Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

MSTIL: Multi-cue Shape-aware Transferable Imbalance Learning for effective graphic API recommendation☆

Rong Qin[a], Zeyu Wang[a], Sheng Huang[a,*], Luwen Huangfu[b,c,**]

[a] School of Big Data & Software Engineering, Chongqing University, Chongqing 400044, China
[b] Fowler College of Business, San Diego State University, San Diego, CA 92182, USA
[c] Center for Human Dynamics in the Mobile Age, San Diego State University, San Diego, CA 92182, USA

A R T I C L E   I N F O

Article history:
Received 20 April 2022
Received in revised form 21 January 2023
Accepted 15 February 2023
Available online 21 February 2023

Keywords:
API recommendation
Data augmentation
Transfer learning
Data visualization
Semantic similarity

A B S T R A C T

Application Programming Interface (API) recommendation based on graphs is a valuable task in the fields of data visualization and software engineering. However, this task was previously undefined until a recently published paper coining the task as Plot2API and utilizing a deep learning-based method named SPGNN. Compared to general image classification methods, this dedicated approach uses semantic parsing to exploit deep features and yields better performance. However, its performance declines sharply in unbalanced datasets, thus limiting its generalizability. To address this issue, we propose a method named Multi-cue Shape and software engineering-aware Transferable Imbalance Learning (MSTIL), consisting of three major components: Cross-Language Shape-Aware Plot Transfer Learning (CLSAPTL), Cross-Language API Semantic Similarity-based Data Augmentation (CLASSDA), and Imbalance Plot2API Learning (IPL). Motivated by the hierarchical classification of the graphs, CLSAPTL guides the model to learn the graphs' class hierarchy and thereby enabling the model to learn more transferable visual features. Given that a graph can be associated with multiple APIs and motivated by the fact that many APIs that exert similar functions in different languages have semantically similar names, CLASSDA leverages the samples of APIs with semantically similar names to assist in feature learning. Finally, inspired by the essence of softmax cross entropy loss, IPL alleviates the imbalances between positive and negative samples during training. We conduct our experiments on two public datasets. Extensive experimental results shows that MSTIL improves the performance of classic CNNs along with the state-of-the-art method, demonstrating its effectiveness. Specifically, MSTIL has an average relative mAP improvement of 12.94% across the models on all datasets.

1. Introduction

As easy-to-interpret and compelling representations of data, graphs are widely used by programming languages, and thus graphical Application Programming Interfaces (APIs) are part of core libraries found in most languages. Consequently, several APIs have been created to satisfy a wide range of graphical functions within any given programming language. However, considering the vast number of different graphic APIs one could implement, the task of selecting which API may be required in any given context becomes challenging for not only beginners, but also experienced professionals. As a result, programmers may seek help from many different sources, such as official documentation, video tutorials, and Stack Overflow discussions. Even with these resources, it is often time-consuming and difficult for programmers to determine which API is best suited for their application (Robillard, 2009). Therefore, the task of providing API recommendations from graphs is meaningful in the fields of data visualization and software engineering. It enables a variety of valuable applications, such as aiding beginners in graphic API selection and facilitating code conversion in agile development.

Nevertheless, traditional approaches mainly utilize source code and textual descriptions while ignoring a key piece of information, the graph itself. As many studies (Zhong et al., 2009; McMillan et al., 2011; Niu et al., 2017; Huang et al., 2018; He et al., 2021; Santos et al., 2017) have shown, traditional API recommendations use embeddings to model and translate between natural language queries and source code, which is a well-studied task in software engineering. It is inconvenient to pre-translate a

☆ Editor: Nicole Novielli.
* Corresponding author.
** Corresponding author at: Fowler College of Business, San Diego State University, San Diego, CA 92182, USA.
E-mail addresses: huangsheng@cqu.edu.cn (S. Huang), lhuangfu@sdsu.edu (L. Huangfu).

https://doi.org/10.1016/j.jss.2023.111650
0164-1212/© 2023 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

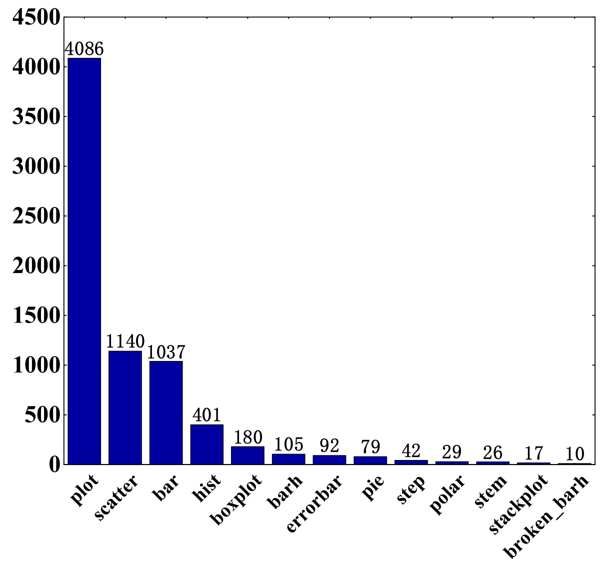


Fig. 1. Statistics on the number of samples contained in different APIs of Python-Plot13 dataset. It can be observed that there is a huge imbalance between the number of samples of each API.

Table 1
Statistics on the number of samples contained in different APIs of R-Plot32 dataset. Similarly, it can be observed that there is also a huge imbalance between the number of samples of each API.

API	#	API	#	API	#	API	#
bar	2111	bin2d	12	density	205	density_2d	4
map	90	jitter	105	boxplot	638	quantile	4
rug	20	smooth	395	segment	385	contour	21
hex	20	curve	8	dotplot	40	errorbar	335
step	38	line	2312	freqpoly	10	errorbarh	39
sf	16	spoke	4	crossbar	14	linerange	49
path	232	violin	46	polygon	303	pointrange	49
point	3665	raster	73	ribbon	223	histogram	387

image into code, or a textual description and then complete the recommended task in a text-to-text method, as the translation often requires excessive computation time and may introduce misinterpretation.

In contrast to traditional API recommendations (Zhong et al., 2009; McMillan et al., 2011; Niu et al., 2017; Huang et al., 2018; He et al., 2021; Santos et al., 2017) that mainly rely on the semantic parsing of texts, it would be useful to also exploit discriminative features relating to different types of graphs which appear in various forms, such as computer-generated or hand-drawn graphs, and directly recommend APIs in a Plot-to-API (Plot2API) manner. Intuitively, this strategy is more convenient, accurate, and effective than traditional API recommendation.

Despite the potential benefits of Plot2API, Plot2API is still in its infancy. The major reason is that Plot2API can suffer from overfitting due to data imbalance, which limits its generalizability. The most relevant study (Wang et al., 2021) convert it into a multi-label image classification task. The proposed model, named the Semantic Parsing Guided Neural Network (SPGNN), extracts the relevant semantic visual information via a semantic parsing module and employs a Convolutional Neural Network (CNN) to recommend APIs. The study also released three new datasets consisting of graph APIs in R and Python programming languages. Even though SPGNN achieved good performance on some datasets containing adequate training data, it suffers from overfitting due to data imbalance. This imbalance is reflected in two levels: API categories and the proportion of positive and negative samples. Since API usage frequencies vary, data imbalance between API categories is common, which results in a long-tail distribution throughout the data (as shown in Fig. 1 and Table 1). Likewise, the number of positive samples for any API equals the sample number of the API category, while the number of negative samples for any API equals the sample number of rest API categories, thus leading to an imbalance between positive and negative samples during model training. Due to these problems, Plot2API can fail to deliver satisfactory performance in recommending APIs with limited training samples.

To alleviate these issues, we propose a novel optimization strategy named Multi-cue Shape-aware Transferable Imbalance Learning (MSTIL), which contains three major components. The first component is named Cross-Language Shape-Aware Plot Transfer Learning (CLSAPTL), which is motivated by the class hierarchy of the data. While different plots exhibit various shapes and forms, we notice they can be categorized into three super-classes based on shape: polygon, circle and dot-line. The shape-aware knowledge about class hierarchy is cross-language and relevant to plot2API, so a cross-language shape-aware pre-training (CLSAPTL) is necessary. To integrate prior knowledge from the class hierarchy into the model, CLSAPTL uides the model to learn the class hierarchy, thereby enabling the learning of more transferable. Furthermore, we notice that many APIs that exert similar functions in different languages have semantically similar names. Thus the samples of API pairs that are semantically similar can be used for cross-language training to alleviate data imbalance between API categories. We propose the Cross-Language API Semantic Similarity-based Data Augmentation (CLASSDA) as second component, which augments the data of some APIs by incorporating the samples of other APIs that have semantically similar names. Inspire by the essence of softmax cross entropy loss, a third component — Imbalance Plot2API Learning (IPL) introduces a novel multi-label classification loss function named Balanced Multi-Label Softmax Loss (BMLSL) to alleviate the imbalance between positive and negative samples during training. BMLSL uses the logsumexp function to divide multiple classes into the target or non-target classes, therefore avoiding the impact of having numerous classes. Thus, separate losses are not calculated for negative samples. Therefore, CLSAPTL, CLASSDA and IPL together constitute the MSTIL. Different from Wang et al. (2021) using semantic parsing to exploit more robust visual information, our approach focus on the data imbalance on API categories and the proportion of positive and negative samples which not only improves the performance of SPGNN but also improves the performance of other classical convolution neural networks.

We used classic CNNs along with the state-of-the-art SPGNN approach proposed by Wang et al. (2021) to evaluate our optimization strategy on the Python-Plot13 and R-Plot32 datasets. We found that our approach improved the performance of all Plot2API models tested on both datasets and exceeded the current state-of-the-art method.

The main contributions of this work are three-fold:

• MSTIL, a novel optimization strategy, is introduced and it demonstrates improved performance by addressing the overfitting problems in graphic API recommendation models caused by data imbalance.

• A shape-aware pre-training method named CLSAPTL is proposed to introduce super-class information to Plot2API models, thus improving the generalizability of the models. A data augmentation scheme named CLASSDA is employed to add additional class samples to the training data and alleviate the associated class imbalances, which improve the performance of Plot2API models with minimal additional computational cost. And a new loss function named BMLSL that can suppress overfitting caused by the imbalance between positive and negative samples on Plot2API models is also proposed.

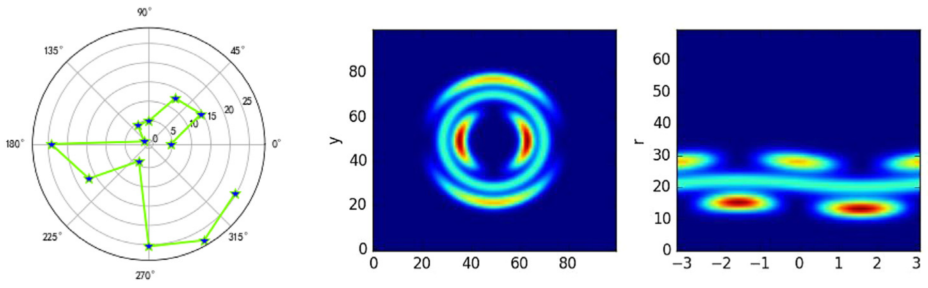


Fig. 2. Two completely different images that both only use polar(). As a result, it is difficult for beginners to identify them.

