Exploring Machine Learning to Support Software Managers' Insights on Software Developer Productivity

Suwarno[1,*], Yefta Christian[2], Keaton Yoputra[3], Yuki Estrada[4]

[1,2,3,4]*Department of Information Systems, Faculty of Computer Science, Universitas Internasional Batam, Batam 29426, Indonesia*

(Received: February 10, 2025; Revised: April 5, 2025; Accepted: May 8, 2025; Available online: June 2, 2025)

Abstract

Software developer productivity is a complex issue with no single, universally accepted definition or measurement. Emerging technologies like machine learning offer a promising opportunity for more accurate productivity measurement. Semi-structured interviews were conducted to gain qualitative insights into software managers' perception of developer productivity to identify issues and inform the development of applied machine learning solutions. It was discovered that digital distractions significantly hinder developer productivity and conventional methods to monitor developer activity were often inefficient. Therefore, machine learning models were developed to monitor developer activity by classifying screenshots captured during activity, along with the URL and text content scraped from accessed URLs. Train and test data were obtained from a cooperating software house, supplemented with online sources. For screenshot classification, transfer learning using EfficientNetV2B0 outperformed InceptionV3, Resnet50V2, and VGG16, reaching 99.6% accuracy. This was achieved without fine-tuning, which resulted in the fastest training and lowest resource consumption. For content classification, SVC hyperparameter-tuned using grid search outperformed six other classifiers, reaching 88.5% accuracy. The design concept for a web application that utilizes the developed models to help managers measure developer productivity was well-received by the managers interviewed.

Keywords: Software Developer Productivity, Semi-structured Interview, Classification, Transfer Learning, Grid Search

1. Introduction

Software developer productivity has become an increasingly crucial theme in software development. To remain competitive, software companies strive to improve developer performance and meet demands for shorter delivery times [1]. A clear understanding of defining, measuring, and predicting developer productivity could help organizations, managers, and developers develop software more efficiently [2]. Productivity is commonly defined as the size of output delivered over the effort spent to build it [1], [3]. However, this equation is insufficient in software development as productivity is dependent on multiple factors instead of a single metric [3], [4], [5]. This feedback prompts further studies to identify the key factors and metrics impacting developer productivity [6], [7].

A study by [8] found developers assess productivity by the number of completed tasks, while others consider technical elements such as lines of code and number of commits, and non-technical elements such as level of focus and progress made. Another study by [7] classified productivity metrics into code-based metrics such as the amount of code produced, and commit-based metrics such as repository commit activity.

It has been discovered that these metrics are diverse and difficult to quantify or control [1], [9]. For example, when two developers produced equally critical source code, the person who coded more lines could be incorrectly deemed more productive if lines of code were the sole metric measured. Despite those complications, there is a growing interest in establishing better methods to measure developer productivity. Research by [3] garnered positive results from creating a gamified website to measure developer productivity by incorporating game mechanics to motivate users. Another research by [10] combined lines of code written with self-assessed measurements to gauge productivity, a method later adopted by Microsoft for daily developer assessments. Given these challenges, it is crucial to explore emerging technologies such as machine learning (ML) and artificial intelligence (AI) to measure developer productivity. ML and

***Corresponding author: Suwarno (swliang@gmail.com)**
DOI: https://doi.org/10.47738/jads.v6i2.661

AI can analyze complex patterns and adapt to changing environments more effectively than conventional metrics. Previous studies also show great potential in using ML and AI to improve developer productivity.

One study [11] reported that GitHub Copilot users accept an average of 30% of code suggestions from the code completion tool, improving productivity. Another study [12] found that developers using GitHub Copilot completed tasks 55.8% faster. This was likely due to its ability to provide valuable suggestions that guide users toward their goals [13]. Despite that, using ML and AI to measure developer productivity still needs further exploration.

This study aims to discover how developer productivity is defined and measured based on software managers' perceptions and explore ML options to help measure developer productivity. The qualitative insights gained can help identify key factors to consider when measuring developer productivity. The developed ML models can help managers measure developer productivity more accurately and objectively. Additionally, this study aims to fill the gap in existing research on developer productivity measurement and the application of ML in productivity measurement within the software development industry. This research contributes to both academic and professional knowledge.

2. Method

This study employed a multi-method approach, combining qualitative and applied research. Results from qualitative analysis were used to identify issues and inform the development of applied ML solutions.

2.1 Qualitative Research

The qualitative research section of this study used semi-structured interviews to gather data from software managers. This approach incorporated the participants' experiential knowledge and naturally focused on issues most important to them while still providing space to include new themes [14]. Twelve managers from six privately owned software houses in Indonesia, primarily those based in Batam, Medan, and Jakarta, were separately interviewed online from March 2024 to July 2024. Using questions adapted from a previous study [15], this section focused on gathering their perspectives to answer two key research questions: “What is the definition of software developer productivity?” and “How is software developer productivity measured?” Thematic analysis was performed afterward to identify patterns, themes, and concepts from the gathered data, as in [7].

2.2 Applied Research

The applied research section of this study used the Cross Industry Standard Process for Data Mining (CRISP-DM) framework, the de facto industry-independent standard process for data mining [16], to maintain a structured approach. Since this section focused on model creation and evaluation, the process was streamlined to data collection, data preparation, model training and development, model evaluation, and model application. The experiments were conducted in Microsoft Visual Studio Code IDE equipped with the Jupyter Notebook extension, using Python version 3.12.4 as the programming language. The hardware used was a Windows 11 machine equipped with a Ryzen 5 4600H processor and 16GB of DDR4-3200 RAM.

Qualitative research has identified digital distractions as a factor that significantly hinders developer productivity. Conventional monitoring methods, such as requesting updates or reports and direct observations by managers are often inefficient. One solution utilizing ML proposed was developing models that can classify screenshots from computer activity and text content from accessed Uniform Resource Locators (URLs) to monitor developer activity and aid managers in measuring developer productivity.

The data collection phase involved collecting data to build the datasets. From July 2024 to September 2024, screenshots and accessed URLs from the work computers of developers in a cooperating software house were collected using recording programs to serve as primary data sources for the image and text datasets. Additional secondary data sourced from the URL Classification Dataset [17] on Kaggle was combined to improve model accuracy. This dataset was chosen for its size, diversity, pre-existing labels, and public availability, aiding reproducibility and comparability.

The data preparation phase involved preparing the dataset for model training and validation. Careful data selection is essential to achieve accurate results [18]. Data collected was labeled as either work-related or not based on its relevance to a developer's role and responsibilities. The image dataset underwent data cleaning, resizing, data augmentation, and

ISSN 2723-6471

balancing by oversampling. Primarily used libraries in this process include Keras (Preprocessing Layers) and NumPy. Data cleaning removes corrupt images to ensure model accuracy and reliability [19]. Resizing images to a smaller uniform size mitigates memory limitations and high computational costs [20], enabling training on powerful resource-intensive deep learning models [21]. Data augmentation transforms images [22] to enhance model generalization using random zooming, brightness, and contrast adjustments. Rotation, flipping, and perspective transformations were initially explored but deemed unnecessary since screenshots are consistently oriented and appear typically the same. Lastly, oversampling mitigates class imbalance by generating augmented images for the minority class [23].

The combined text dataset was first enriched with text content of up to 1000 words scraped from each collected URL. This approach offered a richer context than relying on URLs alone, allowing for a more accurate and effective classification. The contents then underwent lowercasing, stop words removal, and text normalization using stemming and lemmatization before being concatenated with their corresponding URL for feature extraction. Primarily used libraries in this process include scikit-learn, NLTK, and Pandas. Stop words removal eliminates common and insignificant words that do not contribute to contextual meaning [24] to optimize model focus and conserve memory [25]. Stemming trims prefixes and suffixes while lemmatization utilizes vocabulary and morphological analysis to reduce words to their root form [26]. Feature extraction using the Term Frequency-Inverse Document Frequency (TF-IDF) method [27] quantifies the importance of each word by assigning higher importance to words that appear frequently in a specific document but less frequently in the overall corpus. This simple and effective technique is widely used in text classification [28] for information retrieval and data mining [29].

The modeling phase involved building suitable models. Four deep learning Convolutional Neural Network (CNN) architectures: EfficientNetV2BO [30], InceptionV3 [31], Resnet50V2 [32], and VGG16 [33] were trained and evaluated to find the best-performing image classification model. Different models excel in varying aspects of a task, feature extraction, or data patterns, making exploring a range of models crucial. These four models were chosen due to their distinct architectural designs and proven effectiveness in various contemporary image classification tasks, as demonstrated by multiple research [34], [35], [36], [37].

Using the Keras library with Tensorflow backend, the models were pre-trained on ImageNet data to leverage transfer learning, improving accuracy and training efficiency [38]. Each model started with a (224, 224, 3) input layer. Table 1 shows the additional layers added to each model and their respective purposes. The underlying base architectures are not modified. A dropout layer with a common practice rate of 0.2 was added to provide a balance between regularization strength and preventing overfitting, as in [39]. Dropout layer mitigates overfitting by randomly deactivating neurons during training and fostering the learning of robust features. An 80/20 train-test data split was employed to ensure sufficient training data while providing a statistically significant validation set, which is another common practice for datasets of this size as in [40]. The split was also stratified to ensure that the class proportions were maintained across both sets and mitigate potential bias from class imbalances to ensure reliable evaluation.

Table 1. Overview of Additional Layers Added on Top of Each Pre-trained Model

Layer Type	Configuration	Purpose
Global Average Pooling	GlobalAveragePooling2D()	Condense spatial information
Batch Normalization	BatchNormalization()	Normalize activations
Dense	Dense(128, activation='relu')	Improve feature extraction
Dropout	Dropout(0.2)	Prevent overfitting
Output	Dense(1, activation="sigmoid")	Binary classification prediction

Each model was initially trained for up to 100 epochs with the Adam optimizer, a learning rate of 0.01, and a binary cross-entropy loss function. EarlyStopping and ReduceLROnPlateau callback functions are used, as in [41] did. EarlyStopping stops training after 10 epochs without validation loss reduction to prevent overfitting and avoid unnecessary computation. With restore_best_weights enabled, EarlyStopping also automatically restores the weights from the epoch that yielded the lowest validation loss. On the other hand, ReduceLROnPlateau reduces the learning rate by a factor of 0.2 after 5 epochs without validation loss reduction to allow for finer optimization.

The models were then fine-tuned after their initial training to further improve performance. The top 20 layers of each model except the Batch Normalization layers were unfrozen. This allowed the models to learn unique features of the dataset while preserving the stable representations it had learned. Each model was retrained on the same data and with parameters identical to earlier training for up to another 100 epochs, except with a reduced learning rate of 1e-5 to allow for more gradual weight adjustments. Previous callback functions were also reused.

To determine the optimal text classification model, hyperparameter tuning using grid search was employed. Table 2 shows the classifiers and hyperparameters tested during the grid search.

