IMAGE-BASED TRANSIENT DETECTION FOR GOTO SKY SURVEY

TERRY CORTEZ

MASTER OF ENGINEERING
IN
COMPUTER ENGINEERING

SCHOOL OF INFORMATION TECHNOLOGY
MAE FAH LUANG UNIVERSITY
2022

IMAGE-BASED TRANSIENT DETECTION FOR GOTO SKY SURVEY

TERRY CORTEZ

THIS THESIS IS A PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF ENGINEERING
IN
COMPUTER ENGINEERING

SCHOOL OF INFORMATION TECHNOLOGY
MAE FAH LUANG UNIVERSITY
2022

IMAGE-BASED TRANSIENT DETECTION FOR GOTO SKY SURVEY

TERRY CORTEZ

THIS THESIS HAS BEEN APPROVED
TO BE A PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF ENGINEERING
IN
COMPUTER ENGINEERING
2022

EXAMINATION COMMITTEE

.....CHAIRPERSON
(Asst. Prof. Gp. Capt. Thongchai Yooyativong, Ph. D.)

.....ADVISOR
(Assoc. Prof. Natthakan Iam-on, Ph. D.)

.....CO-ADVISOR
(Assoc. Prof. Wg. Cdr. Tossapon Boongoen, Ph. D.)

.....EXTERNAL EXAMINER
(Assoc. Prof. Wg. Cdr. Thiansiri Luangwilai, Ph. D.)

ACKNOWLEDGEMENTS

This thesis was supported by Mae Fah Luang University. The university offered a research grant of 15,000 Thai Baht assisting my project. I would like to thank my collaborator, Dr. James Mullaney, from the University of Sheffield who provided insight and expertise that assisted the research. He also prepared the initial data and simulated samples for the experiment. In addition, I would like to thank my advisors, Assoc. Prof. Dr. Natthakan Iam-on and Assoc. Prof. Dr. Wg.Cdr. Tossapon Boongoen, who greatly provided guidance and information along the process.

Terry Cortez

Thesis Title	Image-based Transient Detection for GOTO Sky Survey
Author	Terry Cortez
Degree	Master of Engineering (Computer Engineering)
Advisor	Assoc. Prof. Natthakan Iam-on, Ph. D.
Co-Advisor	Assoc. Prof. Wg. Cdr. Tossapon Boongoen, Ph. D.

ABSTRACT

GOTO sky survey's objective is to study transient objects requiring big data to be processed regularly. The retrieved information used to be classified. However, the methods based on machine learning are ineffective due to the imbalanced data. Therefore, the objectives of this study are to provide an alternative method and evaluate the model with data with multiple noise levels. A concept of UNET's fully convolutional network is chosen from four possible classification methods. Using the simulated data generated from information collected as an input, outputs are generated based on different model setups. A way to achieve the results is also suggested at the end of the experiment in accordance with the aims of data users.

Keywords: Artificial Intelligence, Deep Learning, Change Detection, Astronomy, Sky Survey

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (continued)

Figure **Page**

LIST OF FIGURES (continued)

Figure **Page**

ABBREVIATION AND SYMBOL

CNN	Convolutional Neural Network
DIA	Difference Image Analysis
FCN	Fully Convolutional Network
FITS	Flexible Image Transport System
GOTO	Gravitational-wave Optical Transient Observer
KNN	K-Nearest Neighbor
ML	Machine Learning
PSF	Point Spread Function
RF	Random Forests
SVM	Support Vector Machine
TD	Transient Detection

CHAPTER 1

INTRODUCTION

1.1 Background

As one important mission of the Gravitational-wave Optical Transient Observer (GOTO) sky survey is for observing astronomical events, namely transient objects, photos of sky patches, showing on Figure 1.1, need to be taken daily in order to update onto a database and find what, when, and where transients occur. Then, these images are used in a computing process, done by either humans or machines. In other words, during this stage astronomers are using former procedures by exacting information from the images onto different tables, and using various Machine Learning (ML) techniques, such as Support Vector Machine (SVM) and Random Forests (RF), to analyze the data exacted. However, the recent results from this method were rather unpredictable, depending on whether data features were chosen and the fact that the number of items in observed classes was distinctively smaller than the number of items in unobserved classes—or a class imbalance problem [1-3].

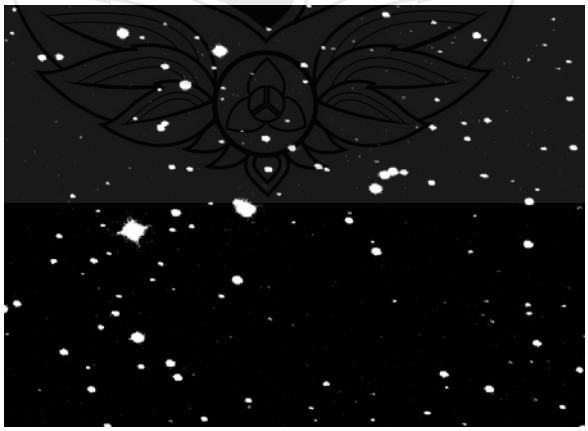

Figure 1.1 An example of a sky image

Even though there were additional methods such as Synthetic Minority Oversampling Technique or SMOTE to simulate the number of wanted classes and oversampling or undersampling to balance the number of items in between each class, the improved results were still unsatisfactory [2-4]. Unfortunately, while machine computing is more efficient than human computing which lacks an amount of accuracy and consumes lots of time, it requires a large amount of compatible data storage for the processed table data, and the mentioned techniques might not be able to vastly support other kinds of data where objects' features are different. In addition, an image-based alternative method, Difference Image Analysis (DIA)—in which input images are processed and compared with reference images using statistical formulae to find changes—gave insufficient accuracy [5-6]. At the same time, as more complex DIA methods resulted in improved accuracy, Convolutional Neural Networks (CNNs), which can analyze raw images directly, were more increasingly preferable due to an avoidability for data pre-processing and storing [7].

Therefore, this study will talk about semantic change detection or, in this case, Transient Detection (TD) using an unconventional method called Fully Convolutional Networks (FCNs)—which is another modification of CNN. The thesis was arranged as follows, 2. Literature Review showing the use of conventional methods on TD and the use of CNNs on a similar kind of work that leads to the use of FCNs in this work, 3. Methodology showing details on GOTO's data and illustration of a model for TD with FCNs, 4. Findings showing the algorithm's results and their implication, and 5. Summary. Additional information is also arranged in the Appendices at the end of the thesis.

1.2 Objectives

1.2.1 To develop an alternative framework for transient detection

1.2.2 To optimize and benchmark the framework algorithm for studying how its parameters and hyperparameters work

1.3 Scope of the Study

1.3.1 The data used in this study belongs to the dataset collected by GOTO for analytic purposes. This includes the simulation of transient objects. As real transients are very rare, it is better to create ones from astronomical collected data and inject them into the images. This data comes in the form of image arrays inside FITS containers in which they are retrieved and used.

1.3.2 One of the deep learning methods, called FCN, is expected to tackle the classification problem found in this kind of work. As being popular for precise segmentation of the results, the UNET model is considered as a foundation of this thesis. Apart from being modified to accept inputs in a specific size, the default configurations including a learning optimizer are derived from the UNET model.

1.3.3 For evaluating purposes, accuracy, recall, precision, and F1-score are calculated for each experiment. This study aims at maximizing both recall and precision at the same time to deal with the imbalance problem. It does not mean to maximize only one of them.

1.4 Project Plan

This study follows the machine learning development process in which each stage is performed accordingly.

1.4.1 Problem definition: recent procedures are examined in order to define a concise problem occurring in the GOTO project. This also defines which approach should be considered when reviewing past studies.

1.4.2 Solution identification: novel approaches are examined in order to find possible solutions for this problem. Possible solutions can be either ones that work with the GOTO data or ones that can be adapted.

1.4.3 Model determination: the best approach is selected from the range of possible solutions. This step is necessary for defining data acquisition.

1.4.4 Data collection: data is obtained from the GOTO pipeline (See 3.1, 3.2), inside FITS containers. This step includes generating transient objects with a quality consistent with the GOTO study.

1.4.5 Data modification: to develop a robust model, data should be left untouched as much as possible. Augmentation is exceptional as it helps improve the model accuracy (See 3.4); although it is canceled due to enough input data. Noise injection is also in this step.

1.4.6 Model evaluating and refining: in this step, evaluation metrics are calculated from the results. These values are then used to find the number of good models for optimization until the expected results are acquired.

1.4.7 Conclusion: the proper model, including its configurations, is acquired to prepare for the final report.

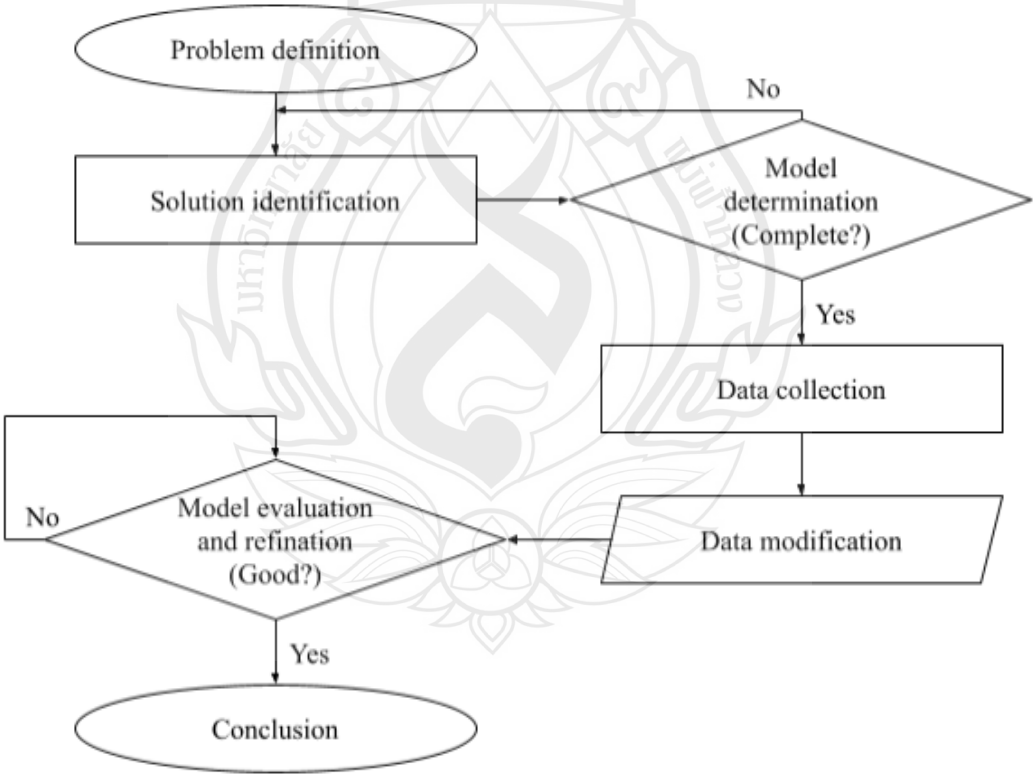

Figure 1.2 Experiment procedures

CHAPTER 2

LITERATURE REVIEW

2.1 Outcome of Machine Learning Procedures

As it was mentioned earlier, traditional ML for TD use features extracted from sky images taken from telescopes. Normally, these features generated by specific pipelines are related to brightness and amplitude, known as photometrics. According to [1], the accuracy rate was around 50 percent while class imbalance existed. Even though the number of items in each class was modified by several techniques, it tackled only the precision rate, leaving the recall rate to be around 80 percent [2]. This means there were a few transient sources left undetected.

Figure 2.1 Machine learning procedures

Apart from the problem between classes, proper feature selection is necessary. In [8], it was found that, within a list of features extracted, only few of them are meaningful—and could be differently based on several classes in the learning process. Unlike TD, galaxy detection and identification using the above method gave better accuracy [4]. In addition, for ML with galaxy images, morphological data could be used with photometric data to improve performance although only some combined features were used [9].

While it is more difficult to extract morphological data from transient sources—as they are just brightness-related events—image data might be another substitute by learning from sky images directly. With no feature required, CNN—which is image-based learning—can avoid some parts of pre-processing; at the same time, CNN is probably less affected by class imbalance [4]. Besides, [10] showed that giving augmentation and additional filters to images before learning could make the model robust. Unfortunately, novel approaches—using CNN to astronomically analyze sky images—were still concerned with only the galaxy but rarely transients.

2.2 Outcome of Difference Image Analysis Procedures

By comparing temporal images with reference images, the DIA can be an alternative method. However, although image-based methods do not require feature extraction and selection, they are unproductive not only for detecting transients but also for simply finding differences between images.

Traditional and popular methods are from Alard & Lupton [11] and Bramich [12] in which two images are matched by their Point Spread Functions (PSFs) or the determination for a degree of light spreading or blurring from a 3D object. They, unfortunately, produce noise or artifacts in case of mismatched PSFs, producing more difficulty in analyzing. Unlikely, another method proceeding toward the statistical experiment provided by Zackay [6] can solve this problem. This method generates less artifacts than the former, so that transients can be detected by defining proper thresholds.

Figure 2.2 Difference image analysis procedures

ML can also take the role of selecting transients from DIA results, in regard to the above methods. According to [7], K-Nearest Neighbor (KNN), SVM, and RF were used to identify transients on images generated from each DIA method, in comparison. Even though the results were better than applying solely DIA and thresholds, they were inconsistent and depending on which sources were real or simulated, showing limited use on large scale data.

Although DIA works with image contents directly overlooking feature-based learning, it misses lots of transient objects. In other words, it has low recall rates. Moreover, DIA occasionally generates artifacts looking like real sources on its results, and it still requires image pre-processing in order to be working.

2.3 Convolutional Neural Networks and Their Limitations

Although CNNs are non-feature-based learning, features—representing lines, shadows, or edges—are automatically extracted along the process [13-14]. These features are used to find dissimilarity between two images during change detection. However, ordinary CNNs do not provide localization for interested objects. It is needed for proper algorithms or threshold definition in order to localize the objects [15]. While some CNNs' implementations might be able to localize the objects, they created only boundary boxes—ignoring objects' shape or line segment [10].

Figure 2.3 Convolutional neural network procedures

At the same time, CNNs require a large scale of data for training instances, in order to create an accurate or even robust model. Unfortunately, in the case of studies in which information is rather new and not abundant, CNNs cannot show their full potential. While some work uses pre-trained models, called transfer learning—to skip

training steps—others might use augmentation to increase the number of instances; for example, see [10,16].

2.4 Process of Fully Convolutional Network on Similar Kinds of Work

Unlike traditional CNNs, FCNs stop the process before the final stage called multilayer perceptron or fully connected layer, leaving a raw feature map untouched as a result. Some of them improve the result by extending a section, called deconvolution or decoding—in contrast with the former convolution or encoding part in CNNs. The results can have the same size as input images, with a number of channels representing classes [13-18]. This requires less training data as the learning process is more efficient [13,15-16]. While some work—such as [13]—provided a method to cut input images into smaller patches and combine the output patches together, others—for example, [17]—provided more flexible models to be able to accept input images in any size.

Figure 2.4 Fully convolutional network procedures

These feature maps generated not only cover almost all foreground objects—ignoring background objects—but also include semantic segmentation, showing precise areas where the objects are located. Moreover, FCNs can learn to ignore background objects and noise when comparing images taken in different circumstances, like angles, viewpoints, or brightness [18]. According to [14], with filter metric and threshold implementation, FCN still did well for images with very distinct viewpoints. They are also able to work with high resolution images [18].

As [17] stated, for multi-temporal remote sensing images, a common FCN was working with 82, 75, and 77 percent for precision, recall, and F1-score, respectively.

Other variances of FCNs have more potential. A UNET model equips FCN with skip connections—keeping the results in each convolution layer as residues to be concatenated and convoluted again in the latter deconvolution layers—increasing its performance [15]. At the same time, Siamese networks, in which two identical FCNs occur for each image, give more accuracy with lower computing time [14,16].

Nevertheless, most change detection studies with deep learning are working with remote sensing images and synthetic-aperture radar or SAR images. For sky surveys, it is considered as a new method for object detection.

Table 2.1 Comparison between four approaches

	ML	**DIA**	**CNN**	**FCN**
Input type	Table	Image	Image	Image
Preprocessing	Required	Required	Not required	Not required
Feature selection	Manual[1]	None/Manual[2]	Automatic	Automatic
Localization	No	Yes	No	Yes
Output type	Class table	Class map	Class table	Class map

Note [1] This is usually manual unless the features are selected by another algorithm beforehand.

[2] This is manual if the postprocess is implemented with ML.

CHAPTER 3

METHODOLOGY

3.1 FITS Containers and Their Drawbacks

When astronomers talk about analyzing sky images, they do not interact with raw images directly. In fact, the sky images captured each day are normally kept in an image archive called the Flexible Image Transport System (FITS), which comprises a header (i.e., metadata and descriptions)—in key-value format—and image contents—in array format. In other words, telescopes produce image data that will be packed into FITS, and FITS files are used by different pipelines to extract feature-rich table data for analyzing purposes. However, FITS comes with its downsides.

Although FITS has backward compatibility and it is compatible with multiple machines or systems, or being a transport medium, it requires its own language for reading and extracting data—even though Python has a library to bypass this step. This is considered a bottleneck as the larger data in FITS is, the slower an operation becomes [19-20].

Due to the downsides, there are many FITS replacements, such as Advance Scientific Data Format or ASDF which does better [21]. However, the GOTO sky survey keeps data in FITS containers and the data acquired from GOTO comes in this format.

3.2 Shape of Data from the GOTO Sky Survey

As mentioned previously, raw astronomical data, collected by telescopes, is contained in FITS files—in a form of images or 2D arrays with additional metadata—from which features or data attributes are traditionally extracted in order to be analyzed. The extraction process depends on a pipeline or groups of code that differently retrieves

user-requested data from related FITS files. While GOTO's objective is to study transient objects, its pipeline mainly considers photometric data, for instance, brightness information—or magnitude—with geolocations and survey timestamps. This data is kept in database as three tables as shown on Figure 3.1: classified by roles, coadd containing reference transient data which is infrequently updated, frc containing regular survey which is increasing every day, and frc_md containing additional metadata such as telescope configurations and timestamps. Besides, the tables have relationships as shown on Figure 3.2.

Figure 3.1 GOTO learning procedures at the time of study

Following the state of the art, a source or each row in the database that belongs to the same objectid must be rearranged by date in order to create a light curve with its magnitude—as light curves can identify whether the objects are transient or persistent. However, this method requires lots of time and storage space as we have already tried to support the idea by setting up a network ecosystem called Hadoop [22]. The system accumulated workable processing units or computers together working on a shared space. Unfortunately, querying and retrieving data on this system are still slow, and the ecosystem lacks enough community support. Therefore, we diverted to the raw images themselves, and conducted the study on the data inside FITS files instead of their hidden photometric features. The image arrays inside the FITS files have a float64 format and can be considered as raw data in this experiment.

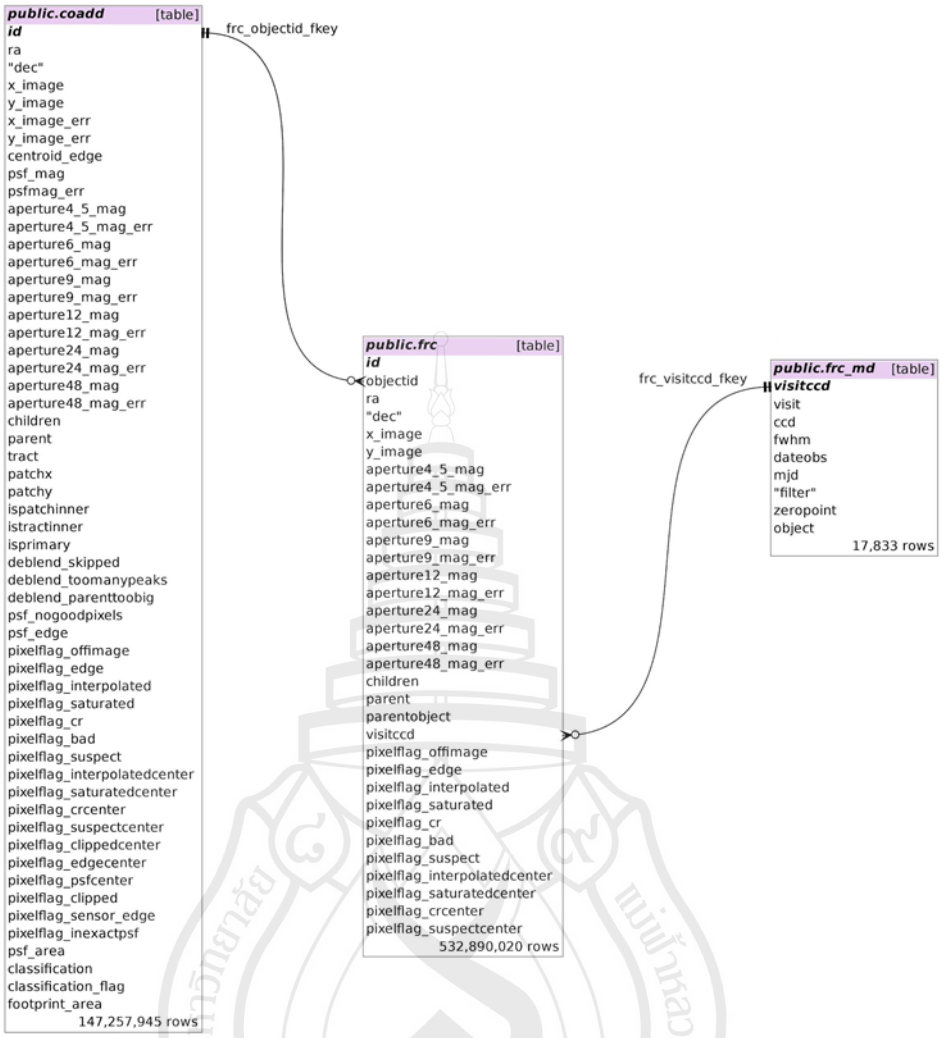

Figure 3.2 GOTO data storage at the time of study

3.3 Simulation of Transient Objects

Because the transient objects that interest GOTO are infrequently appearing, the ones in the experiment are fully simulated. While they are newly generated, they are from the GOTO's collected information, containing the same shape or pixel values as the real objects. The transients generated, then, must be injected or merged into each temporal image, replacing values on the original images. The generation includes ground truths or mask maps telling the injected location that is necessary for the training

and evaluating steps. These preprocessed datasets are from the researcher, Dr. James Mullaney, who is working on the GOTO project.

Figure 3.3 Input images: (left) original image, (center) transient-injected image, and (right) ground truth

Four sky images of the size, 5632x5632, are chosen and require little preparation. This step is not a pre-processing of images, but only makes them suitable as an input for the algorithm. Additionally, apart from transient simulation, environmental noises are also generated to make the situation more practical since external factors can make two images taken at the same location look different (See 3.6).

3.4 Modification of Data

As this work tries to solve robust problems of TD, data modification has to be done as little as possible: the data would only be adapted to be compatible with FCN. In other words, the FCN in this study does not mean to accept an input in multiple shapes but requires a specific one. In this case the input is an array of the size, 256x256, nominalized in float32 format. The reason behind the input size is that during the training step, too large arrays lead to an out-of-memory error. Therefore, giving the algorithm batches of smaller arrays is better. The input array has two layers which respectively contain data from temporal images: a pair of images taken at a different

time. In this experiment temporal images are represented by the same image arrays with different environmental noises.

Figure 3.4 Algorithm in general

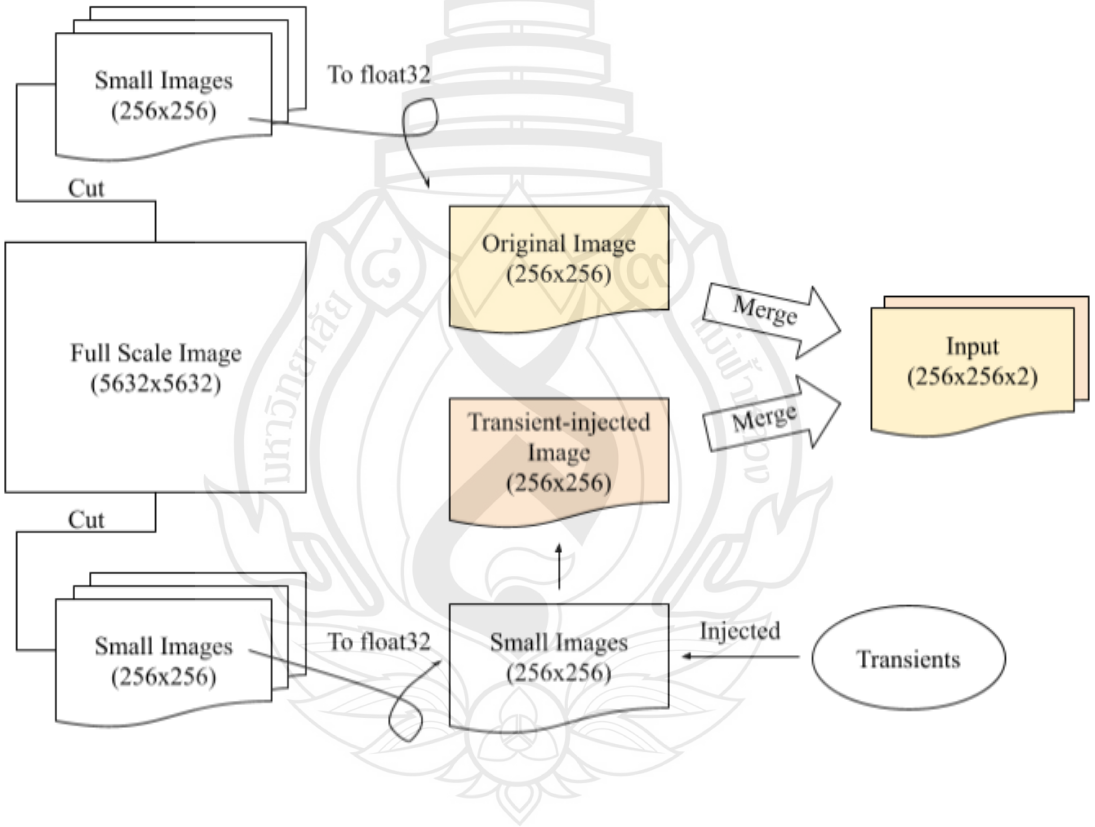

Figure 3.5 Preparing input

Images are initially extracted from FITS files and put into Numpy save files, with ".npy" format, as starter data since a Numpy array can be a substitute for a raw image. Additionally, during the training process a ground truth map is required to let the algorithm adapt learning weights by itself. In other words, a ground truth is what

we show the algorithm what we want as a result while an input is what we tell the algorithm what the future or practical input looks like. This ground truth is a float32 array with only one and zero. One represents the desired objects and zero means the background. Ground truths have the same 5632x5632 size as the original images, and also need to be cut into small batches, having the size of 256x256 as the input. Please note that in a ground truth array, one transient is represented as a 7x7 square regardless of the real transient size. This means after training the algorithm will also understand that the output map should contain 7x7 squares as a result.

To increase the training efficiency and make the model robust, augmentation can be used in the next step, according to the novel studies. By rotating each image by a 90-degree angle and flipped both vertically and horizontally. Additionally, more data can also be created by shifting original images by a little bit before cutting into smaller ones as the algorithm considers them as new images. However, to reduce the amount of time used in training, augmentation is canceled due to an already enough data.

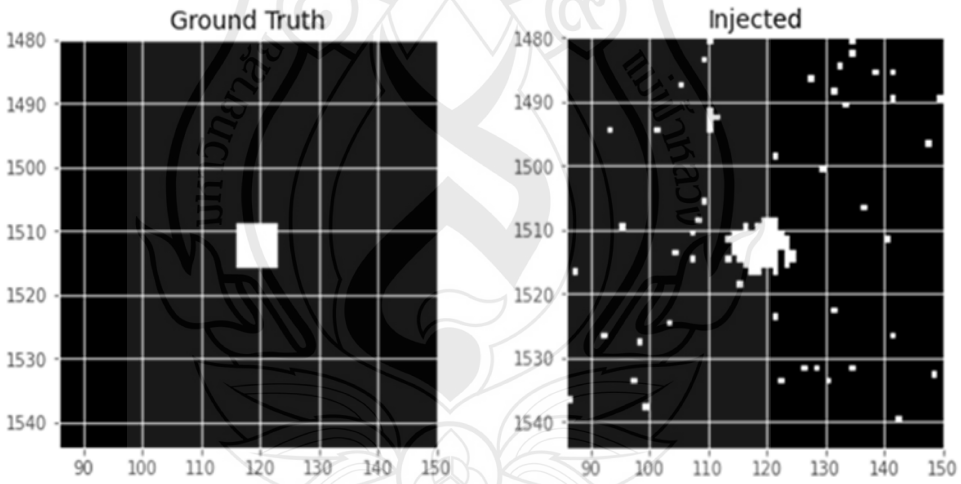

Figure 3.6 Ground truth and transient-injected image (magnified)

After preparing the input and ground truth for four sky images, the numbers of final samples are 113,246,208 and 12,582,912 for training and validating steps respectively. Additionally, another sky image is used afterward to evaluate the model according to the metrics mentioned in the following. This contains 31,719,424 samples.

However, please consider that the numbers of samples are not the transient objects themselves but only pixel values. In other words, one transient might cover more than a single sample. The end of the next chapter will consider how to analyze the result back to a transient object. (See 4.5).

3.5 Procedures for Analyzing the Data with Fully Convolutional Network

CNNs' structure is a layer-like process comprising mainly convolution one and statistics-related components, such as pooling parameters—to change maps' size and activation functions—to modify specific values and prevent overfitting. In FCNs, there is an additional component called a deconvolutional layer, which is a transposed convolutional layer, helping recreate full size feature maps—making semantic segmentation possible.

This study uses a model derived from the UNET architecture in which modified an FCN structure with a skip connection; for more information see [23]. Skip connections are convolution results from each layer retained in order to be concatenated with the deconvolution results of the same levels, represented by gray curved arrows on Figure 3.7. It is believed that re-convoluting the concatenated results again after the deconvolution makes the segmentation more accurate. The initial setup is retrieved in accordance with the UNET model which convolutes using 3x3 filters and 2 strides, followed by a Rectified Linear Unit (ReLU) activation function and a 2x2 max pooling—on the encoding side. The process is repeated until the data has very deep channels. On the decoding side, there is a deconvolutional layer made from transposing the former convolutional layer with the same configurations, followed by convoluting again after concatenation with a skip connection. This is repeated until the result is back to the same size as the beginning—256x256—with 64 channels as a result map. Then, the Sigmoid function is applied to generate the final result with only a single channel left. Note that the code for the FCN model in this study is modified to be used with inputs in any size, as long as they can be half divided until the end of the convoluting process. For the experiment 256x256 is the initial size, the convolution depth is 4, and

the output filter for the first convolution has 64 channels. An additional hyperparameter such as dropout rate—which helps prevent the model overfitting—is set to 0.5.

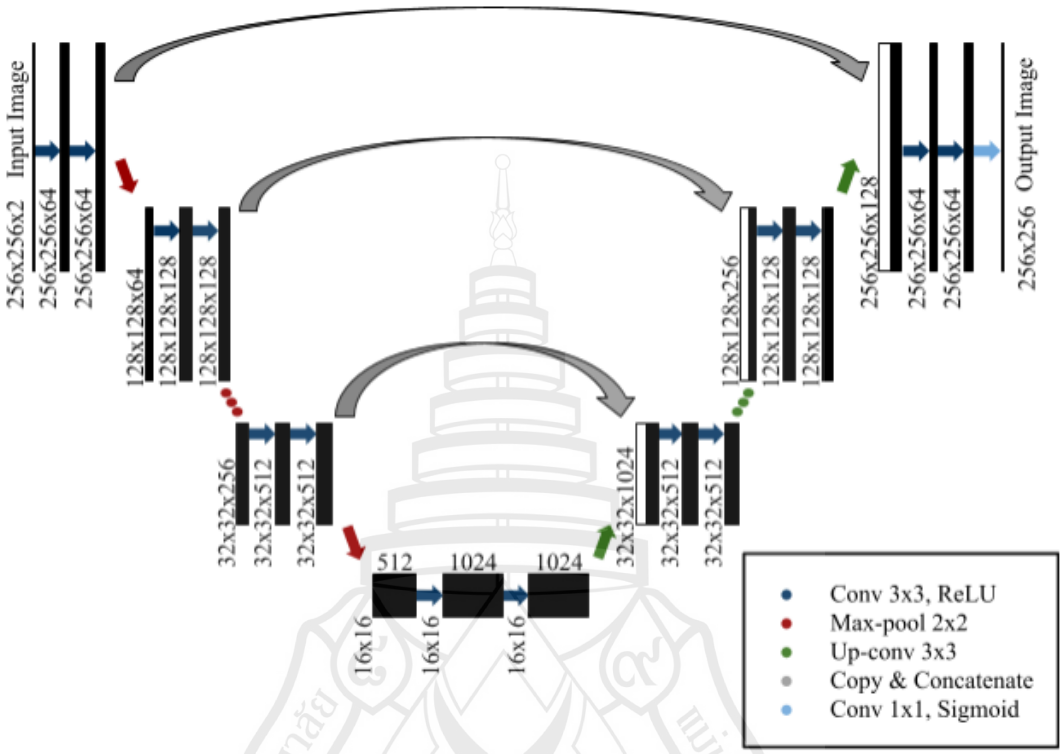

Figure 3.7 Fully convolutional network in the model

For the model optimization, cross entropy is chosen as a loss function, and Adam is chosen as a learning optimization algorithm. As a smaller batch increases the effectiveness of the model with the cost of learning time, this study divides inputs into mini batches with the sizes of 32, 16, and 8 before repeating the experiment again and again in order to test which batch size is the most appropriate. The initial number of iterations is 30—however, it can be stopped early if the validation loss decreases lower than 0.01 for 10 iterations, as a sign of no improvement.

