
Proceedings Paper:
Lu, H. orcid.org/0000-0002-0349-2181, Liu, X., Zhou, S. et al. (6 more authors) (2022) PyKale: Knowledge-aware machine learning from multiple sources in Python. In: Hasan, M.A. and Xiong, L., (eds.) CIKM '22: Proceedings of the 31st ACM International Conference on Information & Knowledge Management. CIKM '22: The 31st ACM International Conference on Information and Knowledge Management, 17-21 Oct 2022, Atlanta GA USA. ACM , pp. 4274-4278. ISBN 9781450392365

https://doi.org/10.1145/3511808.3557676

PyKale: Knowledge-Aware Machine Learning from Multiple Sources in Python

Haiping Lu
The University of Sheffield
h.lu@sheffield.ac.uk

Xianyuan Liu
University of Chinese Academy of Sciences
liuxianyuan16@mails.ucas.ac.cn

Shuo Zhou
The University of Sheffield
shuo.zhou@sheffield.ac.uk

Robert Turner
The University of Sheffield
r.d.turner@sheffield.ac.uk

Peizhen Bai
The University of Sheffield
pbai2@sheffield.ac.uk

Raivo E Koot
The University of Sheffield
raivokoot@gmail.com

Mustafa Chasmai
Indian Institute of Technology Delhi
cs1190341@iitd.ac.in

Lawrence Schobs
The University of Sheffield
laschobs1@sheffield.ac.uk

Hao Xu
Queen's University, Canada
xu.hao@queensu.ca

ABSTRACT

PyKale is a **Py**thon library for **K**nowledge-**a**ware machine **le**arning from multiple sources of data to enable/accelerate interdisciplinary research. It embodies *green machine learning* principles to reduce repetitions/redundancy, reuse existing resources, and recycle learning models across areas. We propose a *pipeline*-based application programming interface (API) so all machine learning workflows follow a standardized six-step pipeline. PyKale focuses on leveraging knowledge from multiple sources for accurate and interpretable prediction, particularly multimodal learning and transfer learning. To be more accessible, it separates code and configurations to enable non-programmers to configure systems without coding. PyKale is officially part of the PyTorch ecosystem and includes interdisciplinary examples in bioinformatics, knowledge graph, image/video recognition, and medical imaging: https://pykale.github.io/.

KEYWORDS

machine learning, multimodal learning, transfer learning, PyTorch

ACM Reference Format:
Haiping Lu, Xianyuan Liu, Shuo Zhou, Robert Turner, Peizhen Bai, Raivo E Koot, Mustafa Chasmai, Lawrence Schobs, and Hao Xu. 2022. PyKale: Knowledge-Aware Machine Learning from Multiple Sources in Python. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM '22), October 17–21, 2022, Atlanta, GA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3511808.3557676

1 INTRODUCTION

A growing number of researchers hope to solve real-world interdisciplinary problems using machine learning (ML). However, navigating through the abundant choices and variety of ML software is not trivial. Solving complex real-world problems often involves analyzing multiple sources of data, e.g., multiple modalities, multiple domains, and multiple knowledge bases. Most ML software packages are developed with a specific domain of application in mind. Popular generic packages such as scikit-learn, PyTorch, and TensorFlow focus on generic frameworks and need additional development to support interdisciplinary research, e.g., with both flexible configurations and high-level integration.

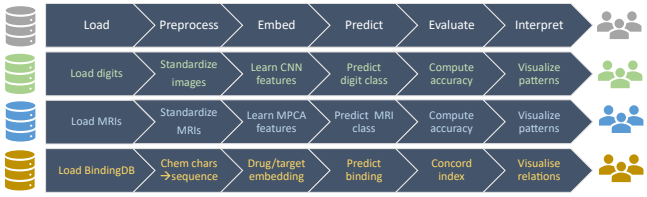

Figure 1: PyKale has a unified pipeline-based API so that all ML workflows follow a standardized six-step pipeline.