Table 2. Overview of Hyperparameters Tested During Grid Search

Classifier	Hyperparameter	Description	Tested Values
Logistic Regression	Regularization Strength (C)	Controls the trade-off between smooth decision boundary and training accuracy	[0.1, 1, 10]
	Optimization Solver (solver)	Algorithm used to optimize the model	['liblinear', 'newton-cg', 'lbfgs', 'sag', 'saga']
	Maximum Iterations (max_iter)	Number of iterations for optimization	[100, 200, 300]
Random Forest Classifier	Number of Trees (n_estimators)	Number of trees in the forest	[50, 100, 200]
	Maximum Depth (max_depth)	Maximum depth of the trees	[10, 20, 30]
	Minimum Samples for Split (min_samples_split)	Minimum number of samples required to split an internal node	[2, 5, 10]
	Criterion (criterion)	Function to measure the quality of a split	['gini', 'entropy']
	Maximum Features (max_features)	Number of features to consider when looking for the best split	['sqrt', 'log2']
	Minimum Samples per Leaf (min_samples_leaf)	Minimum number of samples required to be at a leaf node	[1, 2, 4]
Gradient Boosting Classifier	Number of Boosting Stages (n_estimators)	Number of boosting stages to be run	[50, 100, 200]
	Learning Rate (learning_rate)	Step size shrinkage used in update to prevent overfitting	[0.01, 0.1, 1]
	Subsample (subsample)	Fraction of samples used for fitting the individual base learners	[0.8, 1.0]
	Maximum Features (max_features)	Number of features to consider for the best split	['sqrt', 'log2']
	Minimum Samples for Split (min_samples_split)	Minimum number of samples required to split an internal node	[2, 5]
Support Vector Classifier	Regularization Strength (C)	Controls the trade-off between achieving a low training error and a low testing error	[0.1, 1, 10, 100]
	Kernel Type (kernel)	Specifies the kernel type to be used in the algorithm	['linear', 'rbf']
	Gamma (gamma)	Kernel coefficient for ‘rbf’, ‘poly’, and ‘sigmoid’	['scale', 'auto']
	Degree (degree)	Degree of the polynomial kernel function	[2, 3, 4]
K-Neighbors Classifier	Number of Neighbors (n_neighbors)	Number of neighbors to use for classification	[3, 5, 7]
	Weights (weights)	Weight function used in prediction	['uniform', 'distance']
	Distance Metric (metric)	Metric used to compute distance between neighbors	['euclidean', 'manhattan', 'minkowski']

	Algorithm (algorithm)	Algorithm used to compute the nearest neighbors	['auto', 'ball_tree', 'kd_tree', 'brute']
Stochastic Gradient Descent Classifier	Loss Function (loss)	The loss function to be used	['hinge', 'log_loss', 'modified_huber']
	Penalty (penalty)	The penalty (regularization term) to be used	['l1', 'l2', 'elasticnet']
	Learning Rate (alpha)	Constant that controls the step size in the optimization	[0.0001, 0.001, 0.01]
	Maximum Iterations (max_iter)	Maximum number of iterations for the optimization	[1000, 2000, 3000]
	Learning Rate Schedule (learning_rate)	The learning rate schedule to be used	['constant', 'optimal', 'invscaling', 'adaptive']
	Initial Learning Rate (eta0)	The initial learning rate	[0.1, 0.5]
	Shuffle (shuffle)	Whether to shuffle the training data before each epoch	[True, False]
Multinomial Naive Bayes	Smoothing Parameter (alpha)	Additive Laplace smoothing parameter	[0.1, 0.5, 1.0]
	Fit Prior (fit_prior)	Whether to learn class prior probabilities	[True, False]
	Class Prior (class_prior)	Prior probabilities of the classes	[None, [0.5, 0.5]]

Classical machine learning algorithms were chosen to classify URLs and text content as work-related or not due to their sufficient performance in binary classification. While deep learning models are more powerful, their increased computational cost and resource requirements were unnecessary for this task. Furthermore, the dataset size, though substantial, did not necessitate the complex architectures of deep learning, which typically excel in scenarios demanding nuanced feature extraction, such as sentiment analysis or language detection.

A diverse set of classifiers and hyperparameters was evaluated to ensure thorough exploration and identify the most effective model configuration. For numerical hyperparameters, discrete sets of promising values spanning orders of magnitude based on common practices were selected with scales (e.g., linear, logarithmic) chosen according to established conventions for each parameter. For categorical hyperparameters, all available options were selected, given the limited number of choices, as in [42].

A parameter grid was defined, encompassing the classifiers and hyperparameters to be tested. Subsequently, a pipeline comprising a TF-IDF vectorizer and a placeholder classifier was constructed. Following an 80/20 stratified train-test split, an exhaustive 5-fold cross-validation grid search was conducted on the defined parameter grid using GridSearchCV from scikit-learn. The model underwent five training iterations for each classifier-hyperparameter combination, with each fold serving as the validation set in turns. The average performance across the five folds provided a reliable estimate for each combination. Ultimately, the combination yielding the highest average performance score was selected.

The model evaluation phase involved assessing model performances. The models were evaluated against the remaining test data from earlier train-test data splits using metrics derived from the confusion matrix, such as accuracy, precision, recall, and F1-score, as in [43]. Accuracy measures how often the model correctly classifies items as either work-related or not. Precision gauges how many of the items classified as work-related truly were work-related. Recall quantifies the model's ability to find all the actual work-related cases. The F1-score combines precision and recall for work-related classification into a single metric, giving equal weight to both. Lastly, the model application phase involved designing an application that utilizes the developed models using Figma.

3. Results and Discussion

3.1. Qualitative Research

The qualitative research section is divided into descriptive analysis, defining software developer productivity, measuring software developer productivity, and summary.

 ISSN 2723-6471

3.1.1. Descriptive Analysis

Table 3 shows that the software houses interviewed used various programming languages for web, mobile, and industrial applications with varying workforce sizes. Despite that, many had similar development processes, and most offered Software as a Service (SaaS) solutions, primarily in Enterprise Resource Planning (ERP).

Table 3. Characteristics of Software Houses Interviewed

Organization	No. of Employees	Domain	Programming Languages	Development Process
A	55	ERP	Go, PHP, Python	Agile (Scrum)
B	95	ERP	C#, Java, JavaScript, Kotlin,	Agile (Scrum)
C	12	Industrial Automation	G-code, C#, Verilog	Agile (Extreme Programming)
D	285	ERP	Flutter, JavaScript	Agile (Scrum)
E	5	Delivery Platforms	C#, PHP, Python	Agile (Scrum)
F	20	ERP	JavaScript, PHP	Agile (Scrum)

Table 4 shows that most software managers interviewed had five or more years of experience at their current organization, except for p11, who had three years, and p12, who had less than a year. Most managers have an educational background in Computer Science, while some hold degrees in Electrical Engineering (p2, p3, p9).

Table 4. Characteristics of Software Managers Interviewed

ID	Organization	Position	Education	Tenure
p1	A	Software Engineering Team Leader	Bachelor	9
p2	A	Lead Mobile Developer	Bachelor	5
p3	A	Design (UI/UX) Team Leader	Diploma II	6
p4	B	Product Manager	Bachelor	5
p5	B	Developer Analyst	Bachelor	6
p6	B	Program Manager	Bachelor	7
p7	B	Program Manager	Bachelor	7
p8	C	IT Lead	Bachelor	6
p9	D	QandA Lead	Bachelor	5
p10	D	Head of Tech	Bachelor	8
p11	E	Programming Team Leader	Bachelor	3
p12	F	Programming Team Leader	Diploma III	<1

3.1.2. Defining Software Developer Productivity

All managers except p1 defined developer productivity as completing assigned tasks on time without compromising quality. While p2 preferred early submission to avoid overtime, most agreed on timely delivery. No specifics on daily task quantity were mentioned, but p9 did suggest the importance of fair workload distribution.

> *...ensuring all tasks are finished before the deadline.* (p2)
>
> *...ability to produce quality and efficient code within a certain timeframe.* (p3)
>
> *...ability to complete tasks quickly without sacrificing quality.* (p4)
>
> *...ability to produce high-quality code efficiently.* (p5)
>
> *...able to finish work quickly but still maintain quality.* (p6)

Managers also valued adaptability and improvement as productivity. Early task completion, as p2 believed, would provide more time to improve these skills through exploration and learning. Besides meeting deadlines, p3 also expected developers to adapt to changing requirements or technology and contribute to improving the software development process.

 ISSN 2723-6471

> *...when they don't have tasks, they immediately inquire about what [still] needs to be done.* (p2)
>
> *...capability to collaborate within a team, adapt to changing needs or technologies, and contribute to the overall improvement of software development processes.* (p3)

Collaboration skills were another essential component in productive developers. Team coordination is needed to balance workload and meet organizational targets. Developers in Organization E collaborate to re-allocate tasks when deadlines approach to prevent burnout and ensure timely completion.

> *...ability to work effectively within a team.* (p4)
>
> *...able to work well in a team.* (p6)
>
> *...work effectively in a team.* (p7)
>
> *...when our project is near its deadline, we...assess the remaining tasks.... Most of the cases this problem would be come from the time constraint...the solution was to allocate some of the tasks to other members....* (p11)

The last crucial factors were understanding and achieving stakeholder goals. Organization D used story points to help quantify productivity and aid in planning, prioritization, and goal alignment.

> *...[the code] quality, sustainability, and the impact of the solutions produced on [set] business goals.* (p3)
>
> *...knowing the requirements for projects and the constraints such as time, resource[s], and scopes.* (p11)

3.1.3. Measuring Software Developer Productivity

Most managers regarded effective and timely delivered output as the key indicator of a productive developer. Developers must understand resources, time, and scope management to be considered productive. For example, Organization E held daily meetings to track progress and resolve roadblocks to maintain developer productivity.

> *...write code that is efficient, easy to maintain, and can anticipate potential bugs...effective time management such as prioritizing tasks and maintaining focus is crucial. If these aspects are fulfilled, the assigned tasks are more likely to be completed according to the deadline.* (p1)
>
> *Based on their output, code quality, time taken to complete tasks...* (p5)
>
> *...how fast we can deliver requirements for [our] projects.* (p11)

Several managers used technical metrics and software tools to track deliverable output. For example, Organization A tracked real-time activity, Organization B used Git for code contribution tracking, and Organization C identified productivity based on story points.

> *...the company uses monitoring tools that can track the real-time activities of its employees...[to] obtain information such as the total effective working hours of employees, allowing management to assess their efficiency.* (p1)
>
> *...use project management and software development tools...such as using Git to track code contributions...* (p6)

Beyond deliverable output, behavior and team dynamics were valid indicators. Regular evaluations and client feedback were also considered when assessing developer productivity. For example, Organization E conducted monthly peer feedback meetings to gauge developers' impact on team dynamics beyond deliverables.

> *...only in a collaborative work atmosphere, software developers thrive, innovate, and produce exceptional solutions.* (p3)
>
> *Based on...their contributions to the project and team.* (p5)
>
> *By conducting an anonymous internal survey that requires each developer on a team to provide an assessment of their colleagues' performance.* (p8)
>
> *...how active a developer is in providing support to the other members.* (p11)

3.1.4. Section Summary

The interviews revealed various approaches to measuring software developer productivity between organizations. While several metrics were suggested, the most common consensus was that developer productivity is defined as completing tasks before the deadline without sacrificing quality. This highlighted the high standards expected from developers.

