International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 15, No.1 (Mar-24)

http://dx.doi.org/10.12785/ijcds/150191

A Parallel Approach of Cascade Modelling Using MPI4Py on Imbalanced Dataset

Suprapto[1,*], Wahyono[2], Nur Rokhman[3] and Faisal Dharma Adhinata[4]

[1-3]*Department of Computer Science and Electronics, Universitas Gadjah Mada, Yogyakarta, Indonesia*
[4]*Faculty of Informatics, Institut Teknologi Telkom Purwokerto, Banyumas, Indonesia*

Received 26 Dec. 2023, Revised 23 Feb. 2024, Accepted 2 Mar. 2024, Published 10 Mar. 2024

Abstract: Machine learning is crucial in categorizing data into specific classes based on their features. However, challenges emerge, especially in classification, when dealing with imbalanced datasets. An imbalanced dataset occurs when there is a disproportionate number of samples across different classes. It leads to a machine learning model's bias towards the majority class and poor recognition of minority classes, often resulting in notable prediction inaccuracies for those less represented classes. This research proposes a cascade and parallel architecture in the training process to enhance accuracy and speed compared to non-cascade and sequential. This research will evaluate the performance of the SVM and Random Forest methods. Our findings reveal that employing the Random Forest method, configured with 100 trees, substantially enhances classification accuracy by 4.72%, elevating it from 58.87% to 63.59% compared to non-cascade classifiers. Furthermore, adopting the Message Passing Interface for Python (MPI4Py) for parallel processing across multiple cores or nodes demonstrates a remarkable increase in training speed. Specifically, parallel processing was found to accelerate the training process by up to 4.35 times, reducing the duration from 1725.86 milliseconds to a mere 396.54 milliseconds. These results highlight the advantages of integrating parallel processing with a cascade architecture in machine learning models, particularly in addressing the challenges associated with imbalanced datasets. This research demonstrates the potential for substantial improvements in classification tasks' accuracy and efficiency.

Keywords: Cascade classifier, Imbalanced data, MPI4Py, Parallel processing, SVM

1. Introduction

Data classification using machine learning is a process where computer algorithms are trained to differentiate and categorize data into certain classes or groups based on the features of the data [1]. This process begins with a training phase, where the model is given a dataset that has been properly labeled. Each data entry in this dataset has features representing relevant characteristics and a label indicating its class. The algorithm gradually adjusts its weights and inter-parameters during training to minimize prediction errors. Challenges often arise in classification using machine learning when dealing with imbalanced datasets [2][3]. In the real world, imbalanced datasets usually occur because one event or object occurs less frequently than another [4]. For example, cases of specific diseases, such as rare tumors or certain medical conditions, may be much fewer than normal images or more common conditions. Then, action recognition can occur in an imbalanced dataset when more images are taken for specific actions or there are more sources on the internet for specific actions.

Imbalanced datasets have an uneven class distribution, which can result in the model being biased and favoring the majority class. It is a severe problem in many applications, such as action recognition, where this research uses this dataset. Action recognition is the process of identifying and understanding the subject's activity or behavior in an image [5]. It is an important branch of computer vision and machine learning that focuses on interpreting images to detect and classify various types of movement or activity. Action recognition on imbalanced data is a significant challenge in computer vision and machine learning because models tend to be biased towards the more dominant class in the dataset [6][7]. Imbalanced data occurs when the number of samples for different action classes is not comparable, resulting in poor recognition performance for classes with fewer representations. Apart from that, the training speed is also considered so that the proposed method is time efficient. This research aims to solve the problem of imbalanced data for accurate and efficient action recognition case studies. So, the contribution to this research is how to handle imbalanced data using machine learning, which considers accuracy and training speed.

Various approaches are used to overcome the problem of imbalanced data. Previous research used deep learning

E-mail address: sprapto@ugm.ac.id, wahyo@ugm.ac.id, nurrokhman@ugm.ac.id, faisal@ittelkom-pwt.ac.id

through a modified transfer learning model to overcome the problem of imbalanced datasets in the case of pattern recognition [7]. The use of deep learning indeed automates the feature extraction process in images. However, the dataset is not very large, so the resulting accuracy is not good. Therefore, in this study, we propose another approach. One strategy for overcoming this problem is to use a cascade classifier [8]. Currently, the development of the cascade classifier method is still focused on increasing accuracy. Lu and Guo have developed a cascaded traffic classifier that integrates binary and multiclass sub-classifiers, enhancing the precision of flow-level traffic classification [9]. Neugebauer et al. have introduced two methods for preprocessing data that enhance the effectiveness of cascade classifiers by making the decision boundaries more selective [10]. Meanwhile, Xu and Yuan have designed an enhanced AdaBoost classifier specifically for sonar images, resulting in superior classification accuracy [11]. These three studies still focus on accuracy, but there is still a long way to go in terms of training speed.

Accuracy is an important aspect in developing machine learning models such as cascade classifiers, but speed is also a crucial factor that must be considered. There are several strategies to speed up the cascade classifier process. One strategy enables them to be satisfied more readily and with fewer features, thus reducing the overall number of features the cascade utilizes [12]. Furthermore, a comprehensive algorithm has been suggested that conceptualizes cascade training as a tree search procedure, which helps minimize the total number of features required by the cascade [13]. Reducing the number of features can eliminate important features in the image. Therefore, in this research, we propose using parallel processing in the training stage of cascade classification.

The training process can be very time-consuming because each cascade stage is usually trained sequentially, where each classifier must be powerful enough to recognize complex actions while ensuring that irrelevant data is eliminated efficiently. In this research, we propose using parallel processing in the training stage. With parallel processing, large computing tasks are divided into smaller sub-processes that can be executed simultaneously on different processors or computers in a network [14]. It reduces training time significantly because distributed processing allows many operations to be performed simultaneously rather than sequentially. The strong point in parallel processing is that each cascade stage can train its classifier on different subsets of the data in parallel. In this research, the parallel processing method is the Message Passing Interface for Python (MPI4Py). The choice to use MPI4Py is because MPI4Py has become the industry standard for parallel programming in high-performance computing environments [15]. Then, MPI4Py also provides a relatively easy interface [16], so developers can easily learn it.

This research uses parallel processing to optimize the training speed of the cascade classifier. This research will also compare the performance of Joblib with MPI4Py for parallel processing at the training stage. In this research, the training method used is Support Vector Machine (SVM) or Random Forest. SVM [17] and Random Forest [18] are examples of machine learning methods used for classification. SVM is very powerful because of its ability to handle high-dimensional data and its effectiveness in cases where the number of features exceeds the number of samples [19]. On the other hand, Random Forest is an ensemble algorithm consisting of many decision trees. Random Forest is known for its ability to reduce overfitting [20], which often occurs in single decision trees, and its ability to handle large datasets with high dimensions.

There are several sections in the structure of this paper. Section 2 is a literature review containing the strengths and weaknesses of related previous research. Section 3 is a research methodology that explains the steps for a cascade classifier using parallel processing and its evaluation. Section 4 contains results and discussion related to research experiments. Section 5 is the conclusion of this research.