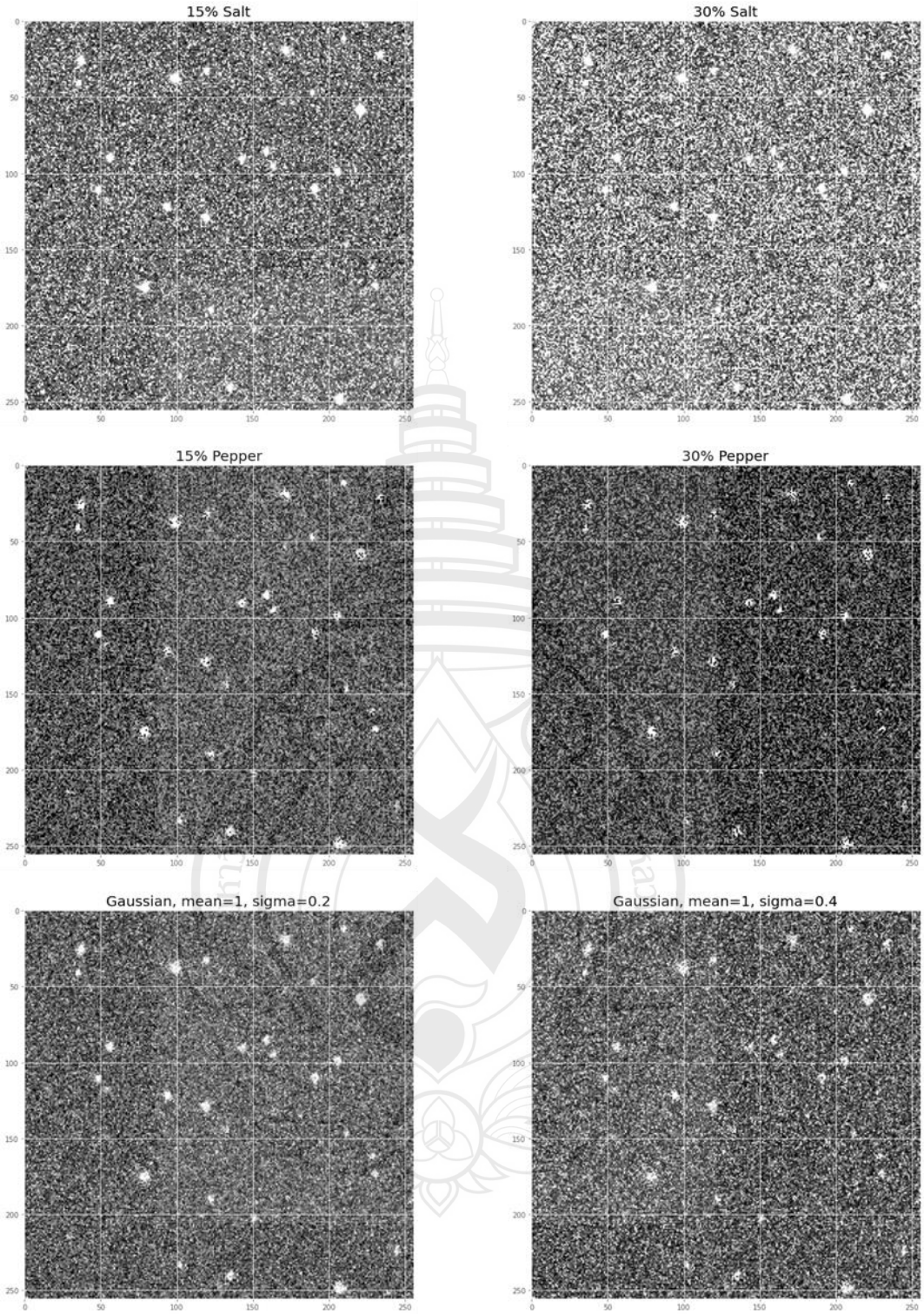

Figure 3.8 Noises: (top) salt, (middle) pepper, and (bottom) Gaussian

3.6 Noises

As a pair of images that is used as an input belongs to the same images, one is original and one is injected with transients, the algorithm does not face any noise. However, in practice, sky images captured from a telescope are changing every time according to the environment variables, such as light levels. Thus, three kinds of noise—salt, pepper, and Gaussian—are considered as a representation of difficulty in detection. salt represents the white spots by proportionally and randomly replacing values in an array by one. On the other hand, pepper represents the black spots by proportionally and randomly replacing values in an array by zero. In this experiment 15 and 30 percent of each salt and pepper are used. Additionally, images can be distorted by multiplying their pixel values by the Gaussian distribution values. In this experiment the standard deviations of 0.2 and 0.4 are chosen with the mean of 1.

3.7 Post Processing

After the full feature map is acquired, it is still unusable as its values are non-binary. It is necessary to define a threshold to separate these values into zero and one. As the result of the Sigmoid function ranges between zero and one, 0.25, 0.50, 0.75 thresholds are selected for the experiment. The 0.50 threshold is used commonly whereas the 0.25 and 0.75 thresholds are implemented for a study purpose (See Figure 3.9).

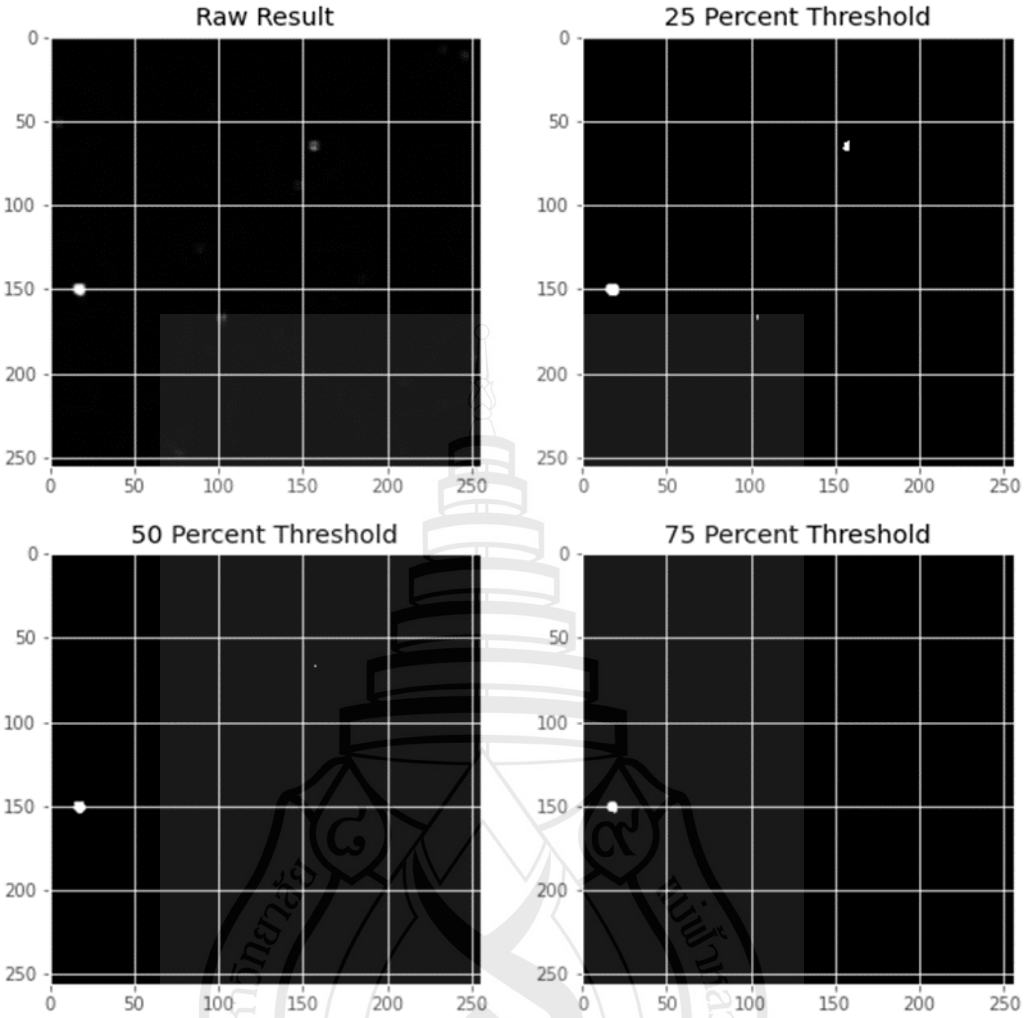

Figure 3.9 Threshold levels

3.8 Evaluation Metrics

It is important to measure the model's effectiveness. By comparing the map with the ground truth, a confusion metric is generated, containing four values: True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). Then, these values are used to calculate four measures: accuracy, recall, precision, an F1-score. Accuracy as shown on Figure 3.9 is for measuring general performance. Due to the imbalanced class, this measure is hardly useful. As shown on Figures 3.10 and 3.11, recall tells how many transients are found by the model while precision tells how many

transients are correctly answered by it. If the model has high precision but low recall, it is still ineffective as there are some transients left behind. Lastly, F1-score comes in handy since it is based on both precision and recall: being shown on Figure 3.12.

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

Figure 3.10 Formula for accuracy

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

Figure 3.11 Formula for recall

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

Figure 3.12 Formula for precision

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Figure 3.13 Formula for F1-score

CHAPTER 4

RESULTS

This chapter presents the experimental results and their implications. According to the model setup—with hyperparameters left untouched—there are 81 possible models generated from three types of noise combinations (with three levels each) and three batch sizes. Each of the models is evaluated with an unused sky image. The evaluation image can also be modified to have noises. According to the three noise combinations, there are 27 possible ways to modify this image. Therefore, with 81 models and 27 noise patterns (See Appendix B for examples), 2,187 different results exist. Each result map is filtered by three threshold levels before being compared with the ground truth. As a result, there are 6,561 results to be analyzed.

4.1 Time Usage

Although time usage is discussed in this section, please note that it has small significance on algorithm efficiency. Due to the fact that time spent on computing depends on GPU or TPU, it is different devices from devices, and there might be a little time lag when starting GPU or TPU.

For learning time, the overall usage is around five minutes for the batch size of eight and three minutes for the batch sizes of 16 and 32. This is consistent with the fact that feeding large batch sizes into the algorithm can speed up the process regardless of accuracy, which might be loose. Besides, there is no clear difference between batch sizes of 16 and 32. In Figure 4.1, the horizontal axis represents all 81 models. The model combinations and their description are on Appendix C.

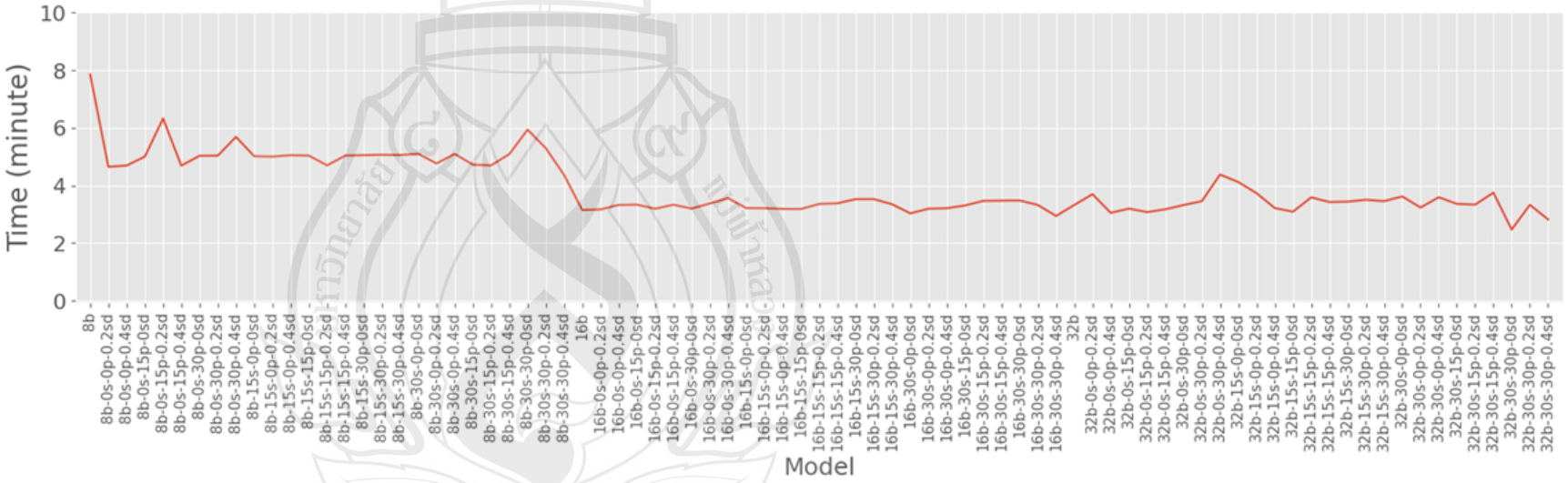

Figure 4.1 Time usage for learning

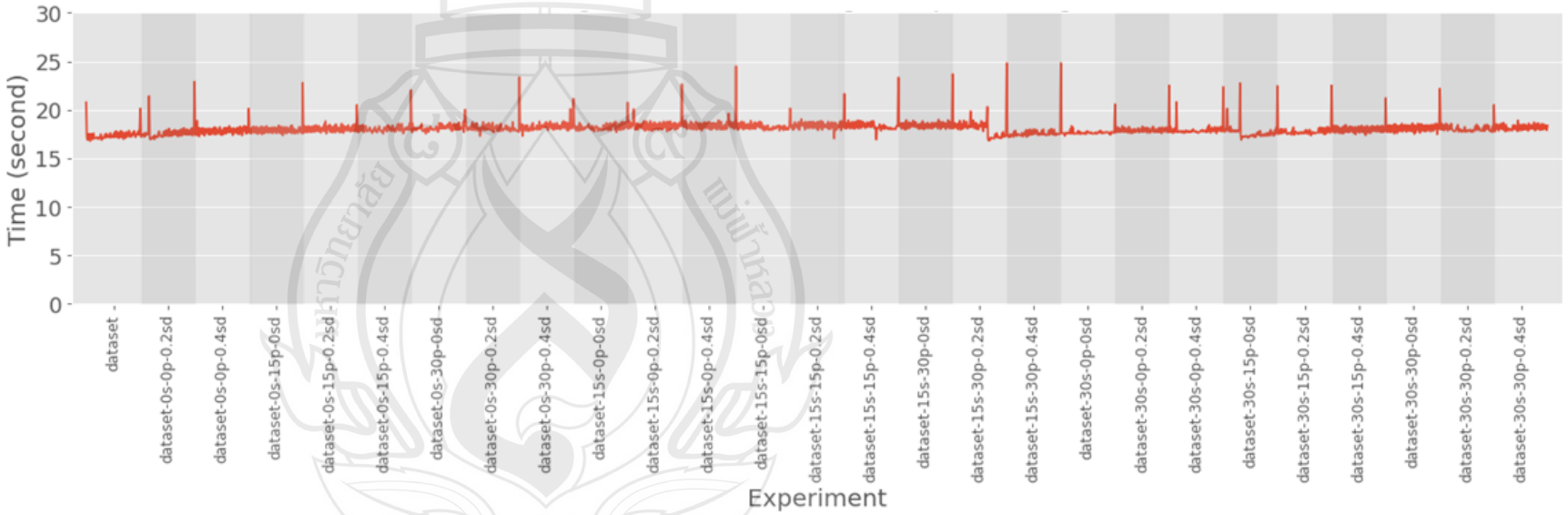

Figure 4.2 Time usage for predicting

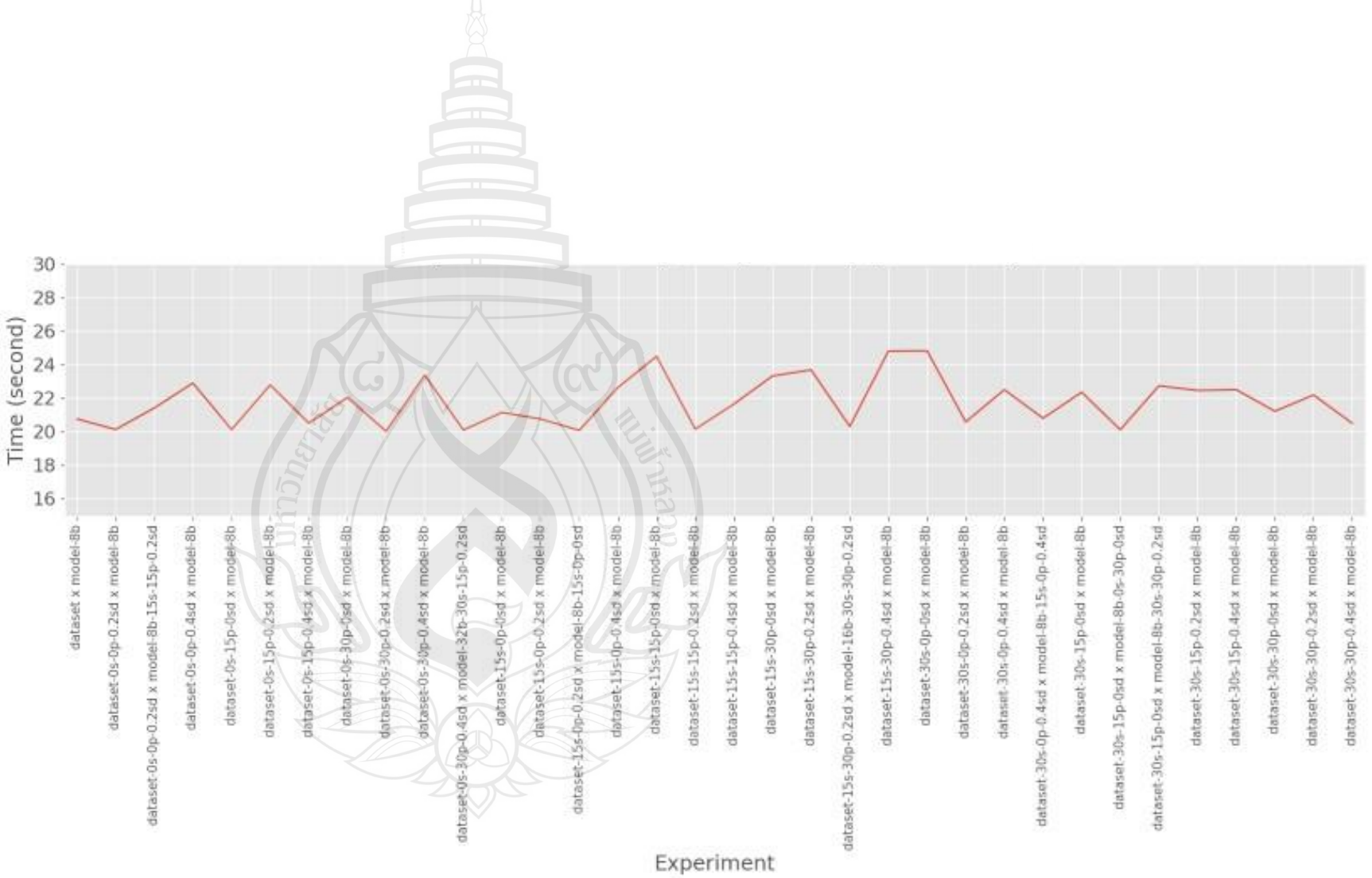

Figure 4.3 Time usage for predicting (Time>=20)

For predicting time, the horizontal axis represents all 2,187 predictions. The labels are also divided into 27 groups while each group has 81 outcomes (See Appendix D for a reference). The overall usage is between 15 and 20 seconds with very few runs using a little bit more. We found that all predictions of any dataset with a 8b model—a model learning with eight batches of data with no noise—takes longer. However, it cannot confirm the correlation between datasets, models, and time usage as there are a few combinations that take longer time either; however, with the same amount of data we can conclude that its complexity has no effect on the time spent as there is only 1.5 percent of predictions taking more time.

4.2 Performance

First, let us look at, dataset, the dataset without noise which is the easiest one for TD. Although the evaluating data remains unchanged, the metrics change, depending on which models are used to predict. For this dataset, the best model is, 8b, the one that learns from data without noise. Its F1 score reached 96.77 with 95.16 percent of recall and 98.44 percent of precision. It is followed by 8b-0s-15p-0.2sd as the recall slightly increases with the cost of precision drops. As shown on Figure 4.4, there is only a very small difference between result maps and ground truths. Interestingly, there are three models that learn from data without noise but only the model feeding the smallest batch size can achieve these rates. Even though the precisions are good, their recall rates are rather lower, dropping their F1 scores.

Table 4.1 Evaluation metrics from the ‘dataset’ dataset

Order	Model	Accuracy	Recall	Precision	F1
1	8b	99.98	95.16	98.44	96.77
2	8b-0s-0p-0.2sd	99.83	45.06	94.51	61.02
3	8b-0s-0p-0.4sd	99.73	7.77	99.4	14.41
4	8b-0s-15p-0sd	99.93	81.86	94.11	87.56
5	8b-0s-15p-0.2sd	99.97	95.66	95.11	95.38

Table 4.1 (continued)

Order	Model	Accuracy	Recall	Precision	F1
6	8b-0s-15p-0.4sd	99.83	44.64	98.93	61.52
7	8b-0s-30p-0sd	99.94	83.52	97.53	89.98
8	8b-0s-30p-0.2sd	99.92	78.97	92.69	85.28
9	8b-0s-30p-0.4sd	99.75	14.76	95.77	25.57
10	8b-15s-0p-0sd	99.93	78.91	98.09	87.46
11	8b-15s-0p-0.2sd	99.83	44.4	97.1	60.93
12	8b-15s-0p-0.4sd	99.86	54.14	98.24	69.81
13	8b-15s-15p-0sd	65.26	62.35	0.53	1.05
14	8b-15s-15p-0.2sd	84.61	76.68	1.46	2.87
15	8b-15s-15p-0.4sd	99.82	42.44	96.03	58.87
16	8b-15s-30p-0sd	99.75	14.4	99.17	25.15
17	8b-15s-30p-0.2sd	99.86	55.67	94.17	69.97
18	8b-15s-30p-0.4sd	99.75	14.59	98.49	25.42
19	8b-30s-0p-0sd	99.91	75.91	93.71	83.88
20	8b-30s-0p-0.2sd	99.91	75.37	92.89	83.22
21	8b-30s-0p-0.4sd	99.83	47.49	88.7	61.86
22	8b-30s-15p-0sd	99.79	30.88	98.09	46.98
23	8b-30s-15p-0.2sd	99.8	41.81	81.13	55.18
24	8b-30s-15p-0.4sd	99.83	47.05	92.28	62.32
25	8b-30s-30p-0sd	99.66	46.97	43.49	45.16
26	8b-30s-30p-0.2sd	99.72	4.7	91.16	8.94
27	8b-30s-30p-0.4sd	99.7	0.0	nan	nan
28	16b	99.86	52.22	98.83	68.33
29	16b-0s-0p-0.2sd	99.88	62.15	98.34	76.17
30	16b-0s-0p-0.4sd	99.64	34.18	38.66	36.28
31	16b-0s-15p-0sd	99.96	89.78	96.57	93.05
32	16b-0s-15p-0.2sd	99.7	0.07	100.0	0.15
33	16b-0s-15p-0.4sd	99.83	45.64	95.73	61.81

Table 4.1 (continued)

Order	Model	Accuracy	Recall	Precision	F1
34	16b-0s-30p-0sd	99.7	0.21	100.0	0.43
35	16b-0s-30p-0.2sd	99.72	5.99	96.79	11.28
36	16b-0s-30p-0.4sd	99.89	63.51	97.89	77.04
37	16b-15s-0p-0sd	99.74	11.36	97.77	20.35
38	16b-15s-0p-0.2sd	99.74	13.99	99.53	24.52
39	16b-15s-0p-0.4sd	99.7	0.0	nan	nan
40	16b-15s-15p-0sd	99.78	25.94	97.05	40.94
41	16b-15s-15p-0.2sd	99.7	0.1	80.33	0.21
42	16b-15s-15p-0.4sd	99.79	49.16	71.6	58.3
43	16b-15s-30p-0sd	99.71	1.4	98.14	2.76
44	16b-15s-30p-0.2sd	99.87	58.76	97.43	73.31
45	16b-15s-30p-0.4sd	99.69	0.32	5.1	0.61
46	16b-30s-0p-0sd	99.7	0.63	99.66	1.25
47	16b-30s-0p-0.2sd	95.35	31.84	2.08	3.91
48	16b-30s-0p-0.4sd	99.82	43.34	93.78	59.29
49	16b-30s-15p-0sd	99.81	37.54	96.45	54.05
50	16b-30s-15p-0.2sd	99.68	70.35	46.97	56.33
51	16b-30s-15p-0.4sd	99.81	35.66	98.75	52.39
52	16b-30s-30p-0sd	99.7	4.42	43.21	8.02
53	16b-30s-30p-0.2sd	98.75	20.2	5.6	8.77
54	16b-30s-30p-0.4sd	99.7	0.0	nan	nan
55	32b	99.86	54.01	96.8	69.33
56	32b-0s-0p-0.2sd	99.95	90.96	91.52	91.24
57	32b-0s-0p-0.4sd	99.71	1.24	99.23	2.44
58	32b-0s-15p-0sd	99.71	1.63	98.27	3.2
59	32b-0s-15p-0.2sd	99.7	0.15	100.0	0.29
60	32b-0s-15p-0.4sd	99.8	34.73	92.58	50.52
61	32b-0s-30p-0sd	99.7	0.12	100.0	0.23

Table 4.1 (continued)

Order	Model	Accuracy	Recall	Precision	F1
62	32b-0s-30p-0.2sd	81.69	63.31	1.02	2.01
63	32b-0s-30p-0.4sd	99.19	79.04	23.87	36.66
64	32b-15s-0p-0sd	99.97	93.23	96.27	94.73
65	32b-15s-0p-0.2sd	99.91	73.16	96.91	83.38
66	32b-15s-0p-0.4sd	99.74	14.13	97.34	24.68
67	32b-15s-15p-0sd	99.6	0.02	0.05	0.03
68	32b-15s-15p-0.2sd	99.76	20.69	95.3	34.0
69	32b-15s-15p-0.4sd	99.87	56.89	97.46	71.85
70	32b-15s-30p-0sd	99.7	0.1	31.4	0.19
71	32b-15s-30p-0.2sd	99.77	24.12	98.13	38.73
72	32b-15s-30p-0.4sd	19.93	91.66	0.34	0.68
73	32b-30s-0p-0sd	99.92	74.75	98.71	85.08
74	32b-30s-0p-0.2sd	99.89	63.21	98.88	77.12
75	32b-30s-0p-0.4sd	99.78	26.36	93.22	41.1
76	32b-30s-15p-0sd	98.53	76.46	13.96	23.61
77	32b-30s-15p-0.2sd	99.77	24.33	94.96	38.74
78	32b-30s-15p-0.4sd	99.71	3.05	98.86	5.92
79	32b-30s-30p-0sd	99.7	0.0	nan	nan
80	32b-30s-30p-0.2sd	99.7	0.07	100.0	0.14
81	32b-30s-30p-0.4sd	99.7	0.0	nan	nan

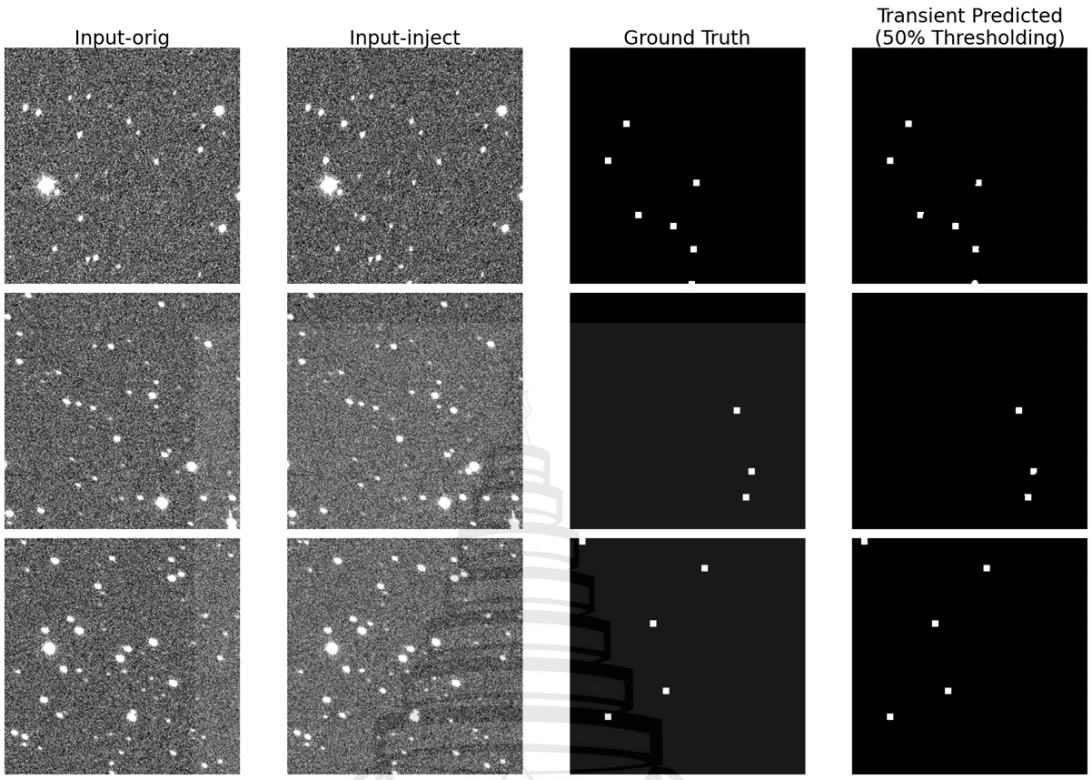

Figure 4.4 Feature maps result from ‘dataset x model-8b’

On the other hand, let us inspect the data with noise. In this case, dataset-30s-30p-0.4sd, the one with extreme noise combinations, should be the most difficult test subject for this study.

Table 4.2 Evaluation metrics from the ‘dataset-30s-30p-0.4sd’ dataset

Order	Model	Accuracy	Recall	Precision	F1
1	8b	86.41	45.32	0.99	1.94
2	8b-0s-0p-0.2sd	99.7	0.28	47.52	0.55
3	8b-0s-0p-0.4sd	99.7	1.22	55.28	2.39
4	8b-0s-15p-0sd	99.12	5.83	2.82	3.8
5	8b-0s-15p-0.2sd	96.31	36.41	2.99	5.53
6	8b-0s-15p-0.4sd	99.01	27.4	9.49	14.09

Table 4.2 (continued)

Order	Model	Accuracy	Recall	Precision	F1
7	8b-0s-30p-0sd	99.68	1.19	11.83	2.16
8	8b-0s-30p-0.2sd	99.32	13.61	8.79	10.68
9	8b-0s-30p-0.4sd	98.48	21.74	4.76	7.81
10	8b-15s-0p-0sd	99.7	0.51	22.45	1.0
11	8b-15s-0p-0.2sd	99.7	0.04	9.74	0.08
12	8b-15s-0p-0.4sd	99.69	0.64	13.15	1.21
13	8b-15s-15p-0sd	99.68	8.68	35.44	13.95
14	8b-15s-15p-0.2sd	99.7	0.21	21.85	0.41
15	8b-15s-15p-0.4sd	99.44	28.23	19.29	22.92
16	8b-15s-30p-0sd	99.57	20.21	23.91	21.91
17	8b-15s-30p-0.2sd	99.56	28.82	26.96	27.86
18	8b-15s-30p-0.4sd	99.71	9.47	58.74	16.31
19	8b-30s-0p-0sd	99.7	0.0	0.0	nan
20	8b-30s-0p-0.2sd	99.7	0.15	37.73	0.3
21	8b-30s-0p-0.4sd	99.7	0.02	12.98	0.04
22	8b-30s-15p-0sd	99.7	0.0	nan	nan
23	8b-30s-15p-0.2sd	99.7	0.0	nan	nan
24	8b-30s-15p-0.4sd	99.64	1.62	6.35	2.58
25	8b-30s-30p-0sd	99.69	1.33	22.23	2.52
26	8b-30s-30p-0.2sd	99.7	0.77	39.36	1.5
27	8b-30s-30p-0.4sd	99.7	0.0	nan	nan
28	16b	99.39	8.65	7.02	7.75
29	16b-0s-0p-0.2sd	99.7	2.15	40.76	4.09
30	16b-0s-0p-0.4sd	99.6	10.62	18.44	13.48
31	16b-0s-15p-0sd	96.52	37.79	3.29	6.05
32	16b-0s-15p-0.2sd	99.7	2.01	52.95	3.87
33	16b-0s-15p-0.4sd	99.62	18.87	28.16	22.6
34	16b-0s-30p-0sd	99.7	2.77	50.83	5.25

Table 4.2 (continued)

Order	Model	Accuracy	Recall	Precision	F1
35	16b-0s-30p-0.2sd	98.63	21.7	5.38	8.62
36	16b-0s-30p-0.4sd	99.63	5.98	16.35	8.76
37	16b-15s-0p-0sd	99.7	0.0	nan	nan
38	16b-15s-0p-0.2sd	99.7	0.0	0.0	nan
39	16b-15s-0p-0.4sd	99.7	0.02	43.4	0.05
40	16b-15s-15p-0sd	99.7	0.02	65.71	0.05
41	16b-15s-15p-0.2sd	99.44	28.65	19.58	23.26
42	16b-15s-15p-0.4sd	99.67	10.94	32.13	16.33
43	16b-15s-30p-0sd	99.72	5.36	82.07	10.06
44	16b-15s-30p-0.2sd	99.73	18.07	64.19	28.2
45	16b-15s-30p-0.4sd	99.72	13.99	65.32	23.05
46	16b-30s-0p-0sd	99.7	0.12	26.35	0.24
47	16b-30s-0p-0.2sd	99.7	0.01	77.78	0.01
48	16b-30s-0p-0.4sd	99.7	0.01	11.93	0.03
49	16b-30s-15p-0sd	99.7	0.0	nan	nan
50	16b-30s-15p-0.2sd	99.7	1.06	63.27	2.09
51	16b-30s-15p-0.4sd	99.69	0.81	10.53	1.5
52	16b-30s-30p-0sd	99.7	0.1	4.81	0.19
53	16b-30s-30p-0.2sd	99.69	4.36	30.04	7.61
54	16b-30s-30p-0.4sd	99.7	0.0	nan	nan
55	32b	98.13	15.76	2.81	4.77
56	32b-0s-0p-0.2sd	97.28	26.63	3.06	5.5
57	32b-0s-0p-0.4sd	99.7	0.08	63.25	0.16
58	32b-0s-15p-0sd	99.71	2.34	63.8	4.52
59	32b-0s-15p-0.2sd	99.71	2.39	60.87	4.61
60	32b-0s-15p-0.4sd	99.47	22.27	18.3	20.09
61	32b-0s-30p-0sd	99.7	0.02	9.6	0.04
62	32b-0s-30p-0.2sd	99.7	0.23	54.14	0.46

Table 4.2 (continued)

Order	Model	Accuracy	Recall	Precision	F1
63	32b-0s-30p-0.4sd	95.59	32.44	2.24	4.19
64	32b-15s-0p-0sd	99.69	0.86	18.54	1.65
65	32b-15s-0p-0.2sd	99.68	1.6	12.68	2.85
66	32b-15s-0p-0.4sd	99.7	0.02	53.33	0.03
67	32b-15s-15p-0sd	99.7	0.05	70.42	0.11
68	32b-15s-15p-0.2sd	99.68	9.24	33.81	14.51
69	32b-15s-15p-0.4sd	99.71	3.86	65.81	7.29
70	32b-15s-30p-0sd	99.7	0.08	92.77	0.16
71	32b-15s-30p-0.2sd	99.71	7.65	59.92	13.56
72	32b-15s-30p-0.4sd	99.71	2.21	85.51	4.3
73	32b-30s-0p-0sd	99.7	0.03	11.79	0.06
74	32b-30s-0p-0.2sd	99.7	0.0	nan	nan
75	32b-30s-0p-0.4sd	99.7	0.09	27.18	0.17
76	32b-30s-15p-0sd	99.7	0.24	43.13	0.47
77	32b-30s-15p-0.2sd	99.65	5.27	19.36	8.29
78	32b-30s-15p-0.4sd	99.4	16.01	11.83	13.61
79	32b-30s-30p-0sd	99.7	0.0	nan	nan
80	32b-30s-30p-0.2sd	99.7	0.03	6.36	0.07
81	32b-30s-30p-0.4sd	99.7	0.0	nan	nan

The metrics are quite low as expected as the transients and background are almost blended. We have to choose only recall or precision as there is no metric which is high at the same time. The best F1 score belongs to 16b-15s-30p-0.2sd, which has 18.07 percent recall and 64.19 percent precision.

