UNIVERSITY OF HERTFORDSHIRE
School of Physics, Engineering and Computer Science

MSc Data Science and Analytics

7COM1039-0901-2023 - Advanced Computer Science Masters Project

8th January 2024

Impact Analysis of Data Pre-processing in Vision-Based Accidental Fall Detection

Name: Binson Sam Thomas
Student ID: 21066966
Supervisor: Dr. Pusp Raj Joshi

MSc Final Project Declaration

This report is submitted in partial fulfilment of the requirement for the degree of Master of Science in Advanced Computer Science Masters Project at the University of Hertfordshire (UH).

It is my own work except where indicated in the report.

No human participants were involved in the research conducted for this MSc Project.

I hereby give permission for the report to be made available on the university website provided the source is acknowledged.

Signed: Binson Sam Thomas (21066966)

Acknowledgment

First and foremost, I thank God Almighty for the grace and blessings bestowed upon me throughout this journey.

I would like to express my sincere gratitude to my supervisor, Dr. Push Raj Joshi for his invaluable guidance and unwavering support throughout this project journey. I would also like to thank Dr. Na Helian, Mr. Say Meng Toh, and Mr. Kudiwa Pasipamire, whose expertise, encouragement, and constructive feedback have been invaluable throughout this research journey.

I extend heartfelt appreciation to my family for their enduring encouragement and understanding during this demanding journey. Your belief in my capabilities has been a constant source of motivation, and I am fortunate to have your love and support.

Abstract

Accidental falls are the second largest cause of unintentional injury deaths worldwide especially among the elderly. Addressing this issue requires a robust and accurate fall detection system that minimises equipment requirements. Based on existing research that performed fall detection using 2D CNN, this project aims to bridge the research gaps and the shortcomings, especially in recall and specificity through the introduction of new pre-processing techniques. With a primary focus on the preprocessing stage, this paper introduces a fall detection model that uses a 3D convolutional Neural Network(CNN). This model analysis involved the experimentation of various combinations of preprocessing techniques such as edge detectors, background subtraction, contour detection, and optical flow alongside its parameter variations. Through a series of meticulously planned experiments, the model's performance metrics were compared to identify its impact on fall detection, aligning with the methodology of the original experiment. The results showed substantial enhancement in performance metrics with certain models achieving a recall as high as 99.16%. The edge detector models showcased a superior performance over the original research model. The model showcased a tenfold reduction in processing and time complexities, highlighting a potential opportunity for creating an improved real-world fall detection system that exclusively uses a vision-based approach.

Keywords: *Accidental Fall Detection, Convolutional Neural Networks(CNN), Edge Detectors, Contour Detection, Optical Flow*

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1. Motivation

In today's world, artificial intelligence (AI) is making its mark in various sectors, from healthcare to education, robotics to finance (‘Artificial intelligence in businesses in different sectors’, 2020). Machine learning is pivotal in shaping the future of AI, thanks in part to advances in GPU-based processing. These advancements have pushed the boundaries of what was once thought impossible, building upon the initial ideas of Frank Rosenblatt's perceptron in 1957. While neural networks saw slow progress over the years, the past decade has been a turning point, driven by innovations such as the introduction of the Big Data ImageNet database, alongside hardware and software advancements, for example, faster Graphical Processing Units (GPU), TensorFlow and PyTorch frameworks. This period has also seen the introduction of significant Convolutional Neural Network(CNN) architectures like LeNet, AlexNet, VGG, GoogLeNet, and ResNet, which have greatly facilitated research in this field. Using deep learning to tackle common occurrences like accidental falls has immense potential for the future. Therefore, establishing a fall detection system based solely on image analysis serves as an appropriate starting point for my research. It has been recorded that a third of people over 65, and half of the people over 80, fall at least once a year. This pervasive issue exacts a considerable toll on healthcare systems, costing the NHS more than £2 billion annually and resulting in over 4 million bed days. Developing a precise and robust system for detecting accidental falls is paramount in mitigating mortality rates among the elderly population(*The human cost of falls - UK Health Security Agency*, 2014).

1.2. Aims and Objectives

Building upon the existing understanding of deep learning, a comprehensive literature review was conducted, revealing certain gaps and unanswered questions in one of the research papers that have the potential to significantly enhance the performance of fall detection models. The replicated portion of the research relevant to the project highlighted several shortcomings, including a lack of diversity in preprocessing techniques, substantial time consumption, and concerns regarding image resolution used. Within the constrained timeframe of the project, it was decided to perform a focused exploration of the crucial aspects of data preprocessing for my research.

This project aims to perform a comparative analysis of various preprocessing techniques to understand the feasibility of an improvement in the performance of the fall detection model in a vision-based approach using the UP-Fall detection dataset(Martínez-Villaseñor *et al.*, 2019). Our main objective is to identify the impact of various preprocessing techniques that could be used to improve the model performance and at the same time to reduce the time complexity. As a preliminary objective, our focus is to analyse the comparative impact of different edge detection methods on the accuracy and robustness of the fall detection model.

Additionally, as the research progressed, a secondary objective emerged in line with the primary objective. This objective involves conducting experiments by varying Canny edge detection parameters to understand how it affects the performance of the fall detection model, along with a possible identification of the optimal parameter settings.

Furthermore, since the existing research was performed using a questionably low resolution, an analysis of the impact of varying resolution demonstrated research potential. Hence, the third objective was to study the relationship between different image resolutions, along with a possible conclusion on which resolution yields the best results.

Lastly, along with the impact of different edge detectors, other image processing methods were tested and visually analysed with the existing feature extraction method, out of which, contour detection showed a potential to outperform the rest of the methods. So, for the final objective, our focus was to conduct experiments to understand whether the incorporation of contour detection, in addition to the existing feature extraction of optical flow, could have a measurable improvement in fall detection performance.

1.3. Research Questions

Based on the objective identified, this project aims to answer the following research questions:-

1. What is the comparative impact of different edge detection methods on the accuracy and robustness of the fall detection model?
2. How do variations in Canny edge detection parameters affect the performance of the fall detection model, and what are the optimal parameter settings?
3. What is the relationship between different image resolutions and the metrics used for fall detection, and which resolution yields the best results?
4. Does the incorporation of contour detection, in addition to optical flow, contribute to a measurable improvement in fall detection performance on the UP-Fall detection dataset?

1.4. Ethical Considerations

It is expected that the robustness of ethics is considered before the release of a publicly available dataset such as our UP-Fall detection dataset. It is acknowledged that the video shot in an indoor environment for model training of people falling raises potential privacy concerns, but it is assumed that the consent of the participants has already been obtained by the researchers. Our research does not involve human participation, thus no further ethical analysis was conducted.

1.5. Report Summary

This report is structured to gain a comprehensive insight into the impact of various preprocessing techniques for accidental fall detection. The report first introduces the project to the reader with an emphasis on the motivation behind the choice of subject followed by the research objectives. Chapter 2 explores the literature review conducted to identify the methods for implementation. Chapter 3 provides a brief understanding of all the data preprocessing techniques implemented throughout the project. Chapter 4 exclusively explains the methodology used, including the tools used, followed by the description of existing research and the modification implemented to the model. This chapter also explains how the experiments are structured. Chapter 5 contains all the results, segregated into sets of experiments addressing the respective research question. Each experiment set has a discussion section where the observations are discussed. Chapter 6 provides an overall conclusion to the

project addressing each research question. The following chapter 7 explains the limitations of the current model and the potential for future work.

2. Literature Review

With the advancement in Deep learning, a wide range of research is underway that incorporates a multimodal approach that leverages a combination of various sensors and image-processing techniques for fall detection. The methods including different sensors currently researched include wearable sensors and ambient or environmental sensors. A wearable sensor with an accelerometer can be used with any traditional machine-learning model to identify a fall. However, the implementation of sensor-based systems forces the user to don multiple sensor devices on various body parts, while also requiring regular device charging. These operational limitations result in a scenario where continuous 24/7 monitoring becomes impractical. Furthermore, these sensors are suspectable to environmental factors including humidity and temperature which may lead to a potential false alarm that could prove costly if repeated regularly (Shu and Shu, 2021). Hence, an image processing-based model, meticulously trained, emerges as a more viable alternative for identifying accidental falls.

A Few of the research available in this field (Doulamis, 2016; Shinde, Kothari and Gupta, 2018; Espinosa *et al.*, 2019; Zhang *et al.*, 2022; *Fall Detection Based on RetinaNet and MobileNet Convolutional Neural Networks | IEEE Conference Publication | IEEE Xplore*, no date) were reviewed to obtain a comprehensive understanding of all the vision-based approaches. To further indulge in this field, it was decided to proceed with the research as experimentation of the research paper, "A vision-based approach for fall detection using multiple cameras and convolutional neural networks: A case study using the UP-Fall detection dataset", authored by R Espinosa et al., which used optical flow and windowing technique for fall detection. The vision-based approaches used in this paper serve as the primary point of research and there are only a limited number of research existing that use a multimodal approach along with a Convolutional Neural Network (Kong *et al.*, 2019). The study focuses on replicating the original methodologies and architectures used in the vision-based fall detection system using CNN, along with the introduction of a combination of image processing and deep learning strategies along the existing pipeline to study and understand its impact in improving the performance of the model. Out of the multiple experiments conducted, the fall detection methods using 2D CNN was picked for replication. While the original model demonstrated remarkable overall performance, especially a notably high accuracy, there is room for improvement. Notably, the model exhibited a comparatively low specificity of 83.08%, which could be improved, moreover, the model achieved an impressive recall of 97.95% which is comparable to a state-of-the-art model, however, in a real-world scenario, given the context of accidental detection of elderly, a continual improvement in the metric is very important as the smallest difference could prove to be a lifesaving intervention. The replication of the original experiment was time-consuming during the optical flow model processing, which is a significant shortcoming.

To further understand the existing research in preprocessing techniques, an in-depth analysis revealed an optimal approach is the identification and isolation of the subject. Studies suggest that the performance of a model that uses optical flow can be improved with the use of edge detection methods(Liao and Liu, 2010). This is achieved by reducing the calculation of unimportant areas, thereby lowering the calculation time. Recent studies also indicate the importance of background subtraction in improving the model performance that uses optical flow. With the help of edge preprocessing and optical flow, we can deal with the fragmentation

caused by background subtraction and effectively separate the object (Beaupré, Bilodeau and Saunier, 2018). This method also improves the recall and precision which are key metrics on an accidental fall detection model. In a single static condition, an intricate combination of dynamic threshold methods and morphological filtering followed by contour projection analysis was already identified as a reliable motion human detection (Zhang and Liang, 2010), which could potentially be a faster alternative. Following this comprehensive literature review, it was decided to proceed with the research with a primary focus on background subtraction, edge detectors, and contour detection.

3. Data Preprocessing and Feature Extraction Techniques.

3.1. Background Subtraction

Background subtraction is a common method used in computer vision to detect a moving object in videos from a stationary camera. Since many applications do not need the whole scene from the video, this method has a wide range of applications, especially in CCTV cameras and cameras installed to monitor the elderly.

In our research, the background subtraction model used for all experiments involves the use of combinations of image processing techniques provided by OpenCV. The background isolation method used is the BackgroundSubtractorMOG function, which as the name suggests is a mixture of Gaussian-based background/foreground segmentation algorithms used to identify moving objects in a stationary background. One distinctive feature of MOG2 is its inherent ability to adapt to changing light conditions. For each pixel in the video, the variations are tracked over a period defined by the history parameter, following which it decides whether to consider that pixel as part of the background or foreground. All the relatively constant pixels were identified as part of the background, while the frequently changing pixels were identified to be part of the foreground. The use of this method yields an effective filtration of statics scenes and helps to emphasize the dynamic elements which is a quality that proves invaluable in applications like motion detection and object tracking.

OpenCV also provides morphological operations which are simple transformations based on the image shape. These operations help in refining the mask, ensuring the foreground object of interest is well-isolated.

3.2. Edge Detectors

Edge detectors, as the name suggests are a combination of mathematical methods that aim at identifying an edge in an image. An edge in an image can be categorised as a significant local change in the image intensity. For our research, we are mainly focussing on 3 different edge detectors, namely the Canny edge detector, Laplacian edge detector, and Sobel edge detector.

Canny Edge Detector is one of the widely used algorithms out there. It is known to produce a smoother and cleaner edge image than Prewit filters (M, 2022).

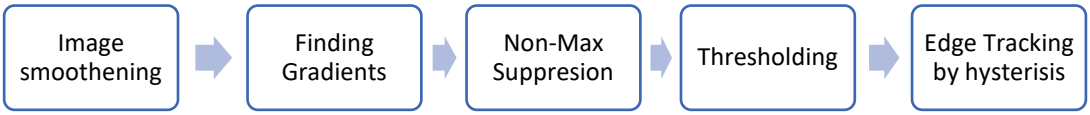

Figure 1: Canny Edge Detector

Figure 1 illustrates the processing involved in a canny edge detector. The input image is smoothened and to find out the gradients, a Sobel filter is used. Then we apply a non-max suppression, where pixels in the gradient directions are retained while the rest are suppressed. A thresholding is applied to remove the pixel below a set threshold which could be the result of noise in the data. Finally, we apply hysteresis tracking to make the pixels strong based on neighbouring pixels.

Laplacian edge detection uses the second-order derivative of the image to detect an edge. It measures the rate of change of gradient, which indicates the presence of an edge. It is passed through a single kernel filter of size 3x3 or 5x5, which then applies a threshold to the filtered image to obtain the edge map. It can identify edges in all directions but is highly susceptible to noise.

The Sobel edge detector is one of the oldest filters that use simple horizontal and vertical 3x3 filters to detect respective edges. They are easy to implement and have faster computation but are highly sensitive towards diagonal lines leading to thick edges.

3.3. Contour detection

Contours are a curve that joins all the continuous points along the boundary with the same colour and intensity. They are widely used in shape analysis as well as object detection and recognition. It uses different approximation methods such as CHAIN_APPROX_NONE, which stores all the boundary points. For simplicity, open CV provides an alternative called CHAIN_APPROX_SIMPLE that removes all the redundant points and compresses the contour, thereby providing faster processing.

3.4. Optical Flow

Optical flow is a technique used to estimate motion between two consecutive frames in a video sequence. Among various optical flow estimation methods, the Farneback method, introduced by Gunnar Farneback in 2003, is a popular choice due to its efficiency and robustness. Other optical flow methods are the Lucas-Kanade method, the Horn–Schunck method, and the Buxton–Buxton method.

For this study, we are using the Farneback method which is based on polynomial expansion. It approximates the neighbourhood of each pixel in both frames with quadratic polynomials using convolution kernels. By comparing the polynomial coefficients from consecutive frames, the displacement field is derived. This displacement field indicates how each pixel moves between two frames. The algorithm then refines the flow estimations over several iterations, considering larger neighbourhoods to capture larger motions.

4. Methodology and Implementation

4.1. Tools and Packages Used.

Python was used as the programming language for the entire project. The code editor Visual Studio code, along with Anaconda distribution, was used for the deployment.

The following Python packages were used for the research:

1. **PyDrive API**: This is a wrapper library of the google-api Python clients and was used to download the dataset.
2. **OpenCV**: To implement various computer vision techniques used across the research.
3. **PyTorch**: To train our models with the integration of CUDA to achieve GPU acceleration, vital in training a vast dataset.
4. **Matplotlib**: To plot the results of different experiments.

4.2. Challenges Of Implementation

One of the major challenges in implementing this project was meeting the computing requirements. A 278 GB of video data requires a powerful system capable of handling an extensive load. Training these models required GPUs which could not be acquired. Through a series of trial and error, the preprocessing was successfully implemented, following which the trainings were performed in the university-provided Remote Desktop Machines (RDPs), which introduced further complexities. Due to routine data cleanup, all the dependencies had to be manually reinstalled on every RDP log-in. Additionally, transferring pre-processed data to RDPs was addressed by compressing files for efficient transfer via RDP or cloud platforms (OneDrive). To optimize training, sessions were scheduled overnight, with batch handling codes. Despite challenges, the project was successfully executed within defined computing constraints.

4.3. Dataset Description

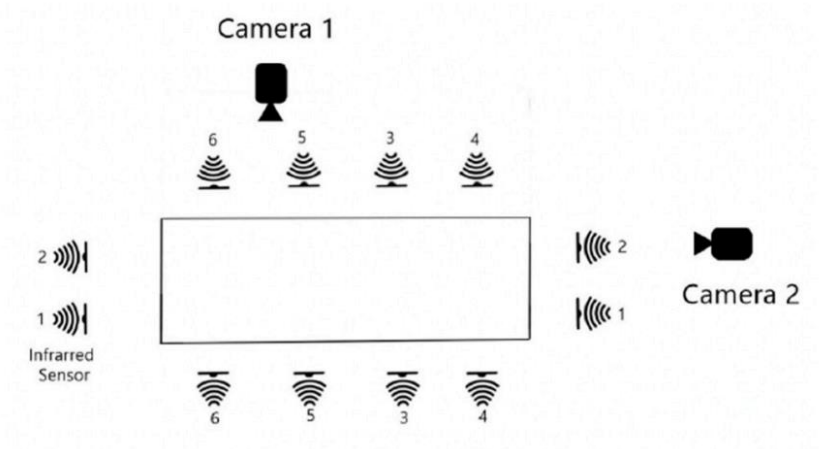

Figure 2: Camera and sensor setup

The foundation of this MSc project rests upon the analysis of the publicly available UP-Fall detection Dataset (Martínez-Villaseñor *et al.*, 2019), which encompasses a diverse array of 11

activities executed by 17 healthy individual subjects. They performed 3 trials per activity, captured using two cameras of lateral and frontal angles. The activities are summarised in Table 1. This gives a total combination of 1,122 videos, each shot at 18fps. Figure 2 illustrates the camera and sensor setup used to obtain the data. Even though the dataset includes the sensor data that depicts various activities and categorises them as falls and non-fall incidents, we are not using this data for the rest of the experiment. We are only focusing on using the video shot from two cameras.

Since our primary focus is to identify accidental falls, the dataset outputs must be classified into a Fall or a No-Fall Scenario. Hence, moving forward in this report all activities predicted to be between activity 1-5 will be mentioned as a fall scenario while all other activities will be mentioned as a no-fall scenario.

Activity	Description	Duration(s)
1	Falling forward using hands	10
2	Falling forward using knees	10
3	Backward Fall	10
4	Sideward Fall	10
5	Falling after sitting in an empty chair	10
6	Walking	60
7	Standing	60
8	Sitting	60
9	Picking up an object	10
10	Jumping	30
11	Laying	60

Table 1: List of activities

4.4. Original Model Design

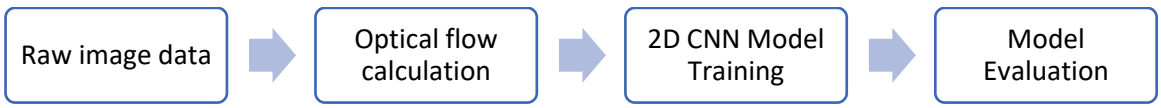

Figure 3: Original pipeline

As a starting point, the original model served as a foundation upon which we built and expanded. A few modifications were introduced to enhance the model capabilities, tailored specifically to obtain faster and reliable results which are described in detail in the next section.

Figure 3 displays the pipeline used in the original research. In the fall detection using the CNN segment of the original research, three CNN models were trained (i) a CNN model using visual features from Cam 1(Lateral View), (ii) a CNN model using visual features from Cam 2(Frontal View) and (iii) A CNN model using both cameras at the same time. After a thorough analysis of the results obtained, we identified that the CNN model that uses both cameras at the same time is the optimal choice for replication as this aligns closely with a real-world scenario wherein cameras could be positioned to different perspectives. This could potentially generate a model capable of adapting to diverse settings.

4.4.1. Data Preprocessing

As explained in the dataset description above, the existing dataset was segregated into folders based on a hierarchy of Subject/Activity/Trial/Camera. The dataset provided already had the frames separated in folders, thus there was no need to perform frame extraction. The original research adopted a sliding window approach to capture the temporal information. Following the same approach, the windowing technique was implemented. In this case, a 1-second window with a 0.5-second overlap is implemented as this was the best-performing windowing technique from the original research. This resulted in multiple 1-second window length series of images to be used in feature extraction.

4.4.2. Feature extraction

The optical flow method described in the previous section was chosen as the method to perform feature extraction. The combination of the horizontal and vertical movements gives rise to the resultant vector, whose magnitude, denoted as D, signifies the speed of the apparent motion at each pixel. The magnitude is calculated as:

$$D = \sqrt{U^2 + V^2}$$

The optical flow information combined with the CNN model was used to extract the features for training. Instead of using the optical flow values directly, we computed the average optical flow over each window and stored this information. The optical flow's magnitude was focused upon, and any value below a threshold of 1 was set to 0, effectively filtering out very low-motion regions. This choice was made to emphasize the more significant motion within each window. The calculated average magnitudes were then saved to a CSV file. Finally, referencing the optical flow file provided by the dataset, appropriate labels were appended to each window's data.

4.4.3. 2D CNN Model

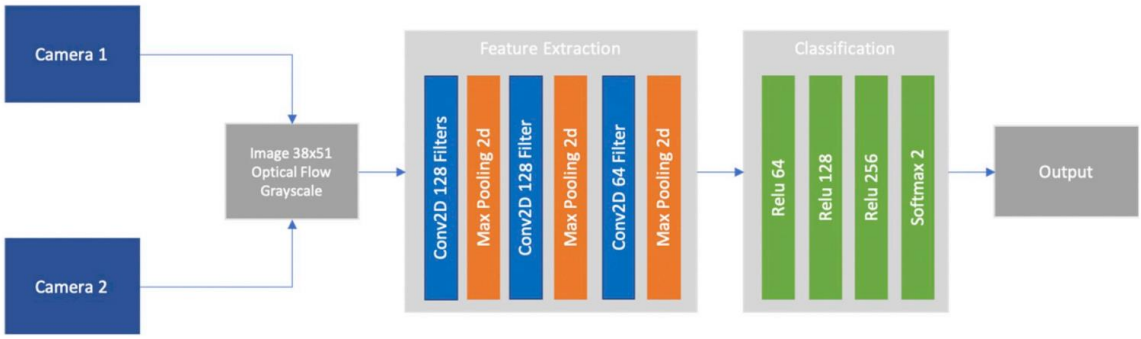

Figure 4: Original 2D CNN Model

Among the multiple methods proposed in the original research, the 2D Convolutional Neural Network(CNN), shown in Figure 4 model was created, and the structure of the model is as follows:-

- **2D Convolutional Layers for Feature Extraction**:
  - Layer 1 consists of 128 convolution filters with a kernel size of 3 x 3.
  - Layer 2 consists of 128 convolution filters with a kernel size of 3 x 3.

- Layer 3 consists of 64 convolution filters with a kernel size of 3 x 3.

- **Max-Pooling Layers**:
  - 2D max-pooling layers are used after each convolutional layer to reduce spatial dimensions.

- **Fully Connected Layers for Fall Detection**
  - Layer 1 consists of 64 ReLU units.
  - Layer 2 consists of 128 ReLU units.
  - Layer 3 consists of 254 ReLU units.

- **Output Layer:**
  2D SoftMax layer with a single binary output:
  - Fall (represented as 1)
  - No-fall (represented as 0)

In context to the dataset, If the activity identified is between 1 and 5, then it is identified as a fall whereas all the other activities are identified as a non-fall.

4.5.Modified Model Design and Implementation

Figure 5: Final Base Model Pipeline

The final base model shown in Figure 5, used in all the experiments is a modified version of the original research model. It includes changes identified through comprehensive background research and methods tested based on the observations made during the course of our research implementation. This involves identifying the optimal methods to address the gaps in the research where specific details of implementation were missing along with methods to improve the model outputs based on visualisation of the processed videos.

4.5.1. Data Preprocessing and Feature Extraction

In line with the original methodology, the data preprocessing and optical flow approach remains consistent. Additionally, a few preprocessing techniques were implemented to address the noise and errors observed during the frame visualisation.

A normalisation was performed to adjust the contrast of each frame, making sure that the pixel intensities were properly scaled. Applying the histogram equalisation to each frame was also tested and was discarded later since no noticeable difference was observed. As the primary focus of the research itself was to try different data preprocessing techniques, respective changes were made in terms of each experiment. Appendix B shows the code used to implement the preprocessing of the base model. Only the contour detection model and the replication model involved the use of an optical flow which was determined using the Farneback method from the OpenCV library.

4.5.2. Data Loading

As we transitioned to 3D CNNs, the approach to data loading needs to be adjusted. The primary difference lies in how we handle the temporal aspect. Instead of compressing or averaging optical flow over a time window, we now create sequences of optical flow frames. Instead of averaging and storing data in CSV format, we store sequences of optical flow components as NumPy arrays. Before the NumPy arrays go into the 3D CNN, the labels are extracted from the optical flow file provided. Instead of appending the label to the CSV file, the labels are now stored in a dictionary alongside the window. Appendix C shows the code used to implement this data loading.

4.5.3. Balancing Dataset

During the implementation of the base model itself, we identified that the dataset was unbalanced with 60,716 no-fall samples and 1916 fall samples. Since an even distribution is always desirable, as it can help to avoid the model from being biased toward the majority class, we decided to balance the data by downsampling the non-fall scenarios to 1916. This was done by iterating through the entire data set to identify the labels of each window. We then identify the total number of Fall Samples and randomly choose the same number of no-fall samples without replacement. This ensures that random instances of the majority class (non-falls) are removed until we are left with a more balanced distribution. Appendix D shows the code used to implement the downsampling.