2. Related Work

The challenge of data imbalance in action recognition significantly impacts the performance and accuracy of recognition systems [7]. This research introduces OccamNets, a novel neural network architecture that mitigates dataset bias by inherently favoring simpler hypotheses. However, the accuracy of the results could be improved. Recent studies have explored various methods, including cascade classifiers, to address this challenge and enhance the accuracy of imbalanced data. Cascade classifiers, which leverage machine learning, offer greater accuracy than single machine learning classifiers [21]. These classifiers divide the data flow into multiple stages, with each focusing on a specific aspect of the classification task. This method has significantly improved the overall recognition accuracy in various tasks, such as physical activity recognition [22], sentiment analysis of tweets [23], and weather forecasting [24]. The structure of cascade classifiers enhances generalization performance and stability in classification tasks, surpassing conventional single classifiers. Furthermore, cascade classifiers can optimize the accuracy of any state-of-the-art classifier, making them an asset in machine learning research. However, the primary limitation of cascade classifiers is their lengthy training time. Mukabe et al. [25] found a critical regarding training time reveals that Neural Networks, especially retrained models, train significantly faster than Haar Cascades. For instance, a neural network model retrained on 3,454 images across different classes completed its training in less than an hour (approximately 40 minutes), achieving a test accuracy of 92%. In contrast, Haar Cascades training took considerably longer, with a cascade completing all 20 stages of its training in over a day (25 hours), achieving a Hit Rate (HR) of 99.52% and a False Alarm Rate (FA) of 47.44%. Annamraju and Singh [26] also focused on optimizing parameters for training cascade

classifiers using Local Binary Pattern (LBP) and Histogram of Gradients (HOG) features to enhance object detection tasks' efficiency. They established optimal ranges for various parameters through experimental analysis, resulting in an average training time of 25,000 seconds (approximately 6.94 hours) for classifiers. With these optimized parameters, the classifier achieved an average true positive detection rate of 88% on a test set of 4,000 images. Wahyono et al. [27] research highlighted the significantly longer training time associated with cascade modelling, nearly four times that of non-cascade models (2403.47 ms vs. 700.95 ms). Although accuracy could be improved, these findings suggest further enhancements to increase efficiency and accuracy. Therefore, employing a cascade classifier can improve accuracy in classification case studies. Nonetheless, training speed emerges as a separate challenge from efficiency. Therefore, the proposed research focuses on accelerating training times and enhancing accuracy beyond previous studies.

3. Research Methodology

Action recognition on imbalanced data using the cascade classifier method in machine learning. Figure 1 shows the action recognition process in this research. The action recognition stage begins with data acquisition in the form of action data. Then, the data goes through a pre-processing stage, where resizing and color conversion to grayscale are carried out to prepare the data before feature extraction. Next, the image is feature extracted using the Histogram of Oriented Gradients (HOG), Local Binary Patterns (LBP), or Scale-Invariant Feature Transform (SIFT) methods, all three of which are popular techniques for visual feature extraction in image processing. After the features have been successfully extracted, the next stage is data training using a cascade classifier in machine learning, which is optimized using Joblib or MPI4Py to speed up the training process. The final stage is to make predictions to evaluate accuracy.

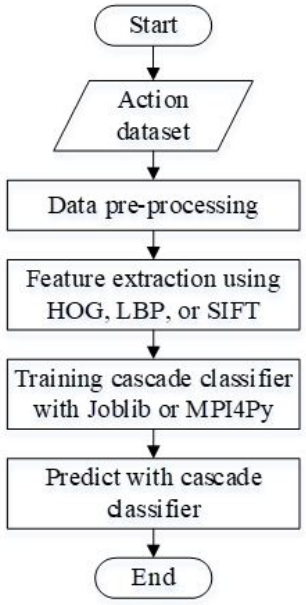

Figure 1. Stages of research method

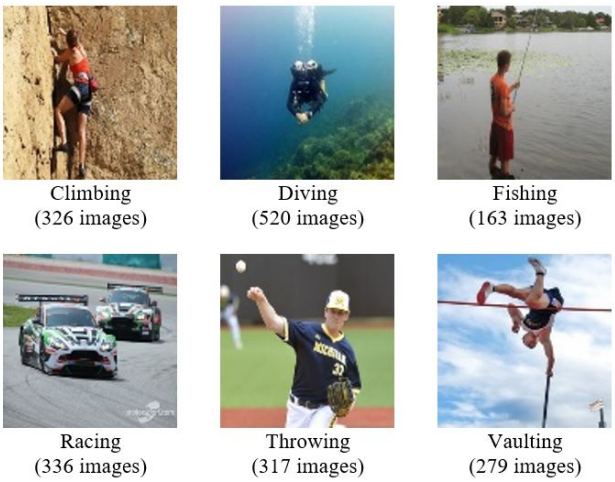

Figure 2. Example of dataset in this research

A. Data Acqusition

The Biased Action Recognition (BAR) dataset is a collection of real-world images that are organized into six categories of actions. These categories are intentionally associated with specific locations. The selection of these six categories was done meticulously by reviewing the imSitu database, which contains various still images depicting actions sourced from Google Image Search, each tagged with the type of action being performed and the location [6]. The categories of actions tend to occur in recognizable locations. These locations needed to be unique enough for each action that one could determine the type of action based on the associated location alone. Figure 2 shows an example of the dataset used in this research.

In this research, the amount of data in each class is different, called imbalanced data [28]. Figure 3 shows the distribution of each class in this research dataset. Even in the Fishing class, the amount of data is three times less than in the Diving class. So, the majority class (Diving) has many samples, while the minority class (Fishing) has very few samples. In contrast to the iris flower dataset, this dataset contains 150 samples from three different iris species (setosa, versicolor, virginica), each with 50 samples [29]. The same number of samples for each species makes this dataset an example of a balanced dataset.

B. Data Pre-processing

Resizing an image dataset is crucial in utilizing Histogram of Oriented Gradients (HOG), Local Binary Patterns (LBP), and Scale-Invariant Feature Transform (SIFT) for machine learning, mainly when the dataset includes images of varying sizes. The main reason is that HOG, LBP, and SIFT describe objects' local texture, shape information, and key points in images by calculating the distribution

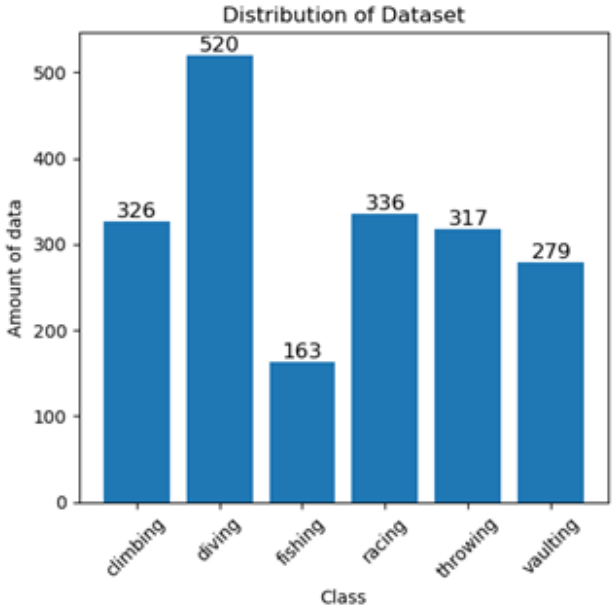

Figure 3. Example of dataset in this research

of gradient or edge directions, local intensity patterns, and scale-invariant features, respectively. For the features extracted by HOG, LBP, and SIFT to be consistent and comparable between images, each image in the dataset must have uniform dimensions. If the image size varies, the gradient distribution, local intensity patterns, and detected key points will differ significantly, causing mismatches in feature representation. This research utilized resizing to 64 x 64 to ensure uniformity [30]. The next pre-processing stage involves color conversion from Red, Green, and Blue (RGB) to grayscale. This conversion is pivotal as it adjusts the use of the HOG, LBP, and SIFT features, which do not require multichannel information. Additionally, using grayscale images eliminates unnecessary information, such as RGB colors, while preserving and highlighting edge, texture, and key point information for pattern recognition, making it more effective for the analysis with HOG, LBP, and SIFT.