 ISSN 2723-6471

For developers to achieve optimal productivity, p11 emphasized understanding project constraints like time, resources, and scope. Although this was crucial, the interviews revealed another recurring theme: the importance for developers to maintain focus and avoid distractions. This was emphasized by p1 and p3, with p9 further highlighting the necessity of identifying distractions that hinder productivity to stay on track.

These distractions are common in environments with frequent digital. interactions [44], [45] Developers, who often rely on online resources like Stack Overflow for troubleshooting and insights [46], [47], are particularly susceptible to these distractions [48]. One research [49] suggests that self-initiated interruptions, such as voluntarily switching tasks within the screen, particularly to engage with non-work activities, can be more disruptive than external interruptions and negatively impact productivity.

To ensure developers stay on track, software houses often monitor developer activity by requesting updates and reports. For example, Organization B reviewed the daily task progress of each developer, including completed, ongoing, and upcoming tasks. However, these processes were often tedious, time-consuming, and prone to subjective bias or human error. Managers also perform observations from time to time, but they cannot be expected to constantly monitor every developer's activity as they have their own tasks to complete. Developers should also have some level of flexibility and autonomy. Hence, there is an opportunity to utilize ML for monitoring developer activity.

One study [50] developed an AI to measure the learning productivity of students by analyzing visual data from annotated screenshots and categorizing them into various categories to block elements recognized as distractions. There was also extensive research on content classification, especially of user-generated tweets, utilizing various ML methods, as reviewed by [51]. A similar approach can be considered to monitor developer activity and thus help measure developer productivity by classifying screenshots and content of accessed URLs.

A follow-up inquiry was conducted with the managers interviewed to get their feedback on the qualitative research results of this study, as in [15]. All managers agreed with the analysis of the insights gathered and the summary provided. There were no dissenting opinions.

3.2. Applied Research

The applied research section is divided into data collection and preparation, model training and evaluation, and model application.

3.2.1. Data Collection and Preparation

A total of 2,210 screenshots and 849 accessed URLs were collected as primary image and text data. The text dataset was supplemented with 3,309 random URLs from the "URL Classification Dataset" as secondary text data. All data collected were labeled as either work-related or not-work-related. An additional column was created to store text content scraped from its associated URL. Figure 1 and figure 2 show a random sample of collected screenshots and collected URLs with scraped text content, respectively.

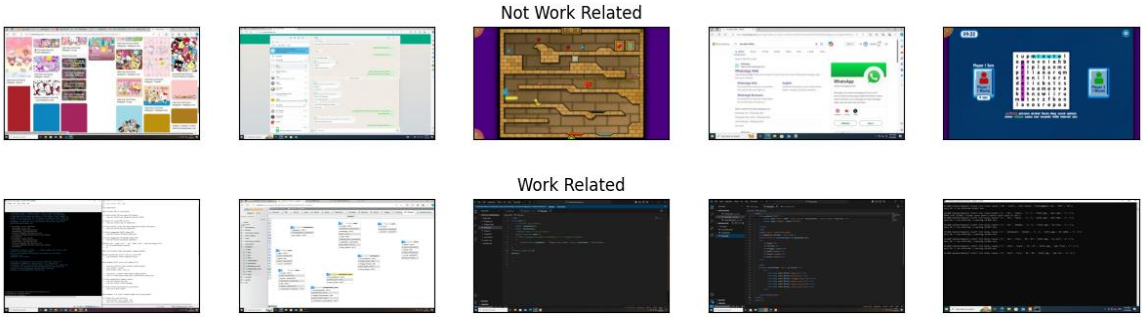

Figure 1. Sample of Collected Screenshots

ISSN 2723-6471

category		url	category	content
Not	1708	http://canberrafloorball.tripod.com/	Not	Canberra Floorball Club Thursday Night Floorba...
	2186	http://www.endurancekarting.com/	Not	Endurance Karting Take Fun Seriously Click on ...
	3313	https://www.microsoft.com/en-us/edge/update/12...	Not	Microsoft Teams Copilot Windows Surface Xbox D...
	2490	http://www.geocities.com/nickie101/songlist1.html	Not	News Todays news US Politics World Tech Review...
	2761	http://www.kerrmuseumproductions.com/	Not	Home ServicesExhibitionsArt Support Services E...
Work	603	http://www.c2s.com/	Work	CONTACT US EMAIL email protected PHONE FAX HOM...
	842	http://www.elock.com	Work	Technical Support Inquiry Email infoelockcom C...
	1016	http://xmp.sourceforge.net/	Work	Extended Module Player The Extended Module Pla...
	557	http://www.tccomm.com/	Work	Solutions Solutions Industries Utilities Publi...
	1390	http://support.microsoft.com/default.aspx?scid...	Work	Home Microsoft Office Products Microsoft Outlo...

Figure 2. Sample of Collected URLs with Scraped Text Content

After data cleaning, 2,190 image data and 3,563 text data remained. The images were resized to a shape of (224x224x3) and augmented. To balance the dataset, the not-work-related class was oversampled with augmented images to approximately match the quantity of the work-related class. The contents from the text dataset underwent lowercasing, stop words removal, and stemming and lemmatization before being concatenated with their corresponding URL for TF-IDF feature extraction. Figure 3 and figure 4 show a random sample of preprocessed images and texts, respectively. Figure 5 and figure 6 show the screenshots and contents data distribution after preprocessing, respectively.

Figure 3. Sample of Preprocessed Image Data

category		category	content
0	3024	0	http://www.animecritic.com/battlearena/anr-battlearena.html anim critic anim review az...
	2772	0	http://www.ruggerland.co.nz/ get involv play coach refere manag teenag rugbi club mori...
	2485	0	http://qualitygermanautoparts.com/ home diesel turbo diesel tdi diesel engin best cc t...
	2437	0	http://www.walkerrowe.com/ southern pacif review histori virginia wine book walker ell...
	2993	0	http://www.eliason-snowmobile.com/ beginningfor invent year eliason eliasonfour wheel ...
1	1119	1	http://www.publicimagelimited.com/ site design intranet extranet web applic put summer...
	730	1	http://www.securitystronghold.com/ english english russian german spanish french engli...
	1154	1	http://jeplite.sourceforge.net/ jeplit jep enlit jeplit offer le featur jep doubl arit...
	658	1	http://mpeglib.sourceforge.net/ mpeglib librari includ three command line player mpwav...
	1291	1	http://www.securitypronews.com home wiki mail list mirror list irc forum bug donat wel...

Figure 4. Sample of Preprocessed Text Data

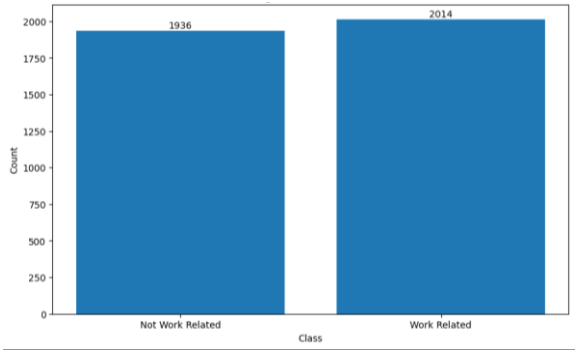

Figure 5. Distribution of Preprocessed Image Data

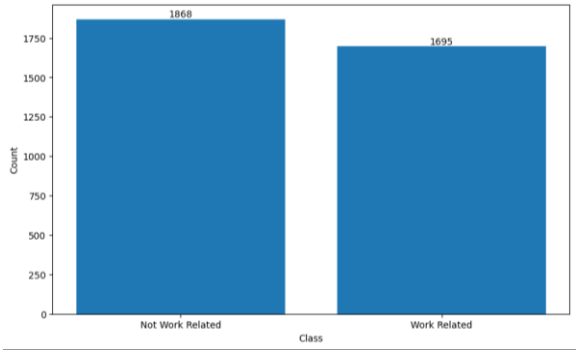

Figure 6. Distribution of Preprocessed Text Data

3.2.2. Model Training and Evaluation

Table 5 compares the performance metrics of various pre-trained models after initial training on the dataset for the image classification task. EfficientNetV2B0 outperformed the others, achieving perfect scores for precision, recall, and F1-score, with an accuracy of 0.9962. VGG16 also performed well, reaching 0.97 for precision, recall, and F1-score, with an accuracy of 0.9671 but it took significant time to train (almost 35 minutes longer). InceptionV3 and ResNet50V2 also performed well, achieving precision, recall, and F1 scores between 0.91 and 0.93. Their accuracies, however, were slightly lower than EfficientNetV2B0 and VGG16 at 0.9139 and 0.9291 respectively.

Table 5. Overview of Pre-trained Model Performances After Training

Pre-trained model	Final epoch	Train time (s)	Precision	Recall	F1-score	Accuracy
EfficientNetV2B0	31	1991.0	1.00	1.00	1.00	0.9962
InceptionV3	10	1369.5	0.91	0.91	0.91	0.9139
ResNet50V2	4	1554.9	0.93	0.93	0.93	0.9291
VGG16	8	4073.6	0.97	0.97	0.97	0.9671

Table 6 compares the performance metrics of the same models after fine-tuning. EfficientNetV2B0 seemed to have already converged, as indicated by EarlyStopping which reverted its weights to the third epoch to prevent overfitting. The rest of the models showed improvement in all metrics, with VGG16 achieving the most significant improvement, reaching 0.99 for precision, recall, and F1-score, with an accuracy of 0.9911. However, VGG16 took significantly longer to fine-tune (more than five hours). InceptionV3 and ResNet50V2 also performed well, with precision, recall, and F1-scores ranging between 0.93 and 0.96 and accuracy of 0.9266 and 0.9557 respectively.

Table 6. Overview of Pre-trained Model Performances After Fine-Tuning

Pre-trained model	Final epoch	Train time (s)	Precision	Recall	F1-score	Accuracy
EfficientNetV2B0	3	635.8	0.99	0.99	0.99	0.9949
InceptionV3	33	2943.8	0.93	0.93	0.93	0.9266
ResNet50V2	18	3425.3	0.96	0.96	0.96	0.9557
VGG16	14	19359.0	0.99	0.99	0.99	0.9911

Figure 7 to figure 14 show the confusion matrices and learning curves of the evaluated models after training and fine-tuning. It is important to note that the final model weights were not taken from the absolute last epoch on the graphs, since EarlyStopping reverts the models' weights to 10 epochs prior, where validation loss was minimal.

During training, all the models demonstrated an initial period of rapid improvement in accuracy and reduction in loss for both training and validation sets over the first few epochs. Subsequently, the learning curves gradually leveled off as the models converged towards optimal performance. After fine-tuning, all models, except for EfficientNetV2B0 (already optimized), demonstrated improved accuracy, reduced loss, and fewer misclassifications.

