Optimizing Hyperparameters of Temporal Convolutional Network Architecture Using the Hyperband Method for Human Eye Movements Classification

1st Daffa Bil Nadzary
Departemen Teknik Elektro dan Teknologi Informasi
Universitas Gadjah Mada
Yogyakarta, Indonesia
daffa.bil.nadzary@mail.ugm.ac.id

2nd Sunu Wibirama
Departemen Teknik Elektro dan Teknologi Informasi
Universitas Gadjah Mada
Yogyakarta, Indonesia
sunu@ugm.ac.id

3rd Syukron Abu Ishaq Alfarozi
Departemen Teknik Elektro dan Teknologi Informasi
Universitas Gadjah Mada
Yogyakarta, Indonesia
syukron.abu@ugm.ac.id

***Abstract*—Gaze-based interaction is one of many implementations of Industry 5.0. The system is highly beneficial in various conditions, especially for individuals with motion-impaired people. The implementation of an eye-based interaction system requires a system that can recognize and classify different types of eye movements in humans. In the eye movement classification field, Temporal Convolutional Network (TCN) model is proven to be a state-of-the-art model. In this research, performance improvement is conducted on the existing TCN model by optimizing hyperparameters using the Hyperband method. We also analyzed the individual hyperparameters to examine the influence of each hyperparameter on the TCN model. The research results show that the TCN model, after undergoing hyperparameter optimization, experiences performance improvement using two different evaluation methods. The LOVO method evaluation shows the highest increase in the saccade movement class by 1%, while the K-Fold Cross Validation evaluation experiences significant improvements, namely 0.9% in fixation, 1.3% in saccade, 3.2% in noise, and 6.6% in smooth pursuit.**

***Index Terms*—Eye movement classification, Temporal Convolutional Network, Hyperparameter Optimization, Hyperband, K-Fold Cross Validation**

I. Introduction

Gaze-based Interaction is one of the implementations of Human-Computer Interaction (HCI). With the rapid development of technology in recent years, HCI serves as a complementary bridge for the transition from Industry 4.0 to Industry 5.0 [1]. Industry 5.0 places greater emphasis on the social aspect of technology, focusing on enhancing collaboration between humans and machines, which was not found in Industry 4.0 [2]. The implementations of Industry 5.0 can be found with the implementation of gaze-based interaction as a new way for humans to interact with machines.

To be able to develop an interactive system based on the movements of the human eye, a system capable of classifying different types of eye movements is needed. Such a system is called eye movement classification [3]. Eye movement classification can be a challenging task due to many reasons, such as noisy data, various types of eye movements, and the lack of algorithms that can handle such data.

With the remarkable strides made in the machine learning field, research on eye movement classification has increasingly shifted towards the application of machine learning techniques for classifying these eye movements. These algorithms are commonly referred to as data-driven algorithms. Elmadjian et al developed an eye movement classification model with the deep learning approach using the Temporal Convolutional Network (TCN) architecture, which has become the current state-of-the-art.

However, there is still potential to enhance the accuracy of the TCN architecture model proposed by Elmadjian et al. through the optimization of hyperparameters. Hyperparameters are parameters that cannot be changed during the training process of a machine learning model [4]. The technique to optimize a set of hyperparameters is known as hyperparameter optimization. The main objective of hyperparameter optimization is to calibrate the parameters in a deep learning model in order to achieve optimal accuracy. In the research conducted by Elmadjian et al, the hyperparameter optimization process is done using the Grid Search method [5]. Grid Search is a brute-force hyperparameter optimization method, meaning the algorithm will force its way to try every combination of hyperparameters possible. Grid search also has several limitations. This algorithm consumes a significant amount of resources and requires high computational power, especially as the hyperparameter space becomes larger and more complex. Therefore, alternative hyperparameter optimization methods are needed to optimize the hyperparameters within the existing TCN model.

This research aims to analyze the differences in the performance of the existing TCN architecture model after performing hyperparameter optimization using a more optimal method compared to the previous method employed. This research proposes the usage of Hyperband as the hyperparameter optimization method, which offers better convergence to the optimal value with low computational resource usage.

II. Related Work

Research on eye movement classification has been a long-standing area of study. Despite the wide range of eye movements that humans can perform, fixation, saccade, and smooth pursuit are the three most commonly used types of eye movements, especially in eye movement classification research. Fixation refers to eye movement where the eyes remain focused on a specific point for a certain duration. Saccade is a rapid, instantaneous eye movement that shifts the gaze from one point to another. On the other hand, smooth pursuit is a slower eye movement compared to saccade, simulating the tracking of a slowly moving point.