In this paper, we propose *PyKale*, an open-source Python library to enable and accelerate interdisciplinary research via knowledge-aware multimodal learning and transfer learning on graphs, images, and videos. It aims to fill the gaps between rich data sources, abundant ML libraries, and eager interdisciplinary researchers, by focusing on leveraging knowledge from multiple sources for accurate and interpretable prediction. It considers both multimodal learning and transfer learning under a common framework of learning from multiple sources. It makes latest ML tools more accessible to accelerate their development. The name of the library consists of **Py** for **Py**thon, and **Kale** for **K**nowledge-**a**ware **le**arning.

PyKale proposes a novel *pipeline-based* application programming interface (API) so that all ML workflows follow a standardized six-step pipeline as shown in Fig. 1. It embodies our **green ML** concepts of reducing repetitions and redundancy (Fig. 2(a)), reusing existing resources (Fig. 2(b)), and recycling learning models across areas (Fig. 2(c)). It has examples in bioinformatics, knowledge graph, image/video recognition, and medical imaging. It was motivated by needs in healthcare applications and considers healthcare as a primary domain of usage. It is largely built on PyTorch and has

CIKM '22, October 17–21, 2022, Atlanta, GA, USA

ACM ISBN 978-1-4503-9236-5/22/10…$15.00
https://doi.org/10.1145/3511808.3557676

(a) Reduce (b) Reuse (c) Recycle (d) The PyKale logo

Figure 2: Green machine learning concepts in PyKale.

been approved officially to join the PyTorch ecosystem https://pytorch.org/ecosystem/. The logo in Fig. 2(d) reflects the above characteristics using multiple (three) kale leaves.

PyKale is available at https://github.com/pykale/pykale under an MIT license, with many community-engaging features. PyKale can be installed via `pip install pykale` from the Python Package Index (PyPI). The primary targeted users are researchers and practitioners who have experience in Python/PyTorch programming and need to apply/develop ML systems to integrate data from multiple sources for prediction tasks in interdisciplinary areas such as healthcare. This paper refers to release version **0.1.1**.

2 RELATED WORK

We have learned from numerous libraries in the public domain to build PyKale. Here, we can only briefly mention several that have been particularly influential or relevant.

PyKale aims to fill the gap within the *PyTorch ecosystem* to support more interdisciplinary research that integrates multiple data sources. Therefore, we make extensive usage of existing libraries from the PyTorch ecosystem to reduce duplicated implementation, including PyTorch Lightning [8], TensorLy [17], TorchVision [26], and PyTorch Geometric [9]. We also learned from MONAI [25], GPyTorch [12], Kornia [34], and TorchIO [32].

PyTorch-based libraries on *multimodal/transfer learning* include the MultiModal Framework (MMF) [38], the Transfer-Learning-Library [15], Cornac [35], the ADA ((Yet) Another Domain Adaptation) library [41], and the Multimodal-Toolkit [13].

PyKale frames multimodal learning and transfer learning under one roof of knowledge-aware ML from multiple sources with a unified pipeline-based API to support interdisciplinary research rather than just popular vision/language tasks. PyKale differs from these existing libraries not only in the API design, but also in the number of modalities supported, the scientific fields covered, and interdisciplinarity. Part of PyKale refactored ADA to our pipeline-based API, with the sources indicated at the top of respective files.

3 PYKALE DESIGN

3.1 Green machine learning

Green ML is a scarcely used term referring to energy-efficient computing [6, 11, 19, 36]. We propose a *different*, new *green ML* perspective for ML software development with the **3R** guiding principles formulated below, by extending similar principles in standard software engineering practices to machine learning.

Reduce *repetition/redundancy*: 1) Refactor code to standardize workflow and enforce styles, e.g., we refactored Deep Drug-Target binding Affinity (DeepDTA) [28] into PyKale API (bottom of Fig. 1); 2) Identify and remove duplicated functionalities, e.g., we built data loading API for popular datasets to avoid repetition.