Based on the confusion matrices in figure 7 and learning curves in figure 8, EfficientNetV2B0 demonstrated excellent performance after initial training. The model achieved near-perfect accuracy and minimal misclassifications, indicating optimal convergence. Fine-tuning was conducted for consistency, but it did not significantly improve performance and instead led to a slight increase in misclassifications. The learning curves during fine-tuning also displayed patterns of fluctuations that indicate overfitting. With EarlyStopping triggered after only 13 epochs, further fine-tuning was deemed not beneficial. Overall, the initial training phase appeared to have already optimized the model.

 ISSN 2723-6471

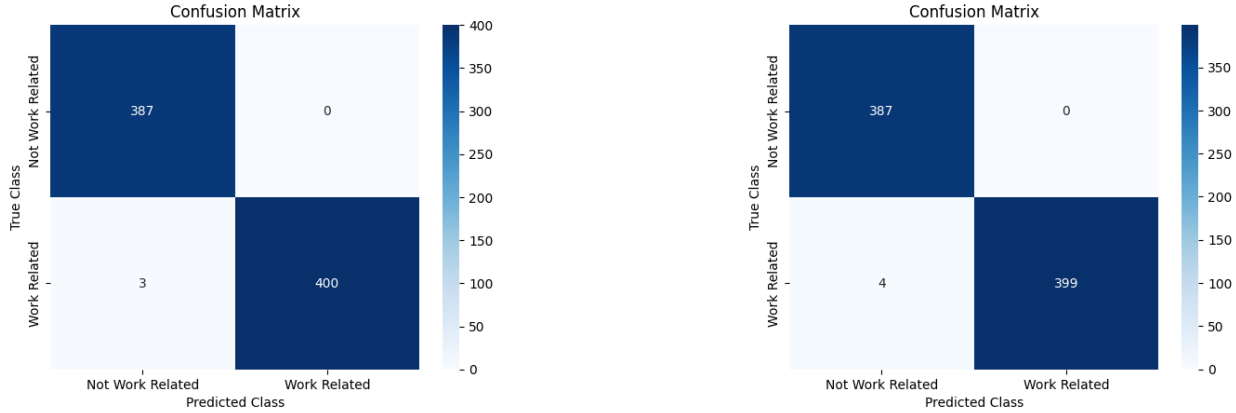

Figure 7. EfficientNetV2B0 Confusion Matrices after (left) Training and (right) Fine-Tuning

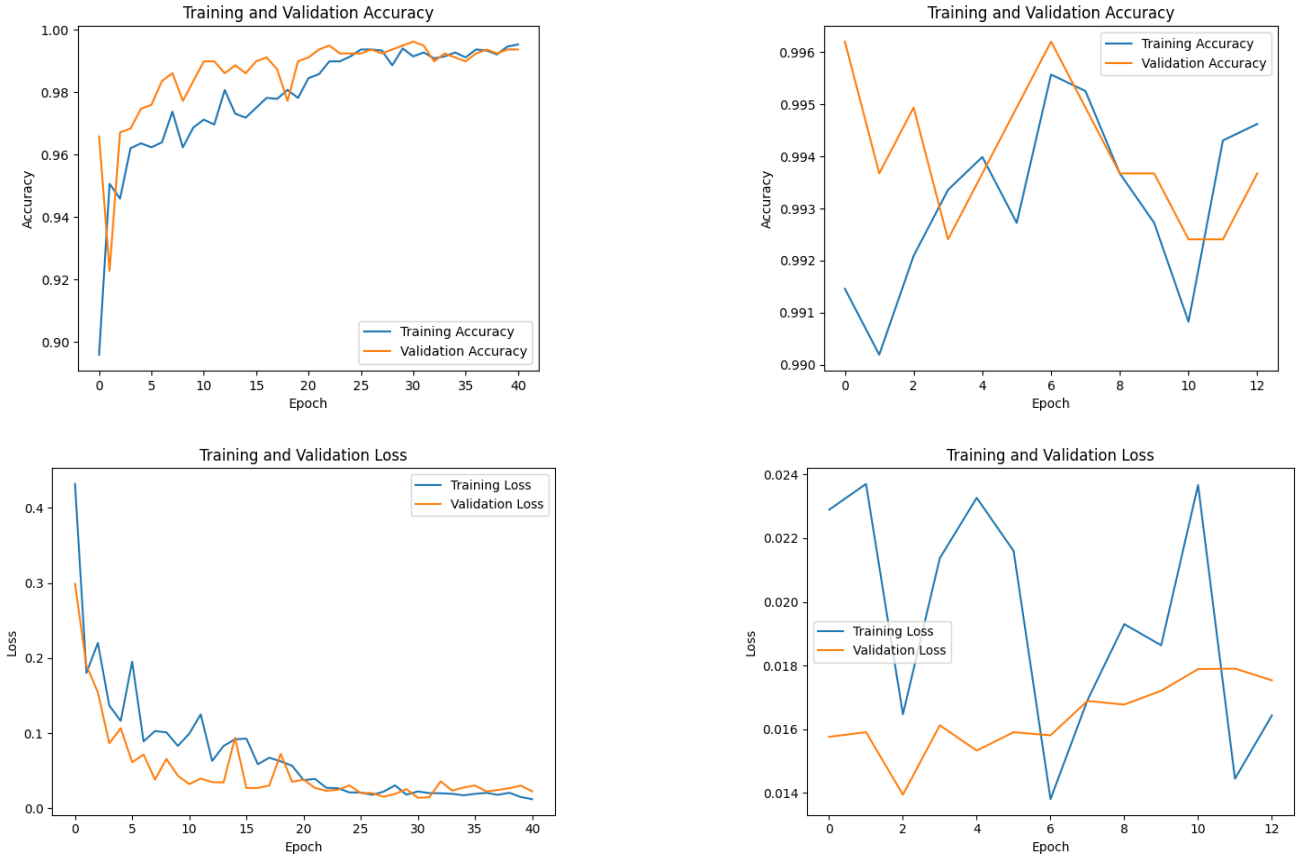

Figure 8. EfficientNetV2B0 Learning Curves after (left) Training and (right) Fine-Tuning

Based on the confusion matrices in figure 9 and learning curves in figure 10, InceptionV3 demonstrated good performance after initial training and improved further after fine-tuning. The initial model achieved high accuracy but still misclassified some images. Fine-tuning significantly reduced misclassifications, especially on work-related images, which led to a more robust and accurate model. The learning curves during fine-tuning also showed a stable and consistent improvement in accuracy and reduction in loss, which suggested that there were benefits from additional training. Overall, both the initial training and fine-tuning phases contributed to the strong performance of the model.

 ISSN 2723-6471

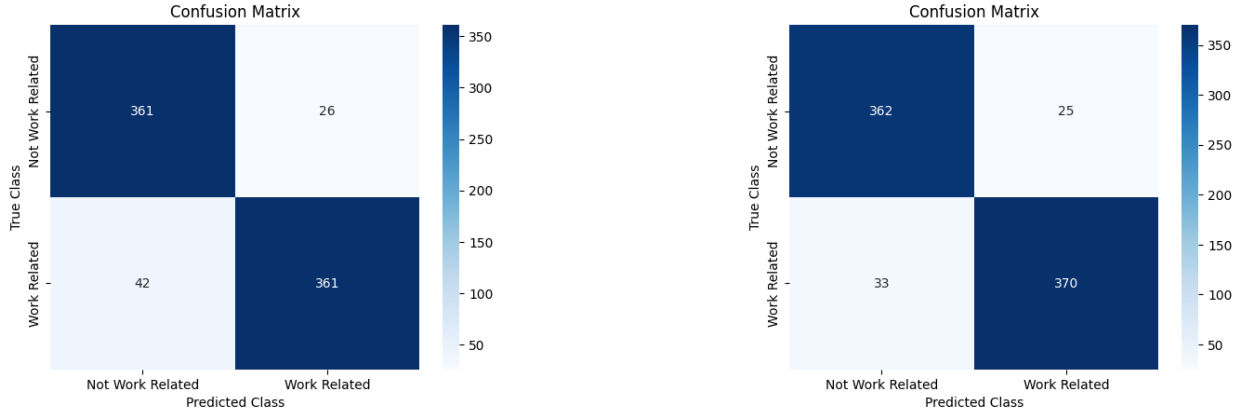

Figure 9. InceptionV3 Confusion Matrices after (left) Training and (right) Fine-Tuning

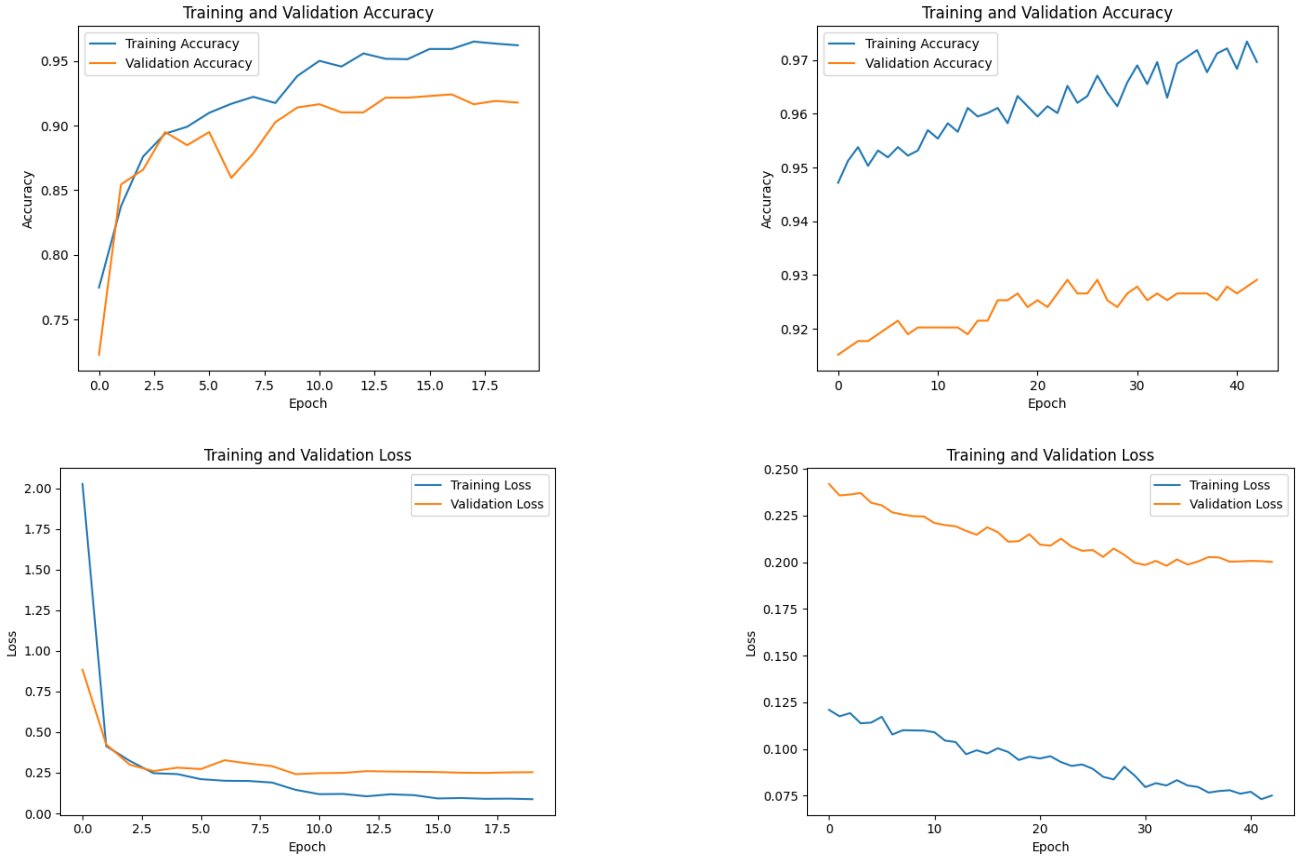

Figure 10. InceptionV3 Learning Curves after (left) Training and (right) Fine-Tuning

Based on the confusion matrices in figure 11 and learning curves in figure 12, ResNet50V2 demonstrated good performance after initial training and improved further after fine-tuning. Like the previous InceptionV3, the initial model achieved high accuracy but still misclassified some images. Fine-tuning in this case also significantly reduced misclassifications on work-related images, which led to a more robust and accurate model. The learning curves during fine-tuning also showed a consistent improvement in accuracy and reduction in loss, which suggested that there were benefits from additional training. Overall, both the initial training and fine-tuning phases once again contributed to the strong performance of the model.

