$ **in** $Class[0 : (Num/2)]$ **then**
    Append 0 to $BinaryOutputs$
  else if $Outputs[i]$ **in** $Class[(Num/2) : Num]$ **then**
    Append 1 to $BinaryOutputs$
  end if
end for
Return $BinaryOutputs$

After preparing the datasets and labeling the examples according to the partitions S_+, S_-, T_+ and T_- for the source and target tasks, the third step was to define the new regularization presented in the research paper instead of L2 regularization when fine-tuning. This new regularization is presented in the algorithm 4 "Paper Regularization". This algorithm does multiple operations with the Machine Learning framework TensorFlow to calculate the square of the difference between each element of the matrix weights and the source weights which were saved after training the model on the source task, sum these squares and multiply the result by the coefficient. This modified regularization penalizes the weights that are very far from the weights learned on S.

Algorithm 4 Paper Regularization

Input: data $MatrixWeights$, data $SourceWeights$, float $Coefficient$
Initialize $Output = Coefficient * sum(square(MatrixWeights - SourceWeights))$
Return $Output$

The final step was to construct the architecture of the Multi Layer Perceptron described in the research paper for the datasets MNIST and NEWSGROUPS using Keras which is a high-level Deep Learning library capable of running on top of TensorFlow, CNTK, or Theano.

In this final step, we used for both datasets $m_T = 500$. For MNIST dataset, we used raw pixel intensities, a 784 x 50 x 1 network and $m_S = 55000$. For NEWSGROUPS dataset, we used TF-IDF weighted counts of most frequent words, a 2000 x 50 x 1 network and $m_S = 10000$. We also used stochastic gradient descent with 500 iterations and a batch size of 32.

In order to experiment on transferring representations, we needed to build the model and train it on the source task. The model is as follows:

- **Layer 0 (input)**: input shape = 784 (MNIST) - 2000 (NEWSGROUPS).
- **Layer 1 (hidden layer f)**: dense with 50 neurons and relu activation.
- **Layer 2 (output g)**: dense with 1 neuron, sigmoid activation and L2 regularization.

This model is compiled using Adam optimizer with a learning rate of 0.001 and then trained on the source train data with 500 epochs and a batch size of 32. The obtained weights for the source task after the training are saved so as to be used in the transferring experiments depending on the following four options:

- **Base :** To learn T from scratch, we use the exact same model used for training the source task using Adam optimizer with a learning rate of 0.1.
- **Fine-Tune $\hat{f}$:** The model is slightly altered for fine-tuning $\hat{f}$ as follows :
  - **Layer 0 (input)**: input shape = 784 (MNIST) - 2000 (NEWSGROUPS).
  - **Layer 1 (hidden layer f)**: dense with 50 neurons, relu activation and the regularization described in the paper.
  - **Layer 2 (output g)**: dense with 1 neuron, sigmoid activation and L2 regularization.

  After loading the saved weights for $\hat{f}$ learned on S, this model is compiled using Adam optimizer with a learning rate of 0.1 and then trained on the target train data with 500 epochs and a batch size of 32. Thus, the transferred weights of $\hat{f}$ are fine-tuned on the target task T using the defined regularization and the output g is completely trained on T.
- **Fix $\hat{f}$:** The model is slightly altered for fix $\hat{f}$ as follows :
  - **Layer 0 (input)**: input shape = 784 (MNIST) - 2000 (NEWSGROUPS).

  - **Layer 1 (hidden layer f)**: dense with 50 neurons, relu activation and trainable = false.
  - **Layer 2 (output g)**: dense with 1 neuron, sigmoid activation and L2 regularization.

  After loading the saved weights for $\hat{f}$ learned on S, this model is compiled using Adam optimizer with a learning rate of 0.1 and then trained on the target train data with 500 epochs and a batch size of 32. Thus, the weights of $\hat{f}$ are transferred to T and fixed with trainable = false and g is completely trained on T.

- **Fix $\hat{g}_S \circ \hat{f}$:** The model used for learning on S is directly used on T.

The evaluation of the experiment for each of the four options (Base, Fine-Tune $\hat{f}$, Fix $\hat{f}$ and Fix $\hat{g}_S \circ \hat{f}$) was based on the accuracy averaged on 10 trials for each γ value (0.6, 0.8 and 1).

3.2. Results

The results of the reproduced experiments on MNIST and NEWSGROUPS datasets with different values of γ (0.6, 0.8 and 1) and according to the four options Base, Fine-Tune $\hat{f}$, Fix $\hat{f}$ and Fix $\hat{g}_S \circ \hat{f}$ are shown in table 2.