4.5.4. 3D CNN Model

Acknowledging the nature of the current dataset which uses video input, it was decided to switch to a 3D CNN model as this model architecture excels in extracting both spatial and temporal features from a video, thereby enhancing our chances of providing a robust representation of the underlying patterns in the data (Kamangir *et al.*, 2022). In applications for fall detection, temporal dynamics play a crucial role, and this is where 3D CNNs have an advantage over 2D CNNs. It was decided to experiment with 3D CNNs to explore whether they could offer enhanced accuracy and capture nuances of fall patterns over time with greater accuracy.

The 3D CNN follows the same Architecture as the 2D CNN model with 2D layers being changed to 3D layers.

- **3D Convolutional Layers for Feature Extraction:**
  - Layer 1 consists of 128 convolution filters with a kernel size of 3 x 3 x 3.
  - Layer 2 consists of 128 convolution filters with a kernel size of 3 x 3 x 3.
  - Layer 3 consists of 64 convolution filters with a kernel size of 3 x 3 x 3.

- **Max-Pooling Layers:**
  - 3D max-pooling layers are used after each convolutional layer to reduce spatial dimensions.

- **Fully Connected Layers for Fall Detection**
  - Layer 1 consists of 64 ReLU units.
  - Layer 2 consists of 128 ReLU units.
  - Layer 3 consists of 254 ReLU units.

- **Output Layer:**

2D SoftMax layer with a single binary output:
- Fall (represented as 1)
- No-fall (represented as 0)

4.5.5. Model Training

Instead of using the conventional approach of a single dataset for training and testing, it was decided to use both balanced and unbalanced datasets in combination for training and testing as it introduces a more realistic representation of real-world scenarios. This allowed us to test the model in a diverse range of data distribution, as it closely resembles complexities and variations of practical application where there exists a large number of no-fall scenarios.

For the training of the model, before feeding the data into the 3D model, the balanced data is split into 75% training, and 25% validation. We use our unbalanced dataset for testing and evaluating the model. The rest of the model hyperparameters and metrics were set to an industry standard for the sake of research that aligns with our objective. We used the Adam optimizer, with a learning rate of 0.0001 and binary cross-entropy as the loss function, over a maximum of 50 epochs. Appendix G shows the code for the main function used for training.

4.6. Result Evaluation Metrics

The model evaluation metrics are used to gain insights about the performance of our model as it provides a quantitative measurement of how good the model is at performing the given task on the dataset. Since our model is a classification model capable of classifying the data into Fall and No-Fall scenarios, we use the following metrics to evaluate our model.

1. **Accuracy**

Accuracy can be described as the ratio of correctly predicted instances to the total instances. In our case, it would be the proportion of true results among the number of fall and no-fall scenarios.

$$Accuracy = \frac{True\ Positives + True\ Negatives}{Total\ Instances}$$

2. **Precision**

It is the ratio of correctly predicted positive observations to the total predicted positives. In this case, it can be described as the proportion of actual fall instances among the instances that are predicted as falls by the model.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

3. **Recall (Sensitivity)**

This is the most important metric for all our analysis. It is the ratio of correctly predicted positive observations to all the actual positives. In our case, it's the proportion of actual falls that are correctly identified by a binary classification model out of all the actual

falls instances. High sensitivity is a key metric for a fall detection model since it implies that the model is effective at capturing most of the actual falls.

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

4. **Specificity**

The ratio of correctly predicted negative observations to all the actual negatives. In our case, it's the proportion of actual no-falls that are correctly identified by a binary classification model out of all the actual no-falls instances.

$$Specificity = \frac{True\ Negatives}{True\ Negatives + False\ Positives}$$

5. **F1-Score**

The weighted average of Precision and Recall

$$F1 - Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Apart from these metrics, we use the confusion matrix and training phase metric plot to gain further insights into the model performance. We have structured all our results discussion (Sections 5.2.7, 5.3.8, 5.4.6, 5.5.1) to discuss the model performance during the training phase, followed by a discussion on the model evaluation metrics obtained during the testing phase, and conclude after a thorough analysis of the confusion matrix.

4.7. Experiments Setup

The experiment setup aligned seamlessly with the overarching objectives, ensuring that the experiments were tailored to address the specific research questions, providing a purposeful approach to the investigation. The original model was replicated, followed by experiments divided into 4 sets, aimed at addressing each research question. Following are the parameters used in all the experiments, with other parameters being changed according to the experiment specifications.

Common Parameters used:

- Learning rate: 0.0001
- Batch size: 32
- Number of epochs: 50
- Training dataset: Imbalanced original dataset.
- Testing dataset: Imbalanced original dataset

4.7.1. Original Model Replication

Even though the original optical flow method is not mentioned in the original research, the Farneback algorithm was picked to proceed with the research. The model was trained using a 2D CNN that follows the same pipeline mentioned in Figure 4: Original 2D CNN Model.

4.7.2. Experiment Set-1

The experiment aims to address our 1st research question.

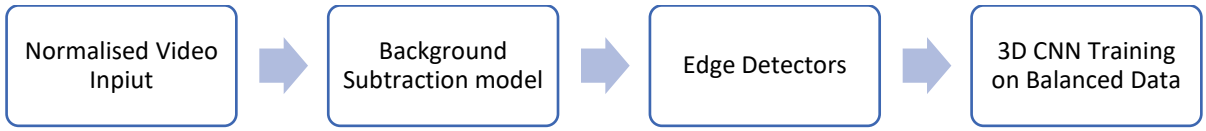

Figure 6: Experiment 1 Pipeline - Type I

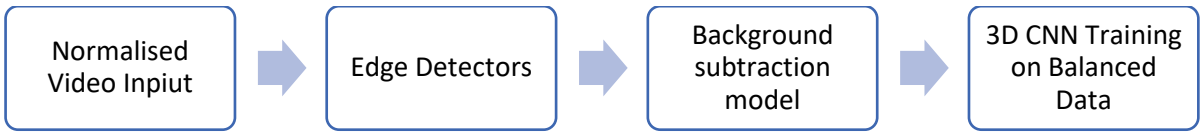

Figure 7: Experiment Set 1 Pipeline - Type II

To comprehensively assess the impact of various edge detectors on the model's performance, drawing upon the insights from the background research, three different edge detectors were picked. Specifically, the Canny Edge Detector, Laplacian Edge Detector, and Sobel Edge Detector were selected as preprocessing techniques for a detailed examination of their impact on the overall model outcomes. To further expand the scope of this experiment, we used background subtraction in combination with these edge detectors. Two types of models, namely Type I and Type II were created for a comprehensive analysis with only difference of processing order of edge detector and background subtraction.

The background subtraction model used here is a combination of two preprocessing techniques offered by OpenCV where the input is passed on to OpenCV's MOG2(Mixture of Gaussians) background subtractor. After the background subtraction, to mitigate noise and other small artifacts, we applied morphological operation, specifically the 'close' operation (which is dilation followed by an erosion) was used to close small holes in the foreground, and the 'open' operation (an erosion followed by a dilation) was applied to remove noise. Figure 6 and Figure 7 show the Type I and Type II pipelines respectively. A total of 6 experiments were created for the research:-

1. Experiment 1.1
   Type I Model using Canny edge detectors.
2. Experiment 1.2
   Type II Model using Canny edge detectors.
3. Experiment 1.3
   Type I Model using Laplacian edge detectors.
4. Experiment 1.4
   Type II Model using Laplacian edge detectors.
5. Experiment 1.5

Type I Model using Sobel edge detectors.

6. Experiment 1.6

Type II Model using Sobel edge detectors.

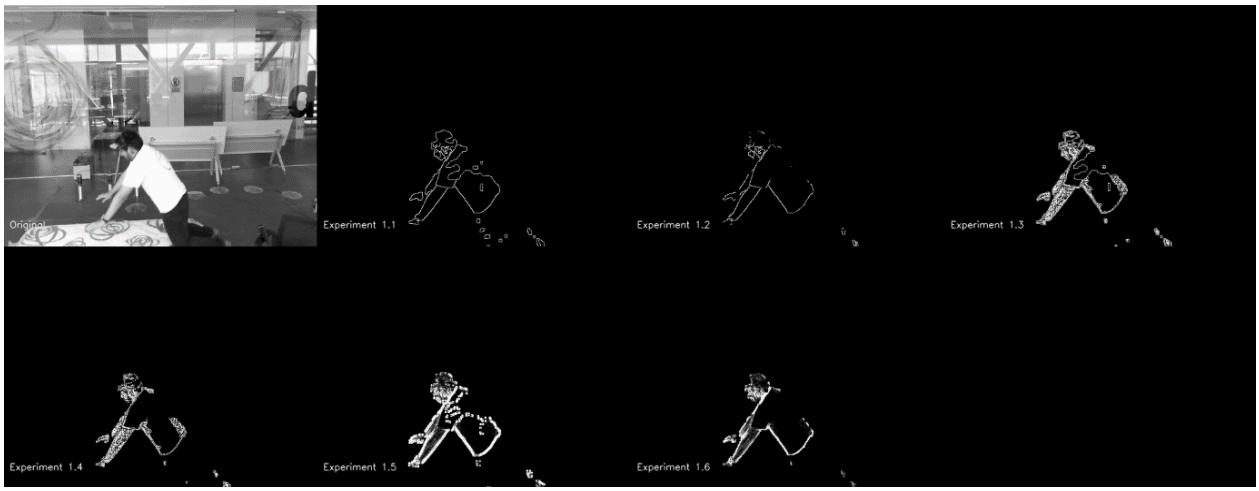

Figure 8: Experiment Set 1 Visualisation.

Figure 8 visually demonstrates the comparison of the set-1 experiments. The model effectively isolates the subject without resorting to optical flow and this underscores the model's proficiency in achieving subject isolation without the need for conventional feature extraction techniques. Appendix H shows the code used for visualising the processed frames. Appendix E shows the function used to compute the edge detector for experiment 1.1 which was modified according to different edge detectors specific to the experiment.

4.7.3. Experiment Set-2

The experiment aims to address our 2nd research question.

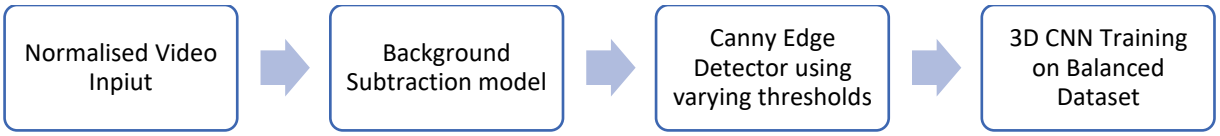

Figure 9: Experiment 2 Pipeline

It was observed that the Canny edge detector exhibits commendable performance without compromising on dataset size and processing time. In this series of experiments, a thorough model comparison was conducted by systematically varying the upper and lower threshold values of the model used in Experiment 1.1 which uses a canny edge detector. Figure 9 shows the pipeline used for this set of experiments. The OpenCV documentation(*OpenCV: Canny Edge Detection*, no date) recommends a lower to the upper threshold ratio between 1:3 and 1:2. For the rest of the report, the ratio 1:3 will be referenced as Type I and the ratio 1:2 will be referenced as Type II. The comparison involved the usage of a series of parameters of both types which are listed below.

Type I (1:3 Ratio)

1. Experiment 2.1

Lower Threshold = 25 and Upper Threshold= 75

2. Experiment 2.2

Lower Threshold = 50 and Upper Threshold= 150

3. Experiment 2.3

Lower Threshold = 75 and Upper Threshold= 225

Type II (1:2 Ratio)

4. Experiment 2.4

Lower Threshold = 50 and Upper Threshold= 100

5. Experiment 2.5

Lower Threshold = 75 and Upper Threshold= 150

6. Experiment 2.6

Lower Threshold = 100 and Upper Threshold= 200

7. Experiment 2.7

Lower Threshold = 125 and Upper Threshold= 250

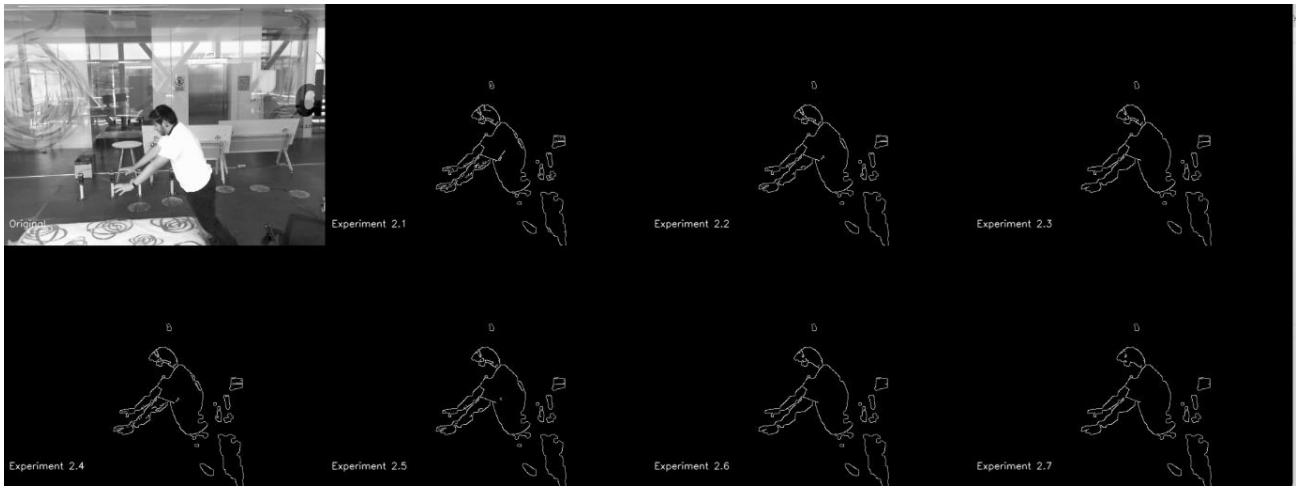

Figure 10: Experiment Set 2 visualisation.

Figure 10 visually demonstrates the comparison of set-2 experiments. The model has effectively isolated the subjects, displaying only minor visible differences. Notably, models with a 1:2 ratio appear to slightly outperform others in isolating subject edges. Nevertheless, it remains uncertain whether the observed performance will generalize until the model undergoes further training.

4.7.4. Experiment Set-3

The experiment aims to address our 3rd research question.

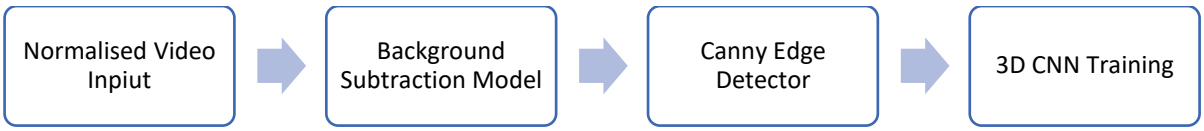

Figure 11: Experiment 3 Pipeline

In this series of experiments, a thorough model comparison was conducted by systematically varying the resolution of the output images. This analysis aimed to understand the effects of escalating the resolution, particularly in response to the original research employing a notably low resolution of 38x51. Figure 11 shows the pipeline used for this set of experiments. The comparison involved the utilization of multiples of the base resolution (38x51), and was rigorously evaluated across the following resolutions:

1. Experiment 3.1 – Resolution of 38 x 51 which is the base resolution.
2. Experiment 3.2 – Resolution of 57x76 which is 1.5 times the base resolution.
3. Experiment 3.3 – Resolution of 76x102, which is 2 times the base resolution.
4. Experiment 3.4 – Resolution of 95x128, which is 2.5 times the base resolution.
5. Experiment 3.5 – Resolution of 114x153, which is 3 times the base resolution.

4.7.5. Experiment 4

The experiment aims to address our 4th research question.

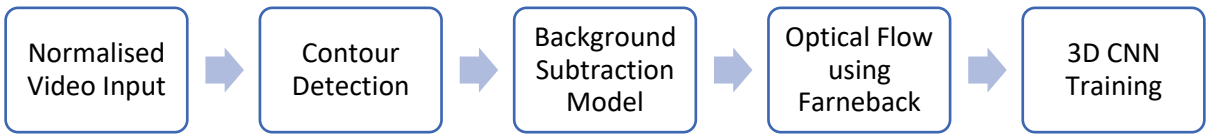

Figure 12: Experiment 4 Pipeline

Figure 12 shows the pipeline used for the contour detection model. It was compared with the original research and edge detection models to conclude the feasibility. This model, at a code level, involved multiple refinements to minimise noises using Gaussian blur and median blur. We also used a canny edge detector to identify the threshold to be used for the contour detection before passing them for background subtraction. The CHAIN_APPROX_SIMPLE approximation method is used here to increase processing speed. To analyse the impact till this point, the processed frames were visualised as shown in Figure 13 before extracting the optical flow image.

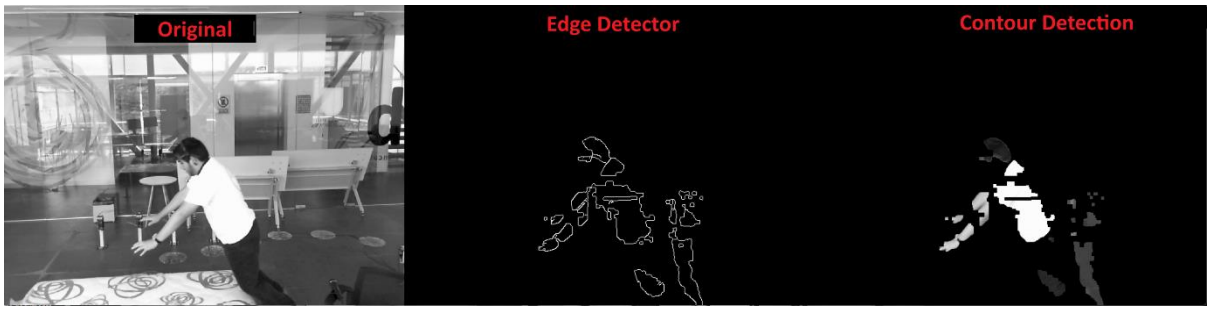

Figure 13: Experiment 4 Visualisation

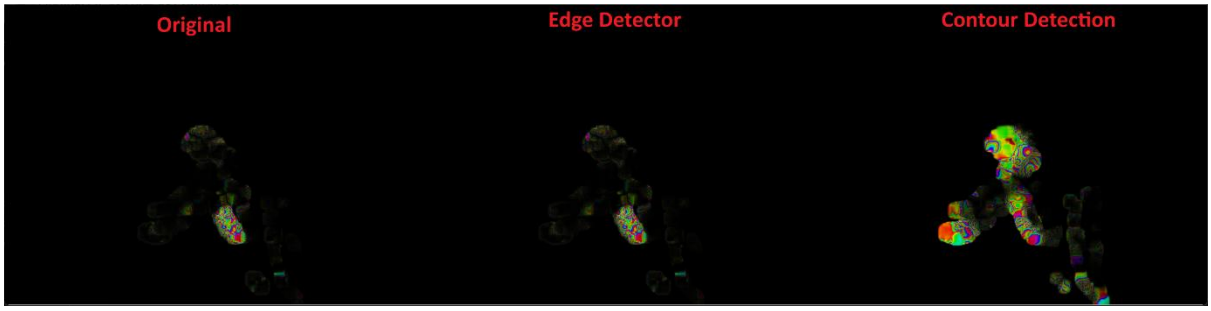

Figure 14: Experiment 4 Optical Flow

The edge detector model as well as the contour detection model demonstrates satisfactory results and was successful in identifying the subject with minimal noise, however, a closer inspection of the optical flow output in Figure 14 highlights the contour detection model's capability of generating a clear, distinct, and robust optical flow output, justifying its integration into our research. Notably, all other models using edge detectors, failed to exhibit a comparable visual clarity, leading to the exclusive utilisation of optical flow in our contour detection model. Appendix F shows the function code used for generating the contour detection model.

5. Experiment Results

5.1.Original Model Replication

Observation	Values
Size of Unbalanced Dataset	16.5 GB
Size of the Balanced Dataset	1.43 GB
Total time taken for processing	4Hr 10 Mins
Total time taken for Training	1 Hr
Test Loss	0.5772
Test Accuracy	0.9450
Test Precision	0.3564
Test Recall	0.9885
Test Specificity	0.9437
Test F1-Score	0.5239

Table 2:Original Model Observations

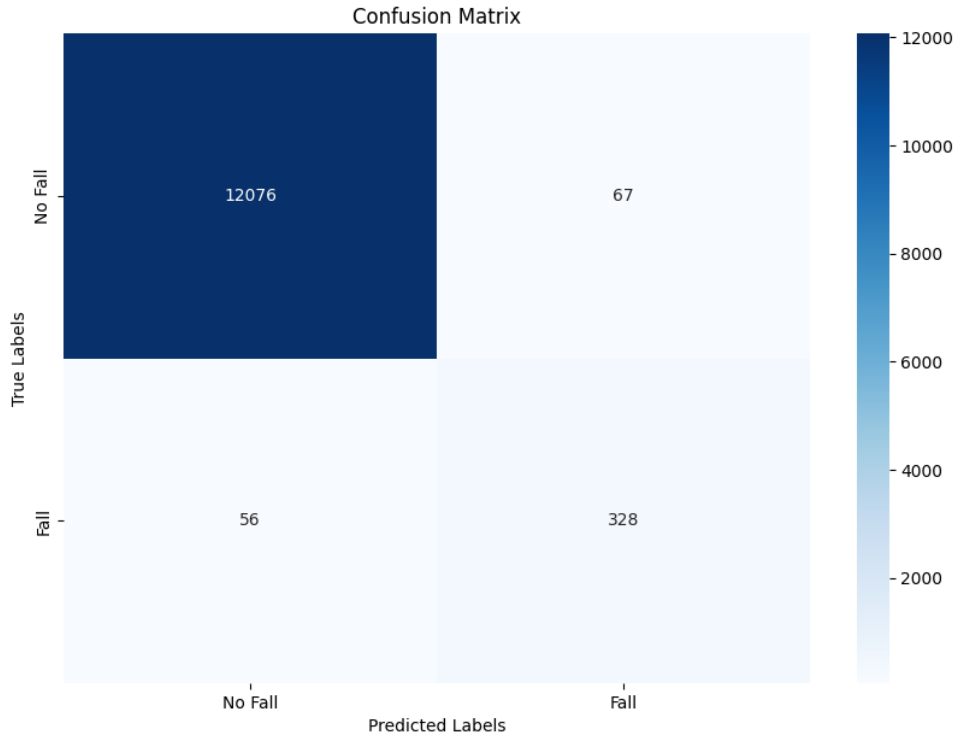

Figure 15: Original model Confusion matrix

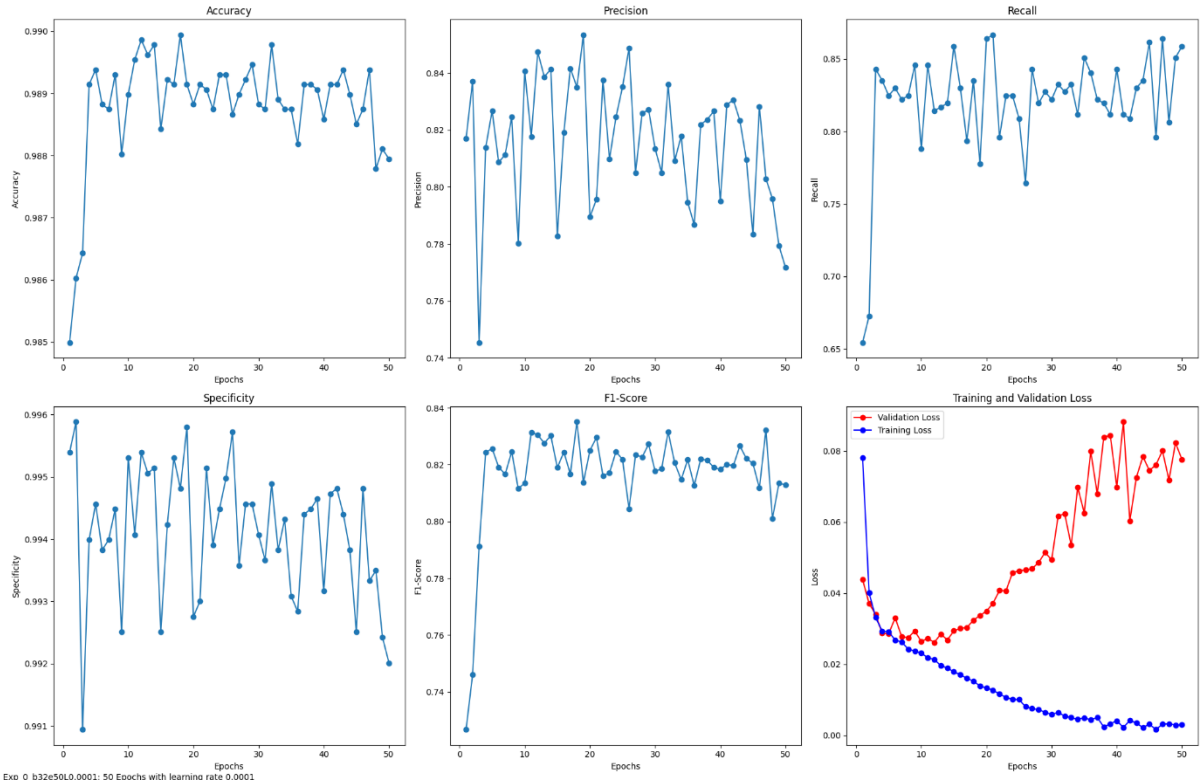

Figure 16: Replication results

5.2.Experiment Set 1 Results

5.2.1. Experiment 1.1 – Canny Type I

Observation	Values
Size of Unbalanced Dataset	1.95 GB
Size of the Balanced Dataset	122 MB
Total time taken for processing	2Hr 10 Mins
Total time taken for Training	8 Mins 49 Seconds
Test Loss	0.5772
Test Accuracy	0.9450
Test Precision	0.3564
Test Recall	0.9885
Test Specificity	0.9437
Test F1-Score	0.5239

