Transmission Line Insulator Fault Classification and Model Size Reduction

Course No.: EE 4000
Course Title: Project and Thesis

by
Shetu Mohanto
1703018

Supervised by
Dr. Md. Salah Uddin Yusuf
Professor
Department of Electrical and Electronic Engineering
Khulna University of Engineering & Technology (KUET)
Khulna-9203, Bangladesh

A thesis report submitted to the Department of Electrical and Electronic Engineering in partial fulfillment of the requirements for the degree of Bachelor of Science in Electrical and Electronic Engineering

Department of Electrical and Electronic Engineering (EEE)
Khulna University of Engineering & Technology (KUET)
Khulna-9203, Bangladesh
February 2023

Declaration

It is hereby declared that

1. I confirm that the thesis I presented during my time at Khulna University of Engineering & Technology was entirely created by me and is my original work.

2. The thesis has not incorporated any content from external sources that has already been published or written by someone else, unless it has been properly acknowledged through comprehensive and precise referencing.

3. The thesis does not include any material that has been approved or presented for any other degree or diploma program at any university or educational institution.

4. I have given proper credit to all the significant sources of assistance that I have received.

Student's Full Name & Signature

Shetu Mohanto
Roll: 1703018

Approval

The undersigned hereby certifies for recommend to the Department of Electrical and Electronic Engineering for acceptance of a thesis entitled **"Transmission Line Insulator Fault Classification and Model Size Reduction"** by Shetu Mohanto in partial fulfillment of the requirements for the degree of Bachelor of Science in Electrical and Electronic Engineering, at Khulna University of Engineering & Technology, Khulna-9203, Bangladesh.

Supervisor

Dr. Md. Salah Uddin Yusuf
Professor
Department of Electrical and Electronic Engineering
Khulna University of Engineering & Technology (KUET)
Khulna-9203, Bangladesh

Signature of the supervisor

Acknowledgement

With gratitude to the almighty, I was fortunate enough to conduct my research and complete it on schedule.

I would like to express my sincere gratitude to my thesis supervisor, **Dr. Md. Salah Uddin Yusuf**, Professor of the Department of Electrical and Electronic Engineering at Khulna University of Engineering & Technology, Khulna-9203, Bangladesh. His constant guidance and invaluable insights into machine learning and deep learning have been instrumental in my research on **Transmission Line Insulator Fault Classification and Model Size Reduction**. During challenging times, he has provided me with encouragement and unwavering support, and his dedication to tackling significant research problems has been a tremendous source of inspiration.

I am also thankful to my friends and family members who have supported and inspired me in both my academic and personal life.

I want to thank **Prof. Song Han** from **MIT**, for his amazing and resourceful course on efficient deep learning, which helped me a lot during my research. I am also grateful to **Andrew Ng**, his course on deep learning helped to build my foundation in this field.

Finally and significantly, I would like to express my deep appreciation to my parents for their unwavering love and support. They have always encouraged me to persevere through obstacles and have demonstrated unwavering faith in my abilities. Without their unwavering support, I would not have achieved this feat.

Thanks to all
Author

“Dedicated to
My Beloved Parents
and
Honorable Thesis Supervisor
Prof. Dr. Md. Salah Uddin Yusuf”

Abstract

In contemporary daily life, electricity plays the driving power of society. To ensure the transmission of electricity 24x7 from generation to the customer, stable transmission infrastructure is required and healthy insulators are essential to maintain the stability. Nevertheless, maintenance of the insulators manually is time-consuming and required numerous safety inspections. An autonomous check-up is both quick and safe. In this context, recent literature has focused on the detection of insulator faults. The deep learning approach has emerged as a highly effective tool in detecting objects from images using state-of-the-art models. However, optimizing these models is crucial to enhance their inference rate and reduce their memory requirements. In this literature classification of insulator faults using the YOLOv5 model has been implemented. Furthermore, the pruning and weight clustering methods have been applied to reduce the memory size and increase the inference rate of these models. Overall, the implementation of the YOLOv5 model, coupled with pruning and weight clustering methods, has led to promising results with 99% accuracy in the classification of insulator faults, thereby improving the maintenance process of transmission infrastructure. This study demonstrates the importance of optimization techniques in deep learning models and highlights their potential in addressing critical challenges.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

The insulator plays a crucial role in a transmission line, serving as a barrier that prevents leakage current from the line to the ground through the transmission tower. Unfortunately, insulators are prone to a variety of defects, including bad insulation material, micro cracks in the insulating material, poor glazing on the surface of the insulator, flashover voltage, and mechanical stresses, as noted in [1]. These defects can cause instability in the power system and lead to insulator breakdown. In particular, defects resulting from flashover voltage and broken parts are visually identifiable, as shown in fig. 2.2a and fig. 2.2b, making them amenable to detection using image data.
In recent years, deep learning architectures have made significant advances in object detection from images, with simple convolutional models, SSD, RCNN, YOLO, and other models being widely used in autonomous tasks involving object detection. Given that flashover voltage and broken parts can be visually identified, a single-stage detection model such as YOLO has been employed to detect these faults from image data, which can be later deployed to the inspection task, as mentioned earlier.
However, deep learning architectures can be large in size and require significant amounts of memory to store, as well as considerable time to transmit over networks. Moreover, they are power-intensive due to the numerous multiplications and additions required during computation, as pointed out in [2]. To optimize the size and computing time of deep learning architectures, techniques such as pruning and weight clustering can be used [2]. These techniques can significantly reduce the number of parameters needed to represent the model, resulting in more efficient deployment of real systems. By pruning and weight clustering the model, it is possible to minimize the time and resources required for transmission, storage, and computation. As a result, the model can be efficiently deployed in real-world settings, allowing for more effective detection of insulator defects and improved stability in the power system.

1.2 Research Background

To detect a class of objects in an image two types of detectors can be used, two-stage and single-stage. Two-stage requires more computation times compared to the single-stage. Single-stage can achieve higher accuracy with less computation time. State-of-art single stage models are, YOLO [3], YOLOv7 [4], YOLOv3 [5], SSD [6], Faster R-CNN [7], YOLO9000 [8]. Some papers [9]–[12] also suggest their own modified models which perform well on the special dataset (insulator detection) unlike benchmark (Microsoft COCO, ImageNet) ones.
All these works have been done on various datasets and some of them contain really

poor images and are not good enough to compare with real-world data. Some have used synthesized data, which is placing the same insulator on various backgrounds. For faults, all are considered missing or broken insulators (table 1.1). The detection of flashover voltage has been studied only in one work [9]. So these models aren't practical for inspection tasks.
After achieving a decent performance on any detection model it is important to analyze its feasibility in deploying to the real-world problem. Factors considered are its size, inference time, energy consumption, complexity, robustness to the real-world noisy data, and also the cyber attack or other modification. Though there is a lot of work present on insulator detection, no one has done an analysis of the size and robustness of the model, which is essential for deployment.
As mentioned in [2], deep learning models can be compressed using techniques like pruning, weight clustering, and quantization. These processes reduce the effective parameters of the network which as result reduces the size and inference time of the model. In [13], YOLO model has been pruned out for efficient object detection on dataset MS-COCO and VisDrone. Face mask has been detected using raspberry pi 4 with the pruned and quantized YOLO in [14]. Channel pruned YOLOv5 has been developed for fruit detection in [15]. Though these aren't on insulator detection but can provide sufficient insight to this field.

Table 1.1: Different insulator faults analyzed in different literature

Paper reference	**Types of faults detected**	**Model used**	**Dataset**
Reference [9]	Missing, broken, flashover with motion blur	CenterNet	From power company
Reference [10]	Missing insulator	Modified network using VGG and ResNet	CPLID
Reference [11]	Missing insulator	Ensemble Learning	Collected using UAV
Reference [12]	Missing insulator	GAN, Image generation	CPLID
Reference [16]	Missing insulator	SSD	Collected using UAV
Reference [17]	Missing insulator	Faster R-CNN	Collected manually
Reference [18]	Missing insulator	YOLOv5	CPLID

1.3 Related Works

Fault detection in insulators is a crucial task to ensure the safe and reliable functioning of power systems. Numerous research works have been conducted to develop efficient and accurate detection models to automate the process of insulator fault detection. In this regard, several papers have been published, which proposed different methodologies and techniques for detecting insulator faults from image data. In one such study [9], a modified detection model using CenterNet was proposed to detect insulator faults. The proposed model utilized ResNet50 as the backbone and

an additional attention module to boost the feature learning process and filter out non-object features. The dataset used in this study was collected from a power company and included motion blur in the images. The final average precision achieved was 96.16% with a frame rate of 30fps.
Another paper [10] proposed a cascaded region proposal network of VGG and ResNet for detecting insulator faults. The authors applied various image augmentation techniques such as gaussian blur, affine transformation, segmentation, and brightness tweaking to make the dataset more robust to real-world applications. The final precision and recall achieved by the model were 0.91 and 0.96, respectively.
In [11], the authors used images captured by a UAV to build the dataset for insulator fault detection. They utilized ensemble learning and multilevel perception using SSD, which they called deep meta-architecture, to achieve robust detection performance. The final recall and precision of the model were 93.69% and 91.23%, respectively.
The paper [12] proposed a GAN model using the CPLID dataset to generate insulator images. The authors also used the discriminator as the detection model and generated the InsuGenSet dataset for model training.
In [16], the authors trained an SSD-based detection model using aerial images of insulators. They achieved a robust model by using transfer learning and fine-tuning with a two-level approach.
In another study [17], a combination of R-CNN and VGG-16 Net was used for insulator fault detection. The final average precision achieved by the model was 0.818.
Finally, a recent study [18] proposed a detection model using YOLOv5 and synthesized images with a foggy environment. The model achieved an F1 score of 96.2% for insulator fault detection.
In conclusion, the research conducted on insulator fault detection has proposed various models and techniques for achieving accurate and efficient detection performance. Each proposed model has its unique strengths and limitations, and researchers continue to explore new methodologies to improve insulator fault detection performance.

1.4 Motivation

After carefully examining the previous discussion, it is evident that insulator faults can be accurately detected using image data. However, due to the lack of available datasets in this field, the author had to collect most of the data required for training the detection model from the power company. Unfortunately, publicly available datasets are not rich and do not possess the quality necessary for comprehensive analysis. One such dataset that stands out is the EPRI Insulator Defect Image Dataset [19], which provides high-quality images of insulators captured by UAVs and offers detailed labeling of faults, as presented in section 3.1. Despite the dataset's advantages, no research has yet been conducted on it.
Furthermore, prior research on this topic has mainly focused on improving model accuracy and analyzing the model's preparation. However, no literature has presented an analysis of the model's size and storage capacity, which is critical to assess the deployment challenges of the inspection task.
Therefore, it is necessary to develop a detection model using insulator image data from the EPRI dataset and conduct an analysis of the model's size and storage capacity. Doing so will contribute significantly to the field of autonomous recognition

of insulator faults.

1.5 Objectives

The main objectives of this thesis are,

1. To prepare the insulator dataset according to the input format of the YOLOv5.
2. To prepare and train detection model.
3. To conduct an analysis of unstructured pruning on the performance of the detection model.
4. To conduct an analysis of weight clustering on the performance of the detection model.
5. To conduct an analysis of structured pruning on the performance of the detection model.
6. To find out the best detection model with reduced parameters.
7. To implement an API for report generation regarding insulator faults from the input image.

1.6 Contribution

Contributions can be listed as

1. Developed a detection model using the EPRI insulator image dataset.
2. Developed a reduced size detection model with the performance of the base model.
3. Developed an API system to automatically generate reports regarding insulator faults from the input image.

Chapter 2

Relevant Theories

2.1 Transmission Line Insulator

In transmission lines, suspension type fig. 2.1a and strain type fig. 2.1b insulators are employed.

(a) Suspension type insulator

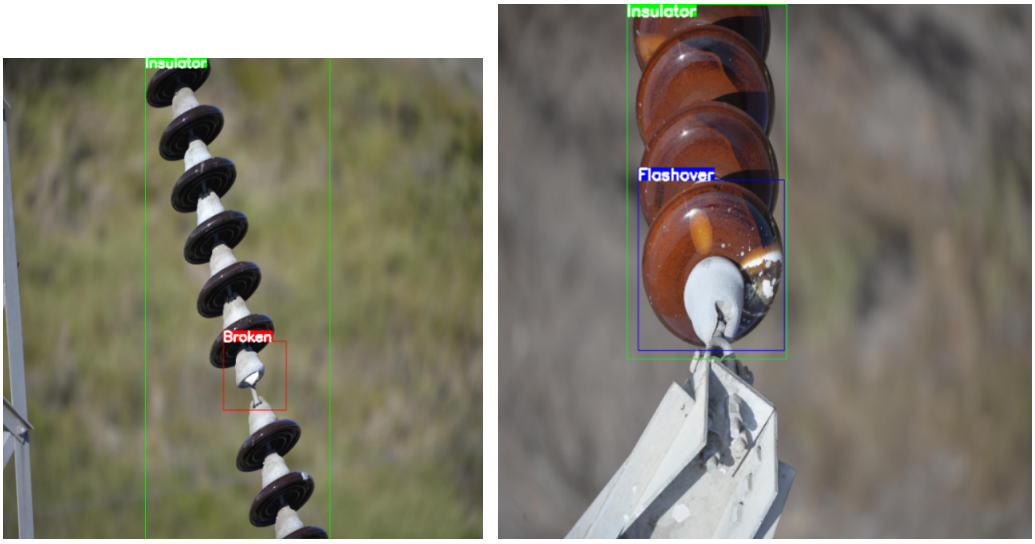
(b) Strain type insulator

Figure 2.1: Insulator used in transmission lines. [19]

The common faults of the insulator in transmission lines are broken or missing disks fig. 2.2a and surface faults fig. 2.2b. Surface faults are caused by the flashover voltage. A particular voltage level requires a certain number of disks, the lack of a disk could lead to significant line faults, potentially causing severe errors and network instability. [20].

