Journal of
Information Systems Engineering
and Business Intelligence

Vol.9, No.2, October 2023
Available online at: http://e-journal.unair.ac.id/index.php/JISEBI

Transfer Learning based Low Shot Classifier for Software Defect Prediction

Vikas Suhag [1)*], Sanjay Kumar Dubey [2)], Bhupendra Kumar Sharma [3)]

[1)2)]*Department of Computer Science and Engineering, Amity School of Engineering and technology, Amity University Uttar Pradesh, Noida, India*

[1)]vikasuhag@hotmail.com, [2)]skdubey1@amity.edu

[3)]*Northern India Textile Research Association, Ghaziabad, Uttar Pradesh, India*

[3)]drbkjpr@ymail.com

Abstract

Background: The rapid growth and increasing complexity of software applications are causing challenges in maintaining software quality within constraints of time and resources. This challenge led to the emergence of a new field of study known as Software Defect Prediction (SDP), which focuses on predicting future defect in advance, thereby reducing costs and improving productivity in software industry.
Objective: This study aimed to address data distribution disparities when applying transfer learning in multi-project scenarios, and to mitigate performance issues resulting from data scarcity in SDP.
Methods: The proposed approach, namely Transfer Learning based Low Shot Classifier (TLLSC), combined transfer learning and low shot learning approaches to create an SDP model. This model was designed for application in both new projects and those with minimal historical defect data.
Results: Experiments were conducted using standard datasets from projects within the National Aeronautics and Space Administration (NASA) and Software Research Laboratory (SOFTLAB) repository. TLLSC showed an average increase in F1-Measure of 31.22%, 27.66%, and 27.54% for project AR3, AR4, and AR5, respectively. These results surpassed those from Transfer Component Analysis (TCA+), Canonical Correlation Analysis (CCA+), and Kernel Canonical Correlation Analysis plus (KCCA+).
Conclusion: The results of the comparison between TLLSC and state-of-the-art algorithms, namely TCA+, CCA+, and KCCA+ from the existing literature consistently showed that TLLSC performed better in terms of F1-Measure.

Keywords: Just-in-time, Defect Prediction, Deep Learning, Transfer Learning, Low Shot Learning

Article history: Received 18 June 2023, first decision 28 August 2023, accepted 7 October 2023, available online 28 October 2023

I. INTRODUCTION

Software requirements are becoming increasingly complex in today's landscape and as the complexity of software grows, its development and maintenance present significant challenges. The foremost issue encountered by development teams is the comprehensive testing of software. However, testing cannot guarantee the absence of defect, necessitating continuous testing even after release and this makes the process to be laborious and costly. When the prospect of source code defect can be identified in advance, based on prior defect knowledge, the solution is known as Software Defect Prediction (SDP), which leverages machine learning or deep learning to construct models. The effectiveness of SDP models relies heavily on the quality and quantity of data used for training. This data comprises various elements, including software metrics of source files, developer experience, log files, and previous bug fixes. SDP models fall into the category of supervised models, necessitating well-labeled data for training predictive models. Moreover, it is important to be aware that many studies have discussed the scarcity of labeled data [1] [2] [3].

Acquiring labeled data for constructing supervised learning models is a resource-intensive, time-consuming process that demands considerable effort when conducted manually. Consequently, studies have explored the application of multiple levels of transfer learning to address the data scarcity and distribution disparities, which often lead to performance overhead. J. Chen et al. [4] and J. Bai et al. [5] applied three levels of transfer learning model training, while another study used a two-level model training approach [6]. Tang et al. even went further by implementing multiple classifier training iterations before constructing the final classifier [7]. In one study, the source code was initially converted into images, followed by the application of deep transfer learning to

* Corresponding author

ISSN 2443-2555 (online) 2598-6333 (print)
doi: http://dx.doi.org/10.20473/jisebi.9.2.228-238

detect defective code from the image data [8]. However, it is crucial to acknowledge that excessive use of multilevel training can counteract the primary purpose of SDP by consuming resources and adding substantial time to the training and retraining of models using neural networks. This shows the need for tools that can efficiently collect data or propose alternative approaches requiring less data and fewer iterations for accurate prediction.

The two approaches that can address these challenges are Transfer Learning and Low Shot Learning. Deep learning, in particular, demands an extensive amount of data for model training but offers advanced ways to repurpose trained models. An example is a scenario where deep learning model is trained using a dataset containing 10 million rows. Training this kind of model is often time-consuming and resource-intensive, hence, it is more practical to freeze the trained models and apply them to test a different target dataset, rather than going through the entire retraining process. This process of transferring knowledge from source data to make prediction on target data is known as deep transfer learning.

Low shot learning presents a solution to the challenge of limited data availability. Essentially, it consists of working with smaller data sets, which differs from the standard practice for constructing machine learning and deep neural network models [1].

In the proposed SDP approach, referred to as Transfer Learning based Low Shot Classifier (TLLSC), transfer learning models serve as a valuable knowledge source for low shot learning. This approach improves predictive accuracy, even when operating with minimal training data, as shown in Fig. 1. SDP approach effectively addresses the issue of data distribution disparities in multi-project scenarios and alleviates performance challenges arising from data scarcity.

The subsequent sections of this paper contain Section II, which offers an overview of the existing studies in the literature review. Section III focuses on the specifics of the proposed approach and Section IV discusses the results and inferences. Section V provides insights derived from the results, while Section VI describes the conclusion of the study.

II. LITERATURE REVIEW

SDP has recently become prominent in the recent years due to the advancement of processing hardware such as robust CPUs and GPUs. This progress has made the development and training of software models for prediction more accessible but there are still critical gaps that require attention in this field.