ISSN 2723-6471

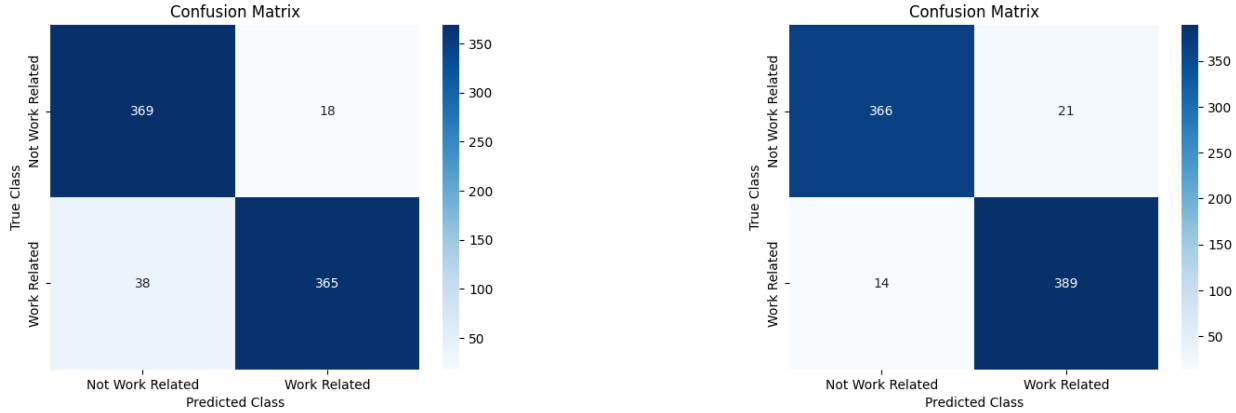

Figure 11. ResNet50V2 Confusion Matrices after (left) Training and (right) Fine-Tuning

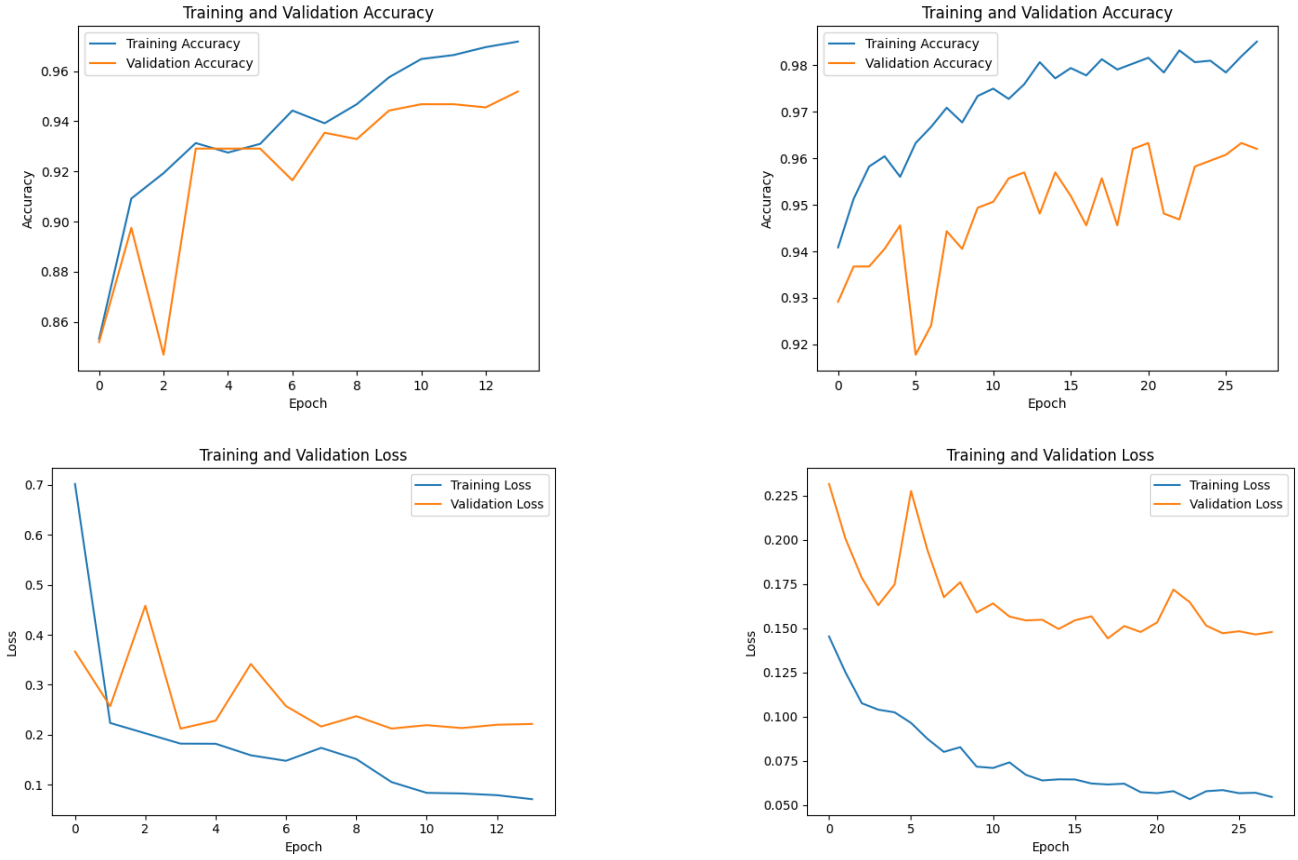

Figure 12. ResNet50V2 Learning Curves after (left) Training and (right) Fine-Tuning

Based on the confusion matrices in figure 13 and learning curves in figure 14, VGG16 demonstrated great performance after initial training and improved further after fine-tuning. The initial model achieved high accuracy but still misclassified some images. Like InceptionV3 and ResNet50V2, fine-tuning significantly reduced misclassifications, especially on work-related images, which allowed it to reach an accuracy close to that of EfficientNetV2B0. The learning curves during fine-tuning also showed improvement in accuracy and reduction in loss, before gradually leveling off, which suggested that up to a certain point, there were benefits from additional training. Overall, both the initial training and fine-tuning phases contributed to the model's strong performance.

 ISSN 2723-6471

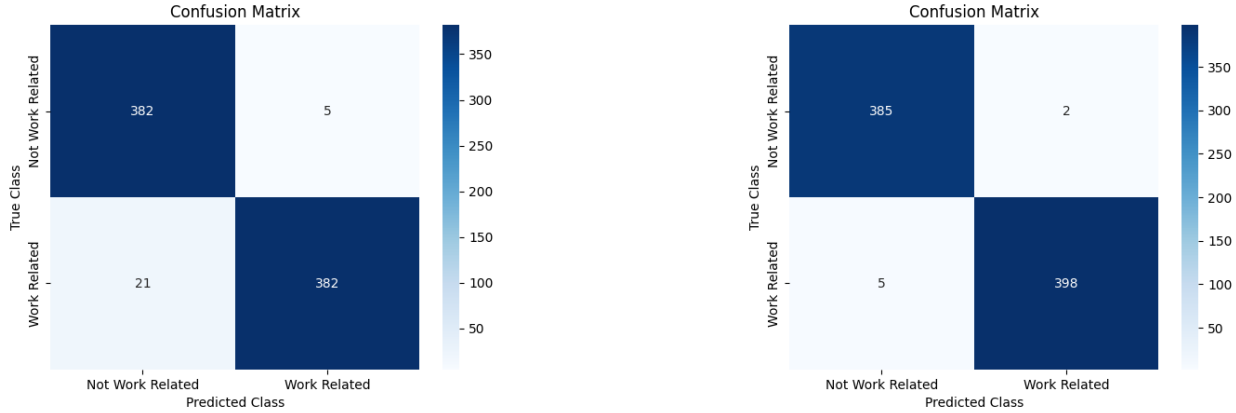

Figure 13. VGG16 Confusion Matrices after (left) Training and (right) Fine-Tuning