C. Feature Extraction

The choice of feature extraction method should be based on the specific characteristics of the data used and the goal of the computer vision application. HOG is better suited for applications that require robust shape and texture analysis [31], LBP is suitable for applications that require efficient texture recognition [32] and SIFT is ideal for applications that require robustness to scale changes and rotation [33].

1) HOG

Image feature extraction using HOG is a technique in image processing and computer vision used to identify objects in images [34]. This process begins with image normalization to reduce lighting and shadow variations. Then, the image is divided into small cells (usually 8x8 pixels). Each pixel's gradient (direction and strength) for each cell is calculated. The next step is to calculate the oriented gradient histogram in each cell. This histogram depicts the frequency of gradient directions in that cell. These cells are then grouped into larger blocks, and the histograms of the cells within the blocks are normalized to increase robustness to lighting changes. The HOG feature of the entire image is generated by combining these histograms of all blocks. Figure 4 shows an illustration of HOG.

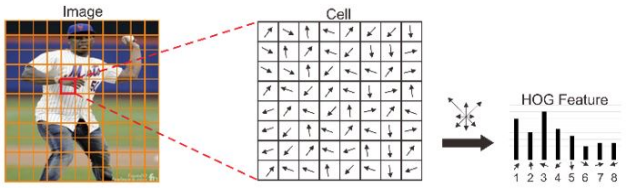

Figure 4. HOG method

In this research, the HOG configuration uses orientation 8, which determines that the gradient histogram will have 8 orientation bins. This means the gradient direction in each cell will be divided into 8 different directions. Second, pixels per cell is used (16,16), which sets the size of each cell to 16x16 pixels. The gradient histogram will be calculated in a 16x16 pixel grid across the entire image. Third, cells_per_block (1,1) indicates that each block (for histogram normalization) will consist of 1x1 cells. This means there is no additional grouping of cells within blocks, and each cell will be considered its block. Fourth, visualize True, which indicates that the function will also return a HOG visualization image in addition to the HOG feature vector. It is useful for understanding how HOG represents imagery. Finally, multichannel=False indicates that the input image is a grayscale image and not a multichannel image (like RGB). This configuration overall determines how HOG features are calculated and represented from a given image, focusing on gradient orientation, cell size, block normalization, visualization, and image channel type.

2) LBP

The Local Binary Patterns (LBP) method is a popular approach in image processing and texture analysis, which was first introduced by Ojala et al. in the 1996 [35]. The essence of this method is to change the pixel values in the image into a binary pattern by comparing the pixel values with the values of neighboring pixels. This process is carried out by surrounding the target pixel with a circle containing several sample points and then comparing the target pixel values with the pixel values at those sample points. If the value of a neighboring pixel is greater than or equal to the central pixel, then it is given a binary value of 1, and if it is smaller, it is given a value of 0. The results of this comparison are then converted into a binary value and interpreted as a decimal value to obtain the LBP pattern for that pixel.

The configuration of the LBP method in this research

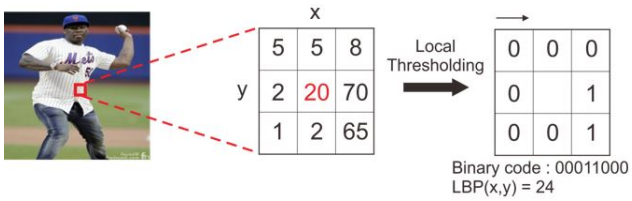

Figure 5. LBP method

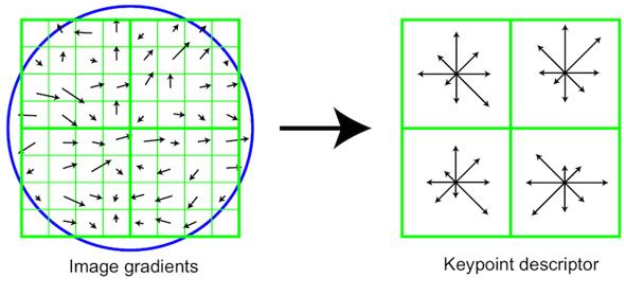

Figure 6. SIFT method [36]

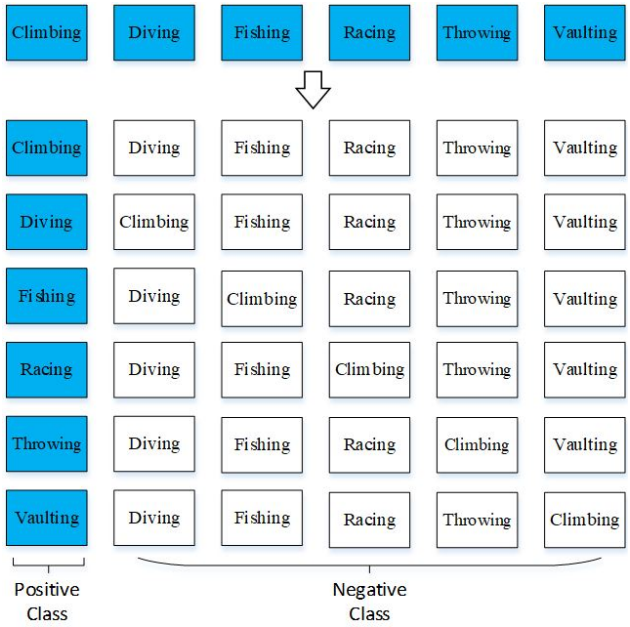

Figure 7. Cascade modelling

uses the feature.local_binary_pattern function to calculate the LBP pattern from the image, where P=8 determines the number of neighboring points used for comparison in the LBP circle, and R=1 determines the circle's radius. The uniform method is used to calculate a uniform LBP pattern. Then, the function calculates a histogram from the LBP values obtained with np.histogram, where the histogram is set to have bins according to the number of possible LBP patterns plus margin. The histogram is then normalized by dividing each value by the total number of values in the histogram plus a minimal value (1e-7) to avoid dividing by zero. Finally, the function returns the normalized histogram as a representation of the LBP features of the image. Figure 5 shows an illustration of LBP.

3) *SIFT*

Scale-Invariant Feature Transform (SIFT) is a feature extraction method used in image processing to identify and describe local features of images in an invariant manner to scale, rotation, and lighting changes. SIFT works in several main stages: first, it detects extreme points in scale space by using the Difference of Gaussian (DoG) to identify potential key points that stand out at various scales. Second, more precise selection and localization of key points by eliminating key points with low contrast or those located at the edges. Third, determining the orientation for each key point is based on the surrounding local gradient, allowing invariance with rotation. Finally, it creates a descriptor for each key point by collecting local gradients around it in an orientation histogram, which produces a powerful and information-rich feature vector.

In this research, the SIFT program configuration will determine the maximum number of desired features. After that, the program initializes the SIFT detector and uses the detectAndCompute method to detect key points and calculate SIFT descriptors from the grayscale image. Since the number of generated descriptors can vary, the program following handles normalization of the number of features: if the descriptor is None, it is flattened, and if the number of features is less than the desired maximum number, padding with zeros is performed to reach that maximum number. If the number of features exceeds the maximum limit, they are truncated. If no descriptors are found, the function returns a null array with the size of the maximum number of features. Figure 6 shows an illustration of SIFT.

D. *Training Data*

We use six data classes, so the cascade classifier function has six models. The training configuration in this research uses one positive class and five other classes as negative classes. Our research used a division of training and testing data with a composition of 80:20. Each class is trained via a loop for single processing. Then, for parallel processing training, this research uses joblib and MPI4Py provided by the Python library. In this research, we will compare the performance of the SVM and Random Forest methods.