As shown on Figure 4.5, both two input images are very ambiguous. There are few correct answers on the feature map. The false positive answers also exist as shown on the first feature map (top).

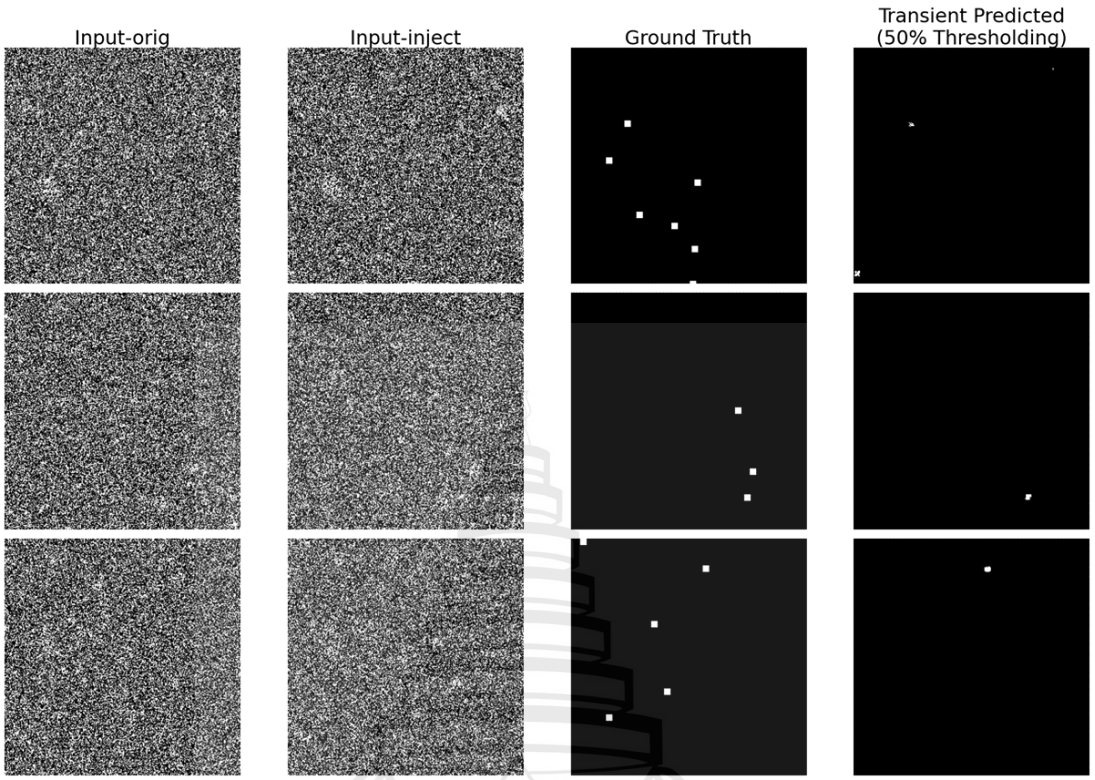

Figure 4.5 Feature maps result from the ‘dataset-30s-30p-0.4sd x model-16b-15s-30p-0.2sd’

We have already examined both the easiest and hardest test subjects. For the remaining runs, there are metric plots for each dataset on the Appendix F. At the same time, let us see the model side to see how a model is affected by noise. The first one is the model, 8b, the one that learns with data with no noise. Because the information fed to the model is less complicated, the model works only for less complicated cases. We can see that the precision drops immensely from the start of the dataset-0s-0p-0.4sd while the recall gradually drops little by little.

On the other hand, too many noises during learning (i.e., 8b-30s-30p-0.4sd, 16b-30s-30p-0.4sd, and 32b-30s-30p-0.4sd) lead to losing all transients as the algorithm cannot learn from the data. Recall rates are zero in every run while precision rates and F1 are undefined. Figure 4.7 compares confusion matrices between one of the -30s-30p-0.4sd models and 8b model, predicting dataset-0s-0p-0.2sd dataset.

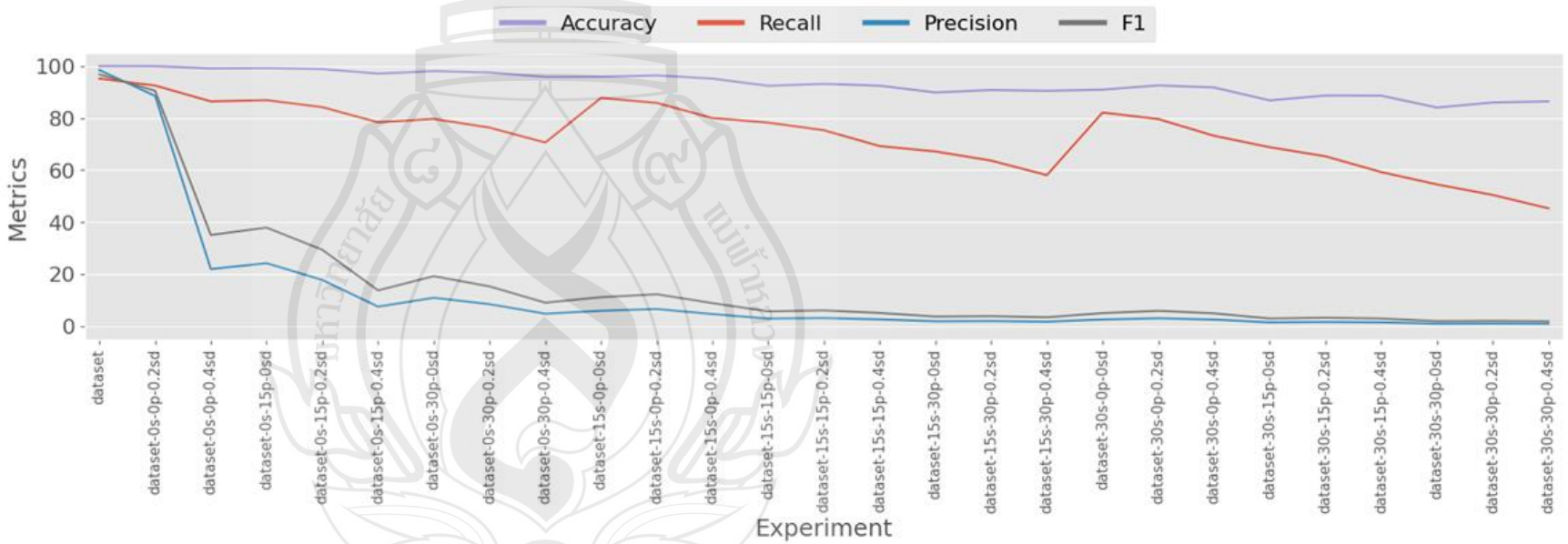

Figure 4.6 Model 8b performance

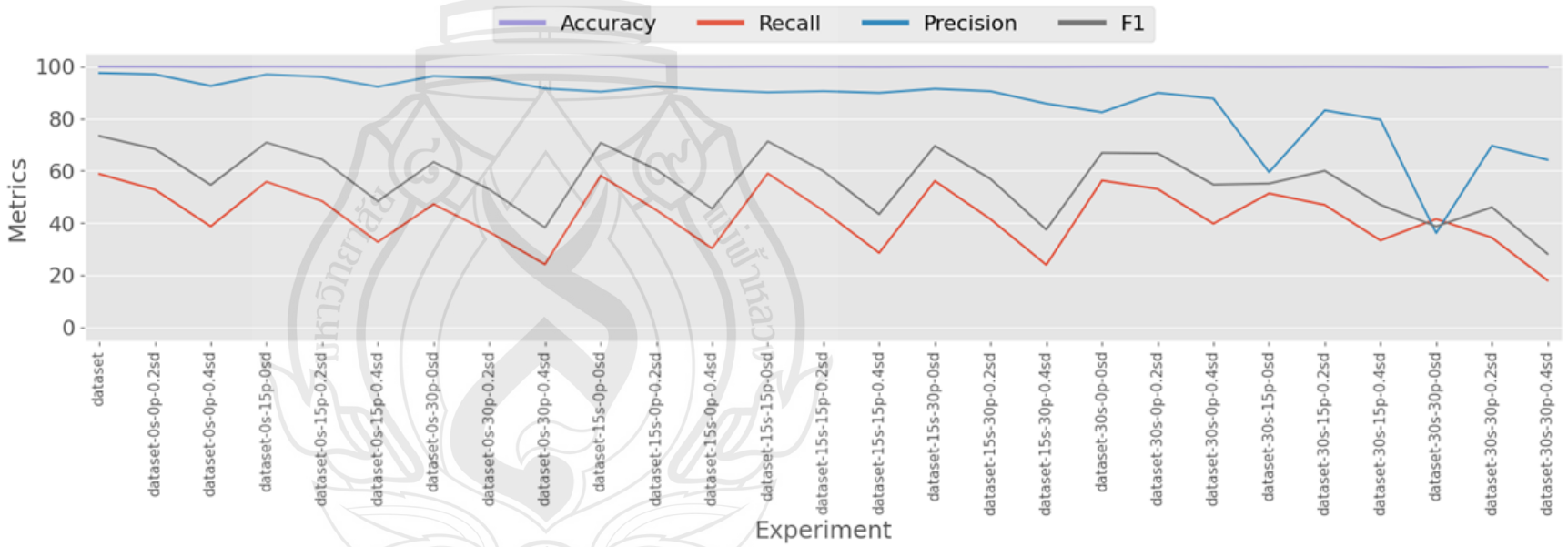

Figure 4.7 Model 16b-15s-30p-0.2sd performance

Figure 4.8 Confusion matrix comparison for dataset-0s-0p-0.2sd prediction (left) 8b model and (right) any-30s-30p-0.4sd model

Lastly, let us see the 16b-15s-30p-0.2sd model mentioned earlier, which is the best model for the most difficult dataset. Figure 4.8 shows that in each run, an average recall rate is around 40-45 with rather high precision rates. The precisions finally drop when the test subject is full of noises. For more information, see Table 4.3 and Appendix E as a reference.

Table 4.3 Evaluation metrics from the prediction of the '16b-15s-30p-0.2sd' model

Order	Dataset	Accuracy	Recall	Precision	F1
1	dataset	99.87	58.76	97.43	73.31
2	dataset-0s-0p-0.2sd	99.85	52.77	96.89	68.33
3	dataset-0s-0p-0.4sd	99.81	38.72	92.46	54.58
4	dataset-0s-15p-0sd	99.86	55.82	96.86	70.83
5	dataset-0s-15p-0.2sd	99.84	48.39	95.93	64.33

Table 4.3 (continued)

Order	Dataset	Accuracy	Recall	Precision	F1
6	dataset-0s-15p-0.4sd	99.79	32.77	92.14	48.35
7	dataset-0s-30p-0sd	99.84	47.22	96.24	63.36
8	dataset-0s-30p-0.2sd	99.81	36.59	95.45	52.9
9	dataset-0s-30p-0.4sd	99.77	24.19	91.47	38.26
10	dataset-15s-0p-0sd	99.86	58.12	90.26	70.71
11	dataset-15s-0p-0.2sd	99.83	44.95	92.29	60.46
12	dataset-15s-0p-0.4sd	99.78	30.32	90.92	45.48
13	dataset-15s-15p-0sd	99.86	59.0	90.01	71.28
14	dataset-15s-15p-0.2sd	99.82	44.74	90.45	59.87
15	dataset-15s-15p-0.4sd	99.78	28.59	89.8	43.37
16	dataset-15s-30p-0sd	99.85	56.11	91.36	69.53
17	dataset-15s-30p-0.2sd	99.81	41.48	90.42	56.87
18	dataset-15s-30p-0.4sd	99.76	24.0	85.69	37.49
19	dataset-30s-0p-0sd	99.83	56.28	82.39	66.88
20	dataset-30s-0p-0.2sd	99.84	53.05	89.83	66.7
21	dataset-30s-0p-0.4sd	99.8	39.77	87.64	54.72
22	dataset-30s-15p-0sd	99.75	51.35	59.49	55.12
23	dataset-30s-15p-0.2sd	99.81	46.94	83.12	60.0
24	dataset-30s-15p-0.4sd	99.78	33.36	79.54	47.01
25	dataset-30s-30p-0sd	99.61	41.56	36.21	38.7
26	dataset-30s-30p-0.2sd	99.76	34.42	69.56	46.06
27	dataset-30s-30p-0.4sd	99.73	18.07	64.19	28.2

Moreover, it is found that the 16b-15s-30p-0.2sd model gives good results equally between each test subject. Namely, with 81 model types, there is a tradeoff between the results. The model, initially, is rather accurate with low noise data and inaccurate with high noise data. As we train the model with more noises, its ability to detect low noise data drops. Therefore, with average performance, it is interesting to

optimize the 16b-15s-30p-0.2sd model for answering the research questions, at the same time, in comparison with the basic model 8b.

4.3 Significance of Threshold Levels

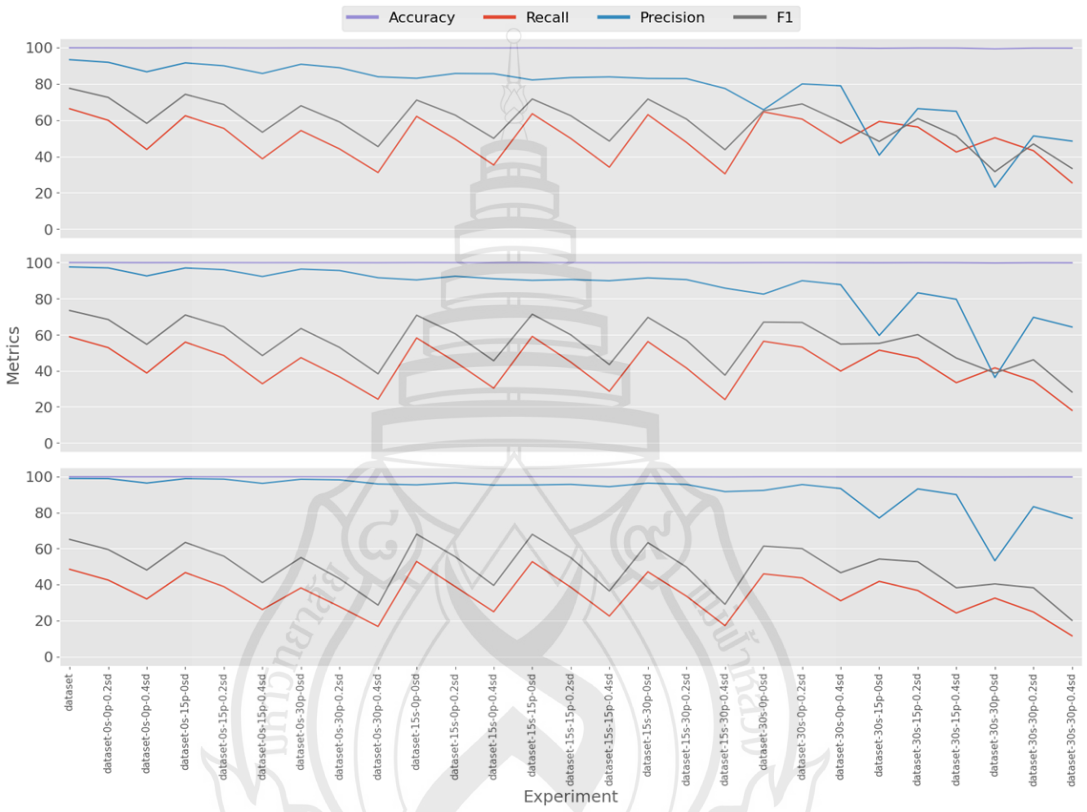

Figure 4.9 Model 16b-15s-30p-0.2sd performance by thresholds: (top) 25%, (middle) 50%, and (bottom) 75%

Threshold levels also have an influence on the detection. As a model output has values ranging between zero and one, depending on the confidence of a result, it is necessary to determine whether they are zero (i.e., backgrounds) or one (i.e., transients). Normally, a 50 percent threshold is used to separate them equally. The lower threshold increases the number of transients detected with a cost of a higher false positive rate. On the other hand, the higher threshold decreases the number of transients detected with a cost of a higher false negative rate.

Figure 4.9 shows the difference between three threshold levels, 25, 50, and 75 percent. According to the expectation, the low threshold raises recall rates and drop precision rates for each test subject whereas the high threshold drops recall rates and raise precision rates in the same manner. This also applies to the basic model like 8b as shown in Figure 4.10.

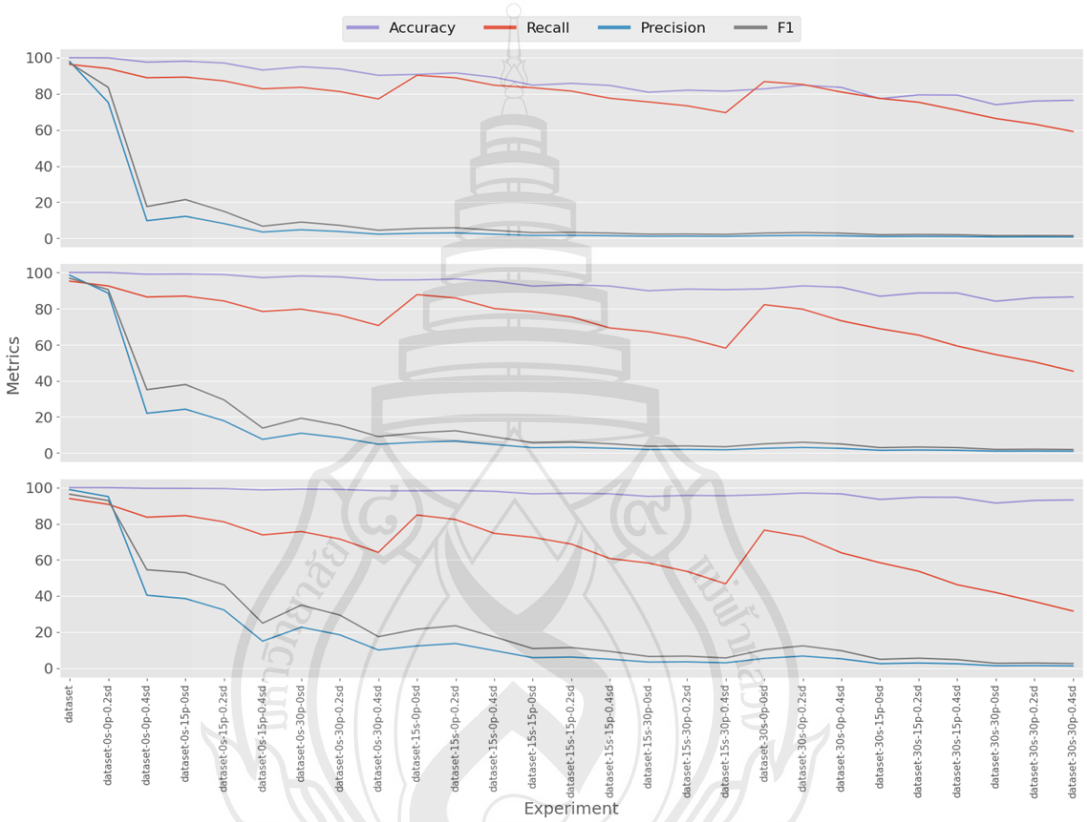

Figure 4.10 Model 8b performance by thresholds: (top) 25%, (middle) 50%, and (bottom) 75%

This, additionally, implies the tradeoff between precision and recall. For our model, 16b-15s-30p-0.2sd, it is better to use a 25 percent threshold to increase overall recall rates and F1 scores with the small cost of precisions.

4.4 Significance of Training Sizes

Normally, the idea of a skip connection in UNET lets the model require less learning input, so we can train the algorithm in a short time with no need for an augmentation. To examine this concept, all the experiments are conducted again with the different amounts of learning data. The amount of evaluating data is still the same.

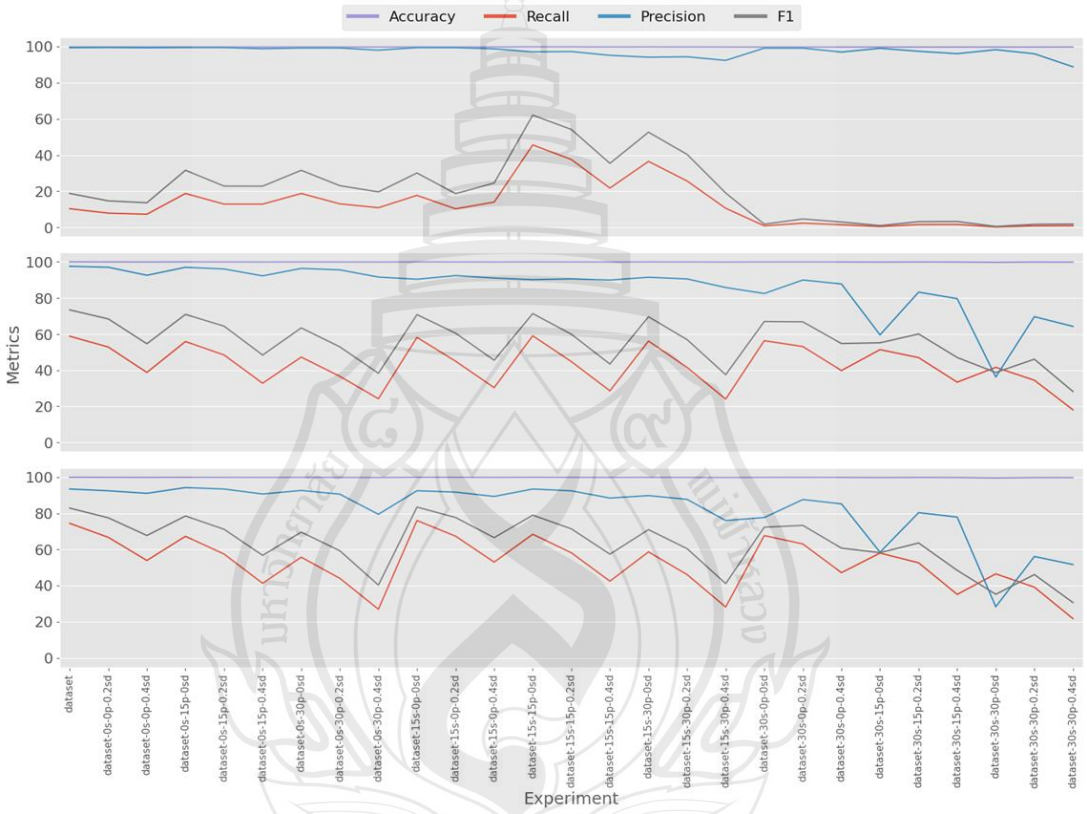

Figure 4.11 Model 16b-15s-30p-0.2sd performance by training sizes: (top) 1/2, (middle) default, and (bottom) x2

For 16b-15s-30p-0.2sd, less amount of training data leads to low recall rates whereas more amount of training data both increases recall rates and decreases precision rates by a little bit. For 8b, both two additional tests improve precision rates with the cost of recall rates. These, however, are not considered as an improvement as the recall rates are also important for detecting the objects.

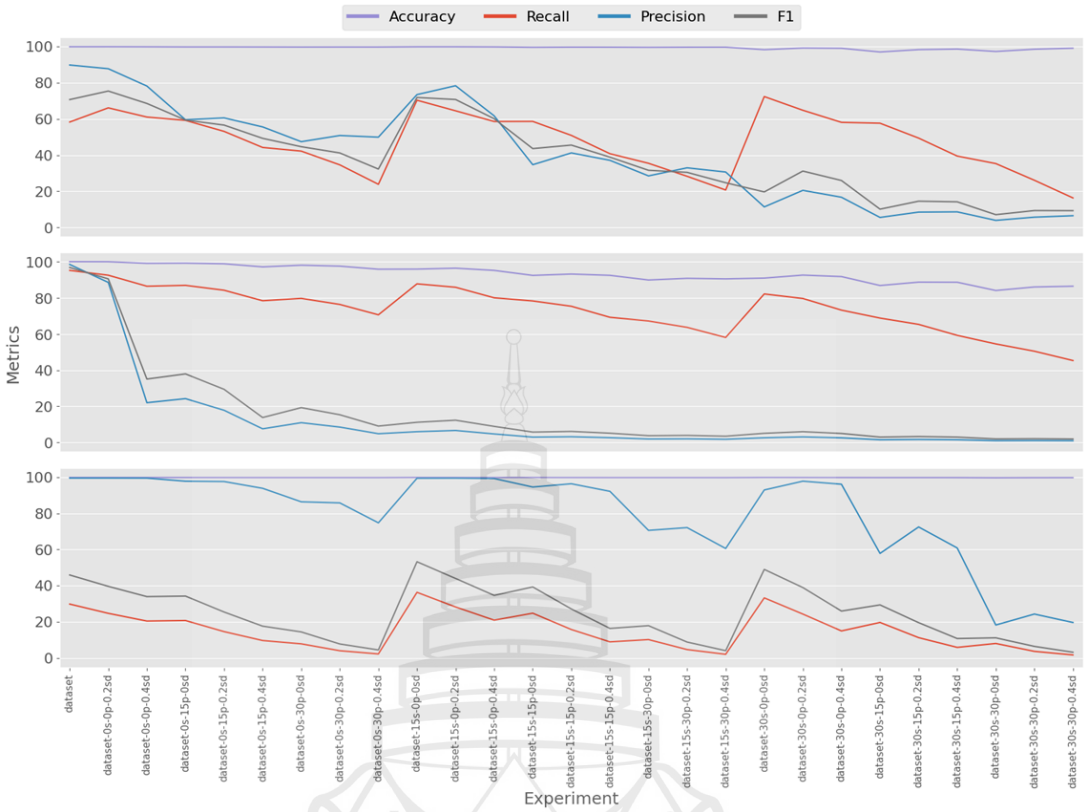

Figure 4.12 Model 8b performance by training sizes: (top) 1/2, (middle) default, and (bottom) x2

This implies that an increase in learning data can modify levels of recall and precision. Like the threshold level, there is a tradeoff between them, so we keep the default amount of learning data for our model.

4.5 Significance of Features

Although CNN does not require manual feature selection, the features are automatically selected behind the model. These features contain information like line and shade. In previous results, the initial number of features was 64. The number of features detected are doubled at every convolution layer. In order to check the importance of this hyperparameter, the experiments are redone with the initial feature set to 32 and 128 instead.

Figure 4.13 Model 16b-15s-30p-0.2sd performance by numbers of features: (top) 32, (middle) 64, and (bottom) 128

Figure 4.13 shows that, with a lower number of features, the model is worsened while the precision seems to be a little bit improved with a higher number of features. However, the predictions of test objects with less noise are worsened. This might be a sign of overfitting as the model tries to focus more on more complicated data.

The effect of changing the number of features is different for the basic model, 8b. As shown on Figure 4.14, both 32 and 128 feature sizes give more precision in exchange for recall. The model is slightly better for predicting more noise datasets. Maybe, the results of this parameter depend on the complexity of data fed to the model.

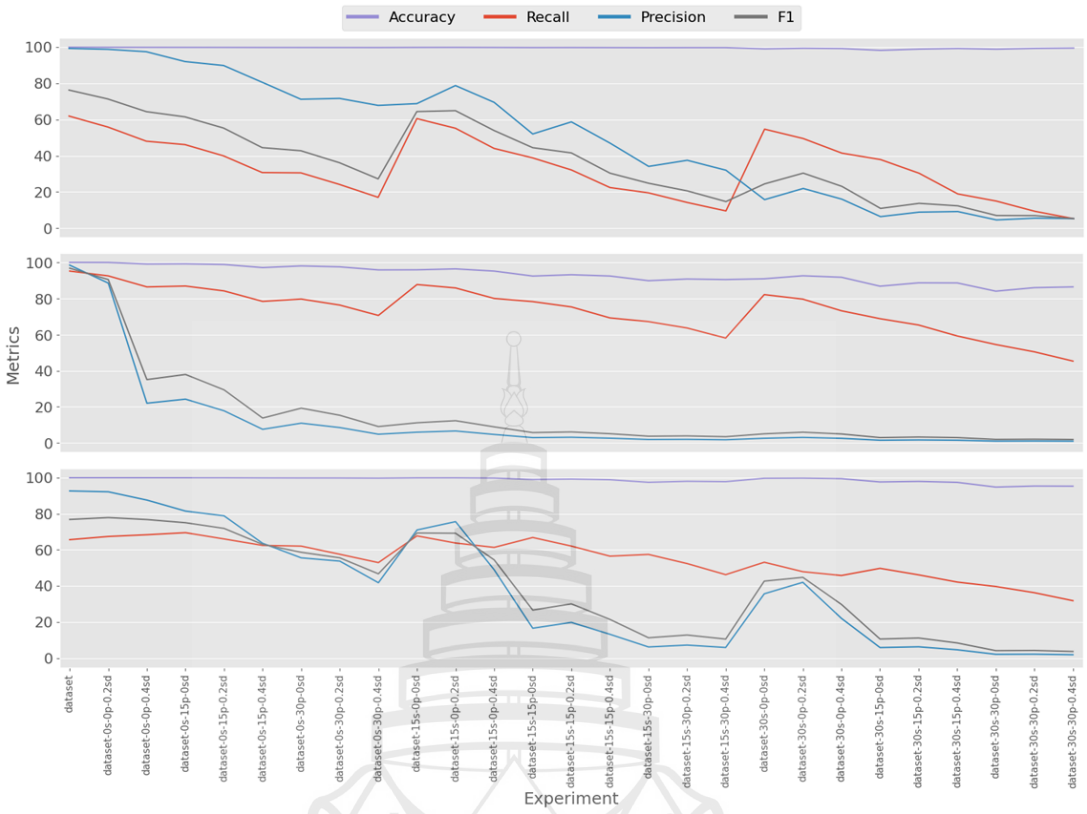

Figure 4.14 Model 8b performance by numbers of features: (top) 32, (middle) 64, and (bottom) 128

4.6 Significance of Network Deepness

According to the novel studies, the depth of four should be sufficient for a convolutional process. However, it is expected that less deep networks can save more time while more deep networks can improve accuracy, so the next experiments we conduct are feeding the same datasets into a network with the depth of three and five.

The results from deepness change are quite similar to the ones from feature size change. For the 16b-15s-30p-0.2sd model, on Figure 4.15, the results are rather indifferent between three and four depths. However, the result is worse with five depths. This could be a sign of overfitting as the model is almost inaccurate only for datasets without noise.

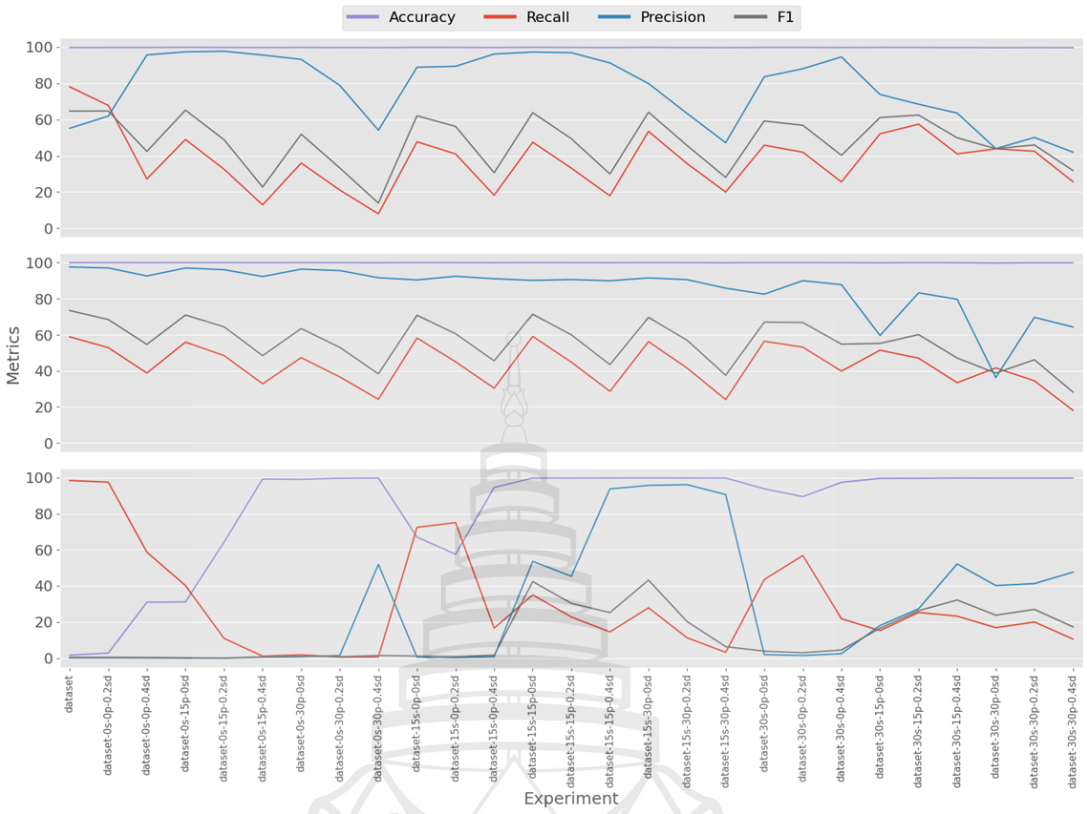

Figure 4.15 Model 16b-15s-30p-0.2sd performance by depths: (top) 3, (middle) 4, and (bottom) 5

At the same manner as feature size, the 8b models have more precision in exchange for recall for the depth numbers of three and five, as shown on Figure 4.16. This can help the basic model achieve higher precision and recall for the test subjects with more noise.