Table 3: Experiment 1.1 Observations

Figure 17:Experiment 1.1 Confusion Matrix

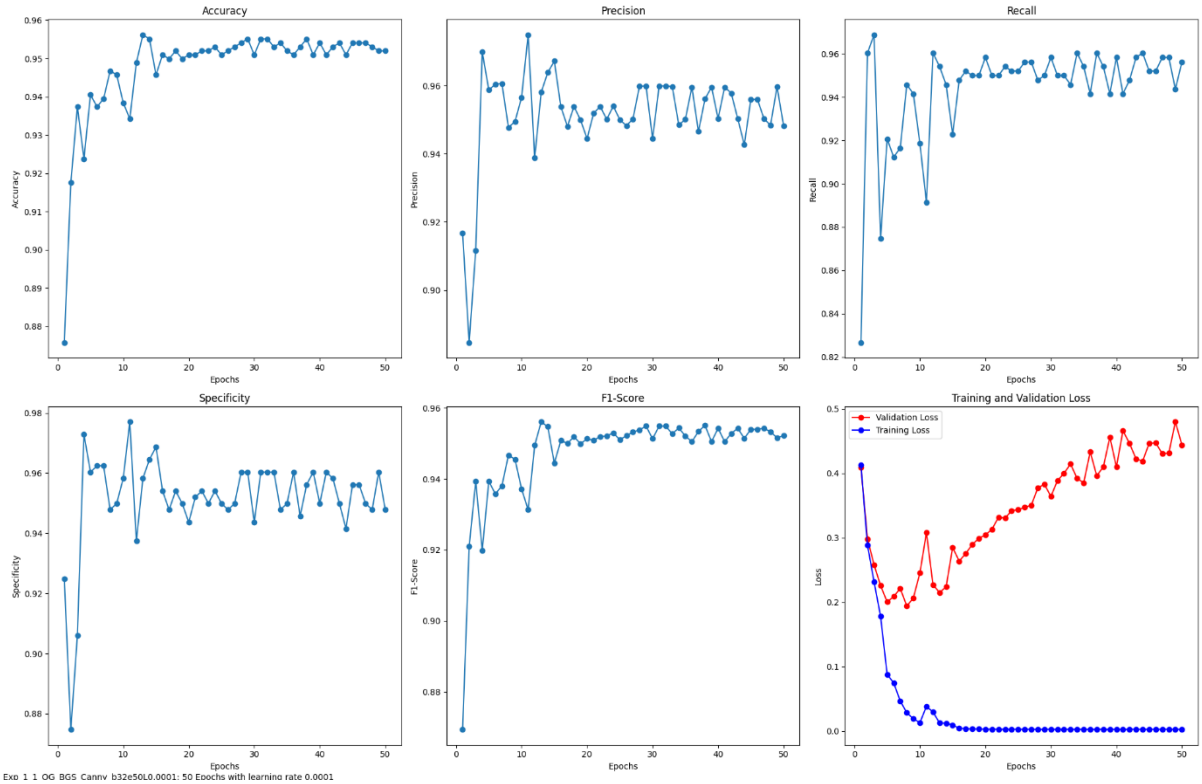

Figure 18: Experiment 1.1 results

5.2.2. Experiment 1.2 - Canny Type II

Observation	Values observed
Size of Unbalanced Dataset	1.95 GB
Size of the Balanced Dataset	122 MB
Total time taken for processing	2Hr
Total time taken for Training	9 Mins 12 Seconds
Test Loss	0.9357
Test Accuracy	0.9154
Test Precision	0.2629
Test Recall	0.9781
Test Specificity	0.9135
Test F1-Score	0.4144

Table 4: Experiment 1.2 Observations

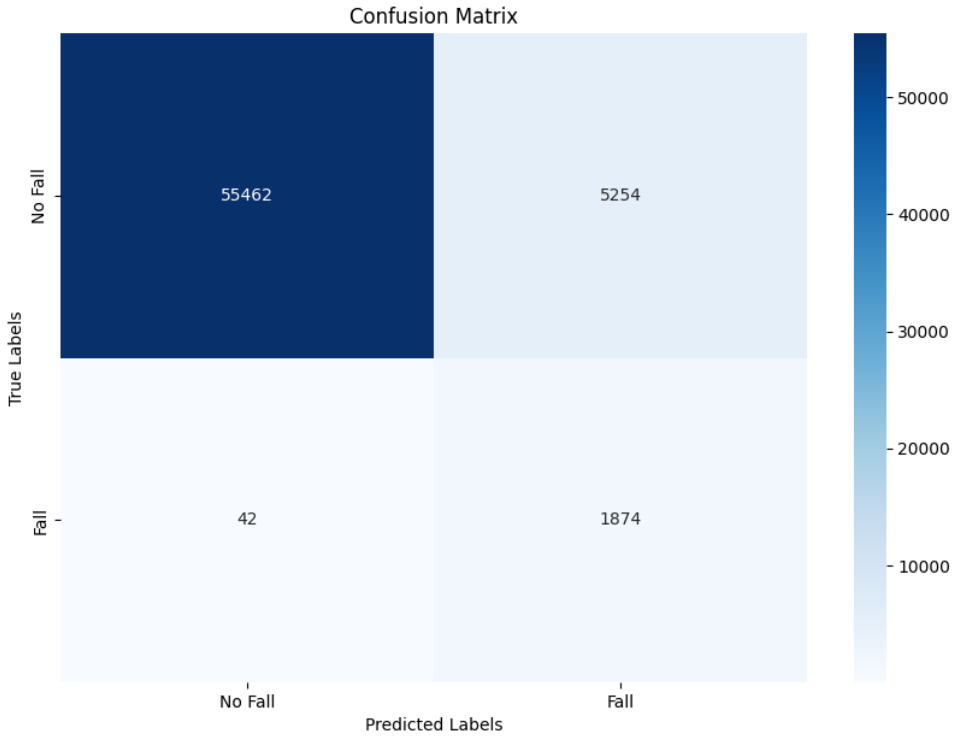

Figure 19:Experiment 1.2 Confusion Matrix

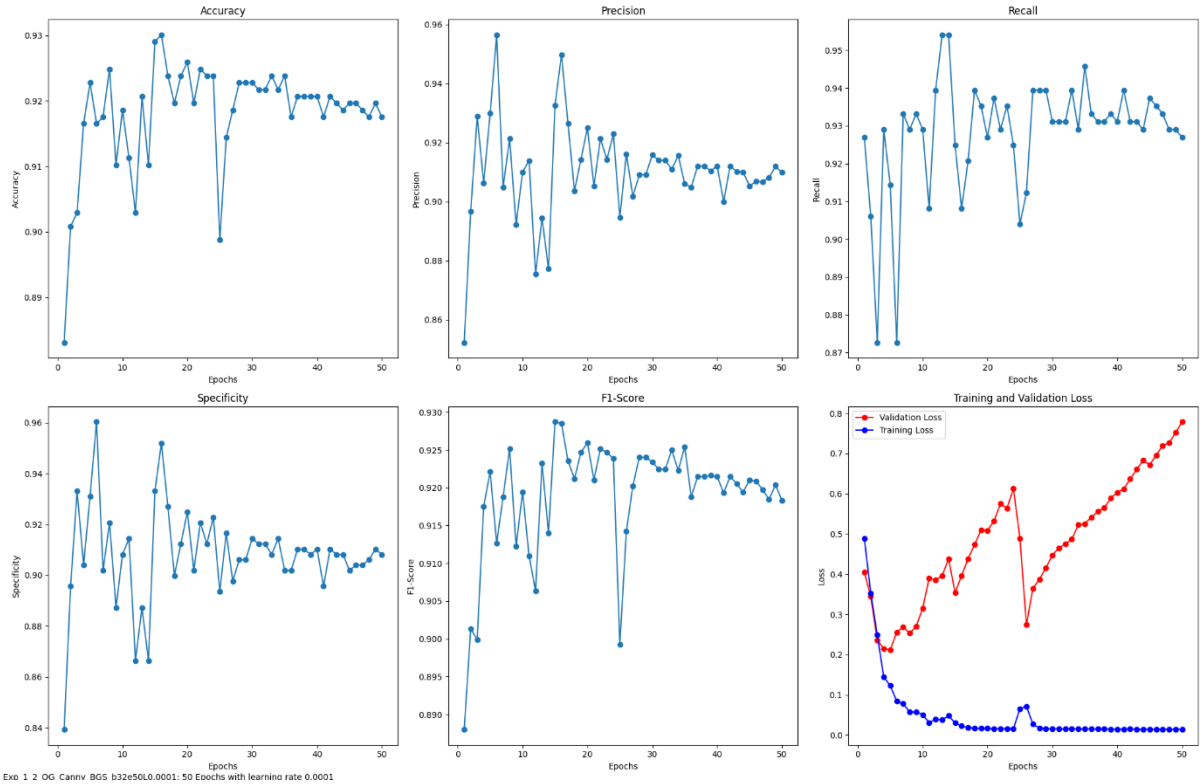

Figure 20:Experiment 1.2 results

5.2.3. Experiment 1.3 - Laplacian Type I

Observation	Values
Size of Unbalanced Dataset	1.95 GB
Size of the Balanced Dataset	122 MB
Total time taken for processing	2Hr 7 Mins
Total time taken for Training	9 Mins 49 Seconds
Test Loss	0.4837
Test Accuracy	0.9543
Test Precision	0.3999
Test Recall	0.9854
Test Specificity	0.9533
Test F1-Score	0.5689

Table 5: Experiment 1.3 Observations

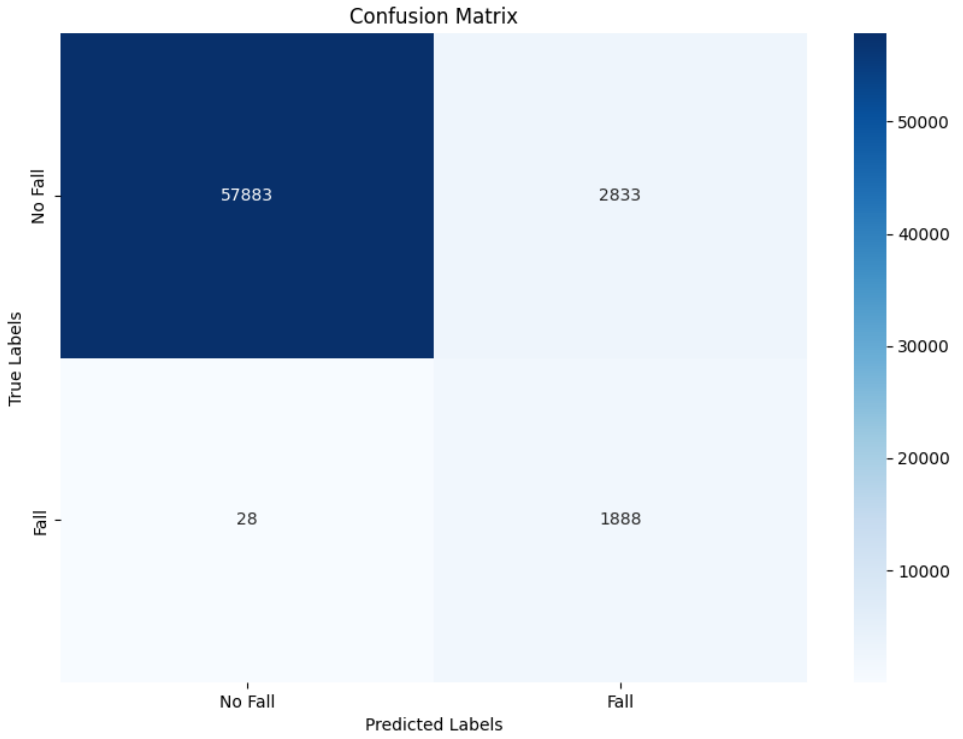

Figure 21: Experiment 1.3 Confusion Matrix

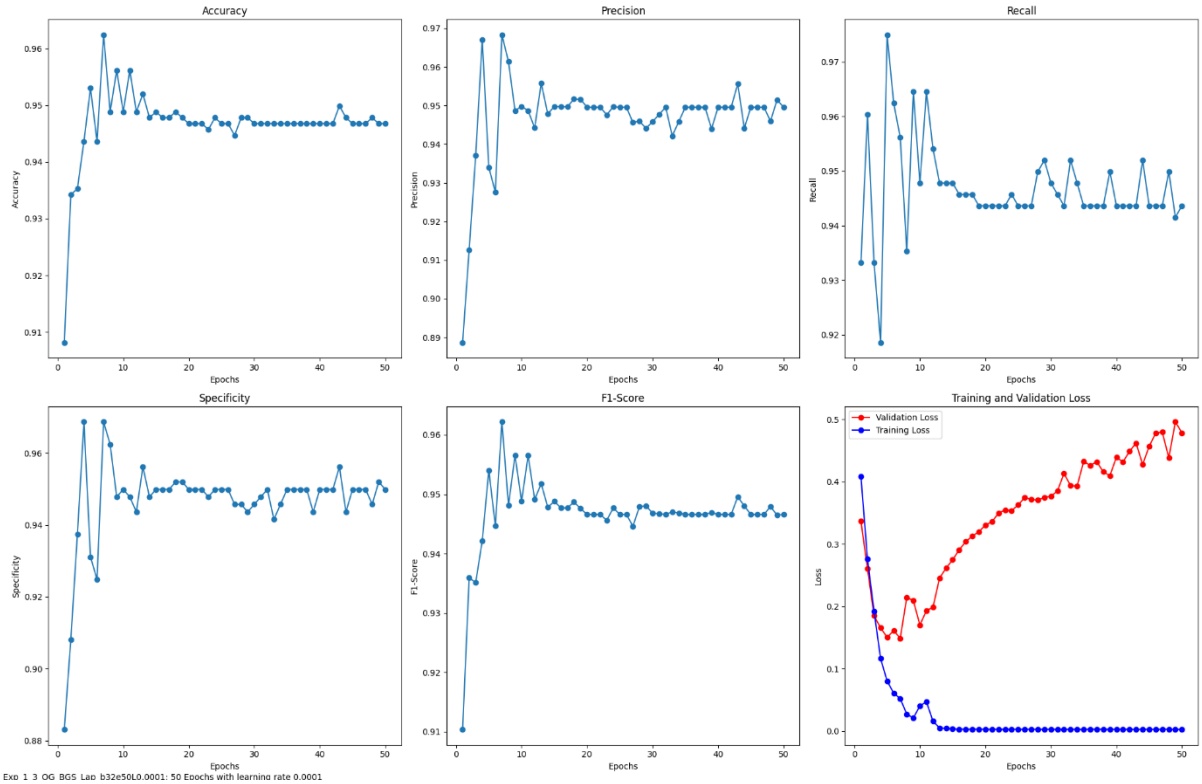

Figure 22:Experiment 1.3 results

5.2.4. Experiment 1.4 - Laplacian Type II

Observation	Values observed
Size of Unbalanced Dataset	1.95 GB
Size of the Balanced Dataset	122 MB
Total time taken for processing	2Hr 6 Mins
Total time taken for Training	10 Mins 15 Seconds
Test Loss	0.6158
Test Accuracy	0.9459
Test Precision	0.3600
Test Recall	0.9880
Test Specificity	0.9446
Test F1-Score	0.5277

Table 6: Experiment 1.4 Observations

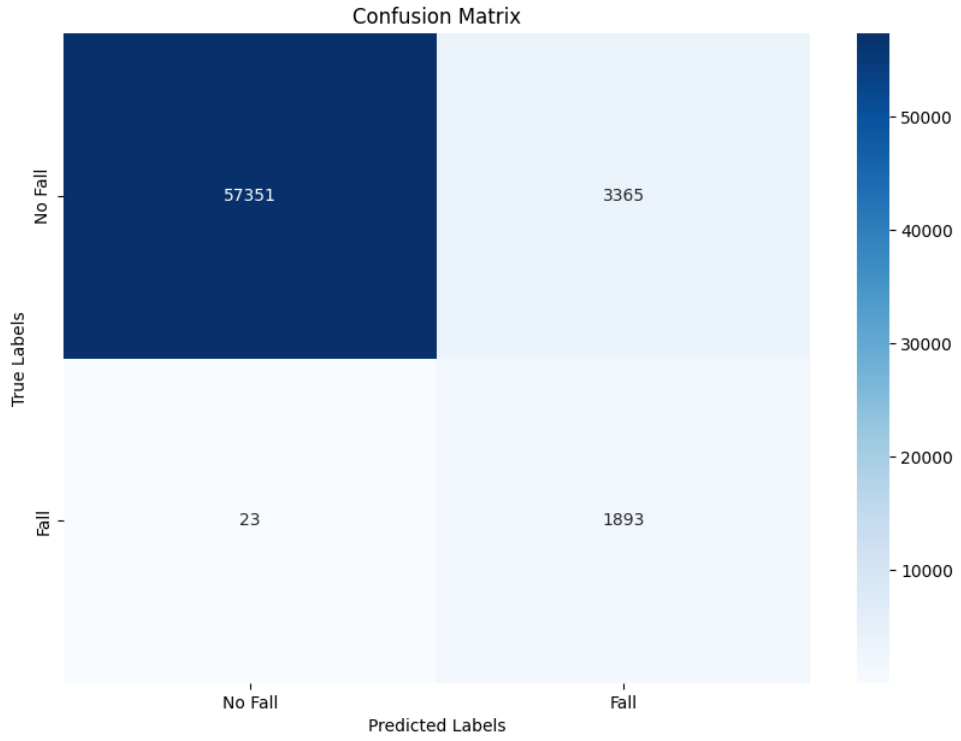

Figure 23:Experiment 1.4 Confusion Matrix

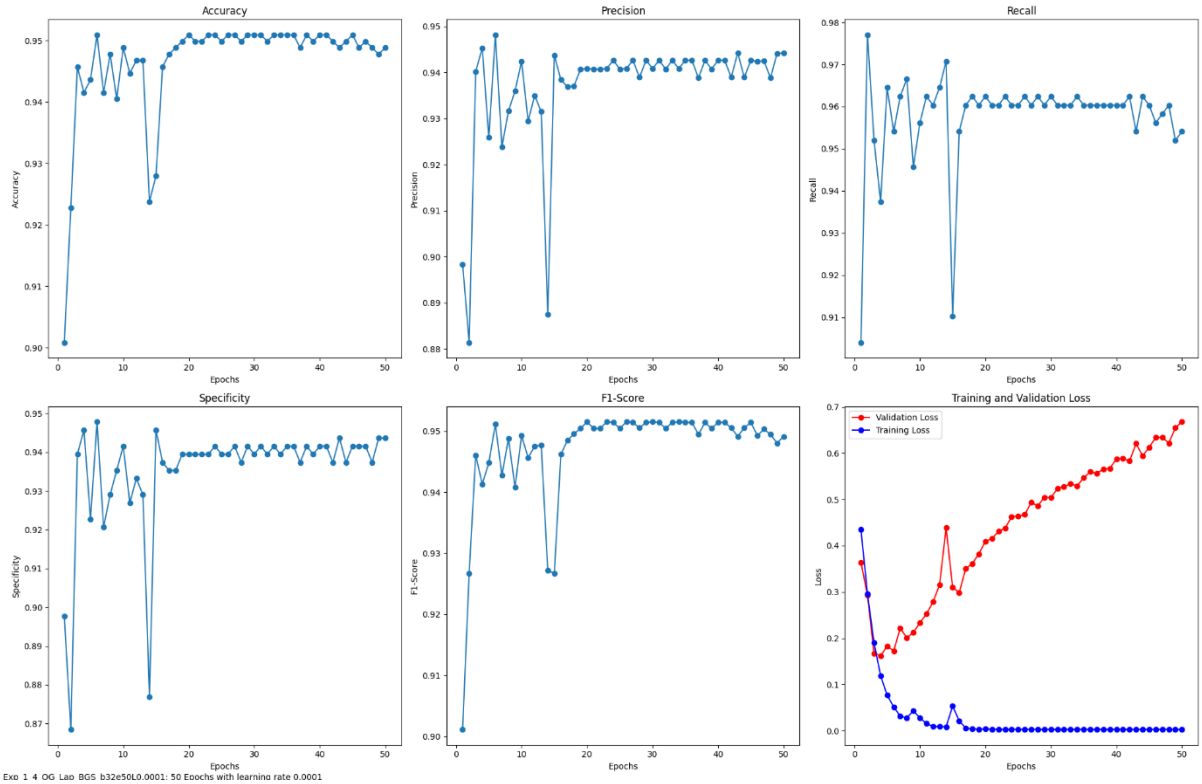

Figure 24:Experiment 1.4 results

5.2.5. Experiment 1.5 – Sobel Type I

Observation	Values observed
Size of Unbalanced Dataset	15.40 GB
Size of the Balanced Dataset	940 MB
Total time taken for processing	2Hr 53 Mins
Total time taken for Training	9 Mins 62 Seconds
Test Loss	0.8487
Test Accuracy	0.9432
Test Precision	0.3485
Test Recall	0.9843
Test Specificity	0.9419
Test F1-Score	0.5147

Table 7: Experiment 1.5 Observations

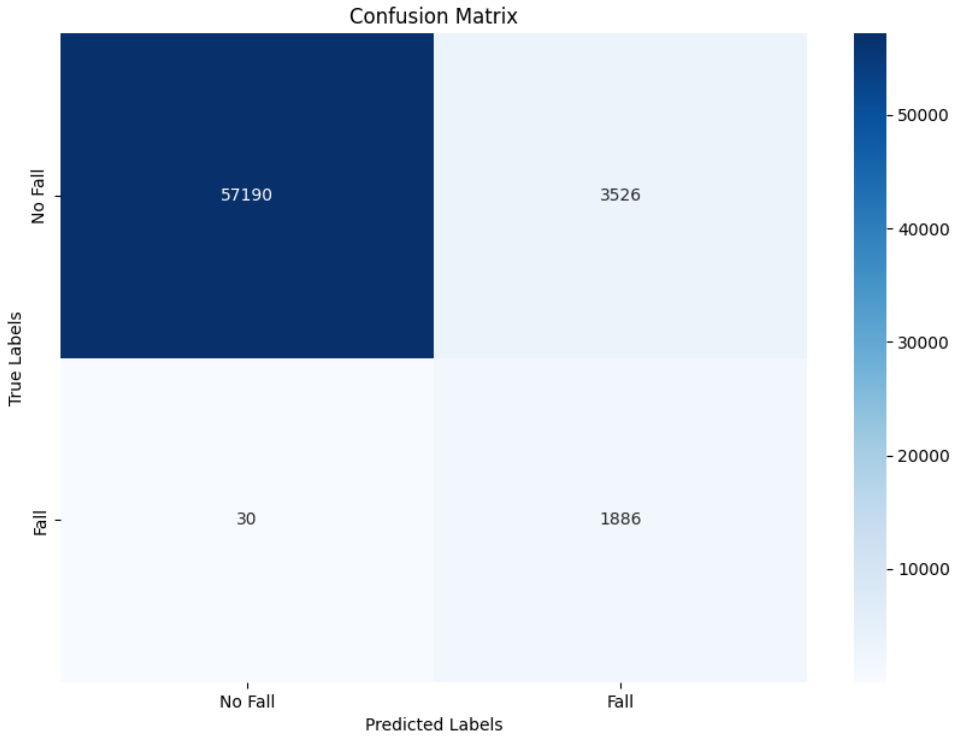

Figure 25:Experiment 1.5 Confusion Matrix

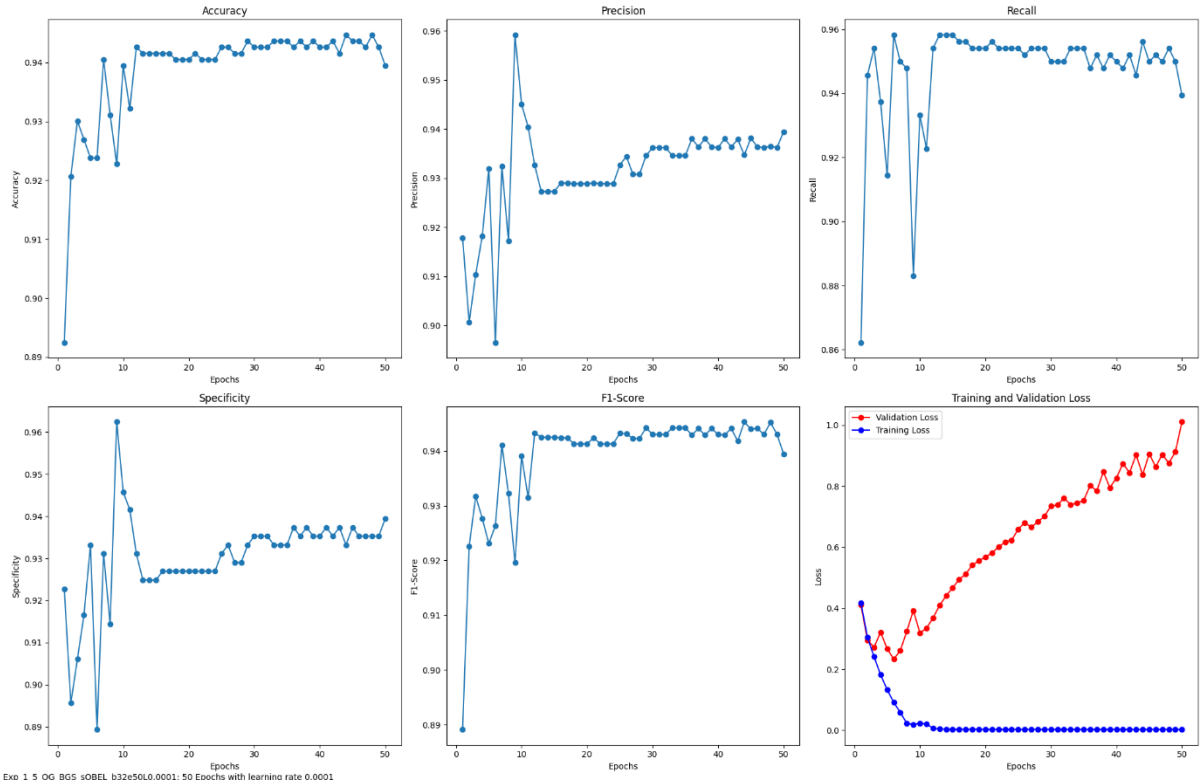

Figure 26:Experiment 1.5 results

5.2.6. Experiment 1.6- Sobel Type II

Observation	Values observed
Size of Unbalanced Dataset	15.40 GB
Size of the Balanced Dataset	940 MB
Total time taken for processing	2Hr 22 Mins
Total time taken for Training	11 Mins 38 Seconds
Test Loss	0.6327
Test Accuracy	0.9419
Test Precision	0.3427
Test Recall	0.9812
Test Specificity	0.9406
Test F1-Score	0.5080