2.2 Object Detection

Object detection is a key area in the field of computer vision, which involves identifying and localizing objects of interest within an image or video. The primary goal of object detection is to detect the presence of objects in an image and also to identify their location within the image. This task is made possible using a wide range of algorithms, including deep learning-based approaches like convolutional neural networks (CNN) and object proposal methods [21].
Object detection systems may involve a two-stage process: object proposal generation and object classification. Object proposal generation involves generating a set of regions of interest (ROIs) that may contain objects, using algorithms such as selective search, edge boxes, or region proposal networks (RPNs). The second



(a) Broken disk (b) Flashover damage

Figure 2.2: Common faults in insulators. [19]

stage involves classifying each ROI as containing an object or not, using techniques such as CNNs. Object detection model of one-stage directly detects object classes in single stage. A two-stage object detection algorithm is a type of computer vision algorithm that involves two stages or steps to detect objects in an image. The first stage generates a set of candidate object locations, often referred to as object proposals or region proposals, by analyzing the image and identifying areas that are likely to contain objects. The second stage then classifies and refines these proposals to accurately detect and locate the objects within the image. These algorithms are known for their high detection accuracy, but they typically require more computation and processing time compared to one-stage detection algorithms.
In contrast to two-stage object detection algorithms, one-stage object detection algorithms perform object detection in a single step. They typically apply a dense set of object proposals across an image and predict the class and location of each proposed object in one forward pass of the neural network. This makes them computationally efficient and faster than two-stage algorithms. One-stage algorithms usually do not require external region proposal methods, unlike two-stage algorithms that depend on a separate proposal generation step. However, one-stage algorithms may suffer from lower accuracy and precision than two-stage algorithms, especially for small or occluded objects. Some examples of one-stage object detection algorithms include YOLO (You Only Look Once) and SSD (Single Shot Detector).

2.3 Metrics

Object detection involves the use of various metrics to evaluate the performance of detection algorithms. Here are some of the commonly used metrics in object detection:

1. **Intersection over Union (IoU):** IoU is a popular metric used to evaluate the overlap between the ground truth bounding boxes and the predicted bounding boxes. It measures the ratio of the area of overlap between the two boxes to

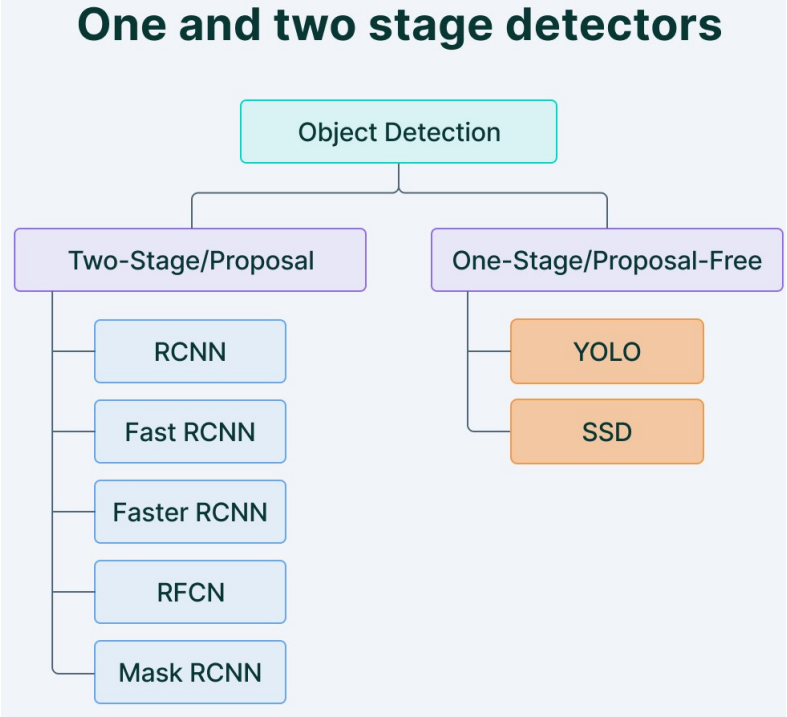


Figure 2.3: Single stage and two stage object detection models.

the area of their union.

$$IoU = \frac{Area\ of\ overlap}{Area\ of\ intersection} \tag{2.1}$$

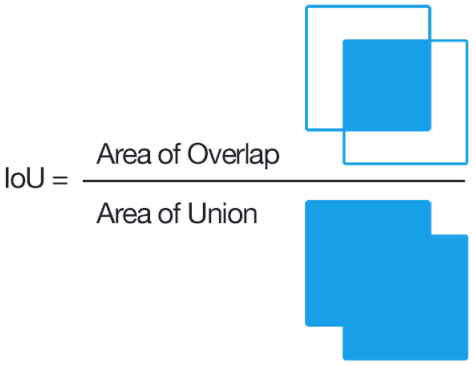


Figure 2.4: Intersection over union (IoU) between two bounding boxes.

2. **Precision and Recall:** These two metrics are used to evaluate the performance of object detection algorithms in terms of true positives (TP), false positives (FP), and false negatives (FN). Precision measures the ratio of the number of true positives to the total number of positive predictions, while recall measures the ratio of the number of true positives to the total number of ground truth positives.

$$Precison = \frac{True\ Positive}{True\ Positive\ +\ False\ Positive} \tag{2.2}$$

$$Recall = \frac{True\ Positive}{True\ Positive\ +\ False\ Negative} \tag{2.3}$$

3. **Mean Average Precision (mAP):** mAP is a popular metric used to evaluate the overall performance of object detection algorithms. It is calculated as the average precision of the algorithm over a range of IoU thresholds. To illustrate, the mean average precision (mAP) calculated for IoU ratios between 0.5 and 0.95 with increments of 0.05 is referred to as **mAP@[0.5:0.05:0.95]**. This is the average over all categories in the dataset.
4. **F1-score:** F1-score is the harmonic mean of precision and recall, and it is a widely used metric in binary classification tasks. It measures the balance between precision and recall, and it is used to evaluate the overall performance of an object detection algorithm.

$$F1\ score = \frac{2 \times Precision \times Recall}{Precision\ +\ Recall} \tag{2.4}$$

2.4 YOLO Object Detection Model

The basic idea behind YOLO is to detect objects in an image by dividing the image into a grid and predicting the class and location of each object within the grid. The YOLO model consists of two main components: a convolutional neural network (CNN) for feature extraction and a fully connected neural network for object detection. The CNN is responsible for extracting features from the input image, while the fully connected network is used to predict the bounding boxes and object classes.

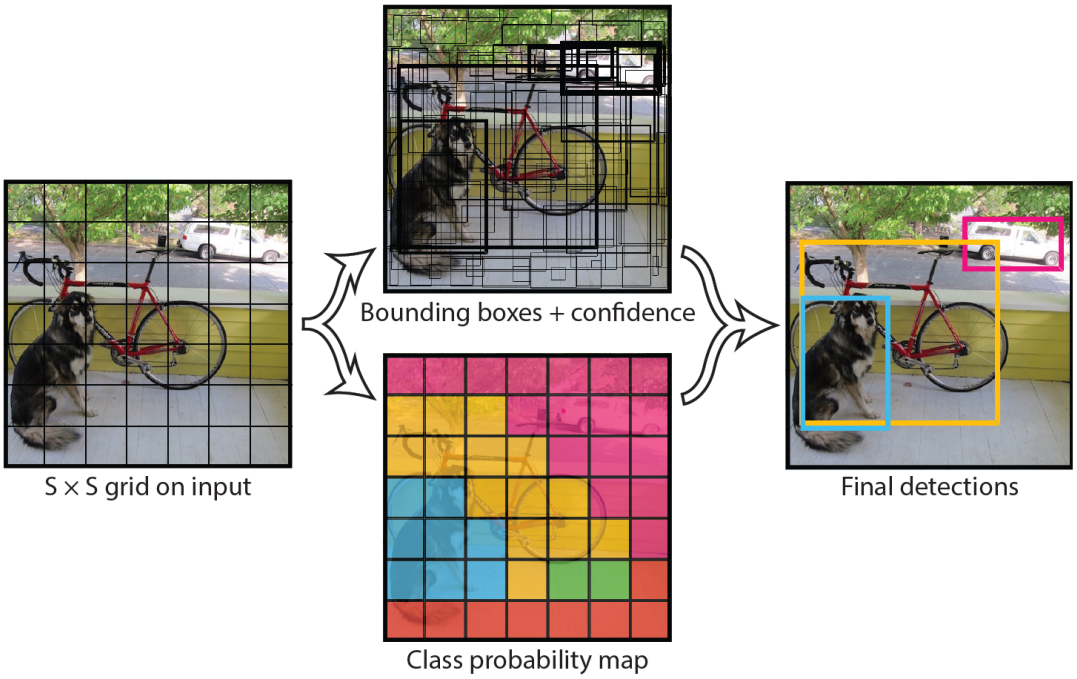


Figure 2.5: Key steps in YOLO approach [3].

The YOLO approach consists of several key steps:

1. **Grid construction:** The image is divided into a grid of cells, each of which is responsible for detecting objects that appear within it.

2. **Object detection:** For each cell in the grid, YOLO predicts the probability of each class of the object being present within the cell and the coordinates of a bounding box around the object.
3. **Non-maximum suppression:** To remove redundant detections, YOLO applies a post-processing step called non-maximum suppression, which removes all but the most confident detections of each object.

One of the main advantages of the YOLO approach is its end-to-end training process, which allows the model to learn directly from raw pixels without requiring pre-processing or post-processing steps. This makes the YOLO approach highly efficient and scalable, enabling it to be used in large-scale systems with high throughput requirements.
Overall, the YOLO approach to object detection is a highly effective and efficient method for detecting objects in real-time, making it a valuable tool for a wide range of applications in computer vision and beyond.

2.5 Pruning of Deep Learning Model

Pruning is a technique used in deep learning to reduce the size of a trained neural network model by removing unimportant or redundant connections, nodes, or layers, without significantly affecting the accuracy of the model. Pruning can be used to simplify a complex model and improve its efficiency by reducing the computational resources required to execute the model.
Pruning can be performed in different ways, including weight pruning, neuron pruning, and filter pruning. Weight pruning involves removing the weights that have the smallest absolute values, while neuron pruning involves removing entire neurons that have the least impact on the output. Filter pruning involves removing entire filters that are less important or redundant.

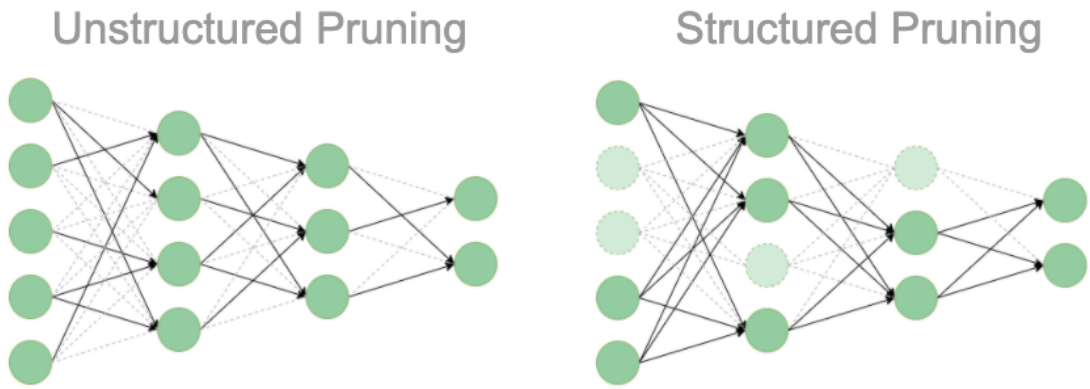


Figure 2.6: Pruning of neural network [22].

Pruning can be applied to both convolutional neural networks (CNNs) and fully connected neural networks. It can be performed during training or after training, and it can be applied to specific layers or the entire network.
Pruning is often used in conjunction with other techniques such as quantization and compression to further reduce the size and improve the efficiency of the model. The

resulting pruned model can have several advantages, including faster inference time, reduced memory requirements, and lower power consumption. Overall, pruning is a useful technique for reducing the size and improving the efficiency of deep learning models, making them more practical for use in resource-constrained environments, such as mobile and embedded devices.

2.6 Weight Clustering Method

Weight clustering is a technique used in deep learning to compress a model by reducing the number of unique weight values. The idea is to group similar weights together and represent them with a single value, thereby reducing the number of unique values in the model. This can result in a smaller model size, which can lead to faster inference time, reduced memory requirements, and lower power consumption.

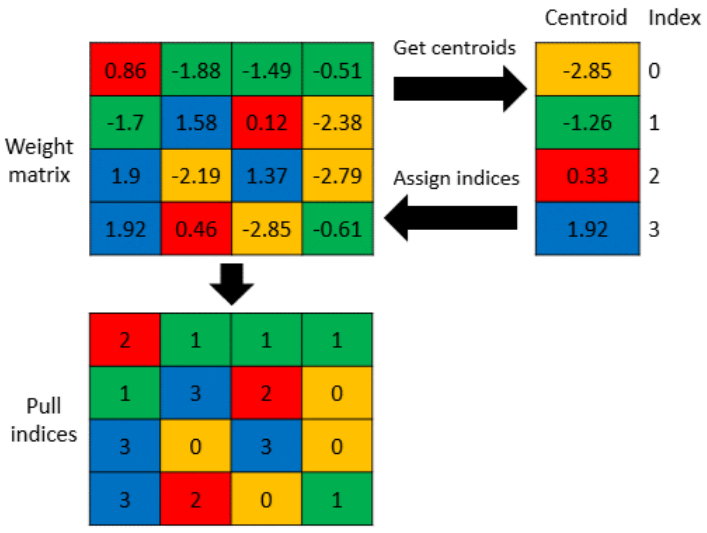


Figure 2.7: Weight Clustering [23]

Weight clustering can be performed in different ways fig. 2.7, including k-means clustering, vector quantization, and product quantization. In k-means clustering, weights are clustered based on their Euclidean distance, while in vector quantization, weights are assigned to a quantized codebook. Product quantization is a hybrid approach that uses a combination of k-means and vector quantization. Weight clustering can be applied to different parts of the model, including the convolutional layers, fully connected layers, or both. It can be applied during training or after training, and it can be used in conjunction with other techniques such as pruning and quantization to further compress the model. Weight clustering has the potential to significantly reduce the size of deep learning models without compromising their accuracy. However, the choice of clustering algorithm and the number of clusters can have a significant impact on the performance of the compressed model. Therefore, careful optimization is required to achieve the right balance between model size and accuracy.

Chapter 3

Materials and Methods

3.1 Dataset

A description of the used dataset is given below with some samples in fig. 3.1

- **EPRI Insulator Defect Image Dataset** [19].
  - Total 1600 images.
  - Images contain labels for the followings,
    1. Insulator string (total sample 1580)
    2. Insulator disks
    3. Flashover damage (total sample 1036)
    4. Broken insulator (total sample 2312)
  - Bounding box labels are provided in JSON file format.