Several algorithms have been introduced to detect and classify these three eye movements. Traditionally, two classes of algorithms have been widely used: threshold-based algorithms and probability-based algorithms. However, in recent years, data-driven approaches have also gained prominence, leading to the existence of three main approaches in eye movement classification.

A. Threshold-based Algorithms

The key characteristic of these algorithms is the utilization of boundaries to define eye movements. In their development, three methods have emerged based on the implementation of these thresholds, where different features are used for thresholding. These implementations are used in three methods: I-VT, I-VMP, and I-DT [6].

I-VT is a method that falls under the category of velocity-based algorithms [7]. Velocity-based algorithms classify eye movements based on the speed of eye movement. Eye movements that exceed a specific velocity threshold are categorized as either saccades or smooth pursuits, while the remaining eye movements are classified as fixations. In contrast to I-VT, I-DT is a dispersion-based algorithm [8]. Similar to I-VT, I-DT also uses thresholds to classify detected eye movements. However, unlike I-VT, which uses velocity as the distinguishing feature, I-DT uses dispersion of the data to classify eye movements as saccades, smooth pursuits, or fixations, with the exception of saccadic eye movements, where velocity is still used as an indicator to determine whether the eye movement falls under the category of a saccade or not. If the velocity of the eye movement is below the first threshold, dispersion is further analyzed to distinguish between smooth pursuits and fixations. Lastly, I-VMP is another method that shares similarities with I-DT. Both methods employ a velocity threshold to identify saccadic eye movements. However, I-VMP differentiates smooth pursuits from fixations using eye movement itself, while I-DT utilizes dispersion. Overall, I-VMP generally provides better evaluation results compared to I-VT [9].

These algorithms present certain limitations, as all three types heavily rely on the predetermined threshold values [8]. Another problem arises when there is noise in the data, which cannot be effectively detected by either of these algorithms.

B. Probability-based Algorithms

The challenges posed by threshold-based algorithms have led to the emergence of alternative solutions, such as probability-based algorithms, which are more sensitive to noise [10]. These algorithms perform classification based on probabilistic principles. There are several eye movement classification methods that fall under the umbrella of probability-based algorithms, such as Naive Segmented Linear Regression-Hidden Markov Model [11] and Bayesian Decision Theory Identification (I-BDT) [10]. These methods offer probabilistic frameworks for eye movement classification, taking into account the probability distributions and likelihoods of different eye movement classes.

NSLR-HMM is a probability-based model designed to be implemented in scenarios with high levels of noise in the data, addressing the limitations of many threshold-based models [11]. The incoming eye movement signals are segmented into independent signal segments and subsequently subjected to a linear regression system. In this method, a Hidden Markov Model (HMM) is employed as a classifier, capable of predicting three to four types of eye movements.

On the other hand, I-BDT is a method used for real-time eye movement classification. It employs Bayesian Decision Theory [12] in its classification process. For a given data set D, the probabilities for each class can be defined as follows

$$f(x) = \frac{p(e)p(D|e)}{p(D)} \tag{1}$$

where $p(e)$ represents the prior probability, indicating the probability of class e occurring in the population E, with $e \in E$. E represents the set of all classes in the model, which in this case refers to the types of eye movements observed (fixation, saccade, and smooth pursuit).

While the algorithms are able to solve several problems that arise within the threshold-based algorithms, this algorithm still suffers from the limitations that are also found in other traditional algorithms, namely the constraints in feature selection, both visible and invisible.

C. Data-driven Algorithms

Data-driven algorithms are a type of algorithm that has emerged alongside the development of machine learning methods, both for solving classification problems (supervised learning) and clustering problems (unsupervised learning). Therefore, machine learning can also be utilized as a means to classify human eye movements. The advancements in machine learning have enabled the automated classification of eye movements without human intervention [13].

With the abundance of machine learning methods that can be applied to the same problem, the development of eye movement classification models has also progressed. Zemblys et al. [14] conducted a comparison of ten machine learning algorithms for classifying human eye movements, including K-Nearest Neighbor, Gradient Boosting, and Random Forest. Additionally, the use of deep learning has gained traction due to its ability to achieve significantly higher accuracy compared to traditional machine learning methods. Hoppe et al. [15] designed a simple Convolutional Neural Network (CNN) architecture with a single layer, surpassing previous machine learning approaches by incorporating two primary input features: velocity and dispersion.