Table 8: Experiment 1.6 Observations

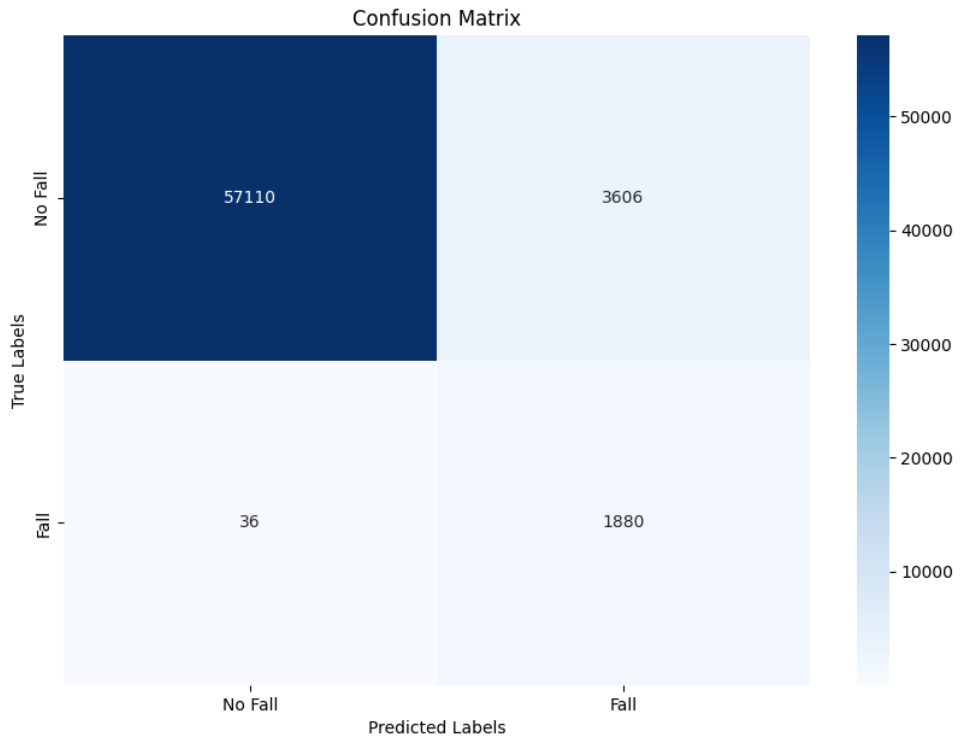

Figure 27: Experiment 1.6 Confusion Matrix

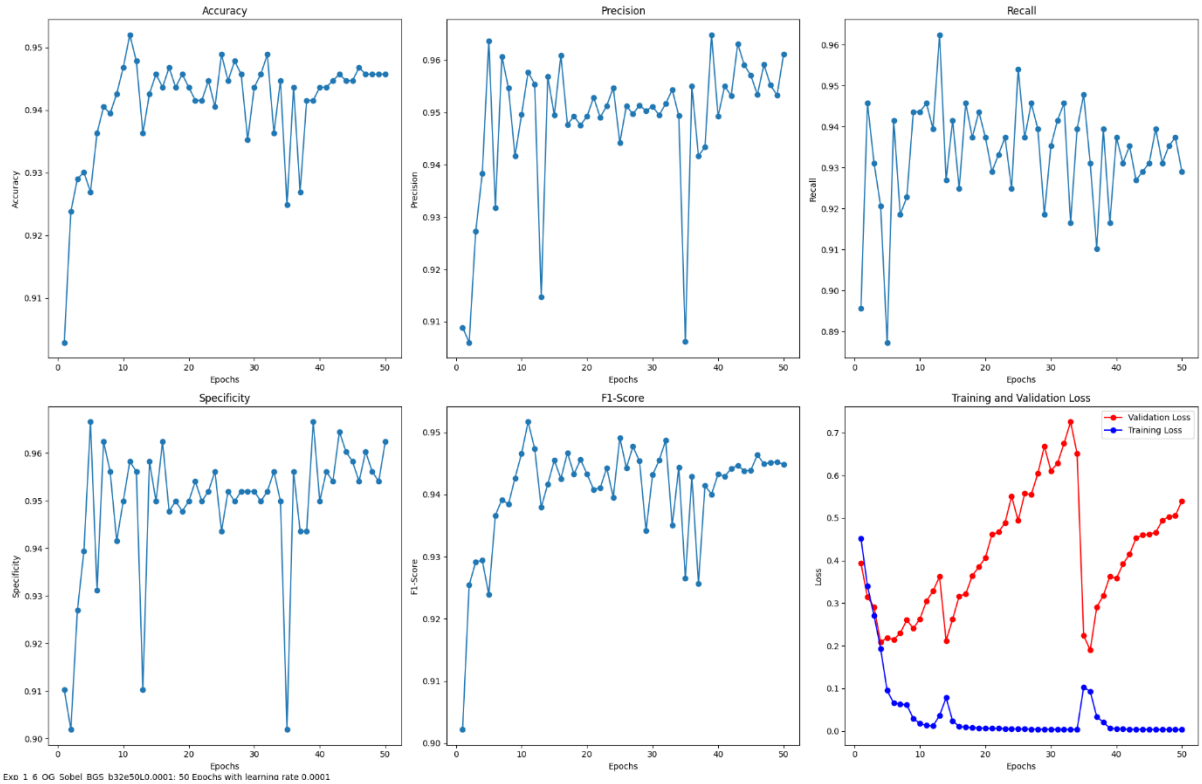

Figure 28:Experiment 1.6 results

5.2.7. Discussion and Evaluation

The depicted plots in Figure 18, Figure 20, Figure 22, Figure 24, Figure 26, and Figure 28 are a collective representation consisting of all metrics relevant to the training performance of all Set-1 experiments.

Throughout the training phase, all models demonstrated promising results. Specifically, all the models achieved a commendable accuracy of 93% or higher as early as the 5th epoch. Furthermore, the models consistently attained a peak accuracy of 95% or above between epochs 10 and 20 for most experiments except for Canny Type II, which could only achieve 94.98% as its highest accuracy. This establishes that all models are adept enough in discerning complex relationships between input features and fall occurrences irrespective of the edge detectors or the type of model used. Since the most relevant metric for our research is recall, a detailed examination revealed a robust performance across all models, with the Laplacian edge detector slightly outperforming other edge detectors, with Laplacian Type I achieving the highest recall of 97.49% in the 5th epoch. On comparison of the specificity Canny Type I performed better on average across different epochs by maintaining a steady value, which is preferable in a real-world application. The specificity of the models exhibits comparatively considerable fluctuations in Type II implying that Type I is better at learning the patterns. The Laplacian models exhibited a contrasting pattern with learning as the models learned the data by the 18th epoch as depicted by the saturation in the plot. All models exhibit remarkable precision and f1 scores throughout the training.

However, it has been observed that all the models showcase decreasing average training loss, but the average validation loss keeps increasing implying that our model is overfitting in

all scenarios. Type II models exhibited a much higher validation loss than Type I. This highlights the challenge of reduced generalisation as the model will perform poorly on unseen data. Also, considering the time taken and size of the pre-processed dataset, both Canny and Laplacian had smaller and similar sizes while the Sobel models were time-consuming to process and resulted in a huge, pre-processed dataset. Both types were exhibiting the same size across different edge detectors implying that there is no impact on time and dataset size for different types of processing.

Experiment	Accuracy	Precision	Recall	Specificity	F1-Score	Loss
1.1 (Canny Type I)	0.9450	0.3564	0.9885	0.9437	0.5293	0.5772
1.2 (Canny Type II)	0.9154	0.2629	0.9781	0.9135	0.4144	0.9357
1.3 (Laplacian Type I)	0.9543	0.3999	0.9854	0.9533	0.5689	0.4837
1.4 (Laplacian Type II)	0.9459	0.3600	0.9880	0.9446	0.5277	0.6158
1.5 (Sobel Type I)	0.9432	0.3485	0.9843	0.9419	0.5147	0.8487
1.6 (Sobel 97Type II)	0.9419	0.3427	0.9812	0.9406	0.5080	0.6327

Table 9: Experiment 1 model evaluation

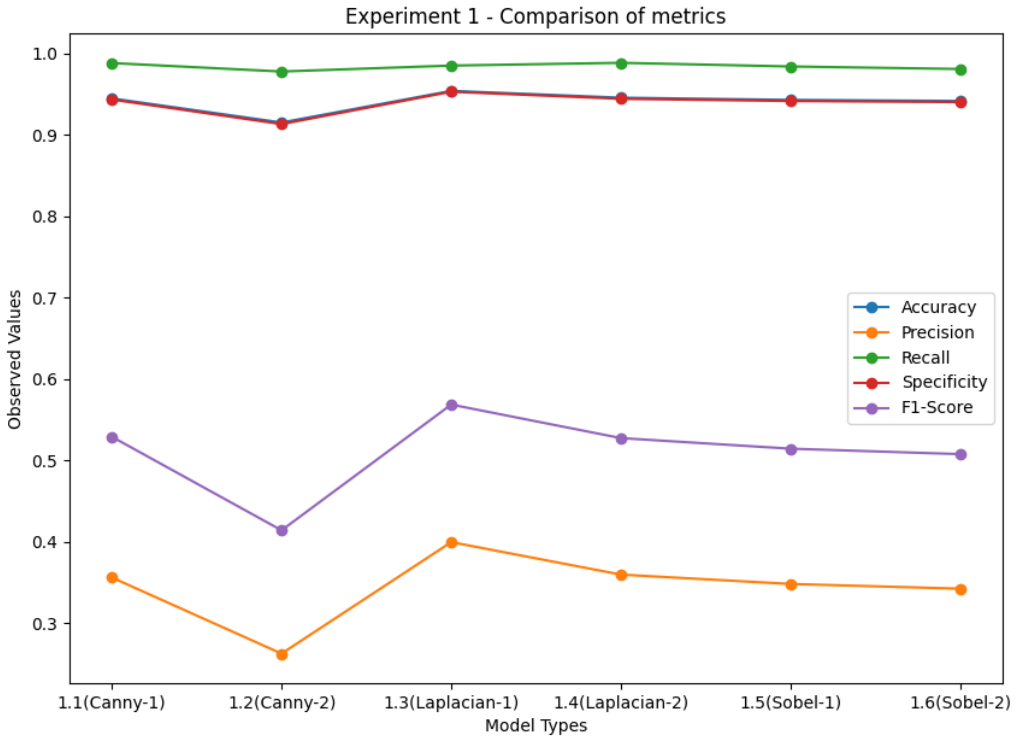

Figure 29: Experiment 1 metrics comparison

Table 9 summarises the results obtained during the model evaluation with Figure 29 providing a visual representation. Notably, the models are exhibiting a consistent performance across different combinations of edge detectors and model types. There is a considerable improvement in precision and recall for the Laplacian Type I model without compromising the accuracy, recall, or specificity of the model. Even though the Canny Type I had demonstrated

a better average metric during the training phase, Laplacian Type I slightly outperforms it during the testing phase.

However, even with the models demonstrating similar performance during the training and testing phase for all metrics, the testing phase showed a considerable drop in precision and F1 Score which needs to be further investigated. Also, Type II models were slightly underperforming compared to their Type I alternatives.

Experiment	True Positives(TP)	True Negatives(TN)	False Positives(FP)	False Negatives(FN)
1.1 (Canny Type I)	1894	57296	3420	22
1.2 (Canny Type II)	1874	55462	5254	42
1.3 (Laplacian Type I)	1888	57883	2833	28
1.4 (Laplacian Type II)	1893	57351	3365	28
1.5 (Sobel Type I)	1886	57190	352	30
1.6 (Sobel 97Type II)	1880	57110	3606	36

Table 10: Experiment 1 Confusion matrix

Table 10 summarises all the confusion matrices. Given our research focuses on recall as the primary metric, a comparison of the False Negatives(FN) and True Positive is important. The lowest False Negative(FN) of 22 was obtained for Canny Type I. The model also obtained the highest True Positive(TP) value of 1894 making it an ideal choice. All Type II models were identified to be comparatively underperforming according to the confusion matrix.

In conclusion, while all the models exhibit commendable performance during training and testing, we notice a slightly better performance from the Type I Model in every phase. Among the edge detectors, the Sobel edge detector will be the least fit for our requirement due to its processing complexity. Laplacian and Canny models were identified to be our optimal choice, with Laplacian Type I demonstrating a commendable edge during the testing phase. However, a thorough analysis of the confusion matrix confirmed that the Canny Type I is the optimal choice, demonstrating optimal trade-offs between accuracy precision, and recall.

5.3. Experiment Set 2

5.3.1. Experiment 2.1

Observation	Values observed
Size of Unbalanced Dataset	1.95 GB
Size of the Balanced Dataset	122 MB
Total time taken for processing	2Hr 16 Mins
Total time taken for Training	9 Mins 11 Seconds
Test Loss	0.5230
Test Accuracy	0.9427
Test Precision	0.3470
Test Recall	0.9890

Test Specificity	0.9413
Test F1-Score	0.5138

Table 11: Experiment 2.1 Observations

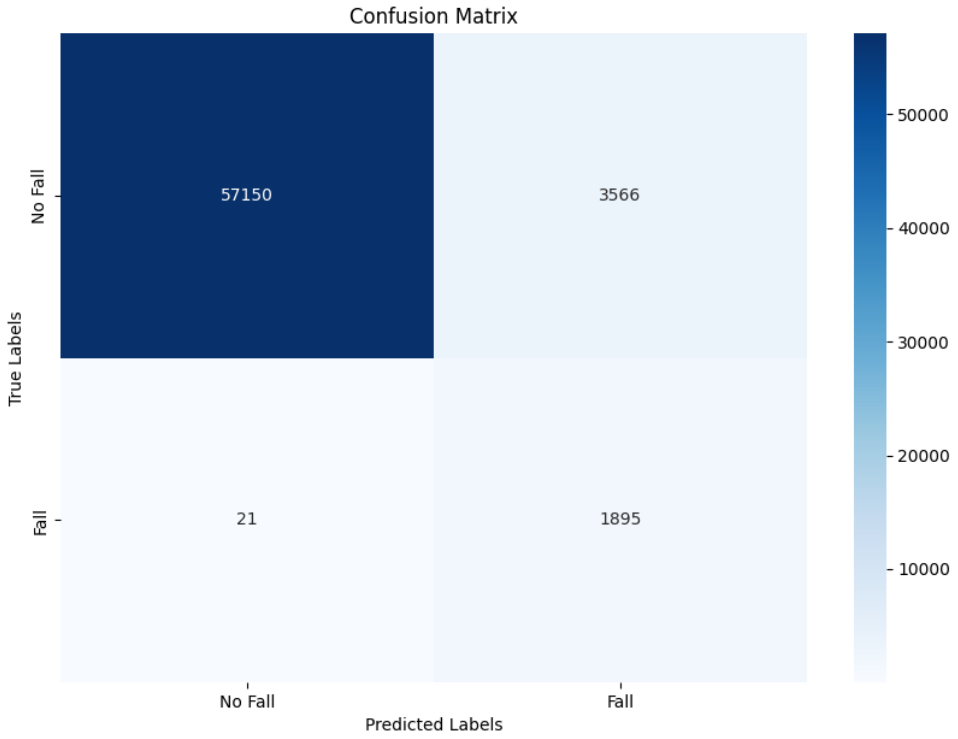

Figure 30: Experiment 2.1 Confusion Matrix

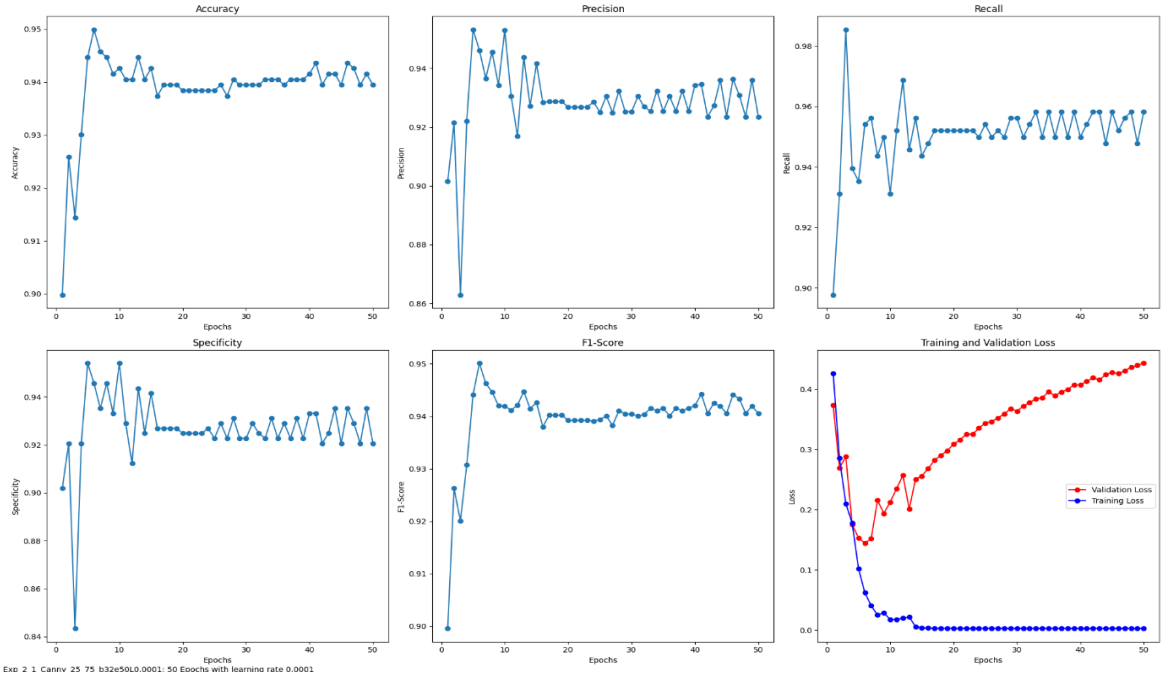

Figure 31:Experiment 2.1 results

5.3.2. Experiment 2.2

Observation	Values observed
Size of Unbalanced Dataset	1.95 GB
Size of the Balanced Dataset	122 MB
Total time taken for processing	2Hr 40 Mins
Total time taken for Training	9 Mins 12 Seconds
Test Loss	0.5556
Test Accuracy	0.9509
Test Precision	0.3829
Test Recall	0.9890
Test Specificity	0.9497
Test F1-Score	0.5521

Table 12: Experiment 2.2 Observations

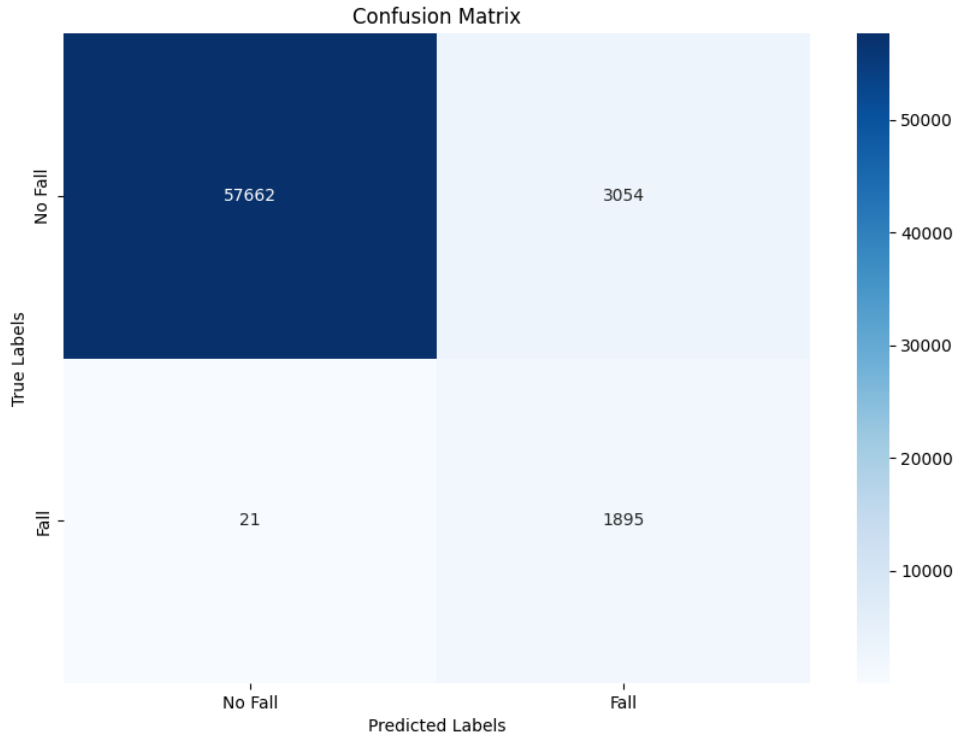

Figure 32: Experiment 2.2 Confusion Matrix

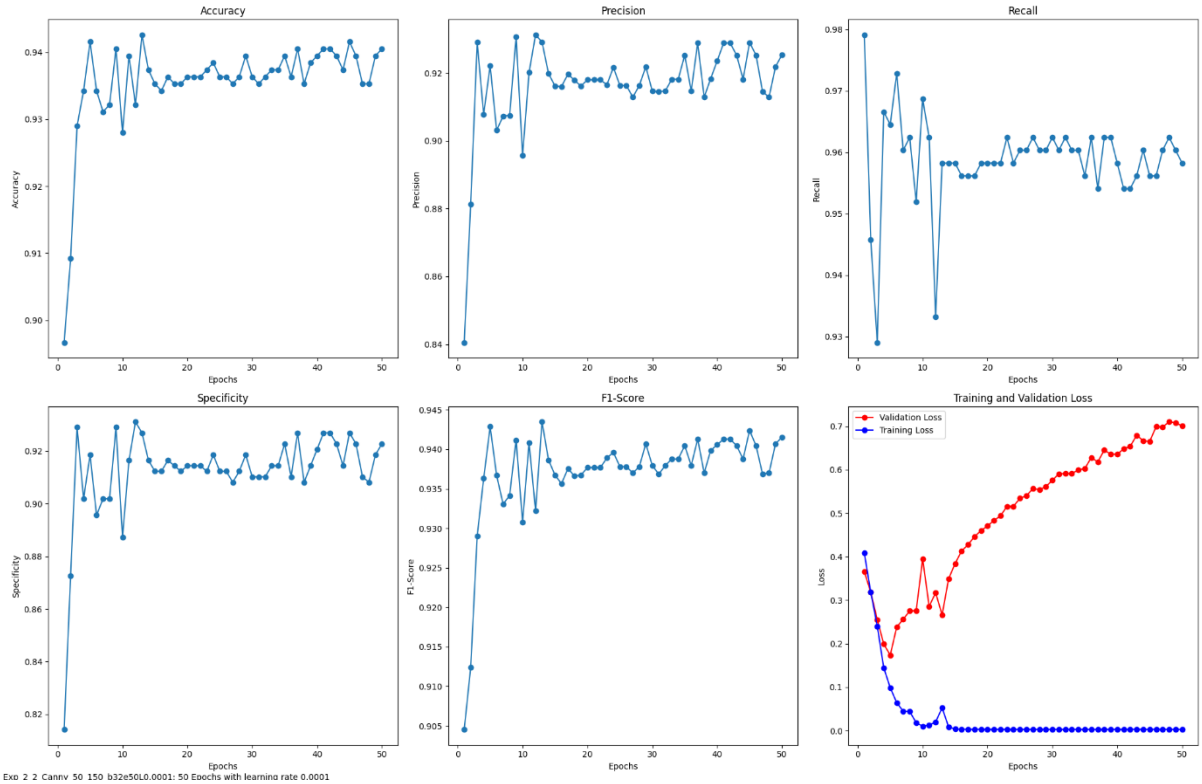

Figure 33:Experiment 2.2 results

5.3.3. Experiment 2.3

Observation	Values observed
Size of Unbalanced Dataset	1.95 GB
Size of the Balanced Dataset	122 MB
Total time taken for processing	2Hr 18 Mins
Total time taken for Training	9 Mins 12 Seconds
Test Loss	0.5689
Test Accuracy	0.9477
Test Precision	0.3676
Test Recall	0.9849
Test Specificity	0.9465
Test F1-Score	0.5354