• Extensive results demonstrate that Cross-Language Shape-Aware Plot Transfer Learning, Cross-Language API Semantic Similarity-based Data Augmentation and Imbalance Plot2API Learning all improve the performance of Plot2API models. Meanwhile, the experimental results show that MSTIL, that is, the combination of these three optimization modules, performs the best for Plot2API.

2. Approach

In this section, we first present a motivating example to introduce the problem this paper addresses. We then discuss the components of the proposed solution. Finally, we elaborate on our proposed optimization strategy.

2.1. Motivating example

Consider the example of a programmer who is attempting to utilize plotting APIs to represent some data. Due to possible lack of experience, or simply because of the extremely large number of APIs available to use, this task can quickly become challenging and time-consuming.

In the specific case shown in Fig. 2, both the left and right images are drawn utilizing polar() in the Python programming language. These two images are completely different, but they use the same APIs. It is difficult for programmers, especially beginners, to identify the API that plots these graphs.

Now, consider a programmer who knows how they would like to visualize their given data and has imitated the desired style either by hand or digitally. Knowing which API would best suit the programmer's needs again quickly becomes a difficult, time-consuming problem to solve which would take away from the programmers other pressing matters. In such cases, programmers may look up documentation, tutorials, and discussion boards in order to help make a decision on which API they should choose. However, this task can easily become cumbersome and time-consuming. Hence, an automatic tool that can recommend graphic APIs (Plot2API) is extremely valuable. In this paper, we propose MSTIL, an optimization approach that aims to mitigate the class imbalances that appear in the most common datasets and combines CLSAPTL, CLASSDA and UPL.

2.2. Overview and preliminary study

Plot2API is a task that recommends graphic APIs via plot image input. In the field of computer science for image processing, Plot2API can be considered a multi-label image classification task since each plot can be drawn by at least one graphic API. Moreover, the usage frequencies of different graphic APIs are inconsistent with one another. For example, the function plot() appears in almost every plotting program, whereas the broken_barh() is seldomly used. Therefore, the amount of data relating to different graphic APIs is imbalanced. Training with imbalanced data will easily cause the model to overfit, a problem in which the model tends to predict the testing sample as a more frequently appearing API. The appearance of this data imbalance phenomenon can be observed in the popular datasets used in Plot2API recommendation models. For example, there are 4,086 samples of plot() but only 10 samples of broken_barh() in the Python-Plot13 dataset. The R-Plot32 dataset suffers from an even worse data imbalance as the density_2d(), quantile(), and spoke() each have only 4 samples. This makes it extremely difficult to train a proper model that can predict these APIs. The Multi-Label Image Classification (MLIC) naturally creates another potential type of imbalance in which the overwhelming number of negative samples overshadows the number of positive samples with respect to each category. This problem occurs as a result of the MLIC task often being considered as a set of binary classification sub-tasks.

To address these imbalance issues, we propose MSTIL, a generic Plot2API framework. MSTIL consists of CLSAPTL, CLASSDA, and Imbalance Plot2API Learning (IPL). The addition of MSTIL generally boosts the performances of different Plot2API models in three ways, namely cross-language shared knowledge exploitation, cross-language data argumentation, and imbalance suppression during model optimization. These processes correspond to CLSAPTL, CLASSDA, and IPL, respectively. CLSAPTL works by grouping graphic APIs from different programming languages into three abstract types (super-classes), namely polygon, circle, and dot-line, according to their output geometries. These super-classes are considered as the shape labels that represent cross-language plot similarities. Then, a Plot2API model can be pre-trained with similar cross-language plots in a supervised manner. Furthermore, the names of APIs usually imply their functionality. Thus, if the functions of two APIs from different languages are identical, the semantics of their names should be highly similar. CLASSDA leverages this property by mining extra samples from another language to augment the samples of some APIs, thereby alleviating the imbalance among APIs. Finally, IPL introduces a novel loss function, BMLSL, to fine-tune the pre-trained model with the augmented data and achieve an enhanced Plot2API model for improved performance.

Let $X^T = \{x_i^T\}_{i=1}^{n_T}$ and $X^S = \{x_i^S\}_{i=1}^{n_S}$ be the samples of target and source languages respectively. $Y^T = \{y_i^T\}_{i=1}^{n_T}$ and $Y^S = \{y_i^S\}_{i=1}^{n_S}$ are the corresponding labels. The datasets of the target and source languages can be respectively represented as $\mathcal{D}^T = \{X^T, Y^T\}$ and $\mathcal{D}^S = \{X^S, Y^S\}$. n_T and n_S are the sample amounts of the target and source languages. x^T and x^S are a d-dimensional plot image while y^T and y^S are respectively m_T and m_S-dimensional binary label vectors where m_T and m_S are the API amounts of the target and source languages. In this paper, the target language is the programming language containing the API we want to recommend, while the source language is the programming language we want to use as the auxiliary data to help train the Plot2API model of the target language. A Plot2API model is trained based on the data of the target language for predicting the APIs of an

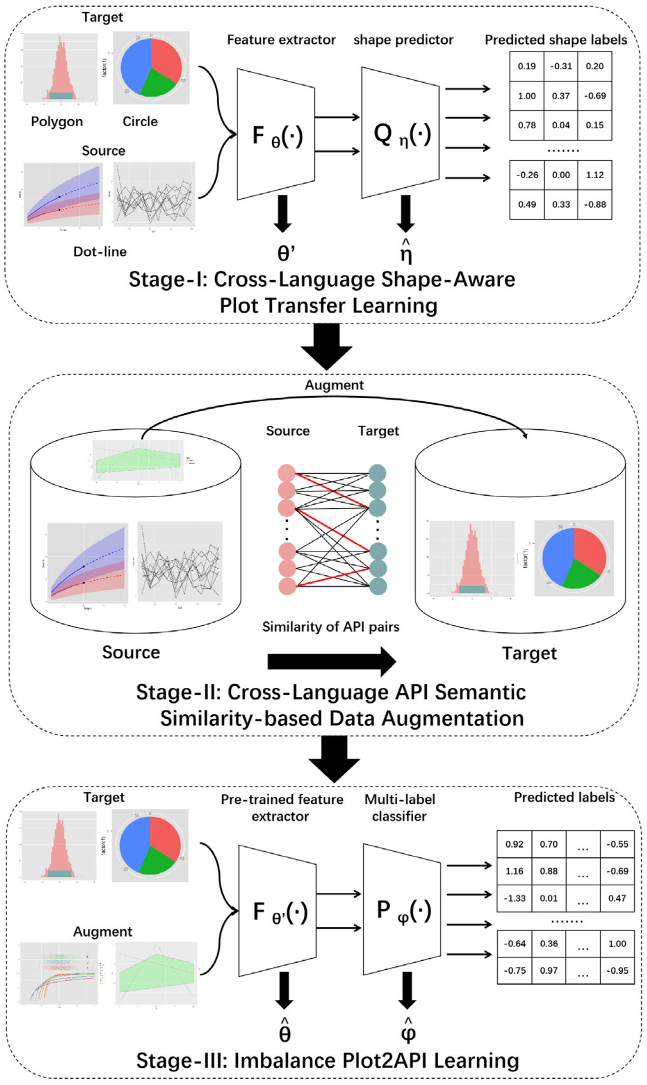


Fig. 3. The flow of MSTIL. First, Cross-Language Shape-Aware Plot Transfer Learning (CLSAPTL) introduce a shape predictor to predict the hierarchical label of each sample and obtains a pre-trained feature extractor. Then Cross-Language API Semantic Similarity-based Data Augmentation (CLASSDA) augments the data of some APIs by collecting the samples of other APIs which have highly semantically similar names. Finally, Imbalance Plot2API Learning (IPL) introduces a novel loss function, Balanced Multi-Label Softmax Loss (BMLSL), to fine-tune the pre-trained model with the augmented data and achieve an enhanced Plot2API model for improved performance.

input plot image,

$$\hat{y}^T \leftarrow P_\phi(F_\theta(x^T)), \tag{1}$$

where $F(\cdot)$ and $P(\cdot)$ are the mapping functions of the convolution layer and multi-label classifier respectively while θ and ϕ are their associated learnable parameters. In deep learning, the classifier is often the last layer, or two full-connection layers, while the feature extractor comprises the rest of the neural network. In this paper, MSTIL aims to leverage the data of other languages, which are also referred to as the source languages, to boost the performance of Plot2API by eliminating the negative effects of data imbalance. The flow of MSTIL is shown in Fig. 3.

2.3. Cross-language shape-aware plot transfer learning

The purpose of the CLSAPTL module is to exploit the shared knowledge from source programming languages in order to mitigate the imbalance of categories. And source programming languages mean the programming languages that have plot2API dataset other than target programming language. Although APIs from different languages draw different types of graphics, these graphics can be classified into three abstract categories. Therefore, we can group APIs across different languages into discrete categories (super-classes) based on the geometric characteristics of their output plots. The super-class categories are polygon, circle, and dot-line. We then consider these super-classes as our shape labels $l \in \mathcal{R}^3$, and all samples can be labeled according to their parent API. We regard the shape information as shared knowledge between programming languages. Fig. 4 lists the shape labels of each API in Python and R. In CLSAPTL, we introduce a shape predictor $Q_\eta()$ to predict the shape label of each sample,

$$\hat{l}_t \leftarrow Q_\eta(F_\theta(x_t)), \text{ s.t. } x_t \in X^T \cup X^S, \tag{2}$$

where $\hat{l}$ is the predicted 3-dimensional shape label of sample x, and η represents the learnable parameters of $Q(\cdot)$. Then, the Plot2API model can be pre-trained by solving the following optimization problem,

$$\{\acute{\theta}, \hat{\eta}\} \leftarrow \arg\min_{\theta,\eta} \sum_{x_t \in X^T \cup X^S} \mathcal{L}(\hat{l}_t, l_t), \tag{3}$$

where $\mathcal{L}(\cdot,\cdot)$ is our proposed loss, BMLSL, which will be introduced in Eq. (6), and $\acute{\theta}$ are the parameters of $F(\cdot)$ obtained by pre-training the Plot2API model with the data of both the target and source languages. Additionally, $\acute{\theta}$ will be treated as the initial parameters for the Plot2API model when fine-tuning the model. Through this transfer learning step, all the samples of the source language also participate in the Plot2API model training of the target language.

