PATTERN RECOGNITION OF PARTIAL DISCHARGE FAULTS USING CONVOLUTIONAL NEURAL NETWORK (CNN)

BY

JAKRIN BUTDEE

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF MASTER OF ENGINEERING
(ARTIFICIAL INTELLIGENCE AND INTERNET OF THINGS)
SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY
THAMMASAT UNIVERSITY
ACADEMIC YEAR 2022

THAMMASAT UNIVERSITY

SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY

THESIS

BY

JAKRIN BUTDEE

ENTITLED

PATTERN RECOGNITION OF PARTIAL DISCHARGE FAULTS USING CONVOLUTIONAL NEURAL NETWORK (CNN)

was approved as partial fulfillment of the requirements for
the degree of Master of Engineering (Artificial Intelligence and Internet of Things)

On June 14, 2023

Chairperson N. Chayopitak
(Nattapon Chayoptiak, Ph.D.)

Member and Advisor
(Associate Professor Waree Kongprawechnon, Ph.D.)

Member
(Seksan Laitrakun, Ph.D.)

Member
(Associate Professor Hiroki Nakahara, Ph.D.)

Director
(Professor Pruettha Nanakorn, D.Eng.)

Thesis Title	PATTERN RECOGNITION OF PARTIAL DISCHARGE FAULTS USING CONVOLUTIONAL NEURAL NETWORK (CNN)
Author	Jakrin Butdee
Degree	Master of Engineering (Artificial Intelligence and Internet of Things)
Faculty/University	Sirindhorn International Institute of Technology/ Thammasat University
Thesis Advisor	Associate Professor Waree Kongprawechnon, Ph.D.
Academic Years	2022

ABSTRACT

Partial Discharge (PD) analysis is one the most widely used methods to monitor and determine the fault conditions of electrical equipment, especially in high-voltage environments such as power transformers and power generators. Conventional method of PD analysis that is widely used in multiple studies and commercial equipment usually rely on a feature extraction technique such as the Phase Resolved Partial Discharge (PRPD) Pattern to assist PD experts to inspect the faults in the system. This study proposes a CNN based method to recognize the PRPD patterns for different types of PD. The differences of each type of PD, data pre-processing steps and visualization of PD waveforms in PRPD patterns are discussed in details. The obtained PRPD pattern images are then used to train a pattern recognition model and the results show that the proposed method can effectively classify different types of PD under consideration.

Keywords: pattern recognition, partial discharge analysis, fault diagnosis, machine learning.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Professor Waree Kongprawechnon for her invaluable guidance, support, and expertise throughout the duration of this study. Professor Kongprawechnon's unwavering commitment to excellence and dedication to her students have been an inspiration to me, and I am truly grateful for the opportunity to work under her mentorship. I would also like to extend my heartfelt appreciation to Dr. Nattapon Chayoptiak, the chairperson of this study. Dr. Chayoptiak's expertise, encouragement, and constructive feedback have been instrumental in shaping the direction of this research. His commitment to academic excellence and his invaluable guidance throughout the entire process have been greatly appreciated.

Lastly, I would like to acknowledge the contributions of all of the research project committee members who have provided constant support and encouragement throughout this endeavor. Their expertise, suggestions and their unwavering support have been instrumental in my personal and academic growth. I am deeply grateful to all those mentioned above and anyone else who has contributed to this study in any way.

This thesis was supported by the Thailand Advanced Institute of Science and Tech-nology (TAIST), National Science and Technology Development Agency (NSTDA), Tokyo Institute of Technology, Sirindhorn International Institute of Technology (SIIT), and Tham-masat University (TU) under the TAIST Tokyo Tech scholarship program.

Jakrin Butdee

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS/ABBREVIATIONS

Symbols	**Terms**
ϕ	Phase

Abbreviations	**Terms**
PD	Partial Discharge
PRPD	Phase Resolved Partial Discharge
CNN	Convolutional Neural Network
SVM	Support Vector Machine

CHAPTER 1
INTRODUCTION

Due to the importance of continuous and reliable operation of power generators in electric energy production, their overall conditions and the integrity of the insulation must be regularly monitored and maintained to avoid a decrease in power generation efficiency and the generators' total insulation breakdown. Instruments within the electrical supply network, particularly in high-voltage environments, are susceptible to extreme electrical, thermal, and mechanical stresses. These stresses lead to continuous degradation of the electrical apparatus, especially its insulation material Fothergill (2007). An event of insulation breakdown of vital equipment like a power generator or power transformer can cripple the local electricity supply network. Scheduled inspections are done periodically to determine the equipment's condition, although manual inspection is time-consuming as it requires full-system shutdown. Inspectors are assisted with partial discharge detection equipment that helps identify and locate the faults in the insulation, reducing manual inspection time Cavallini et al. (2006). Faults and contamination in dielectric insulation material of operating instruments emit partial discharge (PD), which is a discharge that partially bridges the conductors through an insulation layer and further deteriorates the weak point of the insulationLu et al. (2020). PD analysis is one of the most accepted and well-known processes to determine electrical equipment's condition, especially in high-voltage environments, to reduce the downtime to conduct the manual check-up Belkov et al. (2010) and Duan et al. (2019). However, the effective analysis of PD patterns such as phase-resolved partial discharge (PRPD) Ma et al. (2015) usually must rely on expert knowledge to classify closely related and mixed PD fault patterns Sukumar et al. (2021). Also, different measurement sources due to different types of sensors usually lead to complexity in analyzing the fault patterns A. R. Mor, Castro Heredia, et al. (2017) and A. R. Mor, Heredia, et al. (2017).

While Support Vector Machines (SVM) have been extensively utilized for classification tasks in various fields and regarded as the conventional method for PRPD pattern recognition Sharkawy et al. (2007), they may not be as effective as Convolutional Neural Networks (CNN) in handling dual cases of partial discharge and other pattern recognition tasks. However, CNNs have yet to be widely adopted in the electrical power industry for monitoring and diagnosing the condition of electrical equipment Lu et al. (2020). This research aims to contribute to the field by (1) proposing a systematic pre-processing approach

for PD data to obtain PRPD pattern images for analysis and (2) implementing a simple classification method based on CNN for pattern recognition with high accuracy, surpassing SVM. This approach can assist human experts in identifying PD faults in power generation applications.

CHAPTER 2
BACKGROUND KNOWLEDGE

2.1 Partial Discharge Phenomena

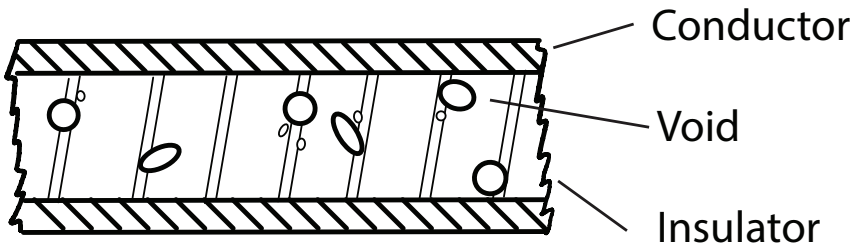
According to IEC 60034-27, PD is a localized electrical discharge partially bridging the insulation between conductors Lu et al. (2020). This discharge typically occurs when there are defects in the insulation material. These defects usually have existed since the manufacturing process as micro defects and will further develop due to aging from withstanding the combination of thermal, mechanical, and electrical stress. Fothergill (2007) PD will start to occur at the point of insulation defect that would act as the catalyst and accelerates with the rate at which the insulation degrades. As the defect grows, the voltage and frequency at which partial discharge occurs also increase. Eventually, electrical treeing will occur at the location, resulting in insulation breakdown and total equipment failure.

2.1.1 Types of Partial Discharge

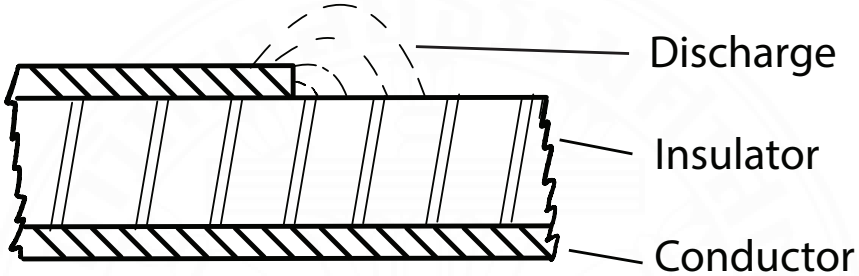
As PD can occur at multiple regions of the electrical equipment, the type of partial discharge that occurs will vary according to the defect region and the medium in which discharge originates, travels, and terminates. Examples of common partial discharges are internal void, surface, corona, floating particle, and floating electrode. However, this paper will focus on only three types of discharge: internal void, surface, and corona discharge, as they are the most common types that occur on the stator and rotor of power generators. Each type of PD occur from different conditions and region of the equipment; an illustration of different PD activity region is shown in Fig. 2.1.

2.1.1.1 Internal Void Discharge

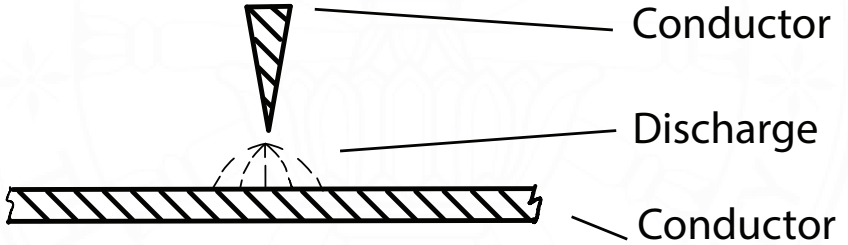
Internal void discharge, also known as internal discharge or void discharge, originates from the unwanted voids in the insulation materials, as shown in Fig. 2.1(a). These voids may exist since the insulation manufacturing process, as the manufacturing environments and methods cannot perfectly prevent microvoids from forming in the materials. These micro-voids from the manufacturing process pose little to no threat to the condition of the insulation. However, after some operation periods under thermal, electrical, and mechanical stress, these voids may enlarge and start to emit PD. PD activities in the voids can lead to accelerated insulation aging, further enlarging the void and escalating the intensity of



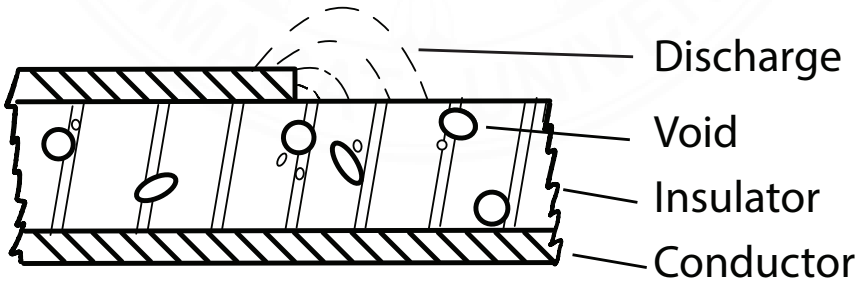
(a) Internal void PD



(b) Surface PD



(c) Corona PD



(d) Dual case of Internal and Surface PD

Figure 2.1 Diagram for each type of PD.

the discharge emitted. After an extended time without servicing can lead to electrical treeing in the insulation and eventually complete failure of the insulation.

2.1.1.2 Surface Discharge

Surface discharge is when the discharge travels along the surface of the insulator, as shown in Fig. 2.1(b). Surface discharge may occur from other means, but typically it originates from moisture intrusion of the insulator, which is caused by high environmental humidity that may be combined with poor maintenance. Overtime humidity or other environmental pollutants may form a layer on the surface of the insulator. If the electrical potential of the electrodes separated by the insulator is large enough, an electrical discharge could occur at the polluted layer. These discharges on the insulator surface generate extreme heat that will carbonize that insulator surface region and leave a permanent carbon track at the discharge location, further reducing the insulator's localized integrity. These carbon tracks could lead to more severe surface discharge and lead to complete failure of the insulation.

2.1.1.3 Corona Discharge

This type of discharge can be visibly observed in the form of purple-arcing light and is sometimes audible by our ears. Corona discharge commonly occurs at the sharp conductor discharging the surrounding gas due to a high voltage gradient, as shown in Fig. 2.1(c). A voltage gradient is the measure of electrical potential difference over the distance; at the extremity point of a sharp conductor, the measure of voltage gradient would be critically high incepting corona discharge. Corona discharge generally does not degrade the integrity of the insulator as the electrical discharge is typically released into surrounding gas. Hence, corona discharge can be left longer than the surface and internal discharge without causing damage to the surrounding dielectric materials. However, over an extended period, the corona effect may occasionally discharge an arc onto the surrounding insulation and continuously reduce the insulation integrity, which could also lead to insulation breakdown.