In the context of Unified Modeling Language (UML) class and sequence diagram classification sourced from the Github repository, Low Shot Learning has found application [1]. The approach comprised the use of a four-layer convolutional neural network for low shot learning. It was observed that low shot learning had traditionally been effective for image data but was less suitable for conventional textual data. Wang et al. [9] introduced a solution to address disparities in metric distribution and class imbalances in heterogeneous defect prediction, which consists of using data from one project to predict defect in another. The study solved the class imbalance challenge through undersampling and addressed distribution differences. Another study addressed distribution differences between source and target projects, and proposed a feature-matching solution that selected features and their distribution curves from both source and target projects to mitigate heterogeneity [10]. In response to distribution disparities, one study suggested a source project selection process in which two sources with matching distributions to the target projects were chosen for maximum performance [6]. Two Transfer Component Analysis (TCA+) models were then constructed for each source-target combination, resulting in improved prediction results.

Transfer Naive Bayes (TNB) was applied to analyze the distribution of test data and transform the training input into weights, which could then be used to construct a model [11]. In the process of improving intrusion detection system accuracy, a solution was proposed. This comprised the use of data from various layers of the Convolutional Neural Network (CNN) and the application of Linear Support Vector Machine (SVM) and Single Neural Network (1-NN) classifier for few-shot intrusion detection [3].

AdaBoost, also known as adaptive boosting, combines multiple weak classifiers into a robust, single strong classifier. Additionally, it assigns more weight to challenging instances for classification and less weight to those that are easier to classify. AdaBoost is versatile, addressing differences in metrics and distribution in datasets, as well as the challenge of class imbalances [9]. It also addresses the issue of insufficient data for creating supervised learning models, introducing Few-Shot Learning-Based Balanced Distribution Adaptation (FSLBDA). The approach leverages Extreme Gradient Boosting to eliminate redundant metrics from data, as opposed to rows. Moreover, Gradient Boosting is a class of ensemble machine learning algorithms suitable for both classification and regression, where ensembles are constructed from decision tree models added iteratively, correcting prediction errors of earlier models.

Domain adaptation falls within the context of machine learning, addressing situations where models trained on a source distribution are applied within a different target distribution. There should be a degree of relatedness between the source and target domains. The objective of domain adaptation is to grapple with varying metrics across projects while achieving satisfactory results with machine learning models. In the study, Usherwood and

Smit [12] compared deep transfer learning with classical machine learning approaches for low shot classification. Given that most machine learning and deep learning approaches demand substantial data for training, the exploration focused on low shot scenario, including 100 to 1000 labeled examples per class.

Data scarcity has consistently caused a significant challenge for prediction models. To address this issue, many studies have combined data from multiple projects to create a larger training dataset. However, the extensive data collection may contain irrelevant information that needs to be filtered out before training the models. A semi-supervised data filtration approach was introduced by using the Semi-Supervised Density-Based Spatial Clustering of Applications with Noise (SSDBSCAN) filter. Additionally, the multi-source Transfer AdaBoost (TrAdaBoost) algorithm was introduced for transfer learning [13].

In cross-project defect prediction [14], the data distribution may differ, but their attribute sets need to be same. This requirement is contrary to transfer learning [15][16][17], a framework proposed in 2019 for identifying bug classes. Bugs were classified as follows:

1) Bohrbugs, which are easily reproducible under controlled conditions,

2) Mandelbugs, which prove challenging to reproduce using traditional approaches [18].

The study by [18] used the TrAdaBoost approach of instance-based transfer learning, incorporating three different classifier, namely the Gradient Boosting Classifier (GBC), the Stochastic Gradient Descent Classifier (SGD), and the AdaBoost Classifier (ABC). One-to-one project mapping was used for both training and testing. These two projects may belong to the same company, with the developer team potentially sharing common members. Cross-company projects were used to develop transfer learning models, emphasizing the need to eliminate irrelevant data from multiple projects used in model development.

In existing literature, low shot learning has primarily found application in computer vision, image detection [19][20], and natural language processing. In these domains, even a single labeled sample can suffice for training and constructing prediction models. However, this does not hold in SDP, which relies on source code metrics data. Applying low shot concept in the context of SDP differs substantially from working with image data. The number of samples required to achieve acceptable results in SDP is considerably higher compared to image data [2]. The following section introduces solutions to these key challenges.

III. METHOD

Transfer learning served as a mechanism for constructing a knowledge base from various data sources, establishing the foundation for model training. This approach effectively resolved the challenges linked to data scarcity, while low shot learning [21][22] improved project performance in situations consisting of limited datasets. The combined influence of transfer learning and low shot learning significantly enhanced prediction accuracy.

In this paper, TLLSC was introduced to address concerns related to data scarcity and distribution disparities among datasets in SDP. The proposed SDP approach integrated deep transfer learning and low shot learning in a hierarchical manner. The primary model was trained using data from multiple projects characterized by minimal distribution differences. Subsequently, it went through retraining using smaller projects with inadequate data to create a comprehensive prediction model. The knowledge acquired from multiple projects was transferred to form a generalized model, improving its capability to address low shot challenges.