Table 13: Experiment 2.3 Observations

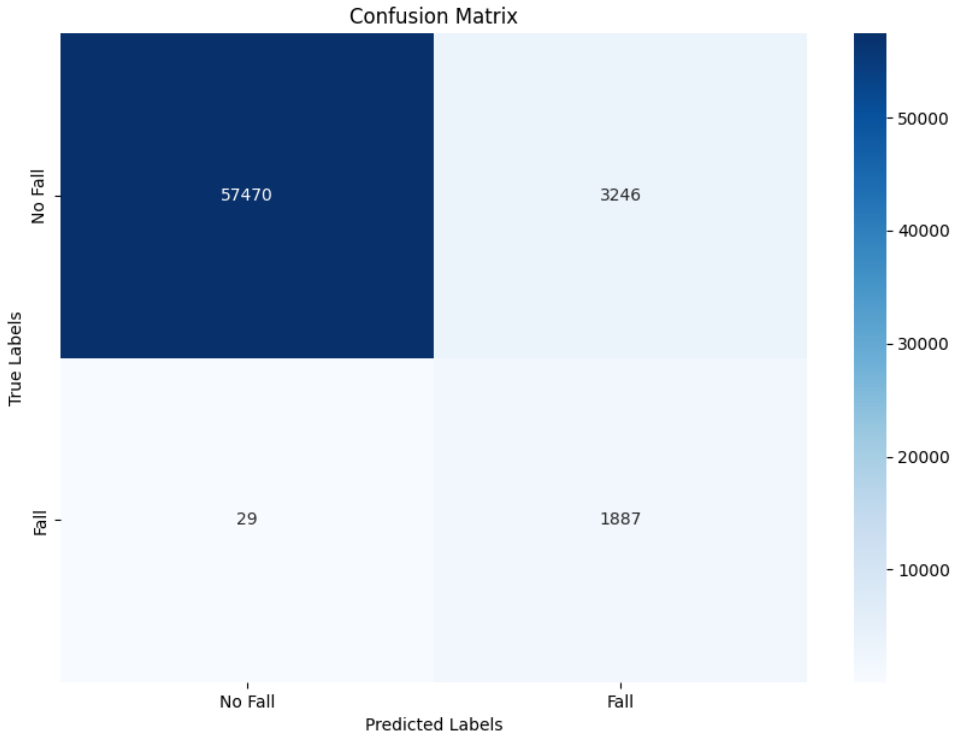

Figure 34: Experiment 2.3 Confusion Matrix

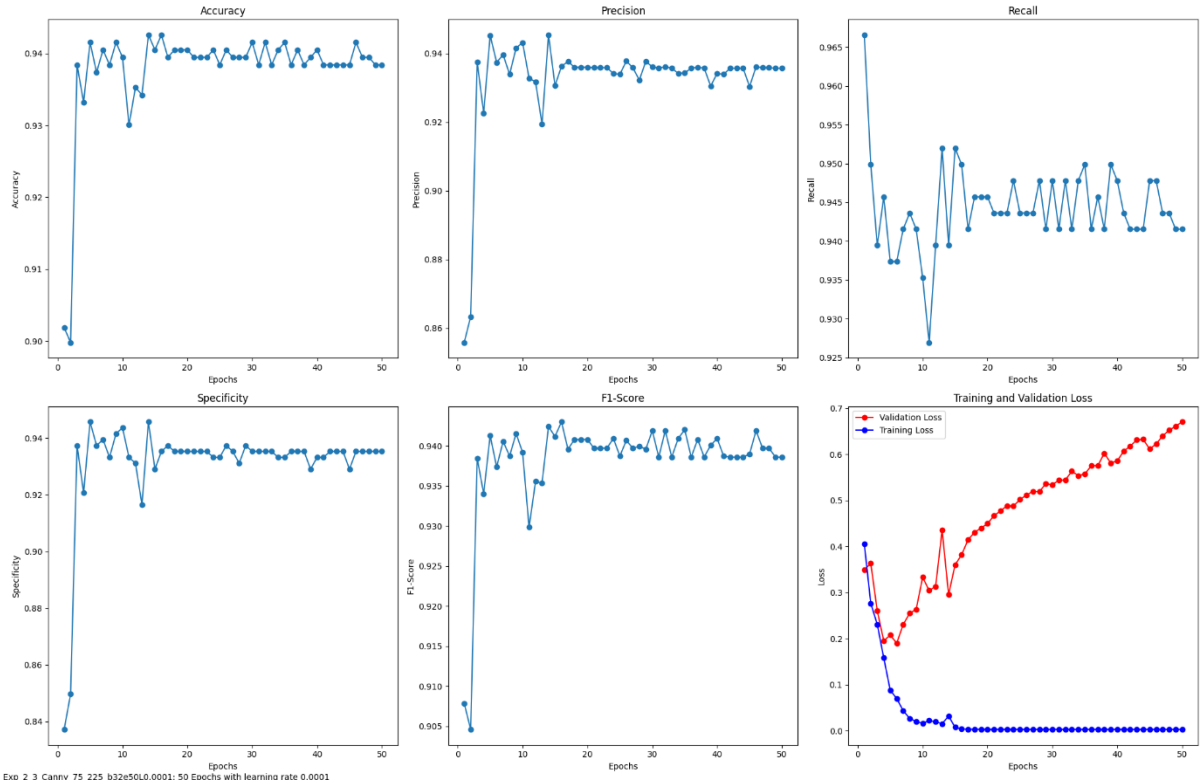

Figure 35:Experiment 2.3 results

5.3.4. Experiment 2.4

Observation	Values observed
Size of Unbalanced Dataset	1.95 GB
Size of the Balanced Dataset	122 MB
Total time taken for processing	2Hr 30 Mins
Total time taken for Training	9 Mins 13 Seconds
Test Loss	0.5514
Test Accuracy	0.9542
Test Precision	0.3994
Test Recall	0.9864
Test Specificity	0.9532
Test F1-Score	0.5686

Table 14: Experiment 2.4 Observations

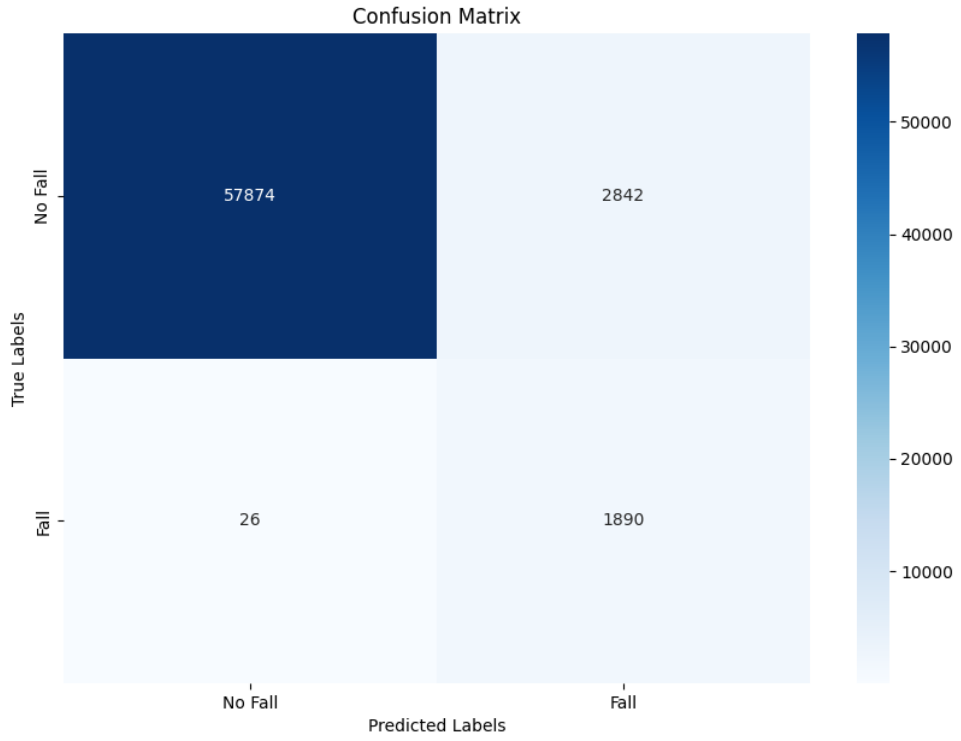

Figure 36: Experiment 2.4 Confusion Matrix

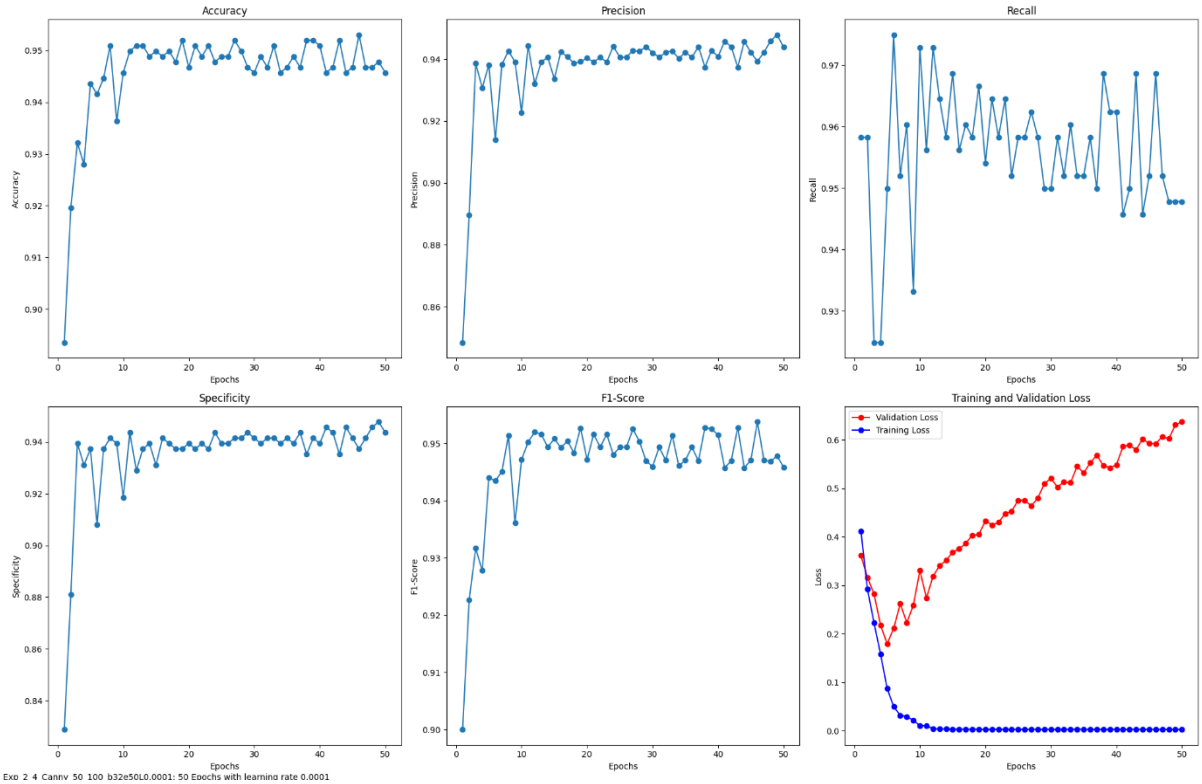

Figure 37:Experiment 2.4 results

5.3.5. Experiment 2.5

Observation	Values observed
Size of Unbalanced Dataset	1.95 GB
Size of the Balanced Dataset	122 MB
Total time taken for processing	2Hr 6 Mins
Total time taken for Training	9 Mins 16 Seconds
Test Loss	0.5352
Test Accuracy	0.9444
Test Precision	0.3539
Test Recall	0.9901
Test Specificity	0.9429
Test F1-Score	0.5214

Table 15: Experiment 2.5 Observations

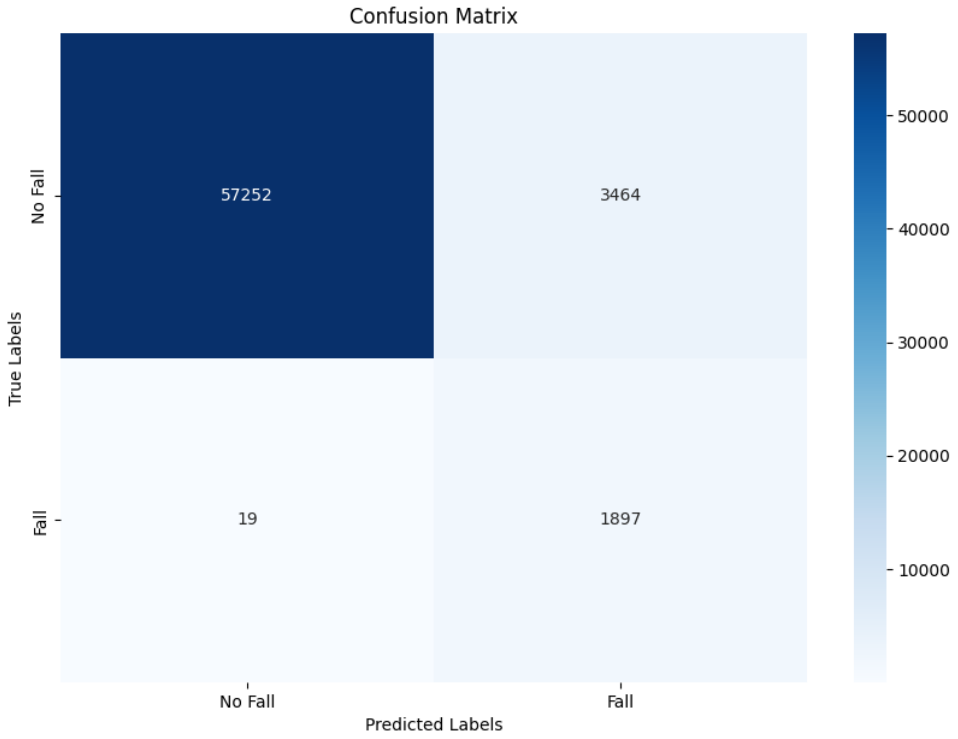

Figure 38: Experiment 2.5 Confusion Matrix

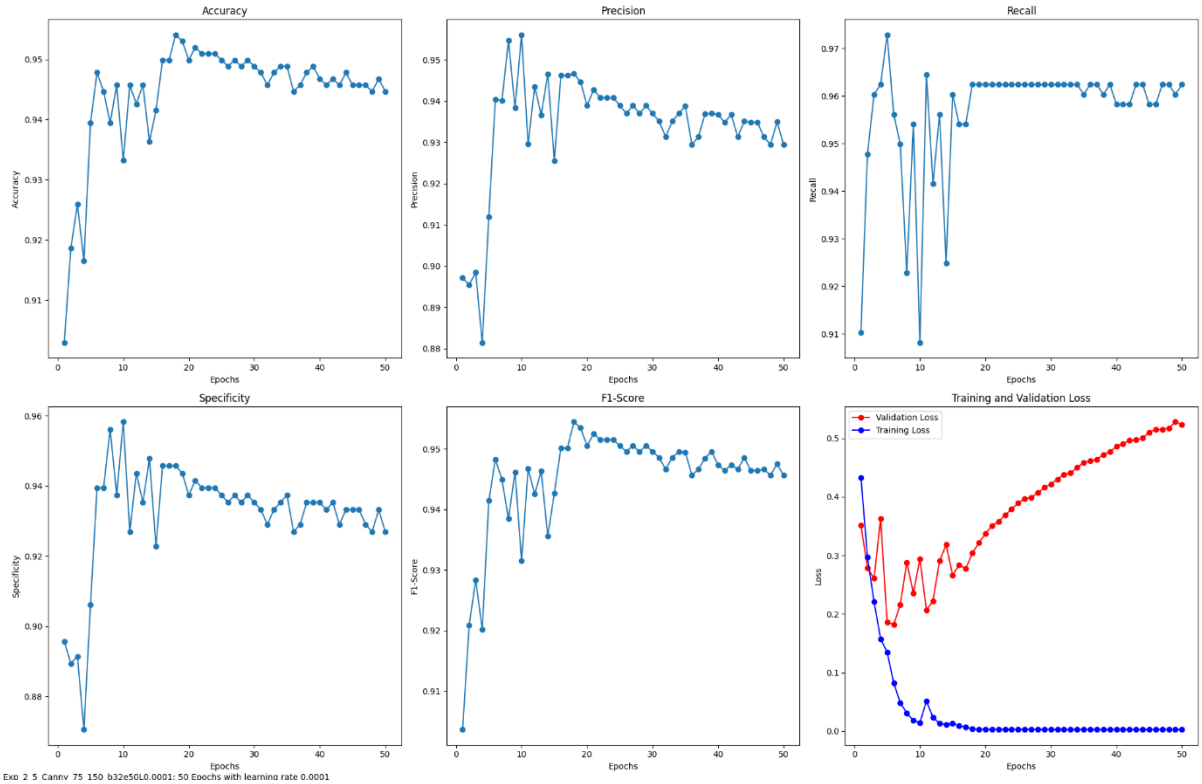

Figure 39:Experiment 2.5 results

5.3.6. Experiment 2.6

Observation	Values observed
Size of Unbalanced Dataset	1.95 GB
Size of the Balanced Dataset	122 MB
Total time taken for processing	2Hr 8 Mins
Total time taken for Training	9 Mins 14 Seconds
Test Loss	0.6075
Test Accuracy	0.9381
Test Precision	0.3297
Test Recall	0.9890
Test Specificity	0.9365
Test F1-Score	0.4945

Table 16: Experiment 2.6 Observations

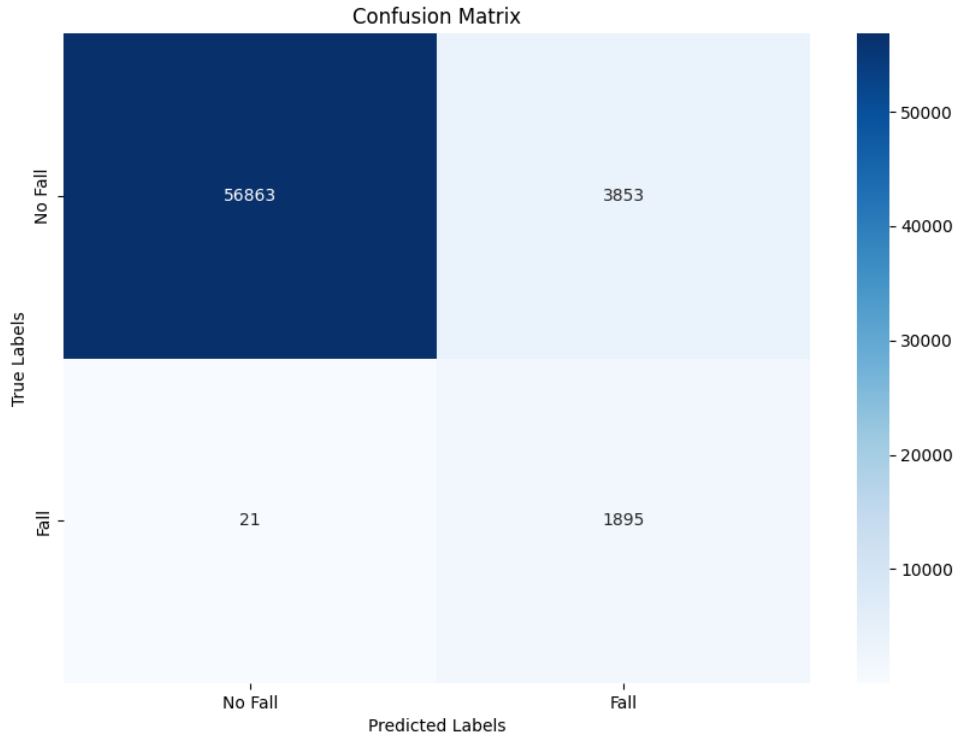

Figure 40: Experiment 2.6 Confusion Matrix

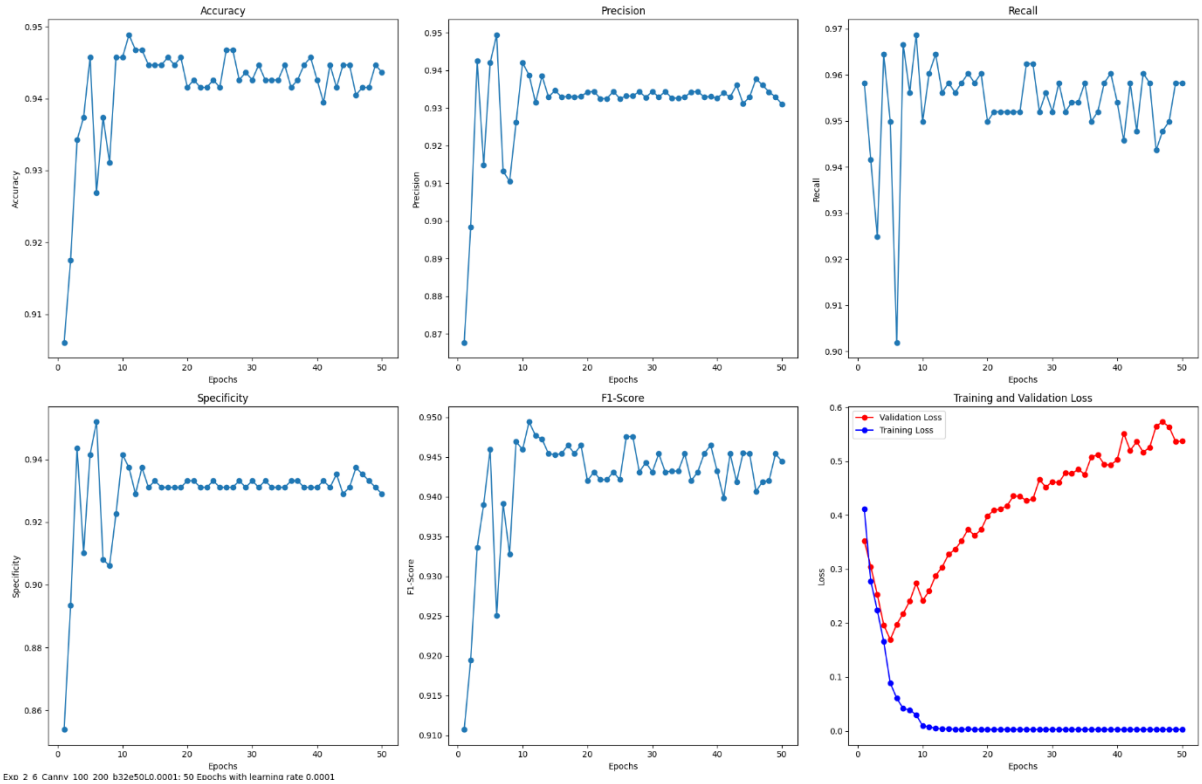

Figure 41:Experiment 2.6 results

5.3.7. Experiment 2.7

Observation	Values observed
Size of Unbalanced Dataset	1.95 GB
Size of the Balanced Dataset	122 MB
Total time taken for processing	2Hr 7 Mins
Total time taken for Training	9 Mins 16 Seconds
Test Loss	0.5862
Test Accuracy	0.9459
Test Precision	0.3599
Test Recall	0.9864
Test Specificity	0.9446
Test F1-Score	0.5273