2.4. Cross-language API semantic similarity-based data augmentation

Since the names of APIs are often synonymous with the plots they output, the names of APIs with similar plot-making functions in different languages usually have similar semantics. The CLASSDA module is used to create this property to augment the data of some APIs by collecting the samples of other APIs that have very semantically similar names. More specifically, we employ the word2vec model (Mikolov et al., 2013) trained on the Wikipedia dataset which is widely used in many studies (Huangfu and Surdeanu, 2018; Huangfu and Zeng, 2018; Minaee et al., 2021), to generate a 400-dimensional word embedding for each API in both the target and source languages as the semantic representation of the respective API. Let $\{v_i^T\}_{i=1}^{m_T}$ and $\{v_j^S\}_{j=1}^{m_S}$ be the semantic representations of APIs in the target and source languages, respectively. We can measure the semantic similarity of a cross-language API pair with a cross-language affinity score by calculating their weighted cosine distance,

$$s_{ij} = \frac{1}{(p_i^T + p_j^S)^\alpha} \times \frac{v_i^T \times v_j^S}{\|v_i^T\|_2 \times \|v_j^S\|_2}, \tag{4}$$

where p_i^T is the sample quantity proportion of the ith target language API while p_j^S is the sample quantity proportion of the jth source language API. $\alpha \geq 0$ is a scaling factor of weight $\frac{1}{(p_i^T+p_j^S)^\alpha}$. We fix this value at 0.1 to make a weight $\frac{1}{(p_i^T+p_j^S)^\alpha}$ in the same order of magnitude to prevent the API quantity disproportion from having a large impact on the results. And $\frac{v_i^T \times v_j^S}{\|v_i^T\|_2 \times \|v_j^S\|_2}$ is the cosine distance of their word embeddings which is used to measure the similarity of the word embeddings, as the representation

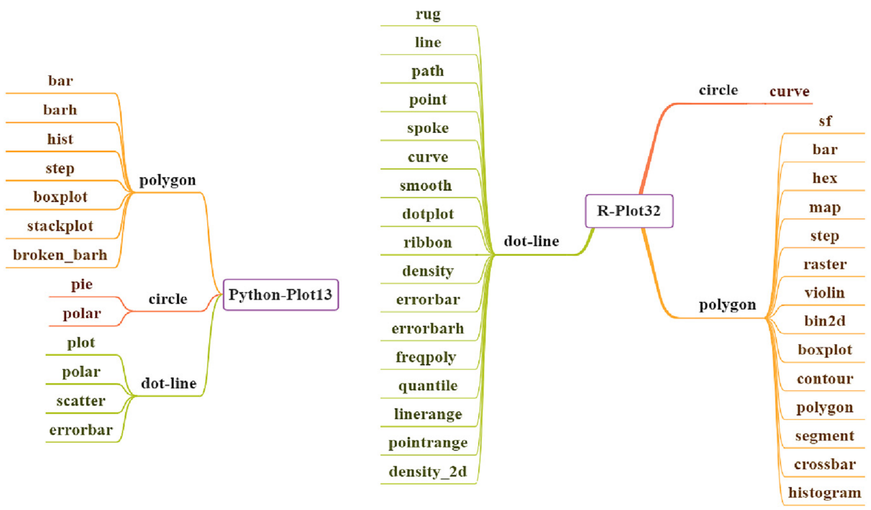


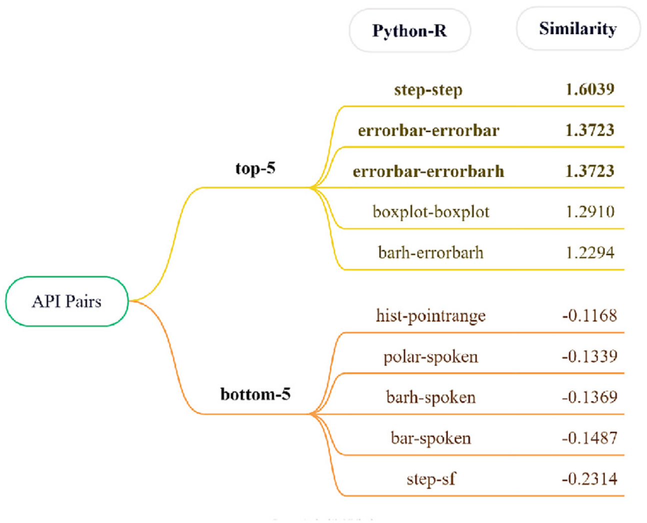
Fig. 4. The shape labels of APIs on Python-Plot13 and R-Plot32 datasets. The shape categories are polygon, circle, and dot-line.

of the semantic similarity between the two APIs. A higher affinity score s_{ij} indicates higher semantic similarity between the cross-language API pair and the corresponding scarcity of their samples in our dataset. Since we intend to find the most relevant source language API for a target language API, particularly for ones with scarce data, we select the pairs corresponding to the top-k highest scores as the likely cross-language API pairs. We then collect the samples of the source language APIs to complement the data of their corresponding target language APIs. Following this process, the augmentation of the datasets for the target language can be mathematically denoted as follows,

$$\hat{\mathcal{D}}^T = \{\hat{X}^T, \hat{Y}^T\}, \text{ s.t. } \hat{X}^T = X^T \cup X^S_{\{\hat{i}\hat{j}\}} \text{ and } \hat{Y}^T = Y^T \cup Y^S_{\{\hat{i}\hat{j}\}}, \tag{5}$$

where $\{\hat{\hat{ij}}\} \leftarrow \rho_k(\{s_{ij}\})$. $\rho_k(\cdot)$ is a descending sort operation and returns the subscript collection of the source language API that corresponds to the top-k highest scores. $X^S_{\{\hat{i}\hat{j}\}}$ is the sample subset of the source language whose API subscripts belong to $\{\hat{j}\}$ and $Y^S_{\{\hat{i}\hat{j}\}}$ is the corresponding target language label set constructed according to the selected API pairs. This newly augmented dataset $\hat{\mathcal{D}}^T$ will replace its original version $\mathcal{D}^T$ for training the final Plot2API model.

Here, according to the experimental results, we set $k = 3$. The top five and bottom five API pairs are shown in Fig. 5.



Python-R	Similarity
top-5	
step-step	1.6039
errorbar-errorbar	1.3723
errorbar-errorbarh	1.3723
boxplot-boxplot	1.2910
barh-errorbarh	1.2294
bottom-5	
hist-pointrange	-0.1168
polar-spoken	-0.1339
barh-spoken	-0.1369
bar-spoken	-0.1487
step-sf	-0.2314

Fig. 5. The top five API pairs and bottom five API pairs. According to the experimental results, we finally choose the top three API pairs for data augmentation.

2.5. Imbalance Plot2API learning

In the final stage, the augmented dataset is used to fine-tune the pre-trained Plot2API model. In the multi-label image classification studies (Chen et al., 2019; You et al., 2020; Zhang et al., 2018), Multi-Label SoftMargin Loss (MLSL) (Lapin et al., 2017) is often used as the loss function. MLSL transforms the final multi-label classification problem into a set of multiple binary classification problems in a one versus rest fashion. However, the proportion of positive samples is always much smaller than the proportion of negative samples with respect to each API. In other words, the sample distribution in such a binary classification scenario still faces the issue of the introduced imbalance. Consequently, the model is easily influenced by superabundant negative samples. An intuitive method for addressing this issue is to manually weight the corresponding positive and negative loss terms according to their proportions for each API. However, these weights cannot accurately reflect the contribution that each sample makes during model optimization. In single-label multi-classification, the softmax cross entropy loss essentially uses the logsumexp function to restrict the weights of negative classes. Thus, these weights are lower than those of the positive classes when multiple classes are divided into positive (target) and negative (non-target) in an attempt to prevent the effects caused by the distribution of class. Following this idea, and inspired by studies (Su, 2020; Sun et al., 2020), we introduce a novel multi-label classification loss function named BMLSL,

$$\begin{aligned}\mathcal{L}(y_i, \hat{y}_i) \;=\; & \log(1 + \sum_{y_i(t)=0} \exp^{\gamma(\hat{y}_i(t)+\epsilon)}) \\ & + \log(1 + \sum_{y_i(t)=1} \exp^{\gamma(-\hat{y}_i(t)+\epsilon)}),\end{aligned} \tag{6}$$

where ϵ is a positive margin parameter and γ is a scaling factor. The main idea of BMLSL is to use the margin to divide multiple classes into target or non-target classes. This ensures that the tth API label prediction of the ith sample $\hat{y}_i(t)$ should always be greater than ϵ when the sample contains the given API, $y_i(t) = 1$, while being smaller than $-\epsilon$ when the sample does not contain this API, $y_i(t) = 0$. We can take advantage of the logsumexp function as it can be regarded as an approximation of the max

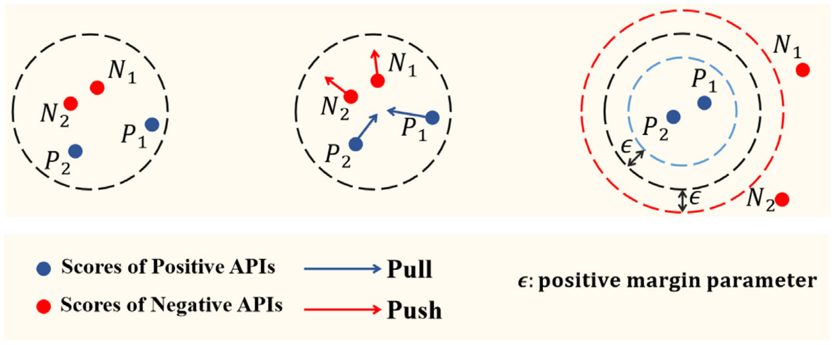

N_1
N_2
P_1
P_2
ϵ
● Scores of Positive APIs
● Scores of Negative APIs
Pull
Push
ϵ: positive margin parameter

Fig. 6. The optimization process of BMLSL. It ensures that the scores of positive APIs should always be greater than ϵ while the scores of negative APIs should always be smaller than $-\epsilon$.

function, thus meaning a greater γ leads to a higher penalty for misclassification based on the margin. The idea of BMLSL is shown in Fig. 6.

Algorithm 1: The MSTIL algorithm

```
Input: Learnable associate parameters $\theta$, $\phi$, $\eta$. The
       samples of target and source languages $X^T$, $X^S$ and
       their corresponding labels $Y^T$, $Y^S$. Shape labels $l_t$
Output: Optimal parameters $\hat{\theta}$, $\hat{\phi}$
1 Stage 1: Cross-Language Shape-Aware Plot Transfer
    Learning:
    a) obtain the predicted shape labels:
    $\hat{l_t} \leftarrow Q_\eta(F_\theta(x_t)),\ \text{s.t. } x_t \in X^T \cup X^S$
    b) pre-training:
    $\{\acute{\theta}, \hat{\eta}\} \leftarrow \arg\min_{\theta,\eta} \sum_{x_t \in X^T \cup X^S} \mathcal{L}(\hat{l_t}, l_t)$
2 Stage 2: Cross-Language API Semantic Similarity-based
    Data
    Augmentation:
    a) calculate the similarities of cross-language API pairs:
    $s_{ij} = \frac{1}{(p_i^T + p_j^S)^\alpha} \times \frac{v_i^T \times v_j^S}{||v_i^T||_2 \times ||v_j^S||_2}$
    b) obtain the final training samples and labels:
    $\hat{X}^T = X^T \cup X^S_{\{\hat{ij}\}}$
    $\hat{Y}^T = Y^T \cup Y^S_{\{\hat{ij}\}}$
3 Stage 3: Imbalance Plot2API Learning:
    a) obtain the predicted labels:
    $\hat{y}_t \leftarrow P_\phi(F_{\acute{\theta}}(x_t)),\ \text{s.t. } x_t \in \hat{X}^T$
    b) fine-tuning:
    $\{\hat{\theta}, \hat{\phi}\} \leftarrow \arg\min_{\acute{\theta},\phi} \sum_{x_t \in \hat{X}^T} \mathcal{L}(y_t, \hat{y}_t),\ \text{s.t. } y_t \in \hat{Y}^T$
4 return $\hat{\theta}$, $\hat{\phi}$
```

We leverage BMLSL as the multi-label classification loss for fine-tuning the Plot2API. The model fine-tuning process can be represented as follows,

$$\{\hat{\theta}, \hat{\phi}\} \leftarrow \arg\min_{\acute{\theta},\phi} \sum_{x_t \in \hat{X}^T} \mathcal{L}(y_t, \hat{y}_t), \tag{7}$$

where $\hat{y}_t \leftarrow P_\phi(F_{\acute{\theta}}(x_t))$ is the prediction of x_t with the Plot2API model initialized with parameters $\acute{\theta}$.