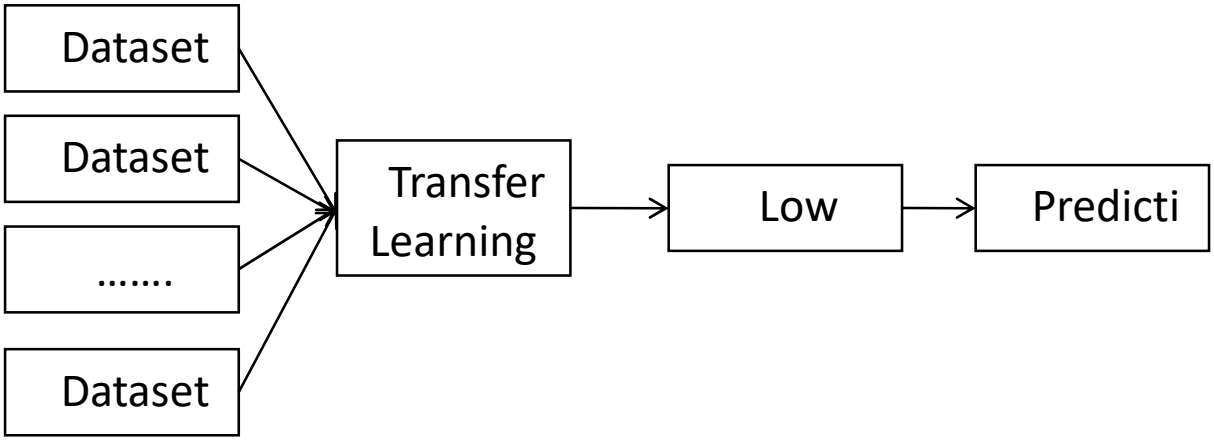

Fig. 1 Flowchart of the proposed approach

A. Dataset

This study used six projects from the existing literature, comprising three projects from the National Aeronautics and Space Administration (NASA) and three projects from Software Research Laboratory (SOFTLAB) repository. These projects were well-established and consistent with the study objectives, offering reliability and precision. Furthermore, the dataset sizes conformed to the study's requirements, as shown in Table 1.

TABLE 1
DATASET SIZE

Repository Name	**Project Name**	**Data Size (Rows, Features)**
SOFTLAB	AR3	(63,21)
SOFTLAB	AR4	(107,21)
SOFTLAB	AR5	(36,21)
NASA	CM1	(327,21)
NASA	MW1	(253,21)
NASA	PC1	(705,21)

B. *Performance metrics*

To assess the performance of the model, a calculation of four widely adopted metrics was executed, namely Accuracy, Precision, Recall, and F1 Measure. These metrics were fundamental for evaluating model effectiveness. It was essential to be aware that relying solely on Precision and Recall individually could produce skewed results in a generalized context. Therefore, the F1 measure was chosen to represent the harmonic mean of Precision and Recall, in order to obtain a comprehensive assessment. All metrics were computed from the confusion matrix, allowing for the recording of True Positive (TP), False Positive (FP), False Negative (FN), and True Negative (TN) values to validate performance.

C. *Data Pre-processing*

Prerequisite to data usage in SDP was data cleaning, optimizing the final model's performance. The proposed algorithm removed duplicate rows in each project dataset, retaining only the initial instance. Furthermore, it addressed rows with null values by substituting them with the feature's mean. The dataset may also have contained outliers, which were excluded when their z-score values exceeded 3, improving the final SDP model's performance.

D. Normalization

Given the diversity of software metrics in each project, each metric showed variation in scale and units. To resolve this discrepancy, normalization was crucial to scale metrics within the range of 0 to 1 using MinMaxScaler, facilitating the model's convergence to local minima.

E. Standardization

With the normalization of metrics within individual project datasets, discrepancies in standard deviation values for identical metrics across different projects posed challenges when consolidating data for training deep learning models. Standardization played a crucial role in supporting software metrics from all projects within a consistent value range via StandardScaler.

F. *Sub sampling or clustering*

Combining multiple project datasets often resulted in a substantial dataset. Large datasets were presumed to contain homogeneous data regions that clustering could identify. Each homogeneous cluster had a unique value range, suitable for serving as training samples for target data, contingent on cluster compatibility. For example, a combined dataset could be grouped into three homogeneous clusters, each showing a heterogeneous distribution of values.

G. Feature selection

The number of rows and features in the dataset significantly impacted deep learning model performance. This simply implied that reducing features played a very important role in SDP model. Software metrics dataset included features loosely connected to the final dependent variable. The algorithm incorporated chi-square-based feature selection through the scikit-learn Python library, thereby eliminating features with minimal impact on prediction classifier and greatly improving model performance.