Table 17: Experiment 2.7 Observations

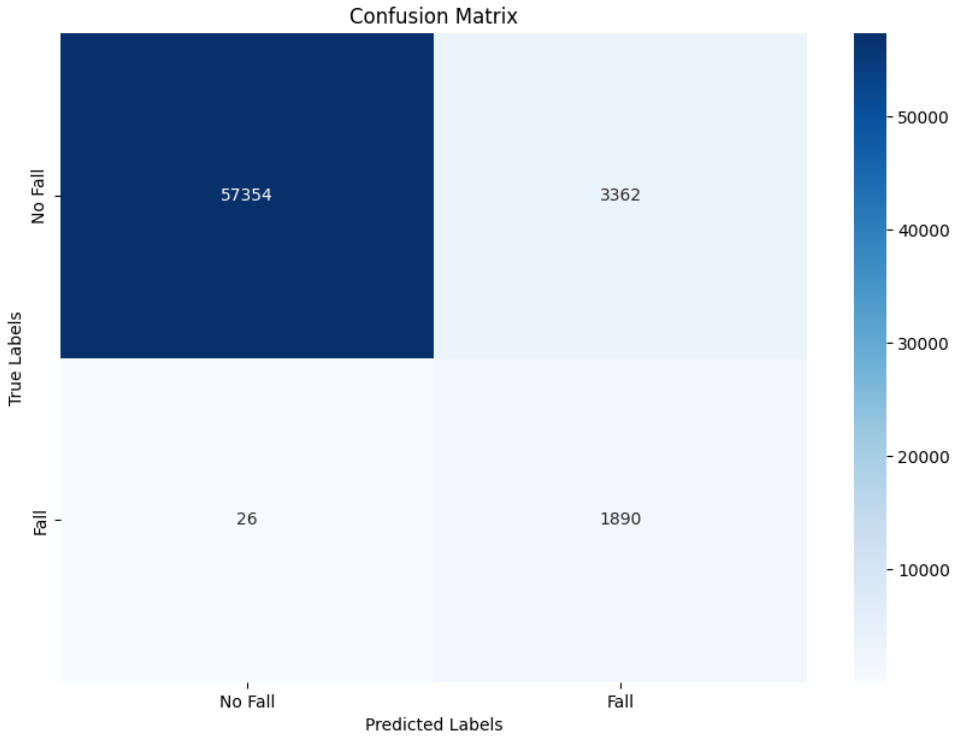

Figure 42: Experiment 2.7 Confusion Matrix

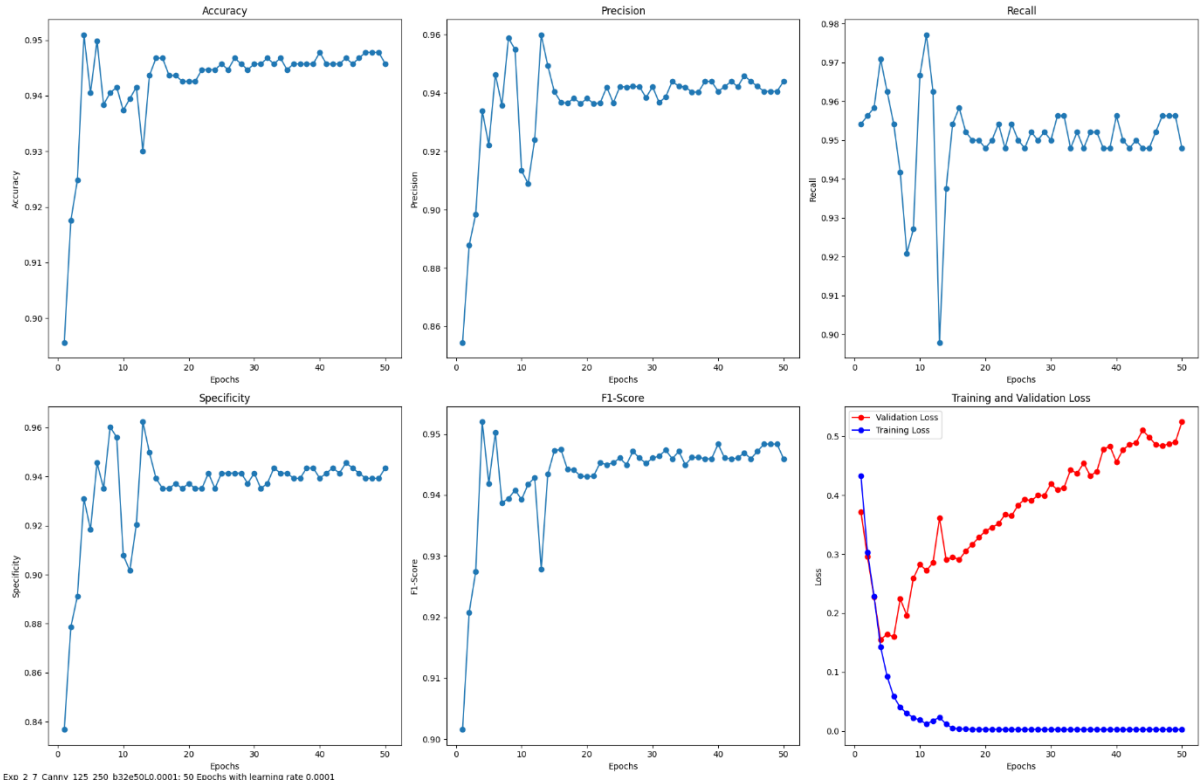

Figure 43:Experiment 2.7 results

5.3.8. Discussion and Evaluation

The depicted plots in Figure 31, Figure 33, Figure 35, Figure 37, Figure 39, Figure 41, and Figure 43 are a collective representation consisting of all metrics relevant to the training phase performance of Set-2 models.

Throughout the training phase, all models demonstrated similar performance, achieving a commendable accuracy of 94% or higher as early as the 5th epoch. Furthermore, all models consistently attained a peak accuracy of 95% or above between the 13th and 18th, except for Model 2.1 (Type I) which achieved a peak of 94.99% by the 6th epoch. Furthermore, the Type I models inhibited comparatively fewer variations in accuracy as the training progressed. A detailed examination of recall revealed an overall robust performance across all models, with the Model 2.1 (Type I) model achieving a peak recall of 98.54%. Although Type I models demonstrated higher peak values, Type II models showed a better average recall performance across the epochs. All models exhibited considerable fluctuations with specificity, except for Model 2.4 (Type I) which showcased a commendably high peak value of 94.78% with minimal fluctuations leading to a better average specificity. It was also observed that all the models except Model 2.2 (Type I), Model 2.4 (Type II), and Model 2.5 (Type II) were quick in learning the data between the 15th and 20th epochs, suggesting that the models' capability to learn has no relation with our experiment parameters. The model also exhibits a remarkable precision and F1 score throughout the training with Model 2.4 (Type II) slightly outperforming all other models with a peak value of 94.78% in the 49th epoch. Notably, Type II models had a slightly better precision performance with minimal fluctuations and a better average. All models took a similar amount of time to process and had almost identical pre-processed dataset sizes, ruling out further analysis into these complexities.

However, similar to our Set-1 experiments, the models were overfitting in all scenarios as observed in the average validation loss plot. Model 2.2 (Type I) had the highest average validation loss across the epochs. Another interesting insight here is the comparatively poor recall performance in Type II models as all the models exhibited considerable fluctuation making this less preferable for a real-world application.

Experiments	Accuracy	Precision	Recall	Specificity	F1-Score	Loss
2.1 (Type I)	0.9427	0.3470	0.9890	0.9413	0.5138	0.5230
2.2 (Type I)	0.9509	0.3829	0.9890	0.9497	0.5521	0.5556
2.3 (Type I)	0.9477	0.3676	0.9849	0.9465	0.5354	0.5689
2.4 (Type II)	0.9542	0.3994	0.9864	0.9532	0.568	0.5514
2.5 (Type II)	0.9444	0.3539	0.9901	0.9429	0.5214	0.5352
2.6 (Type II)	0.9381	0.3297	0.9890	0.9365	0.4945	0.6075
2.7 (Type II)	0.9459	0.3599	0.9864	0.9446	0.5273	0.5862

*Table 18: Experiment 2 Model Evaluation *(Type I - Ratio 1:3, Type II - Ratio 1:2)*

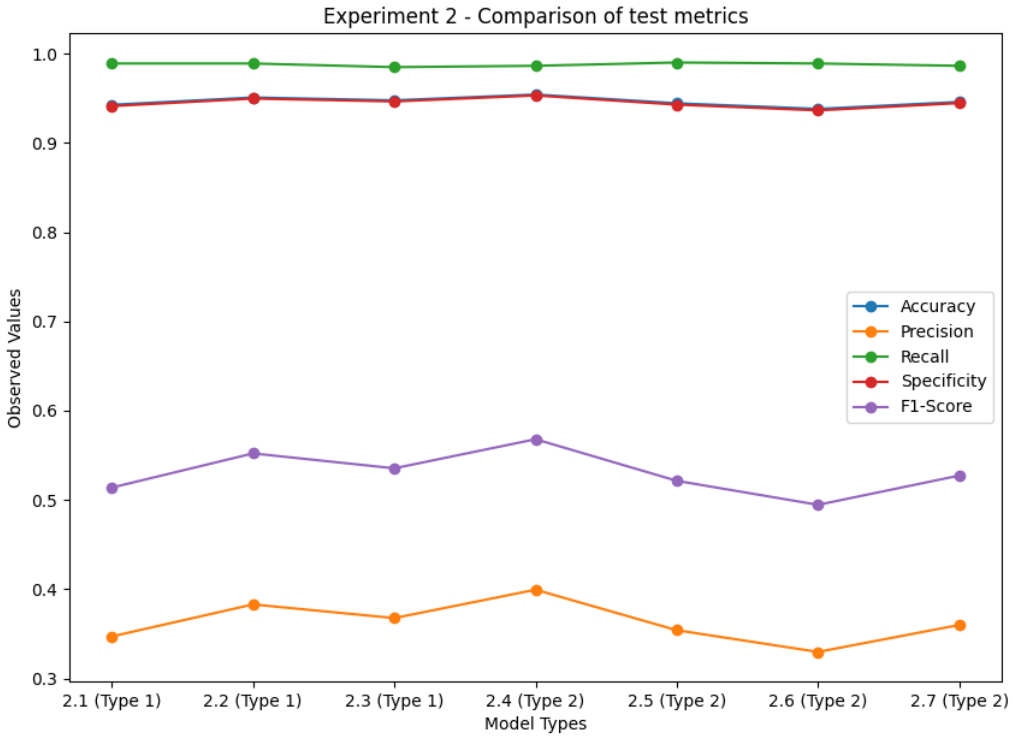

Figure 44: Experiment 2 metrics comparison

Table 18 summarises the testing phase results obtained during the model evaluation with Figure 44 providing a visual representation. Notably, the models are exhibiting a consistent performance across different model parameters for the testing phase as well. There is a distinct improvement in precision and F1 score in Model 2.4 (Type II). All the other metrics are the same across the models indicating that the optimal combination of performance metrics was for Model 2.4 (Type II). However, similar to experiment 1, there was a considerable drop in precision and f1 Score during the testing which needs to be further investigated.

Experiments	True Positives(TP)	True Negatives(TN)	False Positives(FP)	False Negatives(FN)
2.1 (Type I)	1895	57150	3566	21
2.2 (Type I)	1895	57662	3054	21
2.3 (Type I)	1887	57470	3246	29
2.4 (Type II)	1890	57874	2842	26
2.5 (Type II)	1897	57252	3464	19
2.6 (Type II)	1895	56863	3853	21
2.7 (Type II)	1890	57354	3362	26

Table 19: Experiment 2 Confusion matrix

Table 19 summarises the confusion matrix from each experiment. A comparison of the False Negatives(FN) reveals that the lowest False Negative(FN) is obtained for Model 2.5, and the same model also exhibited the highest True Positive (TP) value of 1897.

In conclusion, while all the models exhibited commendable performance during the training and testing phase, however, no distinct superiority was observed among the models. The performance metrics show that the Type II models exhibit some advantages, but they cannot be considered substantial. The testing phase revealed Model 2.4(Type II) with a 1:2 ratio, a lower threshold of 50, and an upper threshold of 100 as a potential optimal choice. However, since the difference is marginal, it's reasonable to conclude that the choice of parameters does not yield a significant impact on the overall model performance.

5.4. Experiment Set 3

5.4.1. Experiment 3.1 – Resolution 51x38

Observation	Values observed
Size of Unbalanced Dataset	2.14 GB
Size of the Balanced Dataset	134 MB
Total time taken for processing	2Hr 40 Mins
Total time taken for Training	9 Mins 18 Seconds
Test Loss	0.2577
Test Accuracy	0.9494
Test Precision	0.3752
Test Recall	0.9828
Test Specificity	0.9483
Test F1-Score	0.5640

Table 20: Experiment 3.1 Observations

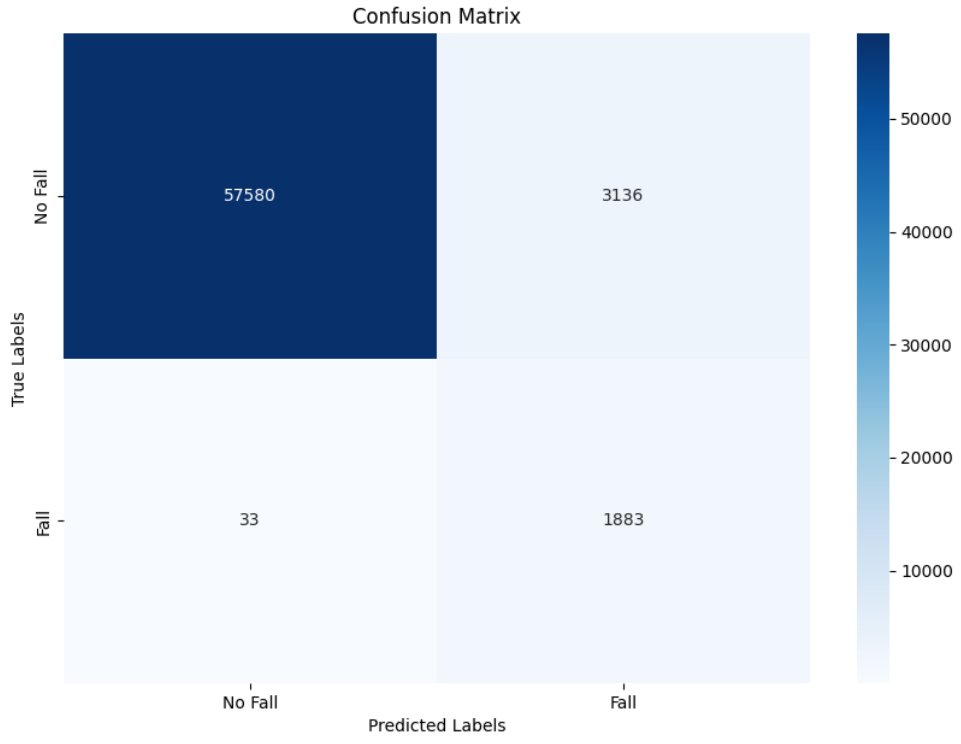

Figure 45: Experiment 3.1 Confusion Matrix

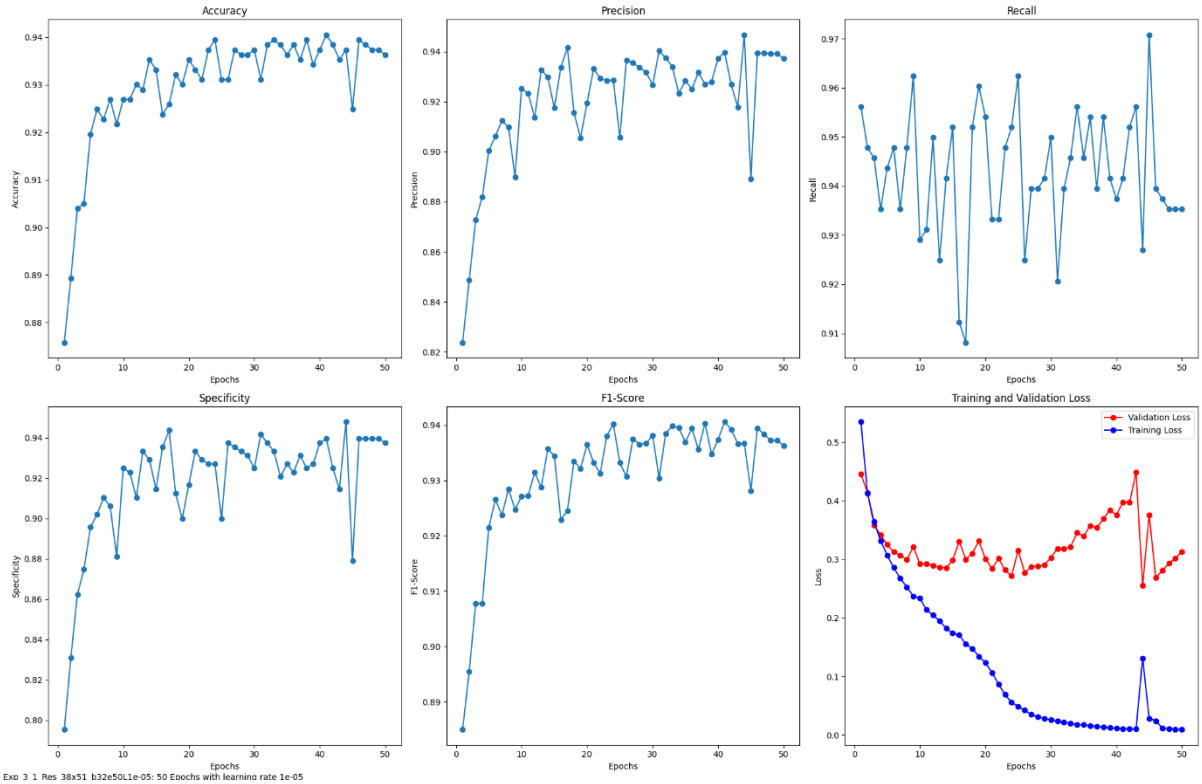

Figure 46: Experiment 3.1 results

5.4.2. Experiment 3.2 – Resolution 76x57

Observation	Values observed
Size of Unbalanced Dataset	4.53 GB
Size of the Balanced Dataset	270 MB
Total time taken for processing	2Hr 35 Mins
Total time taken for Training	18 Mins 51 Seconds
Test Loss	0.2968
Test Accuracy	0.9567
Test Precision	0.4128
Test Recall	0.9870
Test Specificity	0.9557
Test F1-Score	0.5821

Table 21: Experiment 3.2 Observations

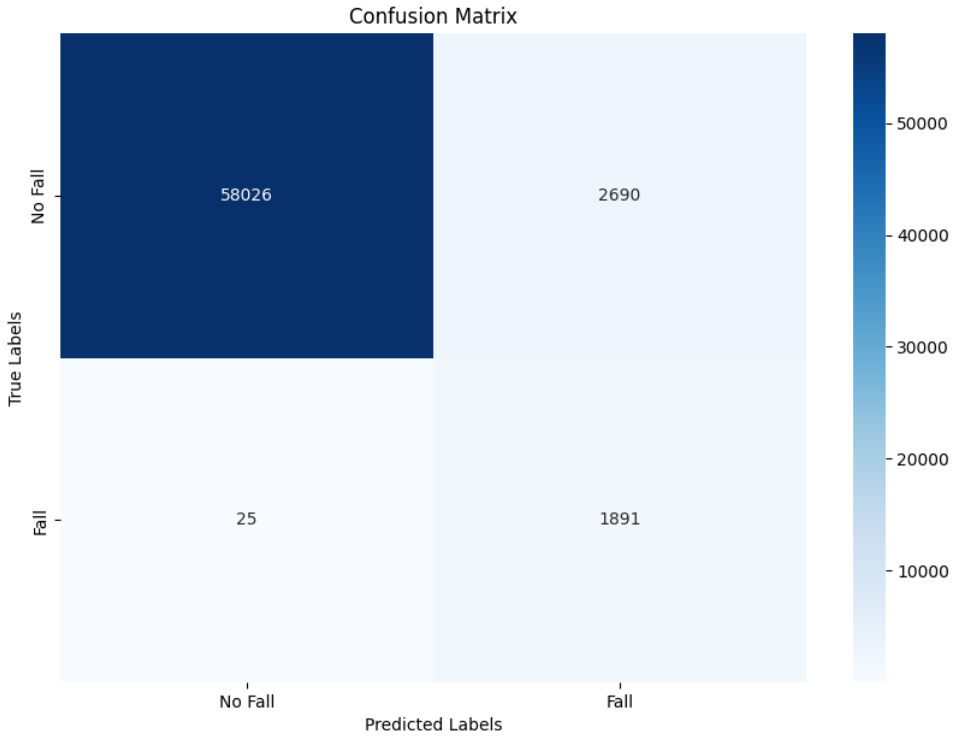

Figure 47:Experiment 3.2 Confusion Matrix

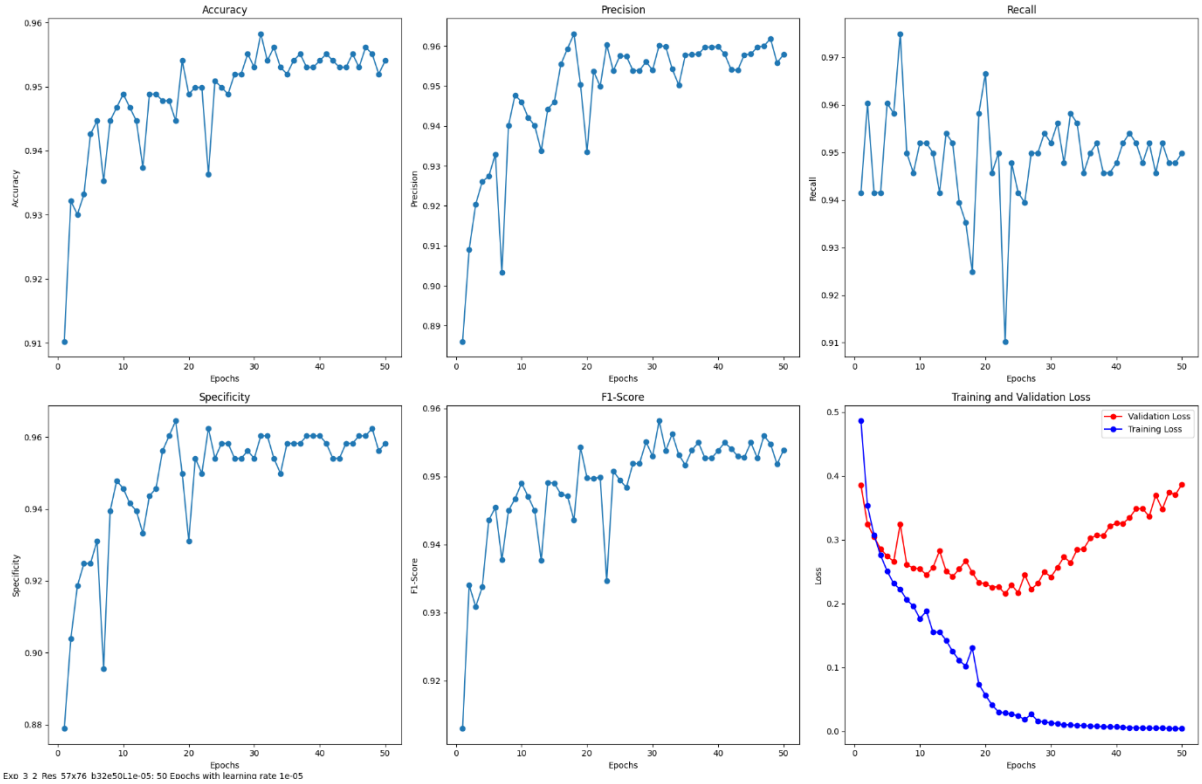

Figure 48: Experiment 3.2 Results

5.4.3. Experiment 3.3 – Resolution 102x76

Observation	Values observed
Size of Unbalanced Dataset	7.88 GB
Size of the Balanced Dataset	483 MB
Total time taken for processing	2Hr 30 Mins
Total time taken for Training	31 Mins 5 Seconds
Test Loss	0.3674
Test Accuracy	0.9555
Test Precision	0.4061
Test Recall	0.9864
Test Specificity	0.9545
Test F1-Score	0.5753