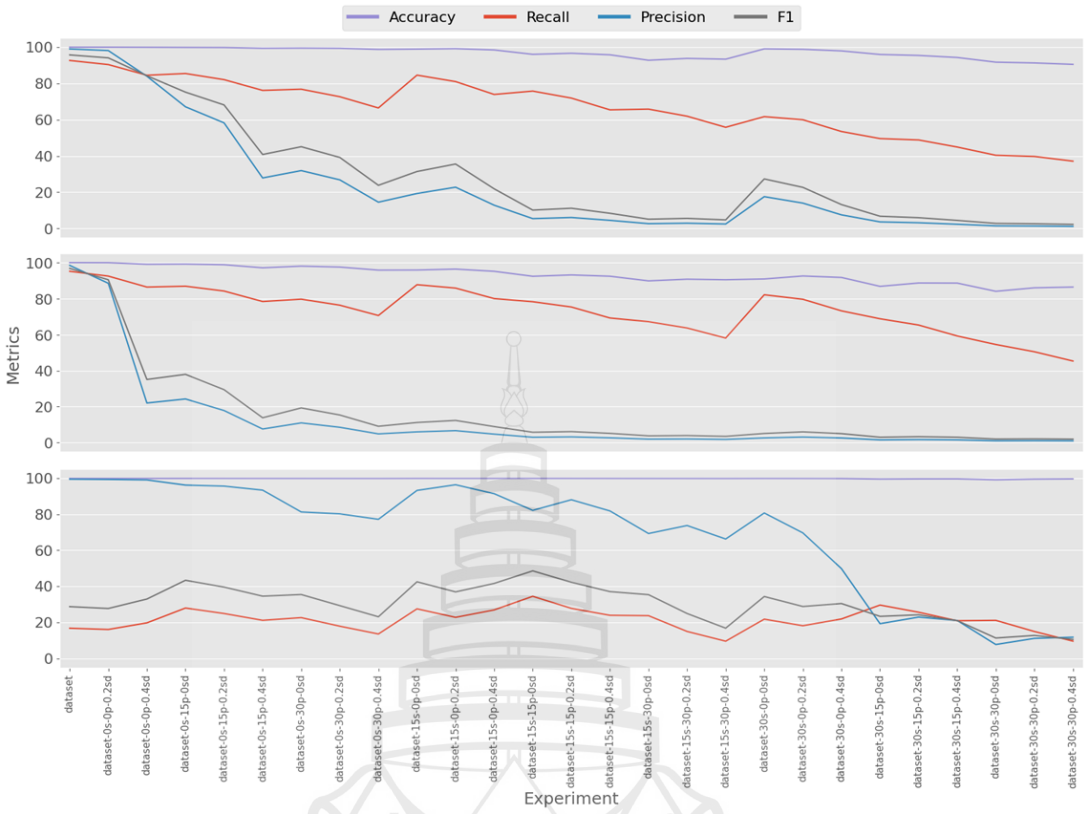

Figure 4.16 Model 8b performance by depths: (top) 3, (middle) 4, and (bottom) 5

4.7 Object Search and Filtration

As the results or feature maps given by the model already segment and locate the object, it depends on the users' determination of what to do next. This section is going to suggest one example. In the case of GOTO sky images, the coordinates of transient objects are the real requirement from the model. Unfortunately, the result map does not count nor geotrack the objects. Literally, it only highlights the areas where objects exist and labels them with one. For this reason, we can use OpenCV's connectedComponentsWithStats to count the number of objects. This function detects a group of pixels, next to each other, in an array with the value of one and assigns each group with a new label—in other words, two, three, four, and so on. Therefore, the number of classes or labels in a result generated by this function represents the number of objects on the map. With a little Mathematics, we can get the center of each object

and so as its coordinate. Moreover, similar to the ground truth, objects on the map have a square shape of 49 pixels. We can use this fact to create more filters for results by counting and occurrence of each label. We can reset it to zero if its number is too big or too small, implying that it is not a transient (See Appendix G).

CHAPTER 5

CONCLUSIONS

This chapter is going to review what we have done so far according to research problems from the beginning to the result summarization. This also includes configuration suggestions and future work recommendations.

As the state-of-the-art detection is inefficient due to the large amount of data, GOTO researchers need to find a framework to deal with the studying of transients. Previously, a few ML approaches were conducted by other researchers. Nevertheless, the results do not meet GOTO expectations according to the class imbalance. We had tried to set up a network sharing computing system, but at that time, the concept lacked community support. Besides, that idea, of setting up the storage and computing system, was still not far from the imbalance problem. A new solution is required. We found that by using deep learning, called an FCN, we can analyze the sky images directly without the need of extracting learning attributes beforehand. This idea can help save time and space lost by the former approaches. However, it did not confirm that the imbalance problem still existed. Thus, several setups were tried in the experiments. The model precision was initially high. On the other hand, we found that the precision and recall were changing, depending on the parameter such as batch size or threshold. Hyperparameters such as depth or the number of convolutions also did matter for tuning the models' accuracy. It needs to be considered that the more complexity the model setup is, the longer time is required for the algorithm training. Moreover, tradeoff between precision recall existed, meaning that we had to decide which one to be improved. Fortunately, this step is only done one time in practice because transient objects have steady shape and quality. They do not appear different from time to time. As a result, a list of the best configurations was shown along with the time usage and their accuracy. While each model gives a result based on the complexity of input data, without further optimization, the model learned with a 16-batch dataset with 15 percent

salt, 30 percent pepper, and 0.20 Gaussian standard deviation gives rather equal results between each input data. For other models, it is found that there is a limitation on how much noise for the learning data can be. As the noise free models are not robust, too much noise can harm the models. To clarify, the models become focusing only on high noise data and lose their robustness.

Please note that all the stages from retrieving data, from FITS, to evaluation are performed with the Python 3 language. The code to generate the algorithm is put on the Appendix A; however, as Python libraries are changing all the time, it might cause an error when being run. For example, the legacy Tensorflow has methods different from the current Tensorflow. In this study all the code is run on Google Colaboratory (Colab) with Tensorflow 2.8.2+zzzcolab20220527125636 and Keras 2.8.0. Also, Google's Tensor Processing Units (TPUs) are used to speed up the process.

5.1 Discussion

First let us look at the model recap and understand its step-by-step approaches to a GOTO's problem. The first three steps are for model training and only one time process.

1. Two sets of sky images are selected to be an input for training the model. The difference between each image pair is based on the time aspect. Images containing transients are preferred to be the sample.

2. Each image pair is cut into mini batches, and all batches are normalized and converted into a four-dimensional array with a number of mini batches, input height, input width, time aspect representing each dimension respectively. The size after cutting will be the model input size from now on.

3. The array is put into the model, in this case, UNET, with parameters—batch size, model depth, and convolution filter size—properly set according to the setup guidelines, that will be explained later. Ground truth is required to teach the model right from wrong. This results in a model weight which can be used in a predicting.

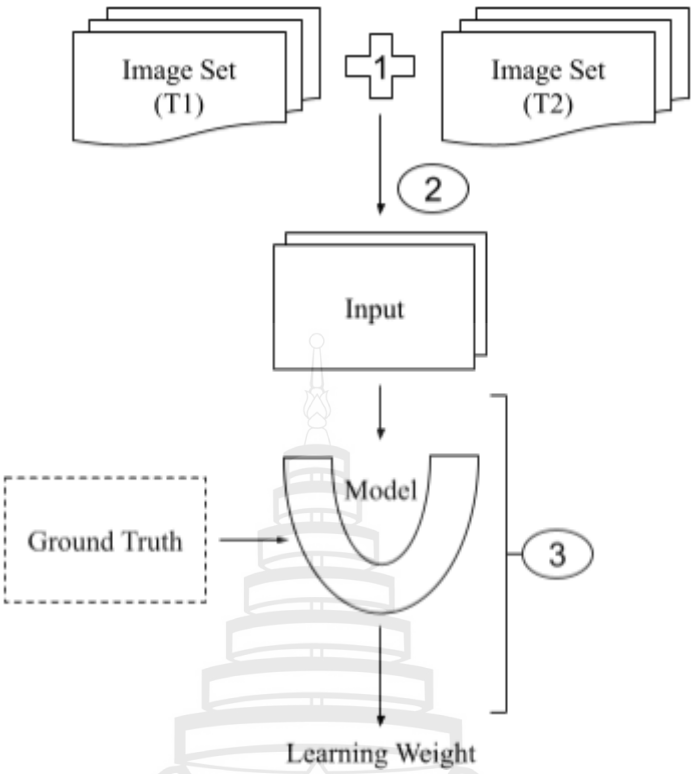

Figure 5.1 Summarization diagram (training)

The next steps are for predicting using a learning weight from the previous steps. These steps can also be done in real time after the telescope takes a sky photo, so that the large storage space is not required by the model.

4. Normally, telescopes keep every image taken in a container called FITS. This step can be bypassed by retrieving only an image or array containing pixel values leaving other metadata alone. However, in the diagram there is still a FITS file shown in case it needs to be analyzed by astronomers.

5. This step is similar to the first step by pairing a new image with a reference one. The reference images should be the ones containing no transient event or minimum events as much as possible for the model to be effective.

6. Similar to the second step, the pair is normalized and converted into a four-dimensional array.

7. The model containing the same structure as the one used in training is preloaded with the learning weight. The input is submitted to the model without any ground truth required this time.

8. The output requires post-processing by being put through a threshold. Normally, a threshold level is fifty percent, but it can be adjusted desirably for more recall or precision.

9. The final result is acquired. The array can be reshaped to be a full size image again and prepared to be analyzed by astronomers. One of the suggestions before analyzing is to get the relative coordinates of each transient object based on positions in an image. This can be done by following the subsection, 4.7 Object search and filtration, in chapter four.

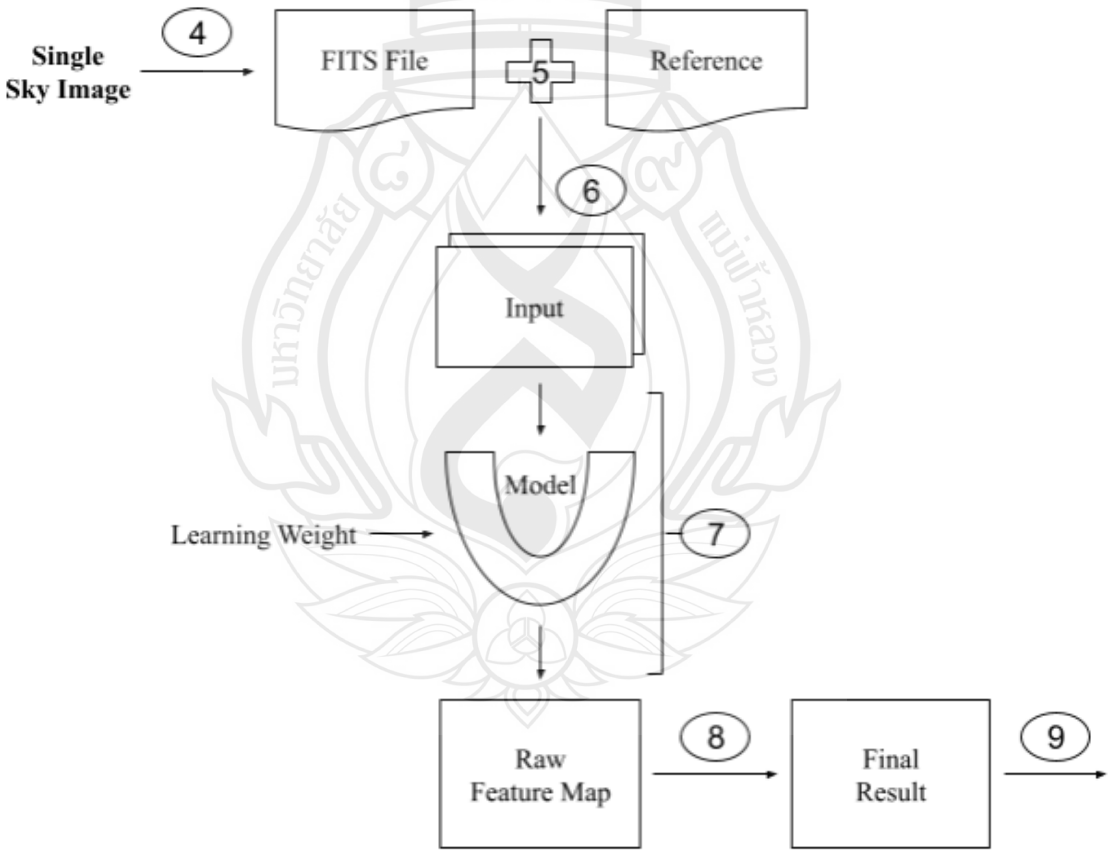

Figure 5.2 Summarization diagram (predicting)

For parameter and hyperparameter setup, there are three suggestions based on the noise levels. In this case threshold level, model depth, and convolution filter size are left untouched until the last optimization as they can only adjust recall and precision.

1. Normally, there should be only a Gaussian noise during the capture process. In this case a model learned with the batch size of 8 or 16 with 15 percent pepper noise is suggested as the prediction accuracies for 0.2 and 0.4 standard deviations of Gaussian noise are rather high.

2. In the case of a robust efficient model that can deal with more kinds of noise environments, the model learned with the batch size of 16, 15 percent salt, 30 percent pepper, and 0.2 standard deviation of Gaussian noise is recommended. However, its average recall and precision are quite lower than the first case in exchange for robustness.

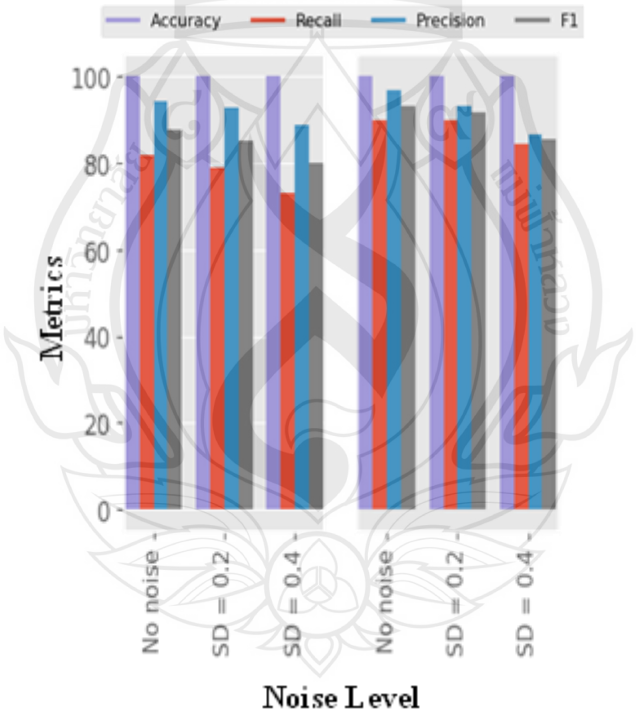

Figure 5.3 Model 0s-15p-0sd performance based on Gaussian noise: (left) batch size of 8 and (right) batch size of 16

3. If astronomers want to focus on recall instead of an overall accuracy for robustness, they can still use the model learned with the batch size of 16 with 15 percent pepper noise. Although the precision rates drastically drop when noise increases, the recall rates only drop by a little. As shown on Figure 5.4, here is the comparison between this model and the model on the second case.

For optimizing precision and recall the model can be reconfigured with increasing or decreasing the threshold level, model depth, and convolution filter size according to the following table. However, we need to decide to focus between recall and precision as when one is optimized, another one is dropped.

Table 5.1 Parameter guidelines

Parameter	Recall	Precision
Threshold level (default: 50)	Should be decreased (↓)	Should be increased (↑)
Depth (default: 4)	—	Can be either decreased (↓) or increased (↑) [1]
Filter size (default: 64)	—	Can be either decreased (↓) or increased (↑) [1]

Note [1] Effect varies according to how well the model is trained.

Be noted that according to the findings only the effect of changing the threshold level is clearly explicit. For depth and filter size, the effects might different between the models, and too high values can lead to model overfitting.

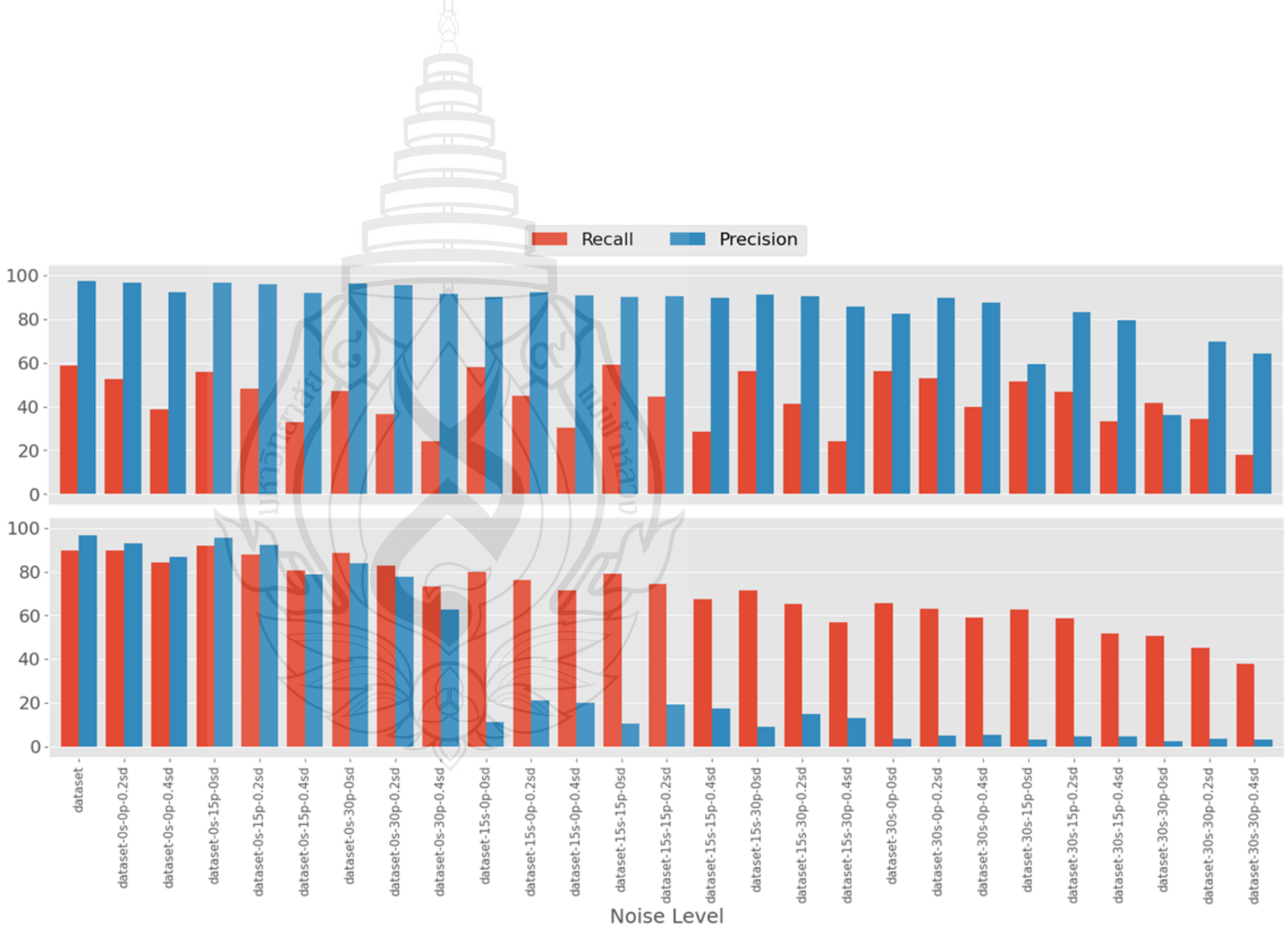

Figure 5.4 Comparison between model 16b-15s-30p-0.2sd and 16b-0s-15p-0sd performance

5.2 Recommendations for Future Research

The framework needs to be studied further to confirm that it works. Although the preprocessing and learning are minimized as much as possible to prevent the overfitting, there might be other factors related to the model accuracy, for instance, the quality of learning input. According to the scope of the study mentioned earlier, there are two things to consider.

First, the transient objects in the experiment are fully simulated. Even though they are created according to the information collected, it is still unknown what model response to them is in practice. It is expected to see the model tell the transients from the static objects correctly; however, it is difficult to measure a result as the transient events are rare to occur.

Besides, as the environmental noises from the sky image capture are represented by the statistical modification of data, it does not guarantee that the model can overcome other kinds of errors. It, however, is expected to have high efficiency due to the fine tuning from several experiments.

Lastly, apart from these limitations, there are hyperparameters we left unchanged. We derived the loss function and optimization function from the novel studies. These functions are the popular ones used in machine learning. However, there are a variety of functions that remain to be tested.

Apart from the data and parameter parts, the model can be further observed, for example, by changing or replacing layers inside. There are also a variety of FCN models that are derived from the UNET concept. At the same time, it is also interesting to separate the detection into multiple parts and implement algorithms onto each of them.

REFERENCES

REFERENCES

[1] Sangjan, T., Boongoen, T., Iam-on, N., & Mullaney, J. (2019, October 3-6). *Classification of astronomical objects using light curve profile* [Paper presentation]. 2019 IEEE Eurasia Conference on IOT, Communication and Engineering (ECICE), Yunlin, Taiwan.

[2] Tabacolde, A. B., Boongoen, T., Iam-On, N., Mullaney, J., Sawangwit, U., & Ulaczyk, K. (2018, July 23-27). *Transient detection modeling as imbalance data classification* [Paper presentation]. 2018 1st IEEE International Conference on Knowledge Innovation and Invention (ICKII), Jeju Island, South Korea.

[3] Tabacolde, A. B., Boongoen, T., Iam-On, N., Mullaney, J., Sawangwit, U., & Ulaczyk, K. (2018, February 26-28). *Transient detection modelling for Gravitational-wave Optical Transient Observer (GOTO) sky survey* [Paper presentation]. 2018 10th International Conference on Machine Learning and Computing (ICMLC), Macau, China.

[4] Barchi, P. H., de Carvalho, R. R., Rosa, R. R., Sautter, R. A., Soares-Santos, M., Marques, B.A.D., . . . Moura, T. C. (2020). Machine and deep learning applied to galaxy morphology - A comparative study. *Astronomy and Computing, 30*, Article 100334.

[5] Padilla, C. (2011). *Identifying electromagnetic transients using image subtraction* [Paper presentation]. 2011 International Research Experience for Undergraduates (IREU), Daejeon, South Korea.

[6] Zackay, B., Ofek, E. O., & Gal-Yam, A. (2016). Proper image subtraction - Optimal transient detection, photometry and hypothesis testing. *The Astrophysical Journal, 830*(1), Article 27.

[7] Sánchez, B., Domínguez R., M. J., Lares, M., Beroiz, M., Cabral, J. B., Gurovich, S., . . . Díaz, M. C. (2019). Machine learning on difference image analysis: A comparison of methods for transient detection. *Astronomy and Computing, 28*, Article 100284.

[8] Djorgovski, S. G., Graham, M. J., Donalek, C., Mahabal, A. A., Drake, A. J., Turmon, M., . . . Fuchs, T. (2016). Real-time data mining of massive data streams from synoptic sky surveys. *Future Generation Computer Systems, 59*, 95-104.

[9] Kuminski, E., & Shamir, L. (2018). A hybrid approach to machine learning annotation of large galaxy image databases. *Astronomy and Computing, 25*, 257-269.

[10] González, R. E., Muñoz, R. P., & Hernández, C. A. (2018). Galaxy detection and identification using deep learning and data augmentation. *Astronomy and Computing, 25*, 103-109.

[11] Alard, C., & Lupton, R. (1998). A method for optimal image subtraction. *The Astrophysical Journal, 503*(1), 325-331.

[12] Bramich, D. M. (2008). A new algorithm for difference image analysis. *Monthly Notices of the Royal Astronomical Society: Letters, 386*(1), L77–L81.

[13] Noh, H., Hong, S., & Han, B. (2015, December 7-13). *Learning deconvolution network for semantic segmentation* [Paper presentation]. 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, Chile.

[14] Guo, E., Fu, X., Zhu, J., Deng, M., Liu, Y., Zhu, Q., . . . Li, H. (2018). *Learning to measure change: Fully convolutional siamese metric networks for scene change detection*. arXiv. https://doi.org/10.48550/arXiv.1810.09111

[15] Peng, D., Zhang, Y., & Guan, H. (2019). End-to-end change detection for high resolution satellite images using improved UNet++. *Remote Sensing, 11*(11), Article 1382.

[16] Daudt, R. C., Saux, B. L., & Boulch, A. (2018, October 7-10). *Fully convolutional siamese networks for change detection* [Paper presentation]. 2018 25th IEEE International Conference on Image Processing (ICIP), Athens, Greece.

[17] Lan, L., Wu, D., & Chi, M. (2019, August 5-7). *Multi-temporal change detection based on deep semantic segmentation networks* [Paper presentation]. 2019 10th International Workshop on the Analysis of Multitemporal Remote Sensing Images (MultiTemp), Shanghai, China.

[18] Varghese, A., Gubbi, J., Ramaswamy, A., & Purushothaman, B. (2018, September 8-14). *ChangeNet: A deep learning architecture for visual change detection* [Paper presentation]. 2018 15th European Conference on Computer Vision (ECCV), Munich, Germany.

[19] Scroggins, M., & Boscoe, B. M. (2020). Once FITS, always FITS? Astronomical infrastructure in transition. *IEEE Annals of the History of Computing, 42*(2), 42-54.

[20] Thomas, B., Jenness, T., Economou, F., Greenfield, P., Hirst, P., Berry, D. S., . . . Homeier, D. (2015). Learning from FITS: Limitations in use in modern astronomical research. *Astronomy and Computing, 12*, 133-145.

[21] Chen, M., Deng, H., Wang, F., & Ji, K. (2013, November 1-3). *A survey on file format for data storage in radio astronomy* [Paper presentation]. 2013 6th International Conference on Intelligent Networks and Intelligent Systems (ICINIS), Shenyang, China.

[22] Apache Software Foundation. (2019, January 29). *Apache Hadoop 3.1.2*. https://hadoop.apache.org/docs/r3.1.2/

[23] Ronneberger, O., Fischer, P., & Brox, T. (2015, October 5-9). *U-Net: Convolutional networks for biomedical image segmentation* [Paper presentation]. 2015 18th Medical Image Computing and Computer-Assisted Intervention (MICCAI), Munich, Germany.

APPENDICES

APPENDIX A

PYTHON UNET SETUP

The Python 3 code used to create the FCN for the experiment is shown below.

```
def get_unet(img_shape, # Input shape (height, width, channels)
                out_ch = 1, # Number of output channels
                start_ch = 64, # Number of channels for the first convolution (filters)
                depth = 4, # Zero indexed depth of the U-structure
                inc_rate = 2., # Filter multiplier for the deeper blocks
                activation = 'relu', # Activation function after convolutions
                dropout = 0.5, # Amount of dropout in the contracting part
                batchnorm = True, # Batch Normalization
                maxpool = True, # Max pooling (True) or strided convolution (False)
                deconv = True, # Transposed convolution (True) or upsampling (False)
                residual = False # Adding residual connections around each convolution block
                ):
    def _conv_block(m, dim, acti, bn, res, do=0): # Create two convolutions
        n = Conv2D(dim, 3, activation=acti, padding='same')(m)
        n = BatchNormalization()(n) if bn else n
        n = Dropout(do)(n) if do else n
        n = Conv2D(dim, 3, activation=acti, padding='same')(n)
        n = BatchNormalization()(n) if bn else n
        return Concatenate()([m, n]) if res else n
    def _level_block(m, dim, depth, inc, acti, do, bn, mp, de, res):
        if depth > 0:
            n = _conv_block(m, dim, acti, bn, res, do)
            m = MaxPooling2D()(n) if mp else Conv2D(dim, 3, strides=2, padding='same')(n)
            m = _level_block(m, int(inc*dim), depth-1, inc, acti, do, bn, mp, de, res) # Recurrence
            if de:
                m = Conv2DTranspose(dim, 3, strides=2, activation=acti, padding='same')(m)
            else:
                m = UpSampling2D()(m)
                m = Conv2D(dim, 2, activation=acti, padding='same')(m)
            n = Concatenate()([n, m])
            m = _conv_block(n, dim, acti, bn, res)
        else:
            m = _conv_block(m, dim, acti, bn, res, do)
        return m
    i = Input(shape=img_shape)
    o = _level_block(i, start_ch, depth, inc_rate, activation,
                     dropout, batchnorm, maxpool, deconv, residual)
    o = Conv2D(out_ch, 1, activation='sigmoid')(o)
    return Model(inputs=i, outputs=o)
```

Figure A1 Python code to create fully convolutional network

APPENDIX B

EXAMPLES OF NOISE COMBINATIONS

27 combinations of salt, pepper, and Gaussian noises are in the following:

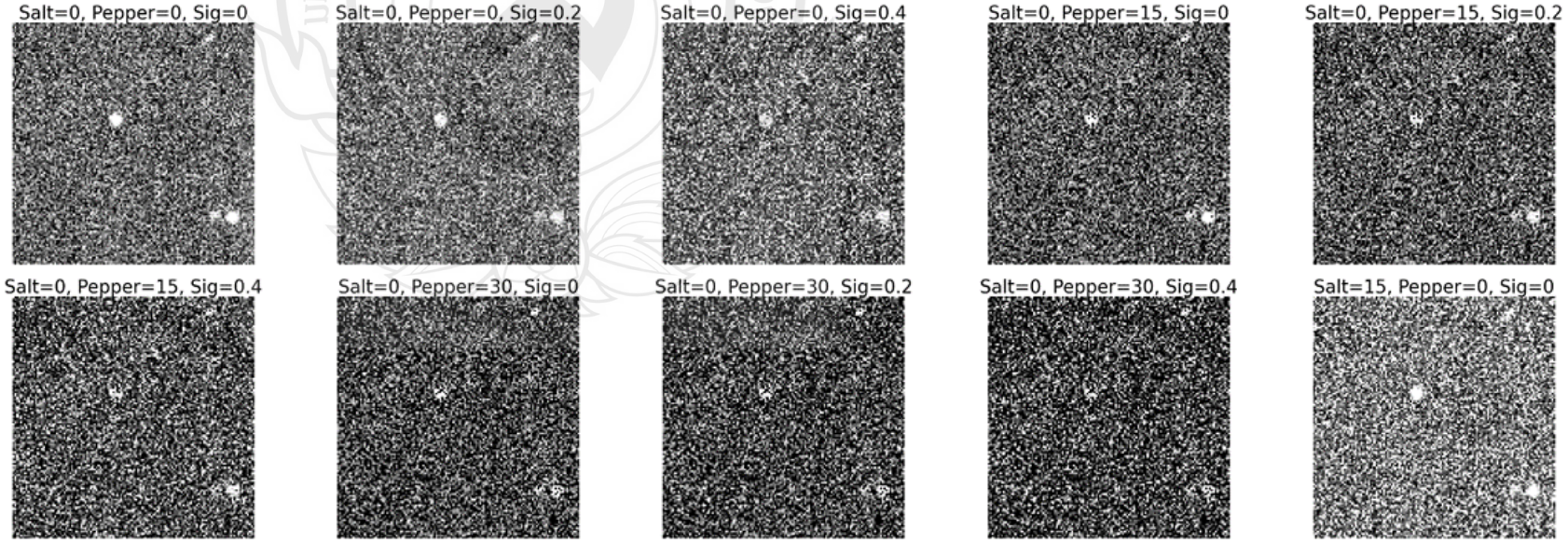

Figure B1 Noise combinations

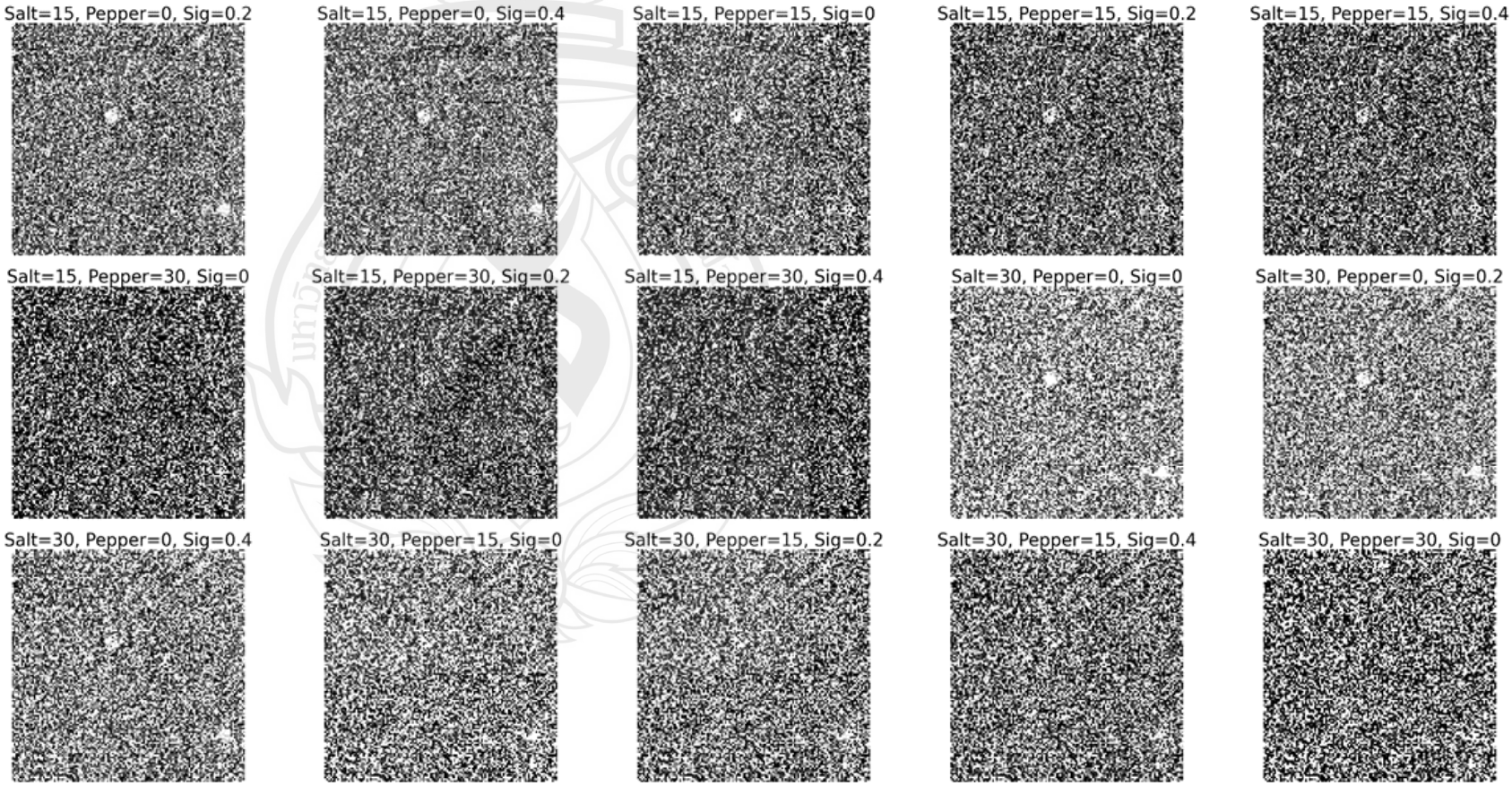

Figure B1 (continued)

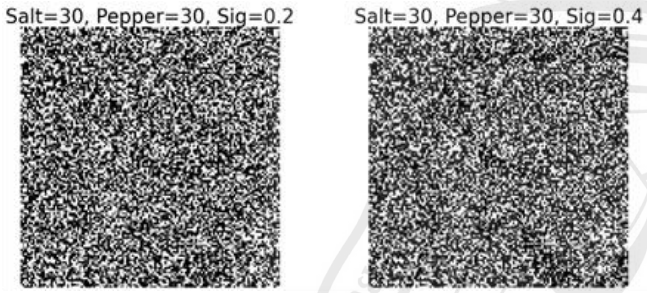

Figure B1 (continued)

APPENDIX C

MODEL COMBINATIONS

The following table lists the models, generated in each learning, that are mentioned in chapter four, with their batch size, different noise setup, and learning time usage.