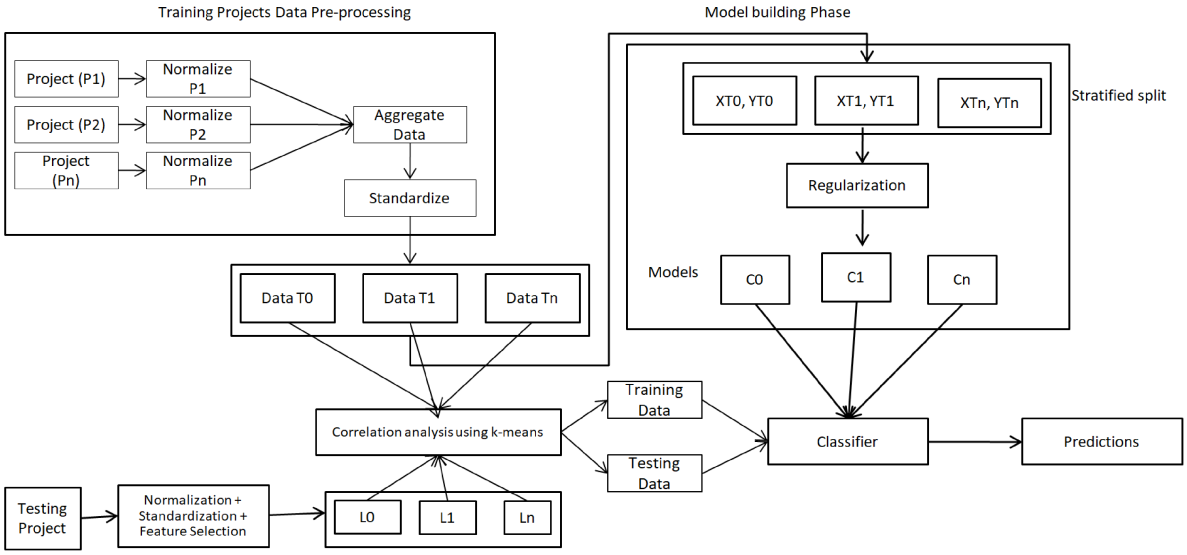

Fig. 2 TLLSC Framework

Algorithm 1

Transfer Learning based Low Shot Classifier (TLLSC)

```
Set project as p1,p2, pn // P1, P2 … Pn denotes software projects metrics dataset
set Testing project as L // L denote the target project software metrics dataset
function TestDataProcessing()
begin
   min_max_scaler(L)
      return Ln
   standard_scaler(Ln) // Ln denotes normalized target dataset
      return Lns
   feature_selection(Lns)
      apply_chi_square(Lns) // Lns denotes scaled Ln
         return Lns'          // Lns' denotes Lns with only selected features.
   create_clusters(Lns')
      return L0,L1, ...Ln  // L1, L2....Ln denote the clusters of target dataset L after preprocessing
end
function build_model()
begin
   set T0,T1... Tn = TrainingDataPreProcessing()
   for each T0, T1... Tn
      apply_stratified_split()
         return XT0,YT0……XTn,YTn
foreach XT0,YT0……XTn,YTn
      function generate_CNN(XT0, YT0)
         apply_dropout_regularization()
            return C0 // C0 denote CNN model corresponding to XT0 and YT0
      return C0, C1....Cn
end
function Predictor()
begin
   for each pair D0(T0,L0), D1(T1,L1), Dn(Tn, Ln)
      function find_correlation(D) // D denotes Training and testing data pair(T,L)
         return correlation_coefficient (CC)
      function MaxCorrelation(CC)
         return D
   selectBestModel from build_model()
         return C
// C denotes best classification model and D denotes Highest correlated Training and testing data pair.
function   MakePrediction(C,D)
end
```

In order to generalize the models, n-clusters will generate n-models, each of which will become prediction model based on the correlation with target set.

H. Transfer Learning based Low Shot Classifier (TLLSC)

The proposed SDP framework, known as TLLSC, is a system where data dimensions of each dataset were shown in

TABLE 1. The TLLSC framework can be seen on Fig. 2. These datasets comprised different software metrics, including total lines of code (LOC), blank LOC, cyclomatic complexity, condition count, decision count, comment LOC, total operators, total operands, total unique operators, total unique operands, halstead length, halstead volume, halstead difficulty, halstead error, halstead time, design complexity, and labels. Additionally, the labels signified whether the source file was defective or defect-free, serving as the dependent variable. Deep learning models scrutinized these dataset attributes for model training. After training, the model became capable of predicting defect proneness of source files based on insights gleaned from the labeled dataset.

The lack of labeled data created a significant challenge in software industry, a matter observed in numerous literature reviews. To address the issue, TLLSC consolidated data from multiple software projects, creating a dataset of sufficient size for training CNN models. These datasets often showed variations in metric values, measurement scales, and condition counts due to different development teams. To rectify these distribution disparities, pre-processing, normalization, and standardization were applied. As outlined in Algorithm 1, once the datasets were standardized, they were used for CNN model training. The "TrainingDataProcessing*()*" function generated a combined dataset for the model-building phase, as shown in Fig. . These datasets were divided into training and testing sets, followed by dropout regularization to fortify the CNN models. The best model was chosen as the final classifier (C) based on the accuracy achieved in the model-building phase.

The issue of scarcity persisted in new projects that lacked sufficient data for training CNN models. In this context, TLLSC leveraged the combined dataset created through the "TrainingDataProcessing*()*" function. This dataset went through scaling, aggregation, and clustering to generate test data. The "TestingDataProcessing*()*" function was then applied to the smaller project dataset, producing a highly correlated mixed dataset (D) using k-means correlation analysis between clusters from the combined dataset and the smaller test dataset, as shown in Fig. . Subsequently, classifier (C) was trained on dataset D, enabling prediction for all new, unseen data in the test project.

IV. RESULTS

In the study, experiments were conducted including three NASA projects (CM1, MW1, and PC1) and three projects from the SOFTLAB repository (AR3, AR4, and AR5). The CNN model was trained with various activation functions, optimizers, loss functions, and iterations to ensure effective prediction performance.
The experiments were carried out on a computer equipped with an Intel i5 dual-core processor and 8 gigabytes (GB) of RAM. The entire experiment was developed using Python 3.9, incorporating scikit-learn, TensorFlow, and other related Python packages. The hyperparameters used for fine-tuning the CNN model are shown in Table 2.

TABLE 2
HYPER PARAMETERS USED FOR TUNING THE CNN MODEL

Parameter name	**Value**
Batch Size	16 and 32
Epochs	50, 100, 200
Optimizers	adam
Train-Test Split Ratio	70:30
Drop-out Ratio	0.2, 0.3
Activation Function	relu
Number of Layers	3
Learning Rate	0.01

The approach consisted of selecting three NASA projects (CM1, MW1, and PC1) for building the CNN models, and three SOFTLAB projects were also used as the target dataset to evaluate the effectiveness. The proposed TLLSC model was applied to the dataset mentioned in

TABLE 1, using different projects for training the model.