Note that each type of PD could occur independently if the conditions for each case are met. For example, the dual case of Internal and Surface PD can simultaneously occur as illustrated in Fig. 2.1(d).

2.1.2 Phase-Resolved Partial Discharge (PRPD) Pattern

Phase-Resolved Partial Discharge Pattern or PRPD pattern is widely accepted as an effective method to classify partial discharge sources. PRPD pattern visualizes PD activity relative to 360° of AC cycle with a 50- or 60 Hz system frequency. PRPD pattern can be

plotted using phase information and measured partial discharge event amplitude. Visualization of different types of PD using the PRPD pattern is shown in Fig. 2.2. Each dot on the PRPD pattern represents one detected PD event, using the PD inception phase as the x-axis value and PD peak amplitude as the y-axis value. Scattering multiple PD events create a recognizable pattern for each type of partial discharge. In practice, the PRPD patterns are interpreted by a partial discharge expert to determine which type of PD is occurring on the equipment under consideration.

2.2 Classification Algorithm

2.2.1 Support Vector Machine

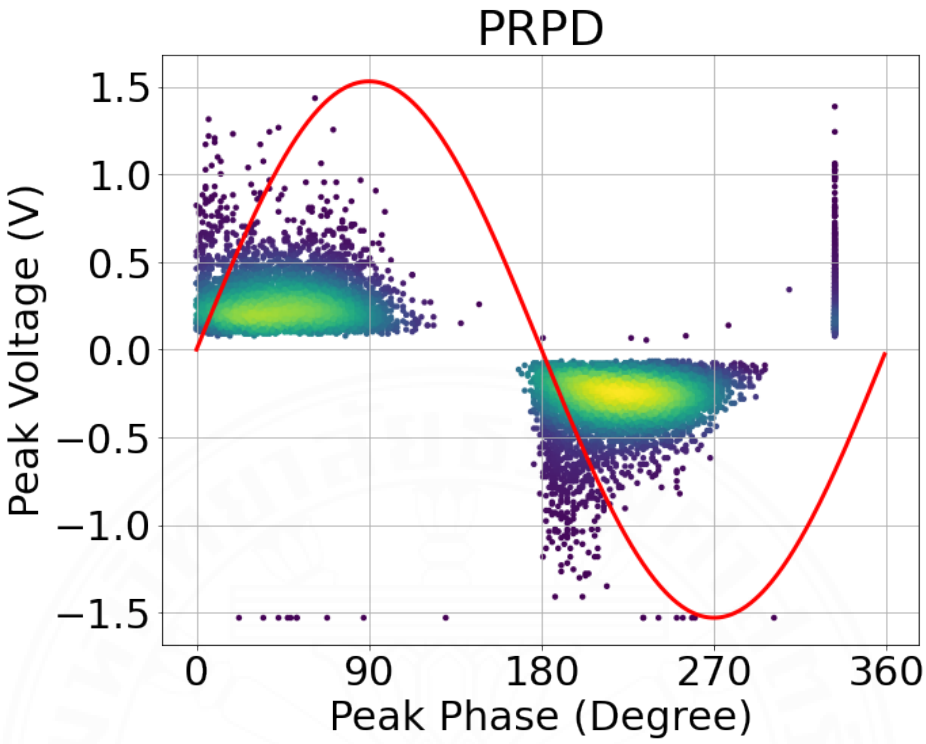
Support Vector Machine (SVM) is generally recognized as one of the better-classifying algorithms among various families of classifiers due to its reliable accuracy and empirical performance Soms (2015). SVM is a deterministic model that can operate both linear and non-linear classifications by mapping the input vector onto a large hyperplane separated by other hyperplanes to create sub-regions. These regions belong to each class of data, and each input vector is classified according to the sub-region it is mapped on. The hyperplane that separates each region is called the margin, and the larger the margin is assumed to make the model have a lower generalization error. This study uses SVM as a conventional baseline method for multiple PRPD pattern classification tasks, including written and string text recognition, image classification, and object recognition.

2.2.2 Convolutional Neural Network

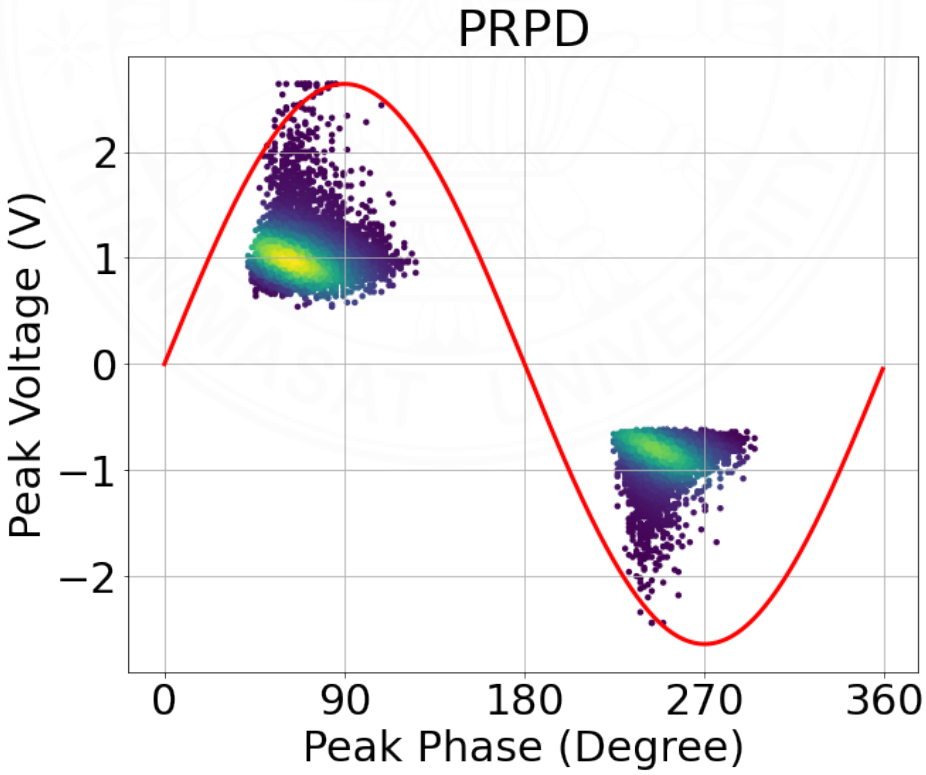
Studies in recent years have shown several empirical performance of Convolutional Neural Networks (CNNs) Ajit et al. (2020) with much higher level of accuracy in comparison with other standard classifiers. Although CNN requires more computing resources, it still falls within an acceptable margin. CNN operation imitates our visual cortex, which utilizes layers of connected neurons to recognize a learned object or particular pattern. The most basic structure of CNN contains three layers: a convolutional layer, a pooling layer, and a fully connected layer.

2.2.2.1 Convolutional Layer

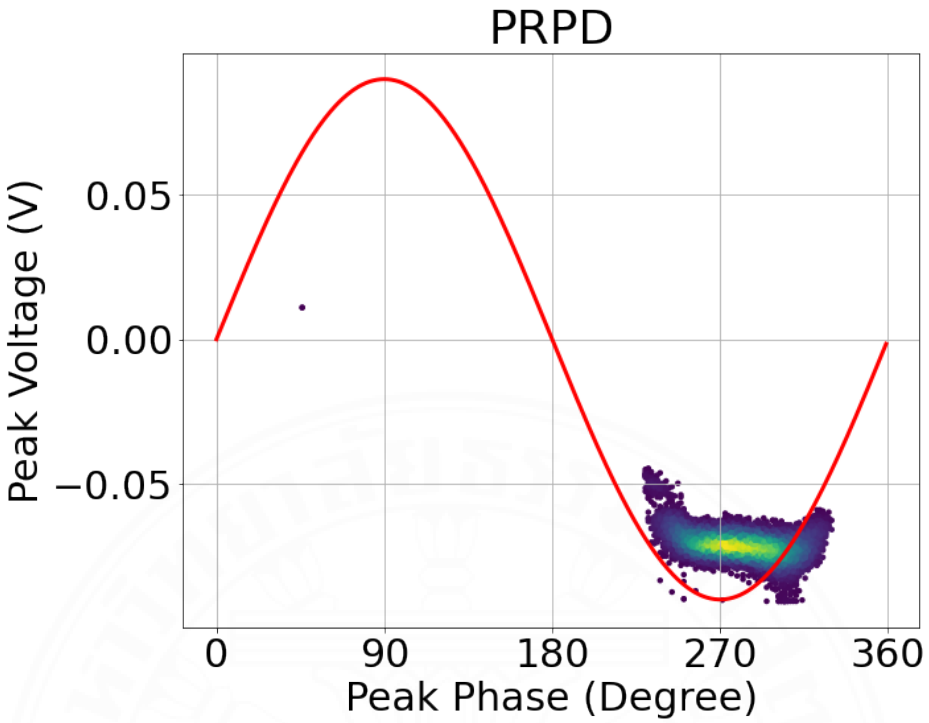
The convolutional layer multiplies or convolutes the input image with a feature detector filter to create an activation map. Each activation map contains features detected by the convoluted feature map, which also decreases the data dimension from the input image.



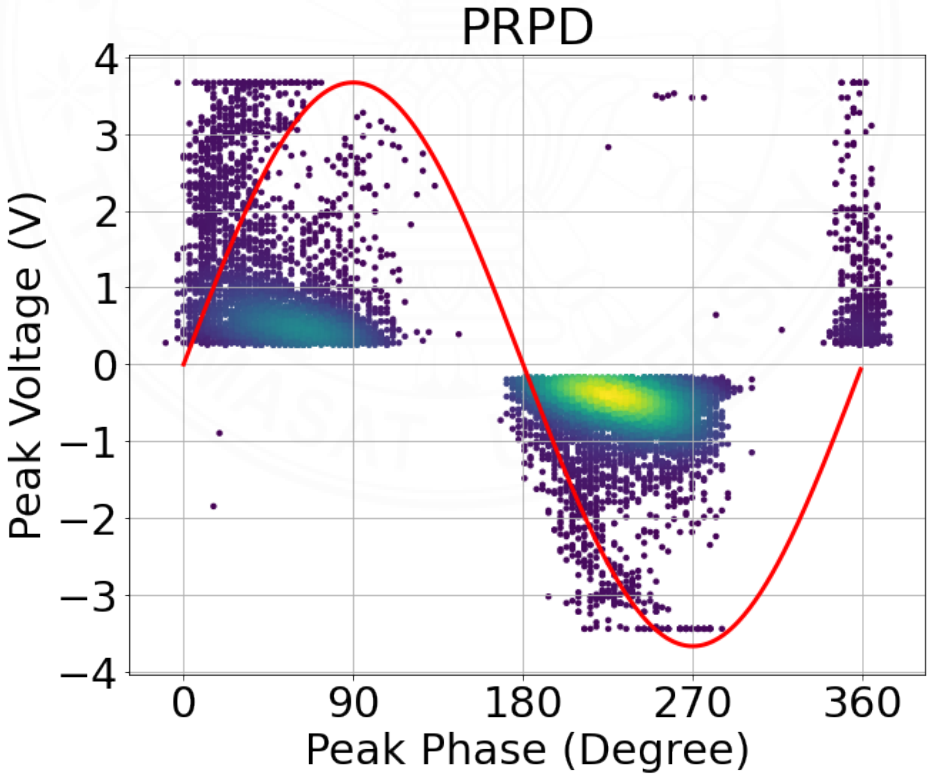
(a) Internal void PD



(b) Surface PD



(c) Corona PD



(d) Internal and Surface PD

Figure 2.2 PRPD patterns for considered PD types.

A single input image is convoluted with multiple feature detection filters to create a collection of activation maps. The stacked collection of activation maps is called the convolutional layer.

2.2.2.2 Pooling Layer

Each activation map may contain multiple detected features, so the pooling layer further reduces the activation map dimensions while preserving important features. There are multiple types of pooling layers: max pooling, average pooling, and stochastic pooling; each typing of the pooling layer signifies the algorithm used to select important features from the activation map. This filtering process helps to avoid over-fitting and improves generalization of the model while also further reduces the data dimension.

2.2.2.3 Fully Connected Layer

the output from pooling layer is first flattened from matrix form into vectors, each vector then fed into each input node of the fully connected layer. The fully connected layer is a feed-forward neural network, which multiplies the received input the node's weight and the output sent forward to the input. The process is repeated until the final output layer, where each nodes value is used in the activation function to calculate the probability that the input belongs to each class.