Table C1 Model combinations

Order	Name	Batch size	Salt (%)	Pepper (%)	Standard deviation	Time (Minute)
1	8b	8	0	0	0	7:51
2	8b-0s-0p-0.2sd	8	0	0	0.2	4:39
3	8b-0s-0p-0.4sd	8	0	0	0.4	4:41
4	8b-0s-15p-0sd	8	0	15	0	4:60
5	8b-0s-15p-0.2sd	8	0	15	0.2	6:19
6	8b-0s-15p-0.4sd	8	0	15	0.4	4:41
7	8b-0s-30p-0sd	8	0	30	0	5:02
8	8b-0s-30p-0.2sd	8	0	30	0.2	5:02
9	8b-0s-30p-0.4sd	8	0	30	0.4	5:41
10	8b-15s-0p-0sd	8	15	0	0	5:01
11	8b-15s-0p-0.2sd	8	15	0	0.2	5:00
12	8b-15s-0p-0.4sd	8	15	0	0.4	5:03
13	8b-15s-15p-0sd	8	15	15	0	5:02
14	8b-15s-15p-0.2sd	8	15	15	0.2	4:42
15	8b-15s-15p-0.4sd	8	15	15	0.4	5:02
16	8b-15s-30p-0sd	8	15	30	0	5:03
17	8b-15s-30p-0.2sd	8	15	30	0.2	5:04

Table C1 (continued)

Order	Name	Batch size	Salt (%)	Pepper (%)	Standard deviation	Time (Minute)
18	8b-15s-30p-0.4sd	8	15	30	0.4	5:03
19	8b-30s-0p-0sd	8	30	0	0	5:06
20	8b-30s-0p-0.2sd	8	30	0	0.2	4:46
21	8b-30s-0p-0.4sd	8	30	0	0.4	5:06
22	8b-30s-15p-0sd	8	30	15	0	4:43
23	8b-30s-15p-0.2sd	8	30	15	0.2	4:42
24	8b-30s-15p-0.4sd	8	30	15	0.4	5:05
25	8b-30s-30p-0sd	8	30	30	0	5:56
26	8b-30s-30p-0.2sd	8	30	30	0.2	5:18
27	8b-30s-30p-0.4sd	8	30	30	0.4	4:22
28	16b	16	0	0	0	3:09
29	16b-0s-0p-0.2sd	16	0	0	0.2	3:10
30	16b-0s-0p-0.4sd	16	0	0	0.4	3:19
31	16b-0s-15p-0sd	16	0	15	0	3:20
32	16b-0s-15p-0.2sd	16	0	15	0.2	3:11
33	16b-0s-15p-0.4sd	16	0	15	0.4	3:20
34	16b-0s-30p-0sd	16	0	30	0	3:12
35	16b-0s-30p-0.2sd	16	0	30	0.2	3:22
36	16b-0s-30p-0.4sd	16	0	30	0.4	3:34
37	16b-15s-0p-0sd	16	15	0	0	3:13
38	16b-15s-0p-0.2sd	16	15	0	0.2	3:13
39	16b-15s-0p-0.4sd	16	15	0	0.4	3:11
40	16b-15s-15p-0sd	16	15	15	0	3:11
41	16b-15s-15p-0.2sd	16	15	15	0.2	3:21
42	16b-15s-15p-0.4sd	16	15	15	0.4	3:23
43	16b-15s-30p-0sd	16	15	30	0	3:31
44	16b-15s-30p-0.2sd	16	15	30	0.2	3:32
45	16b-15s-30p-0.4sd	16	15	30	0.4	3:21

Table C1 (continued)

Order	Name	Batch size	Salt (%)	Pepper (%)	Standard deviation	Time (Minute)
46	16b-30s-0p-0sd	16	30	0	0	3:02
47	16b-30s-0p-0.2sd	16	30	0	0.2	3:12
48	16b-30s-0p-0.4sd	16	30	0	0.4	3:13
49	16b-30s-15p-0sd	16	30	15	0	3:18
50	16b-30s-15p-0.2sd	16	30	15	0.2	3:28
51	16b-30s-15p-0.4sd	16	30	15	0.4	3:28
52	16b-30s-30p-0sd	16	30	30	0	3:29
53	16b-30s-30p-0.2sd	16	30	30	0.2	3:19
54	16b-30s-30p-0.4sd	16	30	30	0.4	2:57
55	32b	32	0	0	0	3:19
56	32b-0s-0p-0.2sd	32	0	0	0.2	3:42
57	32b-0s-0p-0.4sd	32	0	0	0.4	3:03
58	32b-0s-15p-0sd	32	0	15	0	3:12
59	32b-0s-15p-0.2sd	32	0	15	0.2	3:05
60	32b-0s-15p-0.4sd	32	0	15	0.4	3:10
61	32b-0s-30p-0sd	32	0	30	0	3:19
62	32b-0s-30p-0.2sd	32	0	30	0.2	3:27
63	32b-0s-30p-0.4sd	32	0	30	0.4	4:23
64	32b-15s-0p-0sd	32	15	0	0	4:07
65	32b-15s-0p-0.2sd	32	15	0	0.2	3:44
66	32b-15s-0p-0.4sd	32	15	0	0.4	3:13
67	32b-15s-15p-0sd	32	15	15	0	3:05
68	32b-15s-15p-0.2sd	32	15	15	0.2	3:35
69	32b-15s-15p-0.4sd	32	15	15	0.4	3:26
70	32b-15s-30p-0sd	32	15	30	0	3:26
71	32b-15s-30p-0.2sd	32	15	30	0.2	3:30
72	32b-15s-30p-0.4sd	32	15	30	0.4	3:27
73	32b-30s-0p-0sd	32	30	0	0	3:37

Table C1 (continued)

Order	Name	Batch size	Salt (%)	Pepper (%)	Standard deviation	Time (Minute)
74	32b-30s-0p-0.2sd	32	30	0	0.2	3:14
75	32b-30s-0p-0.4sd	32	30	0	0.4	3:35
76	32b-30s-15p-0sd	32	30	15	0	3:22
77	32b-30s-15p-0.2sd	32	30	15	0.2	3:20
78	32b-30s-15p-0.4sd	32	30	15	0.4	3:45
79	32b-30s-30p-0sd	32	30	30	0	2:28
80	32b-30s-30p-0.2sd	32	30	30	0.2	3:20
81	32b-30s-30p-0.4sd	32	30	30	0.4	2:49

APPENDIX D

DATASET COMBINATIONS

Like Appendix C, the following table lists the different datasets used in the experiment. Each set is run with the mentioned models 81 times in order to get all possible outcomes.

Table D1 Dataset combinations

Combination	Name	Salt (%)	Pepper (%)	Standard deviation
1	dataset	0	0	0
2	dataset-0s-0p-0.2sd	0	0	0.2
3	dataset-0s-0p-0.4sd	0	0	0.4
4	dataset-0s-15p-0sd	0	15	0
5	dataset-0s-15p-0.2sd	0	15	0.2
6	dataset-0s-15p-0.4sd	0	15	0.4
7	dataset-0s-30p-0sd	0	30	0
8	dataset-0s-30p-0.2sd	0	30	0.2
9	dataset-0s-30p-0.4sd	0	30	0.4
10	dataset-15s-0p-0sd	15	0	0
11	dataset-15s-0p-0.2sd	15	0	0.2
12	dataset-15s-0p-0.4sd	15	0	0.4
13	dataset-15s-15p-0sd	15	15	0
14	dataset-15s-15p-0.2sd	15	15	0.2
15	dataset-15s-15p-0.4sd	15	15	0.4
16	dataset-15s-30p-0sd	15	30	0
17	dataset-15s-30p-0.2sd	15	30	0.2

Table D1 (continued)

Combination	Name	Salt (%)	Pepper (%)	Standard deviation
18	dataset-15s-30p-0.4sd	15	30	0.4
19	dataset-30s-0p-0sd	30	0	0
20	dataset-30s-0p-0.2sd	30	0	0.2
21	dataset-30s-0p-0.4sd	30	0	0.4
22	dataset-30s-15p-0sd	30	15	0
23	dataset-30s-15p-0.2sd	30	15	0.2
24	dataset-30s-15p-0.4sd	30	15	0.4
25	dataset-30s-30p-0sd	30	30	0
26	dataset-30s-30p-0.2sd	30	30	0.2
27	dataset-30s-30p-0.4sd	30	30	0.4

APPENDIX E

PERFORMANCE CLASSIFIED BY MODEL COMBINATIONS

Here are 2,187 outcomes (with 50% threshold), showing evaluation metrics—accuracy, recall, precision, and F1—separated into 81 groups according to the Appendix C. Each group contains 27 outcomes according to the dataset combination on the Appendix D.

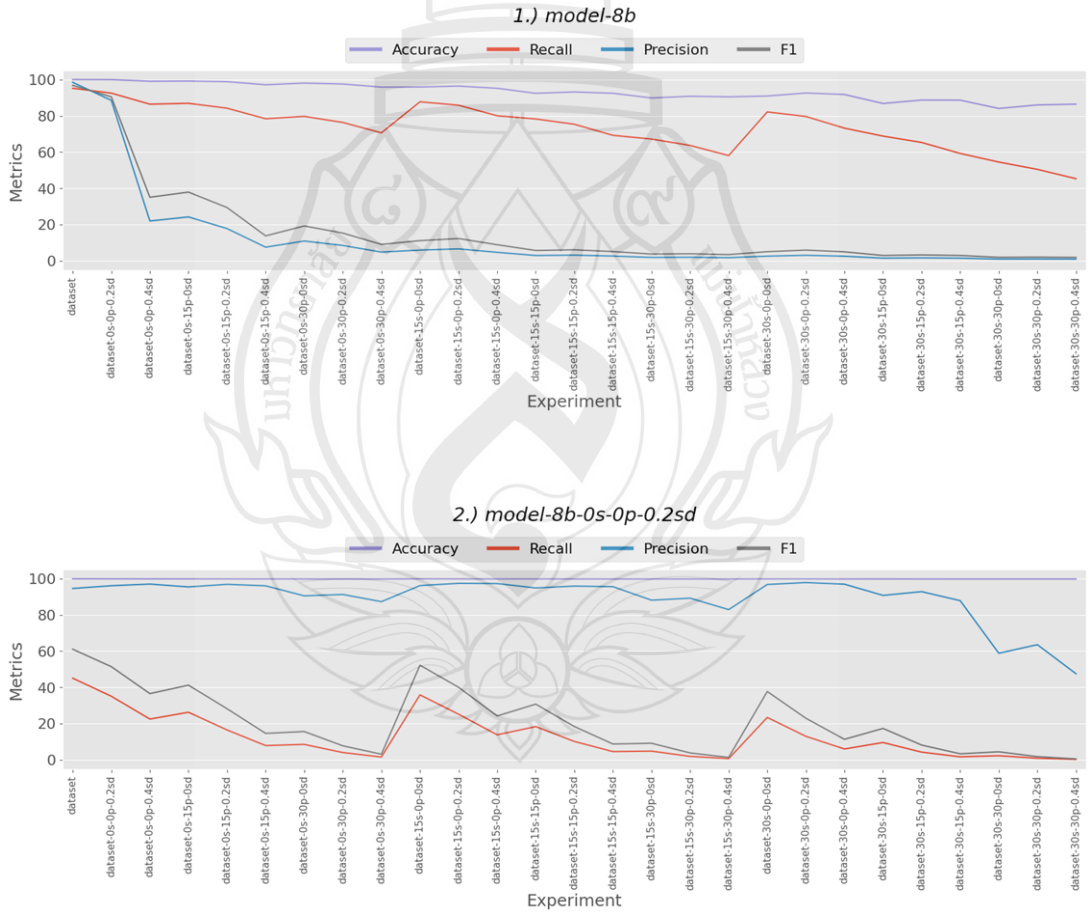

Figure E1 Performance classified by model combinations

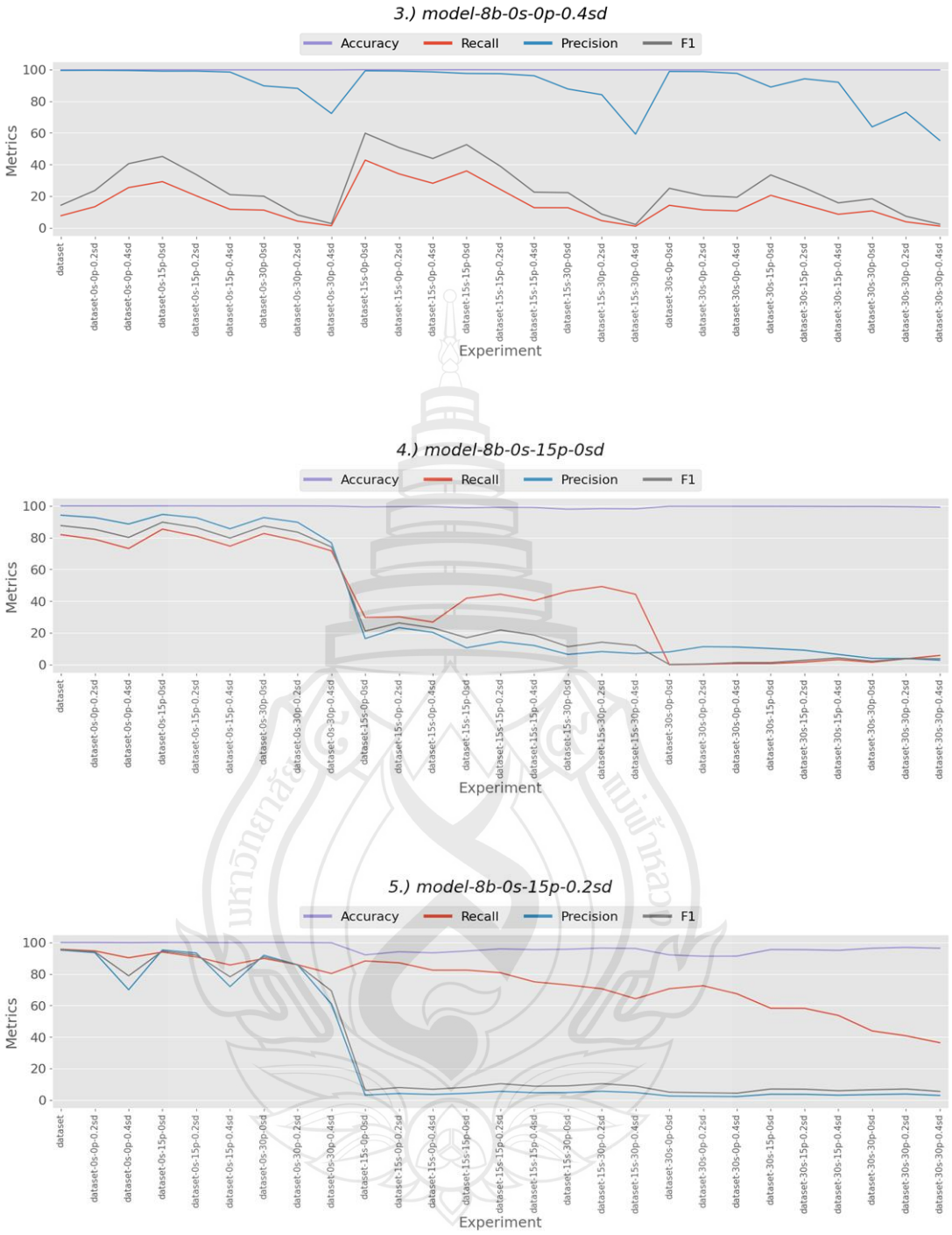

Figure E1 (continued)

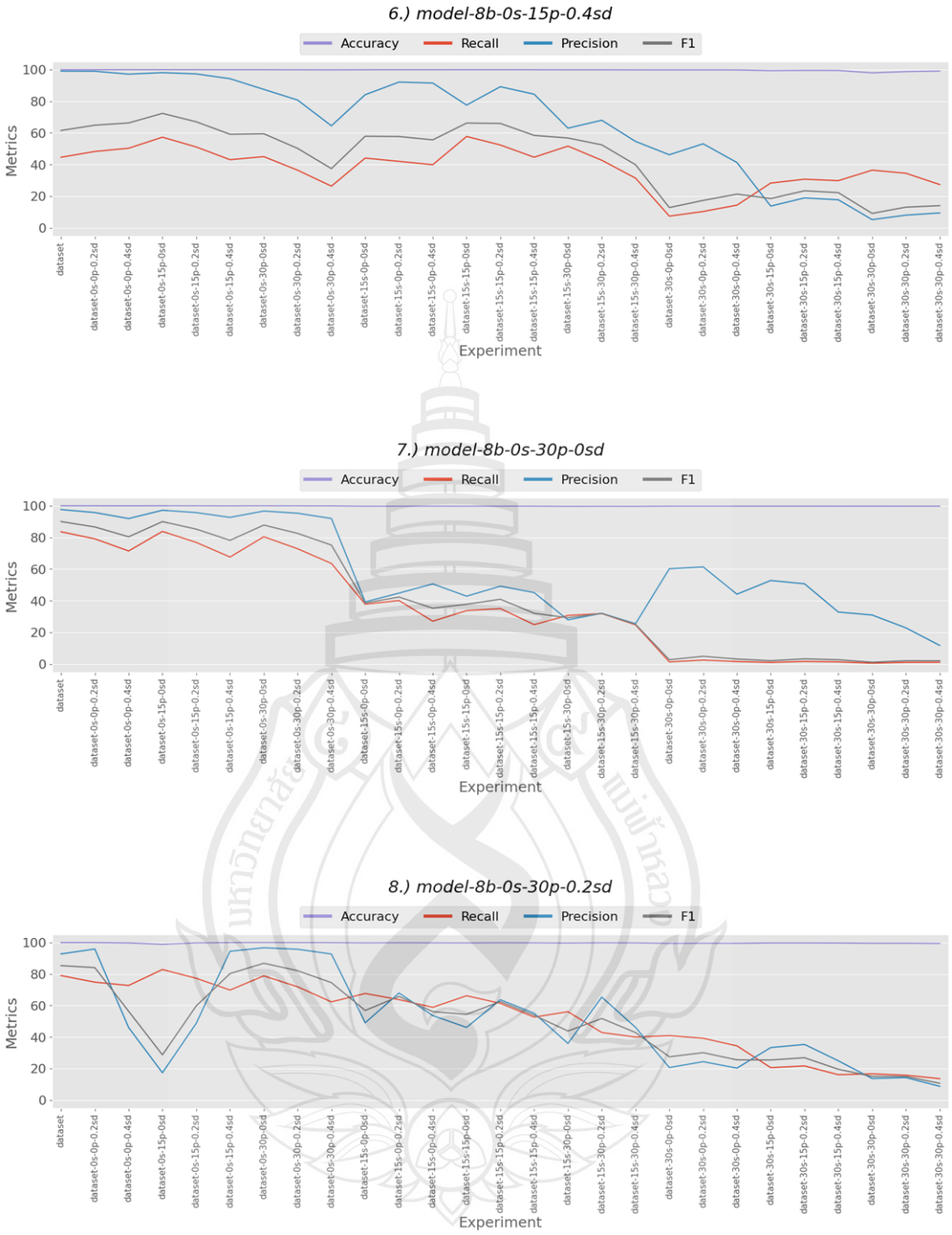

Figure E1 (continued)

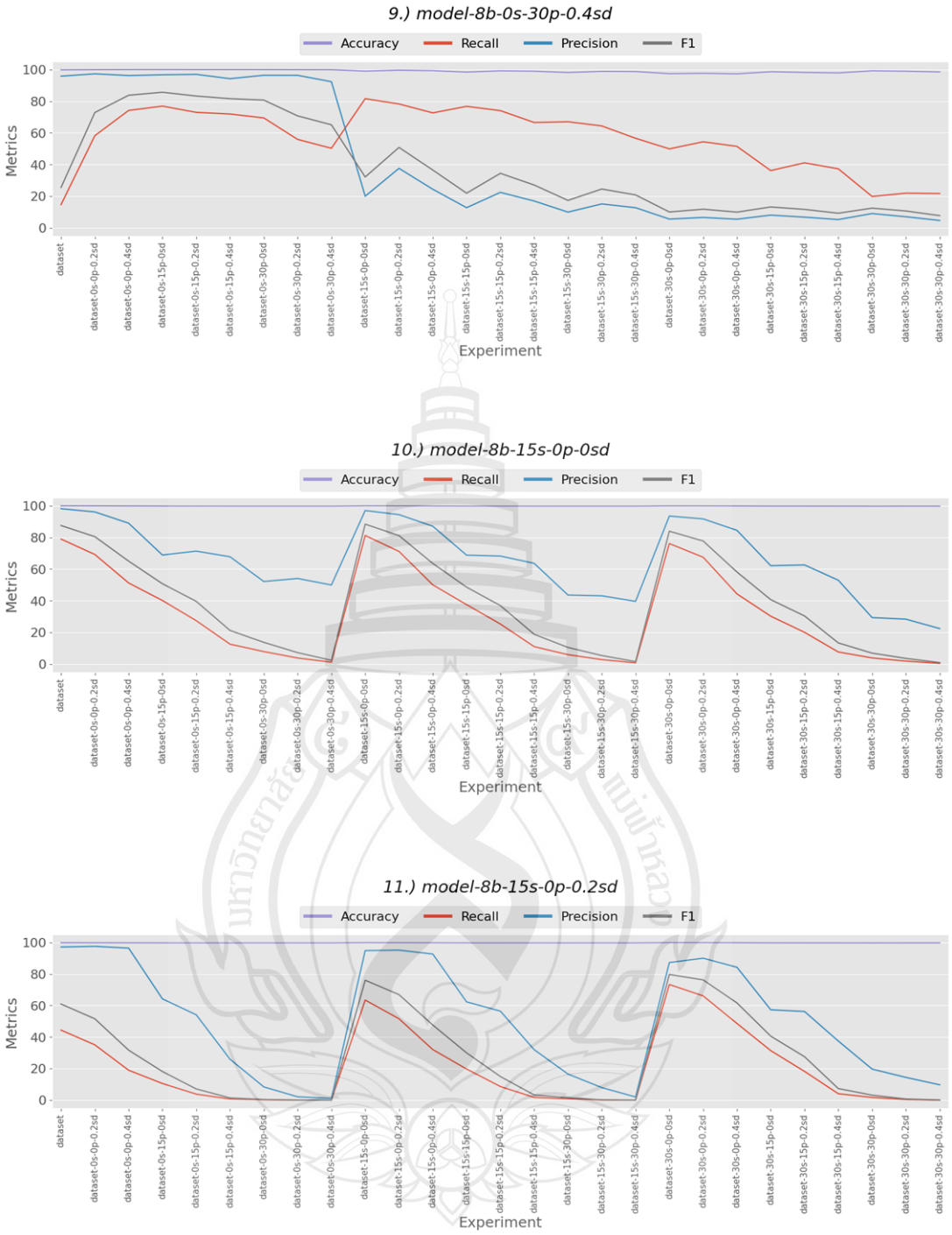

Figure E1 (continued)

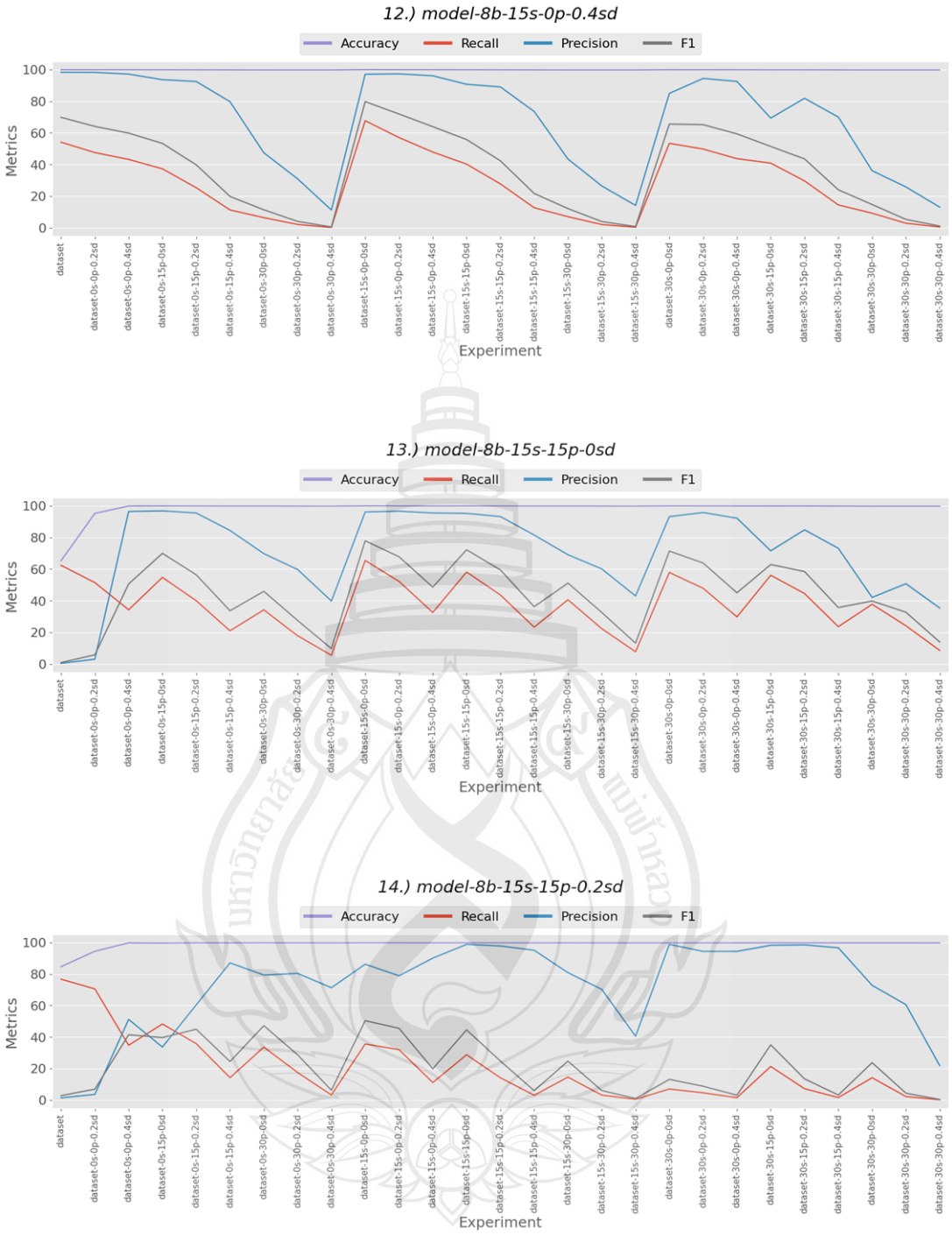

Figure E1 (continued)

Figure E1 (continued)

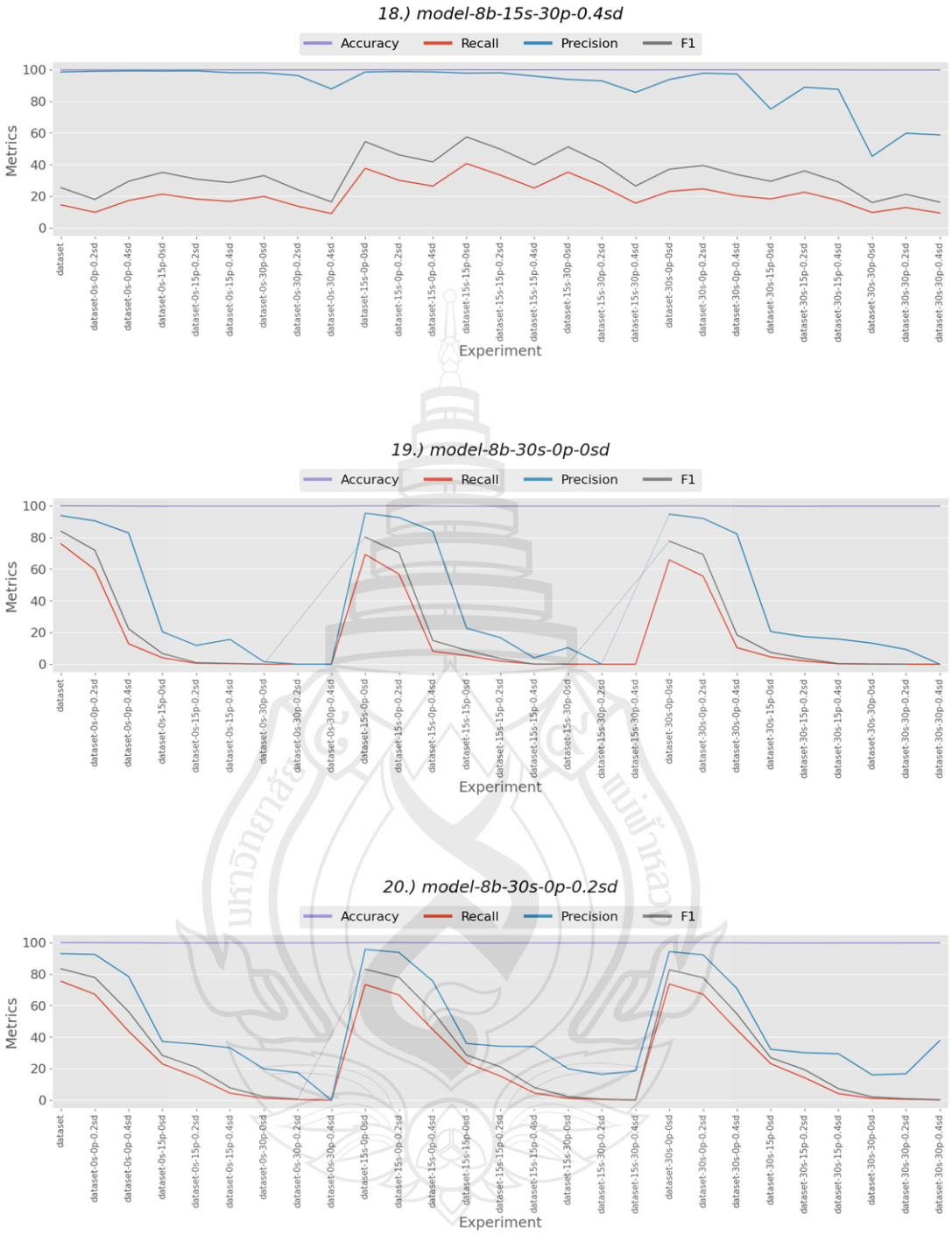

Figure E1 (continued)

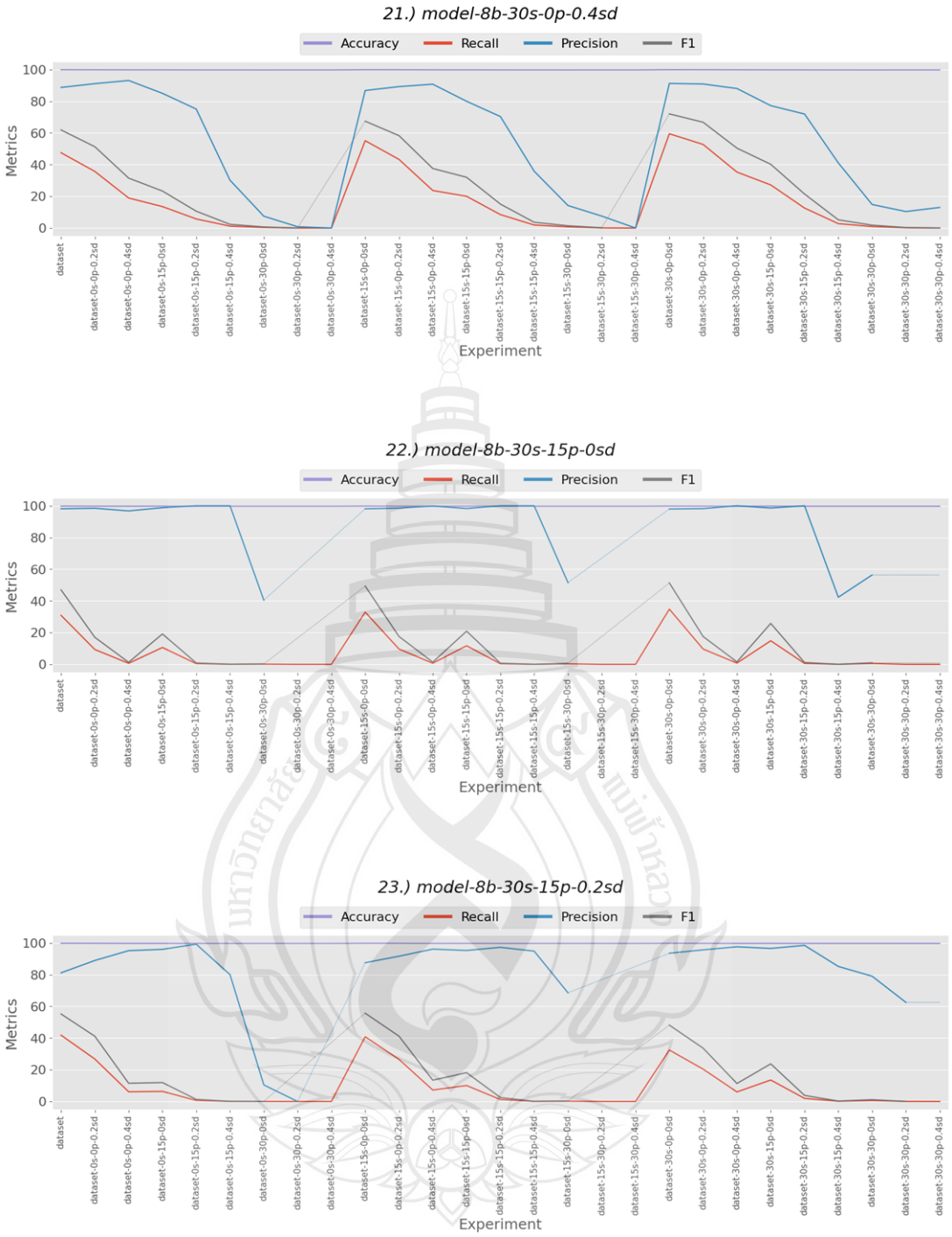

Figure E1 (continued)

