# Finding the Needle in the Pre-Trained Model Zoo

## The Use of Rich Metadata and Graph Learning to Estimate Task Transferability

Hilco van der Wilk

# Finding the Needle in the Pre-Trained Model Zoo

## The Use of Rich Metadata and Graph Learning to Estimate Task Transferability

by

Hilco van der Wilk

to

obtain the degree of Master of Science at the Delft University of Technology,

defended publicly on Tuesday, June 25, 2024.

| | |
|---|---|
| Thesis Advisor: | A. Anand |
| Daily supervisor: | R. Hai |
| Daily co-supervisor: | Z. Li |
| Core committee member: | Q. Song |
| Faculty: | Electrical Engineering, Mathematics and Computer Science, Delft |

Cover: Inspired by Herman Brood's "Porsche", which was above my desk while writing this thesis. It was generated using the at this time state-of-the-art text-to-image generator, Dall-E.

TU Delft

# Finding the Needle in the Pre-Trained Model Zoo


## Abstract

The democratization of machine learning through public repositories, often known as model zoos, has significantly increased the availability of pre-trained models for practitioners. However, this abundance can make it difficult to choose the most suitable pre-trained model for fine-tuning on new tasks. Although various methods have been proposed in the field of transferability estimation to address this issue, these methods can take hours to execute and may still fail to find the optimal pre-trained model for fine-tuning. By exploring a new graph learning-based approach to transferability estimation, we outperform state-of-the-art methods such as LogME, improving the accuracy of the best-predicted model by up to 31.5% in less than 5 minutes.

# Preface & Acknowledgements

Over the course of this thesis, many exciting advancements have taken place in the field of Computer Science. Notably, the launch of ChatGPT just before I began this thesis made Artificial Intelligence more tangible for non-computer scientists. This personal experience made it easier to explain my thesis topic to friends and family by relating it to their interactions with AI. Although machine learning was not my primary focus earlier in my master's program, studying it through this thesis has been very insightful.

I write these words as the fourth summer of my Master's in Computer Science approaches. Completing my degree on time was never my intention; I was too enthusiastic about the company where I completed my bachelor thesis to leave it. After a brief break from my studies, I was introduced to Rihan Hai and Ziyu Li and their research. I would like to thank them both for their enthusiasm during this period. My brief industry experience combined with their academic insights led to many interesting conversations. Thanks to them, I also had the exciting opportunity to co-author a paper published at the esteemed International Conference on Data Engineering (ICDE 2024).

Finally, I would like to thank my colleagues at bunq, specifically Ali el Hassouni and Nick van de Groes, who have supported me not only through discussions related to my thesis, but also by enabling me to balance my work with my studies.

*Hilco van der Wilk*
*June 2024*

# Contents

# List of Figures

# List of Tables

# 1

# Introduction

Deep learning, a branch of machine learning, has transformed artificial intelligence by creating models that can learn and identify complex patterns from large datasets. These models, known as deep neural networks, are made up of many layers that allow them to understand and process information at various levels of detail. Deep learning has achieved remarkable results in many areas, such as recognizing objects in images, understanding spoken language, translating text, and even playing complex games. The power of deep learning is evident in real-world applications (S. Dong et al. 2021). For instance, in self-driving cars, deep learning helps vehicles to see and understand their surroundings, making driving decisions in real-time (Chib & Singh 2023). In healthcare, it helps diagnose diseases from medical images more accurately and quickly than traditional methods (Q. Li et al. 2014). More recently, text and image generation models have shown impressive capabilities in creating human-like text for applications such as chatbots, content creation, and automated translation services (Raiaan et al. 2024; Kandwal & Nehra 2024).

The flexibility of deep learning has played a major role in its success: the knowledge a neural network acquired by training for one task can be reused for a different purpose. Like many concepts within the field of machine learning, this notion of knowledge transfer is likely inspired by human and animal learning (Zhuang et al. 2021). Humans benefit from facing new tasks equipped with knowledge learned from previous (similar) tasks. For example, someone who knows how to play the guitar can learn to play the piano more quickly due to the shared musical knowledge. In the field of machine learning, this practice is known as transfer learning. It has proven to be highly effective, making *first pre-training, then fine-tuning* the de facto paradigm of applying deep learning in practice. In the pre-training phase, a large and diverse *upstream* dataset is used to train a neural network. An often smaller *downstream* dataset is then used to fine-tune the pre-trained model for a specific task. Using a pre-trained model prevents the resource-intensive process of training a model from scratch. Moreover, it bypasses the expensive, time-consuming, and often unrealistic data collection and annotation that training a neural network from scratch requires.

Today, many pre-trained models are available in public online platforms such as HuggingFace[1], Kaggle Models[2], and PyTorch Hub[3]. These repositories of pre-trained models are referred to as *model zoos*. Model zoos have been widely adopted in recent years, as they offer convenient access to a collection of pre-trained models, including cutting-edge deep learning architectures. This lowers the expertise barrier, enabling non-expert individuals to apply complex deep learning models in their applications. By using a model zoo for fine-tuning, practitioners can effectively address various target tasks, even when the available training data are limited in size (Deshpande et al. 2021).

[1] https://huggingface.co/
[2] https://www.kaggle.com/models
[3] https://pytorch.org/hub/

## 1.1. Transferability Estimation

As the number of available pre-trained models grows, a critical question arises: "*Which pre-trained model will have the best performance on my downstream task?*". Choosing the right pre-trained model has a substantial impact on the effectiveness of fine-tuning (Deshpande et al. 2021), especially in scenarios with little training data (Bassignana et al. 2022). A straightforward solution is to fine-tune all available pre-trained models, which is computationally expensive, and often infeasible in practice.

To this end, *Transferability Estimation* has emerged to assess transferability at a low computational cost. In this setting, the goal is to predict the best pre-trained model given a target task, without fine-tuning all pre-trained models. Most transferability estimation methods approach it similarly; they perform a forward pass using all the target task's samples over each pre-trained model's feature extractor and analyze the extracted features (Nguyen et al. 2020; You et al. 2021; Ding et al. 2022; Ibrahim et al. 2023). This analysis often produces a score, and the pre-trained models are ranked according to this score. The effectiveness of a transferability estimation method is typically evaluated by measuring the (rank) correlations between the actual performances and the predicted scores.

### 1.1.1. Effectiveness of Extracted Features in Diverse Model Zoos

A shortcoming of these methods is that they do not consider any meta-information about the target task or the pre-trained model, even though these can be useful indicators of fine-tuning success. Factors such as target dataset size, number of labels, and pre-trained model architecture have been shown to affect downstream fine-tuning performance. H. Li et al. (2023) demonstrate this by reframing the transferability estimation problem as a recommendation problem and using a linear regression model to predict fine-tuning performance based on these basic meta-features. They further observe that transferability estimation methods are often evaluated on only a few pre-trained models, sharing the same architecture and source dataset. In their more diverse setting, methods that use these extracted features become less effective. While promising, H. Li et al. (2023) only include coarse-grained metadata to learn to predict transferability, overlooking valuable insights from previous works and are not able to capture more complex existing relationships between datasets and pre-trained models.

### 1.1.2. Efficiency Matters in Large Model Zoos

These days, public model zoos often contain thousands of pre-trained models. Not only are model zoos growing in size, pre-trained models themselves are also becoming more complex (i.e., they are increasing in number of parameters). While the transferability estimation methods referred to above are indeed orders of magnitude faster than fine-tuning, these aspects make performing a forward pass over all pre-trained models increasingly computationally expensive. *Efficiency* should therefore become a more prominent concern in transferability estimation in the era of large models and zoos.

## 1.2. Introducing TransferGraph

In the field of data management systems, such as data lakes (Hai et al. 2023), datasets have been structured as graphs to, for example, model semantic similarity (Castro Fernandez et al. 2018). Related works on transferability estimation have uncovered various useful metadata and rich relationships between pre-trained models and datasets. To overcome the challenges mentioned above, we borrow the idea of representing the transferability estimation problem as a graph and propose a new framework for transferability estimation called *TransferGraph*[4]. Figure 1.1 gives a high-level overview of our adaptation to transferability estimation. As the name suggests, we convert transferability estimation into a graph learning problem, where relationships between tasks and pre-trained models are modeled as edges between nodes in a graph. Using this graph, the goal is to predict the link between an unseen target dataset and the pre-trained models in

[4]The work on TransferGraph has been accepted at the International Conference on Data Engineering (ICDE) 2024 (https://icde2024.github.io/papers.html). Source code, supplementary material, and experiment artifacts are available at https://github.com/TransferGraph/transfergraph.

the model zoo.

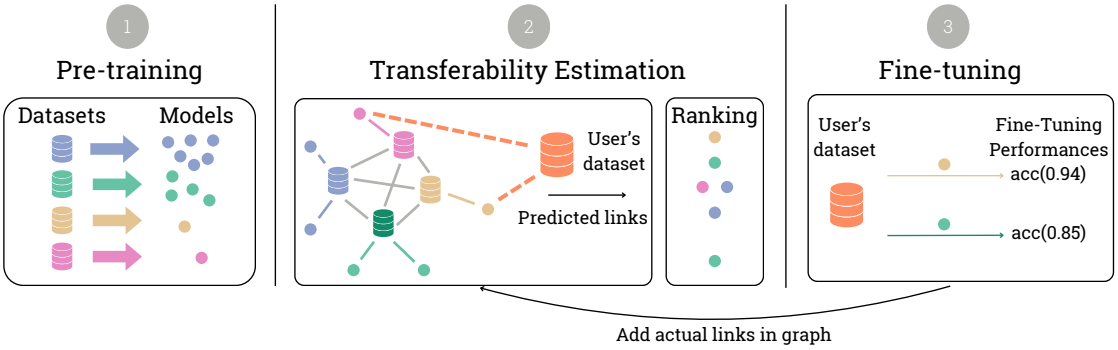
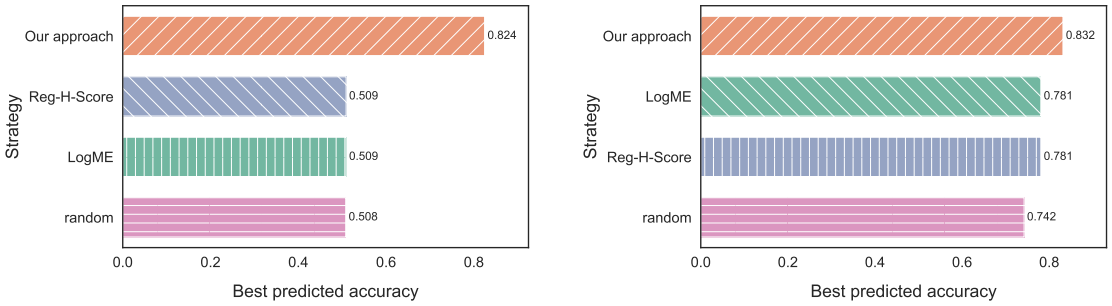



**Figure 1.1:** High-level overview of TransferGraph's adaptation of the paradigm of transferability estimation. Stage ① represents the pre-training stage, where models are trained from scratch. The output of this stage is pre-trained models, upstream datasets and upstream accuracies, which are used in stage ② to construct a graph and predict links for which model is the best fine-tuning candidate. In stage ③, the best few pre-trained models are fine-tuned, and the actual accuracies are added back into the graph for later learning.

Edges can be either *dataset-dataset*, representing the similarity between datasets, or *model-dataset*, representing the historical fine-tuning performances of a pre-trained model on a dataset. By also incorporating the coarse metadata of pre-trained models and datasets, we can better understand the underlying dynamics between them. This results in a notable improvement in final fine-tuning accuracies over the state-of-the-art, as shown in Figure 1.2.

This framework also solves the efficiency issues mentioned earlier. By learning from past fine-tuning performances, we avoid the forward pass most transferability estimation methods require. Contrary to most existing methods, this means that TransferGraph can do efficient predictions on unseen target dataset nodes and scales better with larger model zoos and pre-trained models.




(**a**) Stanfordcars

(**b**) Tweet_Eval/Emotion.

**Figure 1.2:** The accuracy of the best predicted model for our approach and two baselines, compared to randomly picking a model to fine-tune.

## 1.3. Research Questions

The research questions of this thesis are as follows:

**RQ** How can transferability estimation be performed more effectively and efficiently in large and diverse model zoos?

**RQ1** Which methodologies for transferability estimation have been introduced in existing literature?

**RQ2** What types of metadata can be used to effectively estimate transferability?

**RQ3** How can we improve the effectiveness of transferability estimation in diverse model zoos?

**RQ4** How can transferability estimation be made more efficient in large model zoos?

## 1.4. Thesis Outline and Contributions

This section will outline the rest of this thesis and summarize the contributions for each chapter. In Chapter 2, the concepts of transfer learning, model zoos, and graph learning are introduced. Additionally, we introduce the research field of transferability estimation and conduct a survey on related work in transferability estimation, which is more extensive than existing surveys, such as those by Bai et al. (2023) and Agostinelli et al. (2022), and offers a more intuitive taxonomy by categorizing methods based on the used input.

**Contributions Chapter 2**

- A survey on the related work on transferability estimation, which is more extensive than existing surveys, such as those by Bai et al. (2023) and Agostinelli et al. (2022), and offers a more intuitive taxonomy by categorizing methods based on the used input.

In Chapter 3, we formally introduce transferability estimation and discuss the limitations and challenges faced by previously proposed methods. Most notably, they disregard useful metadata in their estimation and are impractically inefficient for large model zoos. These limitations serve as a basis for motivating our solutions aimed at improving transferability estimation, both in terms of effectiveness and efficiency. The solution is two-fold. First, to effectively estimate transferability in diverse model zoos, additional pre-trained model and dataset metadata could be used to learn from past fine-tuning performances. Second, to effectively incorporate these metadata, we propose reframing the transferability estimation problem as a link prediction problem on a graph. By doing so, we hypothesize that our solution can better capture complex relationships between datasets and pre-trained models.

**Contributions Chapter 3**

- We identify the limitations and challenges of previously proposed methods. Most notably, they disregard useful metadata in their estimation and are impractically inefficient for large model zoos.
- A reformulation of the transferability estimation problem to a graph link prediction problem.

Our graph learning-based framework to transferability estimation, called *TransferGraph*, is introduced in Chapter 4. In this chapter, we motivate our system design, and we cover the steps needed to execute the approach, including metadata collection, graph construction and learning, and finally making the prediction.

**Contributions Chapter 4**

- A novel framework which solves transferability estimation through graph learning. It includes an end-to-end process from collecting the metadata, graph construction and learning, and finally making the prediction.

Where Chapter 4 covers the layout of the framework, Chapter 5 shows the implementation details and the design of our system to collect metadata, other transferability estimation scores, and fine-tuning performances. Our system expands on other existing benchmark suites and HuggingFace's Transformers library. We also introduce our experiment setup and show how we use the system to perform the preparation needed for evaluation.

**Contributions Chapter 5**

- Expanding existing benchmark suites and building on top of HuggingFace's Transformers library, we introduce an easily extensible system to load datasets and pre-trained models, compute baseline transferability estimation scores and do result analysis.

In Chapter 6, *TransferGraph* is evaluated against two state-of-the-art baselines, LogME (You et al. 2021) and Reg-Hscore (Ibrahim et al. 2023). The evaluation includes both computer vision and natural language processing tasks, and we compare variation of our method to the baselines in terms of effectiveness and efficiency.

Additionally, we introduce an improved way to measure the success of transferability estimation methods across multiple tasks, which may vary in the difficulty with which they are learned. We evaluate four different graph learning methods and three different supervised learning methods to train a regression model: Node2Vec (Grover & Leskovec 2016), Node2Vec+ (R. Liu et al. 2023), Graph Attention Networks (GAT) (Veličković et al. 2018) and GraphSAGE (Hamilton et al. 2017) for graph learning; linear regression, random forests and eXtreme Gradient Boosting (T. Chen & Guestrin 2016) for the regression model. We also vary the size of the model zoo to show the stability of our proposed method. Ultimately, we show that our most competitive approach outperforms the state-of-the-art methods on most evaluation metrics and is orders of magnitude faster.

**Contributions Chapter 6**

- In contrast to related wroks, we propose a more consistent approach to measure the effectiveness of transferability estimation methods over multiple tasks, which may vary in terms of how much they benefit from transferability estimation.
- We show that TransferGraph outperforms the state-of-the-art transferability estimation methods LogME (You et al. 2021) and Reg-HScore (Ibrahim et al. 2023) in terms of effectiveness on most evaluation metrics.
- We show that TransferGraph can estimate transferability orders of magnitude faster than these methods.
- We show that TransferGraph's performance is stable when varying the model zoo size in terms of number of pre-trained models to choose from.

# 2

# Background & Related Work

Transferability estimation, also known as *model selection* or *model search*, aims to efficiently predict the ability of a model to transfer knowledge from one task to another (Agostinelli et al. 2022). Simply put, given a new dataset of input (e.g., text or images) and label pairs, we do not want to waste computational resources by randomly fine-tuning available pre-trained models for this new task. To draw a parallel with human learning, imagine that you want to learn to play the piano; it would not make sense to spend countless hours getting lessons from a plumber, a mathematician, and a doctor. Instead, you would intuitively seek guidance from a musician.

Except, having this intuition for knowledge transfer in deep learning has proven difficult, even for experts (Bassignana et al. 2022). This chapter will first give background into the concepts relevant to this thesis; transfer learning, model zoos, and graph learning. Next, we expand existing surveys on transferability estimation (Agostinelli et al. 2022; Bai et al. 2023) by including more related works and offering a more intuitive categorization. Finally, we provide background knowledge on graph learning, which is necessary for understanding our solution presented in Chapter 3.

## 2.1. Transfer Learning

Traditional machine learning techniques have seen significant progress in various knowledge engineering areas such as classification, regression, clustering, and data mining. Despite these advances, real-world applications frequently encounter limitations. Ideally, there are many labeled training samples that share the same feature space and distribution as the test samples. Unfortunately, in many scenarios, obtaining sufficient and representative training data can be a costly and time-consuming effort.

Semi-supervised learning partially solves this problem by learning from a small set of labeled data in combination with a large quantity of unlabeled data (Engelen & Hoos 2019). However, gathering unlabeled data is often also challenging, ultimately resulting in training with risk of overfitting and slow convergence. Even with enough labeled training data, training a machine learning model from scratch is computationally expensive and can take up to months on modern hardware.

Transfer learning has been very successful in combating these problems, especially in the domain of deep learning, where *data dependence* is even greater (C. Tan et al. 2018). Transfer learning is a set of techniques that aim to improve the performance of *target learners* in a *target task* by transferring knowledge from a different *source task* (Zhuang et al. 2021). This is commonly achieved through *fine-tuning*, which involves optimizing one or more layers of the model's parameters for the target task, including a newly initialized classification layer. This strategy is also referred to as *network-based* transfer learning (Zhuang et al. 2021).

An example of how this works is given in Figure 2.1. A source dataset with four possible

labels is first used to pre-train a deep learning network. To fine-tune it, the classification head layer is removed and randomly initialized for a binary classification target task. The new layer, including one or more of the pre-trained model layers, is then updated using the new target task.

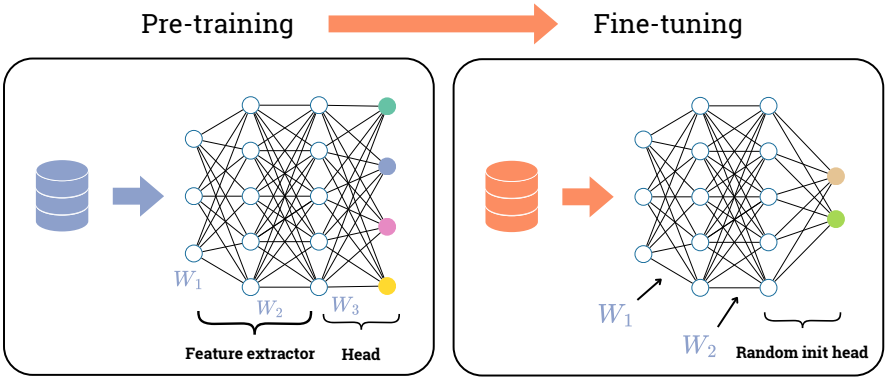



**Figure 2.1:** An example of a four-class pre-trained model (left), fine-tuned for a binary classification task (right). Fine-tuning involves randomly reinitializing a newly classified head and retraining it, potentially along with the other layers' parameters.

Which (layers of) parameters to best (re)train and which to freeze is a mostly undiscovered problem (Goerttler & Obermayer 2024) and is determined by practitioners from empirical experience. For image classification tasks, traditionally all layers were frozen and only a newly initialized classification layer was trained from scratch. Recently, especially in the field of natural language processing, all layers are often retrained by default. Retraining all parameters is more computationally expensive and becomes impractical for modern models with billions of parameters. Techniques such as *LoRA* (Hu et al. 2021) are emerging to do this more efficiently. They freeze all model parameters and insert trainable rank decomposition matrices into each layer of the source model, reducing the number of trainable parameters.

## 2.2. Model Zoos

*Model zoos* or *model hubs* are repositories of pre-trained machine learning models. These terms are often used to refer to public web-based repositories such as HuggingFace[1], Kaggle Models[2], and PyTorch Hub[3]. Not only do they offer easy sharing and reuse of pre-trained models and datasets, but they often also have out-of-the-box tools for the various stages of the machine learning lifecycle. This makes them have a large overlap with other existing systems and tools for managing machine learning artifacts (Schlegel & Sattler 2023) and AutoML systems (X. He et al. 2021).

However, they are unique due to the sheer number of available pre-trained models, something previously only very large organizations would have had access to. This has opened up new opportunities and challenges. One of these is the main topic of this thesis, the challenge of which pre-trained model to select for fine-tuning, which we will expand on in the following sections. There are various other research areas which aim to improve the lifecycle of machine learning using model zoos. For example, by optimizing inference queries under constraints (Ziyu Li et al. 2023) and learning from multiple source models (Q.

| Hub | Models | Datasets |
|---|---|---|
| PyTorch | 52 | 99 |
| Kaggle | 3,183 | 323,228 |
| HuggingFace | 634,589 | 140,255 |

**Table 2.1:** Size of public model zoos, as of May 2024.

[1] https://huggingface.co/
[2] https://www.kaggle.com/models
[3] https://pytorch.org/hub/

Dong et al. 2022; Shu et al. 2021). However, variants of transferability estimation have gained the most scientific interest for model zoos.

Table 2.1 gives an overview of the sizes of popular model zoos. Kaggle used to only host datasets and was known for its competitions, but expanded to host models in collaboration with TensorFlow hub in 2023. Pre-trained model hosting on HuggingFace has been around for longer, and its variety comes mainly due to community datasets and models. PyTorch pioneered reuse of pre-trained models through GitHub repositories. However, it does not offer an easy way to explore these and only hosts a limited number of models and datasets.

Figure 2.2 shows an example of a pre-trained model on HuggingFace, a masked language model called BERT (Devlin et al. 2019). HuggingFace supports sharing details through model cards (Mitchell et al. 2019), which contain information such as pre-training details, intended- and how to use, and limitations and risks. It also links to the datasets used to pre-train the model, which in this case is both the English Wikipedia and a large collection of unpublished books.

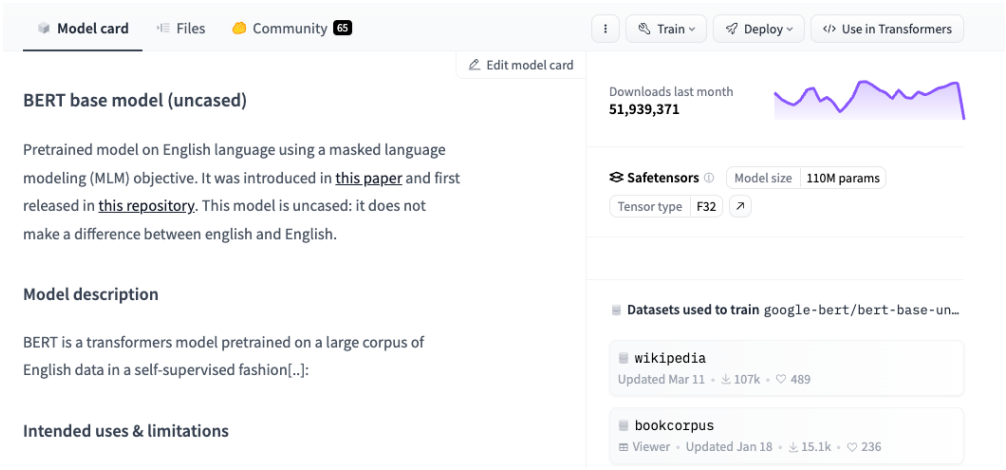



**Figure 2.2:** An example of a pre-trained model on HuggingFace.

## 2.3. Transferability Estimation

Now that we have covered the necessary background information for this thesis, we turn to the actual topic of interest: *Transferability Estimation*. To avoid brute-force fine-tuning all pre-trained models in a model zoo, transferability estimation attempts to offer an efficient heuristic to find the best performing source model for a new target task. Early work on transferability estimation was mainly focused on avoiding *negative transfer* and targeted at simple classifiers, such as hierarchical Naive Bayes (Rosenstein et al. 2005). Negative transfer was said to occur when transferring knowledge from a source task negatively impacted performance on a target task. While some attention was gained before (Pan & Yang 2010), it was not until Z. Wang et al. (2019) first gave a formal definition and performed analysis on when negative transfer occurs for deep neural networks. The limited works before this time mainly focused on task relatedness and clustering to identify such negative transfer (Ben-David & Schuller 2003; Rosenstein et al. 2005; Eaton et al. 2008).