CHAPTER 3
METHODOLOGY

The details of the proposed PD analysis based on Machine Learning are presented, starting from the data pre-processing, the dataset generation of PRPD pattern images, the SVM Classification as a conventional method using extracted features and the proposed method of using CNN classification method for comparison. The flowchart of the overall process is shown in Fig. 3.1.

3.1 Data Pre-Processing

The PD data used in this study are provided publicly by the Electrical Sustainable Power Laboratory of The Technical University of Delft, Netherlands Heredia (2020). These original datasets are generated using an artificial PD source in a laboratory environment to create each specific PD phenomenon and recorded using sensors connected to an oscilloscope. The full description and configuration of the testing platform are thoroughly discussed in A. Mor et al. (2018). Each file contains different accumulated PD events separated in two types of waveforms in two separate channels:

3.1.1 PD Waveforms

The first channel contains the detected partial discharge waveforms, these waveforms contain irregular peaks in the observed voltage. An example of a PD signal waveform is shown in Fig. 3.2(a). The frame of significant peak in the waveform is then identified as the PD inception frame. The amplitude at PD inception frame is V_{PD} which is used as the y-coordinate of a data point on the PRPD pattern. The inception frame number is also used to determine the phase value V_{ϕ} from the corresponding sawtooth waveform.

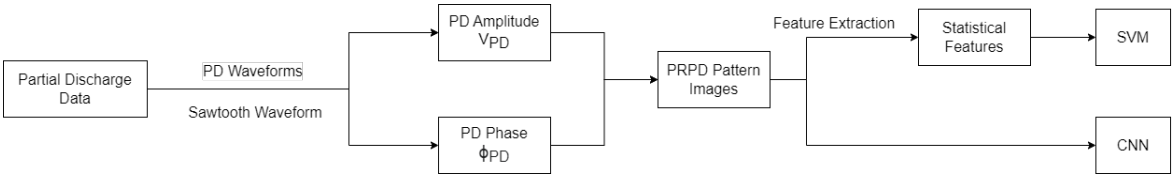
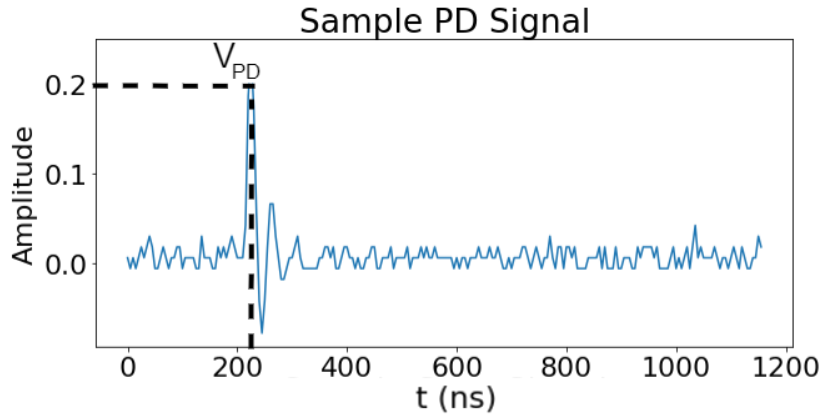
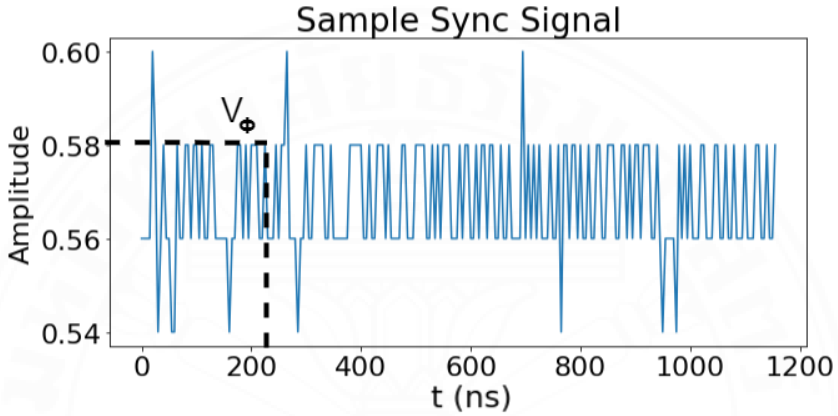


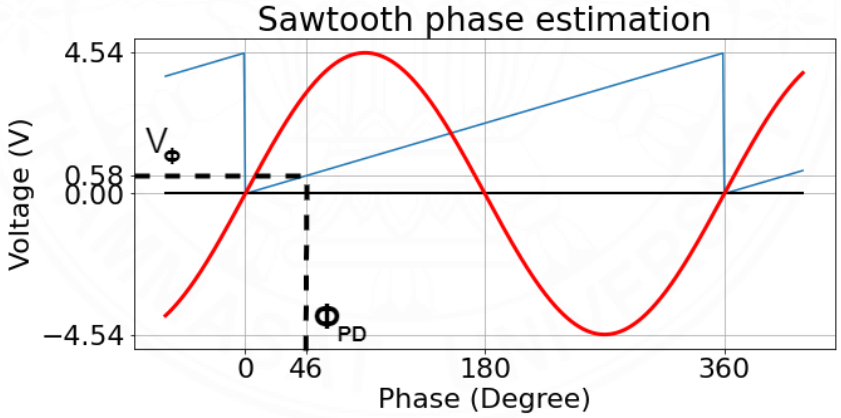
Figure 3.1 Methodology flowchart.



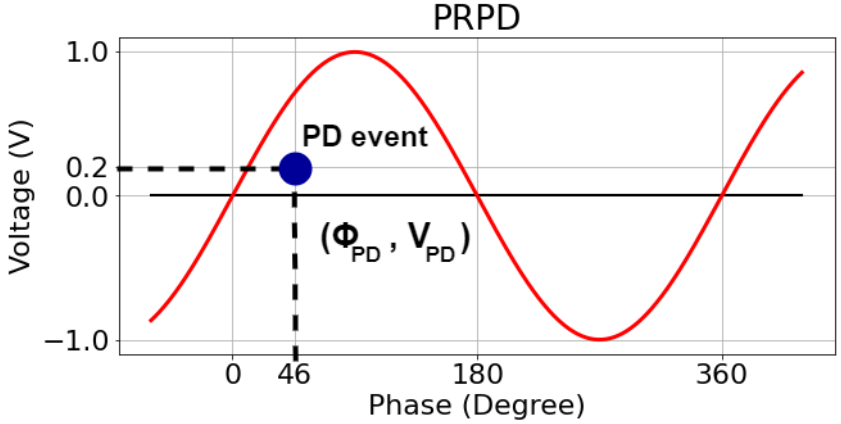
(a) PD inception frame at significant peak in PD waveform.



(b) Sawtooth voltage as V_{ϕ} at PD inception frame.



(c) Phase estimation using (3.1).



(d) Plotting PD event on PRPD pattern.

Figure 3.2 PRPD construction from two channels.

Table 3.1 PRPD dataset with one severity

Corona	Internal	InternalSurface	Surface
100	600	349	149

Table 3.2 PRPD dataset with three levels of severity per one type of PD

	Corona	InternalSurface	Internal	Surface
mild	[488]	[1712]	[2940]	[721]
moderate	[95]	[332]	[574]	[143]
severe	[19]	[63]	[114]	[28]

3.1.2 Sawtooth Waveforms

This channel contains the voltage waveform of a sawtooth signal acting as synchronization signal. It operates in phase with the applied voltage to generate partial discharge. An example of sawtooth signal waveform is shown in Fig. 3.2(b). The calculation of the instantaneous phase information from its voltage magnitude is given by

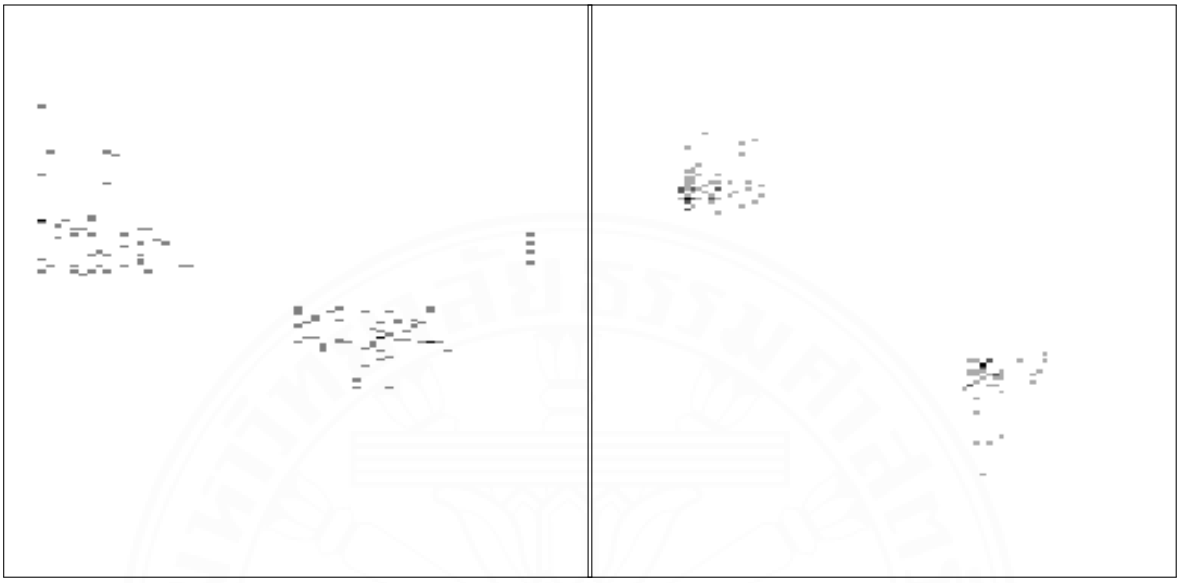
$$\phi_{\mathrm{PD}} = \frac{360^{\circ}}{V_{\mathrm{saw}}} V_{\phi} \tag{3.1}$$

where ϕ_{PD} is the electrical phase when the PD event takes place, V_{ϕ} is the voltage of the reference sawtooth waveform at PD inception frame, and V_{saw} is the maximum value of the reference sawtooth waveform.

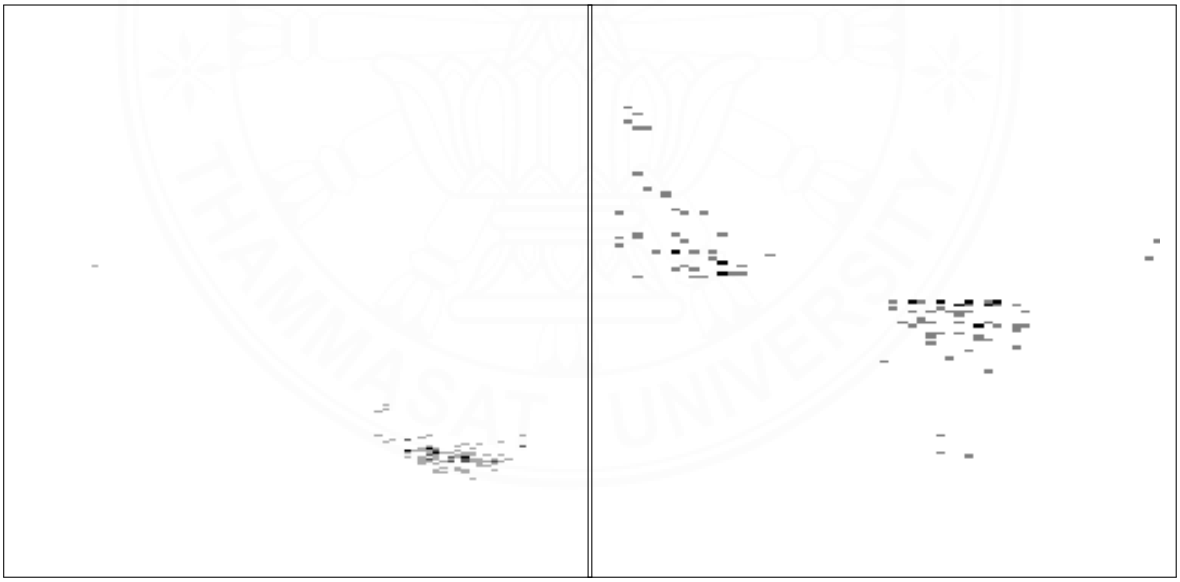
The sawtooth amplitude at the inception frame will be used as V_{ϕ} in (3.1) to calculate the electrical phase value ϕ_{PD} which is used as the x-coordinate of a data point of the PRPD pattern in accordance with V_{PD} obtained from PD waveform as the y-coordinate. Illustrations of this process are shown in Figs. 3.2(c)-(d) with $V_{\mathrm{saw}} = 4.54\ V$ and other values are shown on the graphs.