Figure 3.1: Sample image from EPRI Insulator Defect Image Dataset. [19]

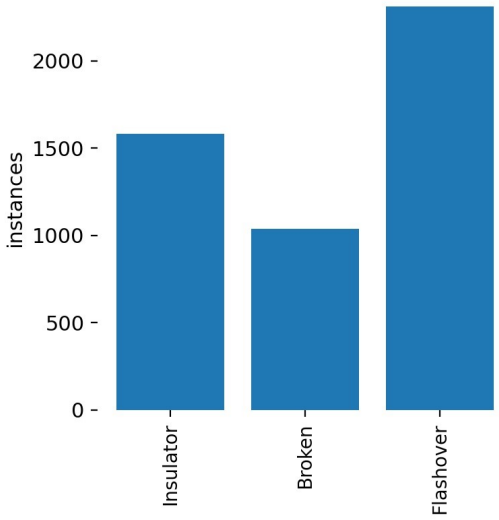


Figure 3.2: Bar plot regarding the total number of objects in dataset [19].

3.2 Modules Used

Ultralytics YOLOv5 is a deep learning model for object detection that builds upon the original YOLO architecture and improves upon it in several ways. Ultralytics, a computer vision and machine learning company, developed YOLOv5 to be faster, more accurate, and more versatile than its predecessors.
One of the main improvements in YOLOv5 is its architecture, which is designed to be more flexible and scalable. YOLOv5 includes a variety of different backbone models, including S, M, L, and X, each with different numbers of layers and computational complexity. This allows users to choose the model that best fits their needs, depending on the size of their dataset, the complexity of their problem, and the resources available.
Another improvement in YOLOv5 is its training process, which uses a novel approach called AutoML to optimize the model's hyperparameters automatically. AutoML is an algorithm that automatically searches for the best hyperparameters for the model, such as the learning rate, batch size, and optimizer, which can improve the model's performance without requiring manual tuning.
Overall, Ultralytics YOLOv5 is a highly optimized and versatile deep learning model for object detection that builds upon the success of the original YOLO architecture. Its flexibility, scalability, and advanced features make it an excellent choice for researchers and practitioners who need a fast and accurate object detection model that can adapt to a variety of different use cases.

3.3 YOLOv5 Architecture

The architecture of YOLOv5 is designed to be flexible and scalable, allowing users to choose from different backbone models with varying sizes and computational complexity. The basic structure of the model follows the YOLOv3 and YOLOv4 architectures, but with several modifications and improvements.

The YOLOv5 architecture consists of several key components:

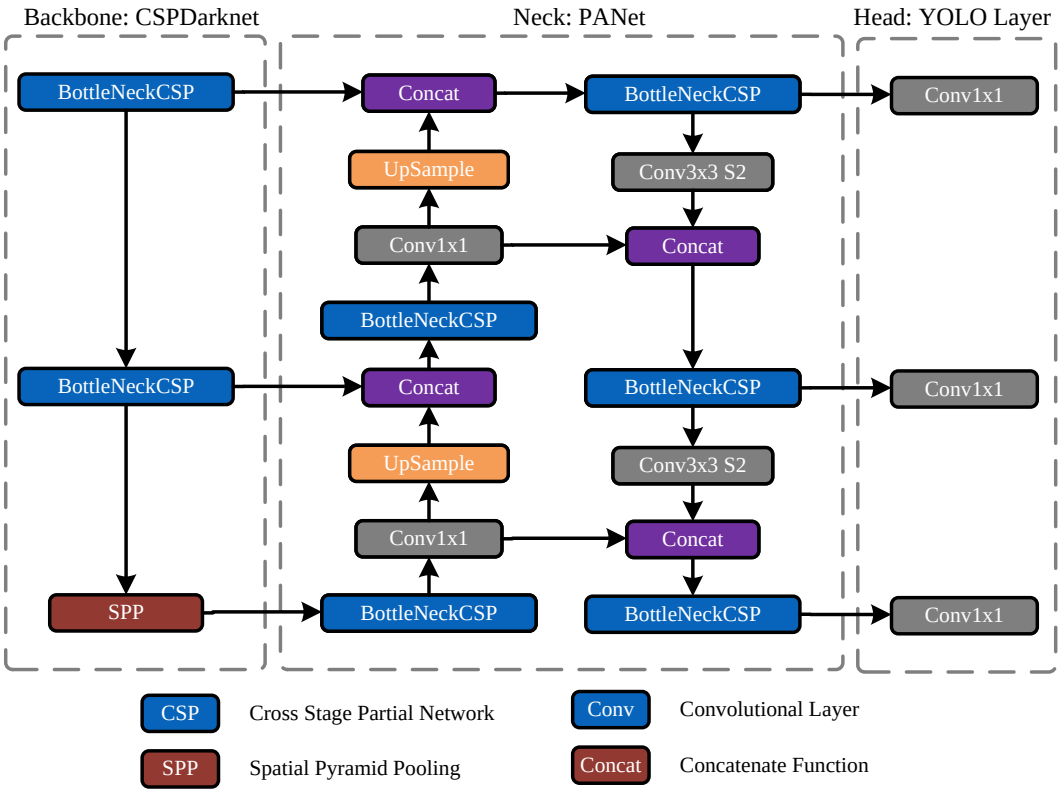


Figure 3.3: Network architecture of YOLOv5.

1. **Backbone:** The backbone network is responsible for extracting features from the input image. YOLOv5 includes several backbone models, including S, M, L, and X, each with different numbers of layers and computational complexity. The smallest model, YOLOv5s, has 8.7 million parameters, while the largest model, YOLOv5x, has 176.5 million parameters.

2. **Neck:** The neck network is used to combine the features extracted by the backbone network into a more compact representation. YOLOv5 uses a spatial pyramid pooling (SPP) module to capture features at multiple scales and a path-aggregation network (PAN) to aggregate features from different layers of the network.

3. **Head:** The head network is responsible for making predictions about the objects in the image. YOLOv5 uses a modified version of the YOLOv3 and YOLOv4 head, which includes anchor-based and anchor-free object detection, multi-scale predictions, and focal loss.

4. **Post-processing:** After the predictions are made, YOLOv5 applies several post-processing steps to refine the output. These steps include non-maximum suppression (NMS), which removes redundant predictions, and score thresholding, which filters out low-confidence predictions.

In addition to these components, YOLOv5 includes several advanced features that improve its accuracy and speed. These include automatic hyperparameter tuning using AutoML, mixed-precision training using half-precision floating-point num-

bers, and multi-scale training and testing. Together, these features make YOLOv5 one of the fastest and most accurate object detection models available.

3.4 Working Procedure

The present study aimed to improve the performance of the YOLOv5l architecture using various techniques. The working procedure of the study, which is illustrated in fig. 3.4, consisted of several steps, including data preprocessing, model training, and evaluation of the model's performance using various metrics. The dataset used in the study was the EPRI dataset, which contained 1600 images with box and object class labels provided in JSON file format. To ensure a balanced distribution of the data, 128 and 64 images were separated from the dataset for the test and validation datasets, respectively.

During the data preprocessing phase, each image was converted to a 640x640 pixel image with corresponding labels saved in a text file in YOLO label format. This was necessary to prepare the data for use in the YOLOv5l architecture. The YOLOv5l architecture from the Ultralytics module, which was pre-trained on the Microsoft COCO dataset, was then trained on the insulator dataset for 25 epochs. The weights of the model were updated based on the loss function, which was calculated using the training data.

The YOLO labeling format generates a separate .txt file with the identical name for each image file present in the same directory. These .txt files contain annotations for their corresponding image files, such as the object class, object coordinates, height, and width. The labelling format is $< object - class >< x >< y >< width >< height >$.

Once the model was trained, it was tested on the validation data to assess its performance. The model was evaluated using various performance metrics, including precision, recall, and F1 score. Based on the results, a study was conducted on the pruning method, which is a technique used to reduce the size of the model by removing unnecessary weights. The pruning method works by setting weights that have a small magnitude to zero. This technique reduces the computational cost of the model without affecting its performance. A reasonable pruning threshold was chosen after the study, and the performance metrics were calculated on the test data using the pruned model.

Next, a study was performed regarding the weight clustering method, which is another technique used to reduce the size of the model. The weight clustering method groups the weights of the model into clusters based on their similarities. This reduces the number of unique weights in the model, which in turn reduces the storage space required to store the model. A suitable layer was chosen to apply weight clustering, and the performance metrics were calculated on the test data using the clustered model.

Finally, an analysis was conducted to determine the effect of noise on the weight value. The study aimed to determine the sensitivity of the model to the noise present in the data. The results of the study could be useful in determining the robustness of the model and its ability to handle noisy data.

In summary, the present study used the EPRI dataset to train and evaluate the YOLOv5l architecture from the Ultralytics module. The study employed various techniques, including pruning and weight clustering, to reduce the size of the model and improve its performance. The results of the study could be useful in developing

more efficient and accurate object detection models.

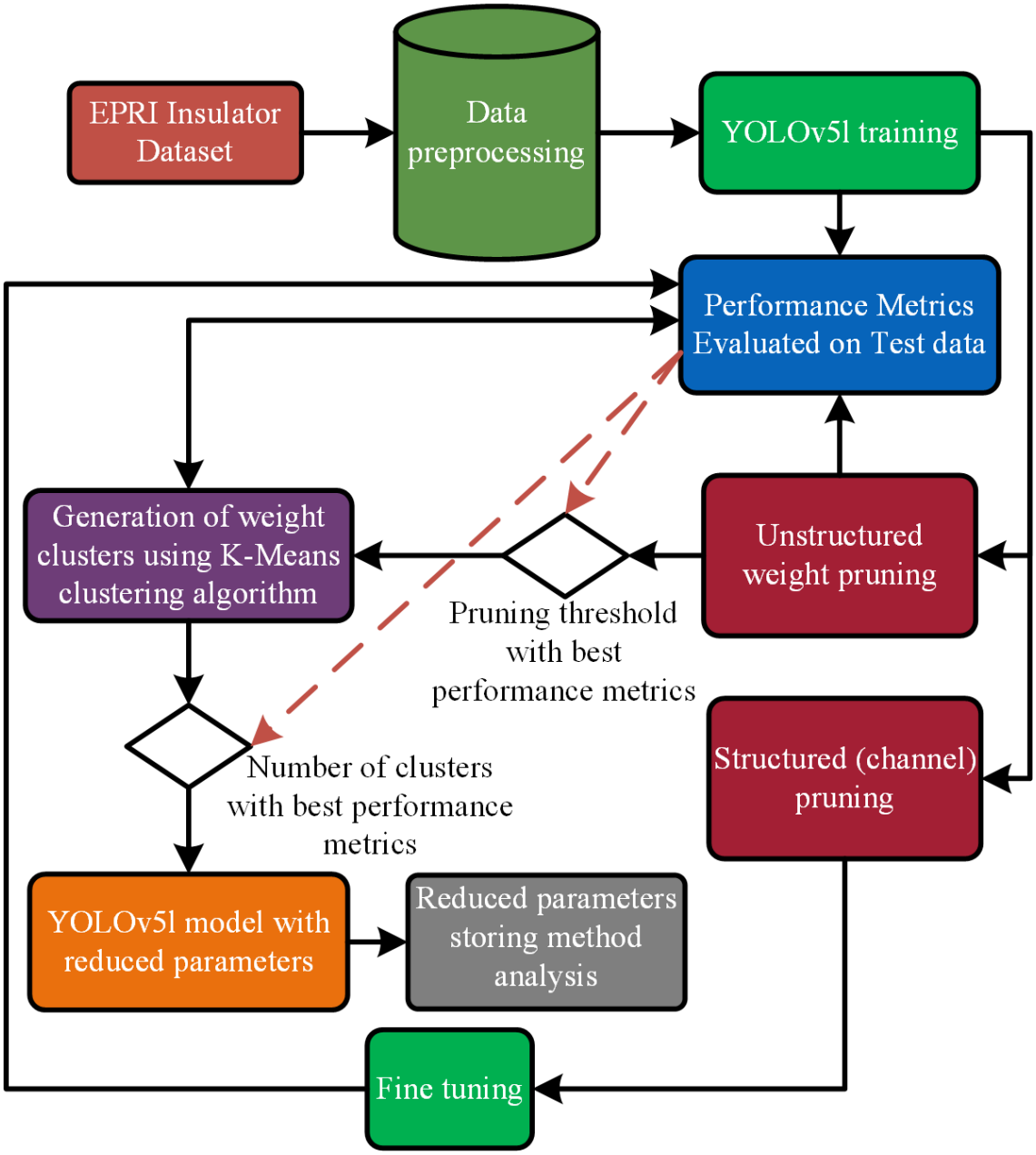


Figure 3.4: Working flow diagram.

Chapter 4

Results and Discussion

4.1 Training Results

The training was performed for 25 epochs using Nvidia GeForce GTX 1080 GPU, as the model was pre-trained on the COCO dataset such a lower epoch was sufficient. This model has the ability to find out various features very efficiently. Various losses with epochs have been shown in fig. 4.1, with epoch all the losses for both training and validation approaching zero thus the model was approaching learning the detection method. From fig. 4.1, for training images per box loss, per object loss, and per class loss are 0.02, 0.018, and 0.0034 as well as for training images per box loss, per object loss, and per class loss are 0.016, 0.0104 and 0.0016. From this, it can be stated that the model has learned and generalized to the given dataset within the trained epoch and further training would overfit the model. Metrics with the epochs are in fig. 4.2, it can be seen that recall and precision for the validation dataset are 0.94 and 0.92 accordingly. This indicates that 92% of faults are correctly detected and 94% of faults that were present in the images were detected correctly. Prediction on some validation images is shown in fig. 4.3, for various faults and oriented images the model performed good predictions.
In table 4.1 and table 4.2 various metrics for training and test images accordingly on the main trained model are given. For insulators and broken class, the model has given a very low false positive. But flashover damage has exhibited very high false positives. One reason for this has been investigated the reflection of sunlight was classified as flashover damage because of the similarity between the appearance in the image. As the precision is 96% and 98% on the train and test data it can be avoided by the proper angle of capturing the image by the UAV. Finally the train dataset has 98% and 99% precision on the trained actual model so it is not overfitted and also it is generalized to the given images.

In fig. 4.4 confusion matrix for test and train image on the actual model is given. As the diagonal values are dominating, the model has a good detection capability. For both test and train broken class has very high detection capability with low background false positives. For insulator true positives decreases from 94% to 85%, in later section it will be seen that this will again surpass the train result after pruning and weight clustered method. For flashover damage, as said earlier, for sun

Table 4.1: Metrics calculated on train images on the base model.