Furthermore, Startsev et al. [13] introduced 1D-CNN-BLSTM, a fusion of 1D Convolutional Neural Network with Bidirectional Long Short-Term Memory architecture. The results demonstrated that this model was capable of achieving accuracy surpassing existing deep learning and machine learning models. Subsequently, Elmadjan et al. [16] designed the TCN architecture, which has become the state-of-the-art in eye movement classification. Compared to models like LSTM, TCN offers several advantages, including longer memory due to its absence of gating mechanisms found in LSTM. Additionally, TCN incorporates a convolutional structure in its layers. With reduced memory requirements for training and the parallelizability of the convolutional structure, the training process becomes faster [17].

III. Approach Method

A. Dataset

The dataset employed in the research is the GazeCom dataset [18]. GazeCom is a human eye movement recording dataset that can be used as training data for the human eye movement classification model. It contains 18 videos, with each video featuring eye movement recordings from 47 to 52 participants. Participants were instructed to watch a video with a duration of approximately 20 seconds while their eye movements were recorded throughout the duration of the video.

The eye movements are captured using a remote eye tracker, with a sampling rate of 250 Hz, resulting in approximately 5000 data rows recorded in each session. Each row of data contains columns for time, x and y coordinates, and a label indicating the classification of the eye movement. The labels are obtained through manual labeling performed by two expert systems. The dataset includes four labels assigned to each movement, namely fixation, saccade, smooth pursuit, and noise. The noise label is assigned when there are errors in the eye tracker or when the participant blinks. Based on observation, the dataset contains roughly 72.5% of fixations, 10% of saccades, 11% of smooth pursuits, and 5.9% of noises.

B. Method

This study is an experimental research with the purpose to optimize the hyperparameters of the TCN model that has become the state-of-the-art in the eye movement classification problem. The research begins with data understanding, data preprocessing, TCN architecture modeling, hyperparameter tuning process, and evaluation of the model's outcomes.

In this study, the analysis step will be conducted after the hyperparameter of the TCN model has been optimized. The TCN model developed by Elmadjian et al. has a set of hyperparameters values obtained using the Grid Search method. This study will focus on improving performance and optimizing the search for these hyperparameters using the Hyperband method, as well as analyzing the impact of each hyperparameter on the TCN architecture model. The performance evaluation of the model will be conducted using two techniques: Leave One Video Out (LOVO) and K-Fold Cross Validation.

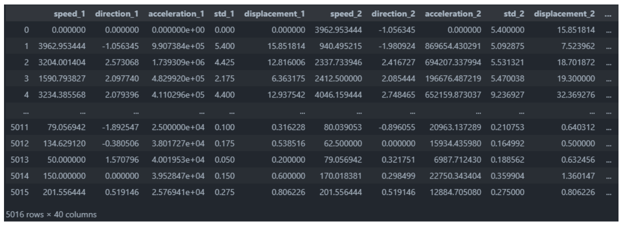
The research is fully conducted using Python as the programming language. We use Pandas, NumPy, and Scipy.io library for data preprocessing and manipulation processes, Matplotlib for data visualization, and Tensorflow combined with Keras-TCN to build the TCN architecture, as well as to tune the hyperparameters of the TCN model.

One of the advantages of using the Hyperband method compared to other methods is its resource allocation efficiency and time savings. The Hyperband method converges to optimal values more quickly, making it particularly advantageous when working with large datasets, such as the GazeCom dataset.

C. Data Preprocessing and Feature Engineering

The GazeCom dataset is mainly preprocessed by converting the data into a .csv file, using a Python library Scipy.io. CSV files offer better readability and are easy to store in a Pandas DataFrame format. The feature engineering process is done to add new features that can explain the data in various properties. These features are created by utilizing and manipulating existing features in the previous dataset, such as time and x-y coordinates. Startsev et al. added three additional features to the GazeCom dataset before feeding it into the deep learning model, namely speed, acceleration, and direction. Furthermore, Elmadjian et al. introduced two additional features, namely standard deviation and displacement.