Table 2. Results of the reproduced experiment

Technique	Mnist, $\gamma =$			Newsgroups, $\gamma =$		
	0.6	0.8	1	0.6	0.8	1
Base	77.5	74.0	83.1	69.7	70.2	67.0
Fine-tune $\hat{f}$	82.9	87.8	86.0	70.3	73.1	77.2
Fix $\hat{f}$	83.8	86.5	97.0	56.8	64.0	74.1
Fix $\hat{g}_S \circ \hat{f}$	59.2	79.6	98.6	57.0	68.1	77.8

The results of the reproduced experiment show that when S and T are non-identical, FINE-TUNE $\hat{f}$ is the best especially on NEWSGROUPS. When the tasks are identical, FIX $\hat{g}_S \circ \hat{f}$ is the strongest. Moreover, it seems that FIX $\hat{f}$ outperforms BASE when the tasks are similar and outperforms FIX $\hat{g}_S \circ \hat{f}$ when the tasks are non-identical on NEWSGROUPS dataset. However, FIX $\hat{g}_S \circ \hat{f}$ and FIX $\hat{f}$ have nearly the same performance on MNIST dataset and so there is no proof of usefulness. Finally, when transferring representations is useful, fine-tuning using the proposed regularization penalty improves the performance compared to using the L2 regularization as stated by the research paper.

3.3. Comparison, difficulties and comments

When comparing the results presented in the research paper and the results of the reproduced experiment, it seems that they lead to the same conclusion even if the accuracy scores are different. Indeed, both experiments state that when S and T are non-identical, FINE-TUNE $\hat{f}$ is the best and when the tasks are identical, FIX $\hat{g}_S \circ \hat{f}$ is the strongest. However, the results are reversed for MNIST and NEWSGROUPS datasets in these experiments. For example, FIX $\hat{g}_S \circ \hat{f}$ and FIX $\hat{f}$ have nearly the same performance on NEWSGROUPS dataset and so there is no proof of usefulness on the paper's experiments, whereas the same thing is found for MNIST dataset in the reproduced experiments.

These differences are due to the fact that the code used in these experiments is not shared by the authors and that there aren't enough details about the conducted experiments in the research paper. The lack of information made it more difficult to reproduce the experiments in the exact same conditions and thus caused these differences in the results.

Indeed, the paper does not mention the partitions for S_+ and S_- used in the experiment which highly influences the accuracy scores as found in the reproduced experiments. This can be explained by whether similar original classes of a dataset are in the same partition or not. For example, if digits 1 and 7 which are similar are in the same partition (S_+ or S_- for the source task and/or T_+ or T_- for the target task) and even if the model confuses 1 and 7, this error will not be considered as in S and/or T they belong to the same class and thus the performance will not be penalized. However, if the same digits 1 and 7 are part of different partitions, the error of confusing both will be taken into consideration and will lead to lower accuracy scores compared to when they belong to the same partition.

Furthermore, another difference between the reproduced experiments and the paper ones is that we used gradient descent in the reproduced experiments instead of conjugate gradient optimization as in the paper since it is already implemented in Keras, more efficient and easier to use in practical Deep Learning problems, especially if we want to transfer representations with fine-tuning using the regularization penalty proposed in the research paper on other tasks.

Other comments unrelated to the implementation details but linked to the practical use of the findings of the paper is that the function ω which is the function measuring a transferability property depends on

the model and the problem setting. Hence, it is hard to find it in the general case, which means that the theoretical guarantees for transferring representations should be hard to use in practice.

Additionally, the paper gives a PAC-Bayes target task risk upper-bound under suitable conditions. This does not prove that in all cases the performance will be better when transferring representations from a source task to a target one as the tightness of the bound depend on the similarity between the two tasks which is difficult ti accurately measure.

4. Further experiments

In order to verify the usefulness of transfer learning in real world applications using deep convolutional neural networks, we used transfer learning on Kaggle competitions that tackle image classification. The competitions that were chosen are "Dog Breed Identification" and "Invasive Species Monitoring" which are classification prediction competitions.

For both competitions, we used the kernel "Keras VGG19 Starter" of the user Orangutan which was modified for our tests. To do so, we changed the deep learning model used "VGG19" with "ResNet50". Both of these Keras Applications are deep learning models that are made available alongside pre-trained weights and that can be used for prediction, feature extraction, and fine-tuning. However, ResNet runs and trains faster compared to VGG19 and its size is actually substantially smaller due to the usage of global average pooling rather than fully-connected layers.