More recently, the terms *model selection*, *model search*, and *transferability estimation* have been used to describe this problem, and the focus mostly shifted to deep neural networks. Bolya et al. (2021) differentiate between model selection and transferability estimation by noting that model selection *usually* attempts to find the best pre-trained model given a fixed target task, while transferability estimation fixes the source pre-trained model and attempts to predict the best target task to transfer to. We no longer see this pattern (Pandy et al. 2022; Shao et al. 2022), and prefer the term transferability estimation, as it is more explicit about the goal of transfer learning.

Initially, most works were in the field of computer vision, likely due to the larger data size and associated computational burden of training models. However, with the recent popularity of language models, especially very large ones (OpenAI et al. 2024; *LLaMA* 2024), transferability estimation has also received notable attention in the field of natural language processing.

Despite great interest in transfer learning and the available surveys (C. Tan et al. 2018; Niu et al. 2020; Zhuang et al. 2021), there have been few comprehensive surveys on transferability estimation, especially covering both computer vision and natural language processing. Bai et al. (2023) provide a fairly comprehensive survey on transferability estimation methods for natural language processing and compare them using the GLUE (A. Wang et al. 2018) benchmark as target tasks. Bassignana et al. (2022) and Agostinelli et al. (2022) thoroughly assess the performance of transferability estimation for natural language processing and computer vision, respectively, but focus on a smaller set of methods.

### 2.3.1. Nomenclature and Notations

Works on transferability estimation often use terminology from the transfer learning community. During the rest of this thesis, we will therefore also use these terms. To improve the reading experience in the remainder of this thesis, Table 2.2 provides an overview of these terms and their meaning.

| **Term** | **Meaning** |
|---|---|
| Task | Refers to a learning problem a model is designed to address. Mostly used synonymously with dataset, except that there often also is an associated objective (such as predicting a label) and an evaluation metric (such as accuracy). |
| Dataset | The data used as input for the learning problem, often in the form of input and label pairs. |
| Pre-training | The step before fine-tuning, often involves training a model from scratch using a large and diverse dataset, which is suitable to fine-tune for a specific task. |
| Upstream, source | Used to refer to the dataset used in the pre-training phase, or the model that results from it. |
| Fine-tuning | Using a pre-trained model to adapt to a new, often smaller task. |
| Downstream, target | Used to refer to the dataset used in the fine-tuning phase, or the model that results from it. |
| Forward/backward pass | Steps in a model training/fine-tuning loop. Forward refers to passing input data forwards through the layers, backwards updating the layers' weights with error information. |

**Table 2.2:** Nomenclature in Transfer Learning

The notations used throughout this thesis are summarized in Table 2.3. These notations follow the standards of works related to transferability estimation and are most similar to those in You et al. (2021).

| **Notation** | **Definition** |
|---|---|
| $S_{m \to t}$ | Predicted transferability score of model $m_i$ on dataset $d_j$ |
| $T_{m \to t}$ | Fine-tuning performance of model $m_i$ on dataset $d_j$ |
| $\phi_i$ | Pre-trained model $i$ |
| $\psi_i$ | Fine-tuned model $i$ |
| $\mathcal{D}$ | Set of datasets |
| $\mathcal{M}$ | Set of pre-trained models |
| $N$ | Total number of datasets |
| $M$ | Total number of pre-trained models |

**Table 2.3:** Transferability estimation notations.

### 2.3.2. Types of Transferability Estimation Methods

In this section, we do an extensive survey and categorization of the related work on transferability estimation, both for natural language processing and computer vision. In our survey, we include 29 works that propose transferability estimation methods. While we largely adopt the survey by Bai et al. (2023), we expand on it and propose a different categorization, primarily based on the input of the approaches. Although more distinctions can be made between the methods, we find that this categorization provides a better idea of the minimum work required for each method. For example, all *source embedding* methods require a forward pass of the target dataset over the source model. Table 2.4 outlines the approaches and desirable properties of transferability estimation metrics. Figure 2.3 presents a categorization of transferability estimation methods. In the following sections, we explain each category and summarize the approach for each of the works in these categories.

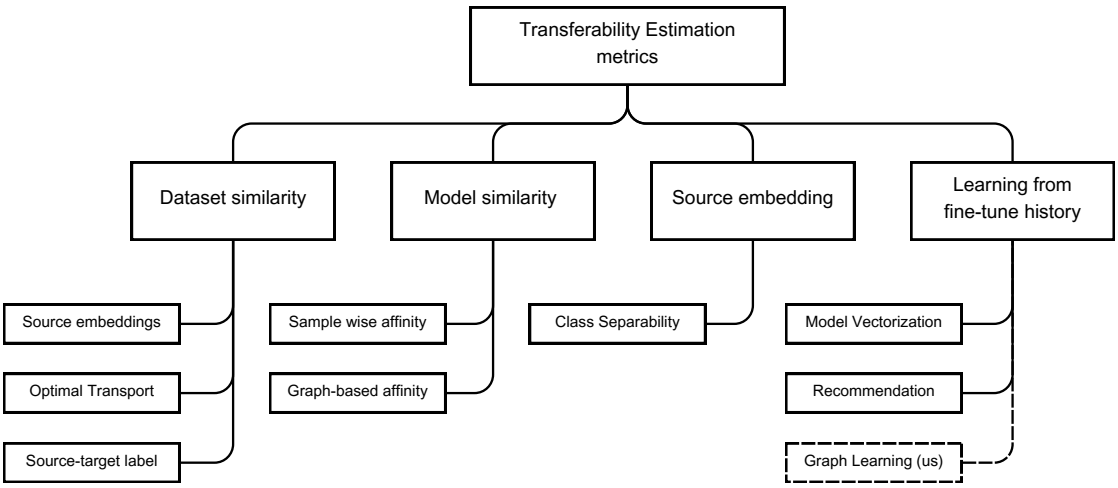



**Figure 2.3:** A taxonomy of surveyed transferability estimation methods.

#### Dataset Similarity with Source Embeddings

Pioneering works in the field of transferability estimation mainly focused on capturing the intuition that the fine-tuning performance should be high when the source and target tasks are similar. Hence, they attempt to embed the tasks in such a way that their distance gives an indication of their similarity. Embeddings are usually generated by running the source dataset $d_s$ and the target dataset $d_t$ through the feature extractor of a large *probe* network, such as BERT (Devlin et al. 2019) for natural language processing tasks and ResNET (Yu et al. 2018) for computer vision tasks. Methods in this category include Task2Vec (Achille et al. 2019) and Domain Similarity (Cui et al. 2018) for computer vision tasks.

#### Dataset Similarity with Optimal Transport

Another set of methods attempts to apply the same procedure as above, but without the need of running the target dataset through a probe network to generate the source embeddings. These methods estimate transferability by solving the Optimal Transport (OT) problem between the source and target distributions. OTCE (Y. Tan et al. 2021) then uses the optimal coupling to compute the Negative Conditional Entropy between the source and target labels. OTDD (Alvarez-Melis & Fusi 2020) propose a hybrid Euclidean-Wasserstein distance over label-feature pairs. Finally, F-OTCE and JC-OTCE (Y. Tan et al. 2024) improve OTCE in terms of efficiency and accuracy, by not requiring auxiliary tasks with known fine-tuning performances.

#### Dataset Similarity with Source-target Label Comparison

Other early transferability methods, such as NCE (Tran et al. 2019) and LEEP (Nguyen et al. 2020), measure the relatedness of the labels of the source and target datasets to construct a score. In NCE, Tran et al. (2019) assume that the samples from the source and target tasks are equal but have different labels. They use Negative Conditional Entropy between the source and target labels as the transferability score. LEEP (Nguyen et al. 2020) is more similar to the methods below;

however, instead of extracting the features, the target labels are extracted only using the source model. These are used to compute the log-likelihood between the source model prediction and the actual target labels.

### Model Similarity

As an improvement over brute force fine-tuning, model similarity-based methods fine-tune one target model $\psi_t$ and attempt to measure the similarity between the target model and the set of source models $\{\phi_m\}_{m=1}^{M}$. The idea is that this reduces the time consumption of fine-tuning to $1/M$, plus a relatively efficient pairwise similarity computation. These methods also usually use sample features extracted from both source and target models to measure similarity. Bai et al. (2023) divide these methods into *Sample-wise Similarity Functions* and *Graph-wise Similarity Functions*.

TaskEMB (Vu et al. 2020) and CogTaskonomy (Luo et al. 2022) use sample-wise similarity functions. They do this by computing the affinity between the mean features and using the average sample affinities of language models, respectively. A-Map (Song et al. 2019) calculate *attribution maps* for computer vision models using various methods, which can also be used to measure the distance between pairs of models.

A second line of model similarity transferability estimation methods use graph-wise similarity functions. Figure 2.4 shows a pipeline of how model similarities are calculated in Graph Based Similarity (GBS) (Z. Chen et al. 2021). Similarly to sample-wise similarity functions, sample images are fed through the pre-trained models' layers to obtain their representations. These are converted to vectors, which are used to represent the pre-trained models as graphs, which are finally used to compute their similarity. Other works in this line include Representation Similarity Analysis (RSA) (Dwivedi & Roig 2019), Duality Diagram Similarity (DDS) (Dwivedi et al. 2020), Kernel Alignment (KA) (J. Huang et al. 2021) and Centered Kernel Alignment (CKA) (Kornblith et al. 2019). These methods differ only slightly in how they compute the graph affinities.

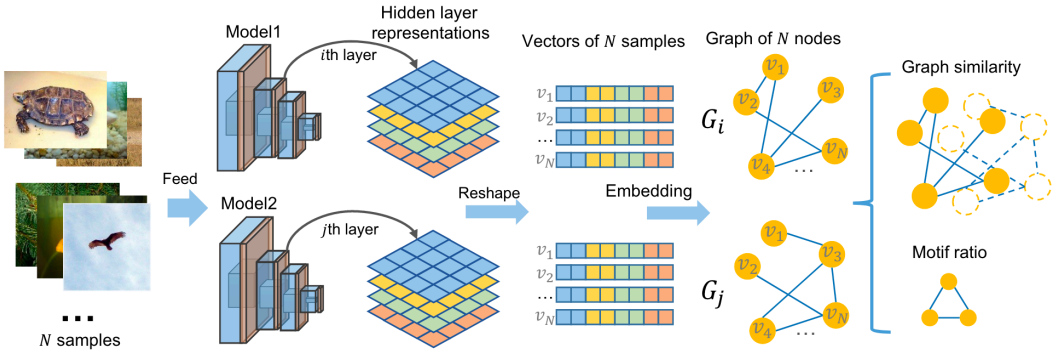



**Figure 2.4:** An example pipeline of computing model similarity from GBS (Z. Chen et al. 2021).

### Source Embedding

Although model similarity methods only need to fine-tune on the target task once, this still comes at a high computational cost. Source embedding methods only use the features extracted from the target task by a source model and the target task labels to efficiently estimate the true performance. Bai et al. (2023) separate these methods into *class separability methods* and *loss approximation methods*. However, the distinction between these categories is not very clear. Most importantly, a majority of the methods in the category of loss approximation do not recognize that this is their strategy. However, the class separability of the extracted features *is* a common factor for these methods.

The assumption of these methods is that if the extracted features by a pre-trained model are similar for the same target class and different from the features of other target classes, the fine-tuning performance of that pre-trained model will be good. In essence, these methods measure

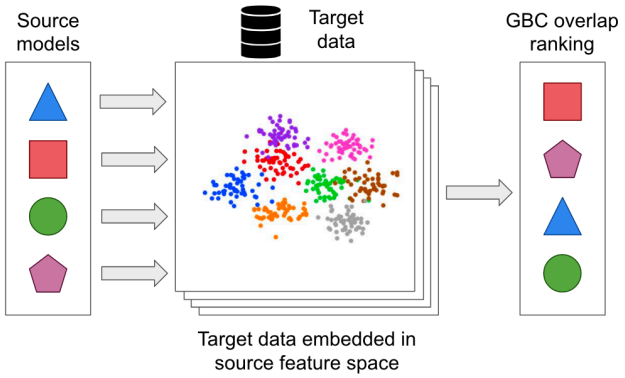



**Figure 2.5:** An illustration from GBC (Pandy et al. 2022) of how source embedding methods generally analyze how well the features extracted by a source model cluster for each target class.

how well the extracted features cluster for each of the target labels. Figure 2.5 illustrates how GBC's (Pandy et al. 2022) approach, but the essence of these methods is the same.

Meiseles & Rokach (2020) directly use the pre-trained features using the Mean Silhouette Coefficient (MSC) to assess the clustering quality of target classes. Bolya et al. (2021) use Pairwise Annotation Representation Comparison (PARC) by computing the pair-wise affinities of all pairs of target samples. Gaussian Bhattacharyya Coefficient (GBC) (Pandy et al. 2022), use the Bhattacharyya coefficient to measure the inter-class overlap of the target features. Puigcerver et al. (2020) fit a KNN classifier and Kumari et al. (2022) a Logistic Regression model on the extracted features to estimate how fitting they are for the target labels. Yandong Li et al. (2021) introduce an extension of LEEP, $\mathcal{N}$LEEP, where they fit a Gaussian Mixture Model of the target data within the embedding space, replacing the source model's classification head to calculate the LEEP score. L.-K. Huang et al. (2022) propose TransRate, which uses a proxy of mutual information (through coding rate) between the extracted features and target labels.

Bao et al. (2019) develop H-Score, which assigns higher transferability scores to embeddings with lower feature redundancy and inter-class overlap. In Ibrahim et al. (2023), the authors propose Reg-Hscore, a shrinkage-based estimation of H-Score, which solves H-Score instability due to poor covariance estimation. The state-of-the-art score, LogME (You et al. 2021), treats each target label as a linear model influenced by Gaussian noise and adjusts the prior distribution parameters to determine the average Logarithm of Maximum Evidence for the labels based on the target sample embeddings. PACTran (Ding et al. 2022) combines ideas from the PAC (Probably Approximately Correct) learning theory with Bayesian statistics to predict the generalizability of the pre-trained model to the target task. SFDA (Shao et al. 2022) (Self-challenging Fisher Discriminant Analysis) measure separability of the target classes through projecting the target embeddings by Fisher Discriminant Analysis.

### Learning from fine-tuning history methods

While performing a forward pass over a model's feature extractor is relatively efficient compared to full fine-tuning, it becomes an increasingly larger burden as the model zoo grows. Furthermore, as mentioned earlier, there is a trend for increasingly larger models, which makes a forward pass even more expensive. By learning from past fine-tuning experiences, some recent works attempt to circumvent this forward pass over the entire model zoo. Y.-K. Zhang et al. (2024) do this by vectorizing models and tasks into a single space and learn from previously obtained fine-tuning performances. H. Li et al. (2023) convert transferability estimation to a recommendation problem. They show that by only incorporating basic meta-features of pre-trained models and tasks, it is possible to train a linear regression model which can accurately predict the performance and have the same benefit in terms of efficiency.

| Method | Input | Free of Target Training | Free of Forward Pass | Learns from Basic Metadata | Learns from Complex Relations |
|---|---|---|---|---|---|
| *Dataset similarity with Source Embedding* | | | | | |
| Task2Vec (Achille et al. 2019) | $\phi_p, d_s, d_t$ | ✗ | ✓ | ✗ | ✗ |
| Domain Similarity (Cui et al. 2018) | $\phi_p, d_s, d_t$ | ✓ | ✓ | ✗ | ✗ |
| *Dataset Similarity with Optimal Transport* | | | | | |
| OTCE (Y. Tan et al. 2021) | $d_s, d_t$ | ✗ | ✓ | ✗ | ✗ |
| (F/JC)-OTCE (Y. Tan et al. 2024) | $d_s, d_t$ | ✗ | ✓ | ✗ | ✗ |
| GDD (Alvarez-Melis & Fusi 2020) | $d_s, d_t$ | ✗ | ✓ | ✗ | ✗ |
| *Dataset similarity with Source-target Label Comparison* | | | | | |
| NCE (Tran et al. 2019) | $d_s, d_t$ | ✗ | ✗ | ✗ | ✗ |
| LEEP (Nguyen et al. 2020) | $d_s, \phi_m, d_t$ | ✗ | ✗ | ✗ | ✗ |
| *Model Similarity* | | | | | |
| DDS (Dwivedi et al. 2020) | $\phi_m, \psi_t$ | ✗ | ✓ | ✗ | ✗ |
| RSA (Dwivedi & Roig 2019) | $\phi_m, \psi_t$ | ✗ | ✓ | ✗ | ✗ |
| GBS (Z. Chen et al. 2021) | $\phi_m, \psi_t$ | ✗ | ✓ | ✗ | ✗ |
| KA (J. Huang et al. 2021) | $\phi_m, \psi_t$ | ✗ | ✓ | ✗ | ✗ |
| CKA (Kornblith et al. 2019) | $\phi_m, \psi_t$ | ✗ | ✓ | ✗ | ✗ |
| TaskEMB (Vu et al. 2020) | $\phi_m, \psi_t$ | ✗ | ✓ | ✗ | ✗ |
| CogTaskonomy (Luo et al. 2022) | $\phi_m, \psi_t$ | ✗ | ✓ | ✗ | ✗ |
| A-Map (Song et al. 2019) | $\phi_m, \psi_t$ | ✗ | ✓ | ✗ | ✗ |
| *Source Embedding* | | | | | |
| LogME (You et al. 2021) | $\phi_m, d_t$ | ✓ | ✗ | ✗ | ✗ |
| MSC (Meiseles & Rokach 2020) | $\phi_m, d_t$ | ✓ | ✗ | ✗ | ✗ |
| PARC (Bolya et al. 2021) | $\phi_m, d_t$ | ✓ | ✗ | ✗ | ✗ |
| kNN (Puigcerver et al. 2020) | $\phi_m, d_t$ | ✓ | ✗ | ✗ | ✗ |
| GBC (Pandy et al. 2022) | $\phi_m, d_t$ | ✓ | ✗ | ✗ | ✗ |
| Logistic (Kumari et al. 2022) | $\phi_m, d_t$ | ✓ | ✗ | ✗ | ✗ |
| PACTran (Ding et al. 2022) | $\phi_m, d_t$ | ✓ | ✗ | ✗ | ✗ |
| TransRate (L.-K. Huang et al. 2022) | $\phi_m, d_t$ | ✓ | ✗ | ✗ | ✗ |
| $\mathcal{N}$LEEP (Yandong Li et al. 2021) | $\phi_m, d_t$ | ✓ | ✗ | ✗ | ✗ |
| H-Score (Bao et al. 2019) | $\phi_m, d_t$ | ✓ | ✗ | ✗ | ✗ |
| Reg. H-score (Ibrahim et al. 2023) | $\phi_m, d_t$ | ✓ | ✗ | ✗ | ✗ |
| SFDA (Shao et al. 2022) | $\phi_m, d_t$ | ✓ | ✗ | ✗ | ✗ |
| *Learning from Fine-tuning History* | | | | | |
| Model Spider (Y.-K. Zhang et al. 2024) | $\phi_m, d_t, S_{m\to t}$ | ✓ | ✓ | ✗ | ✗ |
| Amazon LR (H. Li et al. 2023) | $\phi_m, d_s, d_t, T_{m\to t}$ | ✓ | ✓ | ✓ | ✗ |
| TransferGraph | $\phi_m, d_s, d_t, T_{m\to t}$ | ✓ | ✓ | ✓ | ✓ |

**Table 2.4:** Overview of surveyed transferability methods, where $\phi_p$ and $\phi_p$ denote probe- and source model. $\psi_t$ denotes the fine-tuned target model. $d_s$ and $d_t$ represent the source- and target tasks.

### 2.3.3. Other related work

Below, we will cover other relevant related work which can be useful to understand the context of transferability estimation. These topics are often seen in works on transferability estimation, but differ in their ultimate goal.

#### Transfer learning engine

An omitted work in the above survey is SHiFT (Renggli et al. 2022). It is interesting because it proposes the first *transfer learning engine*. It does not propose its own transferability estimation metric. Instead, practitioners can approach this system with their target task and computational budget, and the system will attempt to select the best source model for transfer learning. It combines a task-aware and task-agnostic approach and, depending on the budget, it will fine-tune the best $k$ returned models by the various approaches.

#### Target task selection

Some works on transferability estimation cover two evaluation scenarios: *1*) *Source model selection*, that is, 'For my target task, which pre-trained model will yield the best fine-tuning performance"; and *2*) *Target task selection*, i.e. "For my pre-trained model, which target task will yield

the best fine-tuning performance" (Bolya et al. 2021; Ibrahim et al. 2023). However, most works only cover source model selection. We will do the same in this thesis, as it fits with the use cases we described earlier. Additionally, the use case of target task selection is not explicitly made clear in these works.

#### Model Ensemble

Another interesting line of work related to transferability estimation is *model ensemble*, or *model soups* (Wortsman et al. 2022). This field focuses on combining weights from layers of different pre-trained models to adapt to a new target task. A different, but overlapping, setting to transferability estimation is selecting the best pre-trained models to combine these weights for. B et al. (2023) apply transferability estimation in this setting, and improve GAN training for large image generation models.

## 2.4. Graph Learning

As mentioned in the Introduction, since our solution to transferability estimation utilizes graph learning to capture relationships between pre-trained models and datasets, we will cover the required background knowledge on graph learning in this section. Graph learning refers to machine learning applied to data structured as graphs. Many complex real-world scenarios, such as social networks (e.g. LinkedIn), biological networks (proteins), chemical networks (molecules), and traffic networks, can be modeled as graphs. Various tasks can then be solved using these graphs; this can be:

- On a **(sub)graph level**, for example *graph generation*, which can be used in drug discovery to find new types of molecules (Ilnicka & Schneider 2023).
- On a **node level**, for example *node property prediction*, to predict interests of users in a social network (Nori et al. 2011).
- On an **edge level**, for example *edge (property) prediction*, with the goal of identifying real world friends (Grover & Leskovec 2016) or adverse side effects of pairs of drugs (Zitnik et al. 2018).

**Definition 2.4.1** (Graph)**.** We denote a graph as $G = (V, E)$ where $V$ is the set of vertices/nodes and $E \subseteq V \times V$ denotes the set of edges that connect the vertices in $V$.

### 2.4.1. How to Represent a Graph

Graphs are very different from typical data structures used for machine learning, such as images, text, or tabular data. These are all relatively straightforward to represent as vectors of features, which is the typical input format for downstream machine learning tasks. To illustrate; intuitively, it might be tempting to represent a graph as an adjacency matrix, indicating which node is directly connected to another. However, such a representation consumes a lot of memory, as its size is $|V| \times |V|$ and only captures neighboring relationships, disregarding more complex structural information about the graph (D. Zhang et al. 2020).

A common approach therefore is to first find meaningful representations or *embeddings* for the subject of interest (subgraph, node, or edge, depending on the task type), and then use these representations as input for a downstream learning task (Xia et al. 2021). Graph representation learning is an active research area, with various available surveys (F. Chen et al. 2020; Khoshraftar & An 2024).

In a recent survey, Khoshraftar & An (2024) categorize graph embedding methods into *traditional* and *Graph Neural Network*(*GNN*)-based approaches. We will briefly cover both types and their advantages below. Their survey also covers *dynamic* graph embedding methods, which aim to additionally embed the evolution of a graph over time. However, this is not relevant to our problem setting and we will omit these methods. Below, we will introduce these methods and their strengths and weaknesses.

### 2.4.2. Traditional Graph Embedding Methods

Traditional graph embedding techniques maintain various properties of nodes and edges in graphs, such as the proximity of nodes. These methods are further divided into three different categories by Khoshraftar & An (2024): *factorization-based methods*, *random walk-based methods* and *non-GNN-based deep learning methods*.

### 2.4.3. Factorization-based

Early works on graph representation learning are primarily based on factorization techniques. The starting point for factorization-based methods is a proximity matrix, where each element is denoted $P_{i,j}$ and represents the proximity of a node $i$ to $j$. Next, factorization techniques decompose this proximity matrix, where the resulting matrices represent lower-dimensional embeddings of the nodes. Methods mainly vary in how they define their proximity matrix. Although matrix factorization-based methods are effective in capturing the global graph structure, scalability is a bottleneck due to the memory consumption of factorizing very large matrices (D. Zhang et al. 2020).

### 2.4.4. Random Walk-based

Random walk-based graph embedding methods are used to learn representations of nodes in a graph in a low-dimensional space while preserving the graph's structure. In this context, *random walk* refers to the random movement of one node to another along the edges of the graph, which helps to explore the structure of the graph and the similarities between nodes. After the random walks have been performed, they are used to train a SkipGram (Mikolov et al. 2013a) model to generate the embedded node vectors.

**Definition 2.4.2** (Random walk)**.** A random walk on a graph $G = (V, E)$, is defined as a sequence of nodes $v_0, v_1, .., .., v_k$, where $(v_i, v_{i+1}) \in E$ and $k + 1$ is the length of the walk. From each node $v_i$, the next node $v_{i+1}$ in the sequence is chosen based on a probabilistic distribution.