	True Positives	False Positives	Precision	Recall	F1 Score	Average Precision	AP50
Insulator	1506	5.00	1.00	0.95	0.97	0.88	0.99
Broken	1031	9.00	0.99	1.00	0.99	0.86	0.99
Flashover	2284	104.00	0.96	0.99	0.97	0.87	0.99
Mean precision		Mean recall		MAP50		MAP	
0.98		0.99		0.99		0.87	

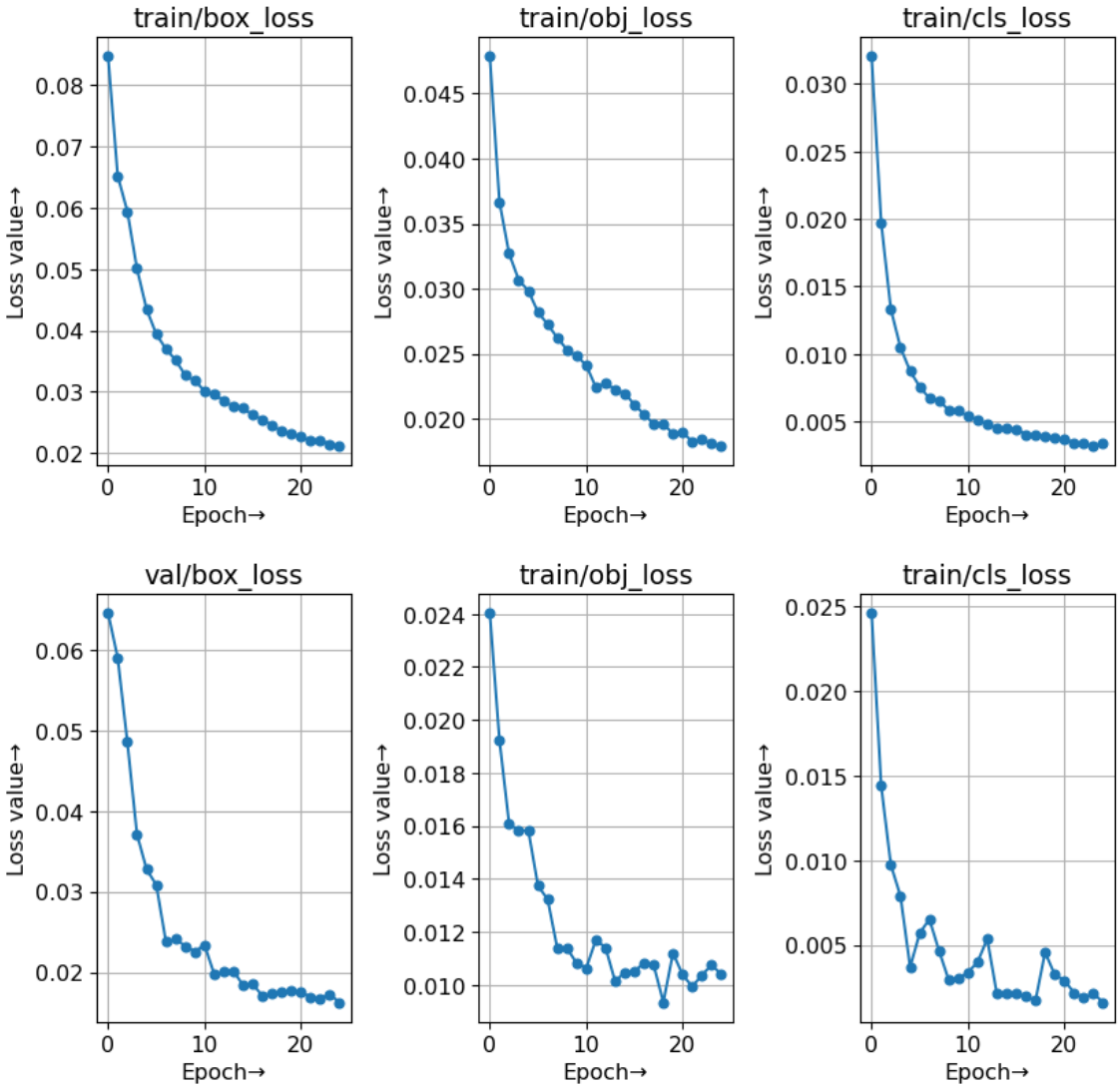


Figure 4.1: Various losses during training vs epoch.

reflection a lots of background false positives are present. Nevertheless, considering a real world inspection problem this model is much better.

4.2 Pruning Method Analysis

The algorithm that has been used to apply unstructured pruning is shown in fig. 4.5. First weight has been sampled from each convolution layer, and after comparing with the threshold value, it set to zero for less than threshold and kept as it is for greater than threshold.

The observed model has a total of 367 layers with total of 46119048 (46 million) parameters. Out of which 46045952 parameters are in convolution layers which is 99.84%. So pruning only the convolution layers is sufficient for the size reduction. From fig. 4.13(a), most of the weights in a convolution layer are zero-centered, and pruning them all to zero value can have a significant effect on the size reduction as sparsity increases after pruning, for example after prune out all the value less than 0.008 the weight distribution is shown in fig. 4.13(b), in fig. 4.13(a), weight distribution is of the main model and in fig. 4.13(b) after pruning, the spike represents

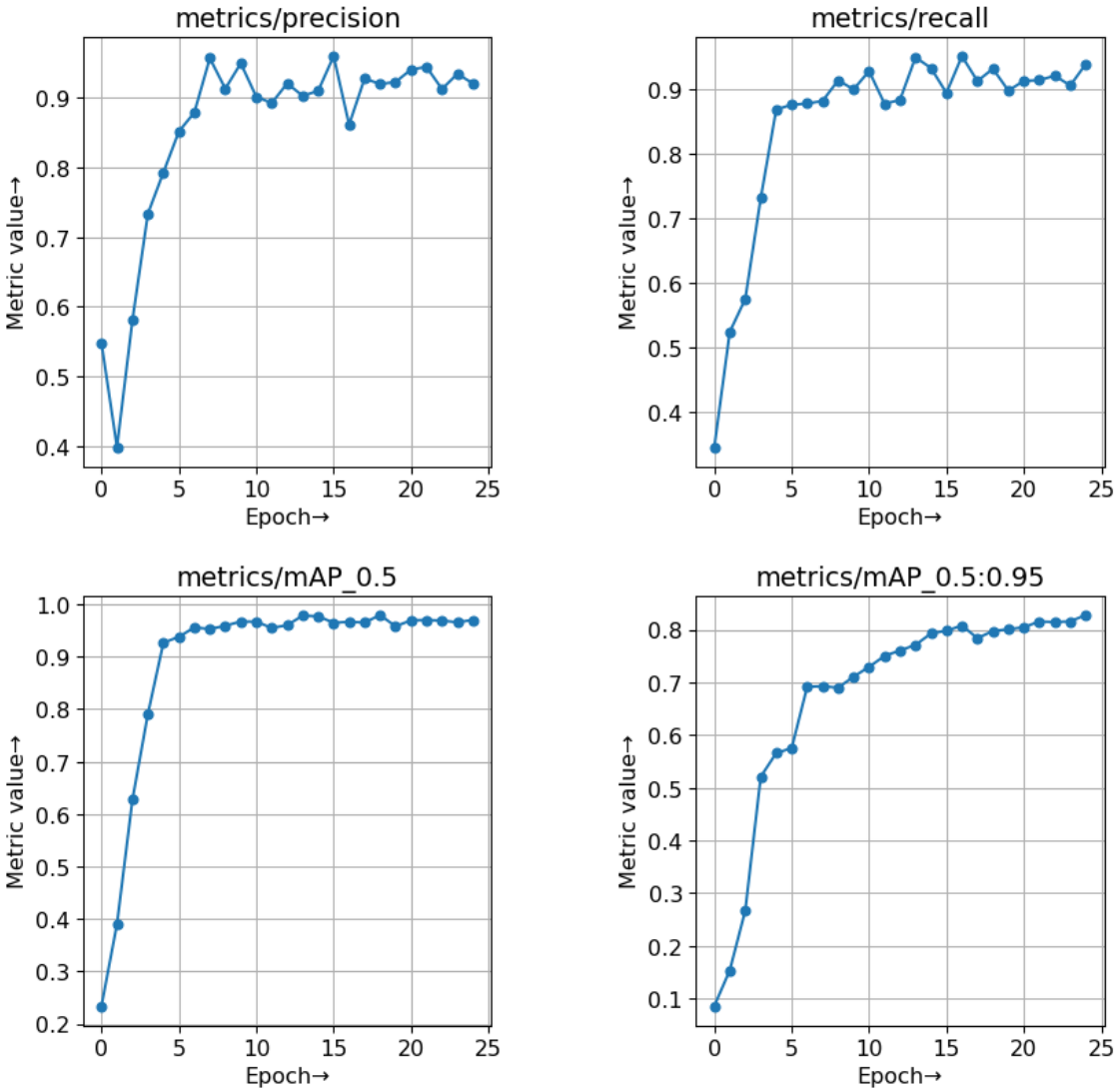


Figure 4.2: Various metrics during training for validation dataset vs epoch.

the zeros out value, as 45.14% weights has been pruned out, so the distribution of other weight values has became smaller.

For various pruning threshold, total weights less than the threshold value and its percentage is given in table 4.3, and corresponding plots are in fig. 4.6. Upto threshold 0.01, 52.19% parameters can be pruned out, which is a large number, it means only around 48% weights are participating in the detection of the faults. So reduction in memory and calculation almost half.

From fig. 4.7, as can be seen, for values up to 0.01 there is almost no reduction in average precision. From fig. 4.7 to fig. 4.12, it is clear that other metrics are also constant up to this threshold value. Therefore, it can be said that using only 48% weights it is possible to preserve the performance of the base model. Again for detecting faults only, this threshold can be increased to 0.02, for which 73.79% weights can be pruned out, and only 27% parameters will be required for performing detection. So for insulator detection, it is highly sensitive to pruning out weights less than 0.01. From fig. 4.12, for value, less than 0.01 metrics are decreasing drastically, this is due to the effect of the high sensitivity of the insulator class, as can be seen from the previous results.

Table 4.2: Metrics calculated on test images on the base model.

	True Positives	False Positives	Precision	Recall	F1 Score	Average Precision	AP50
Insulator	142	0	1.00	0.96	0.98	0.84	0.98
Broken	77	1	0.99	0.96	0.98	0.84	0.99
Flashover	195	3	0.98	0.83	0.90	0.77	0.91
Mean precision		Mean recall		MAP50		MAP	
0.99		0.96		0.96		0.82	

Table 4.3: Table shows weight values less than the threshold in all conv layers.

Conv Layer Number	Total number of weights less than Thereshold	Pruning threshold	Percentage (%)
1	3315159	0.001	7.20
2	6482869	0.002	14.08
3	9433126	0.003	20.49
4	12141679	0.004	26.37
5	14614691	0.005	31.74
6	16869313	0.006	36.64
7	18918227	0.007	41.09
8	20783904	0.008	45.14
9	22482362	0.009	48.83
10	24029539	0.01	52.19
11	33978355	0.02	73.79
12	38774171	0.03	84.21
13	41360018	0.04	89.82
14	42848296	0.05	93.06
15	43766646	0.06	95.05
16	44365891	0.07	96.35
17	44777454	0.08	97.25
18	45069415	0.09	97.88
19	45280559	0.1	98.34
20	45921366	0.2	99.73
21	46006997	0.3	99.92
22	46029103	0.4	99.96
23	46036759	0.5	99.98
24	46040007	0.6	99.99
25	46041607	0.7	99.99
26	46042549	0.8	99.99

Table 4.4: Data represents total weights number before and after pruning in the convolution layers.

Total parameters	Total weights in the convolutional layers	Total weights pruned out from the convolutional layers after pruning with 0.008 threshold
46119048	46045952	20783904
Pruning percentage		45.14%

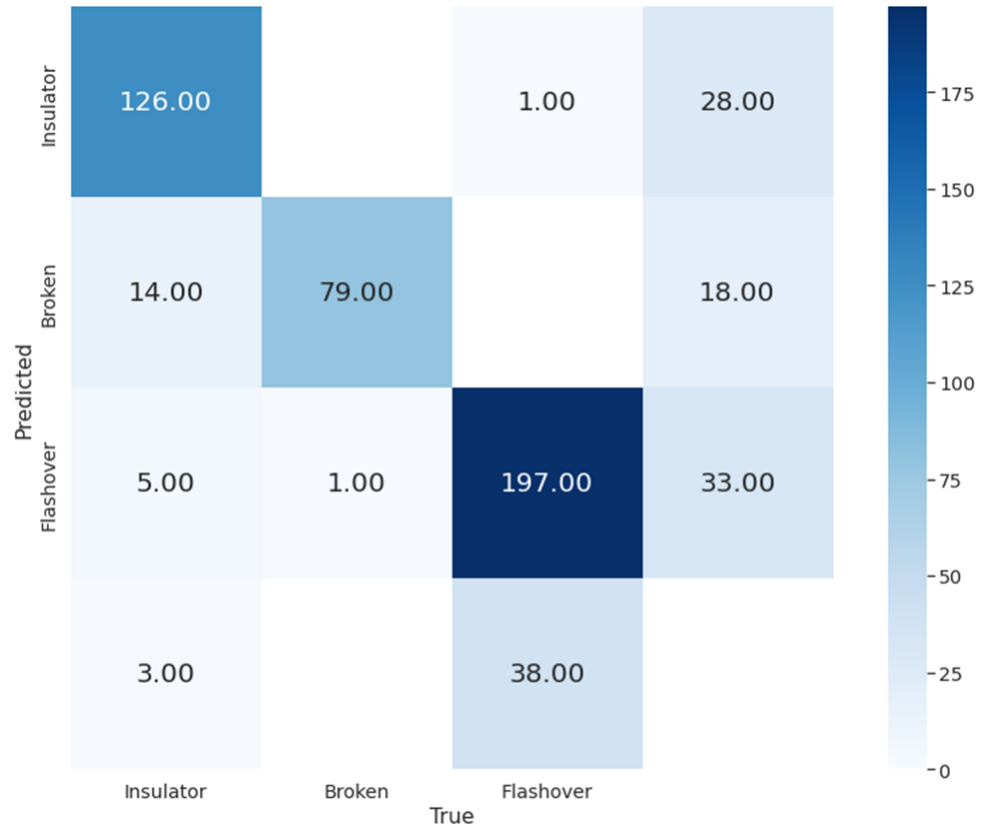
Figure 4.3: Prediction on some insulator images.