These five additional features are extracted using different temporal window sizes, namely 1, 2, 4, 8, 16, 32, 64, and 128. The temporal window serves as a boundary used in feature creation. The purpose is to ensure that each data point represents the temporal relationship with other data points at the feature level. Choosing a larger temporal window size allows for capturing longer-term relationships within the data. Since the dataset was recorded at a sampling rate of 250 Hz, a window size of 128 represents a time span of 512 milliseconds. Figure 1 illustrates the results of the feature engineering process, displaying several additional features extracted with different temporal window sizes.



```
         speed_1  direction_1  acceleration_1  std_1  displacement_1      speed_2  direction_2  acceleration_2     std_2  displacement_2  ...
0       0.000000     0.000000    0.000000e+00  0.000        0.000000  3962.953444    -1.056345        0.000000  5.400000       15.851814  ...
1    3962.953444    -1.056345    9.907384e+05  5.400       15.851814   940.495215    -1.980924   869654.430291  5.092875        7.523962  ...
2    3204.001404     2.573068    1.739309e+06  4.425       12.816006  2337.733946     2.416727   694207.337994  5.531321       18.701872  ...
3    1590.793827     2.097740    4.829920e+05  2.175        6.363175  2412.500000     2.085444   196676.487219  5.470038       19.300000  ...
4    3234.385568     2.079396    4.110296e+05  4.400       12.937542  4046.159444     2.748465   652159.873037  9.236927       32.369276  ...
...          ...          ...             ...    ...             ...          ...          ...             ...       ...             ...  ...
5011   79.056942    -1.892547    2.500000e+04  0.100        0.316228    80.039053    -0.896055    20963.137289  0.210753        0.640312  ...
5012  134.629120    -0.380506    3.801727e+04  0.175        0.538516    62.500000     0.000000    15934.435980  0.164992        0.500000  ...
5013   50.000000     1.570796    4.001953e+04  0.050        0.200000    79.056942     0.321751     6987.712430  0.188562        0.632456  ...
5014  150.000000     0.000000    3.952847e+04  0.150        0.600000   170.018381     0.298499    22750.343404  0.359904        1.360147  ...
5015  201.556444     0.519146    2.576941e+04  0.275        0.806226   201.556444     0.519146    12884.705080  0.275000        0.806226  ...
5016 rows × 40 columns
```

Fig. 1. The GazeCom dataset after the feature engineering process.

D. *Feature Selection*

Before feeding the aggregated GazeCom dataset into the model, feature selection is required by reshaping the data to fit the TCN model. The TCN architecture includes deconvolution layers that allow the output of the architecture to have the same length as the input data. Therefore, we aim to leverage the properties of this architecture by sampling the data using a fixed-size temporal window.

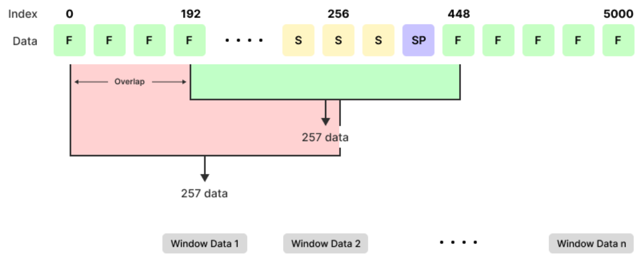


Fig. 2. The feature extraction process with a temporal window of 257 and an overlap of 192.

The temporal window consists of fixed-sized samples from this dataset, which will serve as the features inputted into the TCN model. The size of the temporal window is predetermined before the extraction process begins. Elmadjian et al. conducted experiments using three window sizes: 256, 385, and 514 samples, representing 1, 1.5, and 2 seconds of the captured data, respectively. Figure 2 illustrates the feature extraction process performed using a window size of 256 with an overlap of 192.

E. *Model Architecture Development*

After the feature extraction process is completed, the construction of the TCN model can proceed. The TCN model is built using the Keras-TCN library. This model takes inputs with a size of (None, a, b), where None represents the number of data entries (allowing for multiple predictions on multiple data sets if desired), a represents the size of the window used for data extraction (e.g., 257 samples), and b represents the number of selected features obtained from the previous feature engineering stage. Based on the research conducted by Startsev et al., it was found that the usage of the acceleration feature for the model slightly decreases the model performance. Consequently, Elmadjian et al. employed four primary features, namely speed, direction, displacement, and standard deviation. Thus, the modeling process is carried out using these four main features as input features.

Elmadjian et al. conducted the training process using a batch size of 128 and 20 epochs. The training process in this research was performed entirely on the Google Colab Pro cloud service, utilizing an NVIDIA A100 GPU with 40 GB of memory. To evaluate the model's performance, a specific validation mechanism called Leave One Video Out (LOVO), previously proposed by Startsev et al., was employed. This evaluation technique involves iteratively training the model using all the data from 17 out of 18 videos and evaluating it on the remaining video. The purpose is to generate a model that can generalize well to unseen video types.