The SVM method is a machine learning technique for classification [19] and regression [37]. Figure 7 shows the cascade model in this research. At its core, SVM creates a hyperplane or series of hyperplanes in a multidimensional space separating different data classes. The uniqueness of SVM lies in how it chooses a hyperplane with the largest margin between two data classes, which means this hyperplane not only separates the two classes but is as far away as possible from the closest data points from the two classes.

Random Forest is a machine learning method in the

ensemble learning algorithm category [38]. This algorithm combines many decision trees to create a more powerful and accurate model. Each tree in a Random Forest is trained on a random subset of the data using a sampling technique with replacement, known as bootstrap sampling. Additionally, at each split in the tree, only a random number of features are considered, which increases diversity among trees and reduces the risk of overfitting. When making predictions, Random Forest takes a voting approach. The most common output from all trees is the final prediction for classification tasks.

E. Pseudocode of Action Recognition

The pseudocode of the action recognition classification system that implements parallel processing is shown in Figure 8. MPI initialization contains a comm used to carry out communication and management operations in MPI. Then, rank is used to identify individual processes in MPI operations. Then, size is a variable that stores the total number of these processes. Then, implement data acquisition via the *load_images* function, which retrieves the images in the folder.

The data pre-processing uses resizing and color conversion from RGB to grayscale. Then, the features of the pre-processed image are extracted using HOG, LBP, or SIFT. Next, the extraction of image data features using the HOG, LBP, or SIFT method is carried out by the HoG, LBP, or SIFT function. Then, the *training_data* function is used to process training data using machine learning. This research used SVM and Random Forest methods for training data.

The last function is the *cascade_predict*, which predicts testing data using the cascade classifier model. Predictions using a cascade classifier have as input a series of classifiers and thresholds. For each classifier in the cascade sequence, this function will calculate the probability of a positive prediction and divide the samples into two groups: those that are accepted (probability above the threshold) and those that are rejected (below or equal to the threshold). Accepted samples are labelled with the current classifier index, while rejected samples are prepared for the next iteration. This process continues until all samples are processed, or all classifiers in the cascade have been used. If samples still have not been predicted after passing through all classifiers, this function uses the last classifier to make a final prediction. The result is a list containing the index of the classifier that accepted each sample, indicating at which stage in the cascade the sample was accepted or rejected. This approach increases efficiency and reduces false positives, as only the most likely positive samples proceed through all cascade stages.

In the algorithm section, a training process is carried out in cascade modelling using parallel processing. In this research, MPI4Py is used for parallel processing. For comparison, we also use Joblib.

```
Initialization:
SET comm TO MPI.COMM_WORLD
SET rank TO comm.Get_rank()
SET size TO comm.Get_size()

DEFINE FUNCTION load_images(folder):
    FOR subfolder IN os.listdir(folder):
        SET label
        SET img TO img.resize
        SET img TO color.rgb2gray
        SET features TO FUNCTION(HOG, LBP, or SIFT)
    RETURN features, label

DEFINE FUNCTION HOG, LBP, or SIFT (image):
    SET features to HOG, LBP, or SIFT
    RETURN features

DEFINE FUNCTION training_data(X_train, y_train):
    SET clf TO SVC or Random Forest
    clf.fit(X_train, y_binary)
    RETURN clf

DEFINE FUNCTION cascade_predict(classifiers, thresholds):
    SET test_data
    FOR i, (clf, threshold) IN enumerate zip (classifiers, thresholds):
        SET probs TO clf.predict_proba
        SET accepted_indices TO (probs > threshold)
        SET rejected_indices TO (probs <= threshold)
        SET prediction(passed_indices[accepted_indices])
        SET passed_indices TO passed_indices[rejected_indices]
    RETURN prediction

Algorithm:
IF rank EQUALS 0:
    SET X_train, X_test, y_train, y_test TO train_test_split
    (test_size=0.2)
ELSE:
    SET X_train, X_test, y_train, y_test TO None

SET X_train TO comm.bcast(X_train, root=0)
SET y_train TO comm.bcast(y_train, root=0)
SET X_test TO comm.bcast(X_test, root=0)
SET y_test TO comm.bcast(y_test, root=0)

Divide training_jobs based on rank
SET num_class, start_index, end_index
SET classifier TO training_data(A vs non-A, B vs non-B, ...)
SET all_classifiers TO comm.gather(classifiers, root=0)

IF rank EQUALS 0:
    SET thresholds, all_classifiers
    SET y_pred TO cascade_predict
    SET accuracy TO accuracy_score
```

Figure 8. Pseudocode of cascade classification

1) Training Data using MPI4Py

MPI4Py is a Python library that allows Python programs to communicate using the Message Passing Interface (MPI) [15], a widely used standard for parallel programming in high-performance computing environments [39]. The first step is to send training and testing data to all multi-core nodes using the broadcast function. Then, the division of work uses input rank and size, where the division of work in each process will be responsible for a certain class subset. Then, each node carries out data training according to the

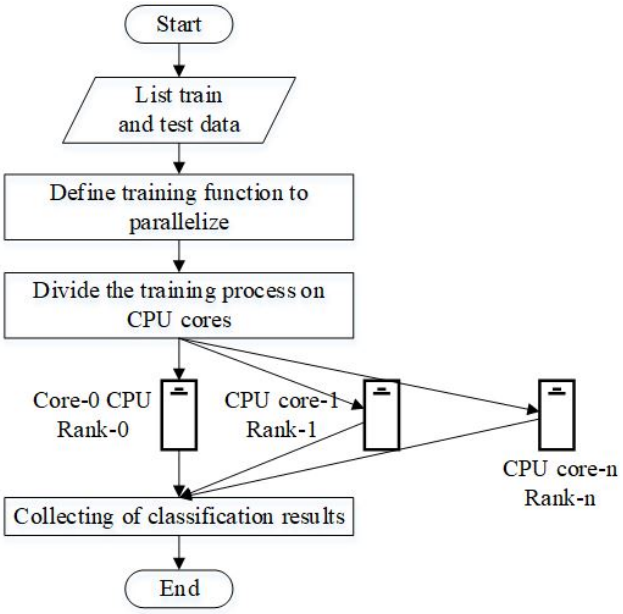

Figure 9. Parallel processing stages using MPI4Py

division. Collecting final training results at the root node using the gather function. Once complete, the root process will list all the classifiers that all processes have trained. Figure 9 shows the parallel processing using MPI4Py in this research.

The ideal use of parallel processing is usually measured using Amdahl's law. In this research, execution time was measured only during the training process. Amdahl's law formula is shown in Equation (1).

$$\text{Speedup} = \frac{1}{(1-P) + \frac{P}{N}} \tag{1}$$

where P is the proportion of programs that can be parallelized, N is the number of processor cores used, and $(1-P)$ is the proportion of programs that must be executed sequentially.

In this research, execution time was carried out at the training stage, so the P value was 100%. Therefore, if the number of processor cores used in parallel processing is four, the ideal speedup value is four times.

2) Training Data using Joblib

The use of joblib is to include training iterations in parallel. A Parallel class in joblib is used to perform parallel operations [40][41]. The Parallel object takes the argument n_jobs, which specifies the number of processes or threads used in parallel. If n_jobs=-1, then all available CPU cores will be used. Then, there is a delayed function, which wraps functions that want to run in parallel. It is necessary because parallel does not accept functions called directly with their

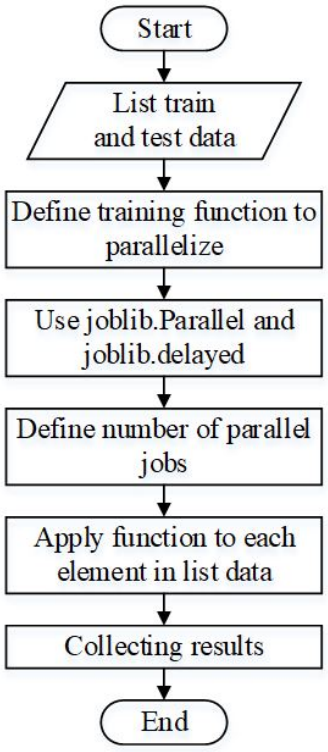

Figure 10. Parallel processing stages using joblib

arguments. Delayed creates a version of the function that is not immediately executed. Figure 10 shows the joblib stages in this research.