Considering pruning with weight clustering pruning threshold 0.008 has been selected as will be seen in section 4.3, weight distribution of a conv layer for this pruning threshold is given in fig. 4.13, and all other layers exhibits the same visual distribution. For this pruning threshold it was possible to prune out around 45.14% weights. In table 4.5 and table 4.6, metrics for train and test images has been attached. Compared to table 4.1 and table 4.2, there is almost no significant change in any metrics, so performance of the original metrics is preserved.
Confusion matrix for test and train dataset is given in fig. 4.14, for insulator class, there is a jump from 85% true positives to 97% true positives and for others there is no significance change. From this it can also be verified that, performance remains same after pruning as it was without pruning.

4.3 Weight Clustering Method Analysis

The weight clustering algorithm used is shown fig. 4.15. Unlike pruning, weight clustering can't be performed on all the layer as will be discussed from fig. 4.17. So

(a) Confusion matrix on train data using actual model



(b) Confusion matrix on test data using actual model

Figure 4.4: Confusion matrix of train and test images on trained model

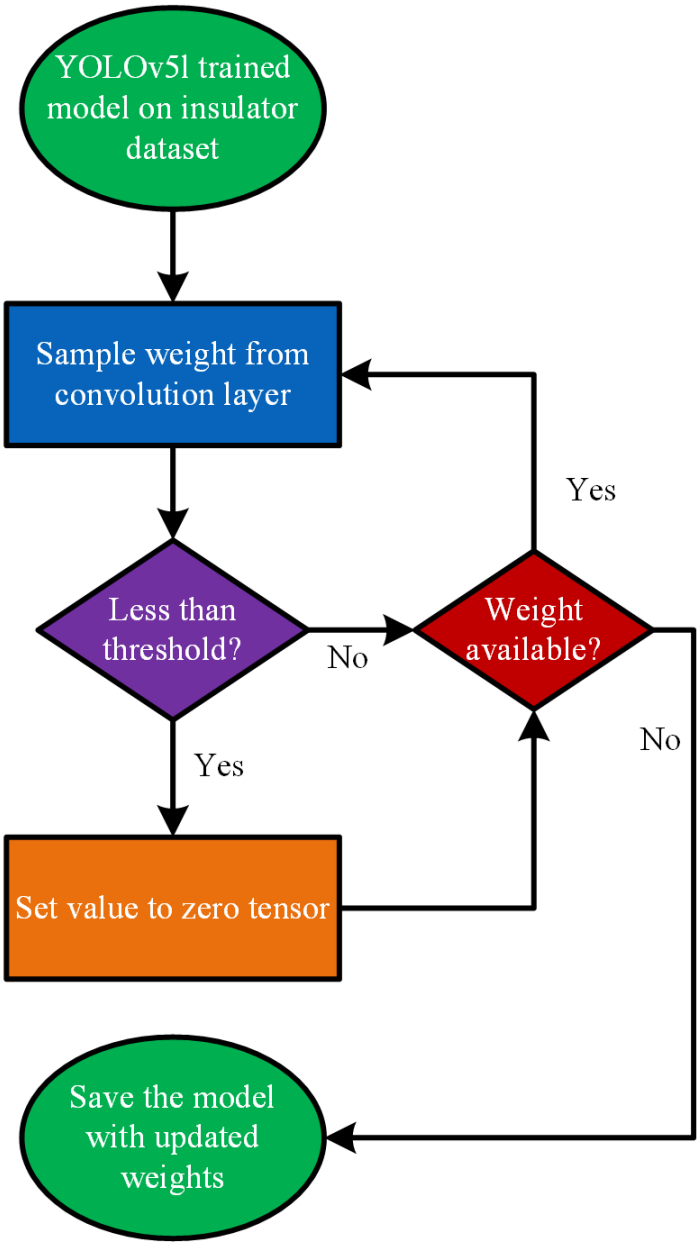
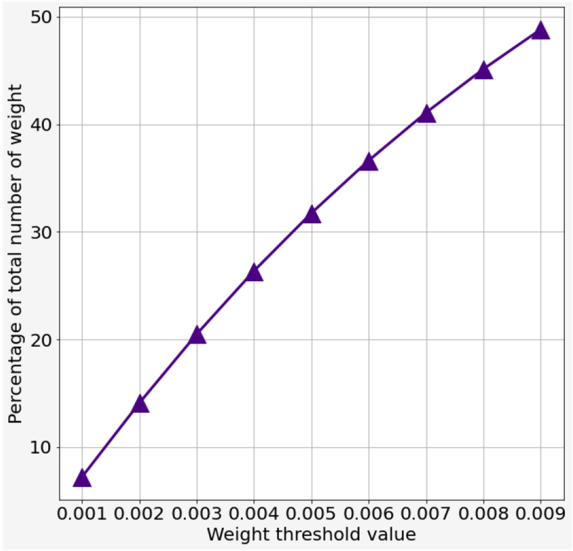


Figure 4.5: Algorithm used to apply unstructured pruning to the model.

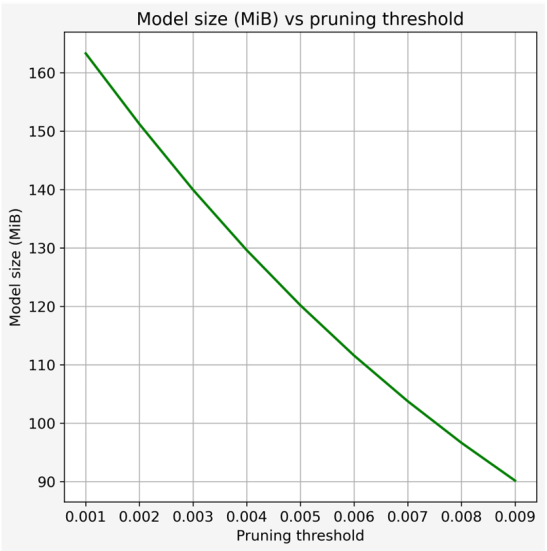
it is applied to only first few layers. First weights of particular layer is loaded, then only non-zero weights are used to apply k-means clustering with specified cluster number. Then from the generated cluster centers, every weights is replaced with their corresponding cluster center.

The distribution of weights before and after clustering with the number of cluster 50 is shown in fig. 4.16. There is a discrete effect in the distribution after the clustering due to the cluster centers of the weights.

The breakdown of the model structure with respect to the pruning threshold is illustrated in fig. 4.17, where the layer number is plotted against the corresponding pruning threshold. The results reveal that weight clustering can be applied up to the first 30 convolution layers of the model with a pruning ratio of 0.008. Consequently, clustering on the first 30 convolution layers (comprising 3349248 weights, accounting for 16.11% of the total effective weights) with a cluster size of 50 for each layer was performed. Prior to clustering, the total number of unique parameters was



(a) Thresholded weights number vs prune threshold (0.001 to 0.009), 45.14% weights are less than 0.008 so huge number of weights can be pruned out.



(b) For 0.008 pruning threshold, model size is 96.65MB compared to original size of 175.93MB, 1.8times reduction in size.

Figure 4.6: Total parameters, and model size comparison with pruning threshold.

Figure 4.7: Average precision vs pruning threshold

2399694, which was significantly reduced to 1550 unique values after clustering was applied in these initial 30 layers, resulting in a remarkable 1548% reduction in unique parameters. This reduction enables the storage of a large number of weights with high precision in small memory, which was previously unattainable. Specifically, for