In addition to the LOVO validation technique, the K-Fold Cross Validation method will also be employed, as it is a commonly used validation technique. Both validation methods will be measured using the F1 Score metric, considering the fact that the GazeCom dataset exhibits an imbalanced label distribution (with more than 70% of the dataset labeled as fixation).

F. *Hyperparameter Tuning*

The hyperparameter tuning process is performed on the created TCN model using the Hyperband method. Prior to the tuning process, several hyperparameters need to be defined along with the range of values to be tested. This is referred to as the hyperparameter space. Table I presents all the hyperparameters used in the tuning process, along with the respective range of values.

TABLE I
THE HYPERPARAMETER SPACE OF THE HYPERPARAMETER OPTIMIZATION PROCESS.

Hyperparameter	Data Type	Value Range
dropout_rate	float	0.1, 0.2, 0.3
number of filters	integer	64, 96, 128
number of stacks	integer	1, 2, 3, 4, 5
kernel size	integer	3, 4, 5, 6, 7, 8
dilations rate	integer	5, 6, 7, 8
batch size	integer	32, 64, 128
activation function	categorical	'relu', 'tanh'

The tuning process is conducted by specifying the objective of the metric to be maximized. In this study, the author aims to maximize the F1 Score on the validation data, as compared to the default accuracy metric.

G. *Evaluation*

The results of the model designed by Elmadjian et al. will be compared with the tuned model based on the findings of this research. The comparison will be conducted on both the overall F1 Score of the data and the F1 Score of each data class. In Elmadjian et al.'s study, a noise class was also added and evaluated due to its significant presence. Therefore, there are four data classes to be compared. In addition to using the LOVO validation technique, the comparison will also be performed using the K-Fold Cross Validation method. Since Elmadjian et al.'s research focused only on the LOVO evaluation technique, the TCN model configuration will be retrained and then evaluated using the K-Fold evaluation method before conducting the comparison.

IV. Results

A. Hyperparameter Optimization using Hyperband

The tuning process is conducted using 80% of the dataset for training and the remaining 20% for validation. The validation process is performed with the F1 Score metric as the objective. The Hyperband method is configured with a maximum number of 20 epochs.

In the implementation, the Hyperband method will perform training with a very small number of epochs, specifically 2. This process is conducted as a step of Hyperband to select subsets of hyperparameters that show potential for further training with a higher number of epochs. This process is iterative, with an increase in the number of epochs used in each tuning session.

Table II presents a comparison between the values of hyperparameters before and after tuning using the Hyperband method. The tuning results reveal four key parameters that yield the best outcomes in the search process using Hyperband. These four parameters are the dropout rate, kernel size, number of stacks, and dilation rate. The number of filters in the architecture, the activation function used, and the batch size used in the training process remain unchanged and did not undergo any modifications.

TABLE II
Hyperparameters value comparison between before and after the Hyperband tuning process.

Hyperparameter	Initial Value	Tuned Value
dropout_rate	0.3	0.1
number of filters	128	128
number of stacks	1	3
kernel size	8	4
dilations rate	8	5
batch size	128	128
activation function	'tanh'	'tanh'

B. Evaluation

The main evaluation in this eye movement classification case is performed using the LOVO and K-Fold validation techniques. For the first evaluation, based on the number of videos in the dataset, there will be 18 iterations, resulting in 18 individual models trained on different subsets of the dataset. In the first iteration, the first video in the sequence is left out, while the remaining videos are used for training the TCN model. Once the training process is completed, the evaluation continues on the previously left-out video.

TABLE III
Tuned and Untuned TCN Model Comparations using the LOVO Evaluation Method.

Model	Fixation	Saccade	SP	Noise	Total
Tuned	**0.9459**	**0.9014**	**0.758**	**0.7137**	**0.8638**
Untuned	0.9441	0.8921	0.7556	0.7039	0.8591

TABLE IV
Tuned and Untuned TCN Model Comparations using the K-Fold Cross Validation Evaluation Method.