Once the Plot2API model has been fine-tuned, the APIs of any plot can be recommended by the following formula,

$$y = P_{\hat{\phi}}(F_{\hat{\theta}}(x)), \tag{8}$$

where $P_{\hat{\phi}}(F_{\hat{\theta}}(\cdot))$ is the Plot2API model enhanced by MSTIL while $\hat{\phi}$ and $\hat{\theta}$ are its optimal parameters. The overall process of our method is shown in Algorithm 1.

2.6. The Plot2API baselines

We choose some classic CNNs, such as VGG-16 (Simonyan and Zisserman, 2014), ResNet-50 (He et al., 2016), EfficientNet-b3 (Tan and Le, 2019), and SPGNN (Wang et al., 2021) as baselines. Moreover, we complement the study with a popular, recently made CNN-based image classification model named Tresnet (Ridnik et al., 2021) as an extra baseline. MSTIL is then applied to optimize these baselines without introducing any extra network parameters. Comparisons between optimized models and the original ones are made to evaluate the effectiveness of MSTIL.

3. Experimental evaluations

In this section, we present the datasets and evaluation metrics that are used in our experiments. We then introduce the details of our model implementations.

3.1. Datasets

To evaluate the performance of our model, we rely on two different datasets. The first is Python-Plot13 (Wang et al., 2021), which is a collection of posts about the Python programming language from the Stack Overflow Data Dump. The dataset consists of 13 APIs with 6350 images in total. We take 5080 samples for training and 1270 samples for testing. The second is R-Plot32 (Wang et al., 2021) which consists of 9114 images generated through APIs in the R programming language, which is similarly taken from the Stack Overflow Data Dump. We use 7292 examples for training and 1822 for testing. The labels(APIs) in these datasets are extracted as binary vectors in which if the jth element is the non-zero element, it indicates that the corresponding image has the jth type of API. The distribution of these datasets is shown in Fig. 1 and Table 1. We demonstrate some examples from each dataset in Fig. 7.

3.2. Evaluation metrics

To evaluate the recommendation performance of the model, we employ Average Precision (AP) as the performance metric. The area under the precision–recall (P–R) curve is denoted as AP. Furthermore, to assess the overall performance of a model, we use the mean value of the APs over all classes known as the mean average precision (mAP). As the most popular performance metric in multi-label image classification, the mAP is regarded as a comprehensive metric in Plot2API. The calculation process of AP is shown in Algorithm 2.

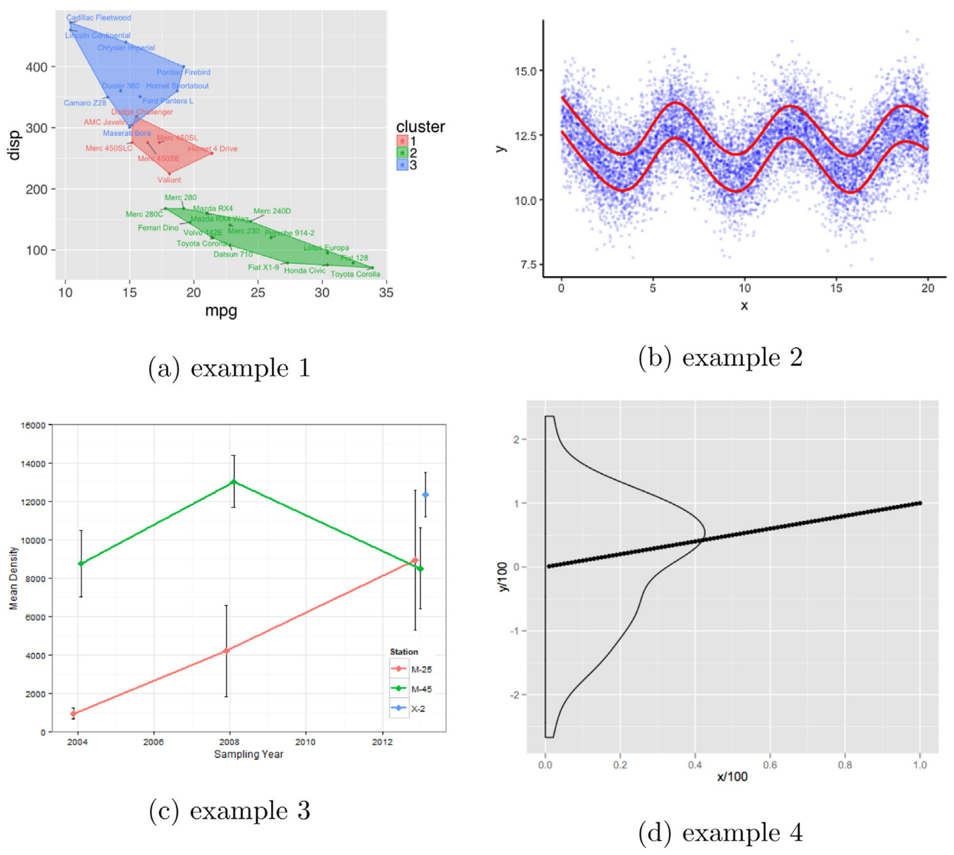


Fig. 7. Some examples of Python-Plot13 and R-Plot32. Most of them use more than one API.

Algorithm 2: The calculation process of AP

```
Input: The output score of model $Y$. The true label of test
       samples $T$. The number of samples $N$.
Output: The Average Precision of API $AP$
1  pos_count = 0.000000001, total_count = 0, precision_at_i
     = 0
2  $indices = Sort(Y)$
3  foreach $i = 1, 2, \ldots, N$ do
4      total_count = total_count + 1
5      if $T[indices[i]] == 1$ then
6          pos_count = pos_count + 1
7          precision_at_i = precision_at_i + 1
8      end
9  end
10 $AP = precision\_at\_i \div pos\_count$
```

3.3. Implementation details

In this paper, we conduct the experiments on VGG-16 (Simonyan and Zisserman, 2014), ResNet-50 (He et al., 2016), EfficientNet-b3 (Tan and Le, 2019), Tresnet (Ridnik et al., 2021) and SPGNN (Wang et al., 2021) to evaluate the generalizability of our model. Following other deep learning studies, all the models are pre-trained on ImageNet (Deng et al., 2009). For SPGNN, we employ the word2vec (Mikolov et al., 2013) model based on the Wikipedia dataset (Rasiwasia et al., 2010), in order to generate a 400-dimensional word embedding for the APIs in all datasets (refer to Wang et al. (2021) for detailed architecture regarding SPGNN). We use a Ranger optimizer with a batch size of 32 and a momentum of 0.99 for training. We take ϵ equal to 0.125 for all datasets and γ equal to 16 and 4 for the Python-Plot13 and the R-Plot32 datasets, respectively. These parameters are empirically discussed and analyzed in the next section. We implement the network through PyTorch.

4. Results and discussions

In this section, we introduce the experimental procedure in which we evaluate MSTIL on the two datasets. We then conduct ablation studies to study the effectiveness of three major components of MSTIL: CLSAPTL, CLASSDA, and BMLSL.

Additionally, we conduct experiments to determine the optimal number of pre-training epochs and evaluate the effectiveness of our shape classification rules. We then examine the performance of MSTIL on hand-drawn images. Finally, we carry out some ablation studies to determine the optimal value for the parameter k in CLASSDA and the value of parameter γ in BMLSL. Overall, we aim to answer the following research questions:

- **RQ1**: How effective is MSTIL for API recommendation?
- **RQ2**: How well do three components of MSTIL, that is, CLSAPTL, CLASSDA, and IPL perform alone?
- **RQ3**: How do the parameter k in CLASSDA and the parameter γ in BMLSL affect the results?
- **RQ4**: How can we determine the number of pre-training epochs and validate the rationality of shape classification rules?
- **RQ5**: How well does MSTIL perform on hand-drawn images?

4.1. RQ1: How effective is MSTIL for API recommendation?

We compare MSTIL with a comprehensive set of baselines, including VGG-16 (Simonyan and Zisserman, 2014), ResNet-50 (He et al., 2016), EfficientNet-B3 (Tan and Le, 2019), TresNet (Ridnik et al., 2021), and SPGNN (Wang et al., 2021). The performance

Table 2
The performance of models with standard training (Baseline) or with MSTIL on both two datasets.

mAP	Datasets	
	Python-Plot13	R-Plot32
SPGNN (Baseline)	75.95	47.76
SPGNN (**Ours**)	**79.12**	**49.41**
Gains	+3.17	+1.65
VGG-16 (Baseline)	64.10	40.84
VGG-16 (**Ours**)	**72.21**	**43.85**
Gains	+8.11	+3.01
ResNet-50 (Baseline)	55.95	29.81
ResNet-50 (**Ours**)	**72.36**	**41.60**
Gains	+16.41	+11.79
EfficientNet-B3 (Baseline)	68.51	44.46
EfficientNet-B3 (**Ours**)	**74.49**	**48.20**
Gains	+5.98	+3.74
Tresnet (Baseline)	67.40	44.46
Tresnet (**Ours**)	**73.46**	**47.46**
Gains	+6.06	+3.00

comparison on all datasets is tabulated in Table 2. Tables 3 and 4 list the AP of per API on all both two datasets, respectively, for the standard training and MSTIL. It can be seen that MSTIL shows improvement over the standard training in different Plot2API models on all datasets. The mAP improvements of our method outperform the state-of-the-art approaches. Therefore, we conclude that MSTIL alleviates the overfitting problem found in Plot2API models caused by data imbalance and improves the performance of all baseline models.

4.1.1. Results of the Python-Plot13 dataset

According to Table 3, we find that MSTIL shows improvement over the standard training of all models on the Python-Plot13 dataset. Our model achieves the following mAP values for the datasets: 72.21% for VGG-16, 72.36% for ResNet-50, 74.49% for EfficientNet-B3, and 73.46% for Tresnet. Overall, MSTIL achieves an average mAP improvement of 9.14% and has an average relative mAP improvement of 14.93% across the models. In addition, with MSTIL, SPGNN has an mAP improvement of 3.17% compared to that of standard training, which exceeds the previous best result. Observing the AP of each API reveals that our optimization method is mainly effective for APIs with limited data. For example, there are respectively 92 samples, 29 samples and 105 samples in errorbar(), polar() and barh(). MSTIL respectively achieves an average AP improvement of 12.95%, 6.39% and 3.53% across the models on errorbar(), polar() and barh(). These experimental results show that models trained with MSTIL perform better on the Python-Plot13 dataset and make improved API recommendations for sets with fewer data, thus mitigating the over-fitting problem caused by data imbalance.