4.1. "Dog Breed Identification" competition

The goal of the competition is to create a classifier which is able to determine a dog's breed from a photo (chihuahua, pomeranian, samoyed...). A strictly canine subset of ImageNet is provided in order to practice fine-grained image classification with 120 breeds of dogs and a limited number training images per class.

First, we get the breed classes and dummify them. Second, we read the train images using OpenCV, resize them to 224x224 pixels and assign for each image its one hot class. The test images are also read using OpenCV and resized to 224x224 pixels. All images are loaded as numpy arrays. Third, and since it is important to create a validation set so that we can watch the performance of the model on independent data which is unseen to the model in training, we split the current training set and the corresponding labels so that we set aside 30 % of the data at random and put these in validation sets. Then, we build the CNN architecture depending on the three following options:

- **Base :** To learn T from scratch, we use the ResNet model available in Keras. We then remove the final layer of the ResNet model and instead replace it with a single dense layer with 120 neurons and softmax activation, with each neuron representing a dog breed class.
- **Fine-Tune $\hat{f}$:** We use the ResNet model available in Keras with the "ImageNet" weights. We then remove the final layer of the ResNet model and instead replace it with a single dense layer with 120 neurons and softmax activation, with each neuron representing a dog breed class. Also, the top layers have their attribute trainable equal to true since in this case we need to Fine-Tune $\hat{f}$.
- **Fix $\hat{f}$:** We use the ResNet model available in Keras with the "ImageNet" weights. We then remove the final layer of the ResNet model and instead replace it with a single dense layer with 120 neurons and softmax activation, with each neuron representing a dog breed class. Also, the top layers have their attribute trainable equal to false since in this case we need to Fix $\hat{f}$.

Finally, the model is compiled with categorical crossentropy loss and trained on the target task train data with 20 epochs and used to predict the target test data which is saved in a CSV file with each test image and the probabilities of belonging to each one of the 120 dog breeds. This file is submitted on Kaggle in order to be evaluated on Multi Class Log Loss between the predicted probability and the observed target.

4.2. "Invasive Species Monitoring" competition

Invasive species can have damaging effects on the environment, the economy, and even human health. Despite widespread impact, efforts to track the location and spread of invasive species are so costly that theyre difficult to undertake at scale. Because scientists cannot sample a large quantity of areas, some machine learning algorithms are used in order to predict the presence or absence of invasive species in areas that have not been sampled. In this playground competition, the objective is to develop algorithms to more accurately identify whether images of forests and foliage contain invasive hydrangea or not.

First, we read the train and test images using OpenCV and resize them to 224x224 pixels. All images are

loaded as numpy arrays. Second, and since it is important to create a validation set so that we can watch the performance of the model on independent data which is unseen to the model in training, we split the current training set and the corresponding labels so that we set aside 30 % of the data at random and put these in validation sets. Third, we build the CNN architecture depending on the three following options:

- **Base :** To learn T from scratch, we use the ResNet model available in Keras. We then remove the final layer of the ResNet model and instead replace it with a single dense layer with 1 neuron for the binary classification (invasive, not invasive) and sigmoid activation.
- **Fine-Tune $\hat{f}$:** We use the ResNet model available in Keras with the "ImageNet" weights. We then remove the final layer of the ResNet model and instead replace it with a single dense layer with 1 neuron for the binary classification (invasive, not invasive) and sigmoid activation. Also, the top layers have their attribute trainable equal to true since in this case we need to Fine-Tune $\hat{f}$.
- **Fix $\hat{f}$:** We use the ResNet model available in Keras with the "ImageNet" weights. We then remove the final layer of the ResNet model and instead replace it with a single dense layer with 1 neuron for the binary classification (invasive, not invasive) and sigmoid activation. Also, the top layers have their attribute trainable equal to false since in this case we need to Fix $\hat{f}$.

Finally, the model is compiled with binary crossentropy loss and trained on the target task train data with 10 epochs and used to predict the target test data which is saved in a CSV file with each test image and the probability of it containing an invasive species. This file is submitted on Kaggle to be evaluated on area under the ROC curve between the predicted probability and the observed target.