Reuse *existing resources*: 1) Reuse the same ML pipeline for different data/applications, e.g., using the same Multilinear Principal Component Analysis (MPCA) pipeline for gait [24], brain [39], and heart [2, 40]; 2) Reuse existing libraries (e.g., scikit-learn) for available functionalities rather than implementing them again.

Recycle *learning models across areas*: 1) Identify commonalities between applications, e.g., the similarity between commercial recommender systems (user-item interactions) and drug discovery (drug-target interactions); 2) Recycle models for one application to another, e.g., from recommender system [3] to drug discovery [45].

Although based on existing practices, this new formulation offers a new perspective to focus on core principles of standardization and minimalism. It has guided us to design a unique pipeline-based API to unify workflow, break barriers between areas and applications, and cross boundaries to fuse existing ideas and nurture new ideas.

3.2 Pipeline-Based API

Inspired by the convenience of ML pipelines in ML libraries Spark MLlib [27] and scikit-learn [30], we design PyKale with a pipeline-based API as shown in Fig. 1. This design has six key steps and embodies our green ML principles by organizing code along a standardized ML pipeline to identify commonalities, reduce redundancy, and minimize cognitive overhead. In the following, we explain our unified API by starting with what the input and output are.

`Load`. The `kale.loaddata` module mainly takes source paths (local or online) as the input and constructs dataloaders for datasets as the output. It aims to load data for input to the ML system/pipeline.

`Preprocess`. The `kale.prepdata` module takes the loaded raw input data as input and preprocesses (transforms) them into a suitable representation for the following ML modules. Preprocessing steps include data normalization, augmentation, and other transformations of data representation *not* involving ML. Its submodules are typically imported in `kale.loaddata`.

`Embed`. The `kale.embed` module takes preprocessed, normalized data representations to *learn* new representations in a new space as the output. It includes dimensionality reduction (feature extraction) algorithms, such as MPCA [24] and Convolutional Neural Networks (CNNs). They can be viewed as encoders or embedding functions that learn suitable representations from data. This is an ML module.

`Predict`. The `kale.predict` module takes the learned (or preprocessed, if skipping `kale.embed`) representations to predict a desired target value as the output. Thus, this module provides prediction functions or decoders that learn a mapping from the input representation to a target prediction. This is also an ML module.

`Evaluate`. The `kale.evaluate` module evaluates the prediction performance using some metrics. We reuse metrics from other libraries (e.g., `sklearn.metrics`) and only implement metrics not commonly available, such as the Concordance Index (CI) [1] for measuring the proportion of concordant pairs.

`Interpret`. The `kale.interpret` module provides functions for interpreting learned features, models, or prediction results/outputs, e.g., via further analysis or visualization. We only implement functions not commonly available, e.g., visualizing trained model weights, showing output distribution, and displaying multiple images.

`Pipeline`. The `kale.pipeline` module provides mature, off-the-shelf ML pipelines for "plug-in usage". Its submodules typically specify an ML workflow by combining several other modules.

`Utilities`. The `kale.utils` module provides common utility functions, e.g., setting seeds, logging results, or downloading data.

3.3 Machine learning models

PyKale focuses on integrating four categories of ML models.

Multimodal learning. To support learning from data of multiple modalities, we leveraged the rich PyTorch ecosystem to build APIs that support learning from each individual modality (e.g., graph, image, and video) into PyKale first. Then, we added support for learning from heterogeneous data sources in data integration. PyKale built a DeepDTA [28] pipeline `kale.pipeline.deepdta` that learns from drug/target data represented as sequences/vectors. PyKale also implemented the GripNet [45] `kale.embed.gripnet` for link prediction and data integration on heterogeneous knowledge graphs, with an example `polypharmacy_gripnet` on a bioinformatics knowledge graph [49, 50].