4.1.2. Results of the R-Plot32 dataset

The R-Plot32 dataset contains more API categories than Python-Plot13, making it a more challenging dataset for Plot2API. Our results show MSTIL again outperforms baseline models and according to Table 4, we find that our optimization method shows improvements over standard training for all models on the R-Plot32 dataset. Specifically with MSTIL, the following mAP values are obtained: 43.85% for VGG-16, 41.60% for ResNet-50, 48.20% for EfficientNet-B3, and 47.46% for Tresnet. Overall, MSTIL has an average mAP improvement of 5.39% and an average relative mAP improvement of 15.52% across the models. Additionally, SPGNN with MSTIL shows an mAP improvement of 1.65% over SPGNN with the standard training method on the R-Plot32 dataset, again exceeding the previous best result. Notably, MSTIL shows improvement on many hard-to-recommend APIs, such as dotplot(), linerange(), path(), segment(), and step(), which all have average relative mAP improvements of 26.88% per model. Specifically, there are respectively 38 samples, 49 samples and 46 samples in step(), linerange() and violin(). MSTIL respectively achieves an average AP improvement of 7.87%, 11.57% and 9.52% across the models on step(), linerange() and violin(). In general, MSTIL performs better on the R-Plot32 dataset and especially boosts the performance of the previously difficult-to-recommend APIs.

4.1.3. Why APIs in some models show a decline in AP

Although the mAP of each model is improved on Python-Plot13 and R-Plot32, in some models the AP decreases for a small number of APIs. This decrease occurs because we select the final model based on overall performance. However, the amount of test data of some APIs is extremely limited, which makes the

Table 3
The performance of models With MSTIL or with standard training (Baseline) on the Python-Plot13 dataset. The AP of APIs with test samples more than 10 are also shown as follows.

Methods	**mAP**	bar	barh	boxplot	broken_barh	errorbar	hist	pie	plot	polar	scatter	stackplot	stem	step
SPGNN (Baseline)	75.95	86.15	56.76	96.72	100.00	55.71	77.50	93.41	94.08	62.93	80.37	80.95	66.81	35.97
SPGNN (**Ours**)	79.12	88.50	58.39	97.50	100.00	67.24	74.84	99.36	94.09	82.35	81.51	86.67	66.85	31.19
Gains	+3.17	+2.35	+1.63	+0.78	+0.00	+11.53	−2.66	+5.95	+0.01	+19.42	+1.14	+5.72	+0.04	−4.78
VGG-16 (Baseline)	64.10	85.72	46.57	96.10	6.87	56.44	71.45	91.69	93.61	73.65	77.66	66.79	25.33	41.50
VGG-16 (**Ours**)	72.21	88.63	51.97	92.01	100.00	61.47	76.21	100.00	93.63	63.47	77.29	55.00	55.71	23.32
Gains	+8.11	+2.91	+5.40	−4.09	+93.13	+5.03	+4.76	+8.31	+0.02	−10.18	−0.37	−11.79	+30.38	−18.18
ResNet-50 (Baseline)	55.95	80.29	47.59	89.49	1.30	24.72	54.17	99.36	92.89	70.33	76.01	40.32	35.22	15.63
ResNet-50 (**Ours**)	72.36	88.35	57.84	91.69	83.33	59.58	75.96	98.81	93.53	78.78	76.96	66.80	47.40	21.65
Gains	+16.41	+8.06	+10.25	+2.20	+82.03	+34.86	+21.79	−0.55	+0.64	+8.45	+0.95	+26.48	+12.18	+6.02
EfficientNet-B3 (Baseline)	68.51	87.67	53.36	97.82	1.72	58.66	78.47	100.00	93.53	68.00	77.82	74.36	66.75	32.48
EfficientNet-B3 (**Ours**)	74.49	89.83	53.11	94.36	100.00	65.13	76.55	99.36	93.89	80.32	79.86	50.19	46.91	38.81
Gains	+5.98	+2.16	−0.25	−3.46	+98.28	+6.47	−1.92	−0.64	+0.36	+12.32	+2.04	−24.17	−19.84	+6.33
Tresnet (Baseline)	67.40	84.98	54.33	95.94	2.58	37.83	67.59	96.92	91.69	71.56	78.29	100.00	65.04	29.51
Tresnet (**Ours**)	73.46	86.01	54.97	99.01	49.94	44.71	71.97	98.67	93.41	73.50	80.25	100.00	66.81	35.77
Gains	+6.06	+1.03	+0.64	+3.07	+47.36	+6.88	+4.38	+1.75	+1.72	+1.94	+1.96	+0.00	+1.77	+6.26

Table 4
The performance of models with MSTIL or with standard training (Baseline) on R-Plot32 dataset. The AP of APIs with test samples more than 10 are also shown as follows.

Methods	**mAP**	bar	polygon	boxplot	smooth	raster	ribbon	density	segment	dotplot	errorbar
SPGNN (Baseline)	47.76	95.96	58.65	91.74	50.97	48.88	65.57	88.69	23.96	47.61	75.36
SPGNN (**Ours**)	49.41	95.57	51.54	92.49	51.69	59.56	58.80	92.64	24.80	51.96	78.44
Gains	+1.65	−0.39	−7.11	+0.75	+0.72	+10.68	−6.77	+3.95	+0.84	+4.35	+3.08
VGG-16 (Baseline)	40.84	93.96	43.70	91.43	49.06	45.63	42.03	82.17	20.43	30.59	75.26
VGG-16 (**Ours**)	43.85	94.61	43.4	92.30	51.35	47.26	58.03	87.52	25.49	38.97	70.37
Gains	+3.01	+0.65	−0.30	+0.87	+2.29	+1.63	+16.00	+5.35	+5.06	+8.38	−4.89
ResNet-50 (Baseline)	29.81	92.05	44.47	83.37	39.42	42.76	28.13	71.83	9.16	12.75	40.66
ResNet-50 (**Ours**)	41.60	92.84	51.52	91.79	51.29	51.21	44.95	86.04	25.49	25.06	63.17
Gains	+11.79	+0.79	+7.05	+8.42	+11.87	+8.45	+16.82	+14.21	+11.65	+12.31	+22.51
EfficientNet-B3 (Baseline)	44.46	95.38	56.90	92.86	50.76	50.93	59.81	92.00	26.81	32.44	71.14
EfficientNet-B3 (**Ours**)	48.20	95.58	50.62	93.42	56.85	62.66	57.41	84.50	28.27	49.60	76.58
Gains	+3.74	+0.20	−6.28	+0.56	+6.09	+11.73	−2.40	−7.50	+1.46	+17.16	+5.44
Tresnet (Baseline)	44.46	94.82	53.24	93.79	57.34	49.10	59.95	87.87	28.29	31.00	71.79
Tresnet (**Ours**)	47.46	95.68	53.98	94.21	61.27	61.40	63.89	93.35	32.14	42.89	72.04
Gains	+3.00	+0.86	+0.74	+0.42	+3.93	+12.30	+3.94	+5.48	+3.85	+11.89	+0.25

Methods	errorbarh	step	violin	histogram	jitter	line	linerange	map	path	point	pointrange
SPGNN (Baseline)	41.49	38.11	60.08	69.57	26.71	86.43	45.73	60.50	23.40	95.13	64.05
SPGNN (**Ours**)	55.48	56.24	72.05	70.29	25.30	87.75	39.97	58.14	27.12	95.54	82.96
Gains	+13.99	+18.13	+11.97	+0.72	−1.41	+1.32	−5.76	−2.36	+3.72	+0.41	+18.91
VGG-16 (Baseline)	56.52	35.66	31.24	63.33	15.25	87.31	16.94	37.60	20.70	93.68	61.31
VGG-16 (**Ours**)	52.77	42.48	29.15	64.26	13.77	87.26	41.55	30.27	22.43	93.38	70.42
Gains	−3.75	+6.82	−2.09	+0.93	−1.48	−0.05	+24.61	−7.33	+1.73	−0.30	+9.11
ResNet-50 (Baseline)	53.58	22.15	11.08	44.13	12.72	79.91	2.51	37.82	19.02	88.28	22.37
ResNet-50 (**Ours**)	29.12	41.44	41.87	68.65	13.45	88.86	18.72	37.71	26.74	93.88	73.25
Gains	−24.46	+19.29	+30.79	+24.52	+0.73	+8.95	+16.21	−0.11	+7.72	+5.60	+50.88
EfficientNet-B3 (Baseline)	54.44	43.18	51.47	67.10	34.03	87.02	30.17	48.90	26.66	95.46	48.76
EfficientNet-B3 (**Ours**)	57.53	33.23	54.96	68.57	26.08	88.98	41.96	55.45	28.56	95.30	81.13
Gains	+3.09	−9.95	+3.49	+1.47	−7.95	+1.96	+11.79	+6.55	+1.9	−0.16	+32.37
Tresnet (Baseline)	54.23	40.74	49.28	70.17	23.56	87.62	37.47	55.27	27.37	94.21	69.94
Tresnet (**Ours**)	59.49	45.82	52.72	64.63	33.13	88.45	48.47	48.12	27.41	95.29	67.79
Gains	+5.26	+5.08	+3.44	−5.54	+9.57	+0.83	+11.00	−7.15	+0.04	+1.08	−2.15

Table 5
The bootstrap estimate of P-value between SPGNN with MSTIL (ours) and SPGNN (Baseline) with standard training and the bootstrap estimate of P-value between EfficientNet-b3 with MSTIL (ours) and EfficientNet-b3 (Baseline) with standard training on both two datasets.

P-value	Datasets	
	Python-Plot13	R-Plot32
SPGNN (Ours)-SPGNN (Baseline)	0.05	0.03
EfficientNet-b3 (Ours)-EfficientNet-b3 (Baseline)	0.04	0.03

AP of models relating to these sparse classes susceptible to fluctuation. If the model makes a single incorrect recommendation, the negative effect on the AP will be extensive. Nevertheless, the performance of each model improves overall.

4.1.4. *Is the performance improvement over models by our method reliable*

Following the study (Berg-Kirkpatrick et al., 2012), we have used bootstrap to calculate the bootstrap estimate of P-value between SPGNN with MSTIL and SPGNN with standard training to ensure that the performance improvement over models by our method is reliable. In addition, we have calculated the bootstrap estimate of P-value between EfficientNet-b3 with MSTIL and EfficientNet-b3 with standard training as shown in Table 5. Specifically, we sampled the test sets of Python-Plot13 and R-Plot32 by taking half of the test set for 100 times, and compared the performance of the model with MSTIL and that with standard training. P-value here represents the probability that the performance of the model with MSTIL is worse than or equal to that with standard training in the sampling test. All bootstrap estimate of P-values are less than or equal to 0.05, so the performance improvement of our model over other models should be reliable.