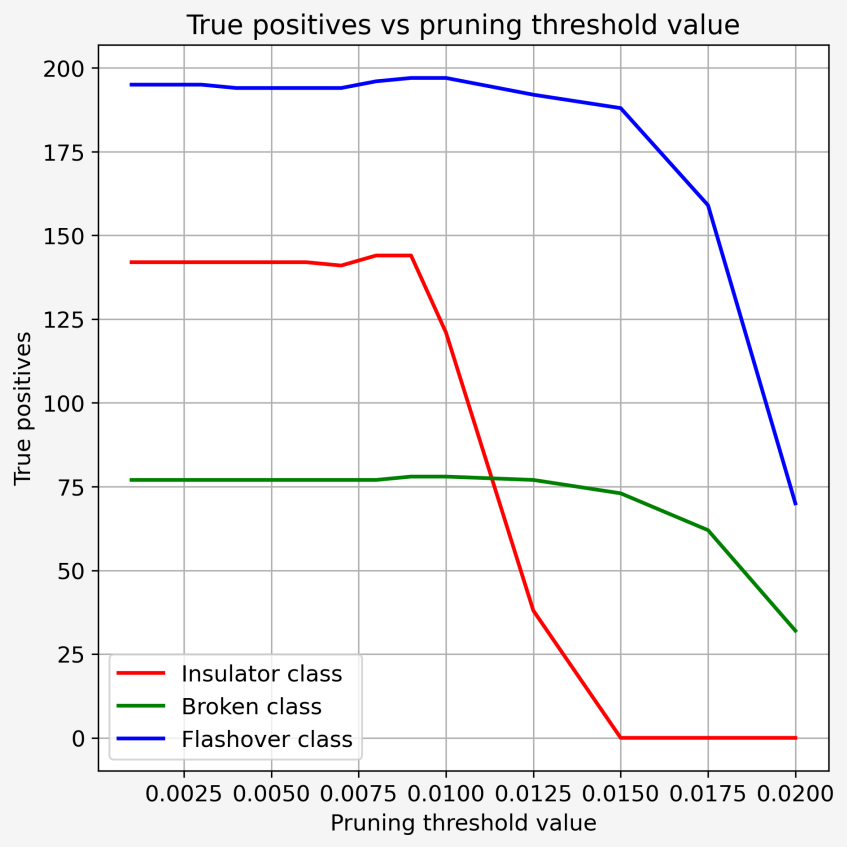


Figure 4.8: True positives vs pruning threshold

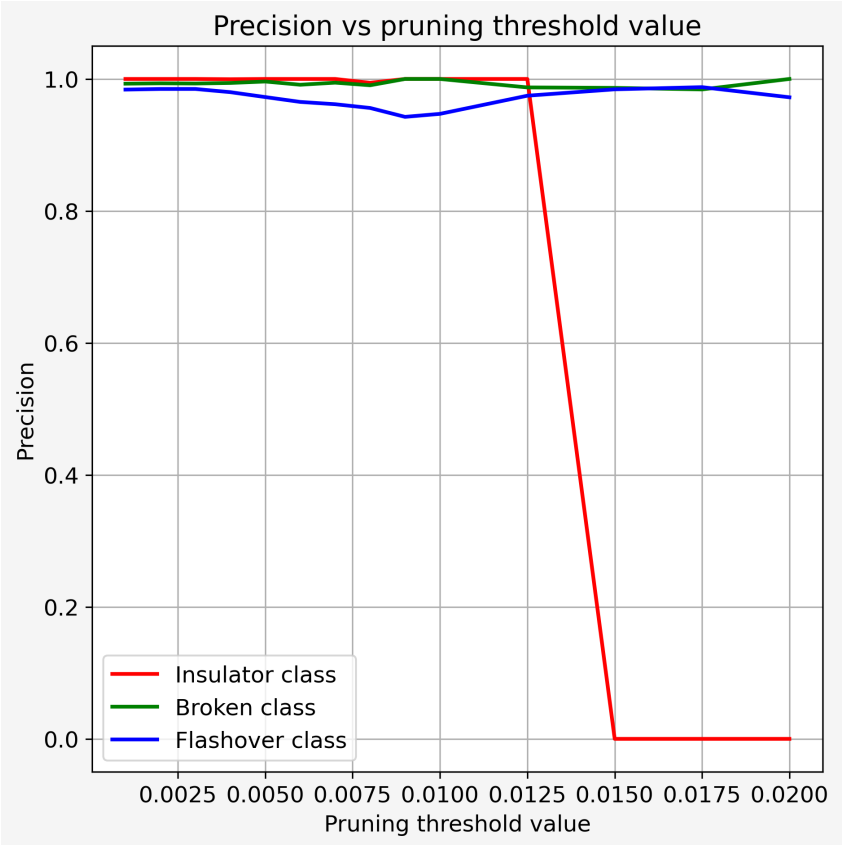


Figure 4.9: Pecision vs pruning threshold

Figure 4.10: Recall vs pruning threshold

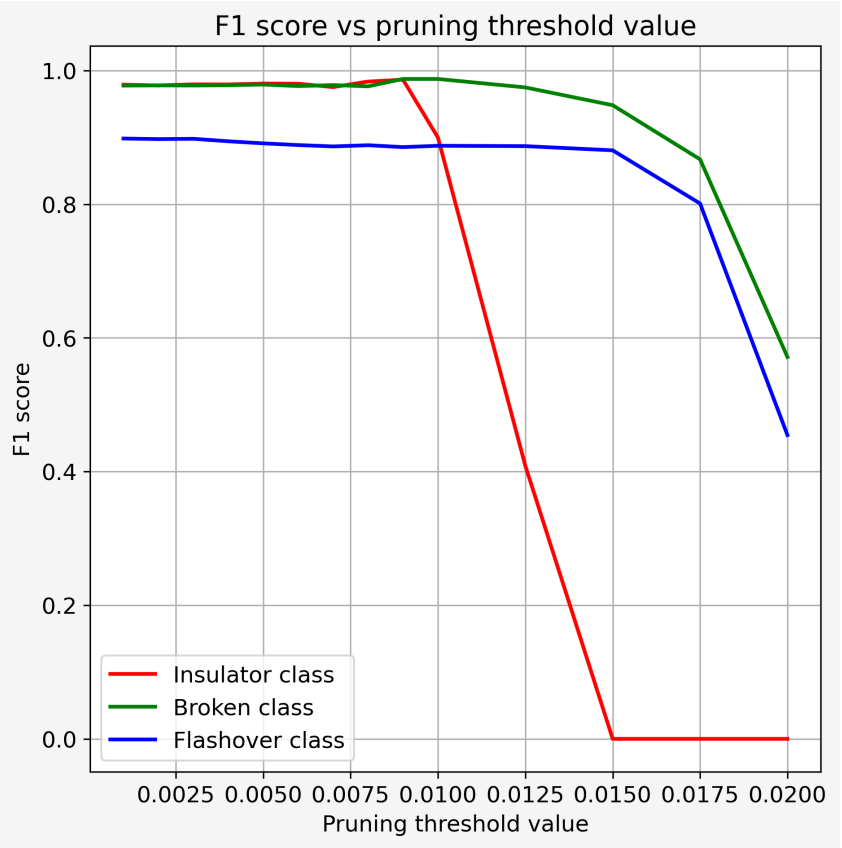


Figure 4.11: F1 score vs pruning threshold

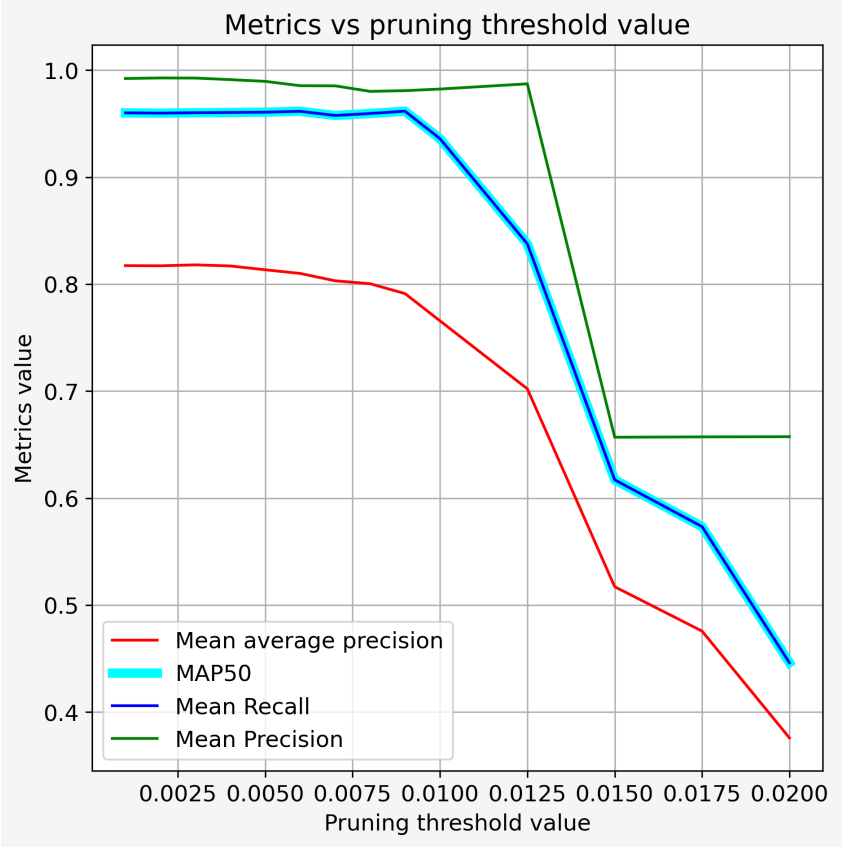
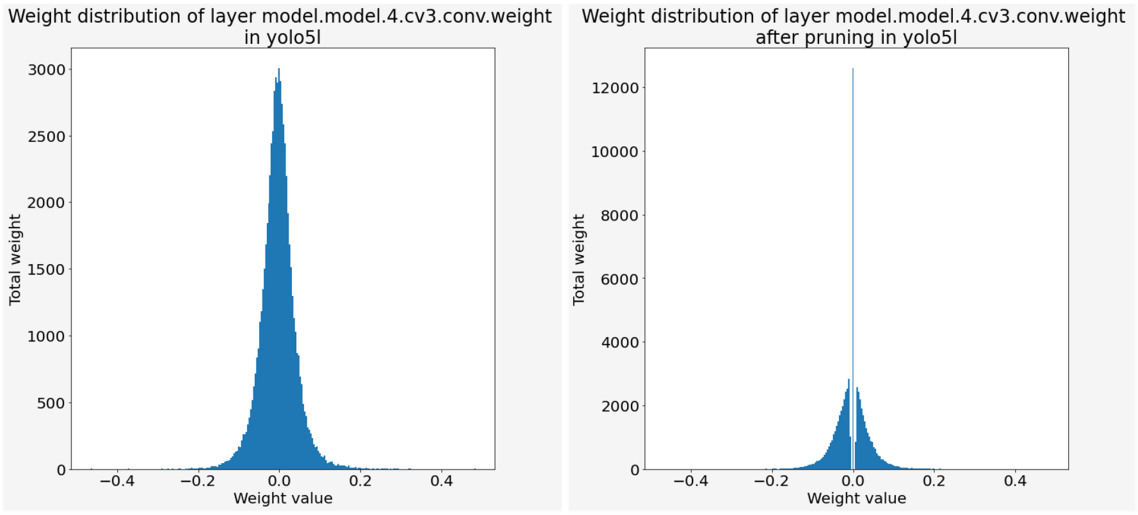


Figure 4.12: Metrics vs pruning threshold



(a) Weight distribution of a layer in actual model, with total number of weights 46045952 in convolution layers

(b) Weight distribution of a layer after pruning (0.008), with total 20783904 (45.14%) weights pruned out.

Figure 4.13: Weight distribution in indicated convolution layer after pruning with threshold 0.008.

each weight of the cluster number 50, an 8-bit integer number can be used to store the cluster center index, and a table of cluster centers for each layer can be used to recover the weights at a later time. This is a considerable improvement over the previous method where each weight was stored using a 32-bit float number, which required significantly more memory.

Table 4.5: Metrics calculated on train images on the pruned (0.008) model.

	True Positives	False Positives	Precision	Recall	F1 Score	Average Precision	AP50
Insulator	1509	16	0.99	0.95	0.97	0.79	0.99
Broken	1031	30	0.97	1.00	0.98	0.86	0.99
Flashover	2294	216	0.91	0.99	0.95	0.86	0.99
Mean precision		Mean recall		MAP50		MAP	
0.96		0.99		0.99		0.84	

Table 4.6: Metrics calculated on test images on the pruned (0.008) model.

	True Positives	False Positives	Precision	Recall	F1 Score	Average Precision	AP50
Insulator	144	1	0.99	0.97	0.98	0.79	0.99
Broken	77	1	0.99	0.96	0.98	0.84	0.98
Flashover	196	9	0.96	0.83	0.89	0.77	0.91
Mean precision		Mean recall		MAP50		MAP	
0.98		0.96		0.96		0.80	

4.4 Final Detection Model Result

Results are given based on clustered applied to first **30** layers with pruning threshold value **0.008**, in fig. 4.19, the weight distribution of this combination is given and both pruning and weight clustering effects are presented in this distribution. The final model size is 87.07 MB, which can be calculated as follow,

- Non-zero weights size: 21985896 x 32-bit
- 50 clusters indices size: 3349548 x 8-bit
- 1550 unique weight size: 1150 x 32-bit
- Finally total size = 87.07MB

The evaluation of the detection model's performance is presented in table 4.7 and table 4.8, where various performance metrics are enlisted for comparison with the performance of the actual and pruned models, as illustrated in table 4.1, table 4.2, table 4.5, and table 4.6. Upon careful analysis of the presented data, it is noteworthy that there is no significant decline in the performance of the detection model when compared to its counterparts. A key finding from the evaluation is that the mean precision (mp) on the test data for the pruned and weight clustered model is equivalent to the base model, both achieving an impressive precision rate of 99%. It is noteworthy that despite pruning out 45.14% of the weights and a substantial reduction of 1548% in the unique value of the effective weights for the first 30 layers, the overall performance of the reduced weight model remains comparable to the base model. This observation is indicative of the effectiveness and efficiency of the pruned and weight clustered model, demonstrating its potential to reduce computational resources while maintaining optimal performance.

The presented fig. 4.21 in the following analysis is the confusion matrix that depicts the evaluation results of the pruned and weight clustered model. Upon observing the matrix, it becomes apparent that no significant alteration has been

Table 4.7: Metrics calculated on train images on the pruned and weight clustered model.

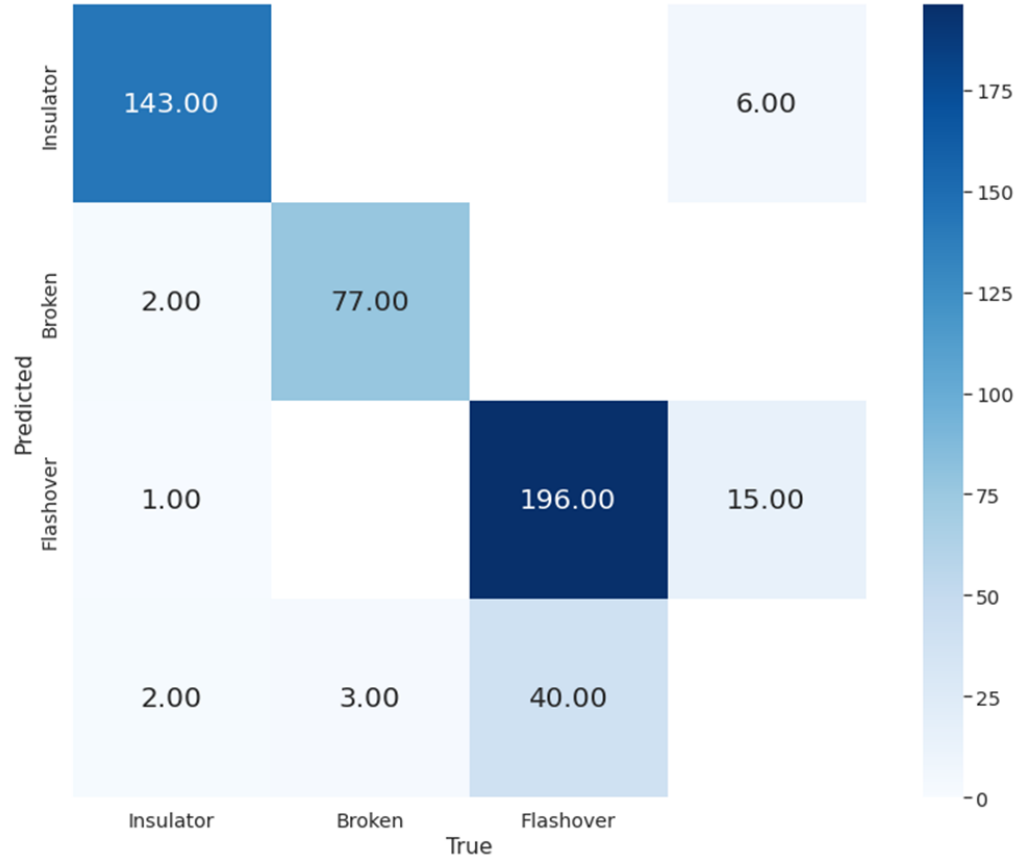
	True Positives	False Positives	Precision	Recall	F1 Score	Average Precision	AP50
Insulator	1504	14	0.99	0.95	0.97	0.79	0.99
Broken	1031	27	0.97	1.00	0.98	0.86	0.99
Flashover	2286	182	0.93	0.99	0.96	0.86	0.99
Mean precision		Mean recall		MAP50		MAP	
0.96		0.99		0.99		0.84	

Table 4.8: Metrics calculated on test images on the pruned and weight clustered model.

	True Positives	False Positives	Precision	Recall	F1 Score	Average Precision	AP50
Insulator	142	1	0.99	0.96	0.98	0.78	0.99
Broken	75	1	0.99	0.94	0.96	0.83	0.97
Flashover	192	5	0.98	0.81	0.89	0.76	0.90
Mean precision		Mean recall		MAP50		MAP	
0.99		0.95		0.95		0.79	

observed in comparison to the performance of the base model. However, upon further analysis of the test data, it is noteworthy that there has been a noteworthy improvement observed in the true positive rate for the insulator class. The test data has shown a remarkable increase from the previous 85% to an impressive 97%.

(a) Confusion matrix on train data using using pruned model (0.008)



(b) Confusion matrix on test data using using pruned model (0.008)

Figure 4.14: Confusion matrix of test and train images on prune model.

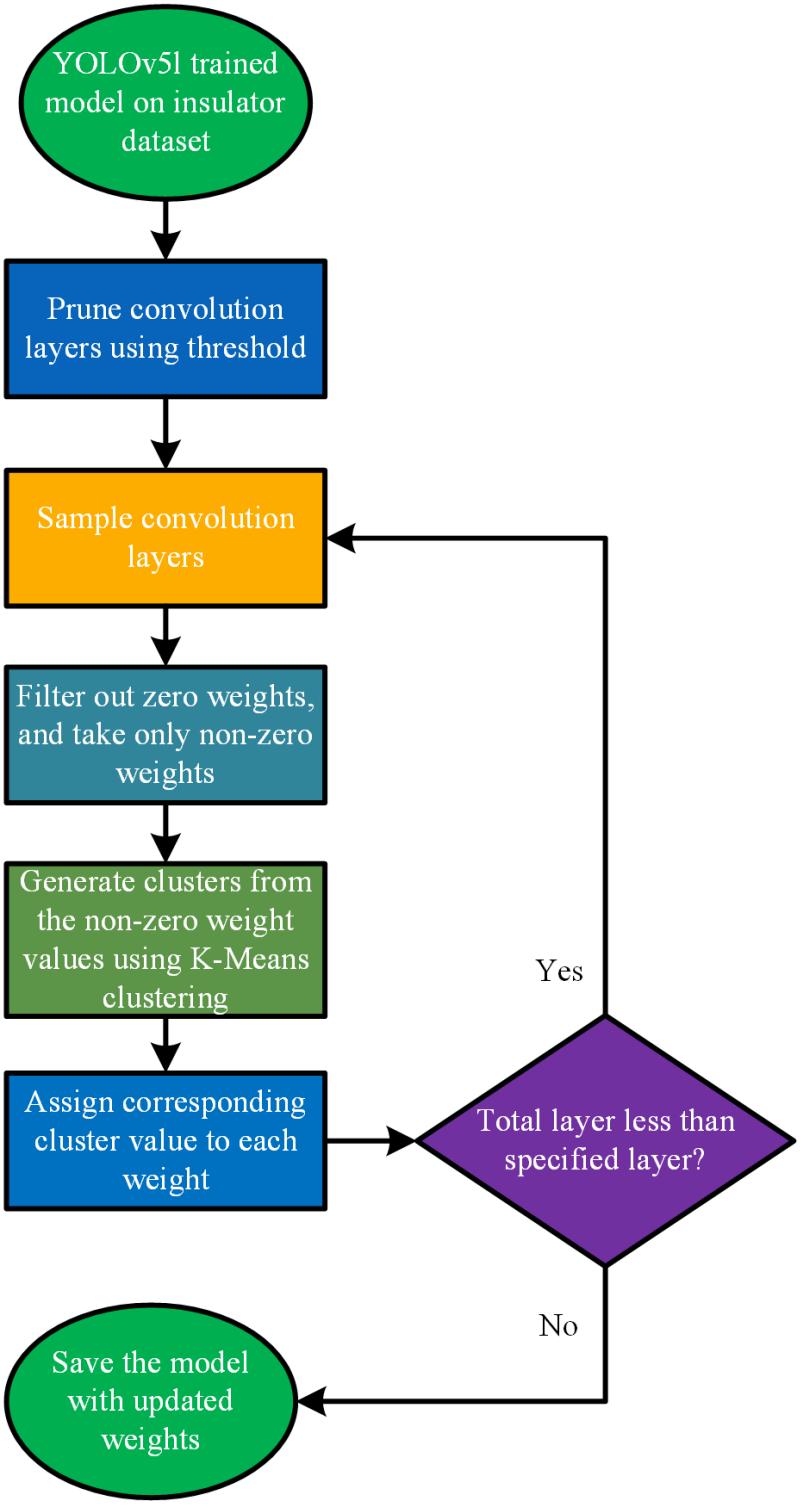
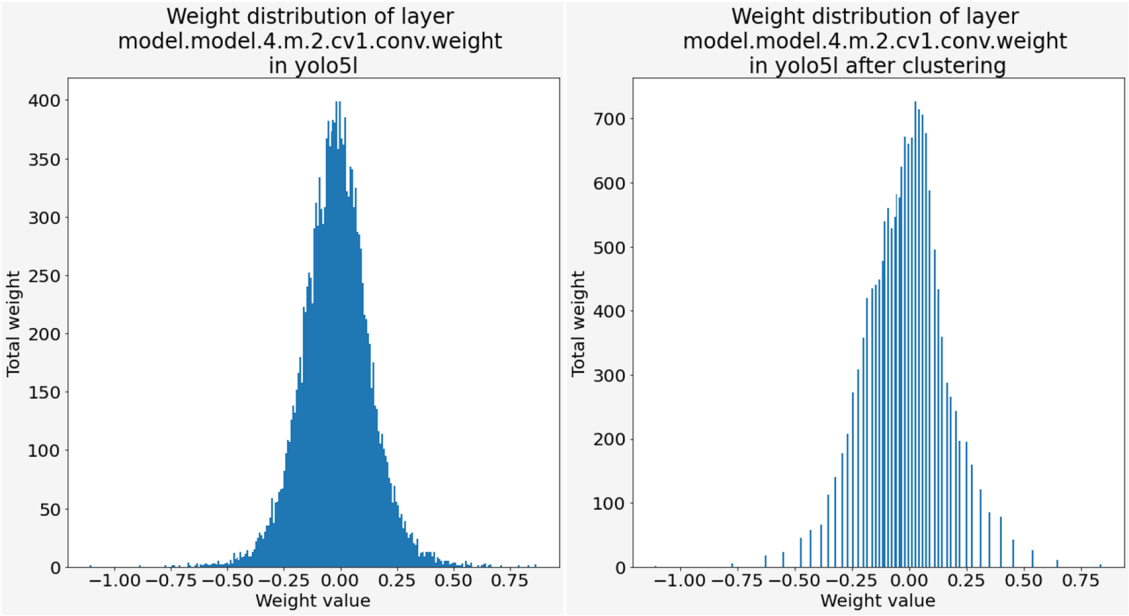