Transfer learning. In transfer learning, PyKale currently focuses on domain adaptation [4, 29]. We largely inherited the excellent, modular architecture from ADA [41] to build important semi-supervised/unsupervised domain adaptation algorithms including Domain-Adversarial Neural Networks (DANN) [10], Conditional adversarial Domain Adaptation Networks (CDAN) [22], Deep Adaptation Networks (DAN) [21], Joint Adaptation Networks (JAN) [23], and Wasserstein Distance Guided Representation Learning (WDGRL) [37]. We extended them to videos and further implemented two multi-source and two independence-based domain adaptation algorithms: Moment Matching for Multi-Source Domain Adaptation (M^3SDA) [31] and Multiple Feature Spaces Adaptation Network (MFSAN) [48], and Maximum Independence Domain Adaptation (MIDA) [46] and Covariate Independence Regularized Least Squares (CoIRLS) classifier [47]. Thus, PyKale has three domain adaptation pipelines: `domain_adapter`, `multi_domain_adapter`, and `video_domain_adapter` in `kale.pipeline`.

Deep learning. PyKale builds Deep Neural Networks (DNNs) upon the PyTorch API. Current implementations include CNNs [18] / 3D CNNs [7, 42], Graph Convolutional Networks (GCNs) [16], and attention-based networks such as transformers [44] and squeeze-and-excitation networks [14]. We use TorchVision [26], PyTorch Geometric [9], and PyTorch Lightning [8] in our implementation.

Dimensionality reduction. PyKale built a Python version of the MPCA algorithm [24] at `kale.embed.factorization` and an MPCA-based pipeline at `kale.pipeline.mpca_trainer` using the scikit-Learn [30] and TensorLy [17] libraries. This pipeline has been used for interpretable prediction in gait recognition from video sequences [24], cardiovascular disease diagnosis [40] and prognosis [2, 43] from cardiac magnetic resonance imaging (MRI), and brain state classification using functional MRI (fMRI) [39].

3.4 Software engineering

The PyKale team includes ML researchers and Research Software Engineers (RSEs). We have adopted good software engineering practices in a research context, often based on other libraries, particularly those in the PyTorch ecosystem. The PyKale GitHub repository

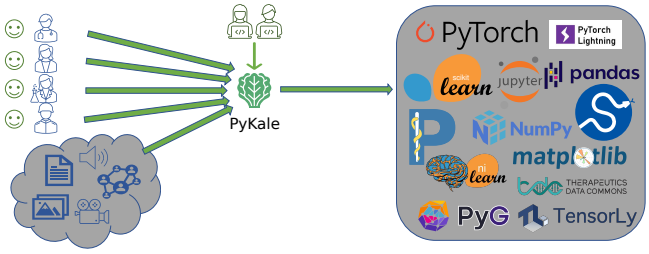

Figure 3: PyKale aims to make abundant ML software accessible for interdisciplinary research, even to non-programmers, and support data of multiple modalities under one roof.

provides three-tier contributing guidelines at https://github.com/pykale/pykale/blob/main/.github/CONTRIBUTING.md and multi-option installation instructions at https://pykale.readthedocs.io/en/latest/installation.html. It uses Sphinx (https://www.sphinx-doc.org/) for automatic html documentation building from "docstrings", and PyTest for testing, with nightly runs and currently achieving 90+% test coverage. Continuous integration (CI) is implemented using GitHub workflows/actions at https://github.com/pykale/pykale/tree/main/.github/workflows, including pre-commit checks, linting, documentation building, project assignment, PyPI release, PyTest tests, and Codecov code coverage report. Merging into the `main` branch requires passing several required checks and at least one approval. To maintain a small repository size (now <1MB), we store data in a separate repository at https://github.com/pykale/data.

4 PYKALE USAGE

PyKale aims to make abundant ML software accessible for interdisciplinary research, even to non-programmers, as depicted in Fig. 3. It lowers the barriers to entry with standardized examples/tutorials.

Usage of pipeline-based API in examples. PyKale examples are highly standardized. Each example typically has three essential modules (`main.py`, `config.py`, `model.py`), one optional directory (`configs`), and other optional modules (e.g., `trainer.py`): `main.py` is the main module to be run, showing the main workflow; `config.py` is the configuration module that sets up the default configuration such as the data, prediction problem, and hyper-parameters; `configs` is the directory to place customized configurations for individual runs, where `.yaml` files (see below) are used for this purpose; `model.py` is the model module to define the ML model and configure its training parameters; `trainer.py` is the trainer module to define the training and testing workflow, which is not needed if using PyTorch Lightning.