4.3. Results

The results for the "Dog Breed Identification" competition are shown in table 3 below:

Table 3. Results of "Dog Breed Identification"

Technique	Log Loss score
Base	4.165
Fine-tune $\hat{f}$	4.195
Fix $\hat{f}$	**1.771**

These results show that learning the target task (Dog Breed Classification) from scratch and transferring representations from the source task (ImageNet classification) into the target task with fine-tuning have approximately the same performance. However, fixing f when transferring representations from the source task to the target one gets much better results with an improvement of nearly 3 points.

The results found for the "Dog Breed Identification" competition can be explained by the fact that the provided data is very few with only 10222 images for the train set and 10357 images for the test set which are divided on 120 dog breed classes (approximately 85 images for each class). Moreover, and since the split on the training set to get the validation set is not fully controlled, we can not guarantee that this validation set contains images from every class of the 120 separate dog breed classes and thus some can not be represented. Hence, in the base and fine-tuning cases, the model is trained on these datasets, the weight values are altered and so the model tends to over-fit after 20 epochs. However, the results are good in the case of fix $\hat{f}$ because the pre-trained model ResNet has already been trained to identify dogs, as well as a lot of other objects from the ImageNet dataset. Thus, these weights which are directly used to predict the dog breed perform better than if they are trained on the small dataset.

This experiment on "Dog Breed Identification" proves that, as said in the research paper, fixing the source task representation when transferring is the best option when over-fitting is a strong concern.

The results for the "Invasive Species Monitoring" competition are shown in table 4 below:

Table 4. Results of "Invasive Species Monitoring"

Technique	AUC ROC score
Base	95.579
Fine-tune $\hat{f}$	96.518
Fix $\hat{f}$	**96.811**

These results prove that transferring representations from a source task (ImageNet classification) into a target task (Image contains invasive hydrangea or not) is better than learning this target task from scratch. Indeed, the Base AUC ROC score is the lowest. We also notice that fixing $\hat{f}$ is slightly better than fine-tuning $\hat{f}$ when transferring representations.

The results found for the "Invasive Species Monitoring" competition can be explained by the fact that

the provided data is very few with only 2295 images for the train set and 1531 images for the test set (approximately 1122 images for each class). Moreover, the pre-trained model ResNet is not already trained to identify if an image contains invasive hydrangea or not. Hence, the results are only slightly better when fixing $\hat{f}$ in comparison with fine-tuning $\hat{f}$ and learning the target task from scratch.

This experiment on "Invasive Species Monitoring" shows also that fixing the source task representation when transferring is a good option when we have over-fitting. However, the difference between fixing $\hat{f}$ and fine-tuning $\hat{f}$ is not as obvious as compared to the one in "Dog Breed Identification" since we have approximately 1122 examples for each class (invasive, not invasive) as opposed to only 85 examples for each dog breed. Hence, the over-fitting is lesser on "Invasive Species Monitoring" dataset than on "Dog Breed Identification" dataset.

5. Conclusion

Transfer learning is one crucial step toward powerful AI and has lead to considerable improvements in deep learning. Thus a theoretical approach for formulating when and how it succeeds is very important and much needed. The paper published in the ICML 2017 "Risk Bounds for Transferring Representations With and Without Fine-Tuning" by D. McNamara and M.F. Balcan is one of the first research papers that tackles this problem to find theoretical guarantees for transfer learning.

Through this project, we were able to analyze, understand and reproduce the experiments of the research paper. We managed to find similar results compared to the paper even with the difficulties encountered. We also tested the practical effectiveness of transferring representations in real world applications using Datasets for "Dog Breed Identification" and "Invasive Species Monitoring" from Kaggle competitions.

Working on replicating the experiments of this research paper was a great learning experience from a technical point of view (implementing the experiments with Python and Keras and working on image classification competitions on Kaggle) and from a theoretical point of view, especially for transferring representations and its usefulness when labeled data is scarce and thus over-fitting is a real problem.

References

Chollet, Francois. Building powerful image classification models using very little data. In *The Keras Blog*, 2016.

Competition, Kaggle. Invasive species monitoring. In *https://www.kaggle.com/c/invasive-species-monitoring*, 2017.

Competition, Kaggle. Dog breed identification. In *https://www.kaggle.com/c/dog-breed-identification*, 2018.

Daniel McNamara, Maria-Florina Balcan. Risk bounds for transferring representations with and without fine-tuning. In *International Conference on Machine Learning (ICML)*, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren Jian Sun. Deep residual learning for image recognition. In *Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

Karen Simonyan, Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015.

Orangutan. Keras vgg19 starter. In *Kaggle Kernels: https://www.kaggle.com/orangutan/keras-vgg19-starter*, 2017.