Table 22: Experiment 3.3 Observations

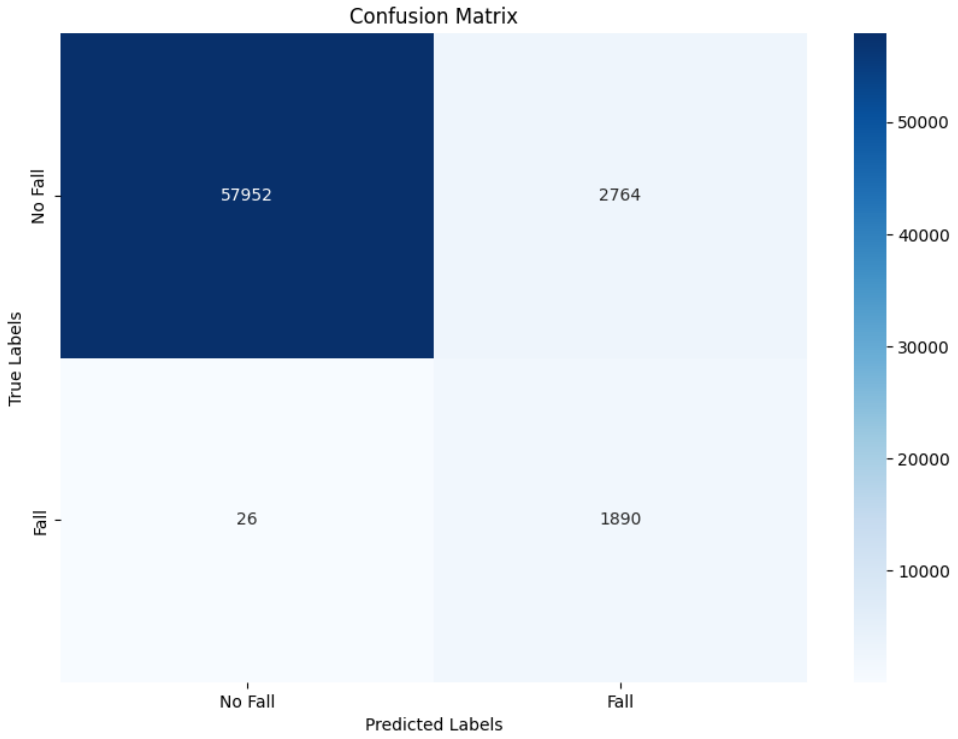

Figure 49: Experiment 3.3 Confusion Matrix

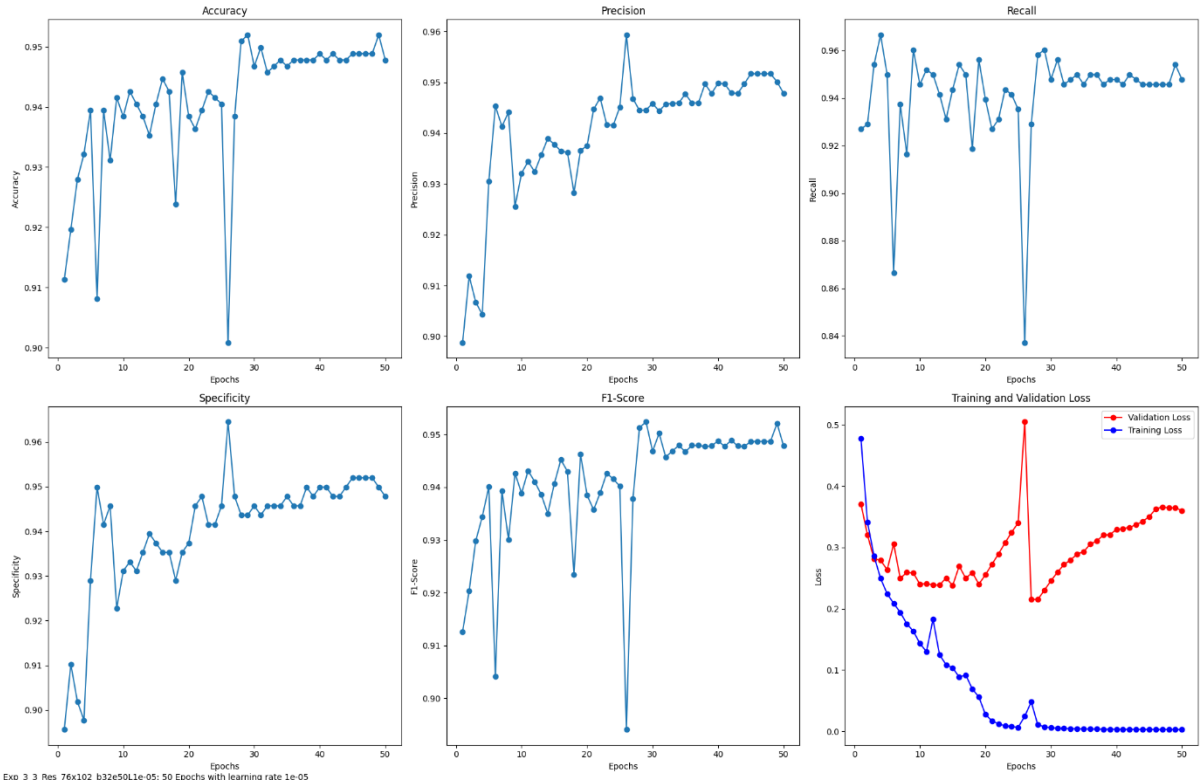

Figure 50: Experiment 3.3 Results

5.4.4. Experiment 3.4 – Resolution 128x95

Observation	Values observed
Size of Unbalanced Dataset	12.1 GB
Size of the Balanced Dataset	763 MB
Total time taken for processing	2Hr 35 Mins
Total time taken for Training	3Hr 21 Mins 51 Seconds
Test Loss	0.631
Test Accuracy	0.9520
Test Precision	0.3878
Test Recall	0.9854
Test Specificity	0.9509
Test F1-Score	0.5565

Table 23: Experiment 3.4 Observations

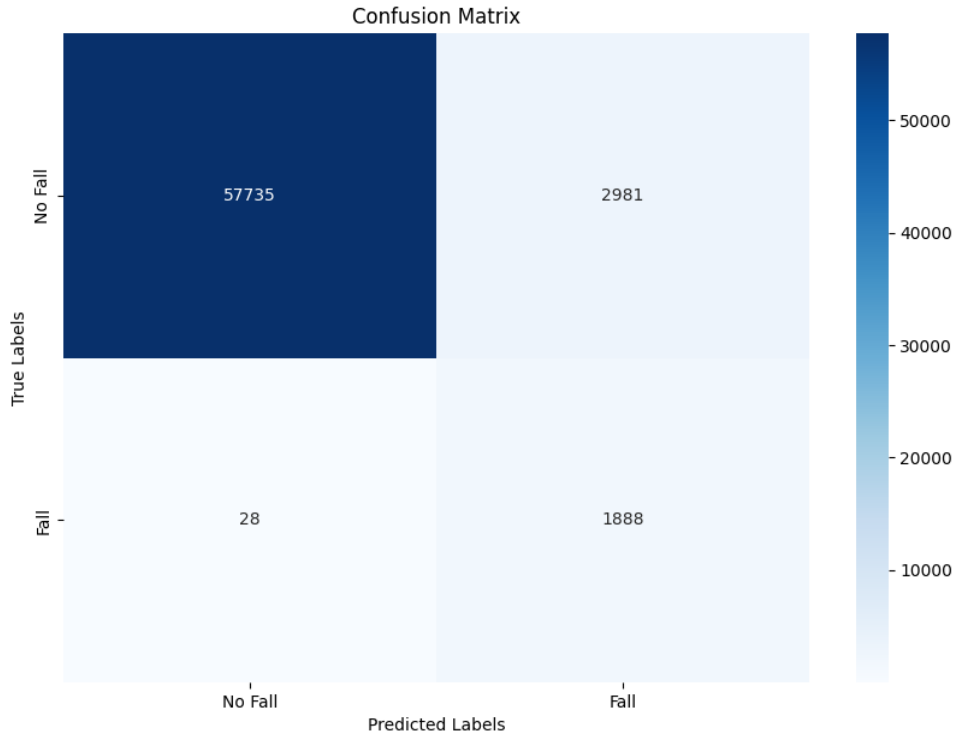

Figure 51:Experiment 3.4 Confusion Matrix

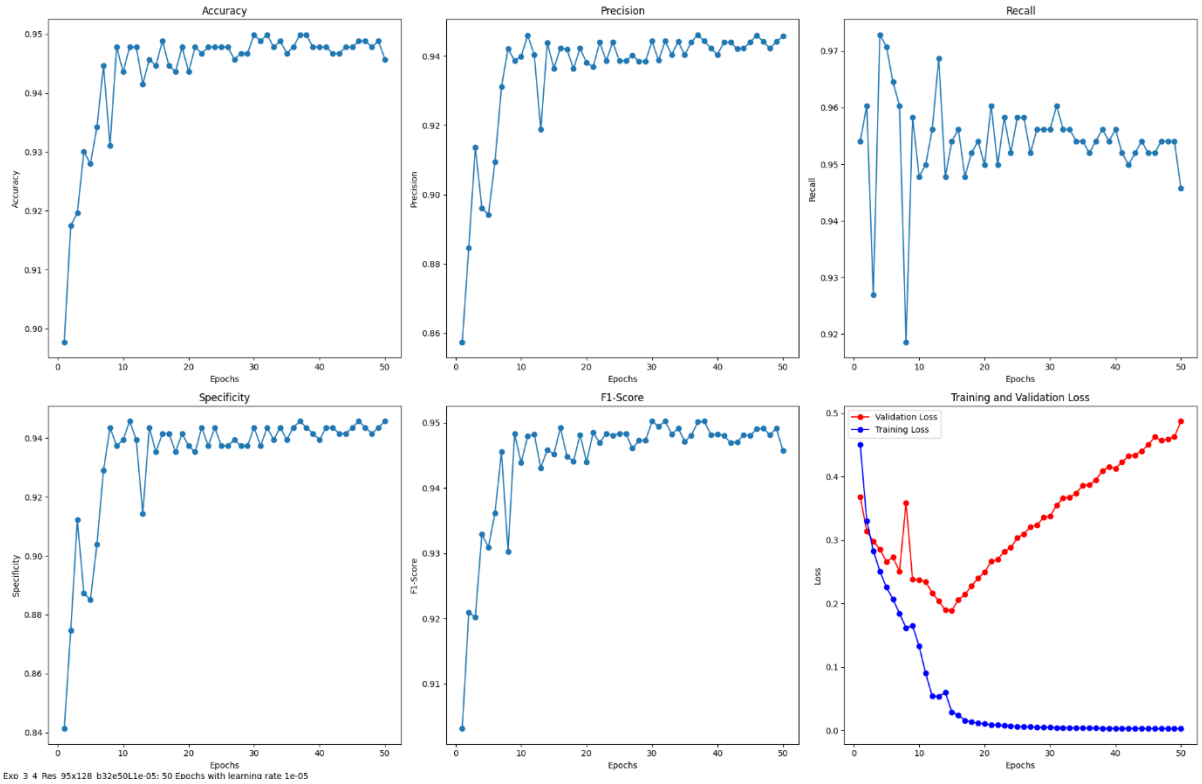

Figure 52: Experiment 3.4 Results

5.4.5. Experiment 3.5 – Resolution 153x114

Observation	Values observed
Size of Unbalanced Dataset	12.1 GB
Size of the Balanced Dataset	1.05 GB
Total time taken for processing	3Hr 40 Mins
Total time taken for Training	3Hr 40Mins
Test Loss	0.4569
Test Accuracy	0.9506
Test Precision	0.3814
Test Recall	0.9916
Test Specificity	0.9493
Test F1-Score	0.5510

Table 24: Experiment 3.5 Observations

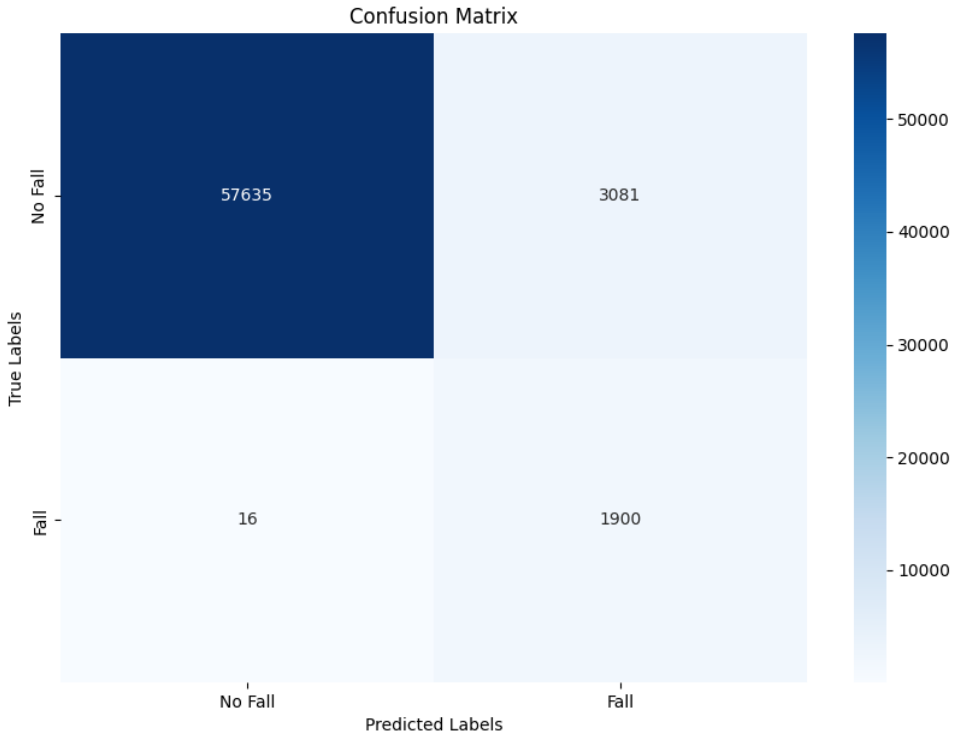

Figure 53:Experiment 3.5 Confusion Matrix

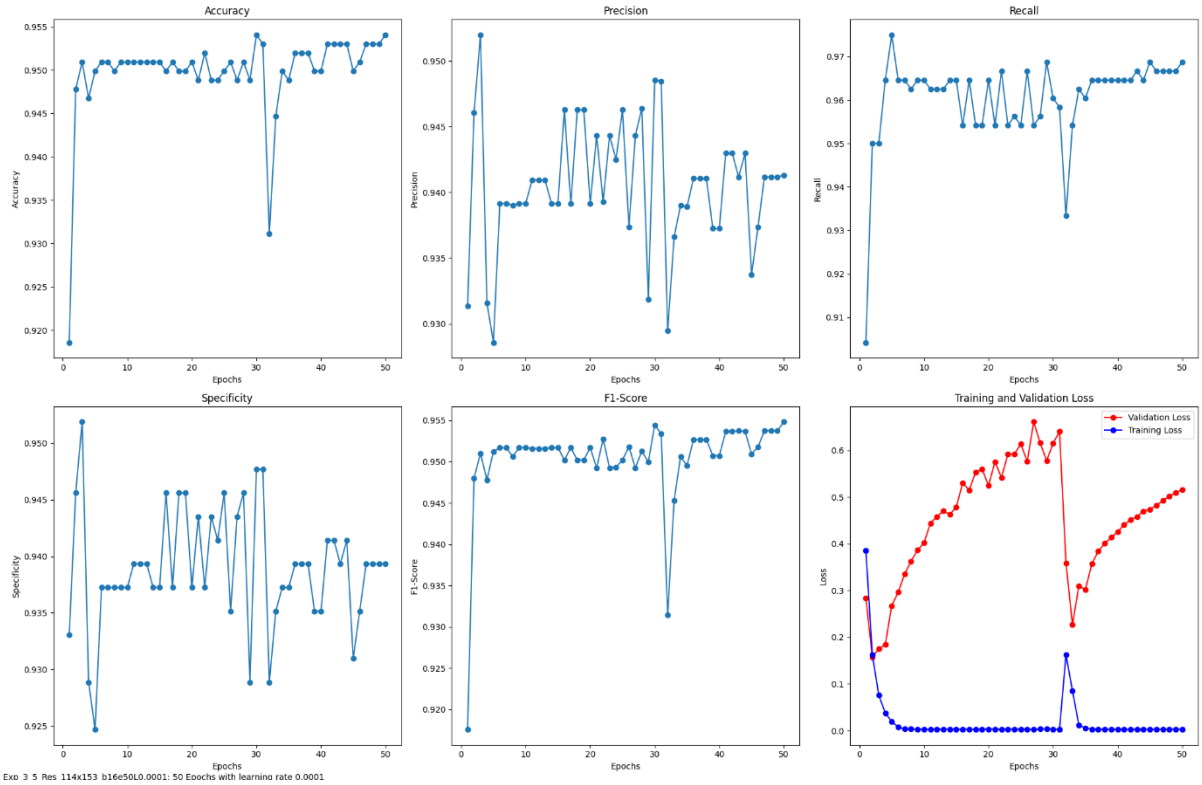

Figure 54:Experiment 3.5 results

5.4.6. Discussion and Evaluation

The depicted plots in Figure 46, Figure 48, Figure 50, Figure 52, and Figure 54 are a representation of training metrics of Set-3 experiments. Throughout the training phase, all models demonstrated promising results, specifically achieving a commendable accuracy of 93% or higher by the 5th epoch. All models consistently attained a peak accuracy of 95% or above around the 30th epoch, showcasing their adeptness in performance. A detailed examination of the recall metric revealed a robust performance across all models, indicating that only a few numbers of falls will go undetected by our model. The specificity of the model exhibits considerable fluctuation in the initial three experiments but stabilises in the last two experiments with the highest resolution. This indicated the model's capability of faster learning and adaptation with increasing resolution. Furthermore, all models exhibited remarkable precision and F1 score throughout the training.

However, similar to our first two sets of experiments, an increasing average validation loss with a declining training loss implies that the models are overfitting in all scenarios. Another interesting insight here is the time taken and size of the pre-processed data. The increasing resolution leads to an increase in both time and size, and it cannot be considered a worthy trade-off as our research aims to improve both performance and time complexity.

Resolution	Accuracy	Precision	Recall	Specificity	F1-Score	Loss
51x38	0.9494	0.3752	0.9828	0.9483	0.5430	0.2577
76x57	0.9567	0.4128	0.9870	0.9557	0.5821	0.2968
102x76	0.9555	0.4061	0.9564	0.9545	0.5753	0.3674
128x95	0.9520	0.3878	0.9854	0.9509	0.5565	0.6312
153x114	0.9506	0.3814	0.9916	0.9493	0.5510	0.4569

Table 25: Experiment 3 model evaluation

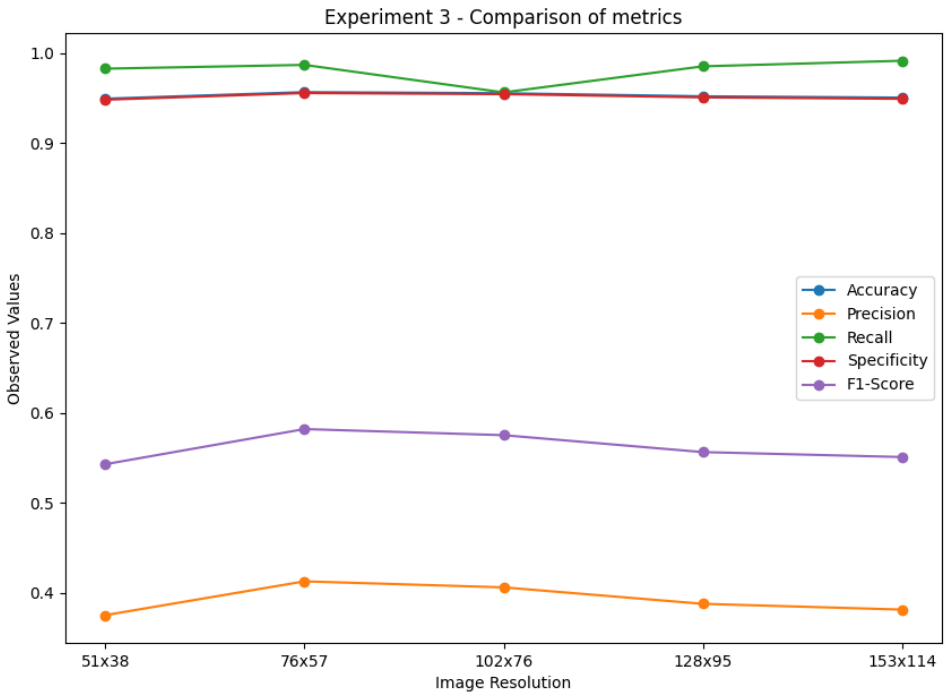

Figure 55:Experiment Set 3 Metrics comparison.

Table 25 summarises the results obtained during the model evaluation with Figure 55 providing a visual representation. Notably, the models are exhibiting a consistent testing performance across different resolutions. There is a distinct improvement in precision and F1 score at a resolution of 76x57 without compromising the accuracy, recall, or specificity of the model. This indicates that the optimal combination of performance metrics was identified to be with this resolution.

However, even with a similar testing and training performance in almost all metrics, there was a considerable drop in precision and F1 Score during the testing which needs to be further investigated.

Resolution	True Positives	True Negatives	False Positives	False Negatives
51x38	1883	57580	3136	33
76x57	1891	58026	2690	25
102x76	1890	57952	2764	26
128x95	1888	57735	2981	28
153x114	1900	57635	3081	16

Table 26: Experiment 3 Confusion Matrix.

Table 26 summarises the confusion matrix from each experiment. The lowest False Negative(FN) was obtained at the highest resolution, however, this brings in further processing and time complexity to the model. In contrast, our experiment showcased an optimal FN value of 25 for the resolution 76x57, suggesting it as the ideal choice for preprocessing.

In conclusion, while all the models exhibit commendable performance during training and testing, a trade-off in terms of False Negatives and preprocessing complexity should be carefully considered. The resolution of 76x57 emerges as an optimal choice, demonstrating optimal trade-offs between accuracy, precision, and recall.

5.5.Experiment 4 – Contour Detection Model

There was only one experiment performed in this analysis.

Observation	Values observed
Size of Unbalanced Dataset	12.1 GB
Size of the Balanced Dataset	1.05 GB
Total time taken for processing	3Hr 40 Mins
Total time taken for Training	1Hr 1 Min 20Sec
Test Loss	0.3821
Test Accuracy	0.9638
Test Precision	0.4579
Test Recall	0.9896
Test Specificity	0.9630
Test F1-Score	0.6261

Table 27: Experiment 4 Observations

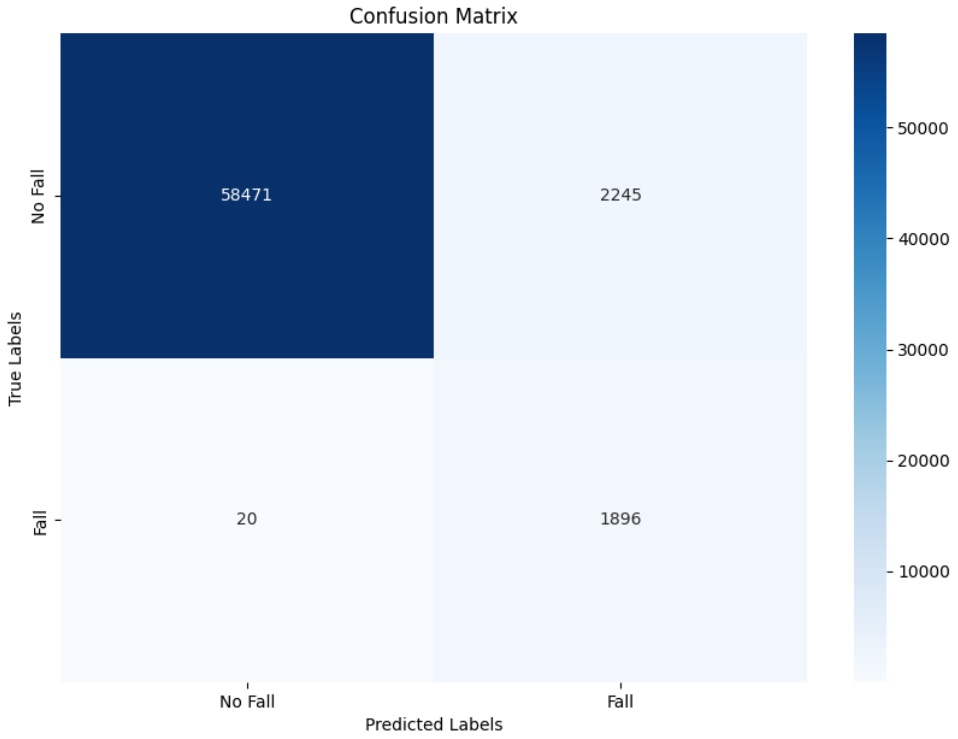

Figure 56:Experiment 4 Confusion Matrix

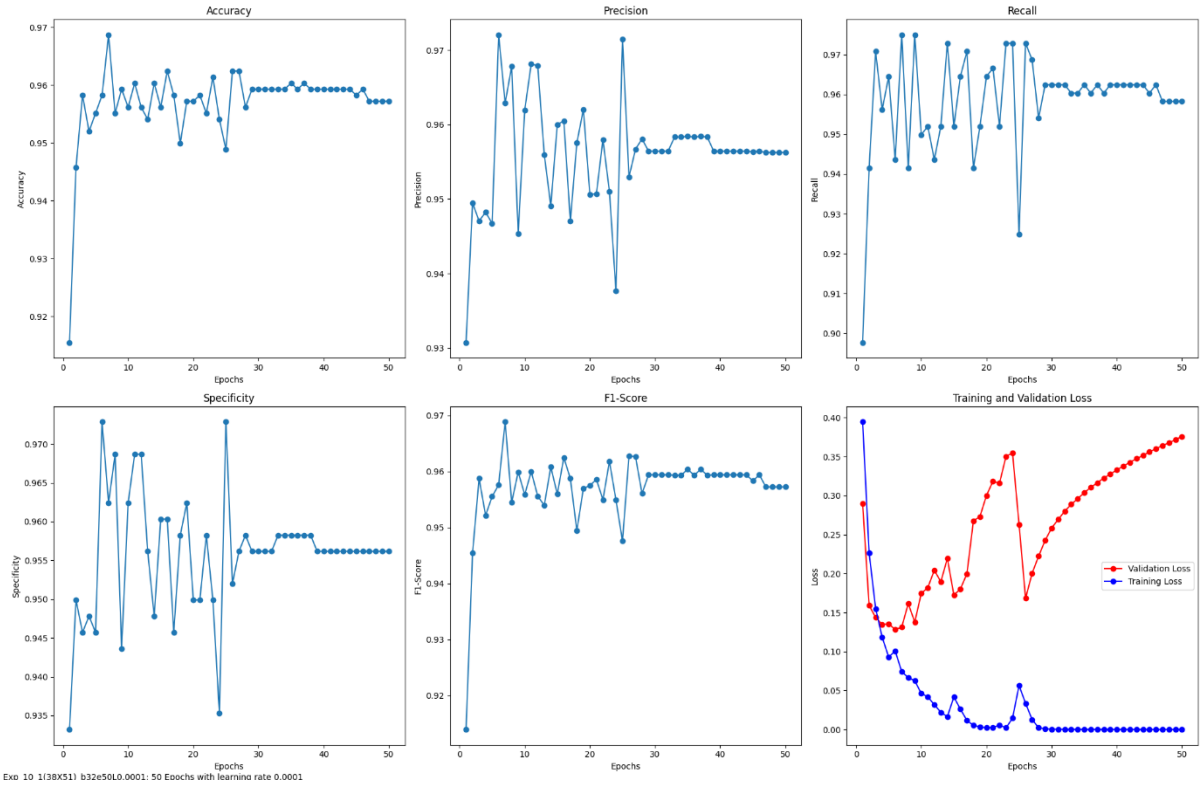

Figure 57:Experiment 4 results

5.5.1. Discussion and Evaluation

For a comprehensive evaluation of the contour detection with the optical flow model, hereafter referred to as the contour detection model, the performance metrics of three distinct models: the original research model, the replicated model, and the Canny Edge Type I model were compared during the testing and training phase. The plots illustrated in Figure 16, Figure 18 and Figure 57 represent performance results observed during the training phase of our models: the replicated model, the Canny Edge Type I model, and the contour detection model respectively, however similar data for the original research is not available for comparison.