3.2 Generation of Image Datasets

As each recorded partial discharge file is the accumulation of multiple PD events, the images in the dataset are generated by sampling a small number of PD events to create PRPD pattern images for a trainable dataset. These images will be used to train a pattern recognition model that can be further classified for each type of PRPD pattern.



(a) Internal void PD (b) Surface PD



(c) Corona PD (d) Internal and Surface PD

Figure 3.3 Generated PRPD pattern images of different PD types for Machine Learning.

3.2.1 Image Generation Algorithm

At first, the author implemented the PRPD image generation code using MATLAB platform as MATLAB has provided visualization tools allowing a much easier understanding of the dataset structure. As the steps of processing PD voltage waveforms into PRPD image are not documented, through an extensive literature review, the following equations presented in the study by Cavallini et al. (2003) helped clarify the process. Although equations (3.2, 3.3,3.4, 3.5) are used for feature extraction of PD waveform to be plotted on to Time-Frequency plane and separate multiple sources of PD and noise from each other. Understanding these equations allowed the author to devise the steps to convert multiple PD waveforms into PRPD patterns discussed in 3.1. Although MATLAB has assisted in understanding the PD waveform data structure, generating PRPD images from a large collection of data requires a long time. MATLAB requires almost two hours to generate about 1,200 images from the dataset of 70,000 pairs of PD waveform. Implementing a similar algorithm using Python improves the performance substantially, completing identical tasks in less than 10 minutes.

$$1/\|s\| = 1/\sqrt{\int_0^T s(t)^2 dt} \tag{3.2}$$

$$\tilde{s}(t) = s(t)/\|s\| \tag{3.3}$$

$$\sigma_T = \sqrt{\int_0^T (t-t_0)^2 \tilde{s}(t)^2 dt} \tag{3.4}$$

$$\sigma_F = \sqrt{\int_0^\infty f^2 |\tilde{S}(f)|^2 df} \tag{3.5}$$

Two sets of data are generated; the first set contains four classes representing each type of PD: internal, surface, corona, and the internal surface dual case. The second set of data contains 12 classes of data, with the same four types of PD, each consisting of three different levels of severity: mild, moderate, and severe. An example of generated images for

each PD type is shown in Fig. 3.3. Table. 3.1 outlines the first set of data, which contains four classes of PD and is used as a proof of method to ensure that both SVM and CNN can correctly classify the regular cases of PD, and the second set of data will be used to evaluate both methods in the more complex task of further classification of each PD severity outlined by Table. 3.2.

3.2.2 PD Severity

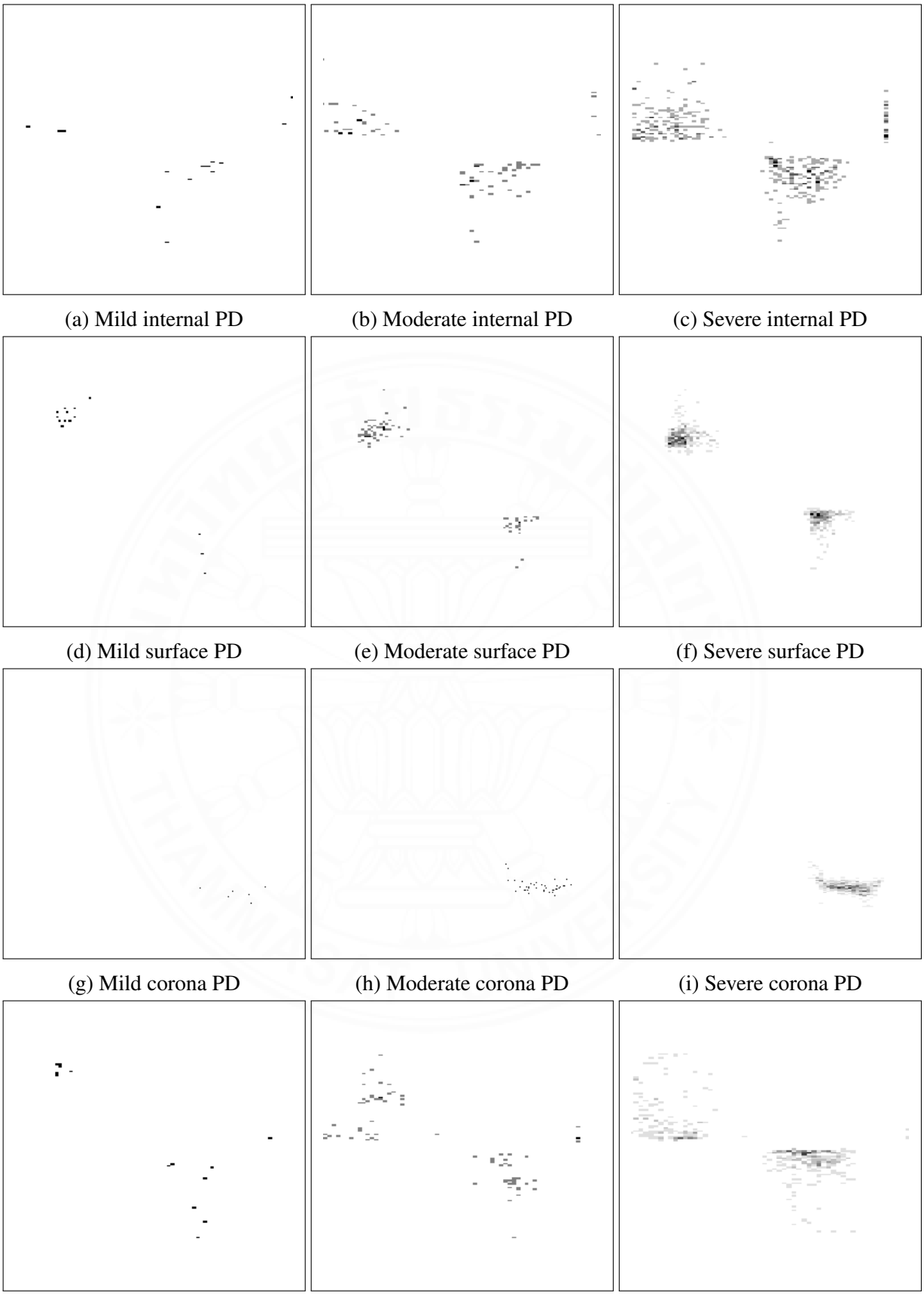
PD experts typically assess the severity of PD based on various factors, although the most straightforward approach involves evaluating the density of PD events in the PRPD pattern. Each severity level corresponds to a different quantity of PD events in each PRPD image. However, to prevent the CNN model from solely relying on the number of pixels in each PRPD image to determine severity, rather than considering the overall pattern and cluster density, a baseline quantity is established for each severity level. Additionally, to introduce randomness in each PRPD image, the baseline quantity is shifted by a random percentage of delta (e.g., 10%) for each image. The introduced randomness of PD event amount ensures that the CNN model learns to identify severity levels based on the overall pattern characteristics rather than relying solely on the number of PD events in each image. The example of the generated PRPD image with varying severity is shown in Fig. 3.4.

3.3 SVM Classification

The first method considered for PD classification is using feature-extracted values to train a classical classifier SVM. After transforming multiple PD events into a PRPD image, the voltage and waveform information of each PD event is not accessible. So the features used to train a recognition model has to be obtained solely from each PRPD image. Effective statistical features from PRPD images that SVM can train on and identify each PRPD pattern are image skewness, kurtosis, and variance. Kartojo et al. (2019), which are calculated as follows:

$$\sigma^2 = \frac{\sum_{i=1}^{N}(x_i - \bar{x})^2}{N} \tag{3.6}$$

$$\tilde{\mu}_3 = \frac{\sum_{i=1}^{N}(x_i - \bar{x})^3}{(N-1)\sigma^3} \tag{3.7}$$



(a) Mild internal PD (b) Moderate internal PD (c) Severe internal PD

(d) Mild surface PD (e) Moderate surface PD (f) Severe surface PD

(g) Mild corona PD (h) Moderate corona PD (i) Severe corona PD

(j) Mild internal & surface PD (k) Moderate internal & surface PD (l) Severe internal & surface PD

Figure 3.4 Sample of each type of PD with different severity.

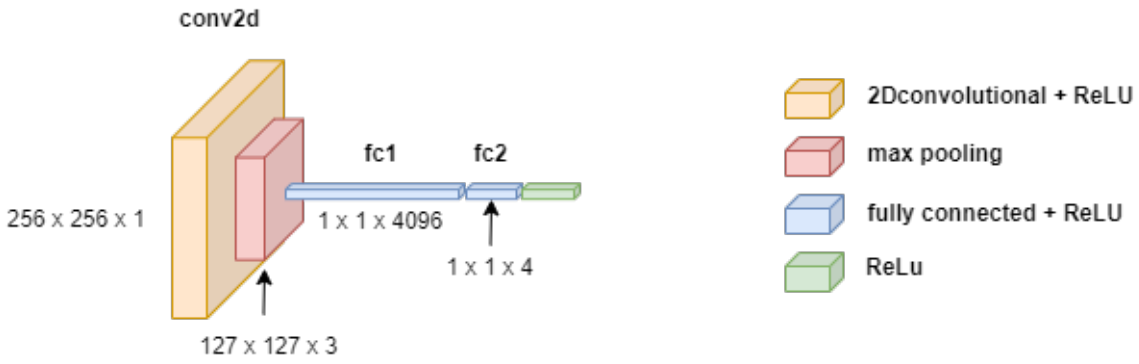


Figure 3.5 Convolutional neural network model layers.

$$\text{Kurt} = \frac{\sum_{i=1}^{N}(x_i - \bar{x})^4}{(N-1)\sigma^4} \tag{3.8}$$

σ^2 is variance of pixel intensity in the observed region, x_i is the pixel i intensity value, $\bar{x}$ is the whole image average pixel intensity, and N is the total pixel count in each image. σ is the standard deviation value of the whole image and is used to calculate $\tilde{\mu}_3$ (skewness) and Kurt (kurtosis) value of the observed region. The locale of the PD cluster on the PRPD pattern is important, whether it lies on the positive phase (left half of PRPD image) or negative phase (right half of PRPD image). Each PRPD image will be transformed into a list of values containing six elements, that is, the σ^2, $\tilde{\mu}_3$, and Kurt of both positive phase and negative phase of the image.

3.4 CNN Classification

Unlike SVM, which requires prior knowledge to identify the important features that can achieve sufficient accuracy classification model, CNN can achieve a better result without going through feature identification and extraction manually. The CNN layers used in this study are configured to minimize the complexity of the model. As shown in Fig. 3.5, the CNN model structure contains only one convolutional layer, one pooling, and two dense layers to minimize the training and inferencing resource.

CHAPTER 4
RESULT AND DISCUSSION

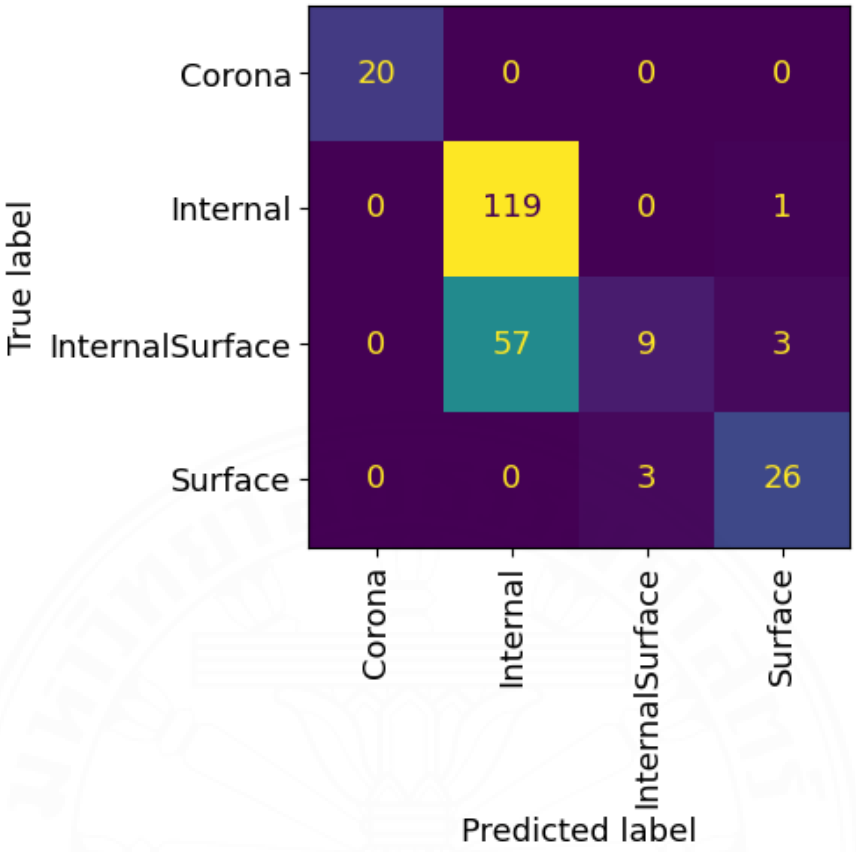
The classification results obtained from SVM and CNN on the first dataset consisting of four classes of PD are presented in Figs. 4.1(a) and (b). The limitations of SVM, which relies on statistical features derived from PRPD images, are evident from the classification results. While SVM performs well in differentiating between the three simple types of PD (internal, surface, and corona) due to their distinct PRPD cluster shapes, it struggles to classify the dual case of internal and surface PD reliably. The statistical features may not provide enough information for SVM to discriminate in such cases. In contrast, even a simple CNN model with minimal layers demonstrates a better capability to classify all unseen test images correctly. CNN proves to be more reliable in classifying all four classes of PRPD patterns.