The use of joblib in this research begins with input in the form of a list of training and testing data. Then, define the function that will be parallelized, namely the data training function using SVM. The next step is to use joblib.Parallel and joblib.delayed to set up parallel processing. This joblib has input to determine the number of parallel jobs to be run. After that, the defined function is applied to each element in the training and testing data list. The results of each parallel job are collected, and the process ends once all the results have been collected. This process effectively utilizes parallel processing capabilities to increase the efficiency and speed of data processing for training needs.

F. System evaluation

This research will evaluate action recognition classification using the accuracy obtained from classification_report in Python. Accuracy is calculated by adding up all correct predictions (true positives and true negatives) and dividing it by the total number of predictions (the sum of true positives, true negatives, false positives, and false negatives) [42]. The accuracy formula is shown in equation (2). The result measures the action recognition model's effectiveness in correctly classifying samples. In this research, the accuracy results of the SVM and Random Forest methods and other supervised learning methods will be evaluated.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FN + FP} \quad (2)$$

Then, evaluate parallel processing using speedup and speedup. Speed refers to the time required to complete a task or program using parallel processing. By dividing a task into several sub-tasks that run simultaneously on a multicore processor, the time to complete the overall task is significantly reduced compared to running the same task sequentially on a single processor. Then, speedup measures how fast a program runs on p systems (for example, p processors) compared to running on just one system. The speedup formula is shown in equation (3) [41]. The main goal of parallel processing is to achieve significant speedup, enabling the completion of more complex and large tasks in less time.

$$S(p) = \frac{T(1)}{T(p)} \quad (3)$$

$T(1)$ is the execution time on one processor, and $T(p)$ is the time on p processors.

4. Results and Discussion

A. Hardware and Software Specifications

Using hardware on a single computer with a multicore greatly affects parallel processing performance because each core on the processor can run processes or threads independently, allowing simultaneous execution of various tasks. In the context of parallel processing, tasks that previously had to be executed sequentially can now be split up and executed simultaneously across multiple cores. It significantly increases throughput and reduces total processing time. This research used a Core i5 computer with 6 Core CPU and 8GB RAM. This multicore architecture will be used to evaluate parallel processing.

This research uses several Python libraries for the action recognition process: NumPy, Scikit-learn, Scikit-image, Joblib, and MPI4Py. NumPy is a fundamental library supporting large-dimensional arrays and mathematical operations. Scikit-learn, often abbreviated to sklearn, is a library that provides simple and effective tools for data analysis and machine learning. Scikit-image or skimage is a library aimed at image processing, providing tools for image manipulation and feature extraction often required in computer vision. Joblib, with Parallel and delayed modules, is used for simple and efficient task parallelization, which can significantly speed up heavy computing. Finally, MPI4Py is a Python implementation of the Message Passing Interface (MPI) for writing programs that can be run in parallel. In this research, the experiments carried out were the influence of threshold on cascade classification, the influence of the kernel on SVM and the number of trees in Random Forest, and the influence of the number of nodes on training speed.

TABLE I. THE ACCURACY RESULT OF THRESHOLD EFFECT

Threshold	Accuracy		
	HOG	LBP	SIFT
0.9	28.59%	33.33%	14.36%
0.8	33.72%	34.36%	20.05%
0.7	39.36%	39.49%	24.62%
0.6	44.48%	45.64%	26.67%
0.5	48.59%	50.26%	28.72%
0.4	52.20%	54.36%	33.85%
0.3	53.72%	55.90%	46.67%
0.2	42.95%	45.64%	40.51%
0.1	26.02%	41.03%	28.21%

B. The Effect of Threshold in Cascade Classification

In the classification of imbalanced datasets using a cascade classifier, threshold adjustments can have a significant impact on model performance. Cascade classifiers generally apply fast and efficient filters to reject non-target areas quickly while maintaining correct detection. In this experiment, the SVM kernel is the Radial Basis Function (RBF), and the number of trees in the Random Forest is 100. Table I compares the accuracy results of the HOG, LBP, and SIFT methods with the influence of threshold on cascade classification.

Based on Table I, the best threshold value for the three feature extraction methods is 0.3. Using a high threshold from the experiments can make the classification results more inaccurate. This threshold is small enough to increase the model's sensitivity to minority classes. A lower threshold means that the model is more likely to classify an example as a minority class, which helps reduce the number of False Negatives (i.e., cases where objects from the minority class are not detected).

Experimental results also show that LBP's accuracy is better than HOG and SIFT. In this research, object segmentation was not carried out. So, background differences can significantly influence classification performance. LBP focuses on relative comparisons between pixels, tending to be more robust to such changes than HOG and SIFT, which are more sensitive to overall context and background variations. The SIFT feature extraction method emphasizes scale and rotation invariance, which requires preprocessing or further adjustments. So, SIFT requires focusing on objects to produce relevant features. In contrast, LBP can be applied directly to images without segmentation with information-rich features [43]. Based on these results, the following experiment will use a threshold value configuration of 0.3 and use the LBP feature extraction method.

C. The effect of parameters on supervised learning

In SVM experiments, the parameters to be evaluated are kernel RBF, linear, and polynomial. Meanwhile, in the Random Forest method, the number of trees is evaluated at this stage. The number of trees evaluated is 20, 40, 60,

TABLE II. THE ACCURACY RESULT USING SVM

SVM Kernel	Accuracy
RBF	55.90%
linear	48.72%
poly	52.31%

TABLE III. THE ACCURACY RESULT USING RANDOM FOREST

Number of trees	Accuracy
20	54.76%
40	55.27%
60	56.92%
80	58.87%
100	63.59%
120	59.90%

80, 100, and 120. The parameter configuration with the best accuracy results will be used for experiments on the number of datasets and parallel processing.

Table II shows that the best results are achieved using the RBF kernel. The RBF kernel is effective in dealing with non-linearities in the data. Imbalanced datasets tend to have complex and non-linear decision boundaries. The RBF kernel can map data to a higher dimensional space, where the classes can be separated non-linearly, allowing the SVM to make more precise classification decisions. The RBF kernel also has adaptive properties to the local structure of the data. It can adjust its decisions based on the local distribution of data points, which is especially important for minority classes that are often scattered or isolated in the feature space. By adapting to this local distribution, the RBF kernel can more accurately recognize and classify examples from the minority class. Therefore, the next experiment will use the RBF kernel on SVM.

When using the Random Forest method for classifying imbalanced datasets, increasing the number of trees can help the model overcome bias toward the majority class. Each tree in the ensemble can explore different aspects of the data, including the unique characteristics of the minority class. With a large enough number of trees, there is a greater likelihood that some trees will become 'specialists' in recognizing minority classes, thereby increasing the model's overall ability to classify samples from that class. So, in this experiment based on Table III, the number of trees was 100, resulting in optimal accuracy. However, there is a saturation point where adding more trees does not significantly improve performance, especially if the trees are highly correlated. So, when the number of trees is increased to 120, the resulting accuracy can decrease. Therefore, the next experiment will use a number of trees of 100 for the Random Forest method.