The results, as shown in Table 2, included the outcomes of the experiments along with the sizes of the training and target datasets. Cluster 1 showed the highest accuracy when MW1 served as the training dataset and AR5 as the target dataset. On the other hand, cluster 2 showed lowest accuracy with the same combination of MW1 and AR5. This implied that cluster 1 was more inclined with the target data, showing a high level of homogeneity, while cluster 2 seemed to contain outliers or presented heterogeneity.

TABLE 3
RESULTS FROM THE EXPERIMENT USING NASA PROJECT AS TRAINING AND SOFTLAB PROJECT AS TARGET

Training Data	Target Data	Training size	Target size	Accuracy (%)	Precision (%)	Recall (%)	F1 Measure (%)	TP	FP	FN	TN	Cluster
CM1	AR3	(42, 21)	(63, 21)	92	93.1	98.1	95.5	54	4	1	4	1
MW1	AR3	(46, 21)	(63, 21)	93.6	93.2	100	96.4	55	4	0	4	1
PC1	AR3	(653, 21)	(63, 21)	88.8	88.7	100	94.0	55	7	0	1	1
CM1	AR3	(285, 21)	(63, 21)	88.8	88.7	100	94.0	55	7	0	1	2
MW1	AR3	(207, 21)	(63, 21)	88.8	91.3	96.3	93.8	53	5	2	3	2
PC1	AR3	(52, 21)	(63, 21)	90.4	90.1	100	94.8	55	6	0	2	2
CM1	AR4	(293, 21)	(107, 21)	86.9	89.2	95.4	92.2	83	10	4	10	1
MW1	AR4	(45, 21)	(107, 21)	86.9	86.8	98.8	92.4	86	13	1	7	1
PC1	AR4	(654, 21)	(107, 21)	83.1	83.4	98.8	90.5	86	17	1	3	1
CM1	AR4	(34, 21)	(107, 21)	82.2	82.0	100	90.1	87	19	0	1	2
MW1	AR4	(208, 21)	(107, 21)	85.0	85.1	98.8	91.4	86	15	1	5	2
PC1	AR4	(51, 21)	(107, 21)	85.9	87.5	96.5	91.8	84	12	3	8	2
CM1	AR5	(285, 21)	(36, 21)	88.8	92.8	92.8	92.8	26	2	2	6	1
MW1	AR5	(46, 21)	(36, 21)	94.4	93.3	100	96.5	28	2	0	6	1
PC1	AR5	(653, 21)	(36, 21)	83.3	82.3	100	90.3	28	6	0	2	1
CM1	AR5	(42, 21)	(36, 21)	91.6	93.1	96.4	94.7	27	2	1	6	2
MW1	AR5	(207, 21)	(36, 21)	77.7	77.7	100	87.5	28	8	0	0	2
PC1	AR5	(52, 21)	(36, 21)	91.6	96.2	92.8	94.5	26	1	2	7	2

A closer examination showed that when CM1 was used as the training project, the highest F1-measure value, reaching 94%, was observed with AR3 as the target project. Subsequently, with AR4, Kernel Canonical Correlation Analysis plus (KCCA+) achieved 84%, Canonical Correlation Analysis (CCA+) was 80%, and Transfer Component Analysis (TCA+) attained 40%. TLLSC also showed excellent performance with maximum F1-measure values of 95% and 94% when MW1 and PC1 were used as training projects, respectively.

These results showed that TLLSC consistently outperformed the other approaches across various combinations of training and target projects with different sizes. Furthermore, the F1-measure values for TLLSC showed less variation compared to the other approaches, signifying its robustness in effectively handling variations.

To further evaluate the effectiveness of the proposed approach, an additional experiment was conducted using a different configuration. Data from three projects (CM1, MW1, and PC1) were combined to create a larger dataset for training the model, and then the model was sequentially tested on three SOFTLAB projects (AR3, AR4, and AR5).

Table 3 showed the experimental results for each cluster, including accuracy, precision, recall, and F1-measure. TLLSC achieved a maximum F1-measure of 94.8% and a minimum of 90.2%. TLLSC showed an impressive accuracy of 88.9% even with a small dataset containing only 36 rows, with maximum values shown in bold and minimum values underlined.

TABLE 4
RESULTS FROM THE EXPERIMENT USING COMBINED 3 NASA PROJECTS AS TRAINING AND SOFTLAB PROJECTS AS TARGET

Training Data	Target Data	Training size	Target size	Accuracy (%)	Precision (%)	Recall (%)	F1 Measure (%)	TP	FP	FN	TN	Cluster
CM1, MW1, PC1	AR3	(1112,21)	(63, 21)	90.5	91.5	98.2	94.7	54	5	1	3	1
CM1, MW1, PC1	AR4	(1112,21)	(107, 21)	82.2	82.1	100	<u>90.2</u>	87	19	0	1	1
CM1, MW1, PC1	AR5	(1112,21)	(36, 21)	88.9	87.5	100	93.3	28	4	0	4	1
CM1, MW1, PC1	AR3	(173, 21)	(63, 21)	90.5	90.2	100	**94.8**	55	6	0	2	2
CM1, MW1, PC1	AR4	(173, 21)	(107, 21)	85.0	85.1	98.9	91.5	86	15	1	5	2
CM1, MW1, PC1	AR5	(173, 21)	(36, 21)	86.1	84.8	100	91.8	28	5	0	3	2

V. DISCUSSION

TLLSC, when applied to AR3 and AR4, achieved the highest F1-Measure of 95% and 92% when MW1 served as the source project. For AR5, the highest F1-Measure of 93% was attained with CM1 as the training source.