Figure 4.15: Algorithm used to apply weight clustering method.



(a) Weight distribution in indicated convolution layer before clustering.

(b) Weight distribution in indicated convolution layer after clustering (50 clusters).

Figure 4.16: Weight distribution in indicated convolution with weight clustering.

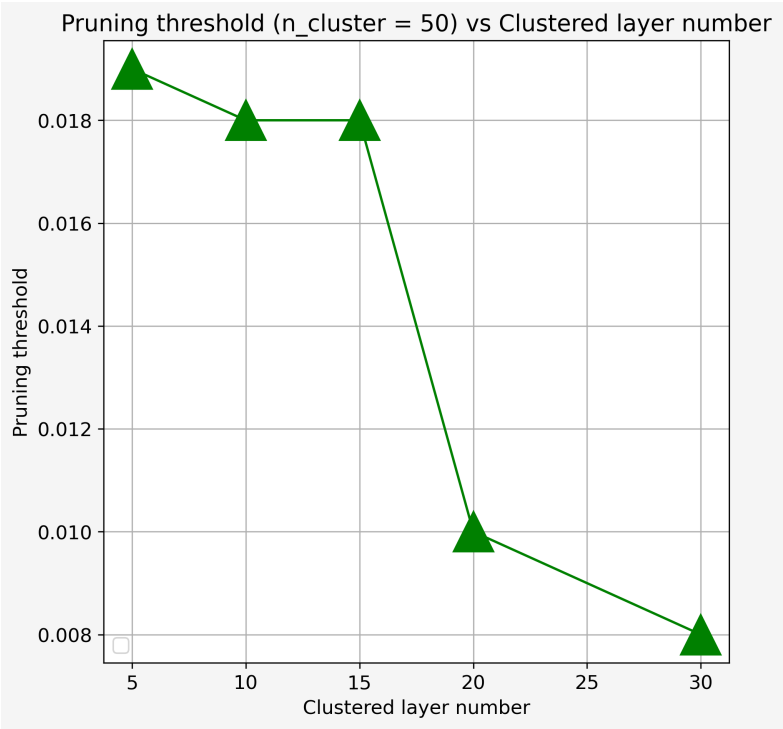
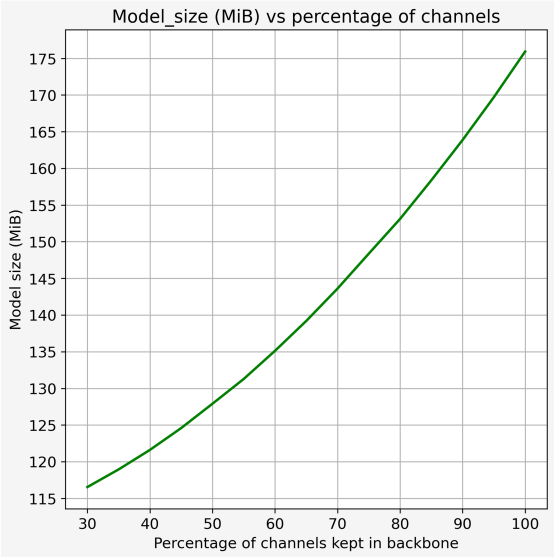
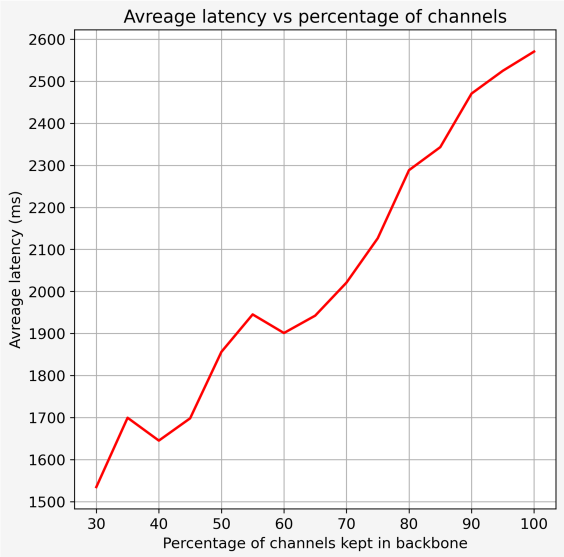


Figure 4.17: Model breakdown limit with various clustered layers and corresponding pruning threshold

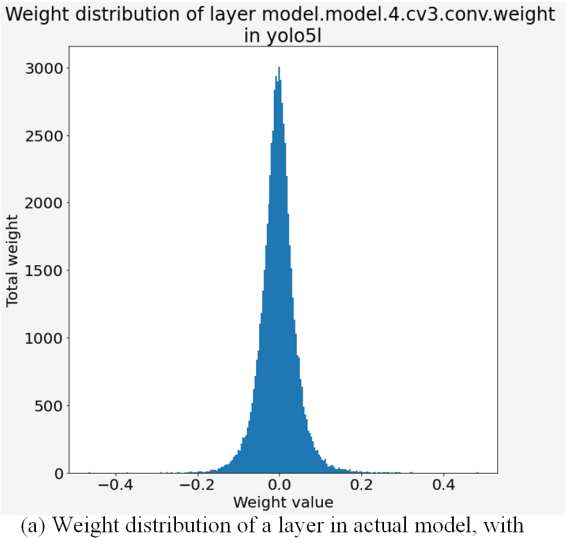


(a) Reduction in model size with less channel kept in the backbone (1.4x smaller model, reduction from 175.9 MB to 128.9 MB).

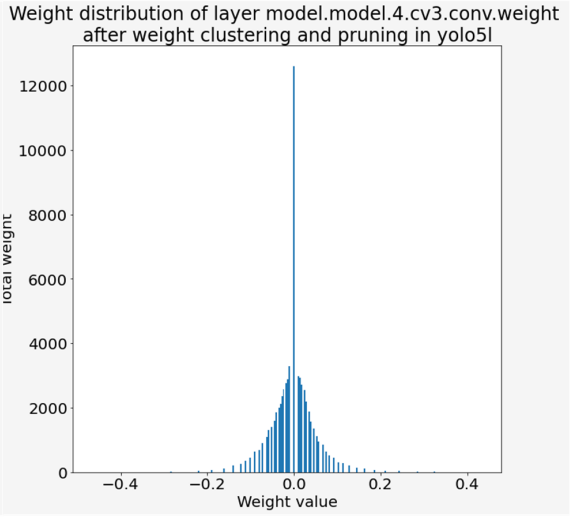


(b) Latency goes down to 1.86 second from 2.57 second, 1.38x faster for 50% channel pruning.

Figure 4.18: Comparison of latency and model size with the change of convolutional channel kept in the backbone of the model.



(a) Weight distribution of a layer in actual model, with size 175.93MB.



(b) Weight distribution of a layer after pruning (0.008) and weight clustering with final model size 87.07MB.

Figure 4.19: Weight distribution in indicated convolution layer after pruning with threshold 0.008 and weight clustering up to 30 layers with the number of cluster size 50 at each layer.

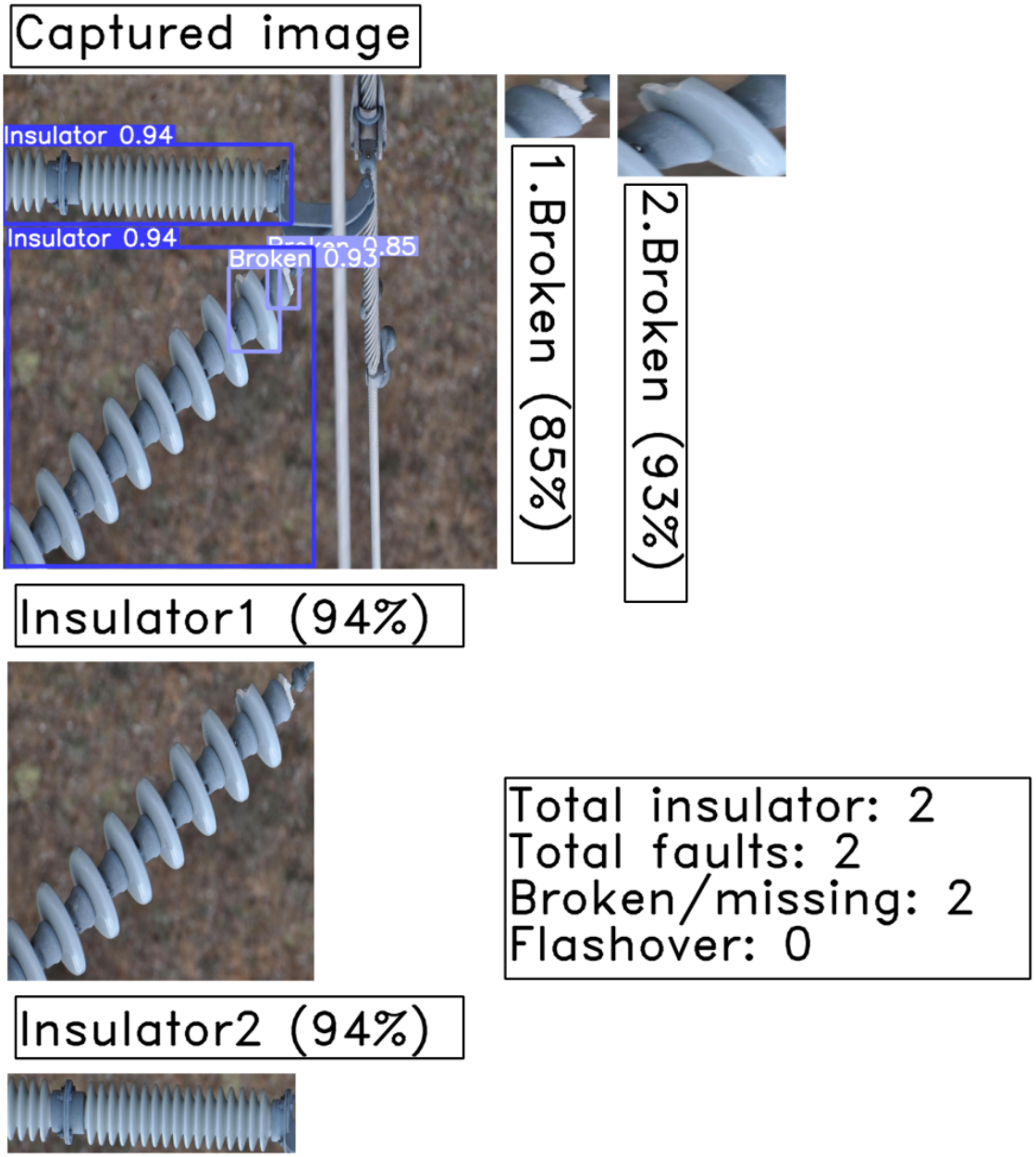
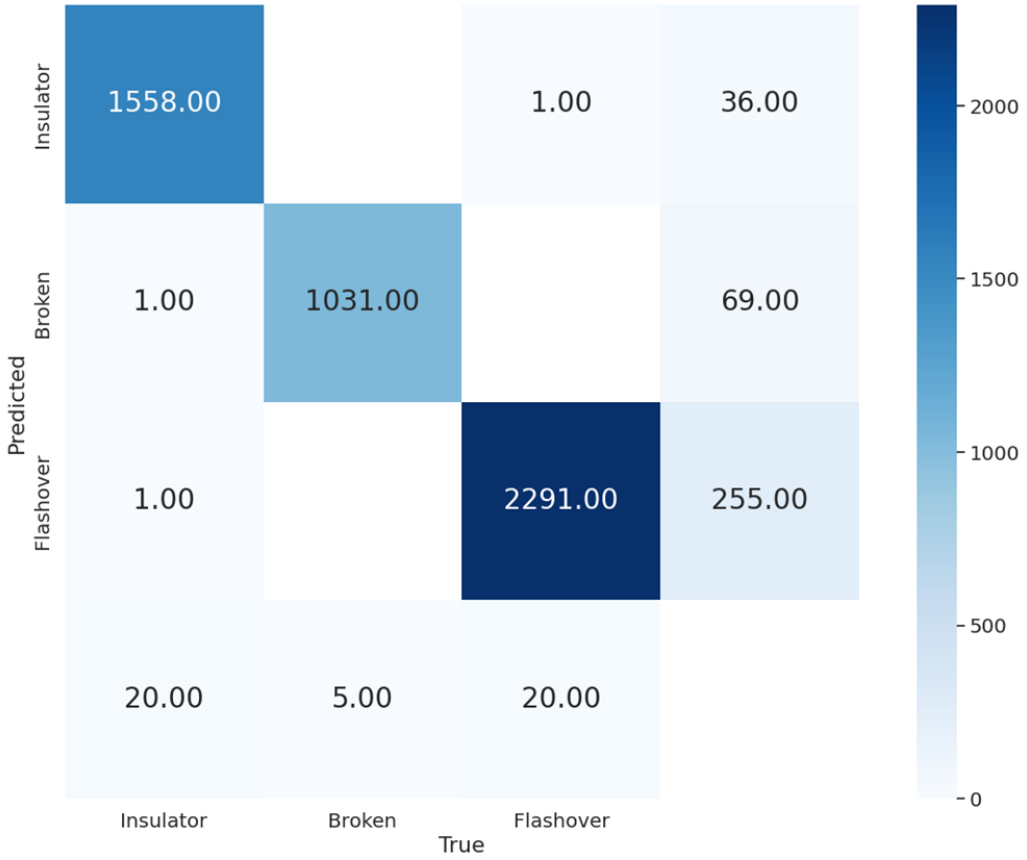
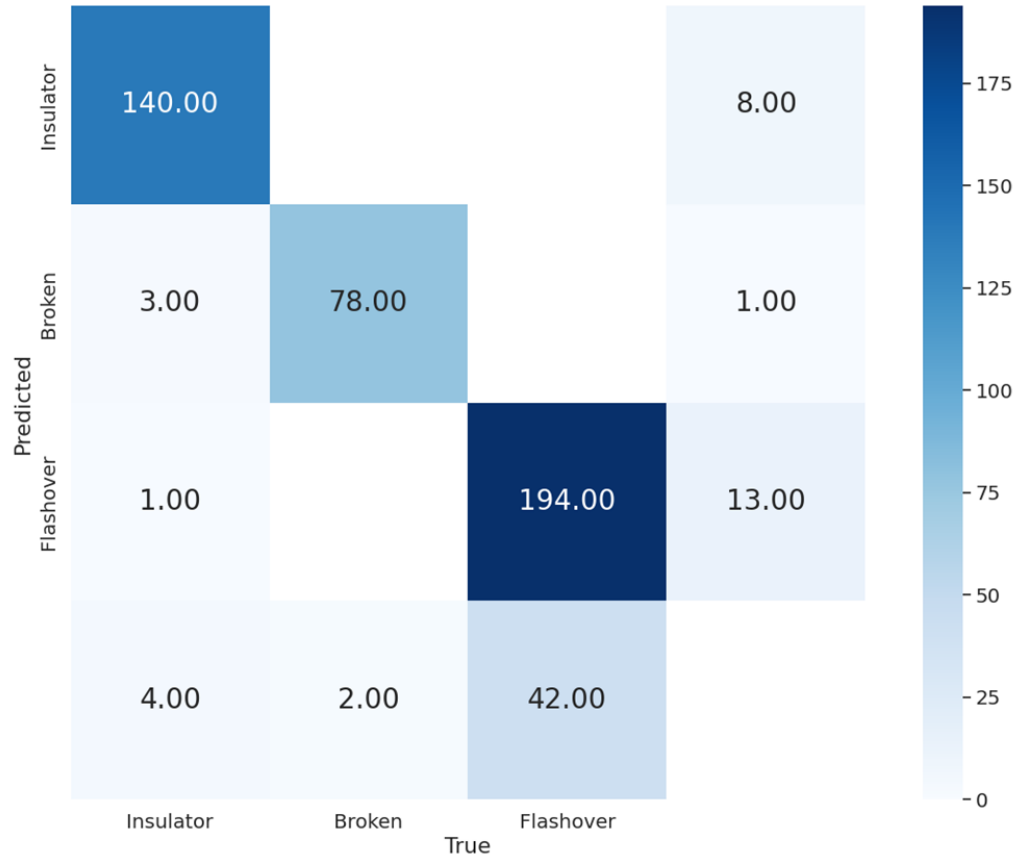