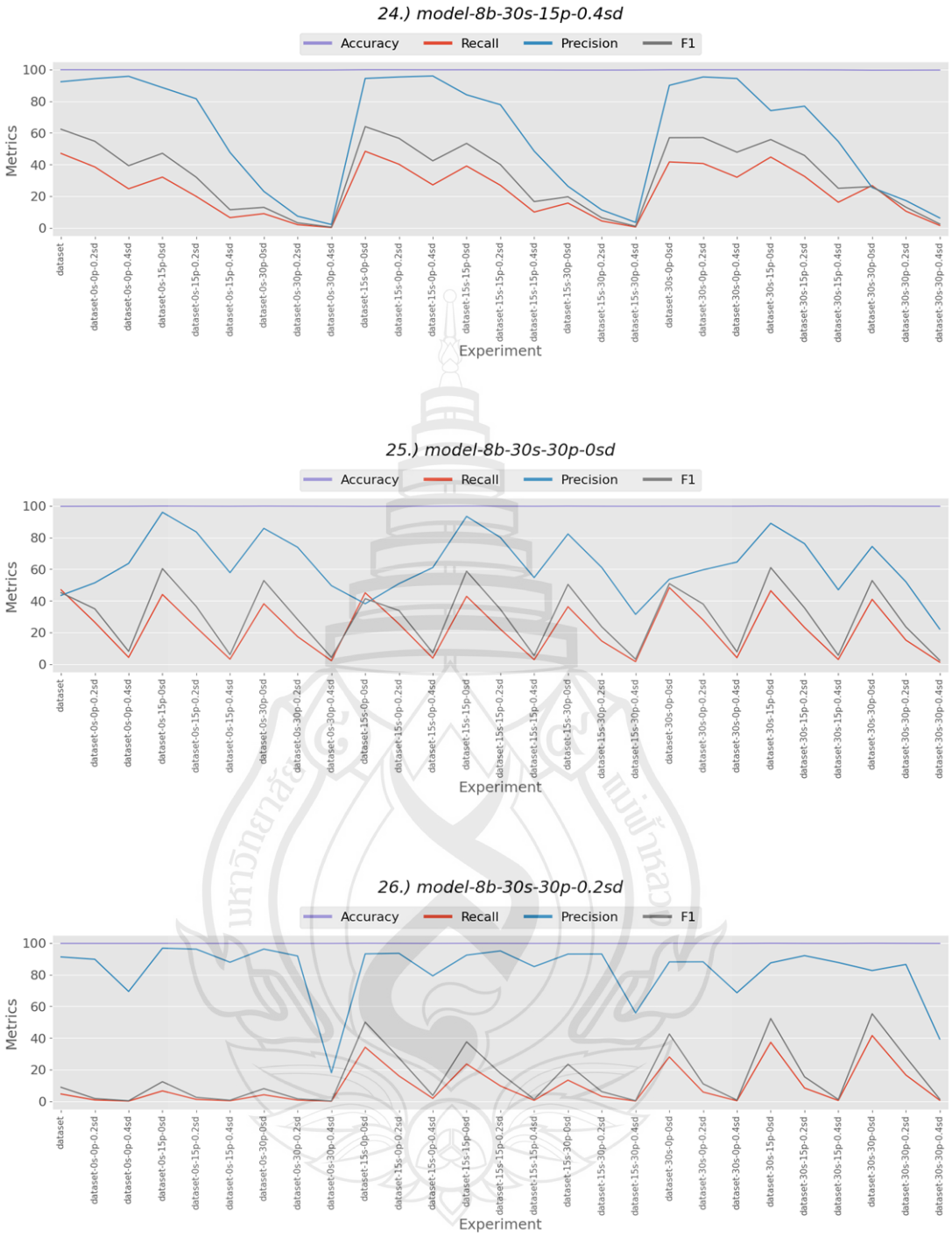

Figure E1 (continued)

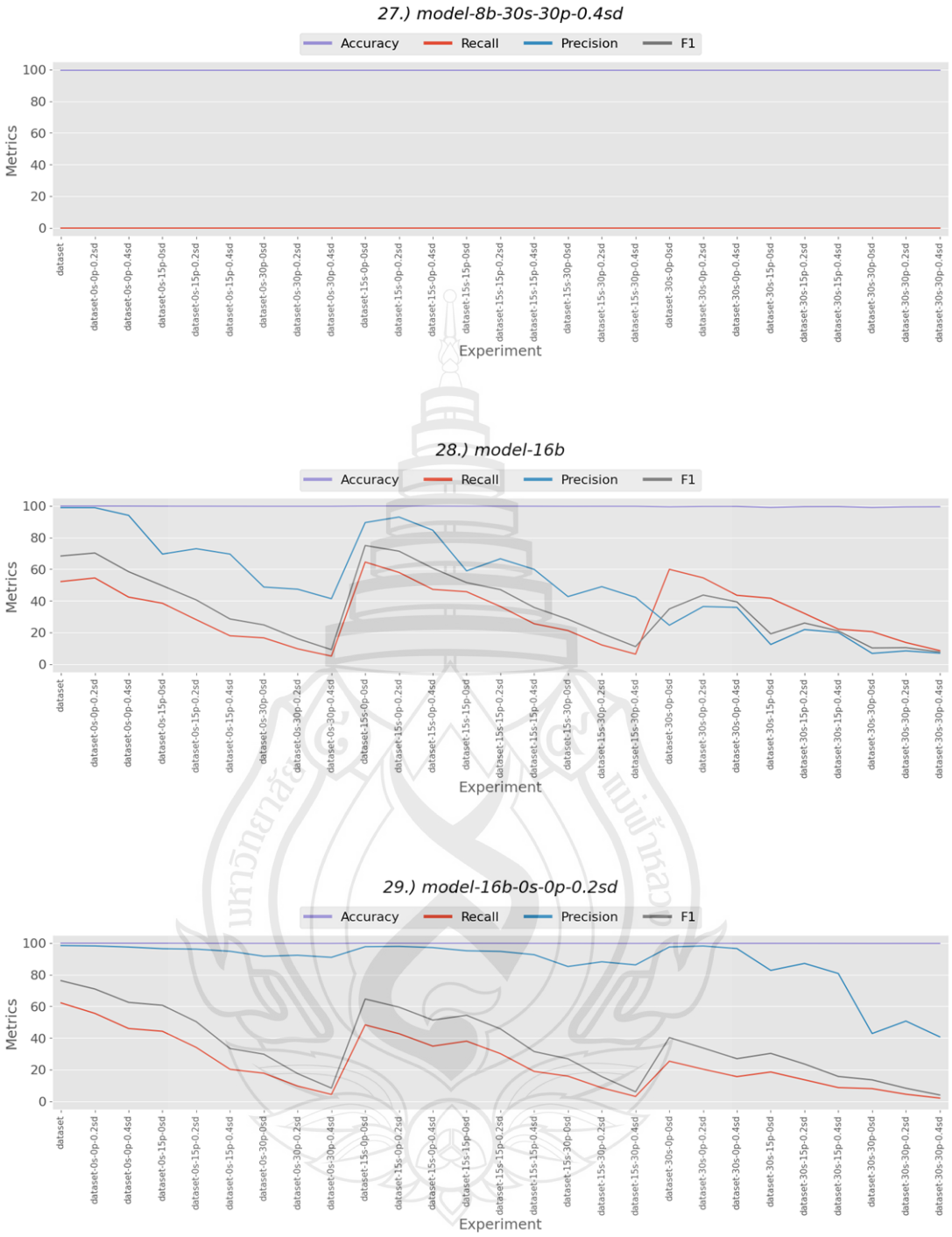

Figure E1 (continued)

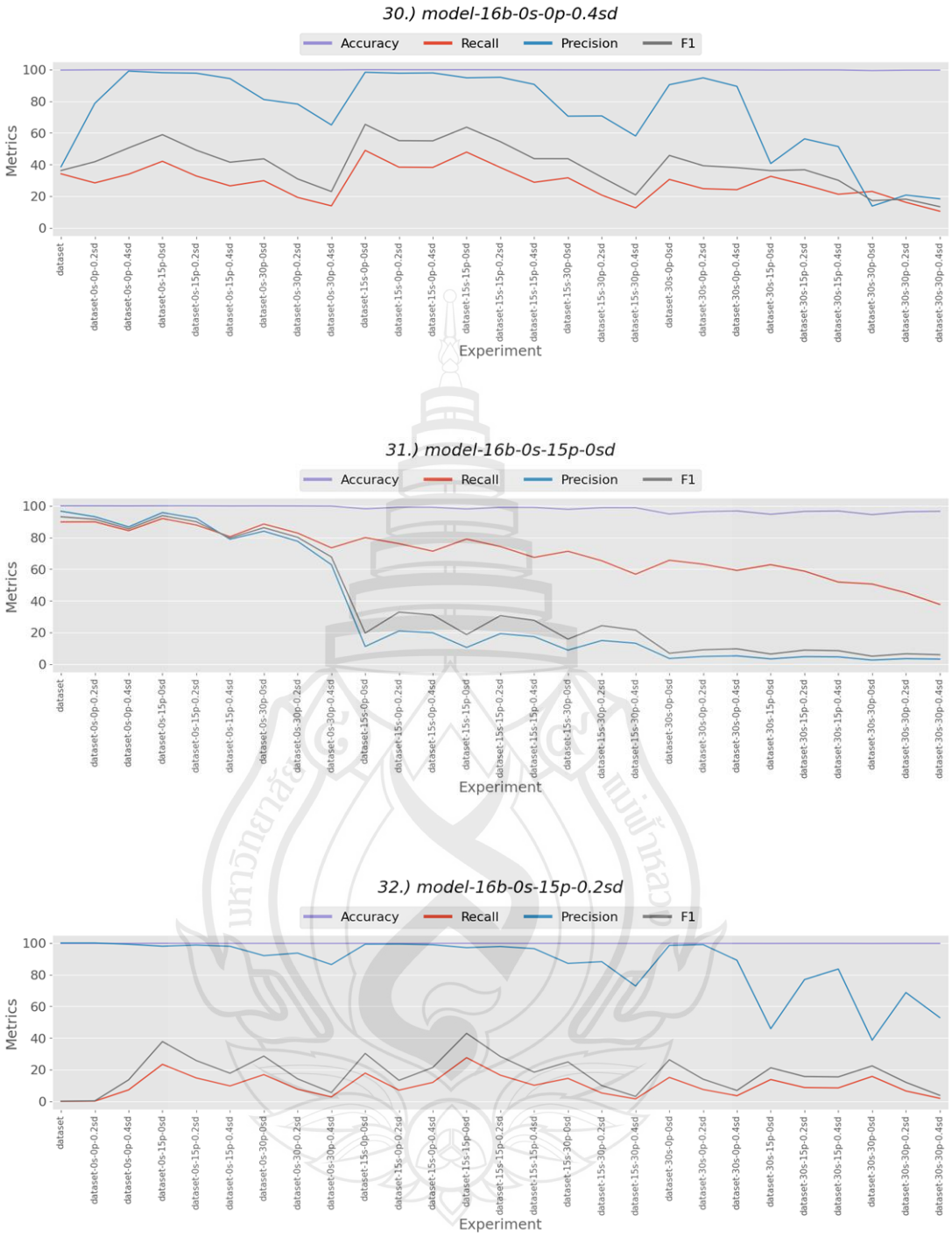

Figure E1 (continued)

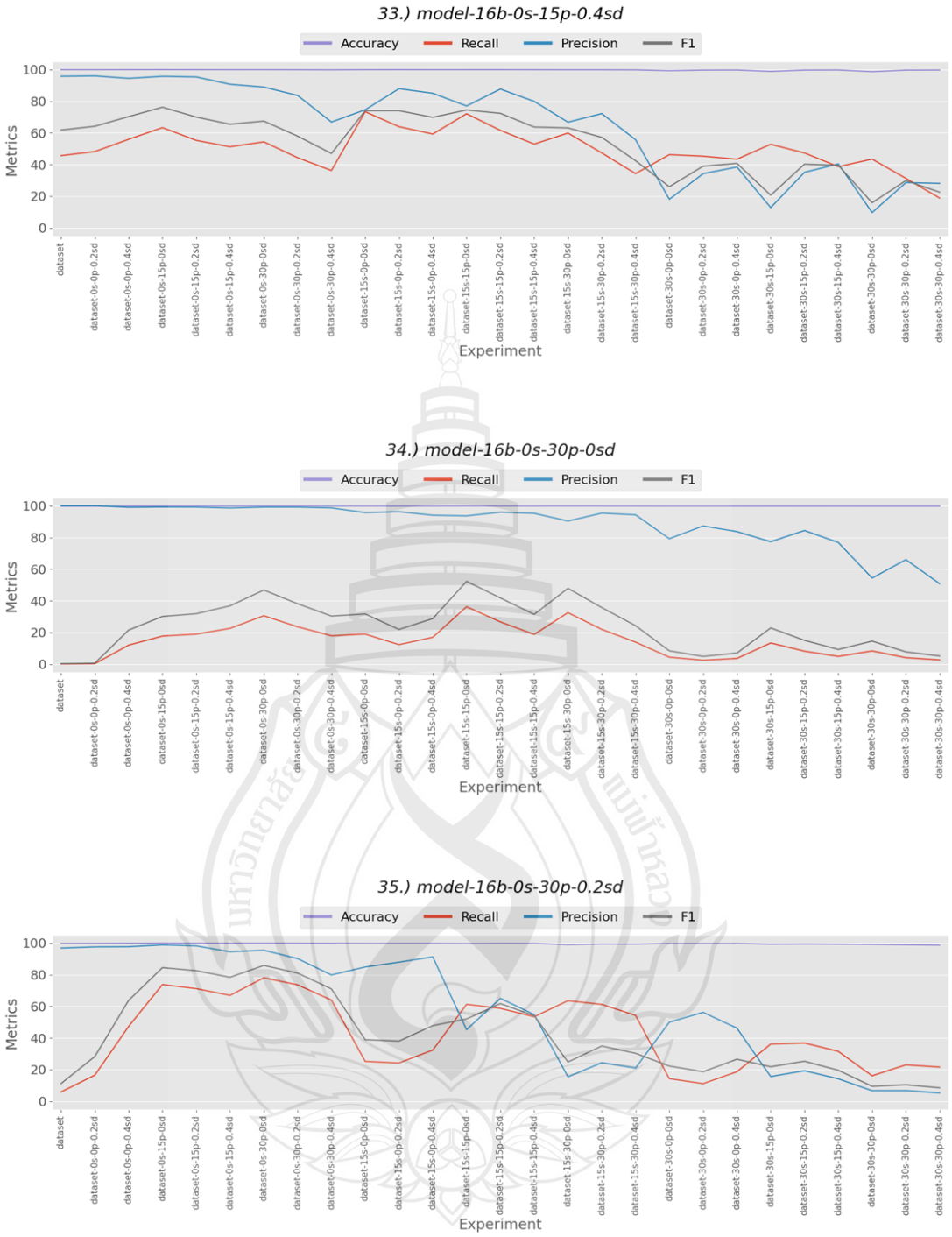

Figure E1 (continued)

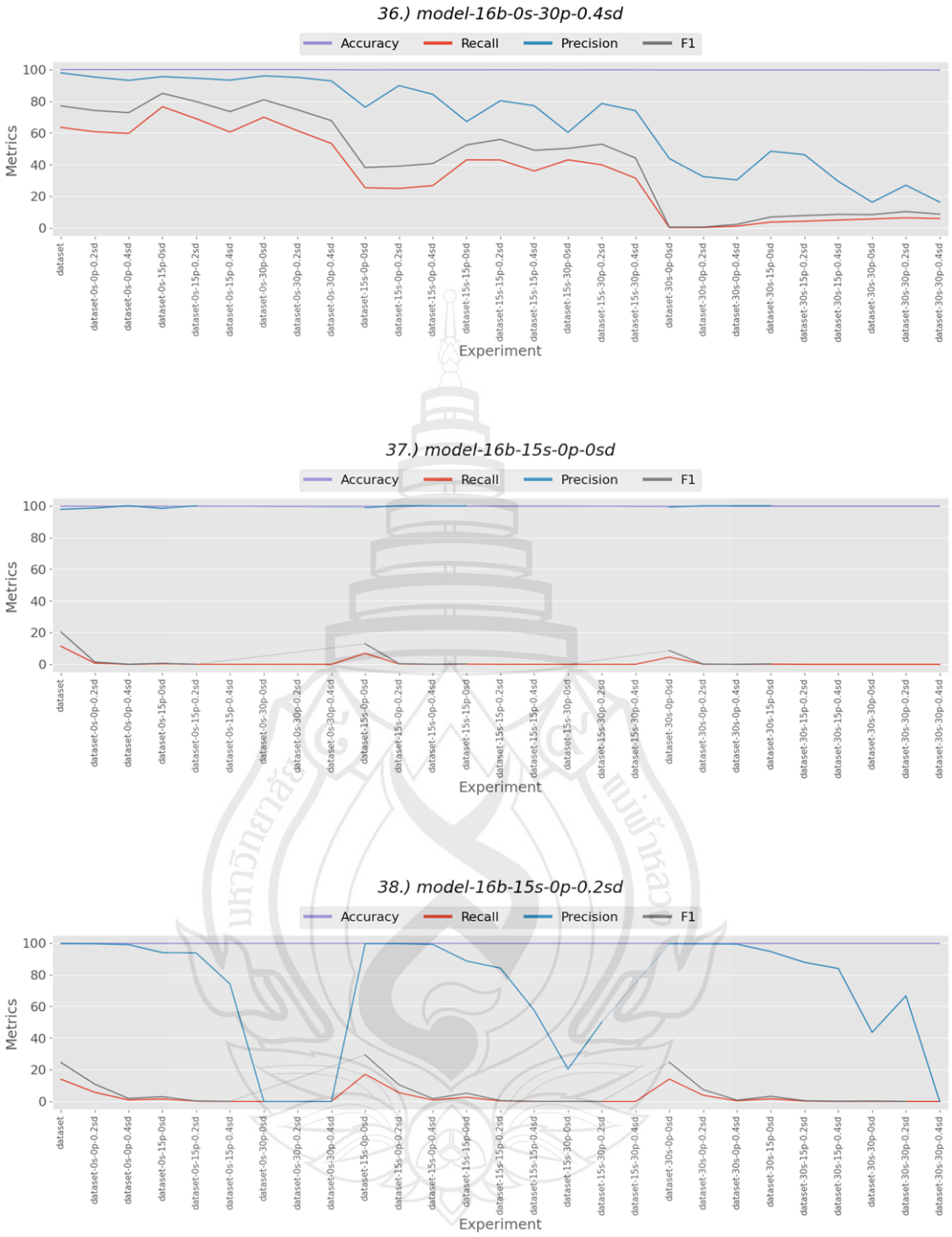

Figure E1 (continued)

Figure E1 (continued)

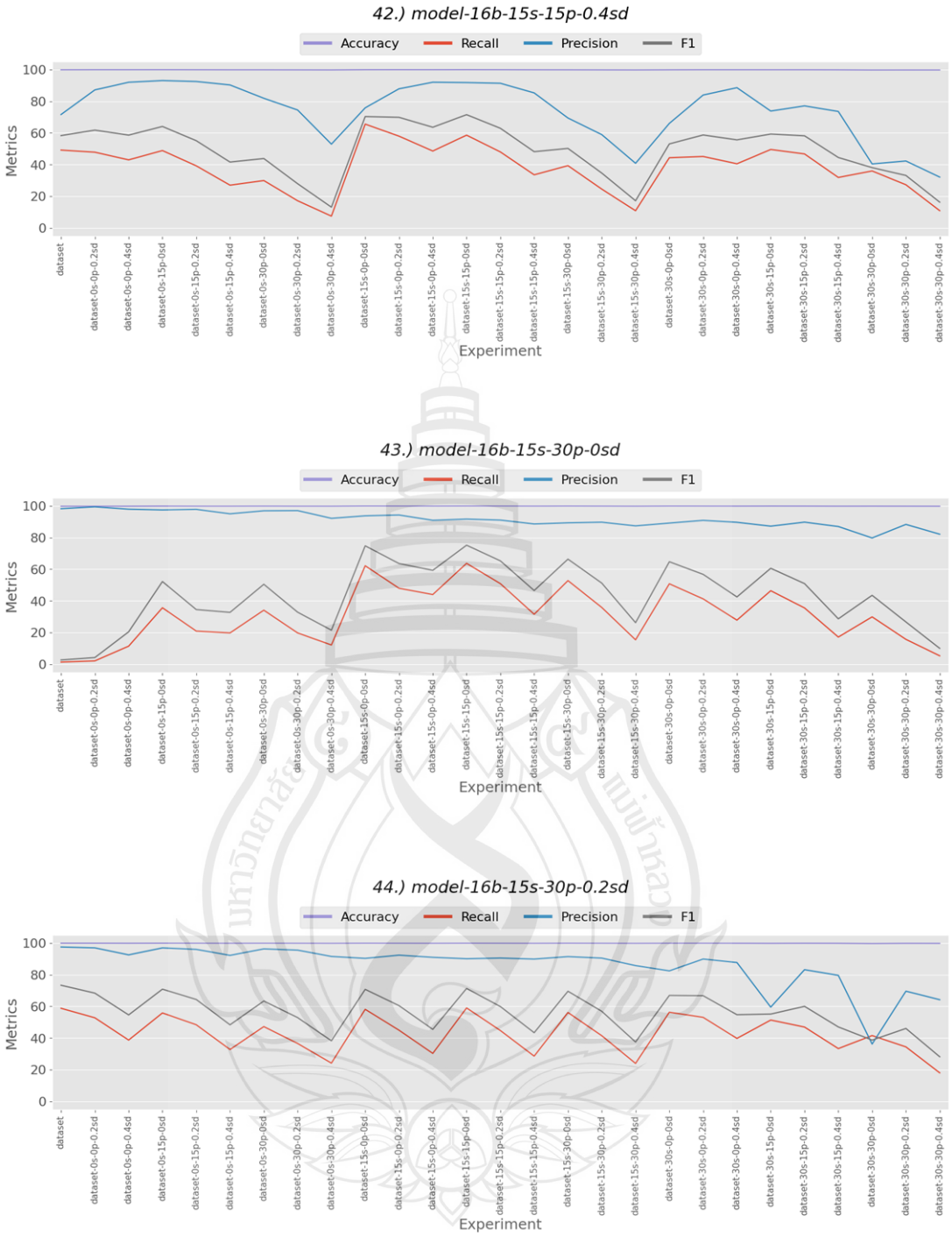

Figure E1 (continued)

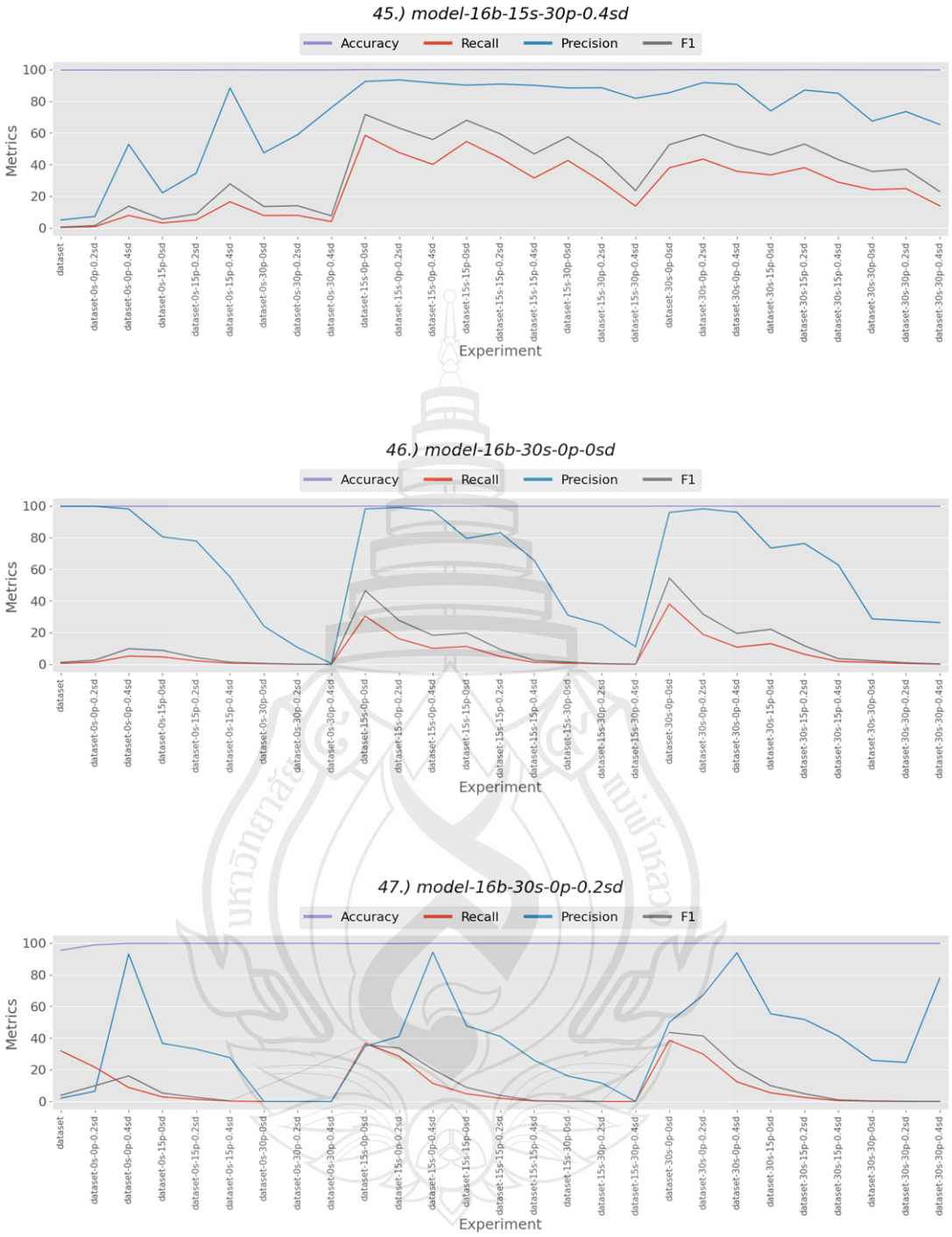

Figure E1 (continued)

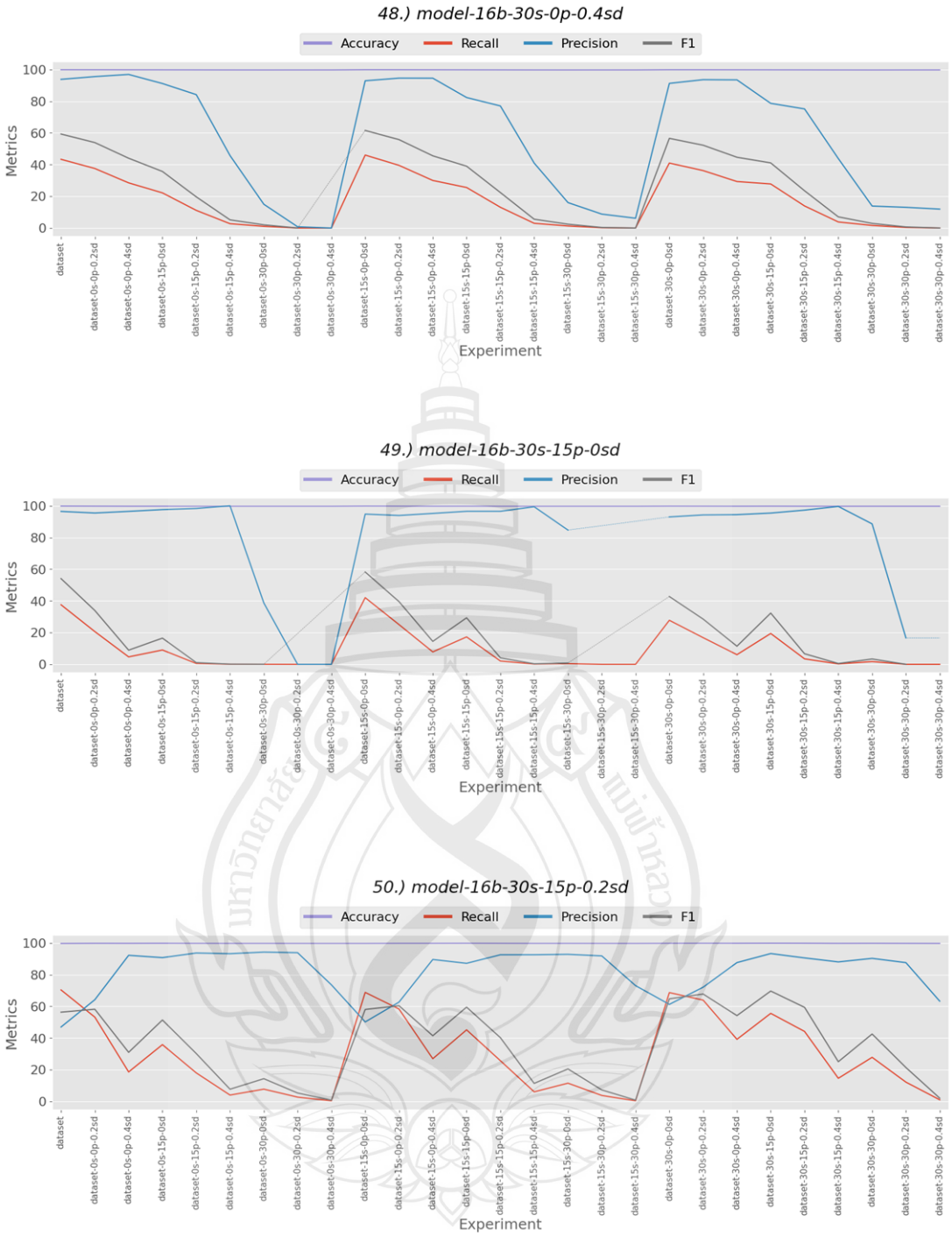

Figure E1 (continued)

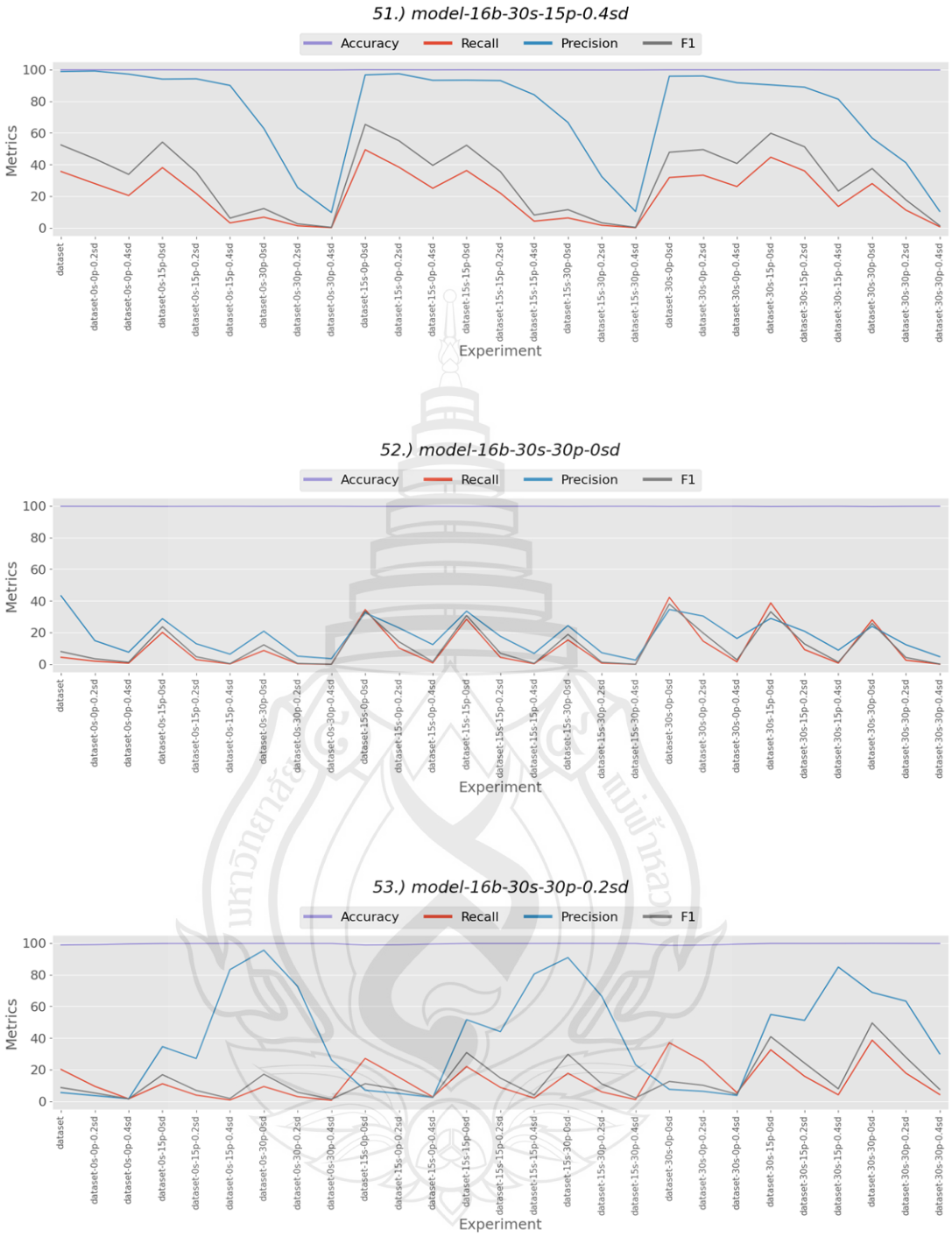

Figure E1 (continued)

Figure E1 (continued)