Model	Fold	Fixation	Saccade	SP	Noise
Tuned	1	0.9524	0.9109	0.7971	0.7234
	2	0.9533	0.9074	0.8293	0.7313
	3	0.9537	0.9072	0.8126	0.7216
	4	0.9554	0.911	0.8191	0.7336
	5	0.9541	0.9028	0.8251	0.74
	Average	**0.9538**	**0.9079**	**0.8167**	**0.73**
Untuned	1	0.9457	0.8965	0.7577	0.7096
	2	0.9446	0.8966	0.7458	0.7041
	3	0.9432	0.8922	0.7698	0.6939
	4	0.9459	0.8957	0.7653	0.6999
	5	0.9466	0.8962	0.774	0.726
	Average	**0.9452**	**0.8955**	**0.7625**	**0.7067**

This study will compare the evaluation results with the study conducted by Elmadjian et al., which utilized the baseline TCN model. Tables III and IV present the final comparison results of the two TCN models. The untuned model refers to the TCN model that employs the parameter values and configurations determined by Elmadjian et al. in their research. Using the same number of features, the tuned TCN model outperforms the F1 score evaluation results of Elmadjian et al.'s model, particularly in the saccade class, with an improvement of 1%. In Elmadjian et al.'s publication, it is mentioned that the same configuration achieved an F1 score of 0.762 for the smooth pursuit class. However, it should be noted that this model was labeled with a remark indicating that its performance was attained through repeated training processes to obtain the highest score.

Furthermore, Elmadjian et al. also introduced their best model which used the same configuration but incorporated an output filtering mechanism. This mechanism assumes that any event that is shorter than 12 ms is a spurious output. These spurious outputs will be relabeled based on the majority label. This filtering process can result in an F1 score of 0.764 for the smooth pursuit class. However, in practical terms, this process cannot be applied, especially in real-time cases. Therefore, the use of such filtering is not employed in this research, and the focus is solely on the general output of the model.

An interesting observation can be seen in the evaluation results using the K-Fold Cross Validation method, where there is a significant improvement in the overall F1 score for all classes, particularly in the smooth pursuit class, reaching 0.8167. The fixation class shows a 0.9% improvement, while the saccade class shows a 1.3% improvement, the noise class shows a 3.2% improvement, and the smooth pursuit class shows a remarkable 6.6% improvement. This indicates that the tuned TCN model is able to better learn the patterns in the training dataset, especially for smooth pursuit movements.

C. TCN Hyperparameter Analysis

The hyperparameter analysis was conducted to examine the influence of each hyperparameter on both the TCN architecture and the training process itself. Several hyperparameters were tested, including batch size, dropout rate, number of kernels, number of stacks, dilation rate, and number of filters. The analysis of all hyperparameters was performed using the K-Fold Cross Validation evaluation method with a fold size of 5.

In the batch size parameter, three values were tested: 32, 64, and 128. Table V shows the comparison results of each batch size value against the evaluation scores. It can be observed that a batch size of 32 yields the highest evaluation score compared to the other two values. However, in terms of training time, a batch size of 128 actually results in faster training compared to 32. This is because the model trains 128 data batches in one iteration, leading to a faster training time.

TABLE V
TCN Model Evaluation Comparison on Different Values of Batch Size.

Batch Size	Fixation (Avg.)	Saccade (Avg.)	SP (Avg.)	Noise (Avg.)	Time per epoch
32	**0.948**	**0.9011**	**0.7811**	**0.7191**	18-19
64	0.9467	0.8992	0.7766	0.719	12-13
128	0.9451	0.8957	0.7711	0.7148	**11-12**

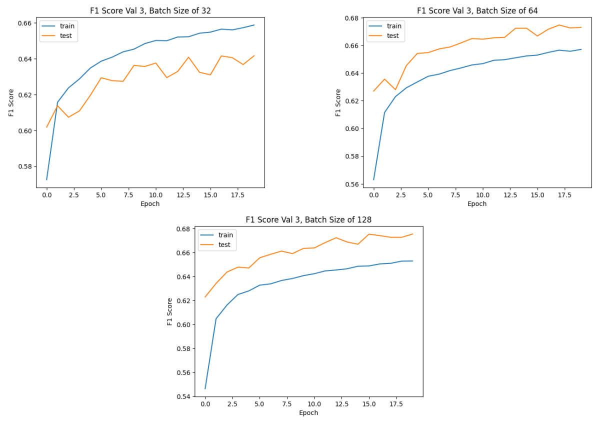


Fig. 3. History plot comparison in the training process of three different values of batch size.