The performance of both methods is evaluated using the weighted f-1 score, which combines recall and precision to account for the class imbalance in the test dataset. SVM achieves 171 correct inferences out of the total 238 test images, resulting in an accuracy of 73% and a weighted f-1 score of only 66%. Notably, the SVM model struggles the most with the dual-case PD classification, resulting in the class's f-1 score of 22%. On the other hand, CNN accurately identifies all unseen test images. The challenging differentiation between internal, surface, and dual cases of PD indicates that the statistical features derived from the PRPD pattern may not be sufficient for SVM to achieve accurate classification. In contrast, CNN excels at automatically extracting features from the PRPD patterns and effectively identifies and extracts features crucial for characterizing each type of PD fault.

The proposed CNN-based method effectively classifies all four PD types, including the visually challenging dual case of internal and surface PD. To ensure the model's generalizability, further experiments were conducted to address the possibility of overfitting. Since the CNN model achieves error-free classification among the four classes of PD, the investigation will be conducted by increasing the complexity of the classification task. The second dataset was expanded into 12 classes, containing three severity levels (mild, moderate, and severe) for each type of PD.

The increased number of classes poses additional challenges, particularly for PRPD images with low severity that exhibit minimal PD events with vague clusters, as seen in Fig. 3.4. However, the classification performance of the CNN model remains reliable in this scenario. As shown in Fig. 4.2a, with the introduction of severity level, SVM struggles

(a) SVM on dual-case test set.



(b) CNN on dual-case test set.

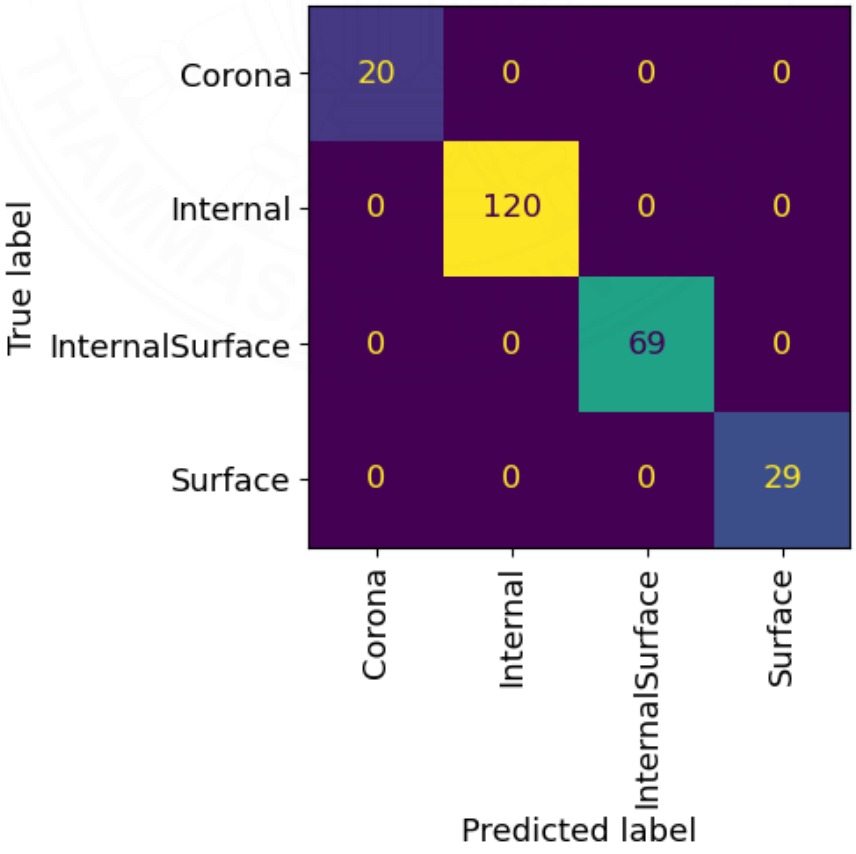
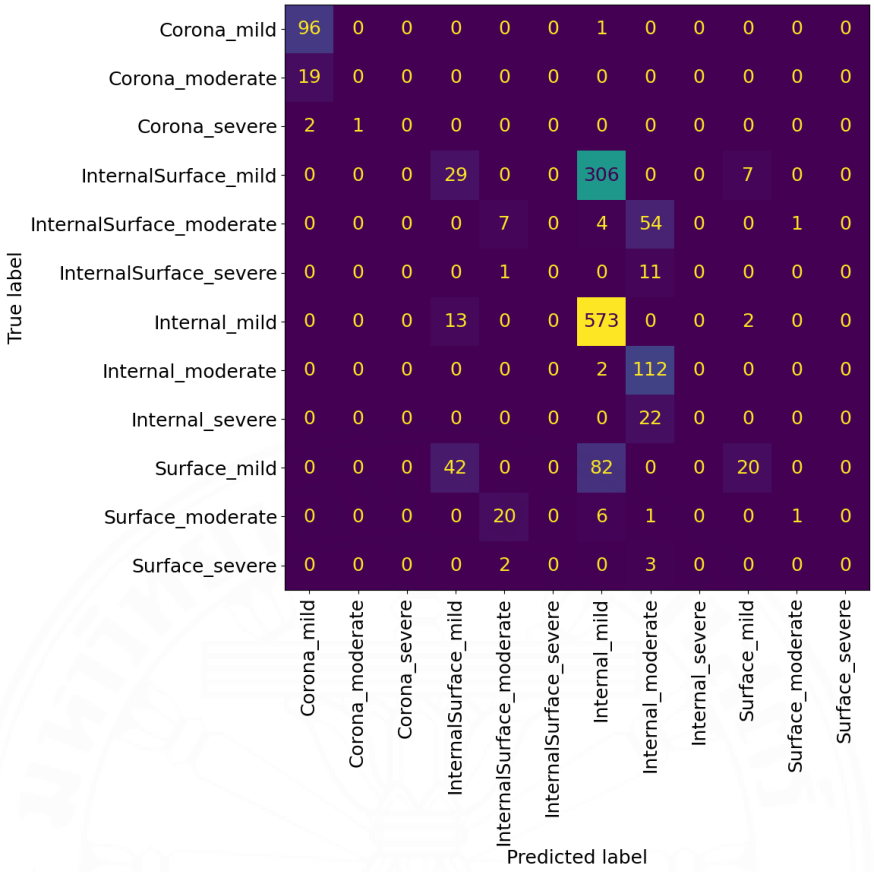


Figure 4.1 CNN confusion matrix.

(a) SVM confusion matrix on severity dataset.



(b) CNN on unseen test set.

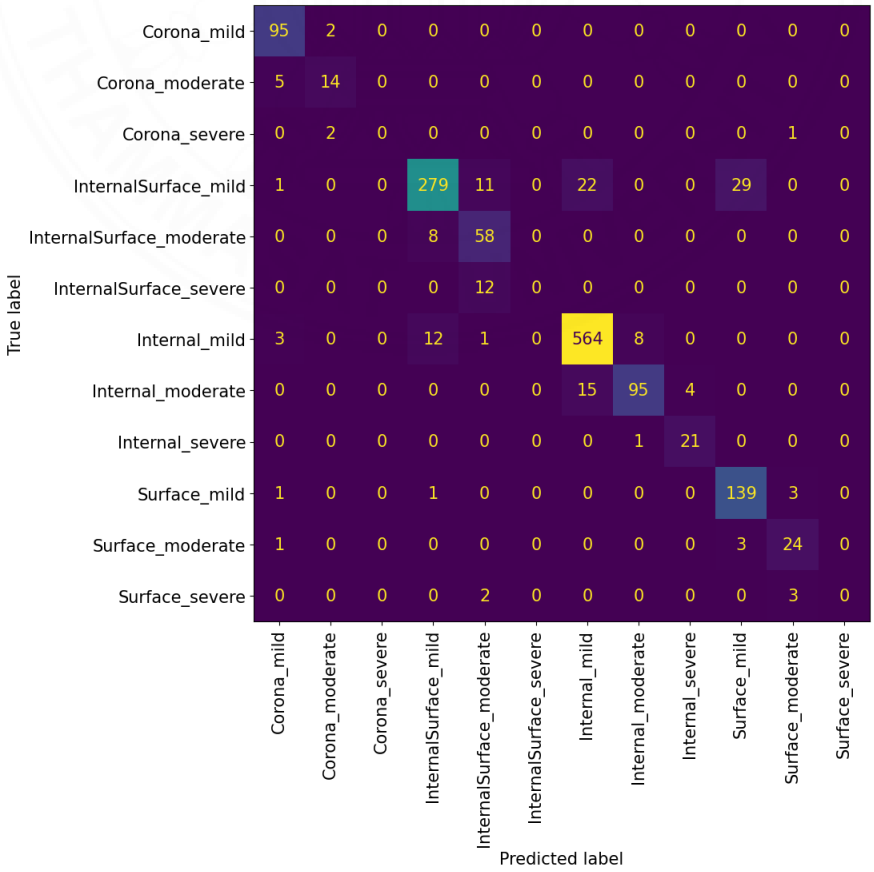


Figure 4.2 CNN confusion matrix on severity dataset.

	precision	recall	f1-score	support
Corona_mild	0.9	0.98	0.94	97
Corona_moderate	0.78	0.74	0.76	19
Corona_severe	0.0	0.0	0.0	3
InternalSurface_mild	0.93	0.82	0.87	342
InternalSurface_moderate	0.69	0.88	0.77	66
InternalSurface_severe	0.0	0.0	0.0	12
Internal_mild	0.94	0.96	0.95	588
Internal_moderate	0.91	0.83	0.87	114
Internal_severe	0.84	0.95	0.89	22
Surface_mild	0.81	0.97	0.88	144
Surface_moderate	0.77	0.86	0.81	28
Surface_severe	0.0	0.0	0.0	5
micro avg	0.9	0.9	0.9	1440
macro avg	0.63	0.67	0.65	1440
weighted avg	0.89	0.9	0.89	1440
samples avg	0.9	0.9	0.9	1440

Figure 4.3 Detailed performance matrix of CNN on severity dataset.

to classify most of the dual cases test image, and its accuracy drops even lower, with the weighted f-1 score decreasing from 66% to 48%. Notably, SVM misclassifies not only the severity level but also the inferred PD type.

The detailed performance of CNN is shown in Fig. 4.3, with an overall weighted f-1 score remaining high at 89%. Although CNN can correctly classify most classes, it faces challenges in some classes, including severe corona, internal, surface PD, where it achieves 0% correct inference. This limitation stems from the fact that these three cases have a very low number of test images, comprising only 12 images out of the set of 1440 test images. Despite this, CNN can still correctly classify the type of the corresponding PD. Observing the confusion matrix results shown in Fig. 4.2, it can be seen that most of the misclassifications by CNN occur in the severity level.

In conclusion, the results demonstrate the superiority of the proposed CNN-based method in accurately classifying PD types; not only is the challenging dual case of internal and surface PD, but it can also perform great inference on the severity level of each PRPD image. The limitations of SVM, relying on statistical features, become apparent when discriminating complex PD cases. In contrast, CNN excels in feature extraction from PRPD patterns, enabling improved classification performance. These findings advocate for the adoption of CNN over SVM for PD analysis. While CNN may require slightly more computational resources and time, the additional time for image classification is negligible in partial discharge analysis, as real-time inference is not a strict requirement. Furthermore, the computational resource difference between SVM and the CNN method in this study is minimal, considering using a simple CNN model in this study.