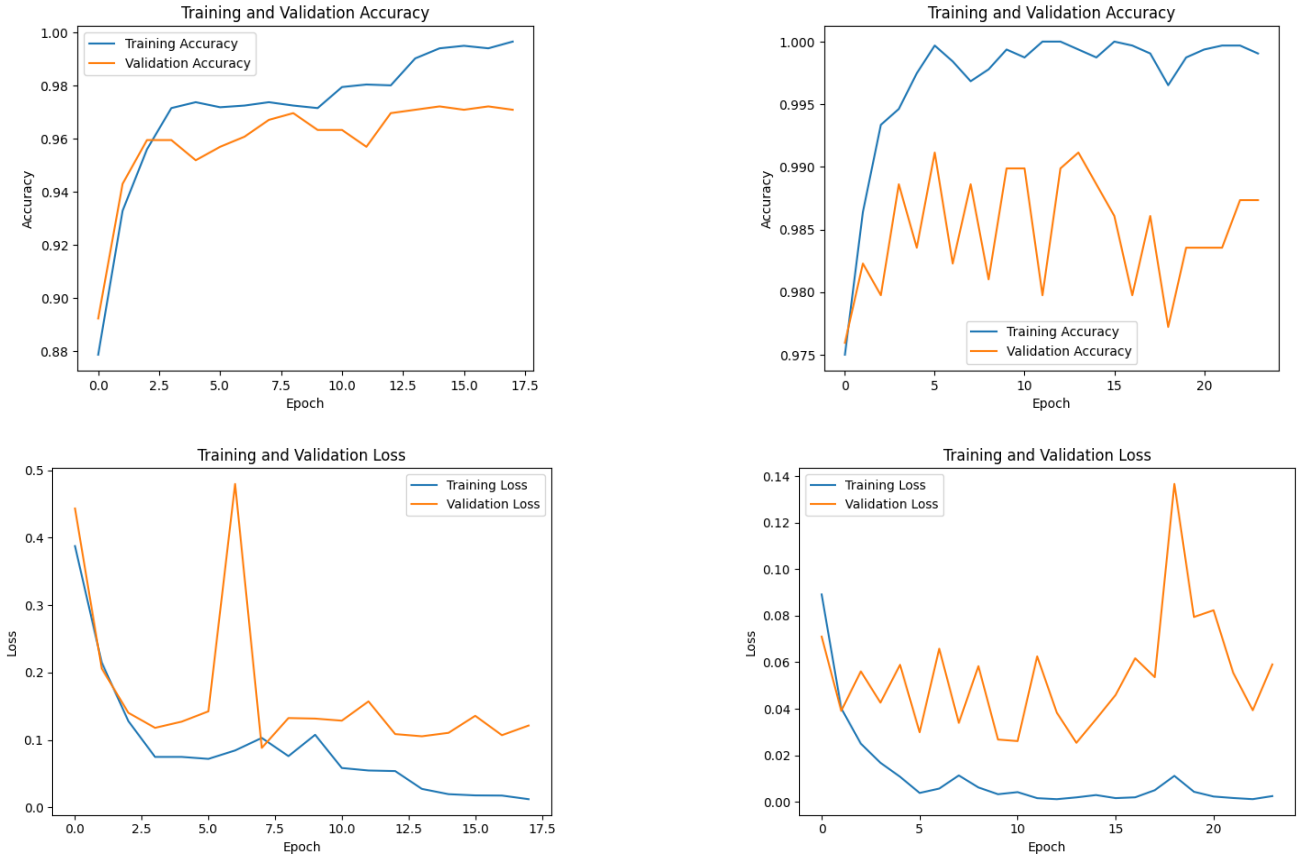

Figure 14. VGG16 Learning Curves after (left) Training and (right) Fine-Tuning

Overall, EfficientNetV2B0 was the most effective in classifying screenshots among the evaluated models. In terms of training speed, this model trained the fastest and converged in the fewest epochs, consequently requiring the least computational resources. VGG16 achieved a similar performance with further fine-tuning that required much higher computational costs. While InceptionV3 and ResNet50V2 performed well, they couldn't match EfficientNetV2B0's superior training speed and resource efficiency.

For text (URL and content) classification, GridSearchCV selected a Support Vector Classifier (SVC) with hyperparameters C=1, degree=2, gamma='scale', and kernel='rbf' as the best-performing model. The regularization parameter ‘C=1’ balances between maximizing margins and minimizing misclassifications, suggesting a preference for a wider margin even at the cost of some errors. The ‘rbf’ kernel has non-linear decision boundaries making it suitable for complex datasets. Gamma set to ‘scale’ automatically calculates based on the data, controlling the influence of individual data points. While ‘degree=2’ is specified, it is irrelevant since the chosen kernel is not polynomial. This

 ISSN 2723-6471

model achieved a best score of 0.9088. Table 7 shows the model classification report, where it was evaluated to have an accuracy of 0.8850, weighted and macro average precision of 0.89, recall of 0.88, and F1-score of 0.88.

Table 7. SVC Model Classification Report

Class	Precision	Recall	F1-score	Support
Not Work Related	0.87	0.92	0.89	374
Work Related	0.91	0.84	0.87	339
Accuracy			0.88	713
Macro Average	0.89	0.88	0.88	713
Weighted Average	0.89	0.88	0.88	713

Figure 15 and figure 16 show the confusion matrix and learning curve of the SVC model, respectively. The confusion matrix showed that 378 out of 405 not-work-related instances and 370 out of 423 work-related instances were correctly classified during evaluation. Despite the good performance, the learning curve hinted the model had slight overfitting. However, the rising test score and plateauing training score trend towards the end of the graph suggest that with more training data, generalization could improve, and the scores would have eventually converged.

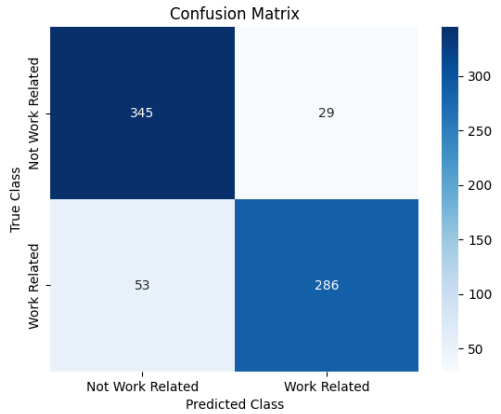

Figure 15. SVC Model Confusion Matrix

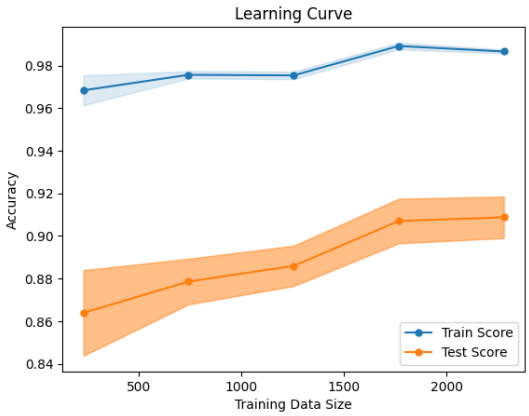

Figure 16. SVC Model Learning Curve

Overall, this research developed two effective models for classifying screenshots and URL content. Image classification would rely on EfficientNetV2B0, which achieved perfect precision, recall, and F1-score scores. Text classification would rely on a hyperparameter-tuned SVC which reached a macro average of 0.89 on the same metrics mentioned earlier. While the text model performed well, there is room for improvement by increasing its training data.

Techniques like EarlyStopping, ReduceLROnPlateau, and GridSearchCV optimized the training process. EarlyStopping prevented overfitting in the image model while ReduceLROnPlateau ensured efficient convergence. GridSearchCV, on the other hand, efficiently found the best text classifier model and its hyperparameters, saving time and effort. These techniques contributed to the development of robust and effective models. Figure 17 and figure 18 show predictions made by the model on a sample of unseen data where it accurately classified all.

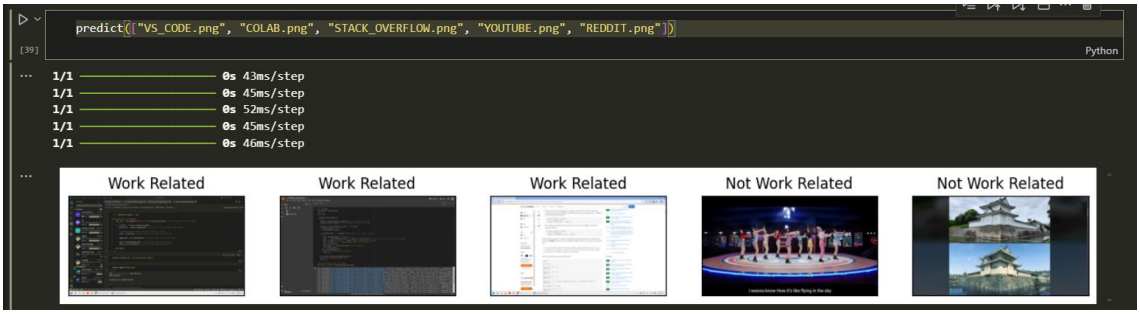

Figure 17. Predictions on a Sample of Unseen Screenshots

 ISSN 2723-6471

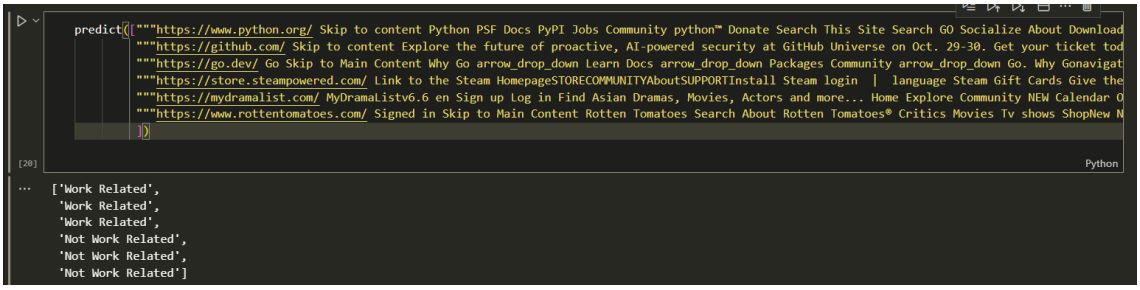

Figure 18. Predictions on a Sample of Unseen Text Content

3.2.3. Model Application

By using the developed models to periodically classify screenshots and content of accessed URLs into work-related or not, the effective working time of a software developer can be estimated. Integrating them into a web application can provide software managers with data to help them more accurately and objectively measure developer productivity, by considering both effective working time and the total number of task story points completed.

Figure 19 to figure 20 show the design concepts for the employee's dashboard and task detail page of the proposed web application, visualized using Figma. Developers initially access the system by logging into their dashboards (figure 19). These dashboards display ongoing and completed tasks, along with the total work duration. To begin activity tracking, developers click on the start button, enabling the system to capture periodic screenshots and URLs accessed. Consequently, the developed model processes these records, classifying them as either work-related or not. For scheduled breaks, developers can engage the pause button. Break intervals are pre-configured to 15 minutes, after which a pop-up notification prompts the developer to resume work. When a developer self-assigns an eligible task, it is designated as an ongoing task. Upon task completion, developers can update the task status to complete (figure 20), thereby updating the activity log.

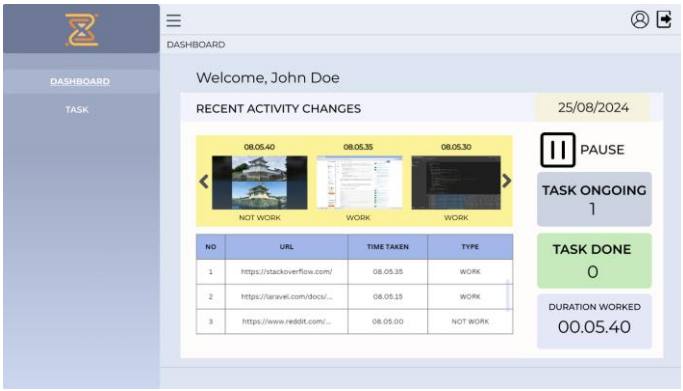

Figure 19. Design Concept for Employee's Dashboard Page of the Proposed Web Application

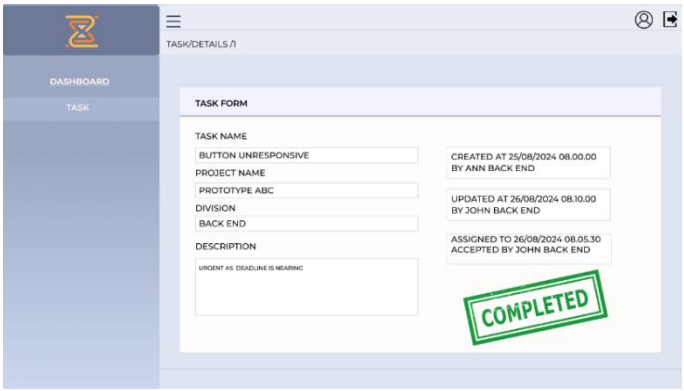

Figure 20. Design Concept for Employee's Task Completion Page of the Proposed Web Application

Figure 21 to figure 22 show the design concepts for the manager's dashboard and employee productivity data page of the proposed web application, visualized using Figma. Managers have a dedicated dashboard displaying overall task progress, work durations tracked, and the total number of currently online employees (figure 21). The dashboard also summarizes the number of ongoing tasks, completed tasks, effective work duration, and total duration for any employee productivity results queried. Managers can also delve into individual developer productivity data, filtering by date to review anonymized classifications of screenshots and URLs (as part of ensuring privacy), provide device names and timestamps of each recorded entry (figure 22). Developer productivity can then be estimated by calculating the proportion of work-related entries out of all recorded entries. Managers can also generate productivity reports, given a date range and an employee name. To uphold accountability and address potential lapses in work resumption, managers are notified if a developer remains offline following the automated break interval reminder. Managers can then send a secondary reminder notification.