Result 1: *MSTIL shows improved performance on both two datasets. The results indicate that our optimization method is effective for Plot2API.*

4.2. *RQ2: How well do three components of MSTIL, that is, the cross-language shape-aware plot transfer learning, cross-language API semantic similarity-based data augmentation, and imbalance Plot2API learning perform alone?*

We compare CLSAPTL, CLASSDA, and IPL with standard training on SPGNN and EfficientNet-B3, which are the top-performing models on both two datasets. The results on our datasets are tabulated in Table 6, respectively, for each module.

Table 6
The performance of models trained with Cross-Language Shape-Aware Plot Transfer Learning (CLSAPTL), Cross-Language API Semantic Similarity-based Data Augmentation (CLASSDA) or Imbalance Plot2API Learning (IPL) on both two datasets.

mAP	Datasets	
	Python-Plot13	R-Plot32
EfficientNet-B3 (Baseline)	68.51 (+0.00)	44.46 (+0.00)
EfficinetNet-B3 (+CLSAPTL of MSTIL)	70.97 (+2.46)	46.62 (+2.16)
EfficinetNet-B3 (+CLASSDA of MSTIL)	70.08 (+1.57)	46.44 (+1.98)
EfficinetNet-B3 (+IPL of MSTIL)	73.32 (+4.81)	46.83 (+2.37)
EfficientNet-B3 (Ours)	**74.49** (+5.98)	**48.20** (+3.74)
SPGNN (Baseline)	75.95 (+0.00)	47.76 (+0.00)
SPGNN (+CLSAPTL of MSTIL)	77.09 (+1.14)	48.69 (+0.93)
SPGNN (+CLASSDA of MSTIL)	77.41 (+1.46)	48.15 (+0.39)
SPGNN (+IPL of MSTIL)	77.20 (+1.25)	48.72 (+0.96)
SPGNN (Ours)	**79.12** (+3.17)	**49.41** (+1.65)

Significantly, CLSAPTL, CLASSDA, and IPL all show improvements when compared to the standard training on both datasets, indicating that each helps to alleviate the overfitting problem found in the Plot2API models. Furthermore, MSTIL outperforms CLSAPTL, CLASSDA, and IPL when implemented individually, suggesting that these modules complement each other in the Plot2API task. This result is expected, given the fact that these modules leverage distinct mechanisms to address the issue of overfitting. CLSAPTL enables Plot2API models to learn class hierarchy as well as more transferable visual features before formal training, while CLASSDA aims to augment data based on the similarities of the APIs, thus directly alleviating data imbalance. Meanwhile, BMLSL of IPL uses the logsumexp function to turn the multi-label classification problem into a comparison between target class scores and non-target class scores, thus suppressing the imbalance between positive and negative samples. Together, MSTIL integrates these modules to resolve the data imbalance problem in a comprehensive manner.

Result 2: *Cross-Language Shape-Aware Plot Transfer Learning, Cross-Language API Semantic Similarity-based Data Augmentation and Imbalance Plot2API Learning all improve the performance of Plot2API models. Meanwhile, the experimental results show that MSTIL, that is, the combination of these three optimization modules, performs the best for Plot2API.*

4.3. RQ3: How does the value of parameter k in cross-language API semantic similarity-based data augmentation and the value of parameter γ in balanced multi-label softmax loss affect the results?

We note the values of parameter k in CLASSDA and parameter γ in BMLSL are statically preset manually, therefore they cannot be learned. As such, it is important to explore how the value of parameter k in CLASSDA and the value of parameter γ in BMLSL affect the accuracy of the models. We compare various values of k and γ with SPGNN and EfficientNet-B3, which are the top-performing models on our datasets.

According to our experimental results, the best value of k is 3, the best value of γ on Python-Plot13 is 16, and the best value of γ on R-Plot32 is 4 (these results are shown in Table 7). In all datasets, the best value of k being 3 is rationalized by if k is small, the effect of data augmentation is very weak, whereas if k is large, inappropriate or unnecessary data may be introduced. Similarly, a large γ value is not better because this will lead to a larger loss value and the model may fall into the local optimum, even though a larger γ can make the logsumexp function closer to the max function. The explanation for why the best value of

Table 7
The analysis of the value of parameter k in Cross-Language API Semantic Similarity-based Data Augmentation (CLASSDA) and the value of parameter γ in Balanced Multi-Label Softmax Loss (BMLSL) on both two datasets.

mAP	Datasets	
	Python-Plot13	R-Plot32
EfficientNet-B3 (+CLASSDA +k = 1)	69.13	45.25
EfficientNet-B3 (+CLASSDA +k = 2)	69.61	45.96
EfficientNet-B3 (+CLASSDA +k = 3)	70.08	46.44
EfficientNet-B3 (+CLASSDA +k = 4)	70.00	46.01
EfficientNet-B3 (+CLASSDA +k = 5)	65.69	42.27
SPGNN (+CLASSDA +k = 1)	76.35	47.81
SPGNN (+CLASSDA +k = 2)	76.98	48.03
SPGNN (+CLASSDA +k = 3)	77.41	48.15
SPGNN (+CLASSDA +k = 4)	77.12	48.00
SPGNN (+CLASSDA +k = 5)	72.10	45.64
EfficientNet-B3 (+BMLSL +γ = 4)	71.97	46.83
EfficientNet-B3 (+BMLSL +γ = 8)	72.83	46.36
EfficientNet-B3 (+BMLSL +γ = 16)	73.32	45.81
EfficientNet-B3 (+BMLSL +γ = 32)	71.24	44.65
EfficientNet-B3 (+BMLSL +γ = 64)	70.79	43.40
SPGNN (+BMLSL +γ = 4)	76.53	48.72
SPGNN (+BMLSL +γ = 8)	76.84	48.41
SPGNN (+BMLSL +γ = 16)	77.20	47.56
SPGNN (+BMLSL +γ = 32)	76.27	47.08
SPGNN (+BMLSL +γ = 64)	75.42	46.33

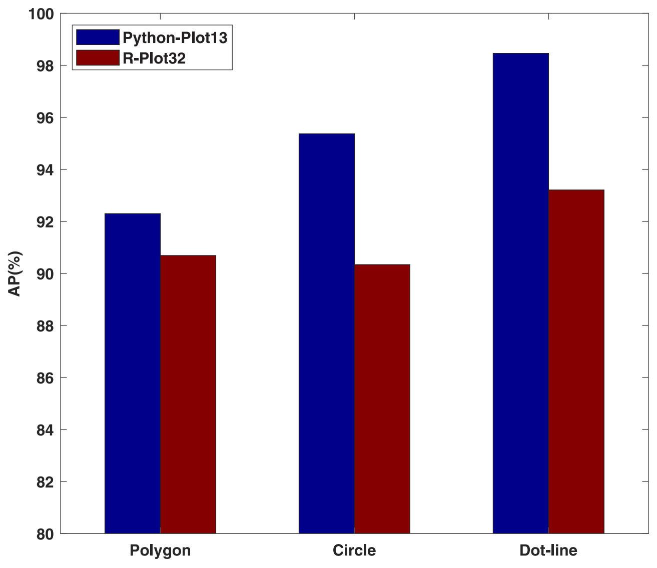
γ is 16 on Python-Plot13 and 4 on R-Plot32 simply results from R-Plot32 having more API classes. If γ is too large, the loss value will similarly be too large for the given dataset.

Result 3: *According to our experimental results, the best value of k is 3, the best value of γ on Python-Plot13 is 16, and the best value of γ on R-Plot32 is 4.*

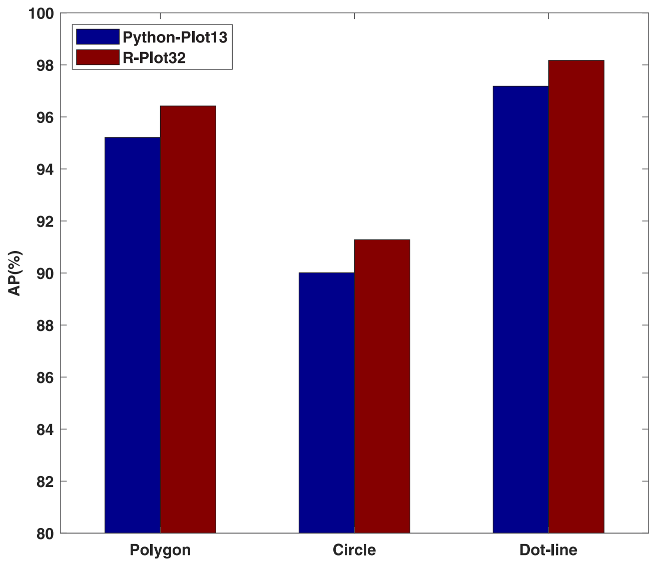
4.4. RQ4: How can we determine the number of pre-training epochs and validate the rationality of shape classification rules?

To validate the rationality of our shape classification rules, we test the pre-training model on the Python-Plot13 and R-Plot32 datasets using shape labels and our testing set. Here, we choose ResNet-50 to make sure that even the worst performing model on the Plot2API problem is compatible with our pre-training module. In contrast to the standard deep learning training/testing strategy, we adopt a cross-test to verify the rationality of our classification rules. In other words, the model pre-trained by a single training set will be tested on all the test sets. If the model pre-trained by a single training set performs well on all test sets, we can conclude that the model has successfully learned transferable shape classification information. Consequently, our shape classification rules can not only distinguish distinct plots but are applicable to plots from different programming languages as well.

The cross-test results of the model after one round of pre-training are shown in Fig. 8. The results show that after one round of pre-training, the model performs well in the cross-test and reaches more than 90% AP in different shape classes for different test sets (Fig. 8). This result not only justifies our shape classification rules, but also shows that one round of pre-training allows the model to learn sufficient shape classification information, thus enhancing the feature extraction ability of the models used in our Plot2API problem.



(a) ResNet-50 pre-training on Python-Plot13.



(b) ResNet-50 pre-training on R-Plot32.

Fig. 8. The test results of ResNet-50 pre-trained by a single training set on all test sets. The results show that after one round of pre-training, the model performs well in the cross-test and reaches more than 90% AP in different shape classes for different test set.

Result 4: *After one round of pre-training, the test model performs well in the cross test, validating our shape classification rules. The experiment results also show that one round of pre-training is enough for the models to learn transferable shape classification knowledge.*

4.5. RQ5: How well does MSTIL perform on hand-drawn images?

An important application of the Plot2API models is to identify APIs from hand-drawn images so that users can receive help even when they have no computer-generated sample images. Since Plot2API models do not use hand-drawn images during training, it is difficult for models to identify APIs from hand-drawn images accurately. Therefore, examining the performance of Plot2API models on non-standard, hand-drawn images is a stringent yet crucial test for evaluating the generalizability and practicality of the models.