Building new modules or projects. Users can build new modules or projects following these steps. ***Step 1 - Examples***: Choose one of the examples of the users' interest to browse through the configuration, main, and model modules, download the data if needed, and run the example following instructions in the example's README. ***Step 2a - New model***: To develop new ML models under PyKale, define the blocks in the users' pipeline to figure out specific methods for data loading, preprocessing data, embedding (encoder/representation), prediction (decoder), evaluation, and interpretation, and then modify existing pipelines with the users' customized blocks or build a new pipeline with `pykale` blocks and

blocks from other libraries. ***Step 2b - New applications***: To develop new applications using PyKale, clarify the input data and the prediction target to find matching functionalities in `pykale` (request if not found), and tailor data loading, preprocessing, and evaluation (and interpretation if needed) to the users' application.

YAML configuration. PyKale examples configure an ML system using YAML [5] to improve *accessibility*. This is inspired by the usage of YAML in the GitHub package for the Isometric Network (ISONet) [33] (https://github.com/HaozhiQi/ISONet). As modern ML systems typically have many settings to configure, specifying many/all settings in command line or Python modules becomes difficult to manage/read. Using YAML greatly improves the readability and reproducibility, and makes configuration changes much easier, via a default configuration specified in `config.py` (e.g., https://github.com/pykale/pykale/blob/main/examples/digits_dann/config.py) and customized configurations specified in a respective `.yaml` file (e.g., https://github.com/pykale/pykale/blob/main/examples/digits_dann/configs/tutorial.yaml), which will be merged to overwrite the default setting at run time. Moreover, *non-programmer users* can learn only YAML configuration to make changes to model/experimental settings, making it highly *accessible*.

Notebook tutorials with Binder/Colab. We have ten real-world examples of PyKale usage at https://github.com/pykale/pykale/tree/main/examples. Tutorials without the need of any installation are helpful for new users to get familiar with the PyKale workflow and API. Therefore, we simplified typical examples into *interactive* Jupyter notebook tutorials so that each tutorial takes minutes instead of hours to run. This strikes a balance between computational requirements and runtime, without resorting to toy examples. Moreover, we set up cloud-based services with both Binder (https://mybinder.org/) and Google Colaboratory (Colab) for our notebook tutorials so that any users can run PyKale tutorials interactively without the need of any installation. We have released five such tutorials: https://pykale.readthedocs.io/en/latest/notebooks.html.

Scope of PyKale examples. PyKale currently has example applications from three areas: 1) Image/video recognition, e.g., classification of images (objects, digits) or videos (actions); 2) Bioinformatics/graph analysis: prediction of links between entities in knowledge graphs (BindingDB [20], BioSNAP-Decagon [50]); 3) Medical imaging: disease diagnosis from cardiac MRI or brain fMRI. These examples deal with graphs, images, and videos. Examples in computer vision applications such as image/video recognition are a good start for most users due to the popularity of vision applications and a low barrier to entry (e.g., no need for specific domain knowledge as in drug discovery). Models first developed in computer vision can be reused or recycled for other applications. Most data used in PyKale examples are real-world data frequently used in research papers. Thus, it may take quite some time to run these examples fully. For quick running and demonstration of the workflow, Jupyter notebook tutorials above should be used.

5 COMMUNITY ENGAGEMENT AND BEYOND

PyKale (https://github.com/pykale/pykale) is under an MIT license, a simple permissive license with minimal restrictions. It has an active discussion board (https://github.com/pykale/pykale/discussions) for open dialogues with users and a project board (https://github.com/pykale/pykale/projects) sharing the development process/plan with users. We also released five YouTube videos (https://www.youtube.com/results?search_query=pykale) to explain the motivation/principles behind PyKale. PyKale has detailed documentation at https://pykale.readthedocs.io/, generated automatically from https://github.com/pykale/pykale/tree/main/docs. PyKale provides highly accessible tutorials and examples in a consistent format (https://github.com/pykale/pykale/tree/main/examples), with detailed contributing guidelines (https://github.com/pykale/pykale/blob/main/.github/CONTRIBUTING.md) and change logs (https://github.com/pykale/pykale/blob/main/.github/CHANGELOG.md).