ISSN 2723-6471

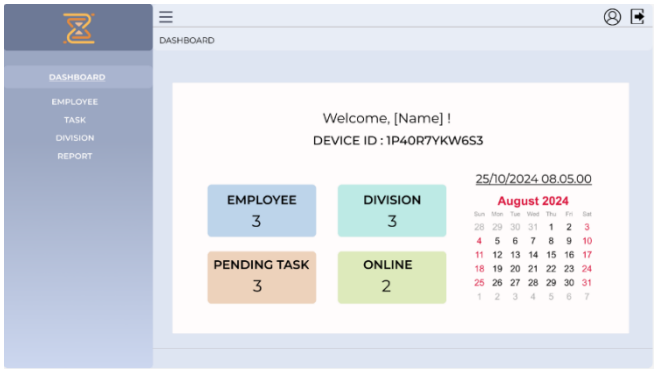

Figure 21. Design Concept for Manager's Dashboard Page of the Proposed Web Application

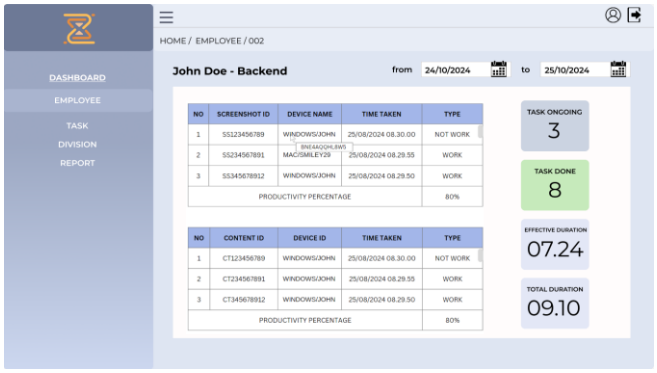

Figure 22. Design Concept for Manager's Employee Productivity Data Page of the Proposed Web Application

This proposed web application can benefit both developers and managers. Developers can use the insights provided for goal setting, progress tracking, and time management. Meanwhile, managers can leverage data and trends collected to improve project management, foster team collaboration, and make informed decisions.

A follow-up inquiry was conducted with the software managers interviewed from the qualitative research to get their feedback on the applied research results of this study. All managers agreed that the proposed web application would be useful in measuring developer productivity. Some managers made additional suggestions: p8 proposed the ability to filter individual employee data, while p3, p9, and p10 proposed integrating a collaborative space for discussing upcoming and ongoing tasks.

4. Conclusion

This study investigated software developer productivity using a multi-method approach with two goals. First, it examined how software managers define and measure developer productivity. Second, it explored potential machine learning techniques to help measure developer productivity.

Through qualitative research, this study found that most managers interviewed primarily defined developer productivity as completing assigned tasks on time without compromising quality, with considerations into adaptability, improvement, collaboration skills, and understanding stakeholder goals. This study also found that most managers often measured developer productivity using indicators like deliverable output efficiency that was tracked using technical metrics and software tools, while also taking into consideration behavior and team dynamics. Another recurring theme found was that distractions significantly impact developer productivity, and that conventional methods were inefficient.

Through applied research, a pair of ML models were developed to accurately classify screenshots and contents of accessed URLs as either work-related or not. After comparing four pre-trained models with transfer learning, EfficientNetV2B0 was the best for classifying screenshots, reaching a near-perfect 0.9962 accuracy without requiring fine-tuning. For URL content, a hyperparameter-tuned SVC outperformed six other classifiers, reaching a substantial 0.8850 accuracy. These models can be used to monitor developer activity, and when integrated with other data into a web application as proposed earlier, can provide managers with a more accurate and objective measurement of developer productivity.

Overall, this study offers a comprehensive understanding of developer productivity by examining qualitative perspectives and exploring applied ML solutions. Future research should leverage technology and data analytics to explore hybrid models and diverse datasets for enhanced metric development. Quantification of critical yet less tangible factors such as collaboration skills, team dynamics, workplace culture, and developer well-being should also be investigated alongside traditional output metrics. Furthermore, research should evaluate advanced techniques for text classification like deep learning and transformer models or optimization of model performance through rigorous multi-step hyperparameter tuning for improved classification. Finally, ethical considerations regarding privacy and bias in

data analysis, and the long-term impact of monitoring tools on developer well-being, must be critically examined to ensure responsible advancements.

The findings of this study reveal the complex relationship between subjective and objective factors affecting developer productivity. While machine learning offers valuable tools for measuring productivity, it is important to recognize the limitations imposed by data-driven approaches. Organizations can create a work environment where developers can thrive and contribute their best to software development projects by fostering a culture of open communication, team collaboration, and supporting employee well-being.

5. Declarations

5.1 Author Contributions

Conceptualization: S.S., Y.C., K.Y., and Y.E.; Methodology: S.S., Y.C., K.Y., and Y.E.; Software: K.Y.; Validation: S.S., and Y.C.; Formal Analysis: S.S., Y.C., K.Y., and Y.E.; Investigation: Y.E.; Resources: S.S.; Data Curation: K.Y., and Y.E.; Writing Original Draft Preparation: S.S., Y.C., K.Y., and Y.E.; Writing Review and Editing: S.S., Y.C., K.Y., and Y.E.; Visualization: Y.E.; All authors have read and agreed to the published version of the manuscript.

5.2 Data Availability Statement

The datasets used and analyzed during the current study are available from the corresponding author upon reasonable request. Due to privacy and proprietary constraints, access to certain data may be restricted. The authors are committed to sharing data per institutional guidelines and policies.

5.3 Funding

The authors thank Universitas Internasional Batam for funding and supporting this research.

5.4 Institutional Review Board Statement

Not applicable.

5.5 Informed Consent Statement

Not applicable.

5.6 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.

Acknowledgment

The authors thank the software houses and software managers who contributed to this research by providing data and participating in interviews. Additionally, the authors thank Mr. Tony Wibowo, S.Kom., M.M.S.I. for his invaluable suggestions and guidance.

References

[1] W. A. Chapetta and G. H. Travassos, “Towards an evidence-based theoretical framework on factors influencing the software development productivity,” *Empir Softw Eng*, vol. 25, no. 5, pp. 3501–3543, Sep. 2020, doi: 10.1007/s10664-020-09844-5.

[2] N. Forsgren, M.-A. Storey, C. Maddila, T. Zimmermann, B. Houck, and J. Butler, “The SPACE of Developer Productivity: There’s more to it than you think.,” *Queue*, vol. 19, no. 1, pp. 20–48, Feb. 2021, doi: 10.1145/3454122.3454124.

[3] J. S. Mota, H. A. Tives, and E. D. Canedo, “Tool for Measuring Productivity in Software Development Teams,” *Information*, vol. 12, no. 10, pp. 396–396, Sep. 2021, doi: 10.3390/info12100396.

[4] J. R. Burnett and T. C. Lisk, “The Future of Employee Engagement: Real-Time Monitoring and Digital Tools for Engaging a Workforce,” *International Studies of Management and Organization*, vol. 49, no. 1, pp. 108–119, Jan. 2019, doi: 10.1080/00208825.2019.1565097.

[5] S. Delaney and D. Schmidt, “A Productivity Framework for Software Development Literature Review,” in *Proceedings of the 2nd International Conference on Software Engineering and Information Management*, New York, NY, USA: ACM, Jan. 2019, vol. 2019, no. 1, pp. 69–74. doi: 10.1145/3305160.3305161.

[6] E. D. Canedo and G. A. Santos, "Factors Affecting Software Development Productivity: An empirical study," in *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, New York, NY, USA: ACM, Sep. 2019, vol. 2019, no. 9, pp. 307–316. doi: 10.1145/3350768.3352491.

[7] E. Oliveira, E. Fernandes, I. Steinmacher, M. Cristo, T. Conte, and A. Garcia, "Code and commit metrics of developer productivity: a study on team leaders perceptions," *Empir Softw Eng*, vol. 25, no. 4, pp. 2519–2549, Jul. 2020, doi: 10.1007/s10664-020-09820-z.

[8] A. N. Meyer, G. C. Murphy, T. Fritz, and T. Zimmermann, "Developers' Diverging Perceptions of Productivity," in *Rethinking Productivity in Software Engineering*, 1st ed., C. Sadowski and T. Zimmermann, Eds., Berkeley, CA: Apress, 2019, vol. 2019, no. 5, pp. 137–146. doi: 10.1007/978-1-4842-4221-6_12.

[9] L. Lavazza, S. Morasca, and D. Tosi, "An empirical study on the factors affecting software development productivity," *E-Informatica Software Engineering Journal*, vol. 12, no. 1, pp. 27–49, 2018, doi: 10.5277/e-Inf180102.

[10] M. Beller, V. Orgovan, S. Buja, and T. Zimmermann, "Mind the Gap: On the Relationship Between Automatically Measured and Self-Reported Productivity," *IEEE Softw*, vol. 38, no. 5, pp. 24–31, Sep. 2021, doi: 10.1109/MS.2020.3048200

[11] T. Dohmke, M. Iansiti, and G. Richards, "Sea Change in Software Development: Economic and Productivity Analysis of the AI-Powered Developer Lifecycle," *Keystone*, vol. 2023, no. 6, pp. 21–29, Jun. 2023, doi: 10.1145/3520312.3534864.

[12] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer, "The Impact of AI on Developer Productivity: Evidence from GitHub Copilot," *Microsoft Research*, vol. 2023, no. 2, pp. 1–19, Feb. 2023, doi: 10.48550/arxiv.2302.06590.

[13] A. Ziegler, E. Kalliamvakou, X. Li, A. Rice, D. Rifkin, S. Simister, G. Sittampalam, and E. Aftandilian, "Productivity assessment of neural code completion," in *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, New York, NY, USA: ACM, Jun. 2022, vol. 2022, no. 6, pp. 21–29. doi: 10.1145/3520312.3534864

[14] J. O'Keeffe, W. Buytaert, A. Mijic, N. Brozović, and R. Sinha, "The use of semi-structured interviews for the characterisation of farmer irrigation practices," *Hydrol Earth Syst Sci*, vol. 20, no. 5, pp. 1911–1924, May 2016, doi: 10.5194/hess-20-1911-2016

[15] E. Oliveira, T. Conte, M. Cristo, and E. Mendes, "Software Project Managers' Perceptions of Productivity Factors," in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, New York, NY, USA: ACM, Sep. 2016, vol. 2016, no. 15, pp. 1–6. doi: 10.1145/2961111.2962626.