Popular examples are DeepWalk (Perozzi et al. 2014), Node2Vec (Grover & Leskovec 2016) and its extension Node2Vec+ (R. Liu et al. 2023). The latter two graph embedding learning methods are based on Word2Vec's (Mikolov et al. 2013b) observation that words occurring in the same sentence have high similarity. These methods apply this in graphs, noting that nodes occurring in the same random walk have a high degree of similarity.

Although these methods scale better than factorization-based methods, they do have a few disadvantages. These methods are *transductive* instead of *inductive*, meaning that embeddings have to be generated from scratch every time a new node is added (Hamilton et al. 2017). This is in contrast to GNN-based graph embedding methods, which train a model which can be used to generalize over unseen nodes. Furthermore, these methods do not easily incorporate edge or node properties.

### 2.4.5. GNN-based Graph Embedding

GNNs are deep learning models that generate node embeddings by aggregating the embeddings of the node's neighbors (Khoshraftar & An 2024). This process is often referred to as message passing. During this process, nodes exchange information with their immediate neighbors, iteratively updating their states based on both their own attributes and the information received. By doing this, GNNs can capture local and global structures within a graph, allowing them to make predictions about nodes, edges, or entire graphs.

Since their first introduction, GNNs have seen a variety of enhancements and architectures, with varying message passing implementations. Key variants include Recurrent Graph Neural Networks (*RecGNNs*) (Scarselli et al. 2009), which are based on RNNs; Convolutional Graph Neural Networks (*CGNNs*) (Kipf & Welling 2016), which generalize convolutional neural networks to graph data; and Graph Attention Networks (*GATs*) (Veličković et al. 2018), which introduce attention mechanisms to weigh the importance of neighboring nodes differently.

Their key difference from traditional methods is that they can make predictions for unseen nodes and their ability to incorporate edge and node properties. However, their computational

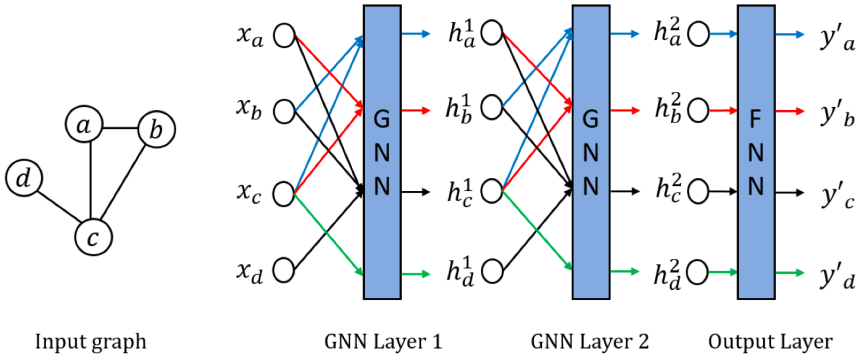



**Figure 2.6:** Image of a general framework for training GNNs taken from Khoshraftar & An (2024). It depicts an input graph with four nodes, two GNN layers and a classification layer. $x_a$ is the feature representation of node $a$ and $h_a^1$ and $h_a^2$ the feature representations of node $a$ after passing through the first and second layer. The colors represent the common neighbors of each of the nodes.

complexity is usually greater than for random walk-based methods (Khoshraftar & An 2024). Below, we will first cover the general architecture of GNNs, followed by a detailed description of the GNN methods which we use later in our experiments.

### Basic Architecture

GNNs generate node representation vectors by stacking multiple GNN layers. A basic architecture of how node representations are updated by a GNN's layer is by the following formula:

$$h_i^{(l+1)} = f\left(h_i^l, \sum_{j \in N(i)} g(i,j)\right), \tag{2.1}$$

where $h_i^l$ is the node embedding for node $i$ at layer $l$, $f$ and $g$ are learnable functions and $N(i)$ the set of neighbors of node $i$. The embeddings of node $i$ at each layer are generated by aggregating the embeddings of the node's neighbors. $h_i^0$ are the initial features of the node and $h_i^L$ is the final representation after passing through $L$ layers.

GNNs are typically trained using a cross-entropy loss function. In case of a node classification task, this can be done as follows:

$$L = \sum_{i \in V_{\text{train}}} y_i \log(\sigma(h_i^T \theta)) + (1 - y_i) \log(1 - \sigma(h_i^T \theta)), \tag{2.2}$$

where $h_i$ is the embedding of node $i$, $y_i$ is the true class label and $\theta$ the classification weights. This loss function can be defined similarly in the case of an edge prediction task using pairwise node representations. Figure 2.6 shows a general framework for training GNNs for node classification tasks.

## 2.5. Summary

In this chapter, we introduced the concept of transferability estimation and the concepts most relevant to it: transfer learning and model zoos. The field of transferability estimation aims to give an efficient heuristic to find the best pre-trained model to fine-tune for a target task, among a set of pre-trained models which can be found in public repositories, often called model zoos. Fine-tuning is a transfer learning technique that retrains the weights of one or more layers of a neural network to adapt it to a new task. We further introduce the concept of graph learning and the available methods.

Although in recent years many transferability estimation methods have been proposed, no comprehensive survey of these methods has been provided. The available surveys are mostly benchmarks of a few methods and do not offer an easy-to-understand categorization of these methods. In this chapter, we provided a categorization based on the input of these methods, which we argue to be more intuitive. We categorized a total of 29 proposed transferability estimation methods and described their approach to solving this problem.

**Summary of contributions**

- A survey on the related work on transferability estimation, which is more extensive than existing surveys, such as those by Bai et al. (2023) and Agostinelli et al. (2022), and offers a more intuitive taxonomy by categorizing methods based on the used input.

# 3

# Transferability Estimation as a Graph Learning Problem

In Chapter 2, we introduced the topics related to this thesis and provided a brief background on transferability estimation, together with a review of the related work. In this chapter, we further explore the concept of transferability estimation. First, we will formally describe the transferability estimation problem and explain three envisioned use cases. Next, we discuss the limitations and challenges of the methods introduced in Chapter 2. These limitations are then used to introduce our solution for estimating transferability. We propose reframing the transferability estimation problem as a link prediction problem on a graph. The missing link represents the fine-tuning performance of a pre-trained model on a new target dataset. This chapter lays out our motivation for this choice and the details of the information we chose to include in the graph.

## 3.1. Problem Definition

To fully understand transferability estimation, we will first explain the problem in more detail. To do so, we first explain the problem through three use cases. After this, we formally describe the problem and explain how transferability estimation methods generally measure their success. Finally, we discuss the limitations of the currently proposed transferability estimation methods.

### 3.1.1. Example Use Cases

To illustrate the problem of transferability estimation and why it is important, we will first provide three example use cases. We will start with a more trivial use case, building up to how we envision transferability estimation as an important step in the machine learning lifecycle across different end-users.

**Use case 1: An individual machine learning practitioner** is interested in building a niche tool, which can take an image and tell whether it contains a Chihuahua or a blueberry muffin. They do not have the resources, both in terms of number of images and computational resources, to train a classifier for this problem from scratch (surprisingly, no one would invest in their tool). The practitioner consults a public model zoo with their limited dataset of images and labels and tries to find a pre-trained image classification model to fine-tune for their task. Currently, this model zoo will give them no guidance on which pre-trained model to pick. They might intuitively pick a pre-trained image classification model which was trained to differentiate between breeds of dogs. Unfortunately, this did not give them satisfying results either, and they wasted the little funds they had to run the fine-tuning on a GPU instance from a cloud provider.

**Use case 2: A medium-sized financial organization,** such as a bank, is required to monitor their transactions for suspicious behavior. At the very least, they should be wary when transaction description somehow have terroristic sentiment in them, mentioning for example guns or religious extremism. Simple heuristics or lists of words generate too many false positives, so they want to explore deploying a text classification model. They have a small dataset of text and label pairs, but from experience know that this is too little to train a text classifier from scratch with. They do not have any pre-trained models which they could fine-tune themselves, so they turn to a public model zoo to search for a pre-trained model. Not able to find any models which were already fine-tuned for recognizing religious extremism in transaction descriptions, they need to choose between any of the thousands of pre-trained models the zoo offers. They randomly pick five popular pre-trained models and fine-tune them. While some of them had promising results, they were extremely slow and not suitable to execute real-time as transactions are being executed.

**Use case 3: A large international tech company** does have the resources to train their own proprietary pre-trained models. They build a private model hub for internal usage, containing both pre-trained models and fine-tuned models for specific tasks. Other teams within the organization are interested in solving their tasks using machine learning, but given the growing size of the model zoo, are unsure on which to best pick for their tasks.

### 3.1.2. Problem Formalization

Formally, the goal of transferability estimation is to estimate a score $S_{m\to t}$ for a source model $\phi_m$ and a target task $d_t$. The target task is a dataset containing $n$ image or text samples and ground truth label pairs $\{(x_i, y_i)\}_{i=1}^{n}$. The dataset has an evaluation metric, such as accuracy or mean squared error, to measure the ground-truth fine-tuning performance $T_{m\to t}$. A good transferability score $S_{m\to t}$ has a high correlation with the ground truth fine-tuning performance $T_{m\to t}$.

### 3.1.3. How to Measure the Success of Transferability Estimation Methods?

For a set of $M$ pre-trained source models $\{\phi_m\}_{m=1}^{M}$, a perfect transferability metric would produce the scores $\{S_{m\to t}\}_{m=1}^{M}$ in exactly the same order as the ground-truth fine-tuning performances $\{T_{m\to t}\}_{m=1}^{M}$. Measures such as top-1 or top-k could be used to measure deviations from this perfect order, but these fail to take into account that the top performing models might perform very similar and punish the metric too hard for not picking the best model. Thus, most transferability estimation methods turn to rank correlation (Fagin et al. 2003). Below, we give an overview of the commonly used correlation metrics in works on transferability estimation.

**Pearson's correlation coefficient** (Pearson 1895) ($r$) is a measure of the linear correlation between two sets of data. It gives information about correlation, as well as the direction of the relationship between the datasets. The values range between $r \in [-1, 1]$, where $-1$ indicates perfect negative linear relationship, $0$ no linear relationship and $1$ perfect positive linear relationship. It is used by Nguyen et al. (2020), but the absence of a linear relationship between the predicted scores $\{S_{m\to t}\}_{m=1}^{M}$ and the ground truth fine-tuning performances $\{T_{m\to t}\}_{m=1}^{M}$ do not mean that the transferability metric is bad. On the contrary, the ranking could be perfect while the relationship is non-linear.

**Spearman's rank correlation coefficient** (Spearman 1904) ($\rho$) - to solve the above issue, most methods turn to rank correlations to measure the success of their method. Spearman's rank correlation coefficient between two variables is equal to the Pearson correlation between the rank values of these variables.

**Kendall rank correlation coefficient** (Kendall 1938) ($\tau$) - enumerates all rank pairs and subtracts all discordant pairs from all concordant pairs. The equation to calculate $\tau$ is:

$$\tau = \frac{2}{M(M-1)} \sum_{1 \le l < m \le M} \text{sgn}(T_l - T_m) \cdot \text{sgn}(S_l - S_m) \tag{3.1}$$

Both $\rho$ and $\tau$ have values ranging from $[-1, 1]$, where 1 indicates perfect agreement of the ranks and $-1$ perfect inverse agreement. You et al. (2021) argue that it has an easy interpretation, in contrast to the coefficients mentioned above. That is, if $S$ and $T$ have a correlation of $\tau$, the probability that $T_i > T_J$ is $\frac{\tau+1}{2}$ when $S_i > S_j$.

**Weighted Kendall rank correlation coefficient** ($\tau_w$) - in reality, practitioners will mostly care for the correct ranking of the top performing models. This can be incorporated into Kendall rank correlation by assigning a higher weight on pairs with higher $T_i$. Agostinelli et al. (2022) use a hyperbolic dropoff in importance of models as $T_i$ decreases.

**Relative top-k accuracy** (Yandong Li et al. 2021) ($Rel@k$) - both $\tau$ and especially $\tau_w$ will overly penalize a transferability metric if the top-performing models are predicted out of order, but perform very similarly. To account for the relative performance of the top-k predicted models, Yandong Li et al. (2021) propose $Rel@k$, which can be calculated by taking the ratio of the best predicted $\{\phi_n\}_{n=1}^{k}$ models according to $\{S_{n\to t}\}_{n=1}^{k}$ and the best fine-tuning accuracy of all models:

$$Rel@k = \frac{\max\{T_{n\to t}\}_{n=1}^{k}}{\max\{T_{m\to t}\}_{m=1}^{M}} \tag{3.2}$$

and they compare their results for $k = 1$ and $k = 3$.

### 3.1.4. Limitations and challenges

In this section, we discuss the limitations of the methods described in Section 2.3.2. Table 2.4 gives an overview of all surveyed methods and desirable properties of transferability estimation methods. Below, we will go over the categories of transferability estimation methods and discuss their strengths and weaknesses.

To prevent brute-force fine-tuning of all pre-trained models, **model similarity** methods generally fine-tune a small target model to compare it to multiple source models. While this requires expensive training, after having trained one target model, the methods usually have an efficient heuristic to compute the score. Strictly speaking, they do require a forward pass, but with the column *free of forward pass*, we mean that it does not require a forward pass for every pre-trained model. **Dataset similarity** methods using **source embeddings** also prevent having to do a forward pass over all pre-trained models. However, by only using a probe model to extract the features, they do not take into account any information about the source model.

**Dataset similarity** methods using **optimal transport** are free of training and a forward pass. However, computing optimal transport distances scales quadratically with the number of samples in the datasets, making it infeasible for larger datasets (Pandy et al. 2022).

While **source embedding** methods are the most popular and competitive methods in terms of effectiveness, when using model zoos consisting of thousands of pre-trained models, they are the most impractical in terms of efficiency. As also noted by Y.-K. Zhang et al. (2024), they require a forward pass of the entire target dataset over all pre-trained models in the model zoo to compute their ranking. Taking either a subset of the pre-trained models or the target dataset samples to estimate transferability will negatively affect the performance of these methods, as shown by Agostinelli et al. (2022).

A major limitation in most related works on transferability estimation is the limited size and diversity of assessed pre-trained models and datasets. For example, the state-of-the-art LogME method (You et al. 2021), was evaluated on only 10 pre-trained models, all trained using ImageNet (Deng et al. 2009). Comparative studies have the same limitation. For example, Agostinelli et al. (2022) and Bai et al. (2023) experiment with only 16 and 6 source models, respectively. Only a few related works, such as those by Renggli et al. (2022) and, more recently, H. Li et al. (2023) and Y.-K. Zhang et al. (2024) use settings with more heterogeneous and larger model zoos, which better represent modern-day model zoos. H. Li et al. (2023) specifically show that in the case of model zoos containing models with different architectures and pre-trained on different source datasets, source embedding methods become more inaccurate.

With a limited number of pre-trained models, performing this forward pass is not a major concern. However, as Y.-K. Zhang et al. (2024) argue, with the increasing number of pre-trained models in model zoos, this approach is becoming infeasible. To address these challenges, some existing methods **learn from fine-tuning history**, such as Model Spider (Y.-K. Zhang et al. 2024) and Amazon LR (H. Li et al. 2023). The work on Model Spider is promising and was proposed simultaneously with this thesis. Therefore, we could not directly compare it to our method. However, it has some apparent disadvantages. It does not incorporate any additional metadata of pre-trained models or target datasets, which H. Li et al. (2023) have shown to be useful. Another shortcoming is their evaluation setup, where they use a combined ranking of other methods as a ground-truth score instead of actual fine-tuning performances. While this may serve as a decent and more efficient proxy, there is no way to ensure the actual performance of their method, as it may rely on the shortcomings of the methods used for ranking. To summarize, we identify the following limitations and challenges:

- Transferability estimation methods are often evaluated against small model zoos with little variation of pre-trained models.
- Only taking into account a part of the available information, they fail to capture complex dynamics of pre-trained models and datasets.
- With a growing number of pre-trained models to choose from, most non-learning based transferability estimation become impractically inefficient.

## 3.2. Solution Overview

In this section, we will present our solution to overcome these challenges and limitations as mentioned in the previous section. We will first motivate our choice for representing the transferability problem as a graph learning problem and discuss the usable relationships in this setting. Next, we explain useful metadata to include as node attributes in our graph as identified by previous works. Finally, we give a formal definition of reframing transferability estimation as a link prediction problem.

### 3.2.1. Motivation for Graph Representation

In Chapter 2, we have discussed the various approaches proposed in existing works to capture the relationships between pre-trained models and target tasks. To summarize, we identify four types of relationships used in these metrics:

Type 1: Source- and unseen target task (e.g. Task2Vec (Achille et al. 2019)).

Type 2: Source model and unseen target task, through extracting the source model features for the target task (e.g. LogME (You et al. 2021)).

Type 3: Source- and target model (e.g. A-Map (Song et al. 2019)).

Type 4: Source model and seen target task, through historic fine-tuning performances (e.g. Amazon LR (H. Li et al. 2023)).

Using these individual relationships, all of these works have had some success in predicting task transferability. The advantages of types 1 and 2 are that they do not require any metadata of the models and datasets, training, or past fine-tuning performances. However, as model zoos grow and become more structured, both metadata and past fine-tuning performances are becoming more readily available – which have been shown to be useful by works of type 4. The disadvantage of type 4 metrics is that they disregard any fine-tuning dynamics that could be exposed by the underlying structure of the datasets and models. Finally, the relationship of type 3 has the disadvantage that it requires to first fine-tune a model on the target task, which is expensive and what we are trying to prevent.

In this work, we advance beyond these metrics by combining multiple types of relationships as weighted connections between tasks and models, and reframe transferability estimation as a graph link prediction problem. We choose this approach, as graph learning can capture more complex relationships than traditional machine learning methods and can be very effective for solving recommendation tasks (Shoujin Wang et al. 2021). Additionally, it is more flexible than

existing approaches, since the general graph structure can still be learned even when certain nodes (e.g., source dataset) or edges (e.g., fine-tuning performances) are unavailable.

Representing this problem as a graph is inspired by other works on data management systems, such as data lakes (Hai et al. 2023), where datasets are structured as graphs (Castro Fernandez et al. 2018; Nargesian et al. 2020; Y. Zhang & Ives 2020). In these works, tables can be presented as nodes and edges as relationships, such as semantic similarity (Castro Fernandez et al. 2018). We also note that we are not the first to consider graph representations to tackle transferability estimation (Z. Chen et al. 2021). However, these works are different as they only represent model similarity relationships in a graph.

In our setting, we will include type 1 and type 4 relationships. In our previous work (Ziyu Li et al. 2024), we also explore the addition of type 2 relationships. While it is shown in that work that transferability scores can be used in our graph as a proxy for the historical fine-tuning performances, we give more focus to efficiency in this work. As noted earlier, obtaining these transferability scores through a forward pass over the entire model zoo adds too much overhead in today's large model zoos. Type 3 requires full fine-tuning of a target model, which we consider to be too inefficient.

### 3.2.2. Basic Metadata

Even though they are coarse-grained, some basic metadata of models and datasets can serve as useful information to predict transferability (H. Li et al. 2023). In this section, we will list the metadata we identified from related works and why they are valuable to use for transferability estimation.

#### Dataset metadata

The metadata of datasets can be indicators of the fine-tuning difficulty, which can affect a model's ability to transfer knowledge to the target task. For example, a dataset with many classes is more difficult to learn than a dataset with binary classes. In our setting, we consider the metadata of the source and target datasets.

- **Number of data samples** Smaller tasks have less information and are likely easier to learn. Similarly, larger tasks with diverse features will likely need a more complex model to achieve good performances.
- **Number of classes** Deep learning models will have more trouble differentiating between classes accurately when there are many instead of when there are for example only two. This also means the task may require more samples to learn properly.

#### Model Metadata

A pre-trained model's metadata can reveal its learning capabilities. A model with more parameters may capture more generalized features, and models with different architectures may have varying inductive biases for different tasks. Below, we present the useful metadata of pre-trained models.

1. **Input shape** Higher-resolution images contain more information, which the pre-trained model has to be able to capture.
2. **Architecture** A pre-trained model's architecture can have a significant effect in the fine-tuning performance. More complex architectures, such as ResNet (K. He et al. 2016) and Inception (Dosovitskiy et al. 2021) may be better for learning more difficult datasets, with larger inputs than simpler architectures, such as LeNet (Lecun et al. 1998).
3. **Pre-trained dataset** The quality and diversity of the dataset used to pre-train the model will have significant impact on the ability to generalize for a new target dataset. If the model was pre-trained with a larger dataset, containing diverse images, it is likely to transfer better to a new task compared to a small and specific dataset.
4. **Upstream performance** The capability of a pre-trained model can be identified by its upstream performance. Given two models pre-trained on the same dataset, the one with

higher upstream accuracy indicates better knowledge of that dataset and could give an indication of better performance on a similar target dataset.

5. **Number of parameters** Similarly to architecture, this can be an indication of complexity of the model and ability to generalize over new target tasks.
6. **Memory consumption** Highly correlating with number of parameters, it is another measure of complexity.

Most of these basic metadata are inspired by Amazon LR (H. Li et al. 2023). However, we exclude their dataset difficulty category because it requires additional computation. All other metadata mentioned above are straightforward to obtain. We expand their set of basic metadata with upstream performance, which is often available in public model zoos.

### 3.2.3. Transferability Estimation as Graph Link Prediction

To effectively predict transferability, we propose to reframe the problem to a graph link prediction problem. In this section, we will motivate this choice and introduce our approach.

#### Link Prediction Problem Formalization

In Section 2.4, we provided a background on graph learning, and in Section 3.1.2, we formally introduced the problem of transferability estimation. In this section, we will introduce the problem reframed as a graph link prediction problem.

In our setting, a node is either a dataset $\mathcal{D} = \{d_i\}_{i=1}^{N}$ or a deep learning model $\mathcal{M} = \{\phi_m\}_{m=1}^{M}$. Hence, for our set of vertices, we have $V = \mathcal{D} \cup \mathcal{M}$, with the number of datasets $N = |\mathcal{D}|$ and the number of models $M = |\mathcal{M}|$. We further construct two types of edges:

1. Dataset-dataset $(d_i, d_j)$, representing the similarity between two datasets.
2. Model-dataset $(\phi_m, d_t)$, representing the historical fine-tuning or upstream training performances $T_{m \to t}$.

**Link prediction** Link prediction aims to predict a missing edge (i.e., link) between two nodes. Most techniques approach it as a ranking problem, where pairs of unconnected nodes are given a probability proportional to the likelihood of the existence of a link between them (Martínez et al. 2017). A threshold is then defined; pairs of nodes with a score above the threshold are assigned a positive edge and below it a negative edge.

We extend Definition 2.4.1 by including a weight matrix $W$ in $G = (V, E, W)$, which describes the adjacency of the graph. Here, the weight value $w$ of an edge $(i, j) \in E$ is assigned a value of $W_{i,j} = w$ and $W_{i,j} = 0$ otherwise. In our problem setting, these weight values correspond to the actual fine-tuning performances or dataset similarities. The implementation of this will be discussed in more detail in Section 4.3. After constructing the graph, we reformulate the transferability estimation problem to a learned function on the graph, which can be denoted by the following equation:

$$T_{m \to t} = F(f_G(\phi_m), f_G(d_t)) \tag{3.3}$$

Where $f_G$ denotes the function learned by various graph embedding methods to learn the existence of an edge in $W$. $f_G(\phi_m)$ takes the pre-trained model node $m$ and outputs the embedding of that node, and $f_G(d_t)$ takes the target dataset node $t$ and returns its embedding. The function $F$ represents the final learned function to predict the actual fine-tuning performance of the source model $\phi_m$ for the target dataset $d_t$.

## 3.3. Summary

In this chapter, we delved deeper into the concept of transferability estimation. By presenting example use cases in Section 3.1.1 and formalizing the problem in Section 3.1.2, we aimed to provide a clearer understanding of the issue. Next, we discussed the limitations and challenges in

Section 3.1.4. We noted that transferability methods are often evaluated only against a few pre-trained models, with little variation in the source datasets and architectures. Moreover, they fail to capture the complex relationships between pre-trained models and datasets by only incorporating a part of the available information, and most methods are impractically inefficient on large model zoos.

To overcome these challenges, we proposed reframing transferability estimation as a link prediction problem on a graph, where pre-trained models and datasets form nodes, and the edges represent either past fine-tuning performances or dataset similarities. Previous research has shown that these relationships are valuable, and by representing them in a graph, we hypothesize that we can capture them more effectively. We also demonstrated how additional metadata can be incorporated as node attributes in this graph, which have been shown to be useful predictors of a model's transferability to a new task.

**Summary of contributions**

- We identify the limitations and challenges of previously proposed methods. Most notably, they disregard useful metadata in their estimation and are impractically inefficient for large model zoos.
- A reformulation of the transferability estimation problem to a graph link prediction problem.

# 4

# System Design: TransferGraph

This chapter will outline the design of our solution to transferability estimation: *TransferGraph*. TransferGraph is an end-to-end framework for estimating transferability using graph learning. It includes the four stages of metadata collection (Section 4.2), graph construction and learning (Section 4.3), regression learning (Section 4.4), and transferability estimation (Section 4.5). Where in Chapter 2, we gave a general background on graph learning, in this chapter we will explain the chosen graph learning methods in more detail.

## 4.1. TransferGraph Overview

In this section, we will introduce the framework for our solution to transferability estimation: *TransferGraph*. To apply our graph learning-based approach, there are a few steps involved, which we will carefully explain in this section. This section will be limited to explaining the framework and motivation behind our solution; for implementation details, we refer to Chapter 5.