Figure 4.20: Figure shows generated report from the designed API for report generation, with individual faults and insulators separated from the original image, and a summary of the detection.



(a) Confusion matrix on train data using using pruned and weight clustered method



(b) Confusion matrix on test data using using pruned and weight clustered method

Figure 4.21: Confusion matrix of test and train images on the final model.

Chapter 5

Conclusion

5.1 Conclusion

The successful fulfillment of all objectives outlined in section 1.5 is clearly evidenced by the discussions and results presented above. Most notably, a final model has been achieved that exhibits a remarkable 99% precision on the test data and 96% precision on the train data. This outcome has been made possible by weight pruning, which resulted in a notable reduction of 45.14%. Moreover, by further clustering, a remarkable reduction of 1548% in unique weights was achieved, with no observed performance degradation compared to the base model.
The result in summary is presented below,

- 1.8x reduction in model size achieved using 0.008 pruning threshold.
- 2x reduction in model size with unstructured pruning and weight clustering.
- Normally no latency reduction in unstructured pruning.
- 1.38x faster and 1.4x smaller model with channel pruning.
- Structured pruning performed better than unstructured pruning.
- Base model performance preserved with size reduction.
- API for inspection has been successfully implemented.

Overall, this model represents a significant contribution to the field of insulator fault inspection and can be employed to enhance the reliability and efficiency of power transmission systems. The identification of potential areas for future research in section 5.2 underscores the need for continued development and innovation in this field to address evolving challenges and emerging applications. In summary, the successful achievement of all objectives outlined in this study represents a notable milestone and underscores the potential of machine learning and deep learning techniques in improving the reliability and efficiency of critical infrastructure systems.

5.2 Direction For Future Research

Future research scope can be listed as:

1. Developing an autonomous UAV system to carry out the inspection on the insulators using the prepared detection model.
2. Study the importance of various layers and prune out less important layers.
3. Finding the optimum number of clusters with the effect of pruning.
4. Developing an autonomous algorithm to find out the above-mentioned parameters.
5. Developing FPGA and software implementation to carry out operations efficiently using the pruned and clustered weights.

Appendix A

List of Abbreviation

1. **YOLO** You Only Look Once
2. **SSD** Single Shot Detector
3. **R-CNN** Regions with Convolutional Neural Networks
4. **EPRI** Electric Power Research Institute
5. **COCO** Common Objects in Context
6. **CPLID** Chinese Power Line Insulator Dataset
7. **VGG** Visual Geometry Group
8. **GAN** Generative Adversarial Networks
9. **UAV** Unmanned Aerial Vehicle
10. **AP** Average Precision
11. **CNN** Convolutional Neural Network
12. **ROI** Region of Interest
13. **IoU** Intersection over Union
14. **TP** True Positives
15. **FP** False Postives
16. **JSON** JavaScript Object Notation
17. **FPGA** Field Programmable Gate Arrays

References

[1] "Electrical insulator testing — cause of insulator failure — electrical4u." (Oct. 27, 2020), [Online]. Available: https://www.electrical4u.com/electrical-insulator-testing-cause-of-insulator-failure/ (visited on 02/17/2023).

[2] S. Han, H. Mao, and W. J. Dally, *Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding*, Feb. 15, 2016. arXiv: 1510.00149[cs]. [Online]. Available: http://arxiv.org/abs/1510.00149 (visited on 02/17/2023).

[3] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA: IEEE, Jun. 2016, pp. 779–788, ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.91. [Online]. Available: http://ieeexplore.ieee.org/document/7780460/ (visited on 02/17/2023).

[4] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*, Jul. 6, 2022. arXiv: 2207.02696[cs]. [Online]. Available: http://arxiv.org/abs/2207.02696 (visited on 02/17/2023).

[5] J. Redmon and A. Farhadi, *YOLOv3: An incremental improvement*, Apr. 8, 2018. arXiv: 1804.02767[cs]. [Online]. Available: http://arxiv.org/abs/1804.02767 (visited on 02/17/2023).

[6] W. Liu, D. Anguelov, D. Erhan, *et al.*, "SSD: Single shot MultiBox detector," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., vol. 9905, Series Title: Lecture Notes in Computer Science, Cham: Springer International Publishing, 2016, pp. 21–37, ISBN: 978-3-319-46447-3 978-3-319-46448-0. DOI: 10.1007/978-3-319-46448-0_2. [Online]. Available: http://link.springer.com/10.1007/978-3-319-46448-0_2 (visited on 02/17/2023).

[7] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-CNN: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, Jun. 1, 2017, ISSN: 0162-8828, 2160-9292. DOI: 10.1109/TPAMI.2016.2577031. [Online]. Available: http://ieeexplore.ieee.org/document/7485869/ (visited on 02/17/2023).

[8] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI: IEEE, Jul. 2017, pp. 6517–6525, ISBN: 978-1-5386-0457-1. DOI: 10.1109/CVPR.2017.690. [Online]. Available: http://ieeexplore.ieee.org/document/8100173/ (visited on 02/17/2023).

[9] C. Wu, X. Ma, X. Kong, and H. Zhu, "Research on insulator defect detection algorithm of transmission line based on CenterNet," *PLOS ONE*, vol. 16, no. 7, C.-H. Chen, Ed., e0255135, Jul. 29, 2021, ISSN: 1932-6203. DOI: 10.1371/journal.pone.0255135. [Online]. Available: https://dx.plos.org/10.1371/journal.pone.0255135 (visited on 09/16/2022).

[10] X. Tao, D. Zhang, Z. Wang, X. Liu, H. Zhang, and D. Xu, "Detection of power line insulator defects using aerial images analyzed with convolutional neural networks," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 50, no. 4, pp. 1486–1498, Apr. 2020, Conference Name: IEEE Transactions on Systems, Man, and Cybernetics: Systems, ISSN: 2168-2232. DOI: 10.1109/TSMC.2018.2871750.

[11] H. Jiang, X. Qiu, J. Chen, X. Liu, X. Miao, and S. Zhuang, "Insulator fault detection in aerial images based on ensemble learning with multi-level perception," *IEEE Access*, vol. 7, pp. 61 797–61 810, 2019, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2915985. [Online]. Available: https://ieeexplore.ieee.org/document/8710232/ (visited on 09/16/2022).

[12] W. Chen, Y. Li, and Z. Zhao, "InsulatorGAN: A transmission line insulator detection model using multi-granularity conditional generative adversarial nets for UAV inspection," *Remote Sensing*, vol. 13, no. 19, p. 3971, Oct. 4, 2021, ISSN: 2072-4292. DOI: 10.3390/rs13193971. [Online]. Available: https://www.mdpi.com/2072-4292/13/19/3971 (visited on 09/16/2022).

[13] J. Zhang, P. Wang, Z. Zhao, and F. Su, "Pruned-YOLO: Learning efficient object detector using model pruning," in *Artificial Neural Networks and Machine Learning – ICANN 2021*, I. Farkaš, P. Masulli, S. Otte, and S. Wermter, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2021, pp. 34–45, ISBN: 978-3-030-86380-7. DOI: 10.1007/978-3-030-86380-7_4.

[14] B. Liberatori, C. A. Mami, G. Santacatterina, M. Zullich, and F. A. Pellegrino, "YOLO-based face mask detection on low-end devices using pruning and quantization," in *2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO)*, ISSN: 2623-8764, May 2022, pp. 900–905. DOI: 10.23919/MIPRO55190.2022.9803406.

[15] D. Wang and D. He, "Channel pruned YOLO v5s-based deep learning approach for rapid and accurate apple fruitlet detection before fruit thinning," *Biosystems Engineering*, vol. 210, pp. 271–281, Oct. 1, 2021, ISSN: 1537-5110. DOI: 10.1016/j.biosystemseng.2021.08.015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1537511021001999 (visited on 02/17/2023).

[16] X. Miao, X. Liu, J. Chen, S. Zhuang, J. Fan, and H. Jiang, "Insulator detection in aerial images for transmission line inspection using single shot multibox detector," *IEEE Access*, vol. 7, pp. 9945–9956, 2019, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2891123. [Online]. Available: https://ieeexplore.ieee.org/document/8603728/ (visited on 09/16/2022).

[17] Z. Zhao, Z. Zhen, L. Zhang, Y. Qi, Y. Kong, and K. Zhang, "Insulator detection method in inspection image based on improved faster r-CNN," *Energies*, vol. 12, no. 7, p. 1204, Mar. 28, 2019, ISSN: 1996-1073. DOI: 10.3390/en12071204. [Online]. Available: https://www.mdpi.com/1996-1073/12/7/1204 (visited on 09/16/2022).

[18] Z. Zhang, B. Zhang, Z.-C. Lan, *et al.*, "FINet: An insulator dataset and detection benchmark based on synthetic fog and improved YOLOv5," Aug. 10, 2022.

[19] D. L and P. K, *EPRI insulator defect image dataset*, Type: dataset, Aug. 11, 2021. DOI: 10.21227/VKDW-X769. [Online]. Available: https://ieee-dataport.org/competitions/epri-insulator-defect-image-dataset (visited on 09/16/2022).

[20] X. Liu, X. Miao, H. Jiang, and J. Chen, "Review of data analysis in vision inspection of power lines with an in-depth discussion of deep learning technology," *Annual Reviews in Control*, vol. 50, pp. 253–277, 2020, ISSN: 13675788. DOI: 10.1016/j.arcontrol.2020.09.002. arXiv: 2003.09802[cs, eess]. [Online]. Available: http://arxiv.org/abs/2003.09802 (visited on 09/16/2022).

[21] "Object detection: Models, architectures & tutorial [2023]." (), [Online]. Available: https://www.v7labs.com/blog/object-detection-guide,%20https://www.v7labs.com/blog/object-detection-guide (visited on 02/18/2023).

[22] "TensorFlow model optimization toolkit — pruning API." (), [Online]. Available: https://blog.tensorflow.org/2019/05/tf-model-optimization-toolkit-pruning-API.html (visited on 02/18/2023).

[23] "TensorFlow model optimization toolkit — weight clustering API." (), [Online]. Available: https://blog.tensorflow.org/2020/08/tensorflow-model-optimization-toolkit-weight-clustering-api.html (visited on 02/18/2023).