REFERENCES

Ajit, A., Acharya, K., & Samanta, A. (2020). A review of convolutional neural networks. *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*, 1–5. https://doi.org/10.1109/ic-ETITE47903.2020.049

Belkov, A., Koltunowicz, W., Obralic, A., & Plath, R. (2010). Advanced approach for automatic prpd pattern recognition in monitoring of hv assets. *2010 IEEE International Symposium on Electrical Insulation*, 1–5. https://doi.org/10.1109/ELINSL.2010.5549804

Cavallini, A., Contin, A., Montanari, G., & Puletti, F. (2003). Advanced pd inference in on-field measurements. i. noise rejection. *IEEE Transactions on Dielectrics and Electrical Insulation*, *10*(2), 216–224. https://doi.org/10.1109/TDEI.2003.1194102

Cavallini, A., Montanari, G., & Puletti, F. (2006). Partial discharge analysis and asset management: Experiences on monitoring of power apparatus. *2006 IEEE/PES Transmission & Distribution Conference and Exposition: Latin America*, 1–6. https://doi.org/10.1109/TDCLA.2006.311544

Duan, L., Hu, J., Zhao, G., Chen, K., He, J., & Wang, S. X. (2019). Identification of partial discharge defects based on deep learning method. *IEEE Transactions on Power Delivery*, *34*(4), 1557–1568. https://doi.org/10.1109/TPWRD.2019.2910583

Fothergill, J. C. (2007). Ageing, space charge and nanodielectrics: Ten things we don't know about dielectrics. *2007 IEEE International Conference on Solid Dielectrics*, 1–10. https://doi.org/10.1109/ICSD.2007.4290739

Heredia, C. (2020). Partial discharges software - pdflex: Pd parameters and clustering. http://pdflex.ewi.tudelft.nl/

Kartojo, I. H., Wang, Y.-B., Zhang, G.-J., & Suwarno. (2019). Partial discharge defect recognition in power transformer using random forest. *2019 IEEE 20th International Conference on Dielectric Liquids (ICDL)*, 1–4. https://doi.org/10.1109/ICDL.2019.8796809

Lu, S., Chai, H., Sahoo, A., & Phung, B. T. (2020). Condition monitoring based on partial discharge diagnostics using machine learning methods: A comprehensive state-of-the-art review. *IEEE Transactions on Dielectrics and Electrical Insulation*, *27*(6), 1861–1888. https://doi.org/10.1109/TDEI.2020.009070

Ma, H., Saha, T., Ekanayake, C., & Martin, D. (2015). Smart transformer for smart grid—intelligent framework and techniques for power transformer asset management. *IEEE Transactions on Smart Grid*, *6*. https://doi.org/10.1109/TSG.2014.2384501

Mor, A., Heredia, L., Harmsen, D., & Muñoz Muñoz, F. A. (2018). A new design of a test platform for testing multiple partial discharge sources. *International Journal of Electrical Power & Energy Systems*, *94*, 374–384. https://doi.org/10.1016/j.ijepes.2017.07.013

Mor, A. R., Castro Heredia, L. C., & Muñoz, F. A. (2017). New clustering techniques based on current peak value, charge and energy calculations for separation of partial discharge sources. *IEEE Transactions on Dielectrics and Electrical Insulation*, *24*(1), 340–348. https://doi.org/10.1109/TDEI.2016.006352

Mor, A. R., Heredia, L. C. C., & Muñoz, F. A. (2017). Estimation of charge, energy and polarity of noisy partial discharge pulses. *IEEE Transactions on Dielectrics and Electrical Insulation*, *24*(4), 2511–2521. https://doi.org/10.1109/TDEI.2017.006381

Sharkawy, R. M., Mangoubi, R. S., Abdel-Galil, T., Salama, M. M., & Bartnikas, R. (2007). Svm classification of contaminating particles in liquid dielectrics using higher order statistics of electrical and acoustic pd measurements. *IEEE Transactions on Dielectrics and Electrical Insulation*, *14*(3), 669–678. https://doi.org/10.1109/TDEI.2007.369530

Soms, N. (2015). A review on support vector machine: A classification method.

Sukumar, T., Balaji, G., Vigneshwaran, B., Prince, A., & Kumar, S. (2021). Recognition of single and multiple partial discharge patterns using deep learning algorithm. *2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS)*, 184–189. https://doi.org/10.1109/ICAIS50930.2021.9395881

APPENDIX

APPENDIX A

PYTHON CODES

```
%matplotlib inline
from wfmread import wfmread
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy.stats import gaussian_kde
from IPython.display import clear_output
from scipy import signal
import matplotlib.pyplot as plot
import numpy as np
# import FlowCal
import cv2
COLOR = 'black'
plt.rcParams['text.color'] = COLOR
plt.rcParams['axes.labelcolor'] = COLOR
plt.rcParams['xtick.color'] = COLOR
plt.rcParams['ytick.color'] = COLOR
plt.rcParams["figure.figsize"] = (6.4,4.8)
plt.rcParams.update({'font.size': 22})
# plt.rcParams['axes.facecolor'] = 'White'

Nframes = 1000
phase_lst = []
peak_lst = []
fig = plt.figure()
fig.set_figheight(8)
fig.set_figwidth(10)
ax = fig.subplots()
# scatter_with_gaussian_kde(ax, phase_lst, peak_lst)

yrange = 1
x = np.arange(-60,420)
```

```
x2 = np.arange(0,360)
y = yrange*np.sin(2*np.pi*1/360*x)
y2 = yrange/2*signal.sawtooth(2*np.pi*1/360*x)+(yrange/2)



t = np.linspace(0, 1, 1000, endpoint=True)


def get_sub(x):
    normal = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789
        +-=()"

    res = x.maketrans(''.join(normal), ''.join(sub_s))
    return x.translate(res)

# txt = 'V{}'.format(get_sub("i"))

# Plot the sawtooth wave
# plt.plot(x, y2)
# plt.text(50, -1, '\u03C6', fontsize=40)
# plt.text(-40, 0.3, txt, fontsize=40)
# plt.text(40, 0.35, 'PD Event', fontsize=25)
yhori = list(range(-40,46))
# plt.axvline(x = 46, ymin = 0, ymax = 0.57, color = 'k', linestyle='dashed
    ',linewidth=3)
y3 = [0]*x
# plt.plot(yhori,[0.2]*len(yhori), color = 'k', linestyle='dashed',linewidth
    =3)
plt.plot(x,y3,'k', linewidth=2)
plt.plot(x,y,'r', linewidth=3)
plt.yticks([-1,0,0.2,1])
# plt.yticks([-4.54,0,0.58,4.54])
plt.xticks([0,46,180,360])
plt.grid()
plt.title('PRPD')
# plt.title('Sawtooth phase estimation')
```

```
plt.ylabel('Voltage (V)')
plt.xlabel('Phase (Degree)')
ratio = .442
x_left, x_right = ax.get_xlim()
y_low, y_high = ax.get_ylim()
ax.set_aspect(abs((x_right-x_left)/(y_low-y_high))*ratio)
plt.show()
# fig.savefig(r'C:\Users\Fusible\Desktop\Thesis\Graphics\Img\sawtooth.png', format='png', dpi=1000)
# fig.savefig(r'C:\Users\Fusible\Desktop\Thesis\Graphics\Img\prpddia.png', format='png', dpi=100)

#Sync Ch1, Sig_Ch2
ch1_dict = {
    '1':'wfm/InternalNew3_N5_300mV_200M_1us_200_Ch1.wfm',
    '2':'wfm/Surface_N5_500mV_200M_1us_200_Ch1.wfm',
    '3':'wfm/Positive2_N5_10mV_5G_2us_10000sa_Ch1.wfm',
    '4':'wfm/FreeMoving1_N5_50mV_200M_1us_200_Ch1.wfm',
    '5':'wfm/FloatingParticle_N5_2V_1G_1us_1000_Ch1.wfm',
    '6':'wfm/1-Internal_45mm33_Ch1.wfm',
    '7':'wfm/6kV_internal_oil_89pF_surface_69pF_N6_L43_HV_plates_Ch1.wfm'
}
ch2_dict = {
    '1':'wfm/InternalNew3_N5_300mV_200M_1us_200_Ch2.wfm',
    '2':'wfm/Surface_N5_500mV_200M_1us_200_Ch2.wfm',
    '3':'wfm/Positive2_N5_10mV_5G_2us_10000sa_Ch2.wfm',
    '4':'wfm/FreeMoving1_N5_50mV_200M_1us_200_Ch2.wfm',
    '5':'wfm/FloatingParticle_N5_2V_1G_1us_1000_Ch2.wfm',
    '6':'wfm/1-Internal_45mm33_Ch2.wfm',
    '7':'wfm/6kV_internal_oil_89pF_surface_69pF_N6_L43_HV_plates_Ch2.wfm'
}

print("1.Internal 2.Surface 3.Corona 4.FreeMoving 5.Floating 6.Internal45mm 7.insur")
choice = input("Select Data:")
ch1 = wfmread(ch1_dict[choice])
ch2 = wfmread(ch2_dict[choice])
```

```
data_name = ch1_dict[choice].split('_')[0].split('/')[-1]
for _ in range(10):
    clear_output(wait=True)
    print('Seleted',data_name)

    tunit = ch1.time_scale*1e9
plt.rcParams.update({'font.size': 22})
x = np.arange(0, tunit*ch1.record_length, tunit)
print(ch2.time_scale*1e9, 'nanoseconds per adc')
plt.subplot(2,1,1)
plt.plot(x, ch2.wflist[0])
plt.title('Sample PD Signal')
plt.ylabel('Amplitude')
# plt.xlabel('Nanoseconds')
plt.subplot(2,1,2)
plt.rcParams.update({'font.size': 22})
plt.plot(x, ch1.wflist[0])
plt.title('Sample Sync Signal')
plt.ylabel('Amplitude')
plt.xlabel('t (ns)')
plt.show()

Nframes = len(ch2.wflist)
phase_lst = []
peak_lst = []
for curframe in range(Nframes):
    peak_Idx = np.argmax(abs(ch2.wflist[curframe]))
    peak_V = ch2.wflist[curframe][peak_Idx]
    peak_Phase = ch1.wflist[curframe][peak_Idx]
    peak_lst.append(peak_V)
    phase_lst.append((360/4.54)*peak_Phase)

def scatter_with_gaussian_kde(ax, x, y):
    xy = np.vstack([x, y])
    z = gaussian_kde(xy)(xy)
    ax.scatter(x, y, c=z, s=20, edgecolor='none')

```

```
fig = plt.figure()
fig.set_figheight(8)
fig.set_figwidth(10)
ax = fig.subplots()
scatter_with_gaussian_kde(ax, phase_lst, peak_lst)

abs_peak = list(map(abs, peak_lst))
yrange = (round(max(abs_peak)*100)/100)
x = np.arange(0,360)
y = yrange*np.sin(2*np.pi*1/360*x)

plt.plot(x,y,'r', linewidth=3)
plt.xticks([0,90,180,270,360])
plt.grid()
plt.title('PRPD')
plt.rcParams['font.size'] = '30'
plt.ylabel('Peak Voltage (V)')
plt.xlabel('Peak Phase (Degree)')
plt.show()
# fig.savefig(r'C:\Users\Fusible\Desktop\Thesis\Graphics\Img\corona.eps',
    format='eps', dpi=1000)
```