A schematic overview of the approach can be found in Figure 4.1. The following sections will cover these stages in detail; Section 4.2 and Section 4.3 will cover the stages of metadata collection and graph learning, respectively. In Section 4.4, we explain the learning of a supervised model, which we use in Section 4.5 to perform the actual transferability estimate.

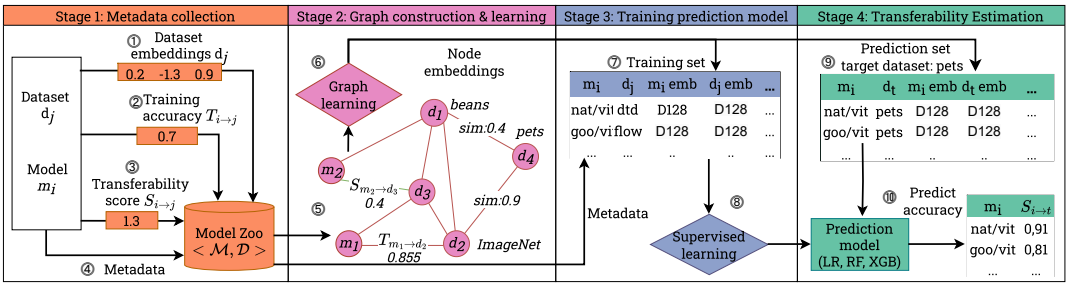



**Figure 4.1:** An overview of TransferGraph for transferability estimation, including model zoo construction (stage 1), training (stage 2-3) and transferability estimation (stage 4).

## 4.2. Stage 1: Metadata Collection

The first stage (steps ①-④) of TransferGraph is the collection of metadata, which will be used in later stages. This stage effectively constructs a model zoo from scratch – a process which we advocate should be done incrementally as models and datasets get added to a model zoo. While some of the information we use is consistently available in public model zoos, a majority needs to be prepared by our framework. Some works propose a more structured storage of model zoo

metadata (Li et al. 2022). This work can be seen as an addition to the metadata that could be used to facilitate efficient and effective transferability estimation.

We start with a set of pre-trained models $\{\phi_m\}_{m=1}^{M}$, and, if available, their source datasets used to pre-train them $\{d_m\}_{m=1}^{M}$. It may occur that the chosen model was already fine-tuned for a different task, in which case we will take the latest fine-tuning dataset as the source dataset. Next, we select a set of target datasets $\{d_t\}_{t=1}^{N}$. This will serve as our model zoo, with the following sections detailing how we gather additional metadata required in our framework.

### 4.2.1. Dataset Embeddings

In step ①, we capture the dataset embeddings. These are used in two places: first, they are used in step ⑤ to construct the edges between the datasets in our graph. Later in step ⑦, they are also used as features when learning the supervised prediction model. Representing datasets as vectors to capture semantic similarity has been proposed in works such as Domain Similarity (Cui et al. 2018) and Task2Vec (Achille et al. 2019). In our previous work, we evaluated both of these methods. However, Task2Vec requires partial fine-tuning and produces vectors of much higher dimension than Domain Similarity. Furthermore, Task2Vec did not show significant advantages. Hence, in this work, we will limit the experiments to Domain Similarity embeddings.

### 4.2.2. Training Performances

One of the key differences from our approach to most works in transferability estimation is that we learn from past fine-tuning performances to predict model performance on an unseen target dataset. In step ②, we fine-tune all target datasets on all pre-trained models in the model zoo and capture the fine-tuning performances $T_{m \to t}$. We use accuracy as the performance metric, but the framework could be used with other performance metrics, such as f1 or MSE. These performances will be used to connect models and datasets when constructing the graph in step ⑤.

Note that at this point, it may seem like we are doing exactly what we were trying to prevent: brute-force fine-tune all pre-trained models to obtain the best fine-tuning performances. However, for our experiments, we are leaving out the fine-tuning performances for each target dataset under test. The other fine-tuning performances we consider to be obtained over time and part of the model zoo's structured metadata. This is not an unrealistic expectation; public model zoos such as HuggingFace quite consistently have this data available, and for organizations building their own model zoos it is considered good practice to capture training metadata through platforms such as MLflow[1] or W&B[2].

To obtain these scores, we adopt a fine-tuning method that retrains all layers' parameters, which is considered effective by reusing all previously learned knowledge. For very large pre-trained models, this becomes more expensive in terms of memory used. In Ziyu Li et al. (2024), we also adopted, *LoRA* (Hu et al. 2021), which aims at training only a part of the model's parameters to allow for more time- and memory-efficient fine-tuning. The mechanism is to freeze all model parameters and inject trainable rank decomposition matrices into each layer to reduce the number of trainable parameters. This enables the use of larger batch sizes and learning rates, achieves quicker convergence, but may lead to slightly reduced performance. However, the results were very similar, and the models in our model zoo do not have the size that requires more efficient fine-tuning methods. Hence, this fine-tuning method will be omitted in this thesis. More details of our fine-tuning setup will be given in Chapter 5 and Chapter 6.

In case the dataset is a source dataset, we do not fine-tune to obtain the performance, but use the reported pre-trained accuracy – if available.

### 4.2.3. Transferability Scores

In our system design for TransferGraph, we include the capture of relationships between datasets and models through existing transferability methods, as shown in step ④. As mentioned earlier

[1] https://mlflow.org/
[2] https://wandb.ai

in this section, we experimented using a combination of transferability scores and fine-tuning performance in our earlier work (Ziyu Li et al. 2024). Adding existing transferability scores such as LogME (You et al. 2021) to the graph could be done as a proxy, if for experiments we would not have access to the computational resources required to obtain the actual ground-truth performances, as in Y.-K. Zhang et al. (2024). However, including both would mean that the time it took to obtain these transferability scores would have to be added to the total time spent by our method. In that case, we can no longer argue that this is a ground-truth obtained over time or a proxy of it, instead it is additional information used by our method.

## 4.3. Stage 2: Graph Construction & Learning

In this stage (steps ⑤ and ⑥), we will use the metadata collected in the previous stage to construct a graph and learn the node embeddings. These node embeddings are then used to learn a final regression model which predicts the links between the unseen target dataset and the source models. As we have discussed in Section 2.4, this is a common approach for graph learning tasks.

### 4.3.1. Graph Construction

To successfully convert transferability estimation to a link prediction problem on a graph, it is crucial to identify the entities and their relationships in the graph. Figure 4.2 gives a detailed breakdown of how we construct the graph. In Table 4.1, we summarize the statistics of the graph constructed in our experiments.

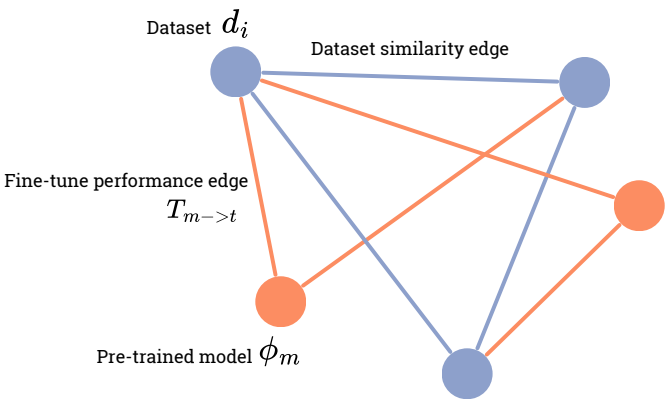



**Figure 4.2:** Illustration of a detailed view of the constructed graph.

#### Nodes and node properties

Nodes in the graph can be either a dataset or a pre-trained model. A dataset $d_i$ can either be a source dataset, used to pre-train the model, or one of our evaluation datasets. While we differentiate slightly in how we construct the edges for these datasets, for the graph learning methods, these nodes will be treated equally. Models in a public model zoo are often pre-trained or fine-tuned using the same benchmark dataset(s), meaning the number of model nodes in our set exceeds the number of dataset nodes. While GNN-based graph learning methods can incorporate node properties, we do not incorporate metadata as node features in the graph. Instead, we always learn from these features by incorporating them in the training of the prediction model in the next stage.

#### Edges

Next, to construct the edges, we use the relationships introduced in Section 3.2.3. These are:

- **Dataset-dataset edges.** Edges between datasets represent their similarity and are constructed using the embeddings generated in the previous stage. These edges are com-

puted pair-wise for each dataset using correlation distance, where shorter distance indicates greater similarity.

- **Model-dataset edges.** For the source datasets, we use the pre-training accuracy if it was reported. For the evaluation datasets, we use historic fine-tuning performances. We vary in our experiments how these are used, because not all graph embedding methods can incorporate edge properties or multiple types of edges.

| Graph property | Image experiments | Text experiments |
|---|---|---|
| Graph type | homogenous | homogenous |
| Threshold on accuracy for edge pruning | 0.5 | 0.5 |
| Threshold of negative edge identification on accuracy | 0.5 | 0.5 |
| Number of nodes | 265 | 188 |
| Average node degree* | 20.1 | 8.6 |
| Number of dataset-dataset edge | 5256 | 550 |
| Number of model-dataset edge with accuracy weight | 1753 | 918 |

**Table 4.1:** Summary of the graph property statistics. (* indicates that the value vary when the dataset and model collection changes)

### 4.3.2. Graph Learning

Now we have a constructed graph, representing a network of transferability relations between models and tasks, we can learn meaningful representations for the nodes in the graph. In Section 2.4, we introduced various types of graph learning methods and how they learn to produce graph embeddings. We also introduced possible downstream tasks, such as node classification, link prediction, and graph generation. As explained in Section 3.2.3, our problem setting can be reformulated as a link prediction problem.

Although our ultimate goal is to predict the accuracy of pre-trained model nodes with respect to the unseen target dataset node (i.e., edge weights), most graph learning techniques cannot or are typically not directly trained using edge weights (Z. Liu et al. 2024). Instead, the training of the graph embedding method is done on negative and positive pairs of nodes. These embeddings are then typically used to train a regression model to predict the edge weights. Since we obtained fine-tuning performances for all pairs of pre-trained models and target dataset, all nodes are connected. Hence, it is important to define positive and negative node pairs. Table 4.1 shows the thresholds that we used for this.

We test the performance of TransferGraph using two random walk-based methods; **Node2Vec** (Grover & Leskovec 2016), **Node2Vec+** (R. Liu et al. 2023), and two GNN-based methods; **GraphSAGE** (Hamilton et al. 2017) and **GAT** (Veličković et al. 2018). We provide a detailed explanation of these methods and outline their respective strengths and weaknesses.

**Node2Vec (Grover & Leskovec 2016)** performs its random walk as follows. Given we want to generate a random walk $v_0, v_1, .., .., v_k$ and that we already passed edge $(v_{i-1}, v_i)$, the next node $v_{i+1}$ in the walk is visited based on the probability:

$$P(v_{i+1}|v_i) = \begin{cases} \frac{\alpha_{v_i v_{i+1}}}{Z} & \text{if } (v_{i+1}, v_i) \in E, \\ 0 & \text{otherwise.} \end{cases} \tag{4.1}$$

where $Z$ is a normalization constant and $\alpha_{v_i v_{i+1}}$ is defined as:

$$\alpha_{v_i v_{i+1}} = \begin{cases} \frac{1}{p} & \text{if } d_{v_{i-1} v_{i+1}} = 0, \\ 1 & \text{if } d_{v_{i-1} v_{i+1}} = 1, \\ \frac{1}{q} & \text{if } d_{v_{i-1} v_{i+1}} = 2. \end{cases} \tag{4.2}$$

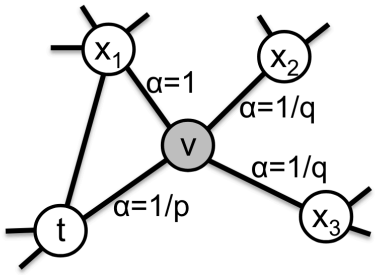



**Figure 4.3:** Illustration of random walk from Grover & Leskovec (2016), where the walk has just transitioned from $t$ to $v$, and is determining where to walk next out of node $v$.

Parameters $p$ and $q$ can be customized for different strategies. The parameter $p$ is the return parameter and controls the probability of immediately returning to the previous node in the walk. The parameter $q$ is the in-out parameter, allowing the walk to differentiate between *inward* and *outward* nodes. Figure 4.3 shows the illustration used in Grover & Leskovec (2016), where the transition from $t$ to $v$ just happened, and we are looking where to go next. As can be seen, a high value of $p$ increases the probability of returning, whereas a high value of $q$ favors inward nodes (i.e., nodes to the just visited nodes), over outward nodes (i.e., nodes not connected to our just visited node $t$).

**Node2Vec+ (R. Liu et al. 2023)** is an extension of Node2Vec which takes edge weights into account when generating the random walk. The probability of visiting a node is influenced by the edge weight. The closer the nodes are according to their edge weight, the higher the probability of visiting the node in the random walk.

**GraphSAGE (Hamilton et al. 2017)** for Graph SAmple and AGregate, was one of the first *inductive* graph learning frameworks, meaning the graph does not have to be retrained to generate node embeddings for unseen nodes. This is a desirable property, especially as graphs become larger. Algorithm 1 shows the original algorithm to compute the node embeddings. Each node's embeddings are first initialized with its attributes. Then, the node embedding at iteration $k$ is computed by concatenating the node and its neighbors embeddings at the previous iteration $k-1$. This can be seen in the inner loop in the algorithm. The authors explore different aggregation functions, such as long-short-term-memory(LSTM), mean and pooling.

**Algorithm 1** GraphSAGE embedding generation (i.e., forward propagation) algorithm

**Input:** Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{x_v, \forall v \in \mathcal{V}\}$; depth $K$; weight matrices $W^k, \forall k \in \{1, \ldots, K\}$; non-linearity $\sigma$; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, \ldots, K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$
**Output:** Vector representations $z_v$ for all $v \in \mathcal{V}$
$h_v^0 \leftarrow x_v, \forall v \in \mathcal{V}$
**for** $k = 1 \ldots K$ **do**
  **for** $v \in \mathcal{V}$ **do**
    $h_{\mathcal{N}(v)}^k \leftarrow \text{Aggregate}_k(\{h_u^{k-1}, \forall u \in \mathcal{N}(v)\})$
    $h_v^k \leftarrow \sigma\left(W^k \cdot \text{Concat}(h_v^{k-1}, h_{\mathcal{N}(v)}^k)\right)$
  **end for**
  $h_v^k \leftarrow h_v^k / \|h_v^k\|_2, \forall v \in \mathcal{V}$
**end for**
$z_v \leftarrow h_v^K, \forall v \in \mathcal{V}$

**GAT (Veličković et al. 2018)** Graph Attention Networks (GAT) are a specific application of self-attention in graph learning. They utilize the self-attention mechanism (Vaswani et al. 2017)

to compute attention scores and aggregate features for each node from its neighbors. In the case of GAT, the state of a node $i$ at layer $l$ is defined as

$$h_i^l = \sigma\left(\sum_{j \in \mathcal{N}(i} \alpha_{i,j}^l W^k h_j^{l-1}\right) \tag{4.3}$$

$$\alpha_{i,j} = \frac{\exp(\text{LeakyReLU}(a^T[Wh_j \| Wh_i]))}{\sum_{k \in \mathcal{N}_i} \exp(\text{LeakyReLU}(a^T[Wh_j \| Wh_l]))} \tag{4.4}$$

$$h_i^0 = x_i, \tag{4.5}$$

where $\alpha_{i,j}$ is the attention coefficient of node $i$ to its neighbor $j$ defined by a softmax function. $N_i$ is the set of neighbors of node $i$, $W$ is the weight matrix, and $a$ is a weight vector. Finally, $||$ refers to concatenation.

Table 4.2, summarizes the most important advantages and disadvantages of these methods, as discussed in Section 2.4.

| **Method** | **Advantage** | **Disadvantage** |
|---|---|---|
| Random walk-based | Scalable | Cannot generalize over unseen nodes, cannot easily incorporate node and edge properties |
| GNN-based | Can generalize over unseen nodes, can incorporate node and edge attributes | Scalability |

**Table 4.2:** Comparison of random walk-based and GNN-based graph learning methods.

We use the following hyperparameters for training the graph learners:

- For **all** graph embedding methods, we use a feature embedding dimension of 128. This is sufficiently large to capture the inherent structure of our graph, which is fairly small for graph learning standards.
- For **Node2Vec**, we use $p = q = 1$ to have a balanced combination of local and global exploration. Additionally, we use ten for both the length of the random walks as the number of walks per node. We train for fifteen epochs.
- For **GraphSAGE** and **GAT**, we train for 50 epochs on a homogeneous graph, using Adam optimizer and binary cross entropy as a loss function.

Finally, these methods will be evaluated in terms of effectiveness and efficiency in Section 6.1.5 and Section 6.2.3, respectively.

## 4.4. Stage 3: Regression Learning

After collecting the metadata, constructing the graph, and learning the node embeddings, we use the training history as the label for the regression problem to predict the performance of an unseen target task. Step ⑦ shows how we combine the node embeddings and basic metadata to construct a supervised learning set for our regression task. There are multiple possible supervised learning methods that are suitable for this regression task. As shown in ⑧, we explore the effectiveness of linear regression (LR), random forest (RF), and eXtreme gradient boosting (XGB).

**Linear regression** Similarly to Amazon LR H. Li et al. 2023, we use linear regression as one of our prediction models. Linear regression is a simple, but often applied supervised learning method which computes the linear relationship between a dependent variable and one or more independent features.

**Random forest** Random forest is an ensemble learning technique that constructs multiple decision trees at training time and outputs the mean prediction of the individual trees in case of regression. We set the number of trees as 100, max depth as 5.

**XGBoost** XGBoost (eXtreme Gradient Boosting) (T. Chen & Guestrin 2016) is an implementation of gradient boosted decision trees, designed for speed and performance. It is a powerful machine learning algorithm that is widely used in competitive machine learning and data science. We set the number of trees as 500, and maximum depth as 5.

Table 4.3 shows two rows which were used as input to train the supervised model for one of the TweetEval (Barbieri et al. 2020) tasks as target dataset, transposed for readability here. These rows represent the edges between the GLUE (A. Wang et al. 2018) CoLA and SST2 target datasets and Distilbert-base-uncased and Roberta-large pre-trained models. First, we can see the basic metadata we introduced in Section 4.2 (as far as they apply to text datasets). This is followed by the reported upstream accuracies of the models on the source datasets, and the fine-tuning performance on the target dataset. Finally, we see the embeddings of the model nodes $mf$ and the target dataset nodes $df$.

| **Feature name** | **Row 1** | **Row 2** |
|---|---|---|
| $m_i$ | distilbert-base-uncased | roberta-large |
| $d_t$ | glue/cola | glue/sst2 |
| Architecture | DistilBertForMaskedLM | RobertaForMaskedLM |
| Model number of labels | 2 | 2 |
| Model number of parameters | 67M | 36M |
| Model memory consumption | 267 MB | 1.4 GB |
| Dataset size | 8550 | 70000 |
| Dataset number of classes | 2 | 2 |
| $d_s$ | Wikipedia | Wikipedia |
| $T_{m_i \to d_t}$ | 0.808 | 0.864 |
| $T_{m_i \to d_s}$ | 0.864 | 0.923 |
| $mf_0$ | 0.010757 | 0.006599 |
| $mf_{1 \to 126}$ | ... | ... |
| $mf_{127}$ | 0.000000 | 0.000000 |
| $df_0$ | 0.016718 | -0.215635 |
| $df_{1 \to 126}$ | ... | ... |
| $df_{127}$ | 0.000000 | 0.000000 |

**Table 4.3:** Example rows used in the supervised learning stage. $mf$ and $df$ represent the source model and the target dataset embeddings, respectively.

## 4.5. Stage 4: Transferability Estimation

In the final stage, we use the learned prediction model to output the expected performance. Step ⑨ shows how we construct the test set similarly to the training set; for each pre-trained model $\phi_m$, we construct a row with both the node embedding of the model and the target dataset $d_t$. The row further contains the basic metadata, as introduced in previous sections. The output is collected in ⑩, which is a set of pairs of pre-trained models and predicted fine-tuning accuracies. These predicted accuracies will be used in Chapter 6, comparing them with the ground-truth fine-tuning accuracies which we left out of the graph and the training set for the prediction model.

## 4.6. Summary

In this chapter, we introduced *TransferGraph*, a novel framework for transferability estimation. We reframed transferability estimation as a link prediction problem on a graph, where pre-trained models and datasets form nodes, and the edges represent either past fine-tuning performances or dataset similarities. Previous research has shown that these relationships are valuable, and by representing them in a graph, we hypothesize that we can capture them more effectively.

We also demonstrated how additional metadata can be incorporated as node attributes in this graph, which have been shown to be useful predictors of a model's transferability to a new task.

Additionally, we discuss the details of constructing the graph and the chosen graph learning methods. We discuss four graph learning methods: Node2Vec (Grover & Leskovec 2016), Node2Vec+ (R. Liu et al. 2023), GraphSAGE (Hamilton et al. 2017) and GAT (Veličković et al. 2018). Finally, we discuss how we use the graph embeddings, metadata, and fine-tuning performances to train a supervised learning model which will produce the estimated transferability score. We discuss three supervised learning methods suited for this regression task: linear regression (LR), random forest (RF) and eXtreme Gradient Boosting (XGB).

**Summary of contributions**

- A novel framework which solves transferability estimation through graph learning. It includes an end-to-end process from collecting the metadata, graph construction and learning, and finally making the prediction.

# 5

# Metadata Collection & Benchmark Suite

In this thesis, we stress the importance of making good use of metadata for meta-machine learning tasks, such as transferability estimation. In this chapter, we give an in-depth technical description of how we collect the various metadata discussed in Chapter 4. Our system for dataset and pre-trained model loading and training is built on HuggingFace's transformers library[1], but is flexible and designed to be easily extensible to other libraries. The source code can be found in our GitHub repository[2].

This system connects to other components that are needed to compute the metadata for TransferGraph, such as the dataset embeddings and other baseline transferability methods. These components are built upon a combination of available papers' open-source code and benchmark repositories, such as those by Bai et al. (2023)[3].

In this chapter, we will go over the technical implementation of all steps needed to collect the metadata for TransferGraph, and show how our framework can be expanded to a new experiment setting. This includes loading the datasets and pre-trained models, dataset preprocessing, dataset embedding, obtaining the baseline transferability scores and the ground truth fine-tuning performances.

## 5.1. Experiment setup

The sections below will give the exact characteristics of the used pre-trained models and tasks, baselines and hardware setup used to run the experiments. The experiments are effectively divided into two model zoos: one for image classification and one for text sequence classification.

### 5.1.1. Target tasks

Table 5.1 gives an overview of the target tasks used for evaluation. In total, we include eight text classification tasks and eight image classification tasks. These are public tasks often used in both classification benchmarks and other transferability estimation studies. As can be seen in the table, the tasks vary in number of samples and classes.

Below, we will give a short description together with some examples of each of the target datasets which are used in our experiments.

**GLUE** (A. Wang et al. 2018) (General Language Understanding Evaluation benchmark[4]) is a widely adopted benchmark for natural language understanding systems. We use CoLA (The

[1] https://github.com/huggingface/transformers/
[2] https://github.com/TransferGraph/transfergraph
[3] https://github.com/Ba1Jun/model-selection-nlp
[4] https://gluebenchmark.com/

| Dataset | Samples | Classes |
|---|---|---|
| *Text Sequence Classification Tasks* | | |
| TweetEval/Sentiment (Barbieri et al. 2020) | 59,900 | 3 |
| RottenTomatoes (Pang & Lee 2005) | 10,662 | 2 |
| Glue/SST-2 (A. Wang et al. 2018) | 70,000 | 2 |
| TweetEval/Emotion (Barbieri et al. 2020) | 5,050 | 4 |
| TweetEval/Irony (Barbieri et al. 2020) | 4,600 | 2 |
| TweetEval/Hate (Barbieri et al. 2020) | 13,000 | 2 |
| Glue/CoLA (A. Wang et al. 2018) | 8,550 | 2 |
| TweetEval/Offensive (Barbieri et al. 2020) | 24,300 | 18 |
| *Image Classification Tasks* | | |
| Caltech101 (Fei-Fei et al. 2004) | 3,060 | 101 |
| Cifar100 (Krizhevsky 2012) | 50,000 | 100 |
| DTD (Cimpoi et al. 2014) | 5,640 | 47 |
| Flowers (Nilsback & Zisserman 2008) | 1,020 | 10 |
| Pets (Parkhi et al. 2012) | 3,680 | 37 |
| SmallNORB/Elevation | 24,300 | 9 |
| Stanfordcars (Krause et al. 2013) | 8,144 | 196 |
| SVHN (Netzer et al. 2011) | 73,257 | 10 |

**Table 5.1:** Overview of used target tasks used for evaluation.

Corpus of Linguistic Acceptability), which are texts annotated with whether they are grammatical English sentences; and SST-2 (The Stanford Sentiment Treebank), which are sentences of movie reviews and human annotations on their sentiment.

| Dataset | Sentence | Label |
|---|---|---|
| CoLA | Our friends won't buy this analysis, let alone the next one we propose. | Acceptable |
| | They drank the pub. | Unacceptable |
| SST-2 | On the worst revenge-of-the-nerds clichés the filmmakers could dredge up | Negative |
| | Is pretty damned funny. | Positive |

**Table 5.2:** Example of used GLUE tasks.

**TweetEval** (Barbieri et al. 2020) is a multi-label tweet classification dataset, consisting of seven tasks: irony, hate, offensive, stance, emoji, emotion, and sentiment.

| Dataset | Tweet | Label |
|---|---|---|
| Emotion | I love swimming for the same reason I love meditating...the feeling of weightlessness. | joy |
| Hate | Another illegal alien that shouldn't be in America killed an innocent American couple! #BuildThatWall | hateful |
| Irony | Leaving whilst its dark is fun. #not | ironic |
| Offensive | Are we all ready to sit and watch Indakurate Passcott play football? | non-offensive |
| Sentiment | Hmmmmm where are the #BlackLivesMatter when matters like this a rise... kids are a disgrace!! | negative |

**Table 5.3:** A tweet sample for each of the tasks in TweetEval, for the tasks used in our experiments.

**Rottentomatoes** (Pang & Lee 2005) is a small dataset containing 5331 positive and 5331 negative processed sentences from the Rotten Tomatoes review database. The goal is to classify whether the sentence has a positive or negative sentiment about the movie.

| Text | Label |
|---|---|
| It's so laddish and juvenile, only teenage boys could possibly find it funny. | Negative |
| Offers that rare combination of entertainment and education. | Positive |

**Table 5.4:** Samples from the Rotten Tomatoes dataset.

**Caltech101** (Fei-Fei et al. 2004) consists of around 9000 images of object belonging to 101 classes. Classes have between 40 and 800 images each, and images are of variable sizes.