[16] C. Schröer, F. Kruse, and J. M. Gómez, "A Systematic Literature Review on Applying CRISP-DM Process Model," *Procedia Comput Sci*, vol. 181, no. 2, pp. 526–534, Feb. 2021, doi: 10.1016/j.procs.2021.01.199.

[17] A. Shawon, Oct. 2019, "URL Classification Dataset [DMOZ]," Kaggle. [Online]. Available: https://www.kaggle.com/datasets/shawon10/url-classification-dataset-dmoz

[18] Y. Christian, Y.-H. Choo, N. Fazilla, and A. Yusof, "Systematic Literature Review on The Use of Machine Learning in Online Learning in the Context of Skill Achievement," *J Theor Appl Inf Technol*, vol. 102, no. 6, pp. 2466–2479, Mar. 2024.

[19] P. Li, X. Rao, J. Blase, Y. Zhang, X. Chu, and C. Zhang, "CleanML: A Study for Evaluating the Impact of Data Cleaning on ML Classification Tasks," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, Chania, Greece: IEEE, Apr. 2021, vol. 2021, no. 4, pp. 13–24. doi: 10.1109/ICDE51399.2021.00009.

[20] H. Talebi and P. Milanfar, "Learning to Resize Images for Computer Vision Tasks," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, Montreal, QC, Canada: IEEE, Oct. 2021, vol. 2021, no. 10, pp. 487–496. doi: 10.1109/ICCV48922.2021.00055

[21] H. Herman, Y. Jaya Kumar, S. Yong Wee, and V. Kumar Perhakaran, "A Systematic Review on Deep Learning Model in Computer-aided Diagnosis for Anterior Cruciate Ligament Injury," *Curr Med Imaging*, vol. 20, no. e15734056295157, pp. 1–10, May 2024, doi: 10.2174/0115734056295157240418043624

[22] E. J. Snider, S. I. Hernandez-Torres, and R. Hennessey, "Using Ultrasound Image Augmentation and Ensemble Predictions to Prevent Machine-Learning Model Overfitting," *Diagnostics*, vol. 13, no. 3, pp. 417–417, Jan. 2023, doi: 10.3390/diagnostics13030417.

[23] G. Kovács, "An empirical comparison and evaluation of minority oversampling techniques on a large number of imbalanced datasets," *Appl Soft Comput*, vol. 83, no. 105662, pp. 1–16, Oct. 2019, doi: 10.1016/j.asoc.2019.105662

[24] R. Khan, Y. Qian, and S. Naeem, "Extractive based Text Summarization Using KMeans and TF-IDF," *International Journal of Information Engineering and Electronic Business*, vol. 11, no. 3, pp. 33–44, May 2019, doi: 10.5815/ijieeb.2019.03.05.

[25] Sanchana.R, Josephine Ruth Fenitha, Shanmughapriya.M, Bhavani Sree. Sk, and Nithyadevi.S, "Analysis of Twitter Data using Machine Learning Algorithms," *EPRA International Journal of Research and Development (IJRD)*, vol. 8, no. 3, pp. 48–56, Mar. 2023, doi: 10.36713/epra12585.

[26] V. A. Kozhevnikov and E. S. Pankratova, "Research of Text Pre-processing Methods for Preparing Data in Russian for Machine Learning," *Theoretical and Applied Science*, vol. 84, no. 04, pp. 313–320, Apr. 2020, doi: 10.15863/TAS.2020.04.84.55

[27] H. P. Luhn, "The Automatic Creation of Literature Abstracts," *IBM J Res Dev*, vol. 2, no. 2, pp. 159–165, Apr. 1958, doi: 10.1147/rd.22.0159.

[28] A. Thakkar and K. Chaudhari, "Predicting stock trend using an integrated term frequency–inverse document frequency-based feature weight matrix with neural networks," *Applied Soft Computing Journal*, vol. 96, no. 106684, pp. 1–13, Nov. 2020, doi: 10.1016/j.asoc.2020.106684.

[29] X. Peng, Y. Bao, and Z. Huang, "Perceiving Beijing's 'City Image' Across Different Groups Based on Geotagged Social Media Data," *IEEE Access*, vol. 8, no. 5, pp. 93868–93881, May 2020, doi: 10.1109/ACCESS.2020.2995066.

[30] M. Tan and Q. V. Le, "EfficientNetV2: Smaller Models and Faster Training," in *Proceedings of the 38th International Conference on Machine Learning*, Virtual: PMLR, Jul. 2021, vol. 2021, no. 7, pp. 10096–10106. doi: 10.48550/arXiv.2104.00298

[31] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA: IEEE, Jun. 2016, vol. 2016, no. 6, pp. 2818–2826. doi: 10.1109/CVPR.2016.308.

[32] K. He, X. Zhang, S. Ren, and J. Sun, "Identity Mappings in Deep Residual Networks," in *14th European Conference on Computer Vision (ECCV 2016)*, Amsterdam, The Netherlands: Springer, 2016, vol. 2016, no. 9, pp. 630–645. doi: 10.1007/978-3-319-46493-0_38.

[33] S. Sharma, K. Guleria, S. Tiwari, and S. Kumar, "A deep learning based convolutional neural network model with VGG16 feature extractor for the detection of Alzheimer Disease using MRI scans," *Measurement: Sensors*, vol. 24, no. 100506, pp. 1–8, Dec. 2020, doi: 10.1016/j.measen.2022.100506.

[34] S. Montaha, S. Azam, A. Rafid, P. Ghosh, M. Hasan, M. Jonkman, and F. De Boer, "BreastNet18: A High Accuracy Fine-Tuned VGG16 Model Evaluated Using Ablation Study for Diagnosing Breast Cancer from Enhanced Mammography Images," *Biology (Basel)*, vol. 10, no. 12, pp. 1347–1347, Dec. 2021, doi: 10.3390/biology10121347.

[35] M. Rahimzadeh and A. Attar, "A modified deep convolutional neural network for detecting COVID-19 and pneumonia from chest X-ray images based on the concatenation of Xception and ResNet50V2," *Inform Med Unlocked*, vol. 19, no. 100360, pp. 1–9, Jan. 2020, doi: 10.1016/j.imu.2020.100360.

[36] Z. Zhao, E. B. A. Bakar, N. B. A. Razak, and M. N. Akhtar, "Corrosion image classification method based on EfficientNetV2," *Heliyon*, vol. 10, no. 17, pp. 1–22, Sep. 2024, doi: 10.1016/j.heliyon.2024.e36754

[37] B. Dey, J. Ferdous, R. Ahmed, and J. Hossain, "Assessing deep convolutional neural network models and their comparative performance for automated medicinal plant identification from leaf images," *Heliyon*, vol. 10, no. 1, pp. 1–15, Jan. 2024, doi: 10.1016/j.heliyon.2023.e23655.

[38] N. Becherer, J. Pecarina, S. Nykl, and K. Hopkinson, "Improving optimization of convolutional neural networks through parameter fine-tuning," *Neural Comput Appl*, vol. 31, no. 8, pp. 3469–3479, Aug. 2019, doi: 10.1007/s00521-017-3285-0.

[39] Y. AbdulAzeem, W. M. Bahgat, and M. Badawy, "A CNN based framework for classification of Alzheimer's disease," *Neural Comput Appl*, vol. 33, no. 16, pp. 10415–10428, Aug. 2021, doi: 10.1007/s00521-021-05799-w

[40] V. R. Joseph, "Optimal ratio for data splitting," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 15, no. 4, pp. 531–538, Aug. 2022, doi: 10.1002/sam.11583.

[41] M. T. R, A. Thakur, M. Gupta, D. Sinha, K. Mishra, V. Venkatesan, and S. Guluwadi, "Transformative Breast Cancer Diagnosis using CNNs with Optimized ReduceLROnPlateau and Early Stopping Enhancements," *International Journal of Computational Intelligence Systems*, vol. 17, no. 14, pp. 1–18, Jan. 2024, doi: 10.1007/s44196-023-00397-1.

[42] R. Bruni, G. Bianchi, and P. Papa, "Hyperparameter Black-Box Optimization to Improve the Automatic Classification of Support Tickets," *Algorithms*, vol. 16, no. 46, pp. 1–14, Jan. 2023, doi: 10.3390/a16010046.

[43] V. Singh, V. K. Asari, and R. Rajasekaran, "A Deep Neural Network for Early Detection and Prediction of Chronic Kidney Disease," *Diagnostics*, vol. 12, no. 116, pp. 1–22, Jan. 2022, doi: 10.3390/diagnostics12010116.

 ISSN 2723-6471

[44] Divya and M. Narwal, "Digital Distractions in the Workplace: Exploring Cyberloafing Impact on Employee Behaviour and Innovation," *Virtual Economics*, vol. 6, no. 4, pp. 7–24, Dec. 2023, doi: 10.34021/ve.2023.06.04(1)

[45] M. Nakayama and C. C. Chen, "Digital Distractions and Remote Work," *Information Resources Management Journal*, vol. 35, no. 1, pp. 1–17, Sep. 2022, doi: 10.4018/IRMJ.308675.

[46] S. L. Vadlamani and O. Baysal, "Studying Software Developer Expertise and Contributions in Stack Overflow and GitHub," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Adelaide, SA, Australia: IEEE, Sep. 2020, vol. 2020, no. 9, pp. 312–323. doi: 10.1109/ICSME46990.2020.00038.

[47] X. Wang, X. Lin, and N. Hajli, "Understanding Software Engineers' Skill Development in Software Development," *Journal of Computer Information Systems*, vol. 61, no. 2, pp. 108–117, Mar. 2021, doi: 10.1080/08874417.2019.1566805

[48] S. Janssens and V. Zaytsev, "Go with the flow: software engineers and distractions," in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, New York, NY, USA: ACM, Oct. 2022, vol. 2022, no. 10, pp. 934–938. doi: 10.1145/3550356.3559101.

[49] V. W.-S. Tseng, M. L. Lee, L. Denoue, and D. Avrahami, "Overcoming Distractions during Transitions from Break to Work using a Conversational Website-Blocking System," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, New York, NY, USA: ACM, May 2019, vol. 2019, no. 5, pp. 1–13. doi: 10.1145/3290605.3300697.

[50] L. Song, "An AI-Powered System to Increase User's Productivity using Computer Vision and Machine Learning," in *Computer Science and Information Technology (CS and IT)*, Academy and Industry Research Collaboration Center, May 2024, vol. 2024, no. 5, pp. 183–192. doi: 10.5121/csit.2024.141016

[51] D. Rogers, A. Preece, M. Innes, and I. Spasic, "Real-Time Text Classification of User-Generated Content on Social Media: Systematic Review," *IEEE Trans Comput Soc Syst*, vol. 9, no. 4, pp. 1154–1166, Aug. 2022, doi: 10.1109/TCSS.2021.3120138.