To prove the effectiveness of the proposed approach, a comparative analysis was conducted with state-of-the-art approaches from the literature [23] [24] [25]. The baseline approaches used identical datasets for their experiments, ensuring a robust and ethical comparison. Various evaluation metrics, including accuracy, precision, recall, and F1-Measure, were generated as prediction outputs. F1-Measure provided a balanced measure of performance compared to recall and precision, which showed an inverse relationship.

TABLE 4 showed a comparison of F1-Measure results achieved by different approaches, namely TCA+ [23], CCA+ [24], KCCA+ [25], and the proposed TLLSC. The evaluation was executed using distinct training projects,

with three projects, AR3, AR4, and AR5, as the target projects. The F1-Measure values in TABLE 4 showed the averages derived from both one-cluster and two-cluster scenarios.

TLLSC approach outperformed all other existing approaches examined in this study as it achieved a minimum F-Measure of 91% and a maximum of 95% when conducting one-to-one project prediction. When training with multi-project data, the F1-Measure ranged from 90% to 94%. However, the other approaches reached maximum values of 84%, 80%, and 55% for KCCA+, CCA+, and TCA+, respectively.

TABLE 4
F-MEASURE VALUES WHEN USING CM1, MW1 AND PC1 AS TRAINING PROJECTS AND AR3, AR4 AND AR5 AS TARGET PROJECTS

Training project	Target project	TCA+ (%)	CCA+ (%)	KCCA+ (%)	TLLSC (%)
	AR3	33	61	66	94
CM1	AR4	40	80	84	91
	AR5	37	71	77	93
	AR3	48	58	65	95
MW1	AR4	38	70	75	92
	AR5	55	68	72	91
	AR3	30	80	79	94
PC1	AR4	37	75	74	91
	AR5	50	76	80	92

A comparison of TLLSC's performance with the existing approach CCA+ [24] was carried out using the combined dataset of CM1, MW1, and PC1. The results in Table 5, clearly showed TLLSC's superiority across various combinations. Specifically, TLLSC outperformed CCA+ by 9%, 2%, and 12% when AR3, AR4, and AR5 were used as the target data in this combined configuration experiment.

The results provided additional validation of the superiority of the proposed TLLSC approach when compared to existing approaches. This findings was recorded because it consistently delivered better results across different data combinations and specifically outperformed CCA+ as seen at Table 6.

TABLE 5
F-MEASURE VALUES WHEN USING CM1, MW1 AND PC1 AS TRAINING PROJECT

Target project	CCA+ (%)	TLLSC (%)
AR3	85	94
AR4	88	90
AR5	80	92

Fig. 3 showed a comparative analysis of the proposed approach TLLSC with other state-of-the-art approaches such as TCA+, CCA+, and KCCA+ using F1-Measure performance metric. The bar graph showed the total comparison for one-to-one and many-to-one scenarios of model training and validation. All these experiments were conducted using standard datasets, making the proposed approach easily reproducible for validation.

The high predictive performance of TLLSC was positioned to enable the swift and efficient resolution of software defect, along with averting future issues through in-depth project code analysis in advance. This promised cost savings in software projects and an augmentation of productivity.

A. Threats to Validity

During the design of the proposed approach and the execution of experiments, specific assumptions were made that could introduce variability when replicating the study. These assumptions included aspects such as the number of epochs needed for training each model, the random state used for the training and testing division, and the impact of feature selection on performance outcomes. Different attribute combinations could produce diverse observations and results.

The computation of metrics, including accuracy, recall, precision, and F1-measure, could fluctuate based on the specific algorithm used. Python offered a range of pre-built packages with distinct implementations of these metrics, potentially resulting in varying values. In this study, all metric calculations were grounded in the confusion matrix, providing a robust foundation for evaluating model performance. Considering these factors, it was crucial to consider the assumptions made and meticulously replicate the experimental conditions to ensure consistent and dependable results.