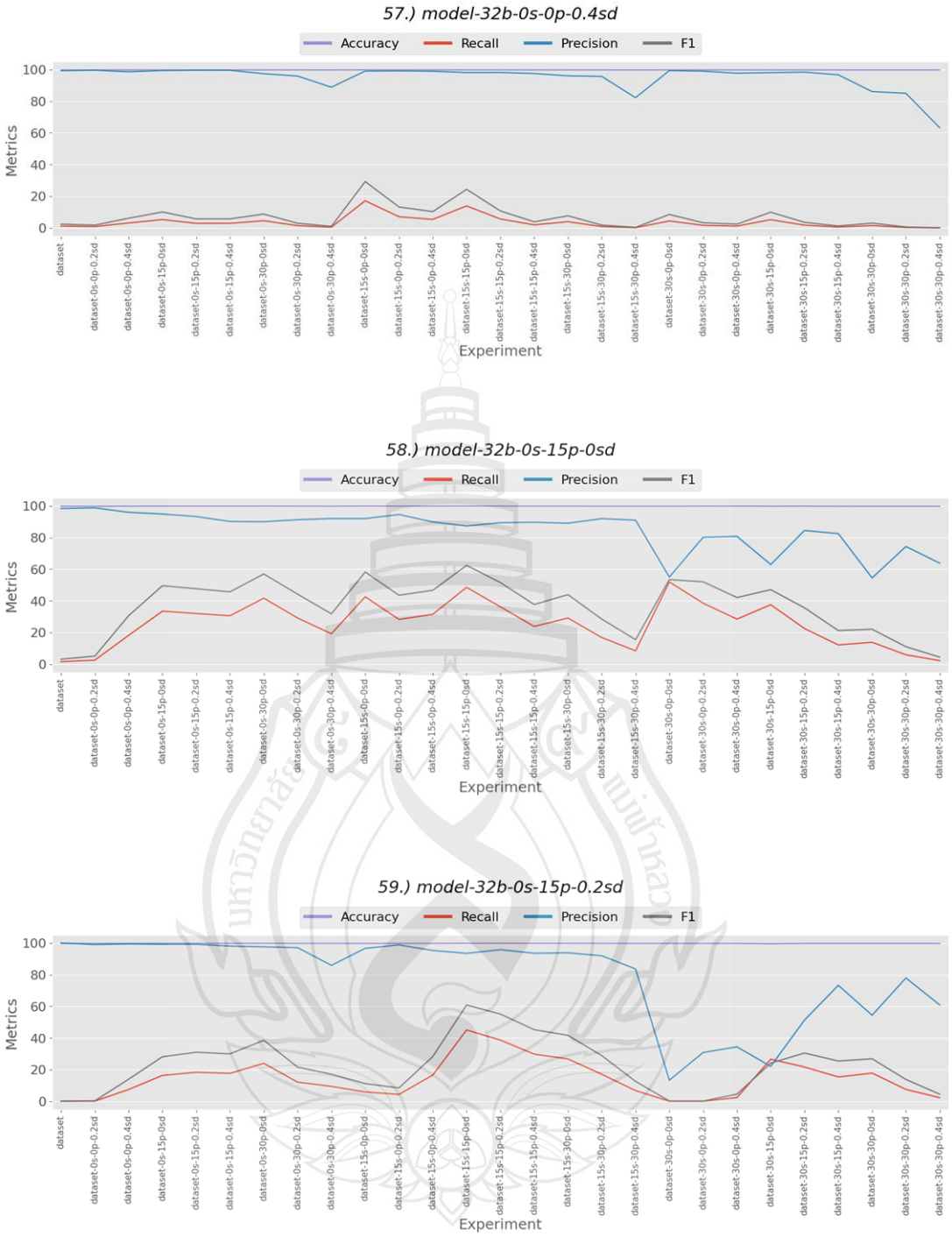

Figure E1 (continued)

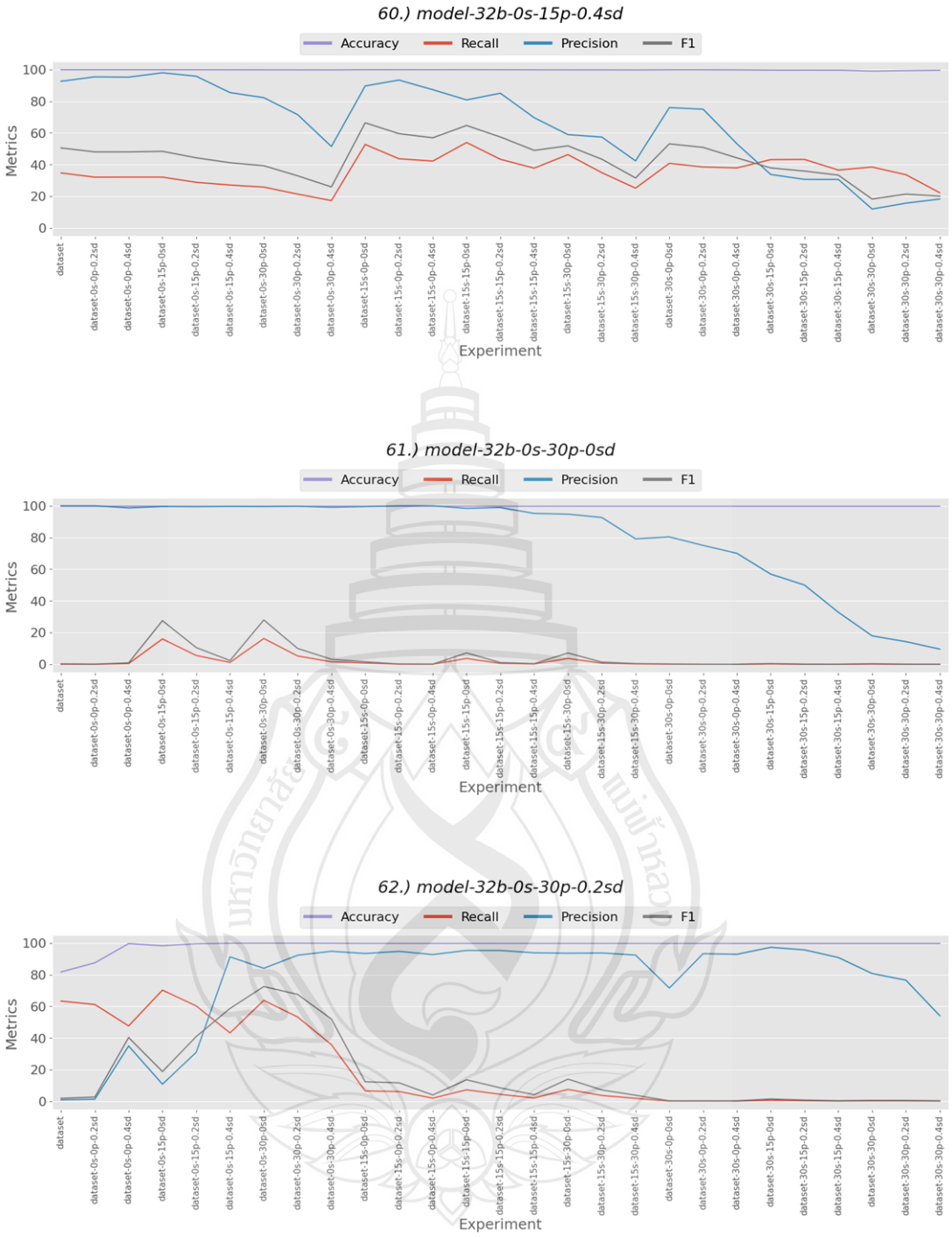

Figure E1 (continued)

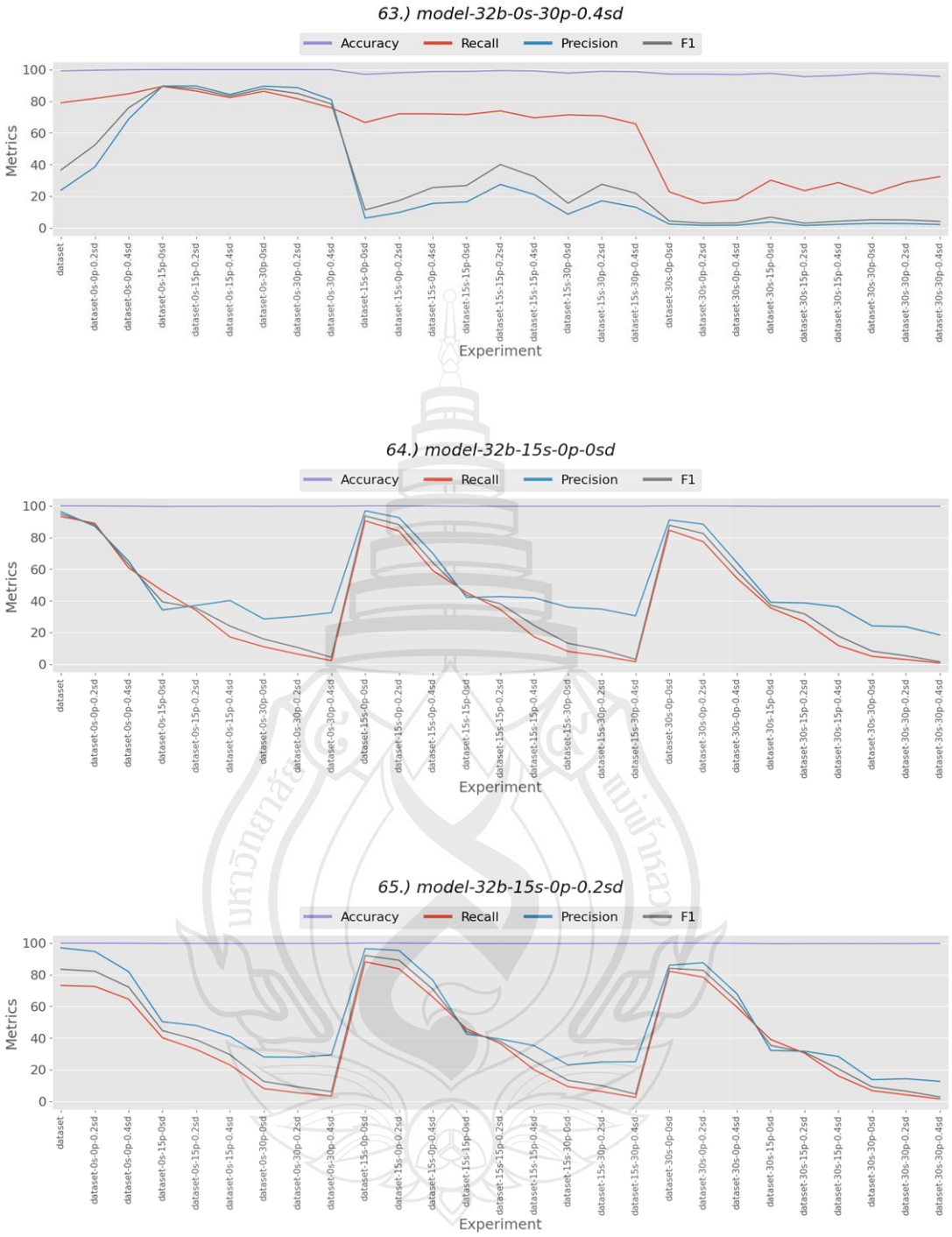

Figure E1 (continued)

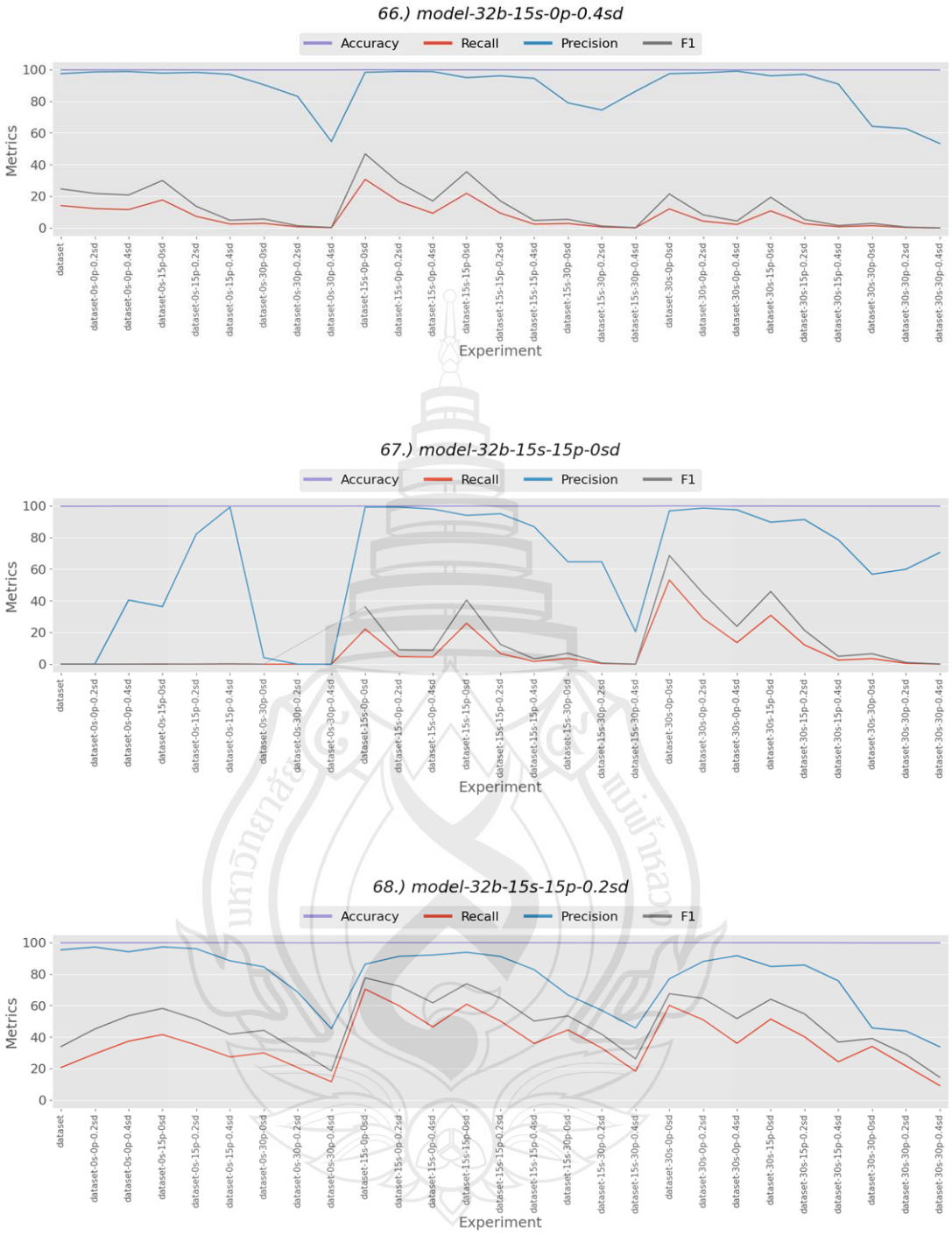

Figure E1 (continued)

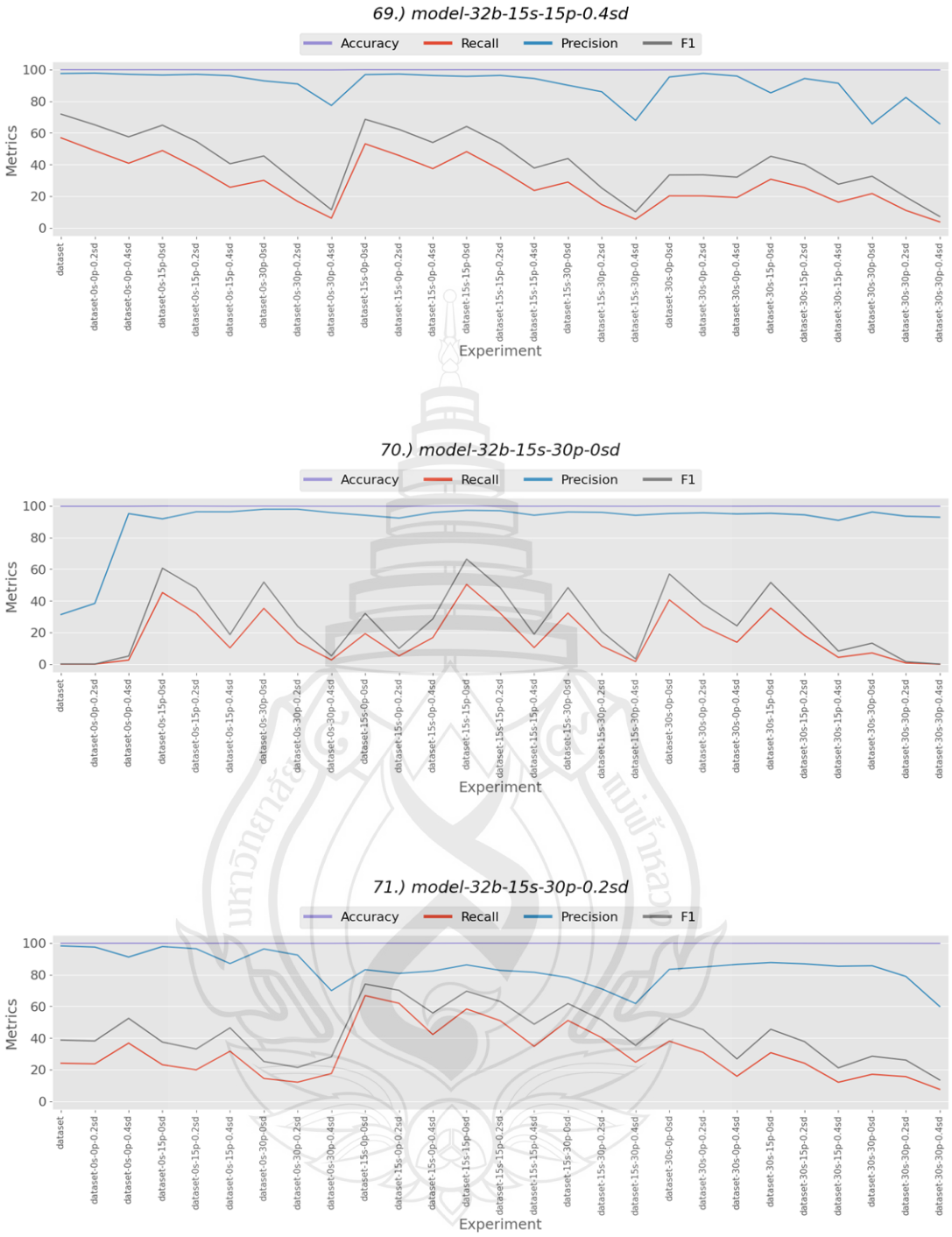

Figure E1 (continued)

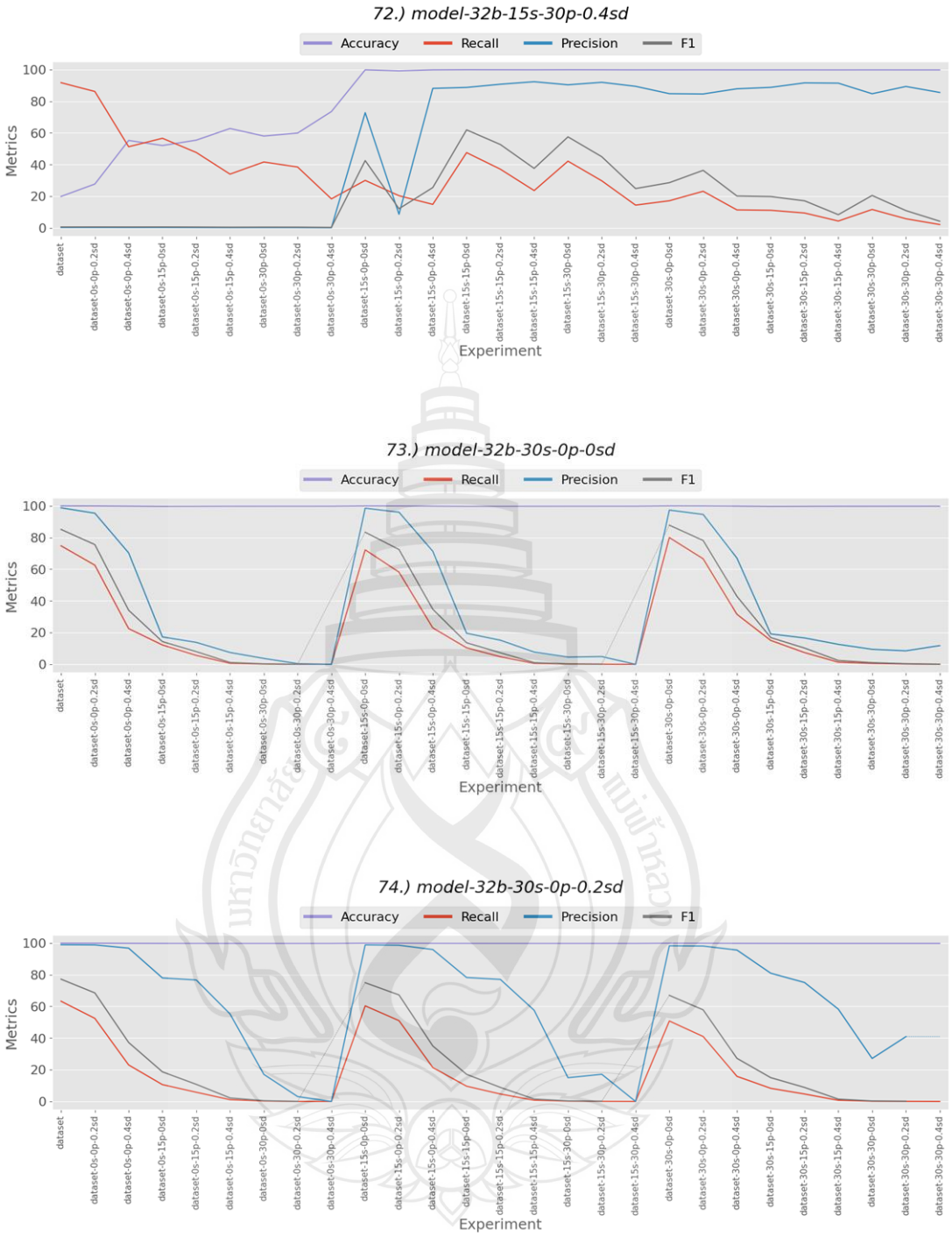

Figure E1 (continued)

Figure E1 (continued)

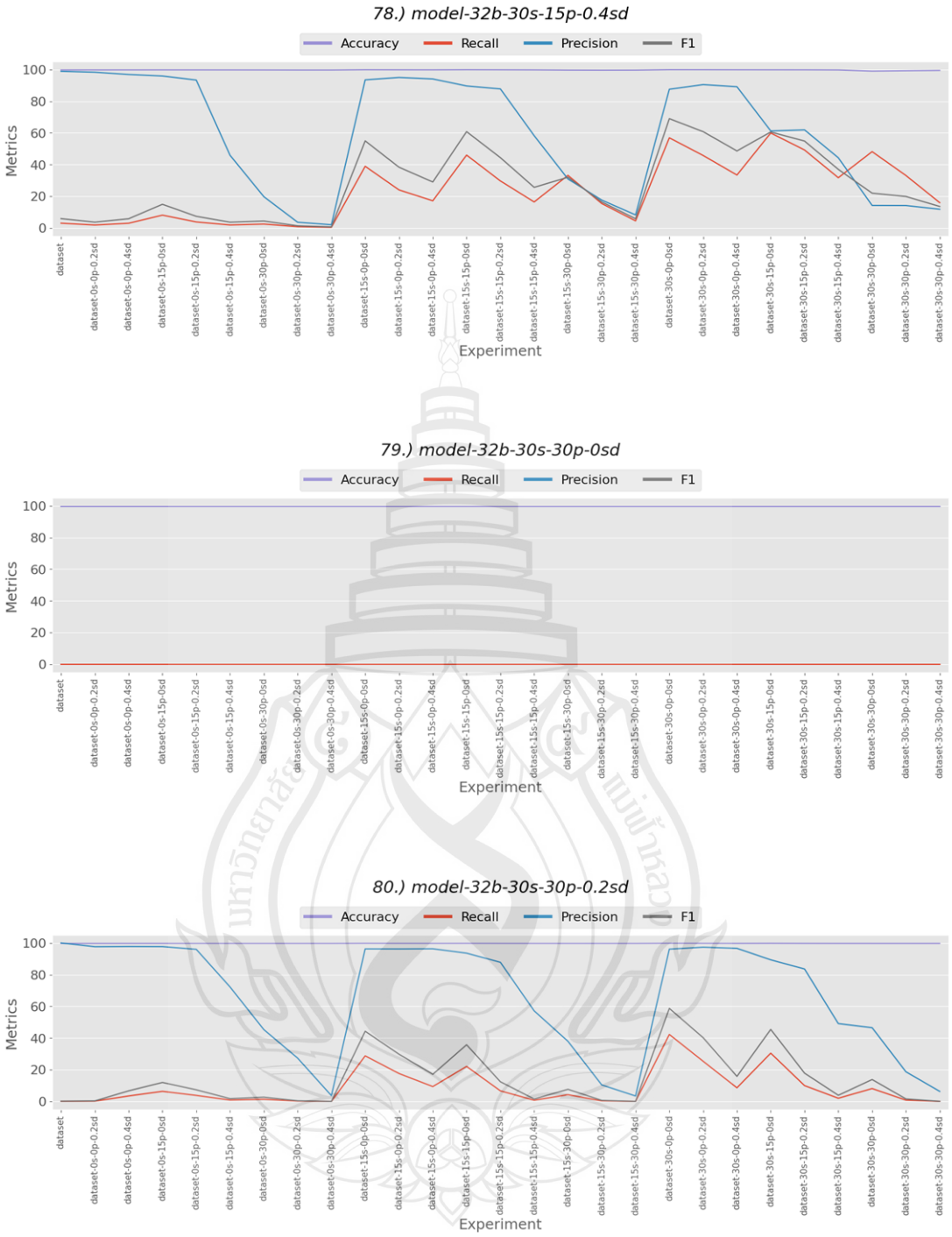

Figure E1 (continued)

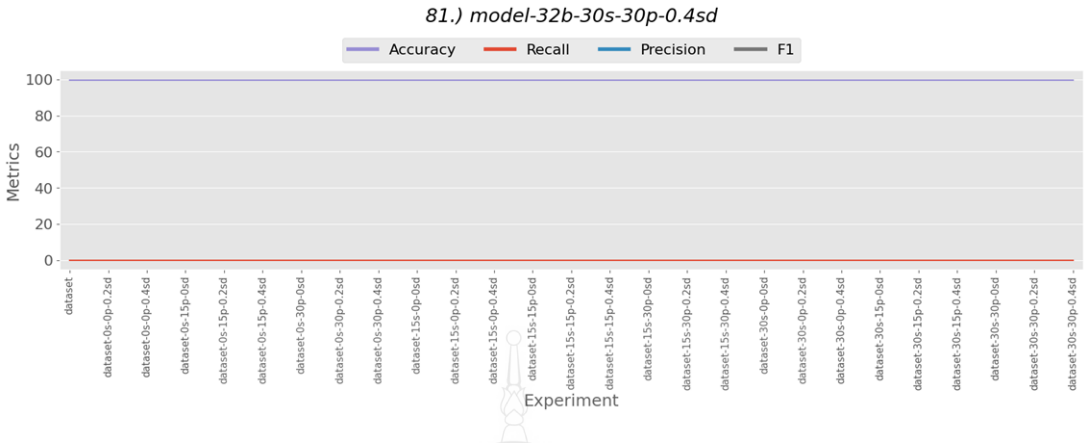

Figure E1 (continued)

APPENDIX F

PERFORMANCE CLASSIFIED BY DATASET COMBINATIONS

Here are 2,187 outcomes (with 50% threshold), showing evaluation metrics—accuracy, recall, precision, and F1—separated into 27 groups according to the Appendix D. Each group contains 81 outcomes according to the model combination on the Appendix C.

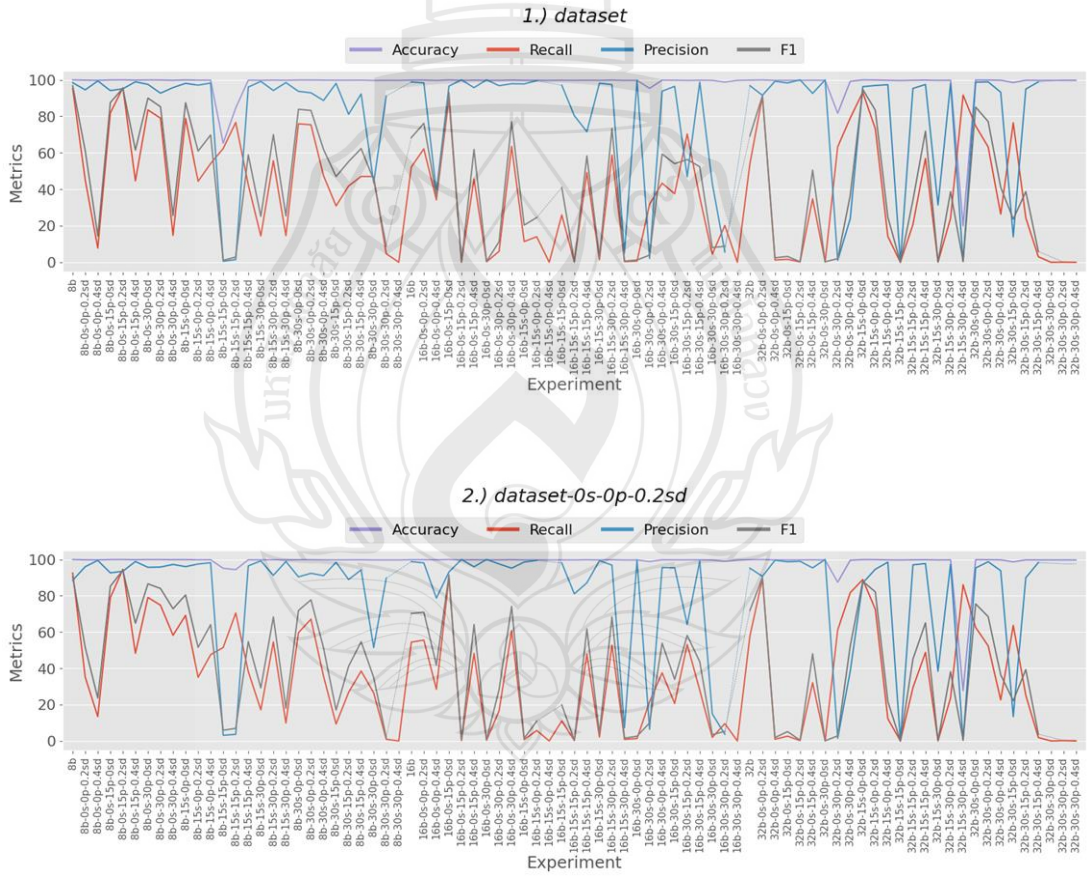

Figure F1 Performance Classified by Dataset Combinations

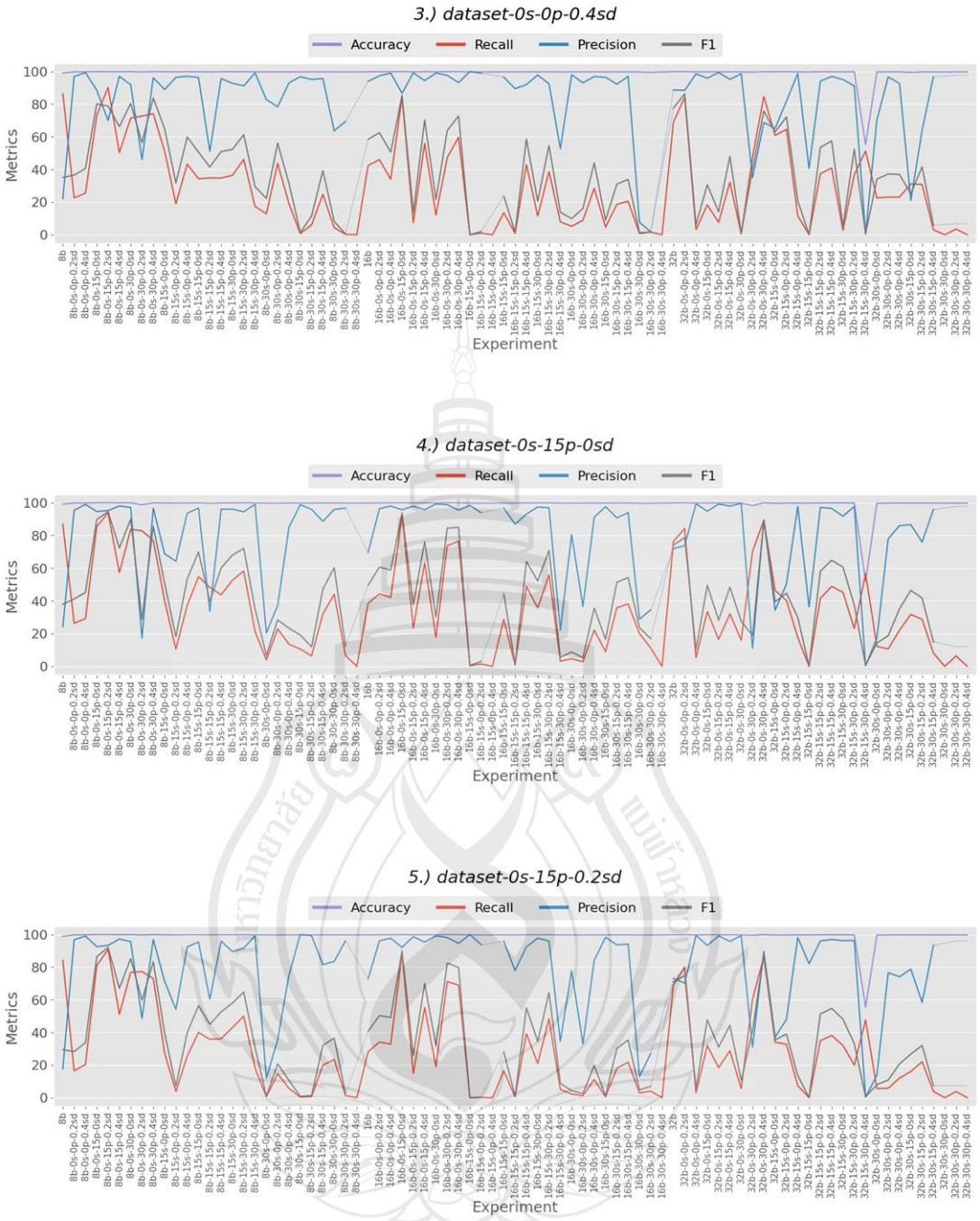

Figure F1 (continued)