To examine whether MSTIL can boost performance in this crucial application, we utilize the state-of-the-art approach SPGNN and evaluate the performance of models trained by the standard method or MSTIL on hand-drawn images. For the hand-drawn dataset, We invited ten volunteers to draw 200 hand-drawn images and label them. The results are shown in Table 8 and Fig. 9. The results show that the SPGNN model trained with MSTIL performs better on the difficult task of recognizing hand-drawn images as compared to the traditionally trained SPGNN model (Fig. 9; green indicates a correct recommendation, gray refers to a missing recommendation, and red represents an incorrect recommendation). The SPGNN model trained without MSTIL can only accurately recognize APIs that are easy to identify (those with the most training examples) such as bar() and plot(), but misidentifies errorbar() as a complex API combination of hist() and plot(). However, when trained with MSTIL, SPGNN can not

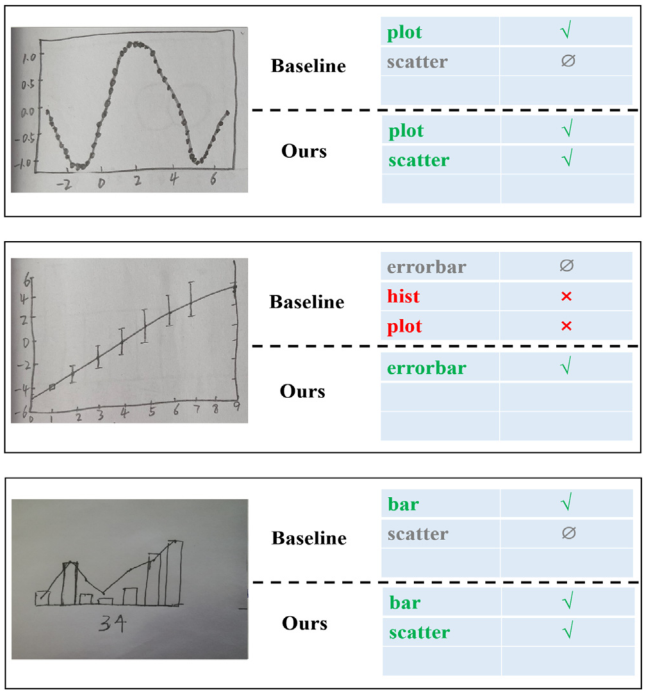


Fig. 9. Testing results on hand-drawn images. Green represents a correct recommendation, gray indicates a missing recommendation and red represents an incorrect recommendation. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 8
The performance of models with standard training (Baseline) or with MSTIL on hand-drawn images.

mAP	Datasets
	Hand-Drawn
SPGNN (Baseline)	28.70
SPGNN (**Ours**)	**33.62**
Gains	+4.92

only accurately recognize APIs with insufficient training data, such as scatter(), but also avoid misidentifying complex APIs.

These experimental results show that the models trained with MSTIL are better at recognizing difficult-to-identify APIs than their traditionally trained counterparts, especially on hand-drawn images. This superior performance is due to the advantages of MSTIL, including its ability to enrich the training samples of some APIs through the addition of cross-language data, pre-train models with shape labels, and use a new loss function to minimize data imbalance.

Result 5: *MSTIL performs better than the state-of-the-art approach on hand-drawn images, thus improving the generalizability and practicality of the Plot2API models.*

5. Threats to validity

We identify several threats to this study's validity which can be divided into threats to internal validity, threats to external validity, and threats to construct validity. The threat to external validity is the limitation of recommendable APIs. In our study, we selected 13 APIs from the Python language, along with 32 APIs from the R language. Although most APIs are covered in both datasets, the number of APIs we used remains quite small. The models cannot recommend any API not included in the dataset. However, it is plausible that our method can still offer a correct prediction for any class that does not appear in the dataset if the training data is sufficient.

The threat to internal validity is that some classes in both datasets may perform inadequately in this task. This is because some APIs have insufficient training data or insignificant visual features. If a larger dataset with sufficient training examples is procured, we could address this issue with the expectation that our approach will provide improved performance since we have proven that our technique can learn distinguishable visual features that facilitate API recommendations.

The threat to construct validity is that some images can be drawn by many different API combinations, but our approach can only predict one of these combinations since the dataset provides only one label for each image. If the dataset could provide multiple API combinations for each image, we can set multiple classifiers for the Plot2API model to output multiple APIs.

6. Related work

We now discuss related studies that pertain to the areas described below.

API Recommendation: Numerous well-established approaches (Zhong et al., 2009; Wang et al., 2013; Nguyen and Nguyen, 2015; Rahman et al., 2016; Ling et al., 2021) for API recommendation have been proposed to help programmers use APIs properly. These studies (Rahman et al., 2016; Raghothaman et al., 2016; Huang et al., 2018; Ling et al., 2021) rely on the semantic parsing of texts and leverage word embeddings to translate between natural language queries, source code repositories, and application documentation. For example, Gu et al. (2016) propose DeepAPI, an approach based on deep learning and an Recurrent Neural Network (RNN) encoder–decoder, adopts a machine translation framework to get API call arrays for text queries. FOCUS (Nguyen et al., 2019), a context-aware collaborative-filtering system, exploits open-source project repositories to find suitable API functions and usage modes. Furthermore, Lee et al. (2020) propose three different approaches to improve the performance of semantic relatedness measurements between two words. An approach called K-CAR (Qi et al., 2019) can return a set of optimal APIs by analyzing the input description of the effects expected by the programmer. Ling et al. (2019) propose the GAPI, which adopts graph neural networks to extract the high-order collaborative information from API calls.

In contrast to the traditional API recommendations, the emerging task Plot2API aims to exploit discriminative features of graphs and directly recommend plot-making APIs in a graph-to-API manner. The sole significant study (Wang et al., 2021) was recently published, which converted the Plot2API into a multi-label classification task. The proposed model Semantic Parsing Guided Neural Network (SPGNN) extracts the relevant semantic visual information via a semantic parsing module and employs a convolutional neural network (CNN) to recommend API. SPGNN has achieved good performance on many datasets containing abundant training data. However, its performance often suffers from overfitting due to data imbalance, therefore limiting its generalizability. This work aims to address this issue through a novel optimization strategy that leverages data augmentation, transfer learning, and a new loss function.

Multi-Label Image Classification: Many efforts have been made to improve the multi-label image recognition task. With the advances of deep learning, deep convolutional networks have become a popular technique for tackling multi-label image recognition studies (Szegedy et al., 2015; He et al., 2016; Hu et al., 2018; Chollet, 2017; Xie et al., 2017; Chen et al., 2019; Tan and Le, 2019) in recent years. The goal of multi-label image recognition is to categorize images into multiple classes. By learning discriminative visual features, a model can identify the objects corresponding to the labels in the figures. For example, Chen et al. (2019) built a directed graph on the object labels by teaching a Graph Convolutional Network to map the label-graph into a set of object classifiers. Another study proposed a conditional label structure learning method for producing image-dependent conditional label structures for multi-label image classification (Li et al., 2016). Meanwhile, Lee et al. (2018) proposed a multi-label learning framework, which leveraged different relationships defined in a constructed knowledge graph by incorporating knowledge maps that describes the relationships between different labels. However, EfficientNet (Tan and Le, 2019) achieves a better performance in balancing network depth, width, and resolution. Additionally, Ridnik et al. (2021) propose TResnet, an improvement of Resnet50, which contains three variants that vary in depth and the number of channels. Our study employs several classic approaches for multi-label image classification as baselines.

Long-Tail Distribution: Most datasets are based on real-world user scenarios and thus have long-tailed distributions, which introduces difficulties in learning features and conducting classification tasks. The following approaches are considered to be the most commonly used to boost performance:

- **Re-sampling**: The primary idea of re-sampling is to balance samples in the training data, such as under-sample head classes or over-sample tail classes. As many studies (Buda et al., 2018; Zou et al., 2018; Byrd and Lipton, 2019) have shown, in doing so, the tail classes can be trained more properly. However, this method may lead to over-fitting, and under-sample head classes can limit the generalization of the network. In contrast, CLASSDA in MSTIL directly supplements training data for scarce API through source programming language. It not only alleviates the problem of long tail distribution, but also avoids the problems of over fitting and noise.
- **Re-weighting**: The main purpose of re-weighting is to distribute different weights for multiple categories in the loss function and set a larger weight for the tail classes to achieve a better balance. And it has been applied in many studies (Huang et al.,

2016; Khan et al., 2017; Sarafianos et al., 2018; Cui et al., 2019; Yu et al., 2021). However, due to the difficulty of optimization, this approach cannot handle large-scale datasets.

• **Learning strategy**: Many researchers use meta-learning and transfer learning to mitigate data imbalance, such as studies (Wang et al., 2017; Kamani et al., 2019; Wang et al., 2019; Lee et al., 2021) and studies (Bengio, 2015; Ouyang et al., 2016; Cui et al., 2018; Zhang et al., 2021), among others. While different plots exhibit various shapes and forms, we notice they can be categorized into three super-classes based on shape: polygon, circle and dot-line. Therefore, a shape-aware pre-training method named CLSAPTL is proposed to mitigate data imbalance in plot2API through transfer learning.

7. Conclusion and future work

In this paper, we propose a novel optimization strategy for API recommendation named Multi-cue Shape-aware Transferable Imbalance Learning (MSTIL). MSTIL combines Cross-Language Shape-Aware Plot Transfer Learning (CLSAPTL), Cross-Language API Semantic Similarity-based Data Augmentation (CLASSDA), and Imbalance Plot2API Learning (IPL). We found that MSTIL improved the performance of all Plot2API models tested on both Python-Plot13 and R-Plot32 datasets and exceeded the performance of current state-of-the-art methods. In future work, we hope to improve our pre-training method, supervising it to learn from classical self-supervised tasks, such as rotation and repair, and further extract the hidden information and features in plots. We plan to validate our approach on more Plot2API models and additional programming languages. Moreover, we will study the effect of semi-supervised learning on Plot2API. We will attempt to use the trained model to label the images of other programming languages, with the goal of achieving more comprehensive cross-language data augmentation to further alleviate data imbalance.

CRediT authorship contribution statement

Rong Qin: Software, Writing – original draft, Formal analysis, Visualization, Investigation. **Zeyu Wang:** Data curation, Validation, Writing – original draft. **Sheng Huang:** Supervision, Conceptualization, Methodology, Project administration, Funding acquisition. **Luwen Huangfu:** Supervision, Conceptualization, Methodology, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 62176030, in part by the Natural Science Foundation of Chongqing, China under Grant cstc2021jcyj-msxmX0568.