In the dropout rate hyperparameter, three values were tested: 0.1, 0.2, and 0.3. The research results show that a dropout rate of 0.1 yields the best F1 score evaluation compared to the other two values. Since the dropout rate hyperparameter controls the random removal of connections from each layer, without affecting the number of neurons or layers involved, the number of trained parameters remains the same. Additionally, all dropout rate values result in the same training time per epoch. Comparing the history plots of the dropout rate hyperparameter, it was found that none of the three models show signs of overfitting or underfitting. Although in general, using a smaller dropout rate increases the risk of overfitting, in the GazeCom dataset, a dropout rate of 0.1 is considered safe for modeling purposes.

In the filter numbers hyperparameter, three values were tested: 64, 96, and 128. This parameter determines the number of filters used in TCN, which directly affects the size of the output that is fed into the output layer. Consequently, the number of filters used significantly impacts the number of parameters involved in the training process, adding complexity to the model and affecting the training time per epoch. The research results show that the highest F1 scores for the fixation and noise classes were achieved by the model with 128 filters, while for the saccade and smooth pursuit classes, the model with 96 filters yielded the highest scores. Considering that model complexity is inversely related to the required training time per epoch, the model with 64 filters exhibited the fastest training time compared to the other two models.

In the stack numbers hyperparameter, five values were tested: 1, 2, 3, 4, and 5. The research results show that overall, the values did not differ significantly. Several parameter values produced similar results. For the fixation and saccade classes, the highest scores were achieved by models with 1 and 3 stacks, while for the smooth pursuit class, the highest score was obtained with 2 stacks, and for the noise class, it was obtained with 1 stack. Unlike the nb filters hyperparameter, which is generally dominated by the highest number of filters and higher complexity, the stack numbers showed a different pattern, at least for the tested parameter set. Similar to the number of filters, the stack numbers also affect the complexity of the model, resulting in longer training times, up to 52 seconds per epoch. The research findings also indicate that in the GazeCom dataset, overfitting can still be avoided even with the highest number of stacks, although it is noted that there is a trade-off between model complexity and significantly longer training time.

In the kernel size hyperparameter, six values were tested: 3, 4, 5, 6, 7, and 8. The research results compare the evaluation outcomes of the six kernel sizes used in the model. One thing to note is that although the kernel size affects the complexity of the model by increasing the number of training parameters, there was no change in the training time per epoch in the training process, unlike the previous hyperparameters. In the table, it can be observed that the highest F1 score values are dominated by models with a kernel size of 8, which also has the highest number of training parameters. Similar to the number of filters and stacks, the plots indicate that none of the six models show signs of overfitting.

The last hyperparameter analyzed in the research is the dilation rate. In this hyperparameter, there are 4 dilation rate values tested: 5, 6, 7, and 8. The research results show that the training time per epoch depends on the complexity of the model determined by the dilation rate used. In the table, it can be observed that dilation rates of 7 and 8 generally result in the highest evaluation scores. However, there is not a significant difference between the four values. Similar to the other parameters, from the obtained plots, it can be seen that all four models show no signs of overfitting.

V. Conclusion

Based on the conducted research, the following conclusions can be drawn:

1) The hyperparameter-tuned TCN model can generate models with improved evaluation scores using both

LOVO and K-Fold Cross Validation methods. The latest TCN method provides a 1% improvement in the saccade class when evaluated using the LOVO method, and it achieves improvements of 0.9%, 1.3%, 6.6%, and 3.2% in the fixation, saccade, smooth pursuit, and noise classes, respectively, when evaluated using the K-Fold Cross Validation method *fixation*, *saccade*, *smooth pursuit*, serta *noise* dengan menggunakan metode evaluasi *K-Fold Cross Validation*.

2) The batch size hyperparameter has a significant impact on the performance of the TCN model. This parameter can affect the training time and the fitness of the model, determining whether the model is overfitting or not.

References

[1] B. Alojaiman, "Technological Modernizations in the Industry 5.0 Era: A Descriptive Analysis and Future Research Directions," *Processes*, vol. 11, no. 5, p. 1318, Apr. 2023. [Online]. Available: https://www.mdpi.com/2227-9717/11/5/1318

[2] X. Xu, Y. Lu, B. Vogel-Heuser, and L. Wang, "Industry 4.0 and Industry 5.0—Inception, conception and perception," *Journal of Manufacturing Systems*, vol. 61, pp. 530–535, Oct. 2021. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0278612521002119

[3] L. E. Sibert and R. J. K. Jacob, "Evaluation of eye gaze interaction," in *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. The Hague The Netherlands: ACM, Apr. 2000, pp. 281–288. [Online]. Available: https://dl.acm.org/doi/10.1145/332040.332445

[4] T. Yu and H. Zhu, "Hyper-Parameter Optimization: A Review of Algorithms and Applications," 2020, publisher: arXiv Version Number: 1. [Online]. Available: https://arxiv.org/abs/2003.05689

[5] G. Montavon, G. Orr, and K.-R. Müller, Eds., *Neural networks: tricks of the trade*, 2nd ed., ser. Lecture notes in computer science. Heidelberg: Springer, 2012, no. 7700, oCLC: ocn828098376.