TABLE IV. THE ACCURACY RESULT USING RANDOM FOREST

Dataset Proportion	Accuracy	
	Non-cascade	Cascade
25%	47.72%	50.52%
50%	55.83%	59.28%
75%	52.12%	56.01%
100%	58.87%	63.59%

D. The Effect of Dataset Number on Accuracy Results

In this experiment, the method used is the Random Forest, utilizing many trees of 100. This configuration was used because the accuracy results of this method produced the best accuracy in previous experiments.

Table IV shows the accuracy results for the non-cascade and cascade classifier. Cascade Classifiers have better accuracy than a non-cascade classifier because of their gradual and hierarchical decision-making approach. An object is detected through a series of stages, each consisting of a different classifier. Initial stages are usually designed to quickly reject sub-regions of the image that do not contain the target object, using a relatively simple classifier. Subsequent stages become increasingly complex, gradually focusing on more promising areas with more accurate and detailed classifiers. The advantages of this approach are efficiency and accuracy. Cascade Classifier can quickly weed out irrelevant areas at low computational cost, allowing more resources to be invested in analyzing more challenging areas in detail. It differs from a non-cascade classifier, which must analyze the entire image thoroughly, which is often inefficient and can produce more errors in the form of false positives and negatives.

E. The Effect of Parallel Processing on Training Stage

In this experiment, the SVM method uses the RBF kernel, while the Random Forest method uses 100 trees. The training time for non-parallel data using the SVM method is 1073.02 ms. Then, the data training time without parallelism using the Random Forest method is 1725.86 ms. In this experiment, the number of nodes used is 2, 3, and 6. This number of nodes refers to the number of classes being 6 to match the division of nodes.

Using Joblib and MPI for Python (MPI4Py) can significantly increase the training speed of the cascade classifier on a single machine with multi-cores. Joblib enables easy and efficient parallelization of tasks, especially on multi-core machines. By dividing the training tasks and running them in parallel across multiple cores, the time required to complete the training process can be reduced dramatically. It is because Joblib effectively leverages modern machines with multi-threading and multi-processing capabilities. On the other hand, MPI4Py provides an interface for Message Passing Interface (MPI), enabling efficient data communication between processes running in parallel. Although typically

used in distributed or cluster computing, MPI4Py can also be leveraged in a multi-core single machine to optimize resource usage and synchronization between processes. With MPI4Py, data and instructions can be transferred quickly between different processes, minimizing waiting times and increasing overall efficiency. Tables V and VI show the training process results using parallel processing on SVM and Random Forest methods.

Both using SVM and Random Forest, the resulting speedup increases equally. With each additional processor core, there is always an increase in speed. However, the speedup value is not ideal, as proven by increasing the use of cores on Joblib and MPI4Py. The speedup is not the same as the number of cores used. The speed is not ideal because there is overhead from parallelization management, namely communication between processors and data sharing. Using MPI4Py is faster than using Joblib. It happens because of differences in how the two manage communication and synchronization between processes. MPI4Py is based on the Message Passing Interface (MPI), an industry standard for highly efficient inter-process communication in distributed and parallel computing. MPI is designed for large-scale and high-performance computing. On the other hand, Joblib focuses more on ease of use and integration with Python.

In the speedup results, the Random Forest method is faster than SVM. Random Forest is an ensemble-based algorithm with many independently trained decision trees, making it suitable for parallel processing. Each tree in a random forest can be built and trained independently of the other trees to distribute tasks to various cores or nodes in a parallel environment quickly and efficiently. It allows maximum use of resources and results in significant speed improvements.

In contrast, margin based SVM searches for optimal hyperplanes to separate data classes. The optimization process used in SVM training, especially on multiclass datasets, is often more complex and involves solving significant optimization problems. So, it is inefficiently divided into parallel tasks compared to Random Forest. Although in this research, parallelization was carried out in SVM training, such as decomposition of the training stage, the ability of this algorithm to benefit from parallel processing is generally less than optimal compared to Random Forest. Thus, differences in algorithm architecture and ease of parallelization are the main reasons why Random Forest shows better speedup in the training process on a cascade classifier with parallel processing compared to SVM. Random Forest naturally supports parallelization at a more granular level, allowing for significant performance improvements when using parallel processing techniques such as Joblib and MPI4Py.

F. Discussion

Handling imbalanced datasets using a cascade classifier involves several special strategies to overcome bias towards more dominant classes. In this research, we used a threshold of 0.3. This threshold is low enough to enhance the model's ability to detect minority classes. A reduced threshold indicates that the model tends to identify a sample as belonging to a minority class, thereby decreasing the incidence of False Negatives. When using SVM, there is an increase from 53.91% to 55.90% when using cascade SVM. The best accuracy was produced using the Random Forest with 100 trees, namely 63.59%, while when using the non-cascade Random Forest, it was 58.87%. However, compared to previous research, as shown in Table VII, this result is better where previous research used deep learning. The dataset in previous research also used the BAR dataset. While deep learning models are powerful in recognizing complex patterns and generally provide better performance in classification, deep learning typically requires large and balanced datasets for effective training, which is often a challenge in the case of imbalanced datasets. Then, previous research also applied a combination of HOG and cascade SVM. The use of HOG is less appropriate in feature extraction for datasets without this segmentation stage. Besides that, the running time for the SVM cascade does not use parallel processing, so the time spent on the training process is 2403.47 ms. This differs from this research, which uses parallel, which only takes 396.54 ms for the training process.

Besides comparing the performance of deep learning in the case of an imbalanced dataset, we also tested traditional supervised learning, namely the Naïve Bayes, Decision tree, and K-Nearest Neighbor (KNN) methods. We also use relatively new and popular machine learning methods, namely XGBoost and LightGBM. The Naïve Bayes method uses the default parameter configuration from the Sklearn library in Python language. Then, the Decision tree method uses a Gini criterion configuration, and the minimum split number of samples is two. Then, in the KNN method, the K value is five, and the Euclidean distance is used to calculate the distance. Then, configure XGBoost and LightGBM using the defaults from the xgboost and lightgbm libraries.

The resulting accuracy is no better than using a cascade classifier. The cascade classifier produces better accuracy than traditional and latest supervised learning because it works with a series of stages where each stage consists of several weak classifiers. Each stage evaluates whether a sample meets the criteria to proceed to the next stage. If a sample fails at an early stage, it is immediately rejected, reducing processing time for samples that are not targeted. It is especially effective in the case of an imbalanced dataset, where the majority of samples are negative (non-target), because it allows the system to discard irrelevant cases quickly. Figures 11 and 12 show the results of the classification report using non-cascade and cascade Random Forest.