PyKale is an open-source project started in Jun. 2020, with the first PyPI release in Jan. 2021. It was officially approved to join the PyTorch ecosystem in Sep. 2021. PyKale was motivated by the growing needs for ML systems that can deal with multiple sources of data, particularly in interdisciplinary areas such as healthcare. For example, clinicians often need to make use of a combination of medical images (e.g., X-rays, CTs, MRIs), biological data (gene, DNA, RNA), and electronic health record for decision making.

To date, PyKale has built APIs supporting ML on graphs, images, and videos, with five mature pipelines implemented. APIs and examples on tabular data, audio data, and text data are in development. Developing projects involving multiple data sources takes considerably longer time than developing those involving a single data source. The current version of PyKale has two examples on multimodal learning involving heterogeneous drug/target data and four examples on domain adaptation for image/video data. These examples laid solid foundations for growing research in these areas and building more advanced examples.

6 CONCLUSIONS

PyKale is a Python library providing accessible machine learning from multiple sources of data for interdisciplinary research, particularly multimodal learning and transfer learning, named collectively as **K**nowledge-**a**ware machine **le**arning (Kale). Motivated by needs in healthcare applications (hence the acronym kale, a healthy vegetable), PyKale aims to make abundant ML software more accessible for interdisciplinary research, even to non-programmers. Building on existing practices, we proposed a new green ML perspective to reduce repetitions/redundancy, reuse existing resources, and recycle learning models across areas. Following such principles, we designed the PyKale API to be pipeline-based to unify all ML workflows into six standardized steps. This design can help break barriers between different areas/applications and facilitate the fusion and nurture of ideas across discipline boundaries. Moreover, we separate code and configurations so that non-programmers can configure ML systems without coding, greatly improving *accessibility*. PyKale also makes it easier to bring ML models developed in one area to another.

ACKNOWLEDGMENTS

This development is partially supported by the Wellcome Trust (grant 215799/Z/19/Z). We thank the support from David Jones and Will Furnass of the RSE team headed by Paul Richmond. We also thank many others who have contributed to the development of PyKale and/or encouraged us to keep moving forward.

REFERENCES

[1] Kartik Ahuja and Mihaela van der Schaar. 2019. Joint Concordance Index. In *Proceedings of the 2019 53rd Asilomar Conference on Signals, Systems, and Computers*. 2206–2213.

[2] Samer Alabed, Johanna Uthoff, Shuo Zhou, Pankaj Garg, Krit Dwivedi, Faisal Alandejani, Rebecca Gosling, Lawrence Schobs, Martin Brook, Yousef Shahin, et al. 2022. Machine learning cardiac-MRI features predict mortality in newly diagnosed pulmonary arterial hypertension. *European Heart Journal-Digital Health* 3, 2 (2022), 265–275.

[3] Peizhen Bai, Yan Ge, Fangling Liu, and Haiping Lu. 2019. Joint interaction with context operation for collaborative filtering. *Pattern Recognition* 88 (2019), 729–738.

[4] Shai Ben-David, John Blitzer, Koby Crammer, Fernando Pereira, et al. 2007. Analysis of representations for domain adaptation. In *Proceedings of the Advances in Neural Information Processing Systems*. 137–144.

[5] Oren Ben-Kiki, Clark Evans, and Brian Ingerson. 2009. Yaml ain't markup language (yaml™) version 1.1. *Working Draft 2008-05* 11 (2009).

[6] Antonio Candelieri, Riccardo Perego, and Francesco Archetti. 2021. Green machine learning via augmented Gaussian processes and multi-information source optimization. *Soft Computing* (2021), 1–13.