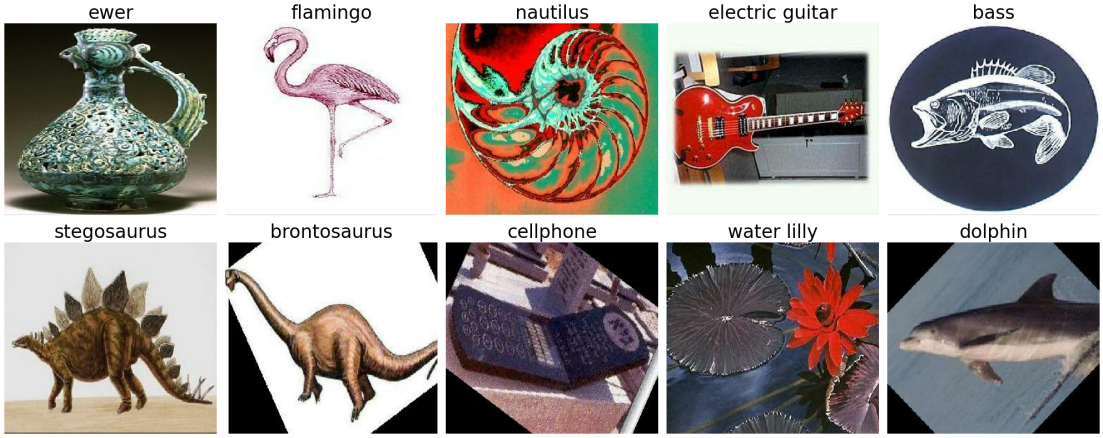



**Figure 5.1:** Examples of the Caltech101 dataset.

**Cifar100** (Krizhevsky 2012) is a labelled subset of the 80 Million Tiny Images Dataset (Birhane & Prabhu 2021). It consists of 100 classes, with 600 images each. The reasoning for using tiny images was that this is more memory efficient, while still holding the core visual patterns. This is also the reason the (enlarged) example images below look blurry.


**Figure 5.2:** Examples of the Cifar100 dataset.

**DTD** (Cimpoi et al. 2014) (Describable Textures Dataset) consists of 5640 images, organized into 47 categories describing their textures according to human perception. Each category has 120 images, and images have variable sizes.

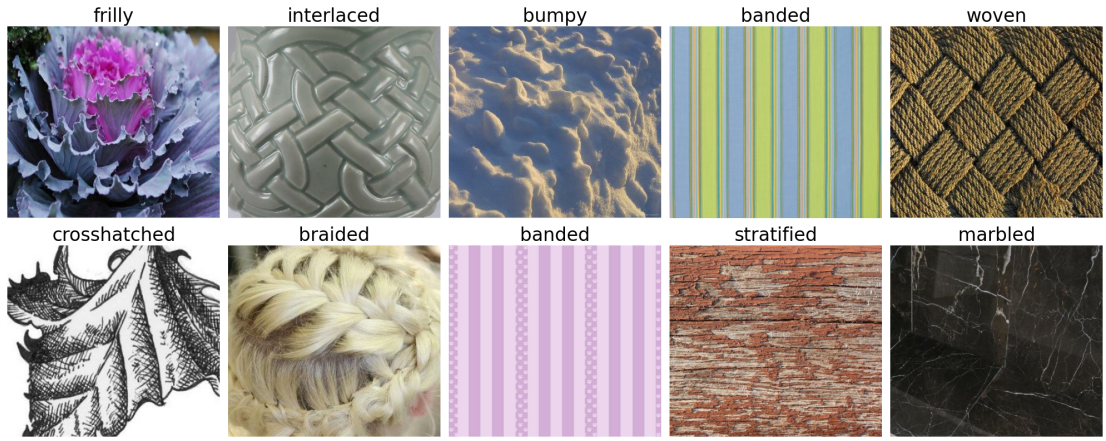



**Figure 5.3:** Examples of the DTD dataset.

**Flowers** (Nilsback & Zisserman 2008) (Oxford Flowers 102) is a dataset of 102 flower categories which occur in the United Kingdom. Each class has between 40 and 258 images, which vary in scale, pose and light.


**Figure 5.4:** Examples of the Flowers dataset.

**Pets** (Parkhi et al. 2012) (Oxford-IIT Pet) has images of 37 categories of breeds of cats and dogs. Each class has roughly 200 images, and have large variations in pose, scale, and lighting.


**Figure 5.5:** Examples of the Pets dataset.

**SmallNORB** (Small NYU Object Recognition Benchmark) is a dataset for 3D object recognition. The objects are toys in five categories: four-legged animals, human figures, airplanes, trucks, and cars. The images of the toys were captured by 2 cameras under 6 lighting conditions, 9 elevations and 18 azimuths. We use the task which classifies the elevation, which can be between 30 and 70 degrees, for every 5 degrees.

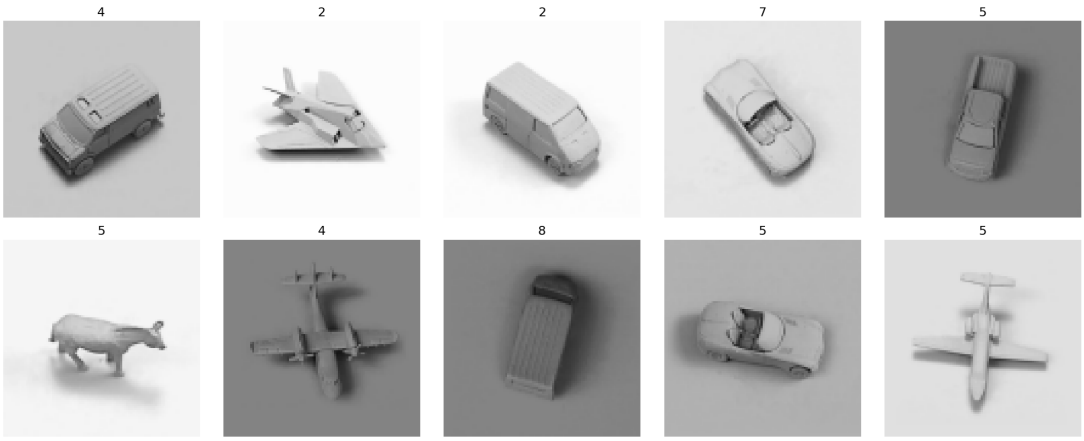



**Figure 5.6:** Examples of the SmallNORB dataset.

**Stanfordcars** (Krause et al. 2013) is a dataset containing images of 196 classes of cars. Classes are typically organized like *make*, *model*, *year* (e.g. Tesla Model S 2012).


**Figure 5.7:** Examples of the Stanfordcars dataset.

**SVHN** (Netzer et al. 2011) (Street View House Numbers) is a large-scale digit classification dataset, containing 600,000 32 by 32 images of house numbers. The goal is to classify the digit centered in the frame.


**Figure 5.8:** Examples of the SVHN dataset.

### 5.1.2. Pre-trained models

For a transferability method to be successful, it should be able to generate a score with high correlation with the actual performance for different types of models and target datasets. As discussed in Section 3.1.4, many related works are limited in the sense that they are validated on one type of architecture of models, or ones which were all pre-trained on the same source dataset. To verify the stability of our estimation, we construct a heterogeneous model zoo consisting of models with varying architectures, source datasets, and sizes. Table 5.5 summarizes the main characteristics of the used pre-trained models.

| **Image Classification Models** | | **Text Classification Models** | |
|---|---|---|---|
| **Architecture** | **Count** | **Architecture** | **Count** |
| vit | 114 | bert | 80 |
| swin | 25 | distilbert | 37 |
| convnext | 24 | roberta | 27 |
| beit | 9 | xlm-roberta | 5 |
| deit | 5 | electra | 5 |
| van | 5 | albert | 3 |
| resnet | 1 | fnet | 2 |
| data2vec-vision | 1 | camembert | 2 |
| | | deberta-v2 | 1 |
| | | perceiver | 1 |
| | | data2vec-text | 1 |
| **Range of Parameters** | | | |
| <100k | 2 | | 1 |
| 100k-1M | 0 | | 0 |
| 1M-10M | 5 | | 1 |
| 10M-100M | 155 | | 46 |
| 100M-1B | 22 | | 116 |
| **Source dataset** | | | |
| hfpics* | 67 | glue/qqp | 16 |
| imagenet | 48 | glue/cola | 16 |
| eurosat | 19 | wikipedia | 9 |
| imagenet-21k | 10 | glue/sst2 | 9 |
| beans | 9 | dair-ai/emotion | 9 |
| cifar10 | 8 | imdb | 8 |
| cats_vs_dogs | 5 | bertweet-covid19-cased-tweet-data | 4 |
| food101 | 3 | quora | 3 |
| chest_xray_classification | 3 | tweet_eval/emotion | 3 |
| age-prediction | 3 | amazon_reviews_multi/en | 2 |
| poolrf2001/facemask | 3 | glue/mnli | 2 |
| Matthijs/snacks | 2 | kaggle-financial-sentiment | 2 |
| Other | 22 | Other | 31 |
| **Totals** | 184 | | 164 |

**Table 5.5:** Summary of characteristics of pre-trained models used in our experiments. Contrary to many related works, our model zoo is heterogeneous in the types of architectures, model size and pre-trained datasets used.
* the entries combined as *hfpics* are multiple types of datasets queried with different keywords, using https://github.com/nateraw/huggingpics.

### 5.1.3. Baselines

We compare TransferGraph with two state-of-the-art baseline methods for transferability estimation. We include LogME (You et al. 2021), as it has been shown to perform well on image classification tasks by Agostinelli et al. (2022). We also include Reg-HScore (Ibrahim et al. 2023), which has been found superior in both efficiency and effectiveness on text classification tasks by Bai et al. (2023).

## 5.2. Metadata Collection

This section will outline how we collect the metadata for the experiment setting above. To do this, we developed a tool that can be easily adapted to different experiment settings. Below, we will show the details of this implementation.

### 5.2.1. Dataset Loading and Preprocessing

Given a set of target datasets, pre-trained models, a fine-tuning configuration, and baselines to compare to, the system supports collecting the required metadata and transferability estimation scores. The first step is to load and preprocess the datasets. Neither local files nor HuggingFace datasets have a consistent way of defining inputs and labels. This means that after selecting a set of target datasets, configuration is required for each dataset. The tool includes predefined configurations for the datasets used in the experiments, but new configurations can be easily added. For code examples on how to use our system, refer to Appendix A. Our tool currently supports loading datasets either locally or from HuggingFace. After downloading, they are preprocessed to be used for subsequent tasks. We apply common practices for preprocessing data to be used for downstream tasks, depending on the type of task:

- **Image classification.** Preprocessing for image classification tasks includes normalizing the pixel values to be on the same scales, and resizing the images to be consistent with the pre-trained model's required input format. For training tasks, we apply a random crop and horizontal flip transforms from the PyTorch library to prevent overfitting.
- **Text sequence classification.** Pre-trained language models on HuggingFace come with their own tokenizer, which map words to input values for the model. Similar to image classification models, language models have a maximum input length. We use the common practice of padding input sequences in a batch to the longest sequence in the batch, and truncating them to the maximum length the model accepts.

### 5.2.2. Pre-trained Model Loading

HuggingFace's implementation of loading pre-trained models works consistently, so our tool does not have an abstraction layer on top of it. A pre-trained model can simply be loaded using its identifier from HuggingFace Hub. For downstream tasks, we configure it to discard its classification layer and initialize a new one with the number of labels associated with that task.

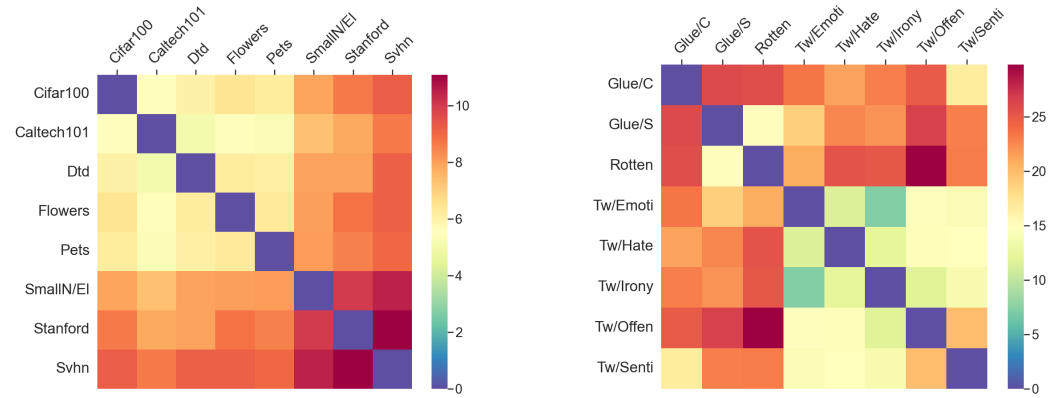
### 5.2.3. Dataset Embedding

As described in Section 4.2, we use Domain Similarity (Cui et al. 2018) as the dataset embedding method. These embeddings are used to calculate the distances between datasets in the graph. Domain Similarity requires a probe model to extract the features. To embed the dataset, the dataset and the probe model can be loaded as described above and fed into the component that computes and stores the embeddings. For convenience in reproducing our experiments, we have made the embeddings available online[5]. In our experiments, we chose Google/vit-base-patch16-224[6] for image datasets and EleutherAI/gpt-neo-125[7] for text datasets.

While most target datasets are manageable in size, especially source datasets used to pre-train the models can be very large. For example, for the models in the BERT (Devlin et al. 2019) family, the English Wikipedia is used as a source dataset, containing more than 6 million Wikipedia pages. The same goes for image pre-trained models, which are pre-trained on ImageNet21k (Ridnik et al. 2021), containing 11 million images. In these cases, while it is valuable for our method to know the relation of the target to the source dataset, it is too expensive to compute the embedding of the entire dataset. Hence, we randomly sample up to 100,000 entries to compute the embeddings.

To illustrate the effectiveness of these embeddings, Figure 5.9 shows the distances between the target datasets for both the text and the image datasets used in our experiments. The domain similarity captured is as you would intuitively expect for these datasets. The Cifar100, Caltech101 and DTD datasets are multi-domain datasets, hence, they are most similar to the other datasets and especially to each other. These datasets also often contain categories related to plants and animals; hence, we see more similarity to the Pets and Flowers datasets. This also holds for the text datasets: all tweet evaluation datasets form a similar group. Likewise, GLUE/SST-2 and RottenTomatoes are also similar, as they both cover movie reviews.

[5] https://huggingface.co/datasets?search=TransferGraph
[6] https://huggingface.co/google/vit-base-patch16-224
[7] https://huggingface.co/EleutherAI/gpt-neo-125m




(**a**) Image datasets. (**b**) Text datasets.

**Figure 5.9:** Heatmap showing dataset distances.

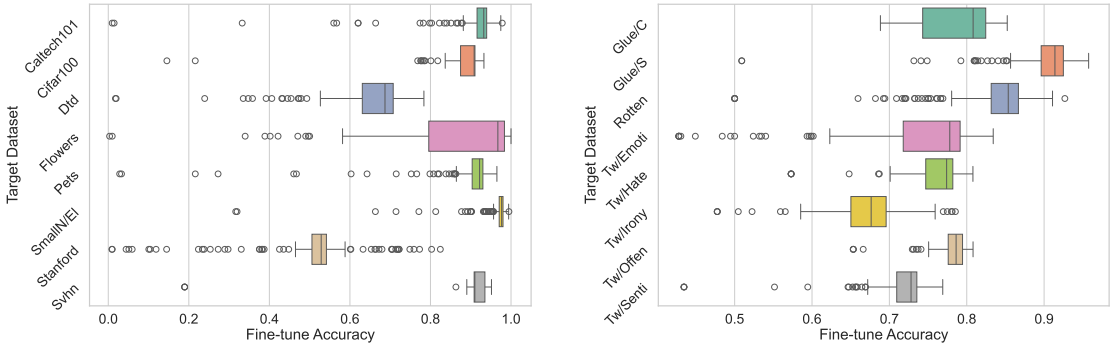
### 5.2.4. Collecting Baseline Transferability Scores

For collecting existing transferability estimation metrics, we primarily integrate a benchmark suite by Bai et al. (2023). They evaluated the effectiveness of using only a subset of samples and reducing the feature dimension using PCA. We collect the baseline scores without reducing the number of samples from the target datasets, as this approach generally achieves the highest performance. Additionally, we did not integrate feature reduction, as the used baselines do not significantly benefit from it. However, this would be an interesting addition to our benchmark suite, especially when comparing metrics that benefit significantly from reduced feature dimensions, such as PARC (Bolya et al. 2021), Logistic (Kumari et al. 2022), and PACTran (Ding et al. 2022).

### 5.2.5. Collecting Fine-Tuning Performances

A major factor in the success of TransferGraph, lies in learning from past fine-tuning performances. In this section, we will describe the details of collecting the fine-tuning performances. In total, we spent 4400 GPU hours fine-tuning the pre-trained image models, and 700 hours for the pre-trained language models.

Figure 5.10 shows the distribution of accuracies after fine-tuning of all pre-trained models. It can be noted that the distribution of these fine-tuning performances are very different among datasets. Especially for the text datasets, the majority of accuracies after fine-tuning are relatively high, between 0.7 and 0.9. In these cases, randomly picking a model to fine-tune might be sufficient, making transferability estimation not necessary or very beneficial. More discussion on this will be done in Section 6.1.2.




(**a**) Image datasets. (**b**) Text datasets.

**Figure 5.10:** Distribution of fine-tuning accuracies for the target datasets.

#### Hardware

Fine-tuning performances were gathered using the Delft AI Cluster (DAIC), formerly known as INSY-HPC. Most fine-tuning performances were gathered using NVIDIA GeForce GTX 1080 Ti with 11 GB of RAM.

For larger, (especially image) pre-trained models, we used NVIDIA Tesla V100 with 16-32 GB of RAM or NVIDIA A40 with up to 48 GB of RAM to speed up the fine-tuning.

### Libraries

To run the fine-tuning on GPUs, we use CUDA Toolkit 12.1[8]. To fine-tune the pre-trained models, we implement the training loop using `transformers=4.37.2` and `torch=2.20.0`. Hyperparameters can be conveniently configured using Transformer's `TrainingArguments`. Our tool by default uses `accelerate`[9] to be able to train using multiple GPUs.

### Hyperparameters

Learning rate, weight decay and momentum (for stochastic gradient descent) play an important role in the final performance of fine-tuning (H. Li et al. 2020). In the paradigm of fine-tuning, the target dataset is typically smaller, and it is common practice to use smaller initial learning rates and fewer epochs. However, which hyperparameters to use depends not only on the target data but also on the similarity between the source and target domains (H. Li et al. 2020).

To find the best hyperparameters for each source model and target dataset, some works resort to hyperparameter grid search (You et al. 2021; Ibrahim et al. 2023). However, they typically only use a few pre-trained models or severely limit the original dataset size. The number of reported GPU hours at the start of this chapter does not include hyperparameter grid search. For the large-scale evaluation used for TransferGraph, hyperparameter grid search would be infeasible, as even a conservative search would increase the required GPU hours by at least a factor of ten. Some learning-based approaches combine rankings from multiple transferability estimation metrics (Y.-K. Zhang et al. 2024). However, as discussed in Section 3.1.4, this approach potentially incorporates the shortcomings of these methods into the newly proposed method, as they are used as ground truth.

To solve these problems, we used modern fine-tuning methods, which we empirically found to generalize well over the target datasets used in our experiments. Table 5.6 shows the hyperparameters used to fine-tune the pre-trained models in the different experiment setups. For the used image datasets, we found stochastic gradient descent to generalize well, especially in combination with a cyclical learning rate (Smith 2017), with maximum learning rate of 0.01. For experiments on text classification, we found that AdamW (Loshchilov & Hutter 2018), combined with linear decreasing learning rates initialized at 2e-4, generalized better across target datasets and pre-trained models. For image classification tasks, we did not notice a big difference in performance when varying the batch size, so we varied it depending on the memory consumption to save time. For text datasets, the experiments ran fast enough and we kept a consistent batch size.

| Hyperparameter | Text | Image |
|---|---|---|
| Batch size | 16 | 8-64 |
| Learning rate | 2e-5 | 1e-2 (Cycle max) |
| Epochs | 5 | 30 |
| Optimizer | AdamW | SGD |
| Scheduler | Linear | OneCycle |

**Table 5.6:** Hyperparameter settings used for collecting ground-truth performances for TransferGraph.

### Experiment Tracking

Our tool for obtaining fine-tuning performances integrates with Weights & Biases and uploads fine-tuned models, including model cards, to HuggingFace Hub. This facilitates analyzing and monitoring of fine-tuning experiments and gives a better historical record of the impact of changing hyperparameters. For the text classification experiments of TransferGraph, the fine-tuning artifacts are made publicly available[10][11].

## 5.3. Summary

This chapter covered the implementation of our system to prepare the metadata required for our framework. This includes extracting the basic metadata of datasets and pre-trained models, computing the

[8] https://developer.nvidia.com/cuda-12-1-0-download-archive
[9] https://huggingface.co/docs/accelerate/index
[10] HuggingFace Hub: https://huggingface.co/TransferGraph
[11] W&B: https://wandb.ai/hvanderwilk/transfer_graph

dataset embeddings, and gathering past fine-tuning performances. We demonstrated this by introducing our experimental setup and providing results for each step needed in preparation for later evaluation.

First, we summarized the characteristics of the target datasets and showed examples of input and label pairs. In total, we used eight image classification tasks and eight text sequence classification tasks for these experiments. Next, we discussed the pre-trained models used in our experiments, which included 184 pre-trained image classification models and 164 text classification models, encompassing various architectures and sizes.

Additionally, building on top of HuggingFace's Transformers library, we showed how our system conveniently loads these datasets and pre-trained models to extract metadata such as dataset embeddings. For both image and text datasets, these embeddings successfully capture the similarity between datasets.

Finally, we discussed our implementation for collecting other baseline transferability scores and fine-tuning performances.

**Summary of contributions**

- Expanding existing benchmark suites and building on top of HuggingFace's Transformers library, we introduce an easily extensible system to load datasets and pre-trained models, compute baseline transferability estimation scores and do result analysis.

# 6

# Evaluating TransferGraph

A solution is not good, just because it is novel or conceptually sound. Hence, we will not assume so for our graph learning-based approach for transferability estimation, and thoroughly evaluate TransferGraph in this chapter. Contrary to most transferability metrics, the experiments cover transferability estimation in the setting of both image and text sequence classification, using a heterogeneous model zoo, consisting of different types and sizes of pre-trained models and task. In Section 5.1, we have introduced our experiment setup, including the used datasets, pre-trained models and baselines. In this chapter, provide an analysis of TransferGraph both in terms of effectiveness (Section 6.1) and efficiency (Section 6.2).

## 6.1. Evaluation: Effectiveness

The evaluation of TransferGraph is separated into two main parts. This first part will cover the effectiveness of our transferability estimation method in comparison to the selected baselines. We will evaluate the effect of varying our approach in the following sections.

### 6.1.1. Evaluation setup

The essence of our evaluation approach is as follows; as outlined in Section 4.2, we construct a model zoo with $M$ pre-trained models $\{\phi_m\}_{m=1}^{M}$ and $N$ target datasets $\{d_t\}_{t=1}^{N}$. Next, we build our supervision set of fine-tuning performances $\{T_{m\to t}\}_{m=1}^{M}$ as described in Section 4.2.2. For every target dataset $d_t$, we leave the fine-tuning performances out of the supervision set and predict a transferability score $S_{m\to t}$ for every pre-trained model. Finally, we measure the success of this transferability score, by comparing the left out supervision fine-tuning performances to the estimated scores using the various correlation metrics discussed in Section 3.1.3. The resulting correlation scores are compared to various state-of-the-art baselines.