In cascade Random Forest, there is an increase in F1-Scores in several classes compared to non-cascade Random Forest. Diving, fishing, racing, and vaulting activities had

TABLE V. THE RESULT OF PARALLEL PROCESSING USING SVM

Number of Nodes	Joblib		MPI4Py	
	Training time (ms)	Speedup	Training time (ms)	Speedup
2	1302.51	0.82	727.28	1.47
3	827.37	1.29	506.26	2.12
6	581.61	1.84	303.65	3.53

TABLE VI. THE RESULT OF PARALLEL PROCESSING USING RANDOM FOREST

Number of Nodes	Joblib		MPI4Py	
	Training time (ms)	Speedup	Training time (ms)	Speedup
2	1449.48	1.19	965.25	1.79
3	978.72	1.76	668.48	2.58
6	669.37	2.58	396.54	4.35

TABLE VII. COMPARISON WITH PREVIOUS RESEARCH

Method	Accuracy
OccamResNet [7]	52.6%
HOG + Cascade SVM [27]	56.38%
KNN	51.41%
Naïve Bayes	42.16%
Decision Tree	46.53%
SVM	53.91%
XGBoost	56.56%
LightGBM	58.44%
Random Forest	58.87%
Ours (LBP + Cascade Random Forest)	63.59%

	precision	recall	f1-score	support
climbing	0.62	0.77	0.69	66
diving	0.62	0.66	0.64	105
fishing	0.27	0.12	0.17	33
racing	0.64	0.67	0.66	57
throwing	0.58	0.54	0.56	69
vaulting	0.53	0.51	0.52	59
accuracy			0.59	389
macro avg	0.54	0.54	0.54	389
weighted avg	0.57	0.59	0.58	389

Accuracy: 58.87%

Figure 11. Classification report using non-cascade Random Forest

	precision	recall	f1-score	support
climbing	0.61	0.77	0.68	43
diving	0.68	0.72	0.70	54
fishing	0.29	0.14	0.19	14
racing	0.67	0.75	0.71	32
throwing	0.52	0.42	0.47	26
vaulting	0.75	0.58	0.65	26
accuracy			0.64	195
macro avg	0.59	0.56	0.57	195
weighted avg	0.62	0.64	0.62	195

Accuracy: 63.59%

Figure 12. Classification report using cascade Random Forest

higher F1 scores, indicating improved model performance in classifying these activities. Through these results, the use of cascade Random Forest can improve performance compared to the use of non-cascade Random Forest. This increase also affects the resulting accuracy value. Accuracy increases by 4.72% using cascade Random Forest.

Using cascade classification also presents challenges in the data training stage. In this research, we propose using parallel processing via Joblib and MPI4Py. The experimental results are shown in Table V and VI. In parallel processing using SVM, the resulting speedup was 3.53 using MPI4Py. Using MPI4Py can also speed up 4.35 times compared to sequential training with Random Forest. Sequential training is a training stage that is sequential or executed one by one while waiting for the model training process to complete. These results demonstrate the significant efficiency of the parallel approach. In sequential training, each cascade stage must be trained one after another. It can be a time-consuming process. In contrast, with parallel processing, each node or core on the computer can be assigned to carry out training simultaneously, which makes training time more efficient.

The increase in training speed in this research is not yet ideal, according to the increase in the number of cores used. It happens because of overhead. Communication between processes is a crucial factor. When the workload is shared between cores, data must be frequently sent and received between these processes. This program uses broadcast and gather operations, so the time required to send and receive data can be significant, especially if the data being transferred is large, such as features in an image.

The limitation of parallel processing is the ability to break a dataset into parts that can be processed independently without requiring too complex communication or coordination between processors. If the dataset cannot be

split well, parallelization becomes less effective and may increase processing time rather than decrease. Additionally, the overhead associated with parallel processing must also be considered, which includes the time required to initialize and coordinate threads or processes, split data, combine results, and manage shared memory access. This overhead can be quite significant, especially on tasks with small dataset sizes or relatively simple computations, where the time gains generated by parallelization are insufficient to cover the additional time spent on managing the overhead. Therefore, parallel processing can improve performance for specific tasks, but it needs to be carefully weighed in the context of the overhead and characteristics of the dataset to be processed.

5. Conclusion

Training data using a cascade classifier can increase accuracy compared to a non-cascade classifier because the cascade classifier approach allows more focused and gradual processing. In a cascade classifier, objects are detected through a series of stages, with each stage using a classifier with a different class of data to quickly reject image areas that do not contain the target object. This approach can increase accuracy, where the accuracy using non-cascade is 52.47%, then using the cascade classifier is 53.72%. It shows that there is an increase in accuracy of 1.25%. On the other hand, using parallel processing in cascade classifier training can speed up the training process significantly. This research proves that the training speed can increase by 3.57 times using parallel processing compared to without parallel processing. By using multicore capabilities, training tasks can be run in parallel on various cores, reducing the time required to complete the training process. Therefore, combining a more accurate and efficient cascade classifier approach with the speed offered by parallel processing provides a very effective solution for increasing accuracy and reducing training time in these action recognition imbalanced data classification applications. In future research, other feature extraction or supervised learning methods can be explored to increase accuracy in imbalanced datasets. The feature extraction method can also use a hybrid of several methods, but the resulting computing time must still be considered.

Acknowledgement

This research was funded by the Department of Computer Science and Electronics, Universitas Gadjah Mada, under Research laboratory Capacity Grant Year 2023 (120/UN1/FMIPA.1.3/TU/PT.01.03/2023).

References

[1] I. H. Sarker, "Machine learning: Algorithms, real-world applications and research directions," *SN computer science*, vol. 2, no. 3, p. 160, 2021.

[2] S. Yadav and G. P. Bhole, "Handling imbalanced dataset classification in machine learning," in *2020 IEEE Pune Section International Conference (PuneCon)*. IEEE, 2020, pp. 38–43.

[3] Z. ao Huang, Y. Sang, Y. Sun, and J. Lv, "A neural network learning algorithm for highly imbalanced data classification," *Information Sciences*, vol. 612, pp. 496–513, 2022.

[4] X.-F. Feng, L.-C. Yang, L.-Z. Tan, and Y.-G. Li, "Risk factor analysis of device-related infections: value of re-sampling method on the real-world imbalanced dataset," *BMC medical informatics and decision making*, vol. 19, pp. 1–8, 2019.

[5] M. G. Morshed, T. Sultana, A. Alam, and Y.-K. Lee, "Human action recognition: A taxonomy-based survey, updates, and opportunities," *Sensors*, vol. 23, no. 4, p. 2182, 2023.

[6] J. Nam, H. Cha, S. Ahn, J. Lee, and J. Shin, "Learning from failure: De-biasing classifier from biased classifier," *Advances in Neural Information Processing Systems*, vol. 33, pp. 20 673–20 684, 2020.

[7] R. Shrestha, K. Kafle, and C. Kanan, "Occamnets: Mitigating dataset bias by favoring simpler hypotheses," in *European Conference on Computer Vision*. Springer, 2022, pp. 702–721.

[8] Y.-G. Fu, H.-Y. Huang, Y. Guan, Y.-M. Wang, W. Liu, and W.-J. Fang, "Ebrb cascade classifier for imbalanced data via rule weight updating," *Knowledge-Based Systems*, vol. 223, p. 107010, 2021.

[9] G. Lu and R. Guo, "Cascaded classifier for improving traffic classification accuracy," *Iet Communications*, vol. 11, no. 11, pp. 1751–1758, 2017.

[10] J. Neugebauer, O. Kramer, and M. Sonnenschein, "Improving cascade classifier precision by instance selection and outlier generation," in *International Conference on Agents and Artificial Intelligence*, vol. 2. SCITEPRESS, 2016, pp. 96–104.

[11] H. Xu and H. Yuan, "An svm-based adaboost cascade classifier for sonar image," *IEEE Access*, vol. 8, pp. 115 857–115 864, 2020.

[12] D. Sychel, P. Klsk, and A. Bera, "Relaxed per-stage requirements for training cascades of classifiers," in *ECAI 2020*. IOS Press, 2020, pp. 1523–1530.

[13] D. Sychel, P. Klesk, and A. Bera, "Branch-and-bound search for training cascades of classifiers," in *International Conference on Computational Science*. Springer, 2020, pp. 18–34.

[14] U. M. Malik, M. A. Javed, J. Frnda, J. Rozhon, and W. U. Khan, "Efficient matching-based parallel task offloading in iot networks," *Sensors*, vol. 22, no. 18, p. 6906, 2022.