[7] Joao Carreira and Andrew Zisserman. 2017. Quo vadis, action recognition? a new model and the kinetics dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 6299–6308.

[8] William A Falcon and et al. 2019. PyTorch Lightning. *GitHub*. 3 (2019). https://github.com/PyTorchLightning/pytorch-lightning

[9] Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

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

[11] Eva García Martín. 2017. Energy efficiency in machine learning: A position paper. In *Proceedings of the 30th Annual Workshop of the Swedish Artificial Intelligence Society*, Vol. 137. 68–72.

[12] Jacob Gardner, Geoff Pleiss, Kilian Q Weinberger, David Bindel, and Andrew G Wilson. 2018. GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration. In *Proceedings of the Advances in Neural Information Processing Systems*, Vol. 31. 7587–7597.

[13] Georgian. 2020. Multimodal-Toolkit. *GitHub* (2020). https://github.com/georgian-io/Multimodal-Toolkit

[14] Jie Hu, Li Shen, and Gang Sun. 2018. Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 7132–7141.

[15] Junguang Jiang, Bo Fu, and Mingsheng Long. 2020. Transfer-Learning-library. *GitHub* (2020). https://github.com/thuml/Transfer-Learning-Library

[16] Thomas Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th International Conference on Learning Representations*.

[17] Jean Kossaifi, Yannis Panagakis, Anima Anandkumar, and Maja Pantic. 2019. TensorLy: Tensor Learning in Python. *Journal of Machine Learning Research* 20, 26 (2019), 1–6.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Proceedings of the Advances in Neural Information Processing Systems*. 1097–1105.

[19] Sun Yuan Kung. 2014. *Kernel methods and machine learning*. Cambridge University Press.

[20] Tiqing Liu, Yuhmei Lin, Xin Wen, R. Jorissen, and M. Gilson. 2007. BindingDB: a web-accessible database of experimentally determined protein–ligand binding affinities. *Nucleic Acids Research* 35 (2007), D198 – D201.

[21] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael Jordan. 2015. Learning transferable features with deep adaptation networks. In *Proceedings of the International Conference on Machine Learning*. 97–105.

[22] Mingsheng Long, Zhangjie Cao, Jianmin Wang, and Michael I Jordan. 2018. Conditional Adversarial Domain Adaptation. In *Proceedings of the Advances in Neural Information Processing Systems*, Vol. 31.

[23] Mingsheng Long, Han Zhu, Jianmin Wang, and Michael I Jordan. 2017. Deep transfer learning with joint adaptation networks. In *Proceedings of the International Conference on Machine Learning*. 2208–2217.

[24] Haiping Lu, Konstantinos N Plataniotis, and Anastasios N Venetsanopoulos. 2008. MPCA: Multilinear principal component analysis of tensor objects. *IEEE Transactions on Neural Networks* 19, 1 (2008), 18–39.

[25] Nic Ma, Wenqi Li, and Richard Brown. 2021. *Project-MONAI/MONAI: 0.5.3*. https://doi.org/10.5281/zenodo.4891800

[26] Sébastien Marcel and Yann Rodriguez. 2010. Torchvision the Machine-Vision Package of Torch. In *Proceedings of the 18th ACM International Conference on Multimedia*. 1485–1488.

[27] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. 2016. Mllib: Machine learning in apache spark. *Journal of Machine Learning Research* 17, 1 (2016), 1235–1241.

[28] Hakime Öztürk, E. Olmez, and Arzucan Özgür. 2018. DeepDTA: deep drug–target binding affinity prediction. *Bioinformatics* 34, 17 (2018), i821 – i829.

[29] Sinno Jialin Pan, Ivor W Tsang, James T Kwok, and Qiang Yang. 2010. Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks* 22, 2 (2010), 199–210.

[30] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[31] Xingchao Peng, Qinxun Bai, Xide Xia, Zijun Huang, Kate Saenko, and Bo Wang. 2019. Moment matching for multi-source domain adaptation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 1406–1415.