### 6.1.2. Evaluation metric

The success of a transferability estimation metric can be measured in various ways, as we discussed in Section 3.1.3. Most related work turn to rank correlation metrics to measure the success of their metric. However, as figure 5.10 reveals, generally, many source pre-trained models will have similar performances. Rank correlation metrics will give a high penalty for making predicting errors in the ranks between these source models, while realistically speaking, this error has no effect in the usefulness of the metric.

The lack of a consistent way to measure the effect of transferability estimation metrics has likely contributed to the contradictory findings in papers on the superiority of proposed methods. As observed by Agostinelli et al. (2022), NCE (Nguyen et al. 2020) consistently outperforms LEEP (Tran et al. 2019) in You et al. (2021), while LEEP outperforms NCE in Y. Tan et al. (2021) and Nguyen et al. (2020). The same applies to LogME (You et al. 2021), which outperforms LEEP, but LEEP outperforms LogME in Pandy et al. (2022).

Considering the real-life use cases as depicted in Section 3.1.1, we need to ask; what are these practitioners interested in? The answer that emerges is that they want a fast recommendation on which pre-trained model to fine-tune, which gives them the best performance on the target task. Furthermore, preferably, they fine-tune just one recommended model to achieve this performance. The only metric that does a decent job at capturing this success is **relative top-k accuracy** ($Rel@k$) – specifically, relative top-1-accuracy.

While Pearson's correlation has similar disadvantages as rank correlation, it does give an indication on how well a metric would perform if not all pre-trained models can be tested due to efficiency concerns. As we have noted earlier, most transferability metrics need a forward of the entire target dataset over a pre-trained model to estimate its ability to transfer knowledge. In these cases, Pearson's correlation would be a good choice, since realistically, practitioners will prefer not to choose a lot of pre-trained models to do this for. However, as we will explore in Section 6.2, our method can infer transferability efficiently once the graph learner is trained. This means testing many pre-trained models is feasible, and makes focusing on $Rel@k$ more logical.

In our evaluation in terms of effectiveness below, we will therefore focus on this metric. We will make an important adjustment when it comes to evaluating the success over all target datasets. In the introduction, we show the top-1 accuracy of the best predicted models and also display the results if you would randomly pick a model. This gives valuable insights in the benefit of doing transferability estimation at all. Simply taking the average of target datasets' $Rel@k$ results may distort the success of a method, since for some datasets the gain over random selection is very little. Hence, we will take a weighted average, taking into account the absolute possible accuracy gain over random selection: $T_{\max} - T_{\text{rand}}$.

### 6.1.3. Hardware

All experiments were run using an NVIDIA A40 GPU with 48 GB of RAM. The preparation steps, including the collection of dataset embeddings and the construction of the supervision set of fine-tuning performances, were conducted using different hardware. However, this will not be considered as part of the efficiency analysis in the following sections, as we assume this data is gathered offline. For details on the hardware used in the preparation for the experiments, we refer to Chapter 5.

### 6.1.4. Main findings

The main findings are represented in Table 6.1, with a breakdown of the effectiveness of our method compared to the other baselines, according to the most commonly used correlation metrics. On average, TransferGraph outperforms the baselines in all evaluation metrics, except for W-Kendall rank correlation on text target datasets. Most importantly, on hard-to-predict datasets, such as `Stanfordcars`, TransferGraph is able to predict the best suited model, with an accuracy of 0.823, whereas LogME and Reg-H-Score only slightly pick a better model than randomly selecting a model, both retrieving a model with an accuracy of 0.509.

The method presented here is our most competitive method, which uses GAT (Veličković et al. 2018) as graph embedding method and eXtreme Gradient Boosting (XGB) (T. Chen & Guestrin 2016) as regression model. In the sections below, we will outline the effect of other explored approaches.

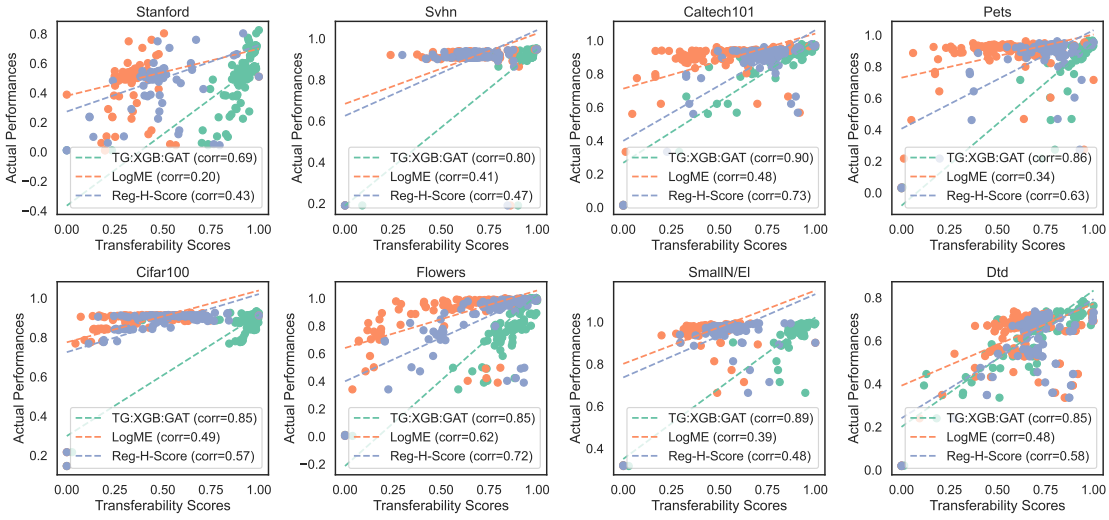



**Figure 6.1:** Image datasets' correlation between the actual performances after fine-tuning and the predicted scores, for both our approach and the baselines.

| Method | Image Target Dataset | | | | | | | | Mean |
|---|---|---|---|---|---|---|---|---|---|
| | Stanford | Svhn | Caltech101 | Pets | Cifar100 | Flowers | SmallN/El | Dtd | |
| **Pearson Correlation** | | | | | | | | | |
| TG:XGB,GAT | **0.691** | **0.803** | **0.897** | **0.860** | **0.847** | **0.847** | **0.886** | **0.846** | **0.795** |
| Reg-H-Score | 0.433 | 0.473 | 0.734 | 0.630 | 0.570 | 0.721 | 0.481 | 0.578 | 0.554 |
| LogME | 0.203 | 0.410 | 0.480 | 0.343 | 0.491 | 0.620 | 0.392 | 0.480 | 0.380 |
| **Rel@1 Accuracy** | | | | | | | | | |
| TG:XGB,GAT | **1.000** | 0.998 | 0.981 | 0.988 | **0.990** | 0.980 | **0.998** | 0.973 | **0.989** |
| Reg-H-Score | 0.634 | **1.000** | **0.994** | **0.993** | 0.986 | **0.990** | 0.907 | **0.993** | 0.858 |
| LogME | 0.634 | **1.000** | **0.994** | 0.744 | 0.986 | **0.990** | 0.907 | 0.940 | 0.827 |
| random | 0.616 | 0.952 | 0.922 | 0.920 | 0.946 | 0.876 | 0.964 | 0.822 | 0.792 |
| **Top@1 Accuracy** | | | | | | | | | |
| TG:XGB,GAT | **0.824** | **0.950** | 0.960 | **0.954** | **0.923** | 0.980 | **0.993** | **0.762** | **0.880** |
| Reg-H-Score | 0.509 | 0.947 | **0.972** | **0.954** | 0.912 | **0.990** | 0.901 | 0.731 | 0.757 |
| LogME | 0.509 | 0.947 | **0.972** | 0.715 | 0.912 | **0.990** | 0.901 | 0.691 | 0.729 |
| random | 0.508 | 0.906 | 0.902 | 0.888 | 0.883 | 0.876 | 0.959 | 0.644 | 0.713 |
| **W-Kendall Correlation** | | | | | | | | | |
| TG:XGB,GAT | **0.684** | **0.595** | **0.625** | **0.684** | **0.402** | **0.641** | **0.629** | **0.704** | **0.653** |
| Reg-H-Score | 0.334 | 0.367 | 0.519 | 0.189 | 0.384 | 0.578 | 0.184 | 0.321 | 0.369 |
| LogME | 0.218 | 0.380 | 0.486 | 0.081 | 0.393 | 0.635 | 0.129 | 0.253 | 0.310 |

| Method | Text Target Dataset | | | | | | | | Mean |
|---|---|---|---|---|---|---|---|---|---|
| | Rotten | Tw/Irony | Tw/Senti | Tw/Offen | Glue/C | Glue/S | Tw/Hate | Tw/Emoti | |
| **Pearson Correlation** | | | | | | | | | |
| TG:XGB,GAT | **0.897** | **0.809** | **0.880** | **0.931** | **0.765** | **0.894** | **0.899** | **0.769** | **0.839** |
| Reg-H-Score | 0.886 | 0.790 | 0.743 | 0.787 | 0.524 | 0.712 | 0.860 | 0.769 | 0.766 |
| LogME | 0.831 | 0.687 | 0.765 | 0.743 | 0.288 | 0.699 | 0.829 | 0.407 | 0.636 |
| **Rel@1 Accuracy** | | | | | | | | | |
| TG:XGB,GAT | 0.989 | **1.000** | 0.977 | 0.992 | 0.960 | **0.981** | 0.964 | **0.997** | **0.986** |
| Reg-H-Score | **0.996** | 0.888 | **0.983** | **0.997** | **0.974** | 0.969 | **0.989** | 0.936 | 0.953 |
| LogME | **0.996** | **1.000** | **0.983** | 0.983 | **0.974** | 0.969 | **0.989** | 0.936 | 0.978 |
| random | 0.919 | 0.853 | 0.949 | 0.962 | 0.934 | 0.948 | 0.940 | 0.890 | 0.910 |
| **Top@1 Accuracy** | | | | | | | | | |
| TG:XGB,GAT | 0.889 | **0.785** | 0.735 | 0.791 | 0.818 | **0.931** | 0.779 | **0.832** | **0.823** |
| Reg-H-Score | **0.896** | 0.697 | **0.740** | **0.795** | **0.830** | 0.920 | **0.799** | 0.781 | 0.797 |
| LogME | **0.896** | **0.785** | **0.740** | 0.783 | **0.830** | 0.920 | **0.799** | 0.781 | 0.816 |
| random | 0.827 | 0.670 | 0.713 | 0.767 | 0.796 | 0.900 | 0.760 | 0.742 | 0.760 |
| **W-Kendall Correlation** | | | | | | | | | |
| TG:XGB,GAT | 0.615 | 0.674 | 0.190 | 0.529 | 0.399 | **0.465** | **0.555** | **0.591** | 0.541 |
| Reg-H-Score | **0.663** | 0.610 | **0.466** | **0.635** | **0.670** | 0.265 | 0.510 | 0.496 | 0.551 |
| LogME | 0.635 | **0.702** | 0.445 | 0.553 | 0.668 | 0.263 | 0.508 | 0.457 | **0.553** |

**Table 6.1:** Overview of evaluation of the effectiveness of TransferGraph compared to other baselines, for all evaluated datasets.

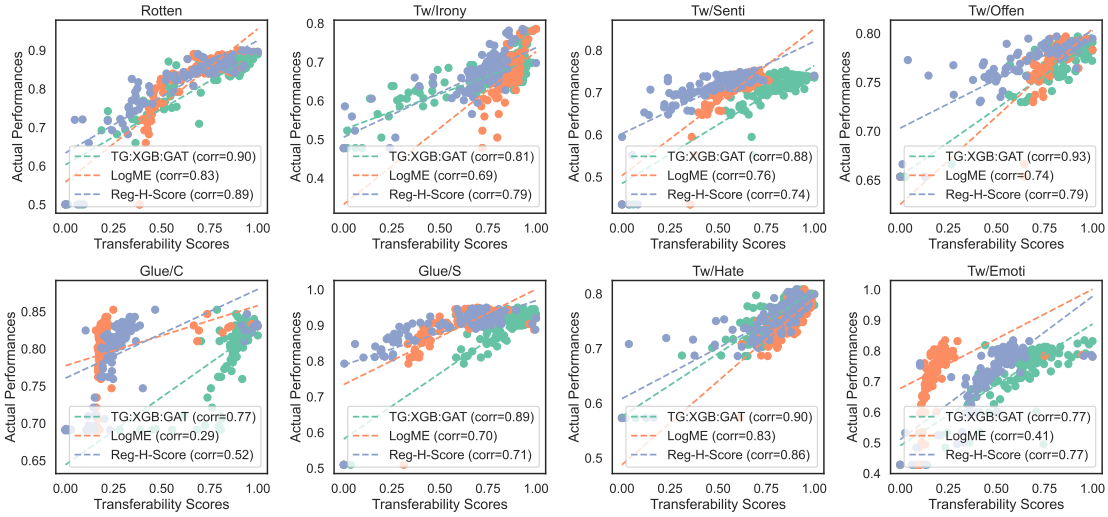
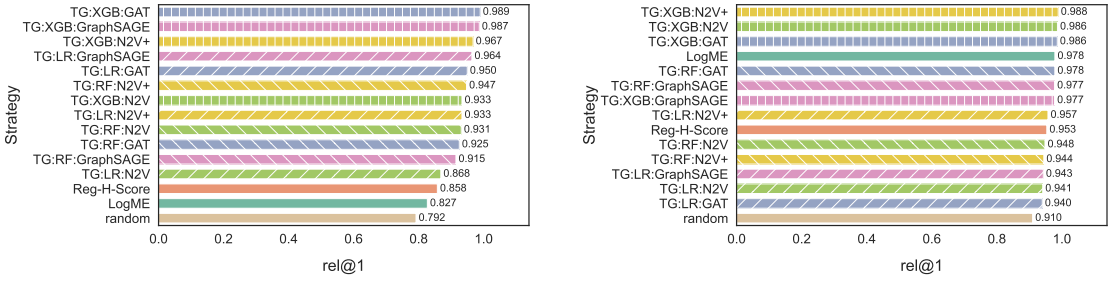



**Figure 6.2:** Text datasets' correlation between the actual performances after fine-tuning and the predicted scores, for both our approach and the baselines.

In Figure 6.1 and Figure 6.2, we present a detailed breakdown of the predicted scores, plotted against the actual performances after fine-tuning. For all datasets, our method has a higher correlation between the predicted scores and actual performances than the baselines. As we discussed at the beginning of this chapter, contrary to most transferability methods, we do not believe this metric should be used as a primary means to evaluate the success of a transferability method. However, this type of plot does give an interesting overview of how the method placed the pre-trained models with respect to the actual performances. One of the reasons why we believe that correlation is not a good metric is also visible in the plots. For example, in the case of the Stanfordcars dataset, our method produces an almost perfect correlation. However, due to a single pre-trained model not following the linear trend, the correlation is significantly lower. This is similar to what we discussed in Section 3.1.3, where we noted that the produced ranking could be perfect, while the predicted scores of the pre-trained models do not follow a linear correlation with the actual performances.

### 6.1.5. Effect of Graph Learning- and Regression learning method

First, we evaluate the effect of varying the graph learning method and the regression learning method to train the prediction model. As introduced in Section 4.3.2, we experiment using Node2Vec (Grover & Leskovec 2016), Node2Vec+ (R. Liu et al. 2023), GraphSAGE (Hamilton et al. 2017) and GAT (Veličković et al. 2018) as graph embedding methods and linear regression (LR), random forest (RF) and eXtreme Gradient Boosting (XGB) (T. Chen & Guestrin 2016) as methods to learn the prediction model.




(**a**) Image datasets.

(**b**) Text datasets.

**Figure 6.3:** Rel@1 score for all variations of graph learning methods and regression model types.

Figure 6.3 shows the results for varying both graph learning method and prediction model learning method. It depicts the average relative top-1 accuracy over all target datasets. As discussed earlier in this chapter, practitioners will ideally not want to perform costly fine-tuning on multiple pre-trained models to find the best one. Hence, a transferability method's best predicted pre-trained model (top-1) should give a high accuracy after fine-tuning.

The colors of the bars and the hatches are grouped by graph embedding method and regression learning method. It can be observed that generally, **XGB** (depicted with vertical hatches) outperforms the other regression learning methods. In combination with **GAT**, this method gives the **best relative top-1 accuracy** on average. To further assess the performance of the variations, in the next sections, we will evaluate the effect of increasing $k$ and the effect of reducing the number of pre-trained models.

### 6.1.6. Effect of $k$

In some scenarios, practitioners might have more resources to attempt fine-tuning more pre-trained models to find the best one. This notion is formalized by some works, by returning a number of pre-trained models to fine-tune depending on an initially defined budget (Renggli et al. 2022). In this section, we will evaluate the effect of increasing $k$, which can be viewed as less resource-constrained scenarios.

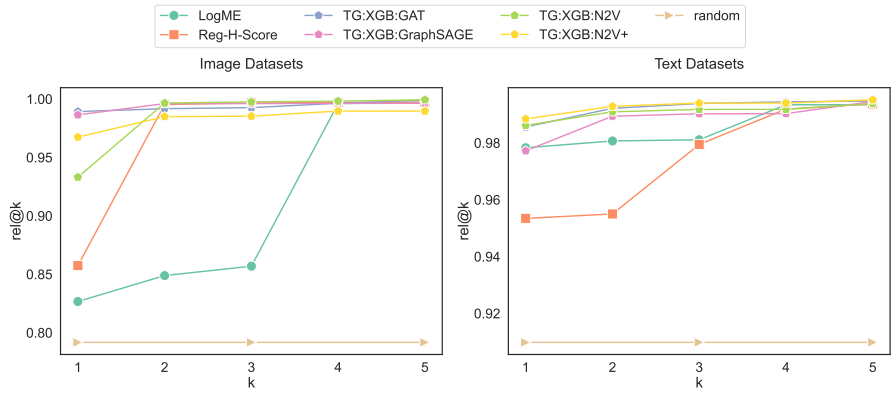



**Figure 6.4:** Average $Rel@k$ when increasing $k$ and varying the graph learning method, for both image (left) and text classification target datasets (right).

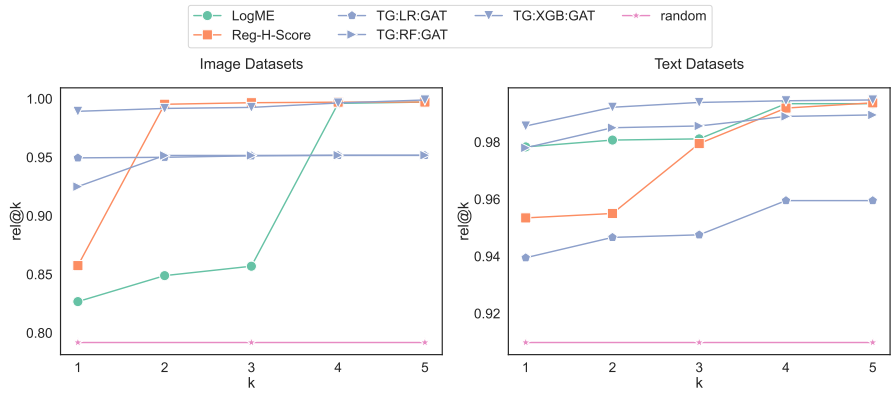



**Figure 6.5:** Average $Rel@k$ when increasing $k$ and varying the regression learning method, for both image (left) and text classification target datasets (right).

For visibility purposes, the figures do not show all variations of graph learning and regression learning methods. Instead, Figure 6.4 and Figure 6.5 fix XGB and GAT and evaluate the effect of varying the graph and regression learning method, respectively.

Generally, for higher values of $k$, both our variations and the baselines start to perform very similarly. Figure 6.5 shows Reg-HScore (Ibrahim et al. 2023) slightly outperforms our most competitive method on the image target datasets for $k = 2$ and $k = 3$. Another observation is that XGB significantly outperforms the other regression learning methods, even for higher values of $k$. As for graph learning methods, Figure 6.4 shows that all variations perform similarly well when increasing $k$.

### 6.1.7. Effect of Number of Pre-trained Models

Graph learning methods, especially those based on GNN, generally need many nodes and edges to learn meaningful representations. To assess the stability of our proposed approach, we will decrease the number of pre-trained models. As discussed in Section 3.1.4, related work often experiments with a very limited set of pre-trained models. For example, the surveys by Agostinelli et al. (2022) and Bai et al. (2023) use 16 and 6 source pre-trained models, respectively. This amounts to approximately ten percent of the number of pre-trained models used in our experiments. While we have argued that this is not a realistic setting in today's pre-trained model zoos, it does offer an indication of the stability of our approach and a better comparison with the results presented in related work.

There are multiple observations to be made from these results. Firstly, our methods perform fairly stable, for both image and text classification tasks. In the case of text classification tasks, all methods perform very similarly, and our approach that combines GAT and XGB is more stable than the baselines. For image classification tasks, we note that **Node2Vec** and **Node2Vec+** are **more stable** than the other methods (and especially the baseline methods) when the size of the model zoo changes.

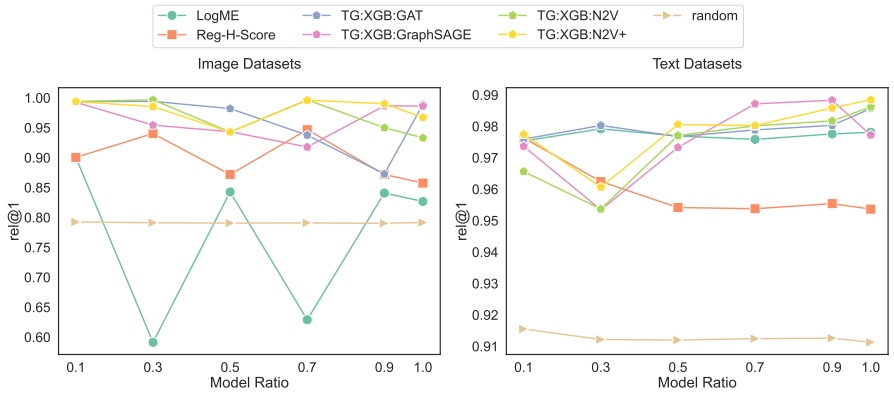



**Figure 6.6:** Average $Rel@k$ when varying the graph learning method under lower number of pre-trained models, for both image (left) and text (right) classification target datasets.

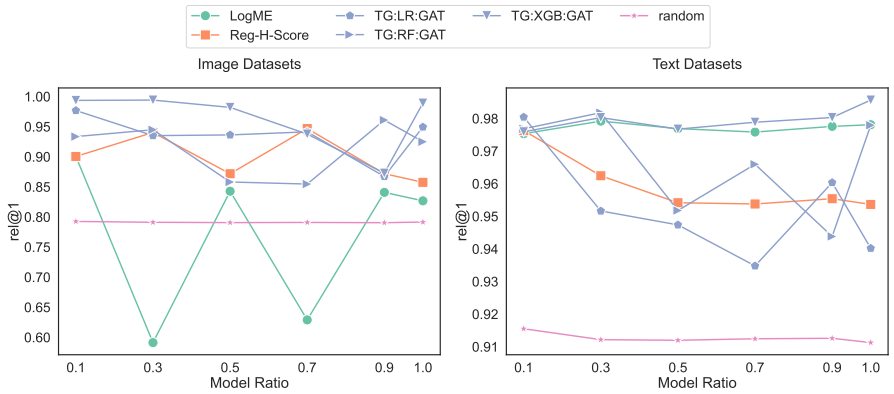



**Figure 6.7:** Average $Rel@k$ when varying the regression learning method under lower number of pre-trained models, for both image (left) and text (right) classification target datasets.

## 6.2. Evaluation: Efficiency

The essence of transferability estimation is finding an efficient heuristic to estimate how well a pre-trained model will transfer to a new task. A transferability estimation method should at least outperform brute force fine-tuning all pre-trained models in the model zoo. As we discussed in Section 3.1.4, most transferability estimation methods require a forward pass of the entire target dataset over all pre-trained models in the model zoo. While this is a lot more efficient than brute-force fine-tuning, this still requires significant resources, especially for a growing model zoo with more complex models.

In this section, we will analyze the efficiency of our proposed method. We again compare against LogME (You et al. 2021) and Reg-HScore (Ibrahim et al. 2023).

### 6.2.1. Runtime Steps

Before covering the efficiency results, we will provide some motivation on what steps of our method and the baselines are included in these results. The baselines we use are both source embedding methods, which means they require a forward pass over all pre-trained models to extract the features for the target dataset. For these methods, this requires the most significant work. After obtaining the extracted features, both methods can perform fairly efficient computation to obtain the score.

As discussed in Section 5.2.5, our method requires obtaining a historical record of fine-tuning performances to learn from. This requires a significant effort if no previous fine-tuning experiences are available. However, we argue that as model zoos mature, this data will become more readily available. This is already the case: when using the HuggingFace library defaults for fine-tuning, the fine-tuned model will be uploaded containing metadata about the fine-tuning experiment, including its accuracy. Since these model zoos are often public, some careful consideration is required when picking fine-tuning experiments to learn from, as the authors of the pre-trained model may have used different and suboptimal hyperparameters and fine-tuning methods. However, we do not see this as an insurmountable problem. Using indications

| Method | Step | Offline | Online |
|---|---|---|---|
| Source embedding | Extracting features | | ✓ |
| | Computing score | | ✓ |
| TransferGraph | Obtaining fine-tuning performances | ✓ | |
| | Obtaining source dataset embeddings | ✓ | |
| | Obtaining target dataset embeddings | | ✓ |
| | Obtaining graph embeddings | | ✓ |
| | Training prediction model | | ✓ |

**Table 6.2:** Runtime steps for the baselines and our method, and whether the steps can be done offline, or online.

of quality of the experiments available in these repositories, such as the organization that authored the model or the popularity, the practitioners we envision in Section 3.1.1 should be able to combine public fine-tuning experiences with their own to train their own transferability estimator.