Listing 5.1 Prior Python code to plot and visualize waveforms into graphs from recorded .wfm file from osilloscope. Plotting PRPD image from the .wfm file.

```
from wfmread import wfmread
from IPython.display import clear_output
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import clear_output
import os
from scipy.optimize import curve_fit
from scipy.stats import gaussian_kde
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.colors import ListedColormap, LinearSegmentedColormap
from PIL import Image
import PIL.ImageOps
import random
import math
ch1_dict = {
    '1':'wfm/InternalNew3_N5_300mV_200M_1us_200_Ch1.wfm',
    '2':'wfm/Surface_N5_500mV_200M_1us_200_Ch1.wfm',
    '3':'wfm/Positive2_N5_10mV_5G_2us_10000sa_Ch1.wfm',
    '4':'wfm/FreeMoving1_N5_50mV_200M_1us_200_Ch1.wfm',
    '5':'wfm/FloatingParticle_N5_2V_1G_1us_1000_Ch1.wfm',
    '6':'wfm/1-Internal_45mm33_Ch1.wfm',
    '7':'wfm/6kV_internal_oil_89pF_surface_69pF_N6_L43_HV_plates_Ch1.wfm'
}
ch2_dict = {
    '1':'wfm/InternalNew3_N5_300mV_200M_1us_200_Ch2.wfm',
    '2':'wfm/Surface_N5_500mV_200M_1us_200_Ch2.wfm',
    '3':'wfm/Positive2_N5_10mV_5G_2us_10000sa_Ch2.wfm',
    '4':'wfm/FreeMoving1_N5_50mV_200M_1us_200_Ch2.wfm',
    '5':'wfm/FloatingParticle_N5_2V_1G_1us_1000_Ch2.wfm',
    '6':'wfm/1-Internal_45mm33_Ch2.wfm',
    '7':'wfm/6kV_internal_oil_89pF_surface_69pF_N6_L43_HV_plates_Ch2.wfm'
}

# print("Creating dataset with classes: 1.Internal 2.Surface 3.Corona 4.
    Freemoving 5.Floating")
```

```
36
37 import numpy as np
38 import matplotlib.pyplot as plt
39 from matplotlib import cm
40 from matplotlib.colors import Normalize
41 from scipy.interpolate import interpn
42
43 def density_scatter( x , y, ax = None, sort = True, bins = 20, **kwargs ) :
44     """
45     Scatter plot colored by 2d histogram
46     """
47     if ax is None :
48         fig , ax = plt.subplots()
49         fig.set_figheight(6)
50         fig.set_figwidth(6)
51     data , x_e, y_e = np.histogram2d( x, y, bins = bins, density = True )
52     z = interpn( ( 0.5*(x_e[1:] + x_e[:-1]) , 0.5*(y_e[1:]+y_e[:-1]) ) , data
           , np.vstack([x,y]).T , method = "splinef2d", bounds_error = False)
53
54     #To be sure to plot all data
55     z[np.where(np.isnan(z))] = 0.0
56
57     # Sort the points by density, so that the densest points are plotted last
58     if sort :
59         idx = z.argsort()
60         x, y, z = x[idx], y[idx], z[idx]
61
62     ax.scatter( x, y, c=z, **kwargs )
63
64 internal = {}
65 internal2 = {}
66 surface = {}
67 corona = {}
68 insur = {}
69 insur2 = {}
70
71 internal['ch1'] = wfmread(ch1_dict['1']).wflist
```

```
internal['ch2'] = wfmread(ch2_dict['1']).wflist
internal2['ch1'] = wfmread(ch1_dict['6']).wflist
internal2['ch2'] = wfmread(ch2_dict['6']).wflist
surface['ch1'] = wfmread(ch1_dict['2']).wflist
surface['ch2'] = wfmread(ch2_dict['2']).wflist
corona['ch1'] = wfmread(ch1_dict['3']).wflist
corona['ch2'] = wfmread(ch2_dict['3']).wflist
insur['ch1'] = np.concatenate((internal['ch1'],surface['ch1']),axis = 0)
insur['ch2'] = np.concatenate((internal['ch2'],surface['ch2']),axis = 0)
insur2['ch1'] = wfmread(ch1_dict['7']).wflist
insur2['ch2'] = wfmread(ch2_dict['7']).wflist
for i in range(len(insur2['ch2'])):
    insur2['ch2'][i] = insur2['ch2'][i]*(-1)
data_list = [internal,internal2,surface,corona,insur,insur2]
dat_name = ['Internal','Internal','Surface','Corona','InternalSurface','
    InternalSurface']
for i in range(len(data_list)):
    ch1 = data_list[i]['ch1']
    ch2 = data_list[i]['ch2']
    Nframes = len(ch1)
    data_name = dat_name[i]

    print('Generating '+data_name+' | '+ str(Nframes)+' pulses')

    rand_idx = np.random.choice(int(Nframes),int(Nframes), replace=False).
        tolist()


    # p_per_img = int(math.floor(Nframes/max_img_amnt))
    datasetAmount = {'Internal':45 \
                    ,'Surface' :90 \
                    ,'Corona' :90 \
                    ,'InternalSurface': 45}
    p_per_img = 100
    max_img_amnt = int(math.floor(Nframes/p_per_img))

    print(str(max_img_amnt)+ ' images |' +str(p_per_img)+' points')
```

```
107
108
109     n = 0;
110     while n < max_img_amnt:
111         # print(str(n+1),end = ', ')
112         phase_lst = []
113         peak_lst = []
114         for _ in range(p_per_img):
115             s = rand_idx.pop()
116             curr_ch1 = ch1[s]
117             curr_ch2 = ch2[s]
118             peak_Idx = np.argmax(abs(curr_ch2))
119             peak_V = curr_ch2[peak_Idx]
120             peak_Phase = curr_ch1[peak_Idx]
121             peak_lst.append(peak_V)
122             phase_lst.append((360/4.54)*peak_Phase)
123         x = np.array(phase_lst)
124         y = np.array(peak_lst)
125
126
127
128         fig = plt.figure(frameon=False)
129         fig.set_size_inches(2.56,2.56)
130         ax = plt.Axes(fig, [0., 0., 1., 1.])
131         ax.set_axis_off()
132         abs_peak = list(map(abs, peak_lst))
133         yrange = (round(max(abs_peak)*100)/100)
134         fig.add_axes(ax)
135         x = np.array(phase_lst)
136         y = np.array(peak_lst)
137         try:
138             posP = max(y[x<180])
139             posP = np.format_float_scientific(posP, precision = 2, exp_digits
                    =1)
140         except:
141             posP = '0'
142         try:
```

```
143             negP = max(y[x>=180])
144             negP = np.format_float_scientific(negP, precision = 2, exp_digits
                    =1)
145         except:
146             negP = '0'
147         ax.hist2d(x, y, (60,100), cmap=plt.cm.binary)
148         ax.set_xlim([0,360])
149         ax.set_ylim([-yrange*1.5,yrange*1.5])
150         # fname = data_name+"("+str(per*100)+")[p"+posP+"][n"+negP+"]_"+ str(
                int(np.random.rand(1)*10**8))+'.png';
151         fname = data_name+"[p"+posP+"][n"+negP+"]_"+ str(int(np.random.rand
                (1)*10**8))+'.png';
152
153
154
155
156
157         # if n >= datasetAmount[data_name]:
158         #     path = '2ddata/.unused/'
159         #     if not os.path.exists(path+data_name):
160         #         os.makedirs(path+data_name)
161         # else:
162         #     path = '2ddata/.dataset/'
163         #     if not os.path.exists(path+data_name):
164         #         os.makedirs(path+data_name)
165         path = '2ddata/.dataset/'
166         if not os.path.exists(path+data_name):
167             os.makedirs(path+data_name)
168
169         fig.savefig('%s%s/%s' %(path,data_name,fname), bbox_inches='tight',
                pad_inches = 0)
170         image = Image.open('%s%s/%s' %(path,data_name,fname))
171         image = PIL.ImageOps.grayscale(image)
172         image = PIL.ImageOps.invert(image)
173
174         image.save('%s%s/%s' %(path,data_name,fname))
175         #
```

```
176
177         plt.close('all')
178         plt.clf()
179         plt.cla()
180         n+=1
181
182 print('file generation completed')
```

Listing 5.2 PRPD image generation of simple dataset without severity.

```
from wfmread import wfmread
from IPython.display import clear_output
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import clear_output
import os
from scipy.optimize import curve_fit
from scipy.stats import gaussian_kde
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.colors import ListedColormap, LinearSegmentedColormap
from PIL import Image
import PIL.ImageOps
import random
import math
ch1_dict = {
    '1':'wfm/InternalNew3_N5_300mV_200M_1us_200_Ch1.wfm',
    '2':'wfm/Surface_N5_500mV_200M_1us_200_Ch1.wfm',
    '3':'wfm/Positive2_N5_10mV_5G_2us_10000sa_Ch1.wfm',
    '4':'wfm/FreeMoving1_N5_50mV_200M_1us_200_Ch1.wfm',
    '5':'wfm/FloatingParticle_N5_2V_1G_1us_1000_Ch1.wfm',
    '6':'wfm/1-Internal_45mm33_Ch1.wfm',
    '7':'wfm/6kV_internal_oil_89pF_surface_69pF_N6_L43_HV_plates_Ch1.wfm'
}
ch2_dict = {
    '1':'wfm/InternalNew3_N5_300mV_200M_1us_200_Ch2.wfm',
    '2':'wfm/Surface_N5_500mV_200M_1us_200_Ch2.wfm',
    '3':'wfm/Positive2_N5_10mV_5G_2us_10000sa_Ch2.wfm',
    '4':'wfm/FreeMoving1_N5_50mV_200M_1us_200_Ch2.wfm',
    '5':'wfm/FloatingParticle_N5_2V_1G_1us_1000_Ch2.wfm',
    '6':'wfm/1-Internal_45mm33_Ch2.wfm',
    '7':'wfm/6kV_internal_oil_89pF_surface_69pF_N6_L43_HV_plates_Ch2.wfm'
}

# print("Creating dataset with classes: 1.Internal 2.Surface 3.Corona 4.
    Freemoving 5.Floating")
```

```
36
37 import numpy as np
38 import matplotlib.pyplot as plt
39 from matplotlib import cm
40 from matplotlib.colors import Normalize
41 from scipy.interpolate import interpn
42
43 def density_scatter( x , y, ax = None, sort = True, bins = 20, **kwargs ) :
44     """
45     Scatter plot colored by 2d histogram
46     """
47     if ax is None :
48         fig , ax = plt.subplots()
49         fig.set_figheight(6)
50         fig.set_figwidth(6)
51     data , x_e, y_e = np.histogram2d( x, y, bins = bins, density = True )
52     z = interpn( ( 0.5*(x_e[1:] + x_e[:-1]) , 0.5*(y_e[1:]+y_e[:-1]) ) , data
           , np.vstack([x,y]).T , method = "splinef2d", bounds_error = False)
53
54     #To be sure to plot all data
55     z[np.where(np.isnan(z))] = 0.0
56
57     # Sort the points by density, so that the densest points are plotted last
58     if sort :
59         idx = z.argsort()
60         x, y, z = x[idx], y[idx], z[idx]
61
62     ax.scatter( x, y, c=z, **kwargs )
63
64 internal = {}
65 internal2 = {}
66 surface = {}
67 corona = {}
68 insur = {}
69 insur2 = {}
70
71 internal['ch1'] = wfmread(ch1_dict['1']).wflist
```

```
internal['ch2'] = wfmread(ch2_dict['1']).wflist
internal2['ch1'] = wfmread(ch1_dict['6']).wflist
internal2['ch2'] = wfmread(ch2_dict['6']).wflist
surface['ch1'] = wfmread(ch1_dict['2']).wflist
surface['ch2'] = wfmread(ch2_dict['2']).wflist
corona['ch1'] = wfmread(ch1_dict['3']).wflist
corona['ch2'] = wfmread(ch2_dict['3']).wflist
insur['ch1'] = np.concatenate((internal['ch1'],surface['ch1']),axis = 0)
insur['ch2'] = np.concatenate((internal['ch2'],surface['ch2']),axis = 0)
insur2['ch1'] = wfmread(ch1_dict['7']).wflist
insur2['ch2'] = wfmread(ch2_dict['7']).wflist
for i in range(len(insur2['ch2'])):
    insur2['ch2'][i] = insur2['ch2'][i]*(-1)
data_list = [internal,internal2,surface,corona,insur,insur2]
dat_name = ['Internal','Internal','Surface','Corona','InternalSurface','
    InternalSurface']
for i in range(len(data_list)):
    ch1 = data_list[i]['ch1']
    ch2 = data_list[i]['ch2']
    Nframes = len(ch1)
    data_name = dat_name[i]

    print('Generating '+data_name+' | '+ str(Nframes)+' pulses')

    rand_idx = np.random.choice(int(Nframesrames),int(Nframes), replace=False
        ).tolist()


    # p_per_img = int(math.floor(Nframes/max_img_amnt))

    p_per_img = [20,100,500]
    n_level = ['mild','moderate','severe']
    for p_in in p_per_img:

        level = n_level[p_per_img.index(p_in)]
        data_name = dat_name[i]+"_"+level
        print(f'Generating {level}')
```

```
107         rand_idx = np.random.choice(int(Nframes),int(Nframes), replace=False)
                .tolist()
108         # max_img_amnt = int(math.floor(Nframes/p))
109         # print(str(max_img_amnt)+ ' images |' +str(p)+' points')
110         n = 0;
111         while len(rand_idx) > p_in*1.3:
112             # print(str(n+1),end = ', ')
113             phase_lst = []
114             peak_lst = []
115             dif = random.uniform(0.7, 1.4)
116             p = math.floor(p_in*dif)
117             for _ in range(p):
118                 s = rand_idx.pop()
119                 curr_ch1 = ch1[s]
120                 curr_ch2 = ch2[s]
121                 peak_Idx = np.argmax(abs(curr_ch2))
122                 peak_V = curr_ch2[peak_Idx]
123                 peak_Phase = curr_ch1[peak_Idx]
124                 peak_lst.append(peak_V)
125                 phase_lst.append((360/4.54)*peak_Phase)
126             x = np.array(phase_lst)
127             y = np.array(peak_lst)
128             if len(x) <5:
129                 break
130 
131 
132             fig = plt.figure(frameon=False)
133             fig.set_size_inches(2.56,2.56)
134             ax = plt.Axes(fig, [0., 0., 1., 1.])
135             ax.set_axis_off()
136             abs_peak = list(map(abs, peak_lst))
137             yrange = (round(max(abs_peak)*100)/100)
138             fig.add_axes(ax)
139             x = np.array(phase_lst)
140             y = np.array(peak_lst)
141             try:
142                 posP = max(y[x<180])
```

```
143             posP = np.format_float_scientific(posP, precision = 2,
                    exp_digits=1)
144         except:
145             posP = '0'
146         try:
147             negP = max(y[x>=180])
148             negP = np.format_float_scientific(negP, precision = 2,
                    exp_digits=1)
149         except:
150             negP = '0'
151         ax.hist2d(x, y, (60,100), cmap=plt.cm.binary)
152         ax.set_xlim([0,360])
153         ax.set_ylim([-yrange*1.5,yrange*1.5])
154         # fname = data_name+"("+str(per*100)+")[p"+posP+"][n"+negP+"]_"+
                str(int(np.random.rand(1)*10**8))+'.png';
155         uID = str(int(np.random.rand(1)*10**8))
156         # fname = str(p)+_+data_name+"[p"+posP+"][n"+negP+"]_"+ str(int(np
                .random.rand(1)*10**8))+'.png';
157         fname = (f"{str(p)}_{data_name}[p{posP}][n{negP}]_{uID}.png")
158
159
160
161
162
163         # if n >= datasetAmount[data_name]:
164         #     path = '2ddata/.unused/'
165         #     if not os.path.exists(path+data_name):
166         #         os.makedirs(path+data_name)
167         # else:
168         #     path = '2ddata/.dataset/'
169         #     if not os.path.exists(path+data_name):
170         #         os.makedirs(path+data_name)
171         path = '2ddata/.dataset/'
172         # if not os.path.exists(path+data_name):
173         #     os.makedirs(path+data_name)
174         if not os.path.exists(f'{path}{data_name}'):
175             os.makedirs(f'{path}{data_name}')
```

```
176             fig.savefig('%s%s/%s' %(path,data_name,fname), bbox_inches='tight'
                    , pad_inches = 0)
177             image = Image.open('%s%s/%s' %(path,data_name,fname))
178             image = PIL.ImageOps.grayscale(image)
179             image = PIL.ImageOps.invert(image)
180 
181             image.save('%s%s/%s' %(path,data_name,fname))
182             #
183 
184             plt.close('all')
185             plt.clf()
186             plt.cla()
187             n+=1
188         print(f'{n} images generated')
189 print('file generation completed')
```