[6] O. V. Komogortsev and A. Karpov, "Automated classification and scoring of smooth pursuit eye movements in the presence of fixations and saccades," *Behavior Research Methods*, vol. 45, no. 1, pp. 203–215, Mar. 2013. [Online]. Available: http://link.springer.com/10.3758/s13428-012-0234-9

[7] A. T. Bahill, A. Brockenbrough, and B. T. Troost, "Variability and development of a normative data base for saccadic eye movements," *Investigative Ophthalmology & Visual Science*, vol. 21, no. 1 Pt 1, pp. 116–125, Jul. 1981.

[8] D. D. Salvucci and J. H. Goldberg, "Identifying fixations and saccades in eye-tracking protocols," in *Proceedings of the symposium on Eye tracking research & applications - ETRA '00*. Palm Beach Gardens, Florida, United States: ACM Press, 2000, pp. 71–78. [Online]. Available: http://portal.acm.org/citation.cfm?doid=355017.355028

[9] S. Wibirama, S. Murnani, N. A. Setiawan, and H. Nurlatifa, "A Survey of Event Detection Methods for Eye Movements Classification in Smooth-Pursuit-Based Interactive Applications," in *2020 International Symposium on Community-centric Systems (CcS)*. Hachioji, Tokyo, Japan: IEEE, Sep. 2020, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/9231384/

[10] T. Santini, W. Fuhl, T. Kübler, and E. Kasneci, "Bayesian identification of fixations, saccades, and smooth pursuits," in *Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research & Applications*. Charleston South Carolina: ACM, Mar. 2016, pp. 163–170. [Online]. Available: https://dl.acm.org/doi/10.1145/2857491.2857512

[11] J. Pekkanen and O. Lappi, "A new and general approach to signal denoising and eye movement classification based on segmented linear regression," *Scientific Reports*, vol. 7, no. 1, p. 17726, Dec. 2017. [Online]. Available: https://www.nature.com/articles/s41598-017-17983-x

[12] J. M. Bernardo and A. F. M. Smith, *Bayesian theory*, ser. Wiley series in probability and mathematical statistics. Chichester, Eng. ; New York: Wiley, 1994.

[13] M. Startsev, I. Agtzidis, and M. Dorr, "1D CNN with BLSTM for automated classification of fixations, saccades, and smooth pursuits," *Behavior Research Methods*, vol. 51, no. 2, pp. 556–572, Apr. 2019. [Online]. Available: http://link.springer.com/10.3758/s13428-018-1144-2

[14] R. Zemblys, "Eye-movement event detection meets machine learning," *BIOMEDICAL ENGINEERING 2016*, vol. 20, no. 1, 2017.

[15] S. Hoppe and A. Bulling, "End-to-End Eye Movement Detection Using Convolutional Neural Networks," 2016, publisher: arXiv Version Number: 1. [Online]. Available: https://arxiv.org/abs/1609.02452

[16] C. Elmadjian, C. Gonzales, and C. H. Morimoto, "Eye Movement Classification with Temporal Convolutional Networks," in *Pattern Recognition. ICPR International Workshops and Challenges*, A. Del Bimbo, R. Cucchiara, S. Sclaroff, G. M. Farinella, T. Mei, M. Bertini, H. J. Escalante, and R. Vezzani, Eds. Cham: Springer International Publishing, 2021, vol. 12663, pp. 390–404, series Title: Lecture Notes in Computer Science. [Online]. Available: http://link.springer.com/10.1007/978-3-030-68796-0$_2$8

[17] S. Bai, J. Z. Kolter, and V. Koltun, "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling," 2018, publisher: arXiv Version Number: 2. [Online]. Available: https://arxiv.org/abs/1803.01271

[18] M. Dorr, T. Martinetz, K. R. Gegenfurtner, and E. Barth, "Variability of eye movements when viewing dynamic natural scenes," *Journal of Vision*, vol. 10, no. 10, pp. 28–28, Aug. 2010. [Online]. Available: http://jov.arvojournals.org/Article.aspx?doi=10.1167/10.10.28