Table 6.2 gives an overview of the components required by the baselines which we compare to and our method, and an indication on whether the component needs to be computed online or offline. Essentially, all steps required to construct the graph, including obtaining fine-tuning performances and embedding the source datasets, can be obtained offline. When estimating transferability for a new target dataset, we first have to compute its embedding to measure the similarity to other datasets that are already in the graph. Currently, every time a target dataset is added to the graph, we have to retrain the graph learner and prediction model. This is because we chose to evaluate many graph learning methods, some of which cannot be used *inductively*, as explained in Section 2.4. Our most competitive method, which uses GAT (Veličković et al. 2018), could be used inductively. Hence, future work could improve efficiency further, by changing the setup to not require retraining for every target dataset.

### 6.2.2. Main findings

Figure 6.8 shows a comparison of the runtime in seconds for obtaining the transferability estimation for our most competitive method, which uses XGB and GAT, compared to LogME and Reg-HScore. Similarly to the effectiveness experiments in Section 6.1, we vary the size of the model zoo to show the effect of more pre-trained models to evaluate. What can be observed is that both baseline methods perform very similarly, which can be explained by that the majority of the work involves extracting the features, which has to be done by both methods.

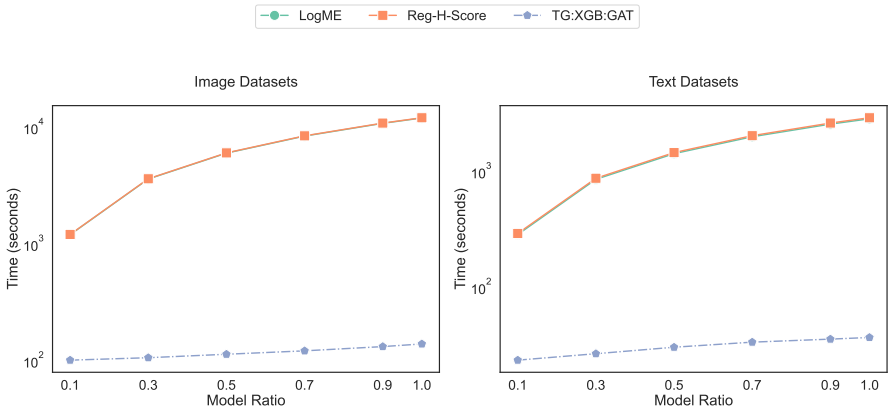



**Figure 6.8:** Average runtime in seconds for obtaining the transferability estimation of our most competitive method against the selected baselines, for image datasets (left) and text datasets (right).

Table 6.3 and Table 6.4 give a more detailed breakdown of the runtime, for the image and text target datasets, respectively. What can be observed is that computing the transferability scores using source embedding methods like LogME and Reg-HScore, especially for larger image datasets, becomes very expensive. For our largest dataset, SVHN (Netzer et al. 2011), with 600,000 image samples, it takes around 7.5 hours. Our method offers a 105 times speedup compared to that, at less than 5 minutes.

| Method | #Models | SmallN/El | Caltech101 | Cifar100 | DTD | Flowers | Pets | Cars | SVHN |
|---|---|---|---|---|---|---|---|---|---|
| LogME | 18 | 1,316s | 178s | 2,355s | 1,011s | 686s | 289s | 1,044s | 2,710s |
| | 55 | 3,948s | 533s | 7,064s | 3,033s | 2,059s | 868s | 3,133s | 8,129s |
| | 92 | 6,581s | 889s | 11,774s | 5,055s | 3,432s | 1,446s | 5,222s | 13,548s |
| | 129 | 9,213s | 1,244s | 16,483s | 7,078s | 4,804s | 2,025s | 7,311s | 18,967s |
| | 166 | 11,845s | 1,600s | 21,193s | 9,100s | 6,177s | 2,604s | 9,400s | 24,387s |
| | 184 | 13,161s | 1,778s | 23,548s | 10,111s | 6,863s | 2,893s | 10,444s | 27,096s |
| Reg-H-Score | 18 | 1327s | 181s | 2359s | 1017s | 689s | 293s | 1044s | 2728s |
| | 55 | 3,981s | 542s | 7,078s | 3,051s | 2,068s | 879s | 3,133s | 8,185s |
| | 92 | 6,636s | 903s | 11,797s | 5,085s | 3,447s | 1,464s | 5,222s | 13,642s |
| | 129 | 9,290s | 1,265s | 16,516s | 7,119s | 4,826s | 2,050s | 7,310s | 19,099s |
| | 166 | 11,944s | 1,626s | 21,235s | 9,154s | 6,205s | 2,636s | 9,399s | 24,556s |
| | 184 | 13,272s | 1,806s | 23,594s | 10,171s | 6,894s | 2,929s | 10,443s | 27,284s |
| TG:XGB:GAT | 18 | 75s | 13s | 151s | 229s | 45s | 21s | 67s | 206s |
| | 55 | 80s | 18s | 156s | 234s | 50s | 25s | 73s | 211s |
| | 92 | 88s | 25s | 163s | 240s | 58s | 34s | 80s | 219s |
| | 129 | 97s | 32s | 172s | 252s | 65s | 40s | 85s | 227s |
| | 166 | 105s | 43s | 181s | 258s | 76s | 52s | 98s | 241s |
| | 184 | 114s | 49s | 187s | 265s | 81s | 57s | 104s | 255s |

**Table 6.3:** Detailed view of runtime per image target dataset.

| Method | #Models | Glue/C | Glue/S | Rotten | Tw/Emoti | Tw/Hate | Tw/Irony | Tw/Offen | Tw/Senti |
|---|---|---|---|---|---|---|---|---|---|
| LogME | 16 | 86s | 843s | 134s | 53s | 178s | 44s | 268s | 670s |
| | 49 | 258s | 2,530s | 401s | 160s | 535s | 132s | 803s | 2,009s |
| | 82 | 430s | 4,217s | 668s | 266s | 892s | 219s | 1,338s | 3,348s |
| | 115 | 602s | 5,904s | 935s | 373s | 1,249s | 307s | 1,873s | 4,687s |
| | 148 | 775s | 7,591s | 1,202s | 479s | 1,606s | 395s | 2,408s | 6,026s |
| | 164 | 861s | 8,434s | 1,336s | 532s | 1,785s | 438s | 2,675s | 6,696s |
| Reg-H-Score | 16 | 92s | 853s | 140s | 58s | 184s | 49s | 274s | 676s |
| | 49 | 275s | 2,560s | 419s | 173s | 551s | 146s | 823s | 2,028s |
| | 82 | 459s | 4,267s | 698s | 289s | 919s | 243s | 1,371s | 3,380s |
| | 115 | 642s | 5,974s | 977s | 404s | 1,287s | 340s | 1,920s | 4,733s |
| | 148 | 825s | 7,681s | 1,256s | 520s | 1,654s | 437s | 2,468s | 6,085s |
| | 164 | 917s | 8,534s | 1,395s | 578s | 1,838s | 485s | 2,742s | 6,761s |
| TG:XGB:GAT | 16 | 11s | 58s | 15s | 5s | 16s | 8s | 24s | 50s |
| | 49 | 12s | 61s | 17s | 10s | 19s | 13s | 28s | 52s |
| | 82 | 18s | 65s | 21s | 11s | 24s | 16s | 34s | 54s |
| | 115 | 21s | 68s | 23s | 17s | 28s | 18s | 35s | 56s |
| | 148 | 23s | 70s | 26s | 19s | 30s | 21s | 36s | 60s |
| | 164 | 25s | 74s | 27s | 20s | 30s | 20s | 37s | 61s |

**Table 6.4:** Detailed view of runtime per text target dataset.

### 6.2.3. Effect of Graph Learning- and Regression Learning Method

In this section, we will evaluate the effect on the runtime when varying the graph embedding and regression learning method. Similarly to our effectiveness analysis in Section 6.1.5, in Figure 6.9, all combinations of graph learners and regression learning methods are displayed. The graph learning methods are grouped by color, whereas the regression learning methods are grouped by line markers. The depicted results are the average for all target datasets in our experiments.

We can see that the chosen graph learning method has the greatest influence on the runtime of our methods. The most efficient methods are Node2Vec (Grover & Leskovec 2016), followed by Node2Vec+ (R. Liu et al. 2023). GraphSAGE (Hamilton et al. 2017) and GAT (Veličković et al. 2018) are the slowest methods. This follows the line of expectations set out in Section 2.4. GNN-based methods are more computationally expensive than random walk-based methods, and Node2Vec+ is more expensive because it has to account for edge weights. The differences in regression learning methods, at least at this scale, are very limited. However, we can clearly see that XGB is more expensive than RF and LR.

Finally, what should be noted is that with the current model zoo sizes, the majority of the runtime of our method consists of embedding the target dataset. This takes between 3–55 seconds for text datasets and 9–255 seconds for image datasets.

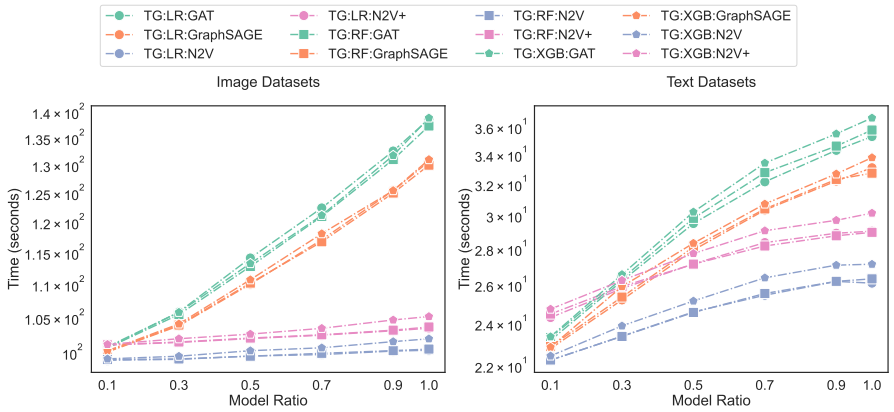



**Figure 6.9:** Average runtime in seconds for obtaining the transferability estimation when varying the graph learning and regression learning methods, for image datasets (left) and text datasets (right).

## 6.3. Summary

In this chapter, the performance of various configurations of *TransferGraph* was compared against two state-of-the-art methods: LogME (You et al. 2021) and Reg-HScore (Ibrahim et al. 2023). All twelve combinations of graph learners and regression learning methods are assessed, both in terms of effectiveness as efficiency.

We evaluated the effectiveness according to commonly used metrics; Pearson correlation, weighted Kendall's rank correlation and relative top-k accuracy. We have shown that except for weighted Kendall correlation on text datasets, our most competitive method outperforms the baselines on all evaluation metrics. Especially on lower values of k, our method is better at identifying a pre-trained model with a good accuracy for the target task.

Given the complete model zoo, using GAT (Veličković et al. 2018) as graph learner combined with XGB (T. Chen & Guestrin 2016) to train the regression model has been shown to achieve the best performance on average, closely followed by using GraphSAGE. Even though Node2Vec (Grover & Leskovec 2016) and Node2Vec+ (R. Liu et al. 2023) outperform the other methods on text datasets, the performance difference is only minor. Node2Vec and Node2Vec+ do have the advantage that they perform more stable when reducing the number of pre-trained models.

Finally, we have shown that TransferGraph is significantly faster than the state-of-the-art methods. As discussed in earlier chapters, a major downside of these methods is that they require a forward pass over the pre-trained model using the entire dataset, making their runtime dependent on both the model size and dataset size. On the largest image task, our method is 106 times faster, and on the largest text task, it is 110 times faster. While our method requires significant preparation effort, we have argued in this chapter that, given the increasing richness and structure of metadata in model zoos, this preparation can be done offline.

**Summary of contributions**

- In contrast to related wroks, we propose a more consistent approach to measure the effectiveness of transferability estimation methods over multiple tasks, which may vary in terms of how much they benefit from transferability estimation.
- We show that TransferGraph outperforms the state-of-the-art transferability estimation methods LogME (You et al. 2021) and Reg-HScore (Ibrahim et al. 2023) in terms of effectiveness on most evaluation metrics.
- We show that TransferGraph can estimate transferability orders of magnitude faster than these methods.
- We show that TransferGraph's performance is stable when varying the model zoo size in terms of number of pre-trained models to choose from.

# 7

# Conclusion and Outlook

As model zoos mature and offer practitioners a growing choice of pre-trained models, it is becoming more and more challenging to choose the right model for a new task, even for experts. Brute-force fine-tuning all pre-trained models to find the best one in such a model zoo, which can be in the thousands, has become infeasible in terms of required computational resources. In the past years, various transferability estimation methods have been proposed to offer an efficient heuristic to find the best pre-trained model to fine-tune. The most successful methods rely solely on the features extracted by a pre-trained model and the target labels of the dataset. Doing so, they disregard any additional (meta-)information of these pre-trained models and datasets, such as the architecture of the pre-trained model or the number of classes in a dataset. These indicators have been shown to be useful to incorporate when estimating transferability, especially in diverse model zoos.

In this thesis, we have investigated whether we can more effectively predict a pre-trained model which has a high performance on a new, downstream task. The main goal of this thesis was to find out:

> ***RQ: How can transferability estimation be performed more effectively and efficiently in large and diverse model zoos?***

Since existing surveys only provided an overview of some transferability estimation methods, the first subgoal we set to find out was:

> *RQ1: Which methodologies for transferability estimation have been introduced in existing literature?*

In Chapter 2, we identified 29 published methods and divided them into six categories; dataset similarity with source embedding, dataset similarity with optimal transport, model similarity, source-target label comparison, source embedding, and methods that learn from fine-tuning history. Each of these categories of related work captures a relation between the source and the target task. Dataset similarity and source-target label comparison methods exploit the similarities of the source and target datasets to estimate transferability. Source embedding methods explore the relation between the source pre-trained model and target task through extracted features. Model similarity methods the relationship between the source and target model. Methods that learn from fine-tuning history learn the relationship of source model and target task through historic fine-tuning performances, and additionally use metadata of pre-trained models and datasets to learn this relation. Given these relationships and available metadata, the next sub-goal was to explore:

> *RQ2: What types of metadata can be used to effectively estimate transferability?*

This was done in Chapter 3. We identified two types of dataset metadata and six types of pre-trained model metadata that have been shown to be useful in predicting fine-tuning performance. Examples of indicators are those of hardness and complexity, such as the number of labels of a dataset, the number of parameters of a model, and its architecture. We also identify two types of relationships, which can be obtained efficiently. First, dataset-dataset relationships, which represent their similarity and can be computed through extracting their features by a probe model and computing the distance. The second relation represents past fine-tuning performances between pre-trained models and target datasets. Using these relations and metadata, the next sub-goal was to find out:

*RQ2: How can we improve the effectiveness of transferability estimation in diverse model zoos?*

In the data lake community, datasets and their relations have been modelled using a graph. Given these rich relationships between datasets and pre-trained models, in Chapter 3, we motivated our hypothesis that modeling transferability estimation as a graph learning problem could improve effectiveness. By modeling datasets and pre-trained models as nodes, and their relationships as edges, we converted transferability estimation to a link prediction problem on a graph. In this setting, the goal is to predict a positive link between a new target dataset and a pre-trained model, if its fine-tuning performance is good.

In Chapter 4, we introduced our novel framework for transferability estimation: *TransferGraph*. To evaluate TransferGraph, we constructed two model zoos; one with 184 pre-trained image classification models, and one with 164 pre-trained text classification models. The details of setting up these model zoos for our experiments are covered in Chapter 5. For both model zoos, we test the transferability estimation performance of our method on eight target datasets compared to state-of-the-art methods: LogME and Reg-HScore. With the constructed graph, we evaluated the performance of 12 variations of our method in different sizes of model zoos. These variations use different graph learning methods, as well as different regression learning methods. The evaluated graph learning methods are Node2Vec, Node2Vec +, GraphSAGE and Graph Attention Networks (GAT) and the evaluated regression learning methods are linear regression (LR), random forest (RF) and eXtreme gradient boosting (XGB).

In Section 6.1, we have shown that GAT in combination with XGB has the best average performance in terms of relative top-1 accuracy among these variations. When reducing the size of the model zoo, we have shown that Node2Vec and Node2Vec+ have more stable performance. Furthermore, we have shown that compared to the state-of-the-art, our variation using GAT and XGB could more effectively estimate transferability. On average, this variation outperformed the baselines in all evaluation metrics, except for one in the text classification experiments. Especially on datasets which were hard to predict a fitting pre-trained model, our method significantly outperformed the baselines.

Although our method might be more effective, the main goal behind transferability estimation is to find an *efficient* way to measure the potential fine-tuning performance of a pre-trained model. A crucial question is therefore to ask:

*RQ3: How can transferability estimation be made more efficient in large model zoos?*

Not all of these metadata and relationships can be retrieved efficiently; model-model similarity requires fine-tuning of a (small) target model to compute the similarity to the others. Source embedding methods have to do a forward pass over all the pre-trained models to extract their specific features to do analysis on. While both are orders of magnitude more efficient than brute-force fine-tuning, with a growing number of pre-trained models to choose from, the effort is still significant. The relation in the form of past fine-tuning performance is not easy to obtain either. However, the advantage is that this only has to be done once and can be obtained incrementally.

The state-of-the-art methods in transferability estimation are source embedding methods, which require a forward pass over the source pre-trained model's feature extractor layer of the entire target dataset, for every pre-trained model. In our experiment set-up, we include up to 184 pre-trained models. While public model zoos have thousands of pre-trained models, already in our small set-up, we have shown that these methods take up to hours to execute. By learning from past fine-tuning performances, we do not have to perform this expensive step and have shown that all variations of our method only take up to five minutes to perform the estimation on all pre-trained models, as shown in Section 6.2. When comparing among our variations, the Node2Vec and Node2Vec+ methods are especially efficient, but as mentioned previously, they trade this efficiency for slightly reduced effectiveness.

To conclude and answer the goal of this thesis, we have shown that transferability estimation can be improved in terms of both effectiveness and efficiency, by utilizing additional dataset and pre-trained model metadata to reframe the problem as a link prediction problem on a graph. This proposed framework, TransferGraph, outperforms the state-of-the-art, is most effective when used with GAT and XGB as learning methods, and is most efficient when used with Node2Vec.

## 7.1. Recommendations for Future Work

In this thesis, we conducted an initial study on the use of graph learning to tackle transferability estimation. While we have explored various aspects, many areas remain open for further investigation. Below, we provide recommendations for future research. We first cover additional research that could be conducted within the scope of this thesis, followed by potential future work beyond its scope.

### 7.1.1. Within the Scope of this Thesis

**Hyperparameter Search for (Graph) Learning Methods** In Chapter 4, we explain the covered graph and supervised learning methods and our used hyperparameters. Future work could improve the effectiveness of this approach, by performing hyperparameter optimization for both learning approaches. For all graph learning methods, this could mean exploring the effect of changing the embedding dimension, both in terms of effectiveness as efficiency, since a higher dimension will require more computational resources.

For Node2Vec and Node2Vec+, the parameters $p$ and $q$ could be adjusted, to, for example, favor capturing the local or global structure. Integrating hyperparameter search as part of the framework could also improve the stability of our approach, since differently structured graphs may benefit from different hyperparameters. Similarly, for GraphSAGE and GAT, future works could study the effect of adjusting the optimizer and loss functions.

**Inductive Graph Node Embedding** When we introduced the different graph embeddings methods in Section 2.4, a mentioned advantage of some graph learning methods is that they are inductive, instead of transductive. In our setting, this essentially means that when a new target dataset is introduced which we want to do transferability estimation for, the graph does not have to be learned from scratch to embed the new target dataset node. This is especially useful in the case of very large graphs. Future research could study the effect of allowing this framework to make inductive embeddings and predictions for GraphSAGE and GAT, which support inductive embeddings. Especially for larger experiment setups, this could amount to significant efficiency gains.

### 7.1.2. Beyond the Scope of this Thesis

**Large (Generative) (Language) Models** In this thesis, we studied transferability estimation in the context of image and text sequence classification, for models with up to 393 million parameters. This is relatively small compared to today's large (language) models. Today's OpenAI flagship model, GPT-4o, has 175 billion parameters. Fine-tuning models at this scale and in the setting of generative models is still largely undiscovered, let alone applying transferability estimation in these settings. A major challenge in research in this area is the cost and environmental impact of fine-tuning models at this scale.

Lin et al. (2024) pioneers research in the field of applying transferability estimation in the setting of generative language models. Y.-K. Zhang et al. (2024) study applying transferability estimation in the setting of fine-tuning large language models to do classification tasks. However, more research is needed to successfully apply transferability estimation in the case of large language models.

**Transferability Estimation for Selecting Efficient Models** Works on transferability estimation have a heavy focus on selecting the most effective model for the downstream task. A critical reader that has some deep learning experience might ask: *"Why do you not just select a very large model, instead of trying to estimate transferability?"*. This would be a decent strategy in general, larger models often generalize good over unseen tasks. However, this cannot offer a reasonable alternative to transferability estimation for two reasons. Firstly, especially for harder to learn task, we have observed large variation even among the largest models. For example, the best performing model on the Stanfordcars dataset is Facebook/convnext-large-384-22k-1k, with an accuracy of 0.824. At 197 million parameters, it is 1.5 times smaller than the largest model in our zoo: Facebook/convnext-xlarge-224-22k. This pre-trained model only achieved an accuracy of 0.721. Second, this would mean you always choose the most inefficient model to use in terms of memory consumption and inference time.

Some works do incorporate efficiency into the choice of selecting a pre-trained mode. In SHiFT, Renggli et al. (2022), the user of their *transfer learning engine* is allowed to filter models by their number of parameters. This essentially allows practitioners to set resource constraints in terms of memory consumption and inference time. Deep learning models are growing in size at a staggering rate, with all associated computational and thus environmental impact. Future works could further investigate the above approach, by capturing the user's preference for effectiveness over efficiency when recommending pre-trained models after transferability estimation.

# Bibliography


Achille, Alessandro, Michael Lam, Rahul Tewari, Avinash Ravichandran, Subhransu Maji, Charless C Fowlkes, Stefano Soatto & Pietro Perona (2019). "Task2vec: Task embedding for meta-learning". In: *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 6430–6439.

Agostinelli, Andrea, Michal Pándy, Jasper Uijlings, Thomas Mensink & Vittorio Ferrari (2022). "How Stable Are Transferability Metrics Evaluations?" In: *Computer Vision – ECCV 2022*, pp. 303–321.

Alvarez-Melis, David & Nicolo Fusi (2020). "Geometric Dataset Distances via Optimal Transport". In: *Advances in Neural Information Processing Systems*. Vol. 33, pp. 21428–21439.

B, Vimal K, Saketh Bachu, Tanmay Garg, Niveditha Lakshmi Narasimhan, Raghavan Konuru & Vineeth N Balasubramanian (2023). "Building a Winning Team: Selecting Source Model Ensembles using a Submodular Transferability Estimation Approach". In: *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 11575–11586.

Bai, Jun, Xiaofeng Zhang, Chen Li, Hanhua Hong, Xi Xu, Chenghua Lin & Wenge Rong (2023). "How to Determine the Most Powerful Pre-trained Language Model without Brute Force Fine-tuning? An Empirical Survey". In: *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 5369–5382.

Bao, Yajie, Yang Li, Shao-Lun Huang, Lin Zhang, Lizhong Zheng, Amir Zamir & Leonidas Guibas (2019). "An Information-Theoretic Approach to Transferability in Task Transfer Learning". In: *2019 IEEE International Conference on Image Processing (ICIP)*, pp. 2309–2313.

Barbieri, Francesco, Jose Camacho-Collados, Luis Espinosa Anke & Leonardo Neves (2020). "TweetEval: Unified Benchmark and Comparative Evaluation for Tweet Classification". In: *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 1644–1650.

Bassignana, Elisa, Max Müller-Eberstein, Mike Zhang & Barbara Plank (2022). "Evidence \textgreater Intuition: Transferability Estimation for Encoder Selection". In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 4218–4227.

Ben-David, Shai & Reba Schuller (2003). "Exploiting Task Relatedness for Multiple Task Learning". In: *Learning Theory and Kernel Machines*. Lecture Notes in Computer Science, pp. 567–580.

Birhane, Abeba & Vinay Uday Prabhu (2021). "Large image datasets: A pyrrhic win for computer vision?" In: *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, pp. 1536–1546.

Bolya, Daniel, Rohit Mittapalli & Judy Hoffman (2021). "Scalable Diverse Model Selection for Accessible Transfer Learning". In: *Advances in Neural Information Processing Systems*. Vol. 34, pp. 19301–19312.

Castro Fernandez, Raul, Ziawasch Abedjan, Famien Koko, Gina Yuan, Samuel Madden & Michael Stonebraker (2018). "Aurum: A Data Discovery System". In: *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pp. 1001–1012.