References

Bengio, S., 2015. Sharing representations for long tail computer vision problems. In: Proceedings of the 2015 ACM on International Conference on Multimodal Interaction. p. 1.
Berg-Kirkpatrick, T., Burkett, D., Klein, D., 2012. An empirical investigation of statistical significance in NLP. In: Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning. pp. 995–1005.
Buda, M., Maki, A., Mazurowski, M.A., 2018. A systematic study of the class imbalance problem in convolutional neural networks. Neural Netw. 106, 249–259.
Byrd, J., Lipton, Z., 2019. What is the effect of importance weighting in deep learning? In: International Conference on Machine Learning. PMLR. pp. 872–881.
Chen, Z.-M., Wei, X.-S., Wang, P., Guo, Y., 2019. Multi-label image recognition with graph convolutional networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 5177–5186.
Chollet, F., 2017. Xception: Deep learning with depthwise separable convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1251–1258.
Cui, Y., Jia, M., Lin, T.-Y., Song, Y., Belongie, S., 2019. Class-balanced loss based on effective number of samples. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 9268–9277.
Cui, Y., Song, Y., Sun, C., Howard, A., Belongie, S., 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, J., Dong, W., Socher, R., Li, L.-J., Li, K., Fei-Fei, L., 2009. Imagenet: A large-scale hierarchical image database. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition. Ieee, pp. 248–255.
Gu, X., Zhang, H., Zhang, D., Kim, S., 2016. Deep API learning. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. pp. 631–642.
He, X., Xu, L., Zhang, X., Hao, R., Feng, Y., Xu, B., 2021. PyART: Python API recommendation in real-time. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering. ICSE, IEEE, pp. 1634–1645.
He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 770–778.
Hu, J., Shen, L., Sun, G., 2018. Squeeze-and-excitation networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 7132–7141.
Huang, C., Li, Y., Loy, C.C., Tang, X., 2016. Learning deep representation for imbalanced classification. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 5375–5384.
Huang, Q., Xia, X., Xing, Z., Lo, D., Wang, X., 2018. API method recommendation without worrying about the task-API knowledge gap. In: 2018 33rd IEEE/ACM International Conference on Automated Software Engineering. ASE, IEEE, pp. 293–304.
Huangfu, L., Surdeanu, M., 2018. Bootstrapping polar-opposite emotion dimensions from online reviews. In: Proceedings of the Eleventh International Conference on Language Resources and Evaluation. LREC, ACL Anthology.
Huangfu, L., Zeng, D., 2018. Social media-based overweight prediction using deep learning. In: Proceedings of Annual Americas Conference on Information Systems. AMCIS, Association for Information Systems.
Kamani, M.M., Farhang, S., Mahdavi, M., Wang, J.Z., 2019. Targeted meta-learning for critical incident detection in weather data. In: International Conference on Machine Learning, Workshop on" Climate Change: How Can AI Help, Vol. 3.
Khan, S.H., Hayat, M., Bennamoun, M., Sohel, F.A., Togneri, R., 2017. Cost-sensitive learning of deep feature representations from imbalanced data. IEEE Trans. Neural Netw. Learn. Syst. 29 (8), 3573–3587.
Lapin, M., Hein, M., Schiele, B., 2017. Analysis and optimization of loss functions for multiclass, top-k, and multilabel classification. IEEE Trans. Pattern Anal. Mach. Intell. 40 (7), 1533–1554.
Lee, C.-W., Fang, W., Yeh, C.-K., Frank Wang, Y.-C., 2018. Multi-label zero-shot learning with structured knowledge graphs. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1576–1585.
Lee, Y.-Y., Ke, H., Yen, T.-Y., Huang, H.-H., Chen, H.-H., 2020. Combining and learning word embedding with WordNet for semantic relatedness and similarity measurement. J. Assoc. Inform. Sci. Technol. 71 (6), 657–670.
Lee, H.-y., Vu, N.T., Li, S.-W., 2021. Meta learning and its applications to natural language processing. In: Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: Tutorial Abstracts. pp. 15–20.
Li, Q., Qiao, M., Bian, W., Tao, D., 2016. Conditional graphical lasso for multi-label image classification. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2977–2986.

Ling, C.-Y., Zou, Y.-Z., Lin, Z.-Q., Xie, B., 2019. Graph embedding based api graph search and recommendation. J. Comput. Sci. Tech. 34 (5), 993–1006.

Ling, C., Zou, Y., Xie, B., 2021. Graph neural network based collaborative filtering for API usage recommendation. In: 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering. SANER, IEEE, pp. 36–47.

McMillan, C., Grechanik, M., Poshyvanyk, D., Xie, Q., Fu, C., 2011. Portfolio: Finding relevant functions and their usage. In: Proceedings of the 33rd International Conference on Software Engineering. pp. 111–120.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J., 2013. Distributed representations of words and phrases and their compositionality. In: Advances in Neural Information Processing Systems. pp. 3111–3119.

Minaee, S., Kalchbrenner, N., Cambria, E., Nikzad, N., Chenaghlu, M., Gao, J., 2021. Deep learning–based text classification: A comprehensive review. ACM Comput. Surv. 54 (3), 1–40.

Nguyen, P.T., Di Rocco, J., Di Ruscio, D., Ochoa, L., Degueule, T., Di Penta, M., 2019. Focus: A recommender system for mining api function calls and usage patterns. In: 2019 IEEE/ACM 41st International Conference on Software Engineering. ICSE, IEEE, pp. 1050–1060.

Nguyen, A.T., Nguyen, T.N., 2015. Graph-based statistical language model for code. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Vol. 1. IEEE, pp. 858–868.

Niu, H., Keivanloo, I., Zou, Y., 2017. API usage pattern recommendation for software development. J. Syst. Softw. 129, 127–139.

Ouyang, W., Wang, X., Zhang, C., Yang, X., 2016. Factors in finetuning deep model for object detection with long-tail distribution. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 864–873.

Qi, L., He, Q., Chen, F., Dou, W., Wan, S., Zhang, X., Xu, X., 2019. Finding all you need: web APIs recommendation in web of things through keywords search. IEEE Trans. Comput. Soc. Syst. 6 (5), 1063–1072.

Raghothaman, M., Wei, Y., Hamadi, Y., 2016. Swim: Synthesizing what i meancode search and idiomatic snippet synthesis. In: 2016 IEEE/ACM 38th International Conference on Software Engineering. ICSE, IEEE, pp. 357–367.

Rahman, M.M., Roy, C.K., Lo, D., 2016. Rack: Automatic api recommendation using crowdsourced knowledge. In: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, Vol. 1. SANER, IEEE. pp. 349–359.

Rasiwasia, N., Costa Pereira, J., Coviello, E., Doyle, G., Lanckriet, G.R., Levy, R., Vasconcelos, N., 2010. A new approach to cross-modal multimedia retrieval. In: Proceedings of the 18th ACM International Conference on Multimedia. pp. 251–260.

Ridnik, T., Lawen, H., Noy, A., Ben Baruch, E., Sharir, G., Friedman, I., 2021. Tresnet: High performance gpu-dedicated architecture. In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision. pp. 1400–1409.

Robillard, M.P., 2009. What makes APIs hard to learn? Answers from developers. IEEE Softw. 26 (6), 27–34.

Santos, A.L., Prendi, G., Sousa, H., Ribeiro, R., 2017. Stepwise API usage assistance using n-gram language models. J. Syst. Softw. 131, 461–474.

Sarafianos, N., Xu, X., Kakadiaris, I.A., 2018. Deep imbalanced attribute classification using visual attention aggregation. In: Proceedings of the European Conference on Computer Vision. ECCV, pp. 680–697.

Simonyan, K., Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

Su, J., 2020. URL https://www.kexue.fm/archives/7359.

Sun, Y., Cheng, C., Zhang, Y., Zhang, C., Zheng, L., Wang, Z., Wei, Y., 2020. Circle loss: A unified perspective of pair similarity optimization. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 6398–6407.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A., 2015. Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1–9.

Tan, M., Le, Q., 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In: International Conference on Machine Learning. PMLR, pp. 6105–6114.

Wang, J., Dang, Y., Zhang, H., Chen, K., Xie, T., Zhang, D., 2013. Mining succinct and high-coverage API usage patterns from source code. In: 2013 10th Working Conference on Mining Software Repositories. MSR, IEEE. pp. 319–328.

Wang, Z., Huang, S., Liu, Z., Yan, M., Xia, X., Wang, B., Yang, D., 2021. Plot2API: Recommending graphic API from plot via semantic parsing guided neural network. In: 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering. SANER, IEEE, pp. 458–469.

Wang, Y.-X., Ramanan, D., Hebert, M., 2017. Learning to model the tail. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. pp. 7032–7042.

Wang, Y.-X., Ramanan, D., Hebert, M., 2019. Meta-learning to detect rare objects. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 9925–9934.

Xie, S., Girshick, R., Dollár, P., Tu, Z., He, K., 2017. Aggregated residual transformations for deep neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1492–1500.

You, R., Guo, Z., Cui, L., Long, X., Bao, Y., Wen, S., 2020. Cross-modality attention with semantic graph embedding for multi-label classification. In: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 34, no. 07. pp. 12709–12716.

Yu, W., Yang, T., Chen, C., 2021. Towards resolving the challenge of long-tail distribution in UAV images for object detection. In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision. pp. 3258–3267.

Zhang, Y., Cheng, D.Z., Yao, T., Yi, X., Hong, L., Chi, E.H., 2021. A model of two tales: Dual transfer learning framework for improved long-tail item recommendation. In: Proceedings of the Web Conference 2021. pp. 2220–2231.

Zhang, J., Wu, Q., Shen, C., Zhang, J., Lu, J., 2018. Multilabel image classification with regional latent semantic dependencies. IEEE Trans. Multimed. 20 (10), 2801–2813.

Zhong, H., Xie, T., Zhang, L., Pei, J., Mei, H., 2009. MAPO: Mining and recommending API usage patterns. In: European Conference on Object-Oriented Programming. Springer, pp. 318–343.

Zou, Y., Yu, Z., Kumar, B., Wang, J., 2018. Unsupervised domain adaptation for semantic segmentation via class-balanced self-training. In: Proceedings of the European Conference on Computer Vision. ECCV, pp. 289–305.

Rong Qin is currently pursuing the B.S. degree at the School of Big Data and Software Engineering, Chongqing University, P.R. China. He is committed to using computer vision technology and machine learning to solve application problems and develop intelligent software.

Zeyu Wang was a master student at the School of Big Data and Software Engineering, Chongqing University, P.R. China. Her research interests are pattern recognition and intelligent software engineering.

Sheng Huang received his BEng and Ph.D. degrees both from Chongqing University, Chongqing, P.R. China, in 2010 and 2015 respectively. He was a visiting Ph.D. student at the department of computer science, Rutgers University, New Brunswick, NJ, USA, from 2012 to 2014. He is currently an associate professor at the school of big data and software engineering, Chongqing University. He has authored/coauthored more than 50 scientific papers in venues, such as CVPR, AAAI, TIP, TIFS and TCSVT. His research interests include computer vision, machine learning, intelligent software engineering and artificial intelligent applications.

Luwen Huangfu received the B.S. degree in software engineering from Chongqing University, Chongqing, China, the M.S. degree in computer science from the Chinese Academy of Sciences, Beijing, China, and the Ph.D. degree in management information systems from University of Arizona, Tucson, AZ, USA. She is currently an Assistant Professor with Fowler College of Business, San Diego State University, San Diego, CA, USA, where she is also with the Center for Human Dynamics in the Mobile Age. She has authored/coauthored more than 30 scientific papers in venues, such as IEEE Transactions on Cybernetics, IJCAI, ACM MM, Journal of Medical Internet Research (JMIR), IEEE Transactions on Intelligent Transportation Systems, Pacific Asia Journal of the Association for Information Systems, IEEE Signal Processing Letters, LREC, ICASSP, and IEEE ISI. Her research interests include business analytics, text mining, data mining, software management, computer vision, artificial intelligence, and healthcare management.