[32] Fernando Pérez-García, Rachel Sparks, and Sebastien Ourselin. 2020. TorchIO: a Python library for efficient loading, preprocessing, augmentation and patch-based sampling of medical images in deep learning. (2020). http://arxiv.org/abs/2003.04696

[33] Haozhi Qi, Chong You, Xiaolong Wang, Yi Ma, and Jitendra Malik. 2020. Deep isometric learning for visual recognition. In *Proceedings of the International Conference on Machine Learning*. 7824–7835.

[34] Edgar Riba, Dmytro Mishkin, Daniel Ponsa, Ethan Rublee, and Gary Bradski. 2020. Kornia: an open source differentiable computer vision library for pytorch. In *Proceedings of the IEEE Winter Conference on Applications of Computer Vision*. 3674–3683.

[35] Aghiles Salah, Quoc-Tuan Truong, and Hady W Lauw. 2020. Cornac: A Comparative Framework for Multimodal Recommender Systems. *Journal of Machine Learning Research* 21, 95 (2020), 1–5.

[36] Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. 2020. Green AI. *Commun. ACM* 63, 12 (2020), 54–63.

[37] Jian Shen, Yanru Qu, Weinan Zhang, and Yong Yu. 2018. Wasserstein distance guided representation learning for domain adaptation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[38] Amanpreet Singh, Vedanuj Goswami, Vivek Natarajan, Yu Jiang, Xinlei Chen, Meet Shah, Marcus Rohrbach, Dhruv Batra, and Devi Parikh. 2020. MMF: A multimodal framework for vision and language research. https://github.com/facebookresearch/mmf.

[39] Xiaonan Song, Lingnan Meng, Qiquan Shi, and Haiping Lu. 2015. Learning tensor-based features for whole-brain fMRI classification. In *Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention*. 613–620.

[40] Andrew J Swift, Haiping Lu, Johanna Uthoff, Pankaj Garg, Marcella Cogliano, Jonathan Taylor, Peter Metherall, Shuo Zhou, Christopher S Johns, Samer Alabed, et al. 2021. A machine learning cardiac magnetic resonance approach to extract disease features and automate pulmonary arterial hypertension diagnosis. *European Heart Journal-Cardiovascular Imaging* 22, 2 (2021), 236–245.

[41] Anne-Marie Tousch and Christophe Renaudin. 2020. (Yet) Another Domain Adaptation library. https://github.com/criteo-research/pytorch-ada

[42] Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. 2018. A closer look at spatiotemporal convolutions for action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 6450–6459.

[43] Johanna Uthoff, Samer Alabed, Andrew J Swift, and Haiping Lu. 2020. Geodesically Smoothed Tensor Features for Pulmonary Hypertension Prognosis Using the Heart and Surrounding Tissues. In *Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention*. 253–262.

[44] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan Gomez, Ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All You Need. In *Proceedings of the Advances in Neural Information Processing Systems*. 6000–6010.

[45] Hao Xu, Shengqi Sang, Peizhen Bai, Ruike Li, Laurence Yang, and Haiping Lu. 2022. GripNet: Graph Information Propagation on Supergraphs for Heterogeneous Graphs. *Pattern Recognition* (2022).

[46] Ke Yan, Lu Kou, and David Zhang. 2017. Learning domain-invariant subspace using domain features and independence maximization. *IEEE transactions on cybernetics* 48, 1 (2017), 288–299.

[47] Shuo Zhou. 2022. *Interpretable Domain-Aware Learning for Neuroimage Classification*. Ph.D. Dissertation. University of Sheffield.

[48] Yongchun Zhu, Fuzhen Zhuang, and Deqing Wang. 2019. Aligning Domain-Specific Distribution and Classifier for Cross-Domain Classification from Multiple Sources. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 5989–5996.

[49] Marinka Zitnik, Monica Agrawal, and Jure Leskovec. 2018. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics* 34, 13 (2018), i457–i466.

[50] Marinka Zitnik, Rok Sosič, Sagar Maheshwari, and Jure Leskovec. 2018. BioSNAP Datasets: Stanford Biomedical Network Dataset Collection.