Chen, Fenxiao, Yun-Cheng Wang, Bin Wang & C.-C. Jay Kuo (2020). "Graph representation learning: a survey". In: *APSIPA Transactions on Signal and Information Processing* 9, e15.

Chen, Tianqi & Carlos Guestrin (2016). "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16, pp. 785–794.

Chen, Zuohui, Yao Lu, Wen Yang, Qi Xuan & Xiaoniu Yang (2021). "Graph-Based Similarity of Neural Network Representations". In: *ArXiv*.

Chib, Pranav Singh & Pravendra Singh (2023). "Recent advancements in end-to-end autonomous driving using deep learning: A survey". In: *IEEE Transactions on Intelligent Vehicles*.

Cimpoi, Mircea, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed & Andrea Vedaldi (2014). "Describing Textures in the Wild". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3606–3613.

Cui, Yin, Yang Song, Chen Sun, Andrew Howard & Serge Belongie (2018). "Large scale fine-grained categorization and domain-specific transfer learning". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4109–4118.

Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li & Li Fei-Fei (2009). "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255.

Deshpande, A., A. Achille, Avinash Ravichandran, Hao Li, L. Zancato, Charless C. Fowlkes, Rahul Bhotika, Stefano Soatto & P. Perona (2021). "A linearized framework and a new benchmark for model selection for fine-tuning". In: *ArXiv pre-print*.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee & Kristina Toutanova (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186.

Ding, Nan, Xi Chen, Tomer Levinboim, Soravit Changpinyo & Radu Soricut (2022). "PACTran: PAC-Bayesian Metrics for Estimating the Transferability of Pretrained Models to Classification Tasks". In: *Computer Vision – ECCV 2022*. Vol. 13694, pp. 252–268.

Dong, Qishi, Awais Muhammad, Fengwei Zhou, Chuanlong Xie, Tianyang Hu, Yongxin Yang, Sung-Ho Bae & Zhenguo Li (2022). "Zood: Exploiting model zoo for out-of-distribution generalization". In: *Advances in Neural Information Processing Systems* 35, pp. 31583–31598.

Dong, Shi, Ping Wang & Khushnood Abbas (2021). "A survey on deep learning and its applications". In: *Computer Science Review* 40, p. 100379.

Dosovitskiy, Alexey, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit & Neil Houlsby (2021). "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*.

Dwivedi, Kshitij, Jiahui Huang, Radoslaw Martin Cichy & Gemma Roig (2020). "Duality Diagram Similarity: A Generic Framework for Initialization Selection in Task Transfer Learning". In: *Computer Vision – ECCV 2020*, pp. 497–513.

Dwivedi, Kshitij & Gemma Roig (2019). "Representation Similarity Analysis for Efficient Task Taxonomy & Transfer Learning". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12379–12388.

Eaton, Eric, Marie desJardins & Terran Lane (2008). "Modeling Transfer Relationships Between Learning Tasks for Improved Inductive Transfer". In: *Machine Learning and Knowledge Discovery in Databases*, pp. 317–332.

Engelen, Jesper E. van & Holger H. Hoos (2019). "A survey on semi-supervised learning". In: *Machine Learning* 109.2, pp. 373–440.

Fagin, Ronald, Ravi Kumar & D. Sivakumar (2003). "Comparing Top $k$ Lists". In: *SIAM Journal on Discrete Mathematics* 17.1, pp. 134–160.

Fei-Fei, Li, R. Fergus & P. Perona (2004). "Learning Generative Visual Models from Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories". In: *2004 Conference on Computer Vision and Pattern Recognition Workshop*, pp. 178–178.

Goerttler, Thomas & Klaus Obermayer (2024). "Unveiling the Dynamics of Transfer Learning Representations". In: *ICLR 2024 Workshop on Representational Alignment*.

Grover, Aditya & Jure Leskovec (2016). "node2vec: Scalable Feature Learning for Networks". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16, pp. 855–864.

Hai, Rihan, Christos Koutras, Christoph Quix & Matthias Jarke (2023). "Data Lakes: A Survey of Functions and Systems". In: *IEEE Transactions on Knowledge and Data Engineering* PP, pp. 1–20.

Hamilton, William L., Rex Ying & Jure Leskovec (2017). "Inductive representation learning on large graphs". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17, pp. 1025–1035.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren & Jian Sun (2016). "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778.

He, Xin, Kaiyong Zhao & Xiaowen Chu (2021). "AutoML: A survey of the state-of-the-art". In: *Knowledge-Based Systems* 212, p. 106622.

Hu, Edward J, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. (2021). "LoRA: Low-Rank Adaptation of Large Language Models". In: *International Conference on Learning Representations*.

Huang, Jiaji, Qiang Qiu & Kenneth Church (2021). "Exploiting a Zoo of Checkpoints for Unseen Tasks". In: *Advances in Neural Information Processing Systems*. Vol. 34, pp. 19423–19434.

Huang, Long-Kai, Junzhou Huang, Yu Rong, Qiang Yang & Ying Wei (2022). "Frustratingly Easy Transferability Estimation". In: *Proceedings of the 39th International Conference on Machine Learning*, pp. 9201–9225.

Ibrahim, Shibal, Natalia Ponomareva & Rahul Mazumder (2023). "Newer is Not Always Better: Rethinking Transferability Metrics, Their Peculiarities, Stability and Performance". In: *Machine Learning and Knowledge Discovery in Databases*. Vol. 13713, pp. 693–709.

Ilnicka, A. & G. Schneider (2023). "Designing molecules with autoencoder networks". In: *Nature Computational Science* 3.11, pp. 922–933.

Kandwal, Siddharth & Vibha Nehra (2024). "A Survey of Text-to-Image Diffusion Models in Generative AI". In: *2024 14th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*. IEEE, pp. 73–78.

Kendall, M. G. (1938). "A New Measure of Rank Correlation". In: *Biometrika* 30.1/2, pp. 81–93.

Khoshraftar, Shima & Aijun An (2024). "A Survey on Graph Representation Learning Methods". In: *ACM Transactions on Intelligent Systems and Technology* 15.1, 19:1–19:55.

Kipf, Thomas N & Max Welling (2016). *Semi-Supervised Classification with Graph Convolutional Networks*.

Kornblith, Simon, Mohammad Norouzi, Honglak Lee & Geoffrey Hinton (2019). "Similarity of Neural Network Representations Revisited". In: *Proceedings of the 36th International Conference on Machine Learning*, pp. 3519–3529.

Krause, Jonathan, Michael Stark, Jia Deng & Li Fei-Fei (2013). "3D Object Representations for Fine-Grained Categorization". In: *2013 IEEE International Conference on Computer Vision Workshops*, pp. 554–561.

Krizhevsky, Alex (2012). "Learning Multiple Layers of Features from Tiny Images". In: *University of Toronto*.

Kumari, Nupur, Richard Zhang, Eli Shechtman & Jun-Yan Zhu (2022). "Ensembling Off-the-shelf Models for GAN Training". In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10641–10652.

Lecun, Y., L. Bottou, Y. Bengio & P. Haffner (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

Li, Hao, Pratik Chaudhari, Hao Yang, Michael Lam, Avinash Ravichandran, Rahul Bhotika & Stefano Soatto (2020). "Rethinking the Hyperparameters for Fine-tuning". In: *International Conference on Learning Representations*.

Li, Hao, Charless Fowlkes, Hao Yang, Onkar Dabeer, Zhuowen Tu & Stefano Soatto (2023). "Guided Recommendation for Model Fine-Tuning". In: *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3633–3642.

Li, Qing, Weidong Cai, Xiaogang Wang, Yun Zhou, David Dagan Feng & Mei Chen (2014). "Medical image classification with convolutional neural network". In: *2014 13th International Conference on Control Automation Robotics & Vision (ICARCV)*, pp. 844–848.

Li, Yandong, Xuhui Jia, Ruoxin Sang, Yukun Zhu, Bradley Green, Liqiang Wang & Boqing Gong (2021). "Ranking neural checkpoints". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2663–2673.

Li, Z, R Hai, A Bozzon & A Katsifodimos (2022). "Metadata Representations for Queryable ML Model Zoos". In: *ICML 2022 Workshop: DataPerf Benchmarking Data for Data-Centric AI*.

Li, Ziyu, Mariette Schönfeld, Wenbo Sun, Marios Fragkoulis, Rihan Hai, Alessandro Bozzon & Asterios Katsifodimos (2023). "Optimizing ML Inference Queries Under Constraints". In: *Web Engineering*. Lecture Notes in Computer Science, pp. 51–66.

Li, Ziyu, Hilco van der Wilk, Danning Zhan, Megha Khosla, Alessandro Bozzon & Rihan Hai (2024). "Model Selection with Model Zoo via Graph Learning". In: *2024 IEEE 40th International Conference on Data Engineering (ICDE)*.

Lin, Haowei, Baizhou Huang, Haotian Ye, Qinyu Chen, Zihao Wang, Sujian Li, Jianzhu Ma, Xiaojun Wan, James Zou & Yitao Liang (2024). "Selecting Large Language Model to Fine-tune via Rectified Scaling Law". In: *arXiv preprint*.

Liu, Renming, Matthew Hirn & Arjun Krishnan (2023). "Accurately modeling biased random walks on weighted networks using *node2vec+*". In: *Bioinformatics* 39.1, btad047.

Liu, Zhen, Wenbo Zuo, Dongning Zhang & Chuan Zhou (2024). "Self-Attention Enhanced Auto-Encoder for Link Weight Prediction With Graph Compression". In: *IEEE Transactions on Network Science and Engineering* 11.1, pp. 89–99.

*LLaMA* (2024). *LLaMA: Open and Efficient Foundation Language Models - Meta Research*. nl. URL: https://research.facebook.com/publications/llama-open-and-efficient-foundation-language-models/ (visited on 03/25/2024).

Loshchilov, Ilya & Frank Hutter (2018). "Decoupled Weight Decay Regularization". In: *International Conference on Learning Representations*.

Luo, Yifei, Minghui Xu & Deyi Xiong (2022). "CogTaskonomy: Cognitively Inspired Task Taxonomy Is Beneficial to Transfer Learning in NLP". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 904–920.

Martínez, Víctor, Fernando Berzal & Juan-Carlos Cubero (2017). "A Survey of Link Prediction in Complex Networks". In: *ACM Computing Surveys* 49.4, pp. 1–33.

Meiseles, Amiel & Lior Rokach (2020). "Source Model Selection for Deep Learning in the Time Series Domain". In: *IEEE Access* 8, pp. 6190–6200.

Mikolov, Tomas, Kai Chen, Greg Corrado & Jeffrey Dean (2013a). "Efficient estimation of word representations in vector space". In: *arXiv preprint*.

Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S Corrado & Jeff Dean (2013b). "Distributed Representations of Words and Phrases and their Compositionality". In: *Advances in Neural Information Processing Systems*. Vol. 26.

Mitchell, Margaret, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji & Timnit Gebru (2019). "Model Cards for Model Reporting". In: *Proceedings of the Conference on Fairness, Accountability, and Transparency*. FAT* '19, pp. 220–229.

Nargesian, Fatemeh, Ken Q. Pu, Erkang Zhu, Bahar Ghadiri Bashardoost & Renée J. Miller (2020). "Organizing Data Lakes for Navigation". In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pp. 1939–1950.

Netzer, Yuval, Tao Wang, Adam Coates, Alessandro Bissacco, Baolin Wu, Andrew Y Ng, et al. (2011). "Reading digits in natural images with unsupervised feature learning". In: *NIPS workshop on deep learning and unsupervised feature learning*. Vol. 2011. 5. Granada, Spain, p. 7.

Nguyen, Cuong, Tal Hassner, Matthias Seeger & Cedric Archambeau (2020). "LEEP: A New Measure to Evaluate Transferability of Learned Representations". In: *Proceedings of the 37th International Conference on Machine Learning*, pp. 7294–7305.

Nilsback, Maria-Elena & Andrew Zisserman (2008). "Automated Flower Classification over a Large Number of Classes". In: *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pp. 722–729.

Niu, Shuteng, Yongxin Liu, Jian Wang & Houbing Song (2020). "A Decade Survey of Transfer Learning (2010–2020)". In: *IEEE Transactions on Artificial Intelligence* 1.2, pp. 151–166.

Nori, N., D. Bollegala & M. Ishizuka (2011). "Interest prediction on multinomial, time-evolving social graphs". In: pp. 2507–2512.

OpenAI et al. (Mar. 2024). *GPT-4 Technical Report*. arXiv:2303.08774 [cs]. DOI: 10.48550/arXiv.2303.08774. URL: http://arxiv.org/abs/2303.08774 (visited on 03/25/2024).

Pan, Sinno Jialin & Qiang Yang (2010). "A Survey on Transfer Learning". In: *IEEE Transactions on Knowledge and Data Engineering* 22.10, pp. 1345–1359.

Pandy, Michal, Andrea Agostinelli, Jasper Uijlings, Vittorio Ferrari & Thomas Mensink (2022). "Transferability Estimation using Bhattacharyya Class Separability". In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9162–9172.

Pang, Bo & Lillian Lee (2005). "Seeing Stars: Exploiting Class Relationships for Sentiment Categorization with Respect to Rating Scales". In: *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pp. 115–124.

Parkhi, Omkar M, Andrea Vedaldi, Andrew Zisserman & C. V. Jawahar (2012). "Cats and dogs". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3498–3505.

Pearson, Karl (1895). "Note on Regression and Inheritance in the Case of Two Parents". In: *Proceedings of the Royal Society of London* 58, pp. 240–242.

Perozzi, Bryan, Rami Al-Rfou & Steven Skiena (2014). "DeepWalk: online learning of social representations". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. KDD '14, pp. 701–710.

Puigcerver, Joan, Carlos Riquelme Ruiz, Basil Mustafa, Cedric Renggli, André Susano Pinto, Sylvain Gelly, Daniel Keysers & Neil Houlsby (2020). "Scalable Transfer Learning with Expert Models". In: *International Conference on Learning Representations*.

Raiaan, Mohaimenul Azam Khan, Md Saddam Hossain Mukta, Kaniz Fatema, Nur Mohammad Fahad, Sadman Sakib, Most Marufatul Jannat Mim, Jubaer Ahmad, Mohammed Eunus Ali & Sami Azam (2024). "A review on large Language Models: Architectures, applications, taxonomies, open issues and challenges". In: *IEEE Access*.

Renggli, Cedric, Xiaozhe Yao, Luka Kolar, Luka Rimanic, Ana Klimovic & Ce Zhang (2022). "SHiFT: an efficient, flexible search engine for transfer learning". In: *Proceedings of the VLDB Endowment* 16.2, pp. 304–316.

Ridnik, Tal, Emanuel Ben-Baruch, Asaf Noy & Lihi Zelnik-Manor (2021). "ImageNet-21K Pretraining for the Masses". In: *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.

Rosenstein, Michael T, Zvika Marx, Leslie Pack Kaelbling & Thomas G Dietterich (2005). "To transfer or not to transfer". In: *NIPS 2005 workshop on transfer learning*. Vol. 898. 3.

Scarselli, Franco, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner & Gabriele Monfardini (2009). "The Graph Neural Network Model". In: *IEEE Transactions on Neural Networks* 20.1, pp. 61–80.

Schlegel, Marius & Kai-Uwe Sattler (2023). "Management of machine learning lifecycle artifacts: A survey". In: *ACM SIGMOD Record* 51.4, pp. 18–35.

Shao, Wenqi, Xun Zhao, Yixiao Ge, Zhaoyang Zhang, Lei Yang, Xiaogang Wang, Ying Shan & Ping Luo (2022). "Not All Models Are Equal: Predicting Model Transferability in a Self-challenging Fisher Space". In: *Computer Vision – ECCV 2022*. Vol. 13694, pp. 286–302.

Shu, Yang, Zhi Kou, Zhangjie Cao, Jianmin Wang & Mingsheng Long (2021). "Zoo-Tuning: Adaptive Transfer from A Zoo of Models". In: *Proceedings of the 38th International Conference on Machine Learning*, pp. 9626–9637.

Smith, Leslie N. (2017). "Cyclical Learning Rates for Training Neural Networks". In: *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 464–472.

Song, Jie, Yixin Chen, Xinchao Wang, Chengchao Shen & Mingli Song (2019). "Deep Model Transferability from Attribution Maps". In: *Advances in Neural Information Processing Systems*. Vol. 32.

Spearman, C. (1904). "The Proof and Measurement of Association between Two Things". In: *The American Journal of Psychology* 15.1, pp. 72–101.

Tan, Chuanqi, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang & Chunfang Liu (2018). "A Survey on Deep Transfer Learning". In: *Artificial Neural Networks and Machine Learning – ICANN 2018*. Lecture Notes in Computer Science, pp. 270–279.

Tan, Yang, Yang Li & Shao-Lun Huang (2021). "OTCE: A transferability metric for cross-domain cross-task representations". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 15779–15788.

Tan, Yang, Enming Zhang, Yang Li, Shao-Lun Huang & Xiao-Ping Zhang (2024). "Transferability-guided cross-domain cross-task transfer learning". In: *IEEE Transactions on Neural Networks and Learning Systems*.

Tran, Anh T, Cuong V Nguyen & Tal Hassner (2019). "Transferability and hardness of supervised classification tasks". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1395–1405.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser & Illia Polosukhin (2017). "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Vol. 30.

Veličković, Petar, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò & Yoshua Bengio (2018). "Graph Attention Networks". In: *International Conference on Learning Representations*.

Vu, Tu, Tong Wang, Tsendsuren Munkhdalai, Alessandro Sordoni, Adam Trischler, Andrew Mattarella-Micke, Subhransu Maji & Mohit Iyyer (2020). "Exploring and Predicting Transferability across NLP Tasks". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 7882–7926.

Wang, Alex, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy & Samuel Bowman (2018). "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding". In: *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pp. 353–355.

Wang, Shoujin, L Hu, Yan Wang, Xiangnan He, Quan Z Sheng, Mehmet A Orgun, Longbing Cao, Francesco Ricci & S Yu Philip (2021). "Graph Learning based Recommender Systems: A Review". In: pp. 4644–4652.

Wang, Zirui, Zihang Dai, Barnabas Poczos & Jaime Carbonell (2019). "Characterizing and Avoiding Negative Transfer". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11285–11294.

Wortsman, Mitchell, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S. Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith & Ludwig Schmidt (2022). "Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time". In: *Proceedings of the 39th International Conference on Machine Learning*, pp. 23965–23998.

Xia, Feng, Ke Sun, Shuo Yu, Abdul Aziz, Liangtian Wan, Shirui Pan & Huan Liu (2021). "Graph Learning: A Survey". In: *IEEE Transactions on Artificial Intelligence* 2.2, pp. 109–127.

You, Kaichao, Yong Liu, Jianmin Wang & Mingsheng Long (2021). "LogME: Practical Assessment of Pre-trained Models for Transfer Learning". In: *Proceedings of the 38th International Conference on Machine Learning*, pp. 12133–12143.

Yu, Xin, Zhiding Yu & Srikumar Ramalingam (2018). "Learning Strict Identity Mappings in Deep Residual Networks". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4432–4440.

Zhang, Daokun, Jie Yin, Xingquan Zhu & Chengqi Zhang (2020). "Network Representation Learning: A Survey". In: *IEEE Transactions on Big Data* 6.1, pp. 3–28.

Zhang, Yi-Kai, Ting-Ji Huang, Yao-Xiang Ding, De-Chuan Zhan & Han-Jia Ye (2024). "Model spider: Learning to rank pre-trained models efficiently". In: vol. 36.

Zhang, Yi & Zachary G. Ives (2020). "Finding Related Tables in Data Lakes for Interactive Data Science". In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. SIGMOD '20, pp. 1951–1966.

Zhuang, Fuzhen, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong & Qing He (2021). "A Comprehensive Survey on Transfer Learning". In: *Proceedings of the IEEE* 109.1, pp. 43–76.

Zitnik, Marinka, Monica Agrawal & Jure Leskovec (2018). "Modeling polypharmacy side effects with graph convolutional networks". In: *Bioinformatics* 34.13, pp. i457–i466.

# A

# Benchmark Suite Code Samples

In this chapter of the appendix, we provide some code samples on how our system can be used for new experiment setups and can be easily extended to new baselines. Figure A.1 shows how we configure loading datasets. While text and image classification datasets can be loaded using HuggingFace into a consistent interface, their input and label keys are not consistent. Our system adds functionality to easily define configuration of datasets. This includes a source on where it should be loaded from. This can currently be either local or from HuggingFace. The rest of the configuration lies in defining which key to use as the input and which as the label for classification.

## A.1. Dataset Configuration

```
    "glue": {
    "source": "huggingface",
    "tasks": {
        "cola": {
            "all_feature_key": ["sentence"],
            "label_key": "label"
        },
        "mnli": {
            "all_feature_key": ["premise", "hypothesis"]
        },
        "sst2": {...},
        ...
    }
}
```

**Figure A.1:** Example configuration for datasets. This example shows small changes to be made when configuring new datasets to load.

## A.2. Collecting Baseline Transferability Scores

Another important component of our system collects transferability estimation scores that were proposed by related works we discussed in Chapter 2. We currently support scores by a few of the most competitive source embedding methods, including the ones used for our own experiments. However, this can be easily extended to add new methods, as shown in Figure A.2.

```
class TransferabilityEstimatorSourceEmbedding:
    def __init__(
            self,
            dataset: BaseDataset,
            model: PreTrainedModel,
            all_baseline: list,
            args: argparse.Namespace
    ):
        self.dataset = dataset
        self.model = model
        self.all_baseline = all_baseline
        self.args = args

    def score(self):
        features_tensor, labels_tensor, _ = extract_features(self.dataset.train_loader, self.model)

        for baseline_method in self.all_baseline:
            if baseline_method == TransferabilityMethod.LOG_ME:
                from .methods.logme import LogME
                metric = LogME()
            elif baseline_method == TransferabilityMethod.NLEEP:
                from .methods.nleep import NLEEP
                metric = NLEEP()
            elif baseline_method == TransferabilityMethod.PARC:
                from .methods.parc import PARC
                metric = PARC(TransferabilityDistanceFunction.CORRELATION)
            elif baseline_method == TransferabilityMethod.H_SCORE:
                from .methods.hscore import HScore
                metric = HScore()
            elif baseline_method == TransferabilityMethod.REG_H_SCORE:
                from .methods.hscore_reg import HScoreR
                metric = HScoreR()
            else:
                raise Exception(f"Unexpected TransferabilityMetric: {baseline_method}")

            score = metric.score(features_tensor, labels_tensor)
```

**Figure A.2:** Code sample from our benchmark suite that shows how it can be easily extended to add new baselines.

## A.3. Collecting Fine-tuning Performances

Figure A.3 gives an example of our main function to fine-tune a pre-trained model using a new target dataset. What can be seen is how we use our implementation to load HuggingFace datasets and pre-trained models and do the required configuration for fine-tuning, such as defining the hyperparameters and indicating whether to enable experiment tracking.

```
def main(args: argparse.Namespace):
    if args.task_type == TaskType.IMAGE_CLASSIFICATION:
        dataset = HuggingFaceDatasetImage.load(
            dataset_path=args.dataset_path,
            dataset_name=args.dataset_name,
            batch_size=args.batch_size,
            image_processor=AutoImageProcessor.from_pretrained(args.model_name)
        )
        config = AutoConfig.from_pretrained(
            args.model_name,
            num_labels=len(dataset.all_class),
            i2label={label: str(i) for i, label in enumerate(dataset.all_class)},
            label2id={str(i): label for i, label in enumerate(dataset.all_class)},
            finetuning_task=args.task_type.value,
        )
        model = AutoModelForImageClassification.from_pretrained(
            args.model_name,
            config=config,
            ignore_mismatched_sizes=True,
        )
    elif args.task_type == TaskType.SEQUENCE_CLASSIFICATION:
        dataset = HuggingFaceDatasetText.load(
            dataset_path=args.dataset_path,
            dataset_name=args.dataset_name,
            batch_size=args.batch_size,
            tokenizer=AutoTokenizer.from_pretrained(args.model_name)
        )
        config = AutoConfig.from_pretrained(
            args.model_name,
            num_labels=len(dataset.all_class),
            i2label={label: str(i) for i, label in enumerate(dataset.all_class)},
            label2id={str(i): label for i, label in enumerate(dataset.all_class)},
            finetuning_task=args.task_type.value,
        )
        model = AutoModelForSequenceClassification.from_pretrained(
            args.model_name,
            config=config,
            ignore_mismatched_sizes=True,
        )
    else:
        raise Exception(f"Unexpected task_type: {args.task_type}")

    all_training_argument = TrainingArguments(
        output_dir=os.path.join(get_root_path_string(), "models"),
        num_train_epochs=args.num_train_epochs,
        lr_scheduler_type=args.lr_scheduler_type,
        learning_rate=args.learning_rate,
        per_device_train_batch_size=args.batch_size,
        per_device_eval_batch_size=args.batch_size,
        gradient_accumulation_steps=args.gradient_accumulation_steps,
        fp16=False,
        seed=args.seed,
        push_to_hub=args.push_to_hub,
        push_to_hub_organization=args.push_to_hub_organization,
    )
    trainer = AccelerateTrainer(model, dataset, all_training_argument, args.task_type, args)
    trainer.train()
```

**Figure A.3:** Code sample for using our system to fine-tune a pre-trained model. Includes examples on how to load datasets and pre-trained models and use our trainer class to fine-tune.