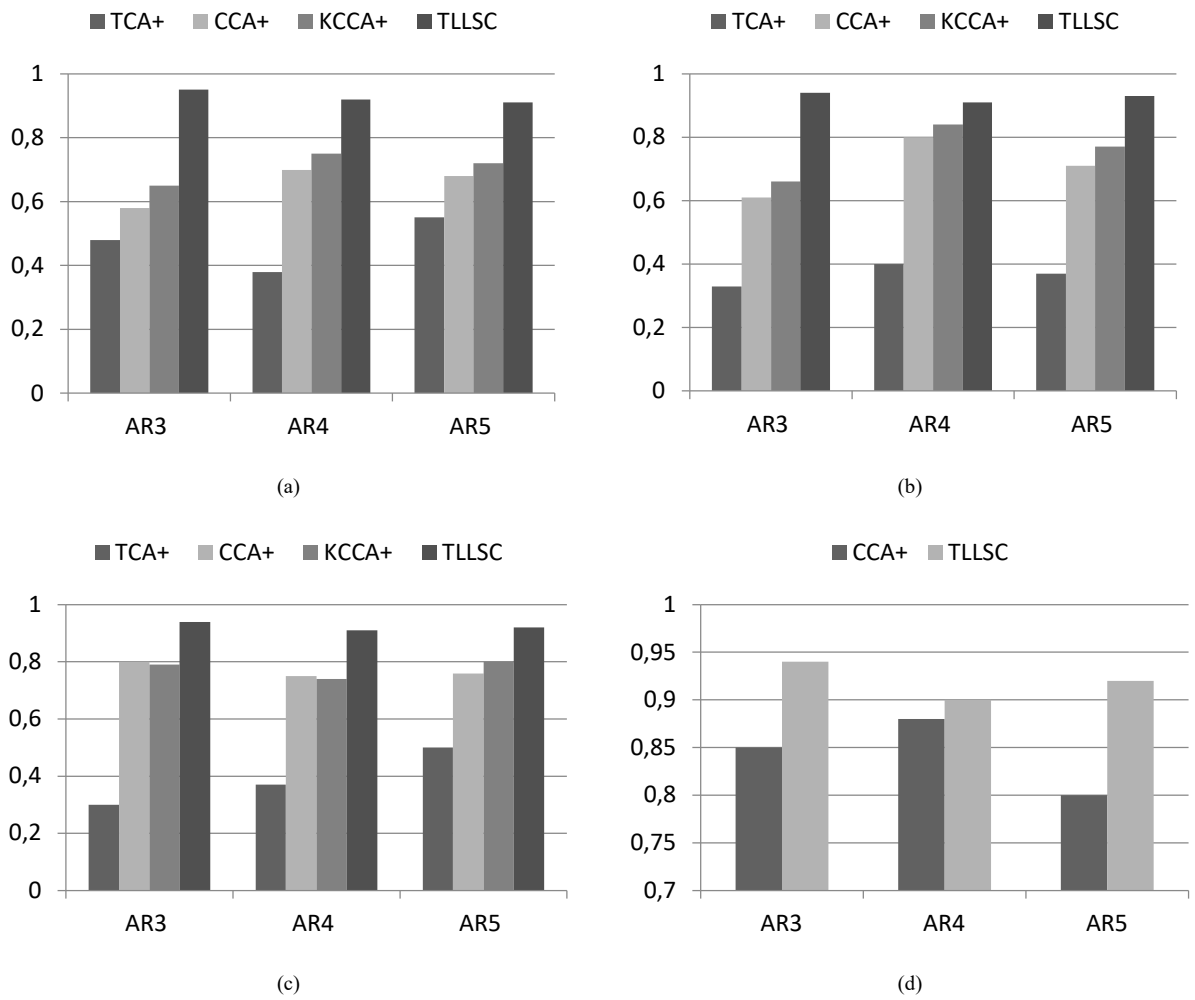

Fig. 3 Comparative analysis of TLLSC with other approaches (a) F-Measure values when using CM1 as training project (b) F-Measure values when using MW as training project. (c) F-Measure values when using PC1 as training project (d) F-Measure values when using CM1, MW1 and PC1 as training project

VI. CONCLUSION

In conclusion, the paper introduced a hybrid approach called TLLSC, which combined transfer learning and low shot learning to predict defect in projects with limited datasets. TLLSC was applied to three NASA projects for model training and three SOFTLAB projects for validation. The experimental results established the superiority of the proposed approach when compared with the existing ones in the literature.

These results affirmed that TLLSC adeptly addressed the challenges stemming from limited data availability and distribution variations. This proficiency was crucial for early detection and rectification of errors in software projects. The important role of transfer learning in constructing a knowledge base for low shot learning significantly improved model training. Both 1-to-1 and n-to-1 project ratios were used in the training and validation of the model.

Clustering the data into homogeneous sets produced an enhancement in the model's performance. TLLSC showed its effectiveness even with datasets as small as 36 rows, showing its capacity to handle projects with limited data.

Future investigations are needed to explore the generalizability of the approach and its application to larger and more diverse datasets.

Author Contributions: *Vikas Suhag:* Conceptualization, Methodology, Software, Formal Analysis, Investigation, data Curation, Writing - Original Draft, Visualization. *Sanjay Kumar Dubey:* Conceptualization, Resources, Writing

- Review & Editing, Supervision. *Bhupendra Kumar Sharma:* Conceptualization, Resources, Writing - Review & Editing, Supervision.

All authors have read and agreed to the published version of the manuscript.

Funding: research received no specific grant from any funding agency.

Conflicts of Interest: The authors declare no conflict of interest.

Data Availability: Datasets used in study are open source dataset, which are available on multiple websites on internet. Some sources are https://www.kaggle.com/datasets/aczy156/software-defect-prediction-nasa and https://figshare.com/articles/dataset/Dataset_For_Software_Defect_Predictions/19516858

Informed Consent: There were no human subjects.

Animal Subjects: There were no animal subjects..

ORCID:
Vikas Suhag: https://orcid.org/0000-0002-6341-6375
Sanjay Kumar Dubey: https://orcid.org/0000-0003-3808-6623
Bhupendra Kumar Sharma: https://orcid.org/0009-0003-1577-3647

REFERENCES

[1] J. Ott, A. Atchison, and E. J. Linstead, "Exploring the applicability of low-shot learning in mining software repositories," *Journal of Big Data*, vol. 6, no. 1, p. 35, Dec. 2019, doi: 10.1186/s40537-019-0198-z.

[2] W. Y. Chen, Y. C. F. Wang, Y. C. Liu, Z. Kira, and J. B. Huang, "A closer look at few-shot classification," *7th International Conference on Learning Representations, ICLR 2019*, no. 2018, pp. 1–17, 2019.

[3] M. M. U. Chowdhury, F. Hammond, G. Konowicz, C. Xin, H. Wu, and J. Li, "A few-shot deep learning approach for improved intrusion detection," *2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference, UEMCON 2017*, vol. 2018-Janua, pp. 1–7, 2018, doi: 10.1109/UEMCON.2017.8249084.

[4] J. Chen, Y. Yang, K. Hu, Q. Xuan, Y. Liu, and C. Yang, "Multiview transfer learning for software defect prediction," *IEEE Access*, vol. 7, pp. 8901–8916, 2019, doi: 10.1109/ACCESS.2018.2890733.