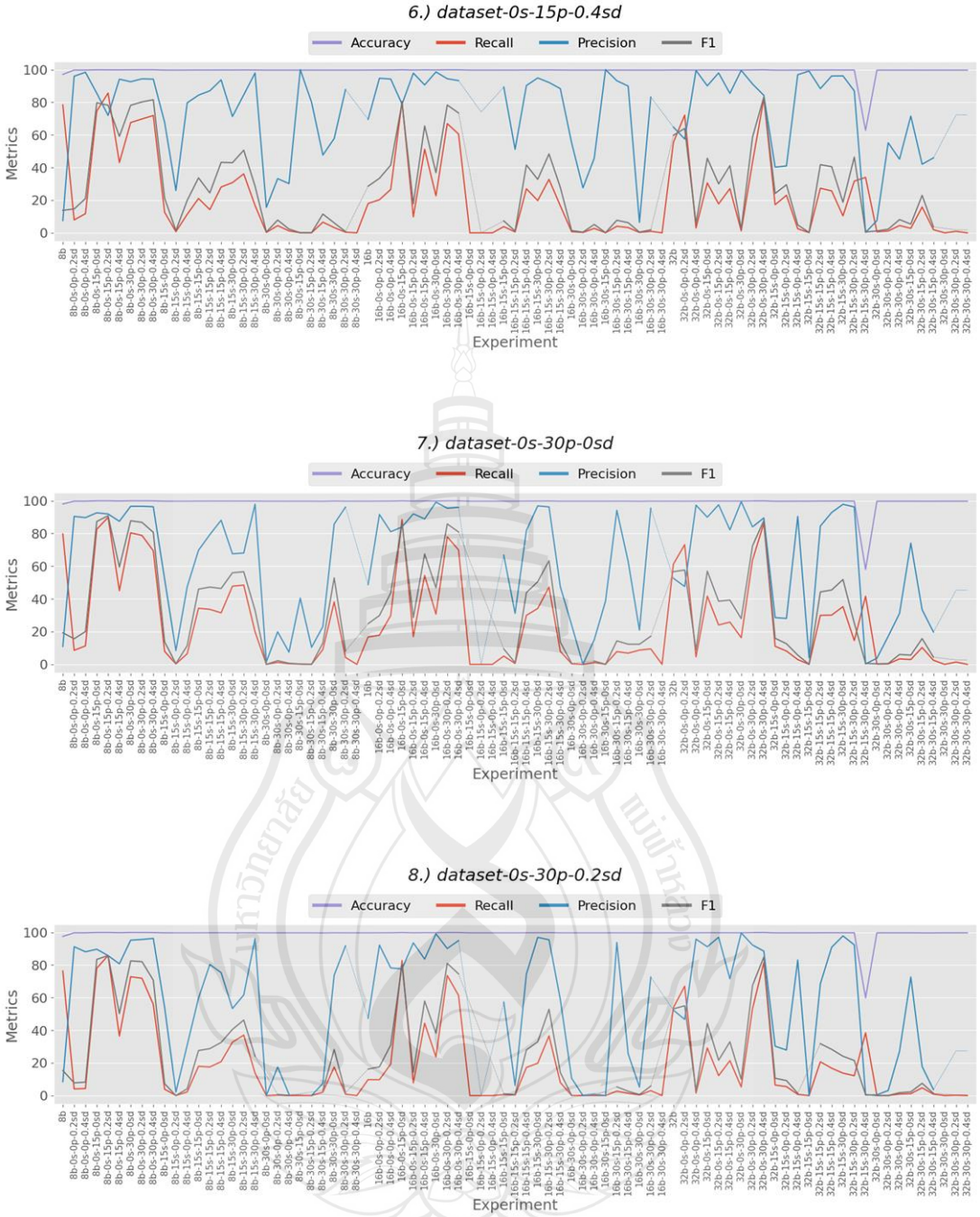

Figure F1 (continued)

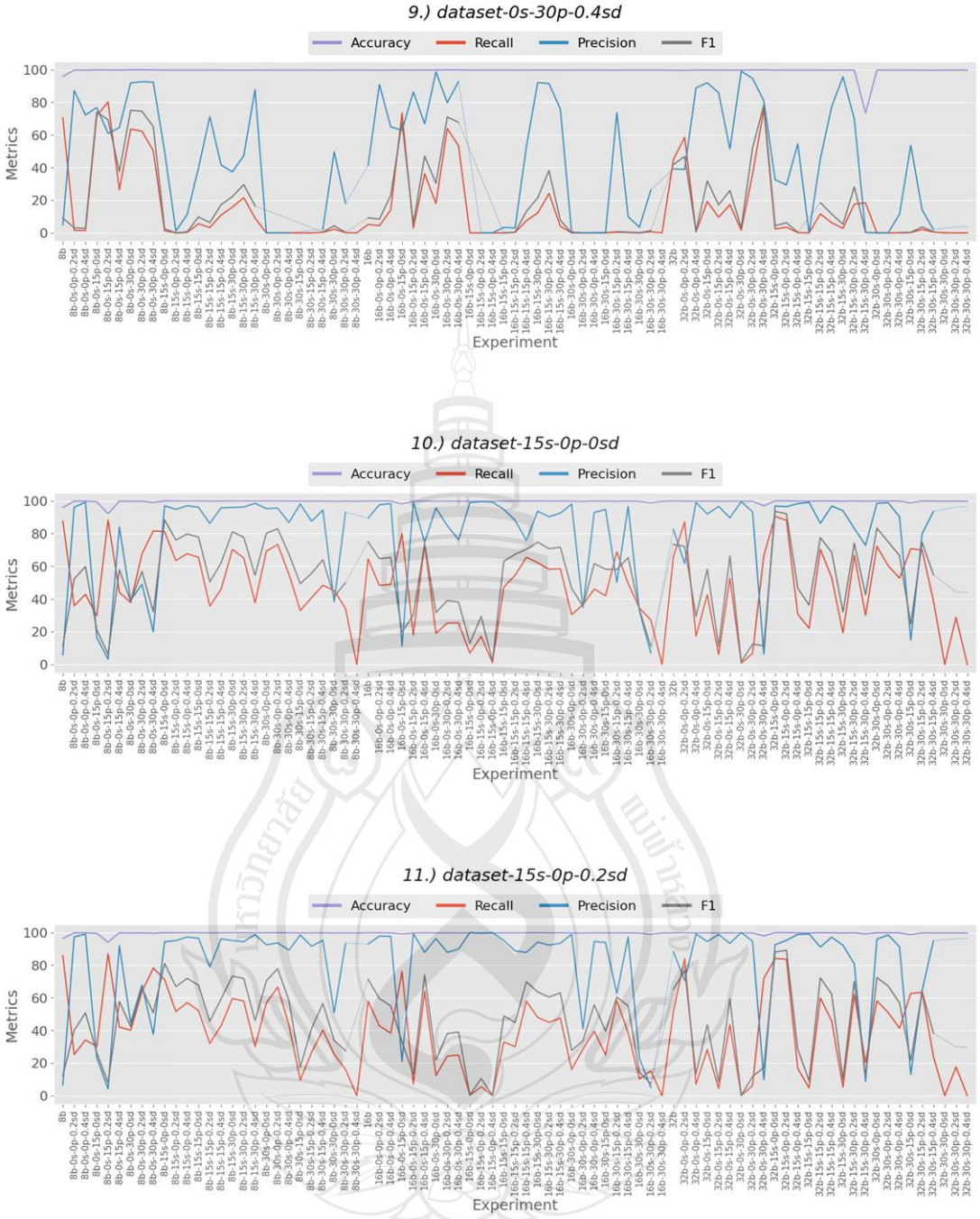

Figure F1 (continued)

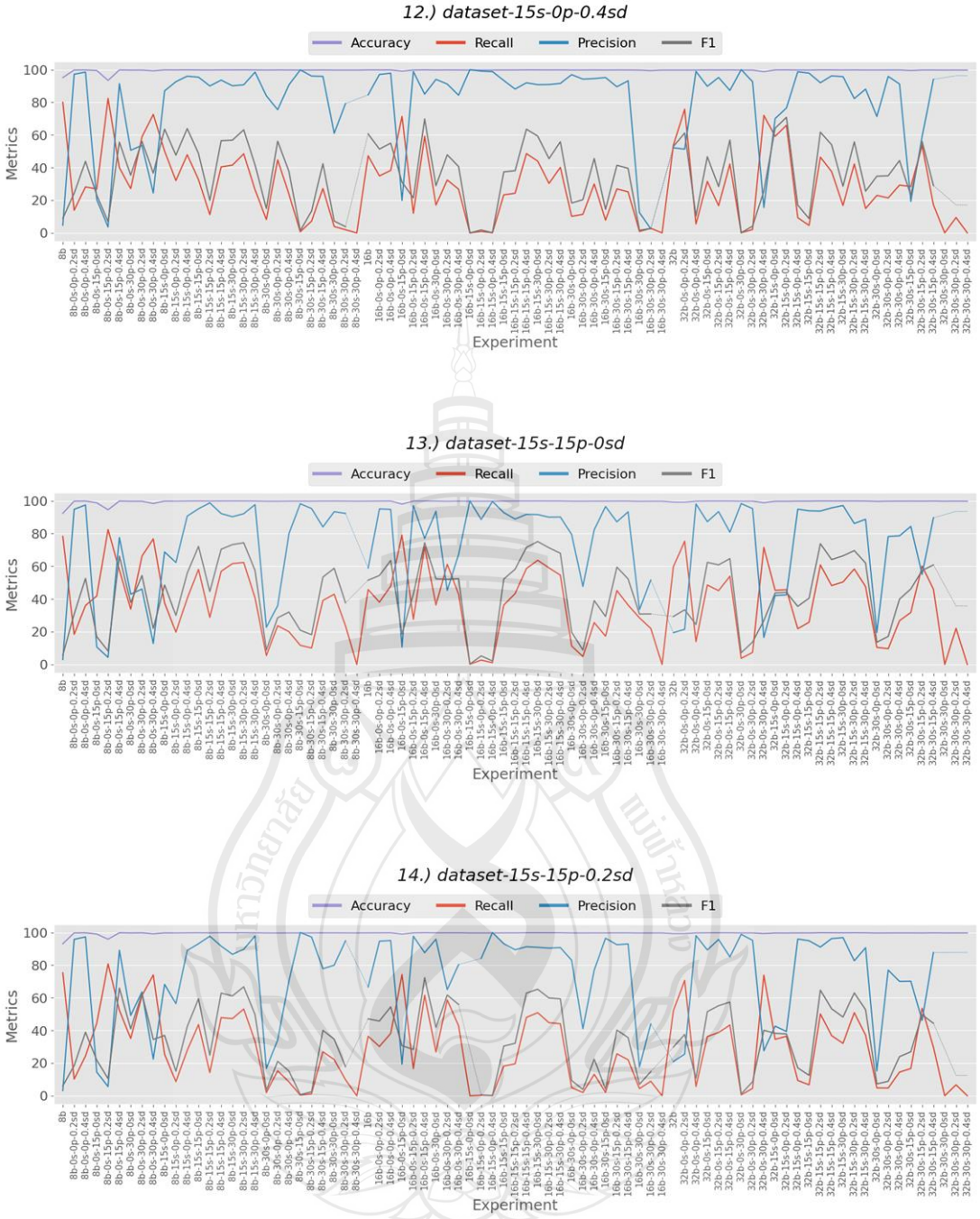

Figure F1 (continued)

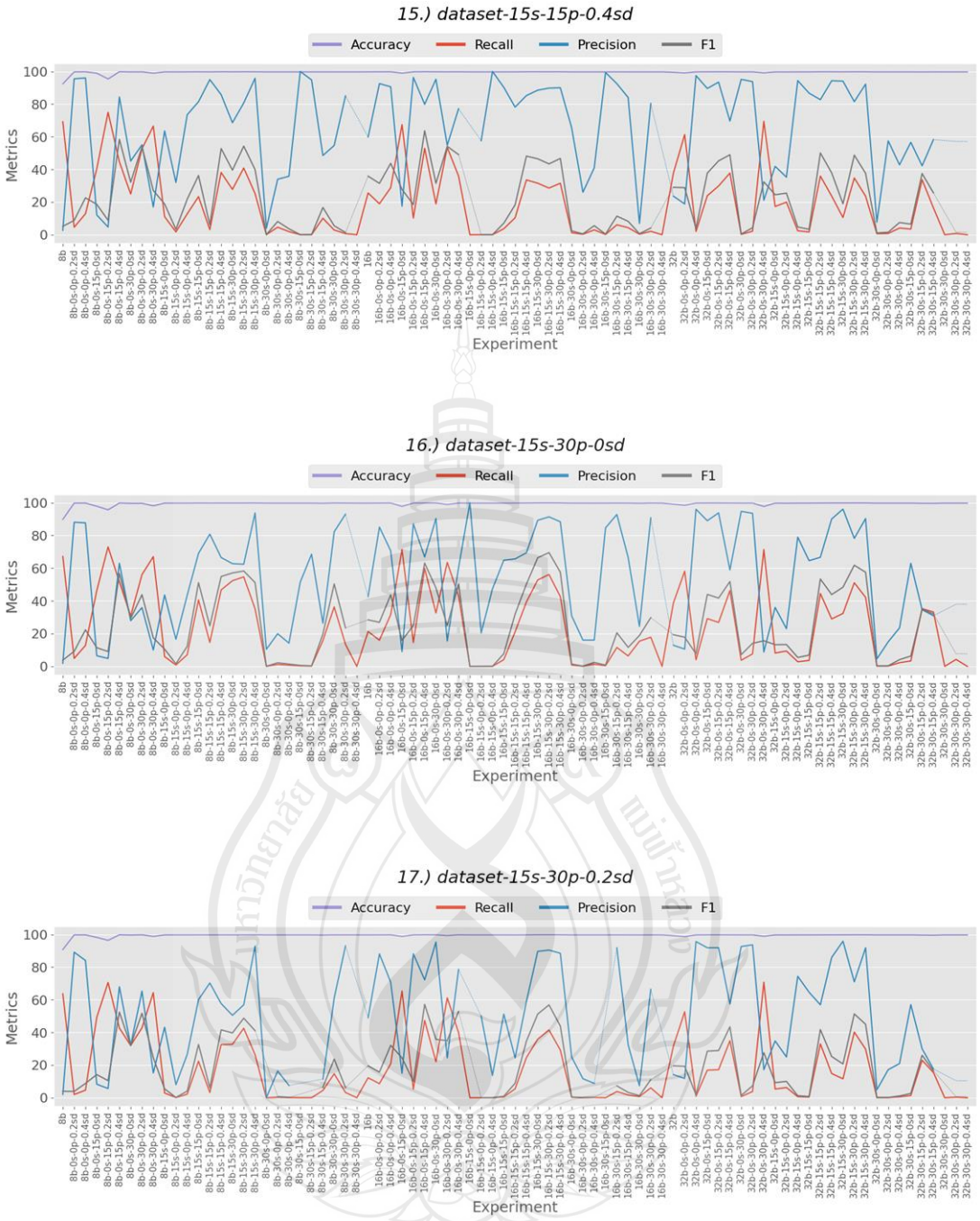

Figure F1 (continued)

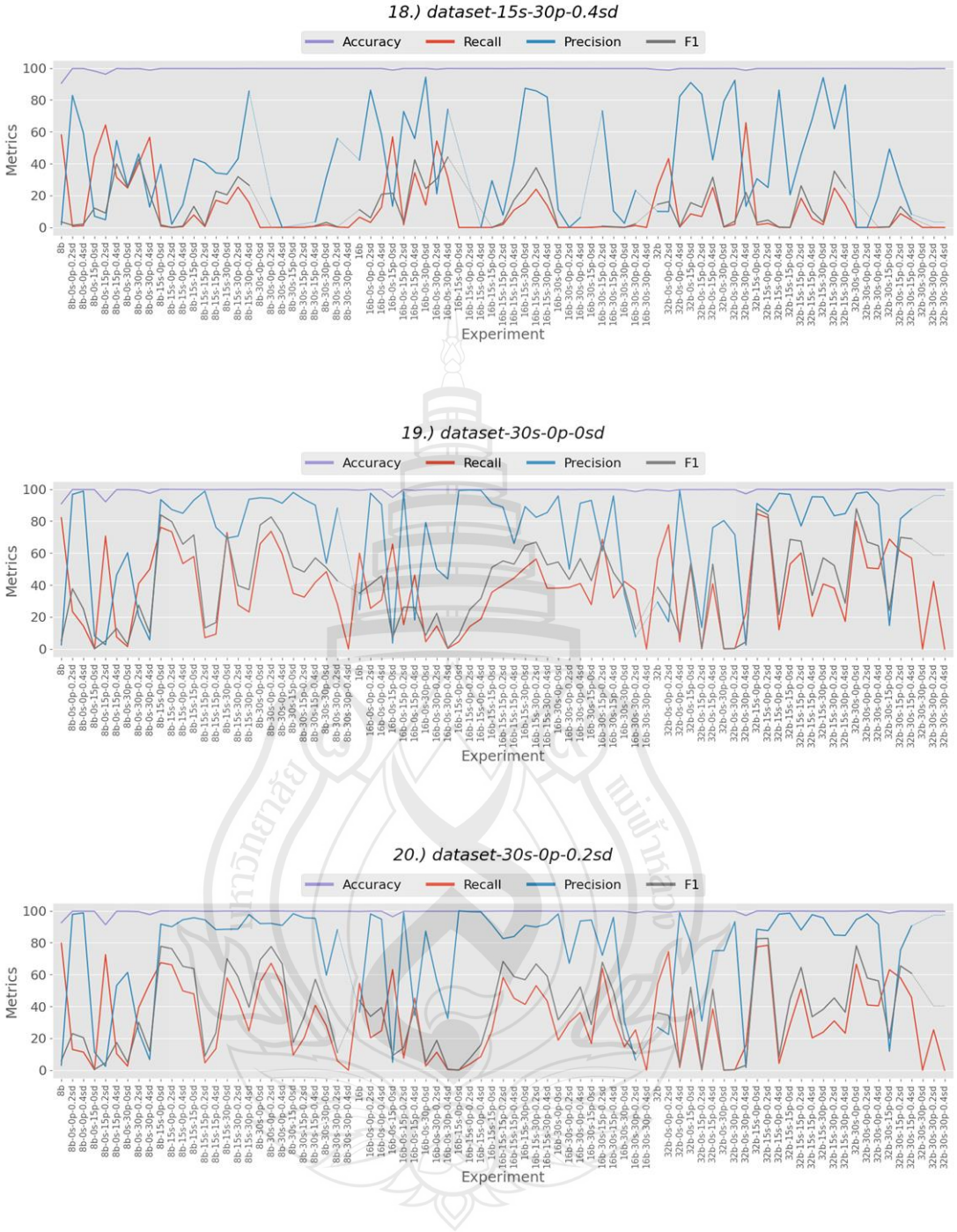

Figure F1 (continued)

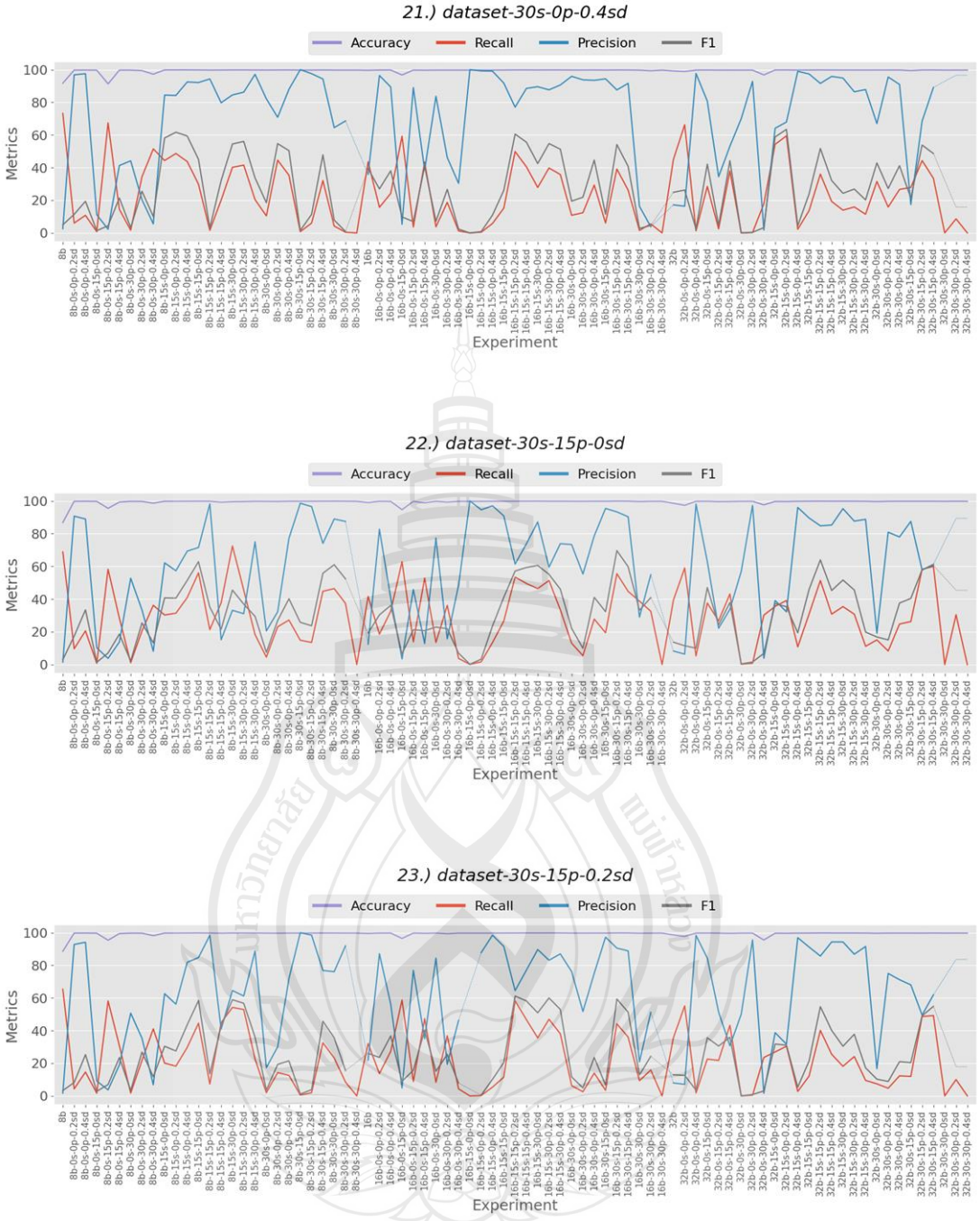

Figure F1 (continued)

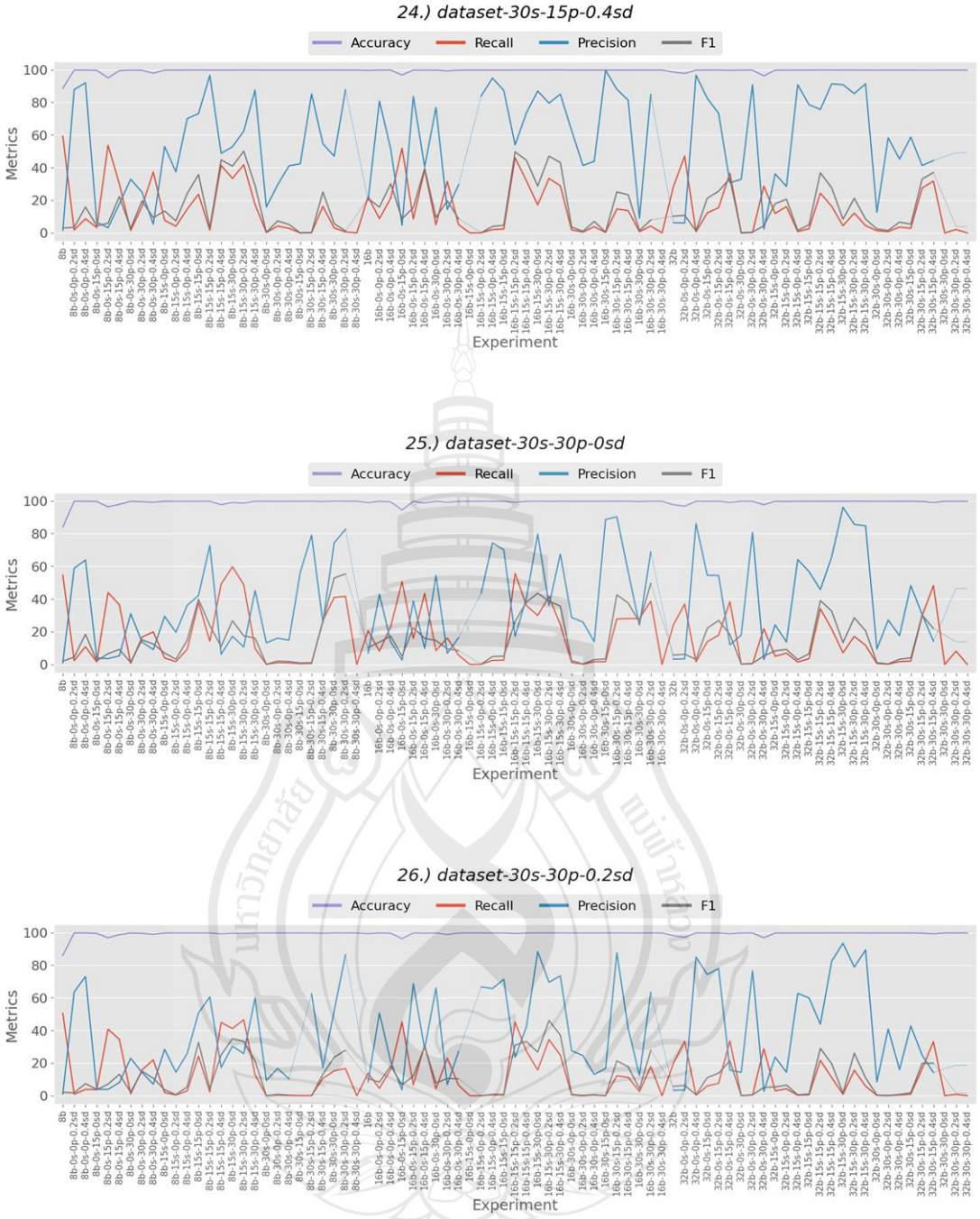

Figure F1 (continued)

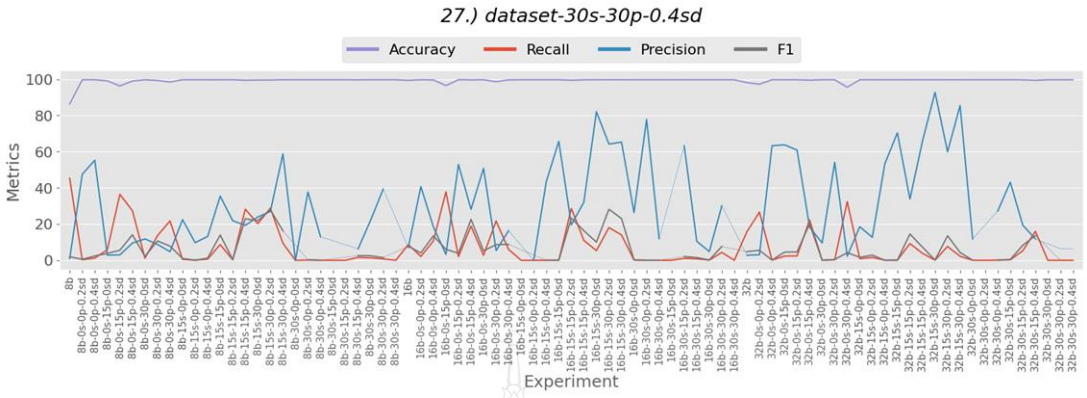

Figure F1 (continued)

APPENDIX G

SEARCH AND FILTRATION SYSTEM

This part shows a result of running OpenCV's connectedComponentsWithStats function on a feature map. The setup is shown as follows:

```
def find_obj(arr, max, min, filter=None, locate=False, verbose=False):
    '''
    Use this method to assign ID tags and return a number of objects appearing in an array.
    It also returns centroids for each object if "locate" is true.
    filter: "KEEP" for keeping objects larger or smaller than max/min parameters and assign them to the
last ID,
    or "REMOVE" to reset them to zero. The size (max/min) of objects in the mask for this dataset is
49.
    Input arrays must have the datatype of uint8.
    '''
    map_base = cv2.connectedComponentsWithStats(arr)
    if filter is not None:
        assert filter == 'KEEP' or filter == 'REMOVE', 'Value for attr "filter" of "'+filter+'" is not in the
list of allowed values: "KEEP", "REMOVE"'
        map_mod = np.copy(map_base[1])
    print('This map contains', map_base[0]-1, 'objects\n')
    res = [0,0,0]
    for i in range(map_base[0]):
        if verbose:
            if i == 0:
                print('Background covers', map_base[2][i,4], 'areas')
            else:
                print('Tag #'+str(i)+' covers', map_base[2][i,4], 'areas')
        if map_base[2][i,4] > max and i != 0:
            res[1] += 1
            if filter is not None:
                map_mod[map_mod == i] = map_base[0]
        elif map_base[2][i,4] < min and i != 0:
            res[2] += 1
            if filter is not None:
                map_mod[map_mod == i] = map_base[0]
        else:
            if i != 0:
                res[0] += 1
    if verbose:
        print()
    print('Found', res[1], 'of', map_base[0]-1, 'bigger than', max, 'areas')
```

Figure G1 Python code for setting up a filtration system

```
    print('Found', res[2], 'of', map_base[0]-1, 'smaller than', min, 'areas')
    print('Found', res[0], 'of', map_base[0]-1, 'with proper size')
    if filter is not None and (res[1] != 0 or res[2] != 0):
        if filter=='KEEP':
            map_keep = np.where(map_mod != map_base[0], 0, res[0]+1)
        map_mod[map_mod == map_base[0]] = 0
        map_mod[map_mod > 0] = 1
        print('New map created', end = '')
        map_mod = cv2.connectedComponentsWithStats(map_mod.astype(np.uint8))
        if filter=='KEEP':
            print(' with the filtered objects at tag #' + str(res[0]+1))
            if locate:
                return map_mod[1] + map_keep, map_mod[3]
            else:
                return map_mod[1] + map_keep
        else:
            print(' with the filtered objects removed')
            if locate:
                return map_mod[1], map_mod[3]
            else:
                return map_mod[1]
    else:
        if locate:
            return map_base[1], map_base[3]
        else:
            return map_base[1]
```

Figure G1 (continued)

The code in Figure G1 defines a function to let OpenCV find a cluster of 1 in arrays which represents a single object. The function can be easily used by giving maximum and minimum sizes of targeted objects as shown on Figure G2.

```
res_preds_map, res_preds_loc = find_obj(res_preds_full, OBJ_MAX, OBJ_MIN, filter='KEEP',
locate=True, verbose=False)
temp = y_loc[1:]
y_loc[1:] = temp[np.lexsort(np.fliplr(temp).T)]
temp = res_preds_loc[1:]
res_preds_loc[1:] = temp[np.lexsort(np.fliplr(temp).T)]
del temp
y_loc, res_preds_loc = y_loc.round(1), res_preds_loc.round(1)
print('\nObject coordinates (0 is background):')
print(pandas.DataFrame({'X': res_preds_loc[...,0], 'Y': res_preds_loc[...,1]}))
```

Figure G2 Python code for filtering the objects

We can count and geo-track the result. In this case, it looks for objects that cover 36-49 areas.

```
Counting objects in ground truth: This map contains 2065 objects

Found 16 of 2065 bigger than 49 areas
Found 16 of 2065 smaller than 36 areas
Found 2033 of 2065 with proper size
New map created with the filtered objects at tag #2034

Counting objects in prediction: This map contains 1038 objects

Found 0 of 1038 bigger than 49 areas
Found 1005 of 1038 smaller than 36 areas
Found 33 of 1038 with proper size
New map created with the filtered objects at tag #34

Object coordinates (0 is background):
       X       Y
0   2943.5  2943.5
1    118.9  1512.4
2    123.5  5719.1
3    127.6  3987.1
4    163.5  5231.5
5    173.7  3522.1
6    278.1   283.6
7    284.1  1604.4
.
.
.
```

Figure G3 An example of a Python output

Then, we can inspect each object as we feel interested, as shown on Figure G4.

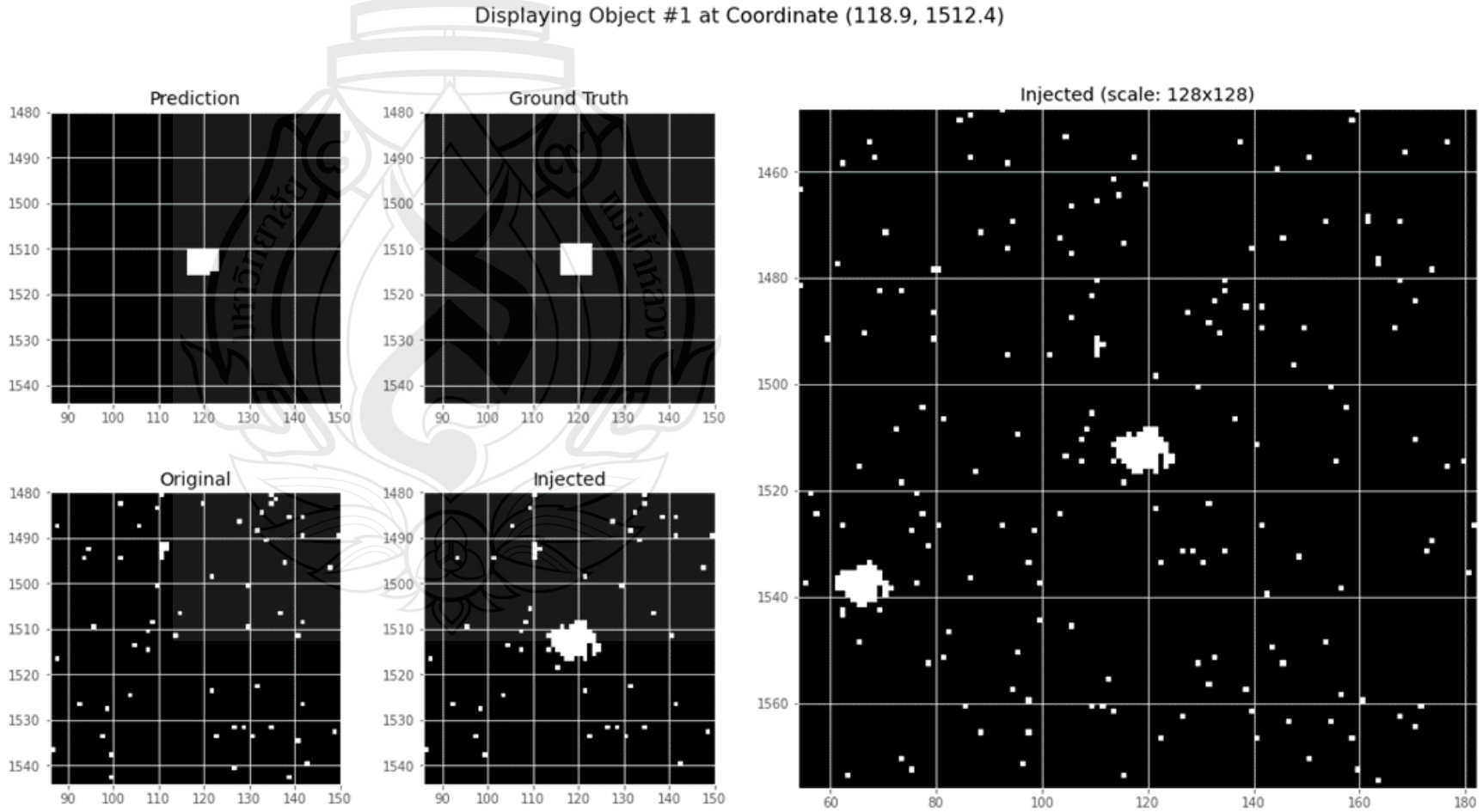

Figure G4 An example of a filtered object