Listing 5.3 PRPD image generation of severity dataset.

```

import os
import matplotlib.pyplot as plt
from tensorboard.plugins import projector
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import datasets, layers, models
from keras.models import Sequential
from keras.layers import Conv2D, MaxPool2D, Flatten, Dense, InputLayer,
    Dropout, MaxPooling2D, BatchNormalization, AveragePooling2D
from tensorflow.keras.callbacks import EarlyStopping
from google.colab.patches import cv2_imshow
import cv2
import numpy as np
from tensorflow.keras.models import load_model

from sklearn.model_selection import train_test_split

import tensorflow as tf
from tensorboard.plugins import projector
from tensorflow.keras.preprocessing.sequence import pad_sequences

from google.colab import drive
drive.mount('/content/drive')
path = "/content/drive/MyDrive/PD_Dataset/pythongen/FinalData/
    random_severity_vary"
os.chdir(path)
# unused_path = '/content/drive/MyDrive/PD_Dataset/pythongen/FinalData/.
    unused'
batch_size = 8
img_height = 256
img_width = 256
data_dir = path
seed = 123
train_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
```

```
  validation_split=0.2,
  subset="training",
  seed=seed,
  image_size=(img_height, img_width),
  batch_size=batch_size,
  label_mode='categorical',
  color_mode='grayscale')
  val_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="validation",
  seed=seed,
  image_size=(img_height, img_width),
  batch_size=batch_size,
  label_mode='categorical',
  color_mode='grayscale')
class_names = train_ds.class_names
print(class_names)
train_size = int(0.5 * len(val_ds))
test_ds = val_ds.take(train_size)
val_ds = val_ds.skip(train_size)
print(int(test_ds.__len__())*batch_size)
print(int(val_ds.__len__())*batch_size)
AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
test_ds = test_ds.cache().prefetch(buffer_size=AUTOTUNE)
model=[]
keras.backend.clear_session()
model=Sequential()
num_classes = len(class_names)

model = tf.keras.Sequential([
  tf.keras.layers.Rescaling(1./255),
  tf.keras.layers.Conv2D(3, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
```

```
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(6, activation='relu'),
  tf.keras.layers.Dense(num_classes)
])

opt = keras.optimizers.Adam(learning_rate=0.001)
model.compile(loss=tf.losses.CategoricalCrossentropy(from_logits=True),
    optimizer=opt, metrics=['accuracy'])
early_stop=EarlyStopping(monitor='val_loss',mode='min',verbose=1,patience=5)
history = model.fit(train_ds, validation_data=val_ds, epochs=15,shuffle=True
    ,batch_size=batch_size)
import matplotlib.pyplot as plt

# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
#get labels from validation dataset
y_test = np.concatenate([y for x, y in val_ds], axis=0)

y_pred = model.predict(val_ds)
#convert from logit to predicted class
for i in range(len(y_pred)):
  max_idx = np.argmax(y_pred[i])
  y_pred[i] = [0]*num_classes
```

```
  y_pred[i][max_idx] = 1
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
plt.rcParams['font.size'] = '16'
plt.rcParams["figure.figsize"] = (10,10)
confusion = confusion_matrix( y_test.argmax(axis=1), y_pred.argmax(axis=1))
disp = ConfusionMatrixDisplay(confusion_matrix=confusion, display_labels=
    class_names)
disp.plot(include_values=True, cmap="viridis", ax=None, xticks_rotation="
    vertical",colorbar=False)
plt.show()

#importing accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import accuracy_score, precision_score, recall_score,
    f1_score
print('\nAccuracy: {:.2f}\n'.format(accuracy_score(y_test, y_pred)))

print('Micro Precision: {:.2f}'.format(precision_score(y_test, y_pred,
    average='micro')))
print('Micro Recall: {:.2f}'.format(recall_score(y_test, y_pred, average='
    micro')))
print('Micro F1-score: {:.2f}\n'.format(f1_score(y_test, y_pred, average='
    micro')))

print('Macro Precision: {:.2f}'.format(precision_score(y_test, y_pred,
    average='macro')))
print('Macro Recall: {:.2f}'.format(recall_score(y_test, y_pred, average='
    macro')))
print('Macro F1-score: {:.2f}\n'.format(f1_score(y_test, y_pred, average='
    macro')))

print('Weighted Precision: {:.2f}'.format(precision_score(y_test, y_pred,
    average='weighted')))
print('Weighted Recall: {:.2f}'.format(recall_score(y_test, y_pred, average=
    'weighted')))
print('Weighted F1-score: {:.2f}'.format(f1_score(y_test, y_pred, average='
    weighted')))
```

```
131
132 from sklearn.metrics import classification_report
133 print('\nClassification Report\n')
134 print(classification_report(y_test, y_pred, target_names=class_names))
```

Listing 5.4 CNN training

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import math
import os
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
import pandas as pd
from pandas import DataFrame
import pandas as pd
import numpy as np
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn import datasets
from sklearn.metrics import ConfusionMatrixDisplay
# plt.rcParams["figure.facecolor"] = 'white'
# plt.rcParams['figure.figsize'] = (5, 5)
# fig, (ax1, ax2, ax3) = plt.subplots(1,3)
# fname = "2ddata\FreeMoving1\FreeMoving1(1.0)[p2.54e-1][n1.32e-1]_97336171.
    png"
# img = Image.open(fname)
# ax1.imshow(img, cmap='gray')
# posH = img.crop((0, 0, 128, 256))
# negH = img.crop((128, 0, 256, 256))
# ax2.imshow(posH,cmap='gray')
# ax3.imshow(negH,cmap='gray')

def img_stats(path):
    img = Image.open(path)
    c = path.split('\\')[1]
    u_P = path.split('[p')[1].split("]")[0]
```

```
    u_N = path.split('[n')[1].split("]")[0]
    posH = img.crop((0, 0, 128, 256))
    pixelCount = np.array(posH.histogram())
    GLs = np.array(range(256))
    numberPixel = np.array(sum(pixelCount))
    meanGL = sum(pixelCount*GLs)/numberPixel
    if meanGL > 0.001:
        varianceGL_P = round(sum((GLs - meanGL)**2 * pixelCount) / (
            numberPixel-1),2)
        sd = round(math.sqrt(varianceGL_P),2)
        skew_P = round(sum((GLs - meanGL)**3 * pixelCount) / ((numberPixel -
            1) * sd**3),2)
        kurtosis_P = round(sum((GLs - meanGL)**4 * pixelCount) / ((
            numberPixel - 1) * sd**4),2)
    else:
        varianceGL_P = 0
        sd = 0
        skew_P = 0
        kurtosis_P = 0

    negH = img.crop((128, 0, 256, 256))
    pixelCount = np.array(negH.histogram())
    GLs = np.array(range(256))
    numberPixel = np.array(sum(pixelCount))
    meanGL = sum(pixelCount*GLs)/numberPixel
    if meanGL > 0.001:
        varianceGL_N = round(sum((GLs - meanGL)**2 * pixelCount) / (
            numberPixel-1),2)
        sd = round(math.sqrt(varianceGL_N),2)
        skew_N = round(sum((GLs - meanGL)**3 * pixelCount) / ((numberPixel -
            1) * sd**3),2)
        kurtosis_N = round(sum((GLs - meanGL)**4 * pixelCount) / ((
            numberPixel - 1) * sd**4),2)
    else:
        varianceGL_N = 0
        sd = 0
        skew_N = 0
```

```
        kurtosis_N = 0
    return [c,skew_P,skew_N,kurtosis_P,kurtosis_N,varianceGL_P,varianceGL_N]
#     return [c,u_P,u_N,skew_P,skew_N,kurtosis_P,kurtosis_N,varianceGL_P,
    varianceGL_N]

files_Dir = []
for root, dirs, files in os.walk("2ddata/random_severity_vary"):
    for file in files:
        if file.endswith(".png"):
            files_Dir.append(os.path.join(root, file))
all_Stats = []
for file in files_Dir:
    all_Stats.append(img_stats(file))
pd_Data = DataFrame(all_Stats)
# names = ['class', 'u+max', 'u-max', 'sk+', 'sk-','ku+','ku-','var+','var
    -']
names = ['class', 'sk+', 'sk-','ku+','ku-','var+','var-']
pd_Data.columns = names
all_Class = list(set(pd_Data['class']))
print(f"Found {len(files_Dir)} files, belonging to {len(all_Class)} classes"
    )
print(f"{all_Class}")
x = pd_Data.iloc[:, 1:].values
y = pd_Data.iloc[:, 0].values
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.10)
print(f"train: {len(X_train)} valid: {len(X_test)}")

# Creating training and test split

# X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3,
    random_state=1, stratify = y)
plt.rcParams['font.size'] = '8'
# plt.rcParams['font.size'] = '10'
plt.rcParams["figure.figsize"] = (4,4)
# Feature Scaling

sc = StandardScaler()
```

```
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

# Training a SVM classifier using SVC class
svm = SVC(kernel= 'linear', random_state=0, C=0.03)
svm.fit(X_train_std, y_train)

# Mode performance
names = svm.classes_
y_pred = svm.predict(X_test_std)
# print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
confusion = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=confusion, display_labels=
    names)
disp.plot(include_values=True, cmap="viridis", ax=None, xticks_rotation="
    vertical",colorbar=False)
plt.show()
```

Listing 5.5 SVM feature extraction and training, testing

BIOGRAPHY

Name	Jakrin Butdee
Education	2020: Bachelor of Engineering
	(Electronic and Communication Engineering)
	Sirindhorn International Institute of Technology
	Thammasat University

Publication

Butdee, J., Laimek, P., Kongprawechnon, W., Kondo, T., Leelasawassuk, T. & Wongcumchang, N. (2020). Automate Pre-Screening System of Posterior Capsular Opacification (PCO). *17th International Conference On Electrical Engineering/ Electronics, Computer, Telecommunications And Information Technology (ECTI-CON).* doi: 10.1109/ECTI-CON49241.2020.9158283