[5] J. Bai, J. Jia, and L. F. Capretz, "A three-stage transfer learning framework for multi-source cross-project software defect prediction," *Information and Software Technology*, vol. 150, Oct. 2022, doi: 10.1016/j.infsof.2022.106985.

[6] C. Liu, D. Yang, X. Xia, M. Yan, and X. Zhang, "A two-phase transfer learning model for cross-project defect prediction," *Information and Software Technology*, vol. 107, no. September 2018, pp. 125–136, Mar. 2019, doi: 10.1016/j.infsof.2018.11.005.

[7] S. Tang, S. Huang, C. Zheng, E. Liu, C. Zong, and Y. Ding, "A Novel Cross-Project Software Defect Prediction Algorithm Based on Transfer Learning," 2022.

[8] Z. Lu, R. H. Kazi, L. Y. Wei, M. Dontcheva, and K. Karahalios, "Software Visualization and Deep Transfer Learning for Effective Software Defect Prediction," *Proceedings of the ACM on Human-Computer Interaction*, vol. 5, no. CSCW1, Apr. 2021, doi: 10.1145/1122445.1122456.

[9] A. Wang, Y. Zhang, H. Wu, K. Jiang, and M. Wang, "Few-Shot Learning Based Balanced Distribution Adaptation for Heterogeneous Defect Prediction," *IEEE Access*, vol. 8, pp. 32989–33001, 2020, doi: 10.1109/ACCESS.2020.2973924.

[10] Q. Yu, S. Jiang, and Y. Zhang, "A feature matching and transfer approach for cross-company defect prediction," *Journal of Systems and Software*, vol. 132, pp. 366–378, 2017, doi: 10.1016/j.jss.2017.06.070.

[11] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," *Information and Software Technology*, vol. 54, no. 3, pp. 248–256, 2012, doi: 10.1016/j.infsof.2011.09.007.

[12] P. Usherwood and S. Smit, "Low-Shot Classification: A Comparison of Classical and Deep Transfer Machine Learning Approaches," Jul. 2019, [Online]. Available: http://arxiv.org/abs/1907.07543

[13] X. Yu, M. Wu, Y. Jian, K. E. Bennin, M. Fu, and C. Ma, "Cross-company defect prediction via semi-supervised clustering-based data filtering and MSTrA-based transfer learning," *Soft Computing*, vol. 22, no. 10, pp. 3461–3472, 2018, doi: 10.1007/s00500-018-3093-1.

[14] W. Wen, B. Zhang, X. Gu, and X. Ju, "An Empirical Study on Combining Source Selection and Transfer Learning for Cross-Project Defect Prediction," *IBF 2019 - 2019 IEEE 1st International Workshop on Intelligent Bug Fixing*, pp. 29–38, 2019, doi: 10.1109/IBF.2019.8665492.

[15] Y. Chen and X. Ding, "Research on cross - Project software defect prediction based on transfer learning," in *AIP Conference Proceedings*, 2018, p. 040083. doi: 10.1063/1.5033747.

[16] H. Qing, L. Biwen, S. Beijun, and Y. Xia, "Cross-Project Software Defect Prediction Using Feature-Based Transfer Learning," in *Proceedings of the 7th Asia-Pacific Symposium on Internetware - Internetware '15*, New York, New York, USA: ACM Press, 2015, pp. 74–82. doi: 10.1145/2875913.2875944.

[17] K. Li, Z. Xiang, T. Chen, S. Wang, and K. C. Tan, "Understanding the automated parameter optimization on transfer learning for cross-project defect prediction," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, New York, NY, USA: ACM, Jun. 2020, pp. 566–577. doi: 10.1145/3377811.3380360.

[18] X. Du, Z. Zhou, B. Yin, and G. Xiao, "Cross-project bug type prediction based on transfer learning," *Software Quality Journal*, vol. 28, no. 1, pp. 39–57, Mar. 2020, doi: 10.1007/s11219-019-09467-0.

[19] Y. Li *et al.*, "Incremental Few-Shot Object Detection for Robotics," pp. 1–11, May 2020.

[20] Q. Sun, Y. Liu, T. S. Chua, and B. Schiele, "Meta-transfer learning for few-shot learning," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Dec. 2019, pp. 403–412. doi: 10.1109/CVPR.2019.00049.

[21] A. Li, W. Huang, X. Lan, J. Feng, Z. Li, and L. Wang, "Boosting Few-Shot Learning with Adaptive Margin Loss."

[22] O. Melamud, M. Bornea, and K. Barker, "Combining unsupervised pre-training and annotator rationales to improve low-shot text classification," *EMNLP-IJCNLP 2019 - 2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing, Proceedings of the Conference*, no. 2011, pp. 3884–3893, 2020, doi: 10.18653/v1/d19-1401.

[23] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," *Proceedings - International Conference on Software Engineering*, pp. 382–391, 2013, doi: 10.1109/ICSE.2013.6606584.

[24] X. Jing, F. Wu, X. Dong, F. Qi, and B. Xu, "Heterogeneous cross-company defect prediction by unified metric representation and CCA-based transfer learning," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015*, New York, New York, USA: ACM Press, 2015, pp. 496–507. doi: 10.1145/2786805.2786813.

[25] Y. Ma, S. Zhu, Y. Chen, and J. Li, "Kernel CCA based transfer learning for software defect prediction," *IEICE Transactions on Information and Systems*, vol. E100D, no. 8, pp. 1903–1906, 2017, doi: 10.1587/transinf.2016EDL8238.