[15] E. Oluwasakin, T. Torku, T. Sun, A. Yinusa, S. Hamden, S. Poudel, J. Vargas, and K. N. Poudel, "Minimization of high computational cost in data preprocessing and modeling using mpi4py," *Available at SSRN 4455401*, 2023.

[16] L. A. Barba, A. Klockner, P. Ramachandran, and R. Thomas, "Scientific computing with python on high-performance heterogeneous systems," *Computing in Science & Engineering*, vol. 23, no. 04, pp. 5–7, 2021.

[17] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*, 1992, pp. 144–152.

[18] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[19] F. D. Adhinata, A. Harjoko, and Wahyono, "Object searching on

video using orb descriptor and support vector machine," in *Advances in Computational Collective Intelligence: 12th International Conference, ICCCI 2020, Da Nang, Vietnam, November 30–December 3, 2020, Proceedings 12*. Springer, 2020, pp. 239–251.

[20] C. Gao, H. Lin, and H. Hu, "Forest-fire-risk prediction based on random forest and backpropagation neural network of heihe area in heilongjiang province, china," *Forests*, vol. 14, no. 2, p. 170, 2023.

[21] A. B. Shetty, J. Rebeiro *et al.*, "Facial recognition using haar cascade and lbp classifiers," *Global Transitions Proceedings*, vol. 2, no. 2, pp. 330–335, 2021.

[22] L. Mo, Y. Zhu, and L. Zeng, "A multi-label based physical activity recognition via cascade classifier," *Sensors*, vol. 23, no. 5, p. 2593, 2023.

[23] A. Yenkikar, C. N. Babu, and D. J. Hemanth, "Semantic relational machine learning model for sentiment analysis using cascade feature selection and heterogeneous classifier ensemble," *PeerJ Computer Science*, vol. 8, p. e1100, 2022.

[24] L. Chen, X. Zhong, F. Zhang, Y. Cheng, Y. Xu, Y. Qi, and H. Li, "Fuxi: A cascade machine learning forecasting system for 15-day global weather forecast," *npj Climate and Atmospheric Science*, 2023.

[25] C. Mukabe, N. Suresh, V. Hashiyana, T. Haiduwa, and W. Sverdlik, "Object detection and classification using machine learning techniques: A comparison of haar cascades and neural networks." New York, NY, USA: Association for Computing Machinery, 2022, p. 86–97.

[26] A. Kumar Annamraju and A. Deep Singh, "Analysis and optimization of parameters used in training a cascade classifier," vol. 3, p. 25, May 2015.

[27] W. Wahyono, Suprapto, A. Rezky, N. Rokhman, K.-H. Jo *et al.*, "Handling imbalanced data using a cascade model for image-based human action recognition," *Journal of Computing Science and Engineering*, vol. 17, no. 4, pp. 207–215, 2023.

[28] M. M. Taamneh, S. Taamneh, A. H. Alomari, and M. Abuaddous, "Analyzing the effectiveness of imbalanced data handling techniques in predicting driver phone use," *Sustainability*, vol. 15, no. 13, p. 10668, 2023.

[29] Y. Chen, Z. Hua, Y. Tang, and B. Li, "Multi-source information fusion based on negation of reconstructed basic probability assignment with padded gaussian distribution and belief entropy," *Entropy*, vol. 24, no. 8, p. 1164, 2022.

[30] K. Lei and L. Yujing, "A new pedestrian detection method based on histogram of oriented gradients and support vector data description," 2024.

[31] Z. Xiang, H. Tan, and W. Ye, "The excellent properties of a dense grid-based hog feature on face recognition compared to gabor and lbp," *IEEE Access*, vol. 6, pp. 29 306–29 319, 2018.

[32] X. Qian, X.-S. Hua, P. Chen, and L. Ke, "Plbp: An effective local binary patterns texture descriptor with pyramid representation," *Pattern Recognition*, vol. 44, no. 10-11, pp. 2502–2515, 2011.

[33] L. Tang, S. Ma, X. Ma, and H. You, "Research on image matching of improved sift algorithm based on stability factor and feature descriptor simplification," *Applied Sciences*, vol. 12, no. 17, p. 8448, 2022.

[34] K. T. Islam, S. Wijewickrema, R. G. Raj, and S. O'Leary, "Street sign recognition using histogram of oriented gradients and artificial neural networks," *Journal of imaging*, vol. 5, no. 4, p. 44, 2019.

[35] T. Ojala, M. Pietikäinen, and D. Harwood, "A comparative study of texture measures with classification based on featured distributions," *Pattern recognition*, vol. 29, no. 1, pp. 51–59, 1996.

[36] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, pp. 91–110, 2004.

[37] F. Rustam, A. A. Reshi, A. Mehmood, S. Ullah, B.-W. On, W. Aslam, and G. S. Choi, "Covid-19 future forecasting using supervised machine learning models," *IEEE access*, vol. 8, pp. 101 489–101 499, 2020.

[38] M. Belgiu and L. Drăguţ, "Random forest in remote sensing: A review of applications and future directions," *ISPRS journal of photogrammetry and remote sensing*, vol. 114, pp. 24–31, 2016.

[39] K. Nölp and L. Oden, "Simplifying non-contiguous data transfer with mpi for python," *The Journal of Supercomputing*, pp. 1–22, 2023.

[40] A. R. J. Pangestu, R. Kurniawan, I. W. A. Swardiana, A. L. Latifah *et al.*, "Parallel computing implementation of marine heat waves detection," in *2023 International Conference on Computer, Control, Informatics and its Applications (IC3INA)*. IEEE, 2023, pp. 436–439.

[41] N. Nagy, M. Aljabri, A. Shaahid, A. A. Ahmed, F. Alnasser, L. Almakramy, M. Alhadab, and S. Alfaddagh, "Phishing urls detection using sequential and parallel ml techniques: Comparative analysis," *Sensors*, vol. 23, no. 7, p. 3467, 2023.

[42] T. Fawcett, "An introduction to roc analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[43] Z. Zhang and M. Wang, "A simple and efficient method for finger vein recognition," *Sensors*, vol. 22, no. 6, p. 2234, 2022.

Suprapto received the B.Sc. degree in Computer Science from Universitas Gadjah Mada, Indonesia, the Master Degree from the University of Indonesia Sandwich Program with University of Maryland University College, USA and the Ph.D. degree from the Universitas Gadjah Mada, Indonesia. He is an Associate Professor at the Department of Computer Science and Electronics, Universitas Gadjah Mada, Indonesia. His research interests include algorithm analysis and design, simulation, computational logic, graph theory, and pattern recognition.

Wahyono received the B.Sc. degree in Computer Science from Universitas Gadjah Mada, Indonesia, and the Ph.D. degree from The University of Ulsan, South Korea. He is an Associate Professor at the Department of Computer Science and Electronics, Universitas Gadjah Mada, Indonesia. His research interests include machine learning, computer vision, and pattern recognition. He actively participates as a member of the societies, a reviewer in reputable international journals, and an editor in several journals.

Nur Rokhman received the B.Sc., Master, and Ph.D. degrees in Computer Science from Universitas Gadjah Mada, Indonesia. He is an Associate Professor at the Department of Computer Science and Electronics, Universitas Gadjah Mada, Indonesia. His research interests include Computation, Information Systems, and outlier detection. He actively participates as a member of societies and a reviewer in reputable international journals.

Faisal Dharma Adhinata earned his Master of Computer Science (M.Cs.) degree in computer science from Universitas Gadjah Mada, Indonesia. He is pursuing a Doctoral Degree in Computer Science at the same university. In addition, he serves as a lecturer in the Department of Software Engineering at Institut Teknologi Telkom Purwokerto, Indonesia. His research areas include Artificial Intelligence, image processing, and computer vision.