Throughout the training phase, all models displayed exceptional performance across metrics. Specifically, all the models achieved a commendable accuracy with the original model achieving a 98.94% as early as the 5th epoch. Our contour detection model achieved a peak accuracy of 96.87% by the 7th epoch. A detailed examination revealed that our modified model's recall stands superior to the replication model. Both canny's and contour detection's recall performed exceptionally well with the contour detection model performing slightly better than the canny model. The specificity of the original model exhibits considerable fluctuations across the epochs whereas, the contour detection model stabilised at 95.62% after fluctuating till the 27th epoch. The models demonstrated noteworthy precision and F1 score throughout the training, however, the replicated model only achieved a peak precision of 84.17% by the 17th epoch. It is also noteworthy that in the contour detection model, all the metrics stabilised by the 27th epoch, suggesting that the model is capable of learning faster compared to other models, indicating a faster adaptation to patterns in data.

However, it has been observed that all the models exhibit a declining average training loss, while the average validation consistently increases, implying overfitting in all scenarios. Out of all the models, the contour detection model had the least average validation loss suggesting our model will exhibit a better performance, even though all models highlight the challenge of reduced generalisation. Another interesting insight here is with time and processing complexity, with replicated and contour detection models demonstrating a tenfold increase in both time and pre-processed data size, primarily due to the presence of Optical Flow.

Models	Accuracy	Precision	Recall	Specificity	F1-Score
Research Paper	0.9564	0.9691	0.9795	0.8308	0.9743
Replication	0.9902	0.8304	0.8542	0.9557	0.8421
Canny Edge Type I (Experiment 1.1)	0.9450	0.3564	0.9885	0.9437	0.5293
Contour Detection with Optical Flow	0.9638	0.4579	0.9896	0.9630	0.6261

Table 28: Experiment 4 Results comparison

Table 28 summarises the results obtained during the model evaluation. The observations suggest that our replicated model performed better than the research paper in accuracy but exhibited contrasting behaviour in all other metrics. Notably, our models showcased poor precision and recall in the testing phase compared to the original and replicated model which needs to be further investigated. We notice that both canny and contour detection models outperformed the replicated model. Furthermore, our contour detection model

outperforms the edge detector model in all metrics, indicating that this model is an optimal choice concerning the model evaluation/testing performance.

In conclusion, all model's evaluation reveals an overall satisfactory result with the contour detection model emerging as the preferred option, demonstrating optimal trade-offs between accuracy, precision specificity, and recall. Nevertheless, the heightened complexities in terms of time and dataset size raise valid concerns. All the models can be further refined to enhance their generalisation to fit in with a real-world scenario.

6. Conclusion

To conclude, this study aimed to address the research questions through a comprehensive comparison of performance metrics throughout the model's training and testing phases. Our research was aimed at analysing the impact of employing various data preprocessing techniques on existing research, to gain a profound understanding of the domain. We could summarise that data preprocessing has a significant influence on the performance of an accidental fall detection model. Notably, substantial improvements have been observed in the model's Recall/Sensitivity, emphasizing the efficacy of preprocessing techniques. Our findings indicate that the overall model exhibits significantly reduced time and processing complexity, suggesting a potential opportunity to train models on a much larger dataset without compromising on the model performance.

From our results for the first research question, we conclude that a model utilising a background subtraction, followed by a canny edge detector outperforms models that use Laplacian or Sobel filters. The experiments not only showcase the model's capability to learn and discern intricate data patterns for the fall detection model but also confirm the reduced time and data processing complexity compared to the original paper. For our second research question, exploring varying threshold values using the optimal model from our first research question revealed a minimal impact on the overall model performance. However, a model with a threshold ratio of 1:2 demonstrated a slightly superior performance compared to a threshold ratio of 1:3.

The analysis of the third question, focusing on the impact of the output resolution on the model performance, revealed that a resolution of 76x57 should be considered an optimal choice for processing. This is after careful consideration of the trade-off between the model performance and the associated time and processing complexity introduced in the data preprocessing. Finally, our fourth set of experiments unveiled that a model with contour detection with optical flow surpasses both the original and edge detector models in terms of performance. However, this improvement comes at a cost of increased processing and dataset size complexities.

These conclusive results offer valuable insights, contributing to the expansion of knowledge in developing and deploying an accidental fall detection model that exclusively relies on a vision-based approach. This project lays the foundation for future endeavours that could lead to the development and deployment of a robust and efficient fall detection system.

7. Limitations and Future Works

Although our study yields invaluable insights, it is essential to acknowledge that almost all the models exhibited a few limitations that need to be addressed. One major limitation was the overfitting tendency of all models which would reduce the model's ability to perform on unseen data. The exact reason for the model's behaviour needs to be investigated further, but an improvement in the generalisation could potentially help us in making a model best fits the real world. Another notable limitation observed was the consistent generation of low precision and F1 Score during model evaluation. The model exhibited a high number of False Positives(FP) which is predominantly contributing towards the low precision and F1 score. One potential contributing factor could be the high disparity in sample count, resulting in a balanced dataset with as few as 1916 fall and non-fall scenarios. This issue needs to be addressed before implementing this in a real-life accidental fall detection system.

The insights gained from this project raised a few concerns that need to be addressed with further investigation and model refinement. Future work can address the limitation of overfitting by conducting experiments with few established methods such as changing the model structure, introducing regularisation, or deploying an early stopping mechanism. We can also investigate the possibility of data leakage as well. Addressing the limitation of poor precision can be achieved through modification of the dataset using data augmentation techniques to increase the fall samples. Furthermore, an investigation into refining the contour detection optical flow model, by conducting a similar model comparison can be planned. Experimenting with the different optical flow methods also holds potential for prospective future endeavours.

8. References

1. ‘Artificial intelligence in businesses in different sectors’ (2020), 5 August. Available at: https://opensistemas.com/en/artificial-intelligence-applied-to-businesses-in-different-sectors/ (Accessed: 5 November 2023).

2. Beaupré, D.-A., Bilodeau, G.-A. and Saunier, N. (2018) ‘Improving Multiple Object Tracking with Optical Flow and Edge Preprocessing’. arXiv. Available at: https://doi.org/10.48550/arXiv.1801.09646.

3. Doulamis, N. (2016) ‘Vision Based Fall Detector Exploiting Deep Learning’, in *Proceedings of the 9th ACM International Conference on PErvasive Technologies Related to Assistive Environments*. New York, NY, USA: Association for Computing Machinery (PETRA ’16), pp. 1–8. Available at: https://doi.org/10.1145/2910674.2935836.

4. Espinosa, R. *et al.* (2019) ‘A vision-based approach for fall detection using multiple cameras and convolutional neural networks: A case study using the UP-Fall detection dataset’, *Computers in Biology and Medicine*, 115, p. 103520. Available at: https://doi.org/10.1016/j.compbiomed.2019.103520.

5. *Fall Detection Based on RetinaNet and MobileNet Convolutional Neural Networks | IEEE Conference Publication | IEEE Xplore* (no date). Available at: https://ieeexplore.ieee.org/document/9334570 (Accessed: 6 November 2023).

6. Kamangir, H. *et al.* (2022) ‘Importance of 3D convolution and physics on a deep learning coastal fog model’, *Environmental Modelling & Software*, 154, p. 105424. Available at: https://doi.org/10.1016/j.envsoft.2022.105424.

7. Kong, Y. *et al.* (2019) ‘Learning spatiotemporal representations for human fall detection in surveillance video’, *Journal of Visual Communication and Image Representation*, 59, pp. 215–230. Available at: https://doi.org/10.1016/j.jvcir.2019.01.024.

8. Liao, S.-K. and Liu, B.-Y. (2010) ‘An edge-based approach to improve optical flow algorithm’, in *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*. *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*, pp. V6-45-V6-51. Available at: https://doi.org/10.1109/ICACTE.2010.5579363.

9. M, V. (2022) ‘Comprehensive Guide to Edge Detection Algorithms’, *Analytics Vidhya*, 6 August. Available at: https://www.analyticsvidhya.com/blog/2022/08/comprehensive-guide-to-edge-detection-algorithms/ (Accessed: 7 January 2024).

10. Martínez-Villaseñor, L. *et al.* (2019) ‘UP-Fall Detection Dataset: A Multimodal Approach’, *Sensors*, 19(9), p. 1988. Available at: https://doi.org/10.3390/s19091988.

11. *OpenCV: Canny Edge Detection* (no date). Available at: https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html (Accessed: 4 January 2024).

12. Shinde, S., Kothari, A. and Gupta, V. (2018) 'YOLO based Human Action Recognition and Localization', *Procedia Computer Science*, 133, pp. 831–838. Available at: https://doi.org/10.1016/j.procs.2018.07.112.

13. Shu, F. and Shu, J. (2021) 'An eight-camera fall detection system using human fall pattern recognition via machine learning by a low-cost android box', *Scientific Reports*, 11(1), p. 2471. Available at: https://doi.org/10.1038/s41598-021-81115-9.

14. *The human cost of falls - UK Health Security Agency* (2014). Available at: https://ukhsa.blog.gov.uk/2014/07/17/the-human-cost-of-falls/ (Accessed: 6 November 2023).

15. Zhang, L. and Liang, Y. (2010) 'Motion Human Detection Based on Background Subtraction', in *2010 Second International Workshop on Education Technology and Computer Science*. *2010 Second International Workshop on Education Technology and Computer Science*, pp. 284–287. Available at: https://doi.org/10.1109/ETCS.2010.440.

16. Zhang, Y. *et al.* (2022) 'Visual Surveillance for Human Fall Detection in Healthcare IoT', *IEEE MultiMedia*, 29(1), pp. 36–46. Available at: https://doi.org/10.1109/MMUL.2022.3155768.

9. Appendices

9.1. Appendix A

Link to the repository:

https://github.com/binson95thomas/MscProjectHAR.git

9.2. Appendix B

Code for preprocessing(Double click to view the entire code)

```
import cv2
import os
import numpy as np
import csv
import re
from timeit import default_timer as timer
from datetime import datetime


# Class for handling the reading of dataset
class DatasetDirectoryHandler:
    def __init__(self, base_folder):
        self.base_folder = base_folder

    def get_subject_folders(self):
        return self._get_subfolders(self.base_folder)

    def get_activity_folders(self, subject_folder):
        subject_path = os.path.join(self.base_folder, subject_folder)
        return self._get_subfolders(subject_path)

    def get_trial_folders(self, subject_folder, activity_folder):
        activity_path = os.path.join(self.base_folder, subject_folder,
activity_folder)
        return self._get_subfolders(activity_path)

    def get_camera_folders(self, subject_folder, activity_folder,
trial_folder):
        trial_path = os.path.join(
            self.base_folder, subject_folder, activity_folder, trial_folder
        )
        return self._get_subfolders(trial_path)

    def _get_subfolders(self, folder):
        folders = [
            d for d in os.listdir(folder) if
os.path.isdir(os.path.join(folder, d))
```

```
        ]
        return sorted(folders, key=lambda x: int(re.search(r"\d+",
x).group()))


# Class for loading frames
class FrameLoader:
    def __init__(self, dataset_folder):
        self.dataset_folder = dataset_folder

    def get_video_folders(self):
        return [
            d
            for d in os.listdir(self.dataset_folder)
            if os.path.isdir(os.path.join(self.dataset_folder, d))
        ]

    def load_frames_from_video(self, video_folder):
        image_folder = os.path.join(self.dataset_folder, video_folder)
        file_names = sorted(
            [
                f
                for f in os.listdir(image_folder)
                if f.endswith(".jpg") or f.endswith(".png")
            ]
        )

        return [
            (fn[:-4], cv2.imread(os.path.join(image_folder, fn),
cv2.IMREAD_GRAYSCALE))
            for fn in file_names
        ]


# defines the background Subtraction Model
class bg_sub:
    def __init__(self):
        self.fgbg = cv2.createBackgroundSubtractorMOG2(detectShadows=True)
        self.fgbg.setShadowValue(0)
        self.fgbg.setShadowThreshold(0.5)

    # Function to perform edge Detection
    def compute_edge_detector(self, current_frame):
        current_frame = cv2.normalize(
            src=current_frame,
            dst=None,
            alpha=0,
            beta=255,
            norm_type=cv2.NORM_MINMAX,
```

```
            dtype=cv2.CV_8U,
        )
        fgmask_curr = self.fgbg.apply(current_frame)
        kernel = np.ones((5, 5), np.uint8)
        fgmask_curr = cv2.morphologyEx(fgmask_curr, cv2.MORPH_OPEN, kernel)
        fgmask_curr = cv2.morphologyEx(fgmask_curr, cv2.MORPH_CLOSE, kernel)
        current_frame_masked = cv2.bitwise_and(
            current_frame, current_frame, mask=fgmask_curr
        )
        current_frame_masked = cv2.Canny(current_frame_masked, 65, 195)
        resized_frame = cv2.resize(current_frame_masked, (51, 38))
        return resized_frame


# class for writing the data into array
class NumpyWriter:
    def __init__(self, output_folder):
        self.output_folder = output_folder
        os.makedirs(output_folder, exist_ok=True)

    def write_array(self, array, name):
        file_path = os.path.join(self.output_folder, f"{name}.npy")
        directory = os.path.dirname(file_path)
        os.makedirs(directory, exist_ok=True)
        np.save(file_path, array)


# Class to process Optical Flow
class OpticalFlowProcessor:
    def __init__(self, dataset_folder, output_folder, fps=18):
        self.frame_loader = FrameLoader(dataset_folder)
        self.numpy_writer = NumpyWriter(output_folder)
        self.fps = fps
        self.window_size = fps
        self.overlap = fps // 2
        self.bg_subtraction = bg_sub()

    def total_seconds_from_timestamp(timestamp: str) -> float:
        hours, minutes, seconds = map(float,
timestamp.split("T")[1].split("_"))
        total_seconds = hours * 3600 + minutes * 60 + seconds
        return total_seconds

    def increment_timestamp(timestamp: str) -> str:
        date, time = timestamp.split("T")
        try:
            hours, minutes, remainder = time.split("_")
            seconds, ms = remainder.split(".")
        except ValueError:
```

```
            print(f"Error with timestamp: {time}")
            raise

        ms = int(ms)
        seconds = int(seconds)
        minutes = int(minutes)
        hours = int(hours)

        ms += 500000
        if ms >= 1000000:
            ms -= 1000000
            seconds += 1

        if seconds >= 60:
            seconds -= 60
            minutes += 1

        if minutes >= 60:
            minutes -= 60
            hours += 1

        time_str = f"{hours:02}_{minutes:02}_{seconds:02}.{ms:06}"
        return f"{date}T{time_str}"

    # procedure to process videos
    def process_video(self, video_folder):
        frames = self.frame_loader.load_frames_from_video(video_folder)
        i = 0
        num_frames_in_window = int(self.fps)
        overlap_frames = int(self.overlap)
        last_frame_time = frames[-1][0]
        last_frame_seconds =
OpticalFlowProcessor.total_seconds_from_timestamp(
            last_frame_time
        )
        timestamp = frames[i][0]
        while i < len(frames) - num_frames_in_window:
            window_end = min(i + num_frames_in_window, len(frames))
            # optical_flows_u= []
            # optical_flows_v = []
            canny_frames = []
            for j in range(i, window_end - 1):
                try:
                    final_components =
self.bg_subtraction.compute_edge_detector(
                        frames[j + 1][1]
                    )
                    canny_frames.append(final_components)
```

```
                except cv2.error as e:
                    print(
                        f"Error processing frame {frames[j][0]} from video
{video_folder}. Error: {e}"
                    )
                    continue
            components_stacked = np.stack(canny_frames, axis=0)
            window_name = f"{video_folder}_{timestamp}"
            self.numpy_writer.write_array(components_stacked, window_name)
            timestamp = OpticalFlowProcessor.increment_timestamp(timestamp)
            next_increment_seconds =
OpticalFlowProcessor.total_seconds_from_timestamp(
                timestamp
            )
            if last_frame_seconds - next_increment_seconds < 1.0:
                break
            i += num_frames_in_window - overlap_frames

    def run(self):
        dir_handler =
DatasetDirectoryHandler(self.frame_loader.dataset_folder)
        logging_output(f"# Data Process starts for {output_folder} ##",
log_path)
        for subject_folder in dir_handler.get_subject_folders():
            for activity_folder in
dir_handler.get_activity_folders(subject_folder):
                for trial_folder in dir_handler.get_trial_folders(
                    subject_folder, activity_folder
                ):
                    for camera_folder in dir_handler.get_camera_folders(
                        subject_folder, activity_folder, trial_folder
                    ):
                        print(f"Processing video: {camera_folder}")
                        logging_output(f"Processing video:
{camera_folder},log_path")

                        start = timer()
                        self.process_video(
                            os.path.join(
                                subject_folder,
                                activity_folder,
                                trial_folder,
                                camera_folder,
                            )
                        )
                        print("Time Taken: ", timer() - start)
                        logging_output(f"Time Taken: {timer() - start}",
log_path)
```

```
                        now = datetime.now()
                        current_time = now.strftime("%H:%M:%S")
                        print("Process Completed at : ", current_time)


def logging_output(message, file_path=f"../Outputs/prep_log.txt"):
    try:
        # Write to file
        now = datetime.now()
        current_time = now.strftime("%H:%M:%S")
        with open(file_path, "a") as file:
            file.write(f"{current_time} : {message} \n")
    except Exception as e:
        print(f"Error logging: {e}")


if __name__ == "__main__":
    try:
        model_type = "Exp_1_1_OG_BGS_Canny"
        # To print the processing logs
        log_path = f"../Outputs/{model_type}/prep_log.log"
        dataset_folder = "../UP-Fall"
        output_folder = f"../Outputs/{model_type}/Unbalanced"
        processor = OpticalFlowProcessor(dataset_folder, output_folder)
        processor.run()
        print("################Data Process Complete############")
        logging_output(f"# Data Process Complete for {output_folder} ##",
log_path)
        print("~~~~~~~~Entering Hibernation Mode ~~~~~~~~~~~~~")
    except:
        print("~~~~~~~~Error Occurred~~~~~~~~")
```

9.3. Appendix C

Code for loading the data to Numpy Array

```
import os
import numpy as np
import pandas as pd
if __name__ == "__main__":
    #Loading File with the labels
    csv_file_path = './Features_1&0.5_Vision.csv'
    labels_df = pd.read_csv(csv_file_path, skiprows=1)
    base_folder = '../Outputs/Exp_1_1_OG_BGS_Canny/Unbalanced'
    print(f"============================================================ ")
    print(f"file is {base_folder}")
    filename_label_dict = {}
    print(f"============================================================ ")
    for root, dirs, files in os.walk(base_folder):
        for file in files:
            if file.endswith('.npy'):
                parts = file.split('_')
                if len(parts) < 4:  # if there are not enough parts, skip this
file
                    print(f"Invalid filename {file}, skipping.")
                    continue
                timestamp = '_'.join(parts[1:])  # Join the timestamp parts
                timestamp = timestamp.rsplit('.', 1)[0]  # Remove the file
extension
                timestamp = timestamp.replace('_', ':', 2)
                print(f"Processing {file} with timestamp {timestamp}")

                matched_rows =
labels_df[labels_df['Timestamp'].str.contains(timestamp, na=False)]
                #Matching the Label with timestamp
                if not matched_rows.empty:
                    # Use the first matched row
                    label_row = matched_rows.iloc[0]
                    label = label_row['Tag']
                    # Load the numpy file
                    file_path = os.path.join(root, file)
                    array = np.load(file_path, allow_pickle=True)

                    # Replace existing label or add new one
                    if isinstance(array, dict) and 'array' in array:
```

```
                        array['label'] = label
                    else:
                        array = {'array': array, 'label': label}

                    # Save the updated data back to the numpy file
                    np.save(file_path, array)

                    # Update the filename_label_dict
                    filename_label_dict[file] = label

                else:
                    print(f"No label found for {file}, deleting the file.")
                    os.remove(os.path.join(root, file))


    for filename, assigned_label in filename_label_dict.items():
        print(f"verifying {filename}")
        parts = filename.split('_')
        timestamp = f"{parts[1]}T{parts[2]}:{parts[3].split('.', 1)[0]}"

        # Find corresponding row in CSV
        label_row = labels_df[labels_df['Timestamp'].str.contains(timestamp,
na=False)]

        # Extract the original label from the CSV
        if not label_row.empty:
            original_label = label_row.iloc[0]['Tag']

            # Check if the original label and assigned label match
            if assigned_label != original_label:
                print(f"Label mismatch for {filename}: assigned
{assigned_label}, original {original_label}")

    print(f"**********************************************************
")
    with open('../Outputs/Exp_1_1_OG_BGS_Canny/loader_log.log', 'a') as file:
                file.write(f' Process complete for {base_folder} \n')
```

9.4. Appendix D

Downsampling

```
import os
import numpy as np
import shutil

def downsample_dataset(original_folder, output_folder):
    dataset_info = []
    for root, _, files in os.walk(original_folder):
        for file in files:
            if file.endswith('.npy'):
                print(f"Processing {file} ")
                file_path = os.path.join(root, file)
                data = np.load(file_path, allow_pickle=True).item()
                label = data['label']
                dataset_info.append((file_path, label))
    #Identifying the fall and No Fall samples based on activity id
    # In context with dataset
                #Activity ID - 1-5 is a fall
                #Acitivity ID - 6-11 is a No fall
    fall_samples = [info for info in dataset_info if 1 <= info[1] <= 5]
    non_fall_samples = [info for info in dataset_info if info[1] > 5]
    #Calculating total falls
    num_falls = len(fall_samples)
    print(num_falls)
    #Randomly choosing no fall scenario
    downsampled_non_falls_indices = np.random.choice(len(non_fall_samples),
size = num_falls, replace = False)
    downsampled_non_falls = [non_fall_samples[i] for i in
downsampled_non_falls_indices]

    for file_path, _ in fall_samples + list(downsampled_non_falls):
        print(f"Creating file {file_path} ")
        relative_path = os.path.relpath(file_path, original_folder)
        new_path = os.path.join(output_folder, relative_path)

        os.makedirs(os.path.dirname(new_path), exist_ok=True)

        shutil.copyfile(file_path, new_path)

if __name__ == "__main__":
    script_path = os.path.abspath(__file__)
    script_name = os.path.basename(script_path)
    name_only=os.path.splitext(script_name)
```

```
    model_type=name_only[0]
    print(f"Downsampling ModelName is {model_type}")
    original_folder = f'../Outputs/{model_type}/Unbalanced'
    output_folder = f'../Outputs/{model_type}/Balanced/'
    with open(f'../Outputs/{model_type}/DS_log.log', 'a') as file:
            file.write(f' Downsampling started for {original_folder} with ou
in {output_folder} \n')
    print(f"file is {original_folder} and {output_folder}")
    downsample_dataset(original_folder, output_folder)
    with open(f'../Outputs/{model_type}/DS_log.log', 'a') as file:
        file.write(f' Downsampling Complete for {original_folder} with ou in
{output_folder} \n')
```

9.5. Appendix E

Edge detector codes

```
#Function to compute edge detector for experiment
#Function takes each frame as an input

def compute_edge_detector(self, current_frame):
        #Applying normalisation
         current_frame = cv2.normalize(
             src=current_frame,
             dst=None,
             alpha=0,
             beta=255,
             norm_type=cv2.NORM_MINMAX,
             dtype=cv2.CV_8U,
         )
         #Applying the background subtraction on frame
         fgmask_curr = self.fgbg.apply(current_frame)
         #Kernel for morphological operations
         kernel = np.ones((5, 5), np.uint8)
         #Generating the mask
         fgmask_curr = cv2.morphologyEx(fgmask_curr, cv2.MORPH_OPEN, kernel)
         fgmask_curr = cv2.morphologyEx(fgmask_curr, cv2.MORPH_CLOSE, kernel)
         #Applying the mask on current frame
         current_frame_masked = cv2.bitwise_and(
             current_frame, current_frame, mask=fgmask_curr
         )
         #Performing edge detection(This is changed to Sobel and Laplacian
according to experiment performed)
         current_frame_masked = cv2.Canny(current_frame_masked, 65, 195)
         #Resizing the frame to desired resolution
         resized_frame = cv2.resize(current_frame_masked, (51, 38))
         return resized_frame
```

9.6. Appendix F

Contour detection

```
#Function to compute the optical flow for contour detection
#Function takes the current and previous frames as input
def compute_optical_flow(self, prev_frame, current_frame):
        #Perfomring Normalisation
        prev_frame = cv2.normalize(src=prev_frame, dst=None, alpha=0,
beta=255, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)
        current_frame = cv2.normalize(src=current_frame, dst=None, alpha=0,
beta=255, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)

         # Using histogram equilised image to compute dynamic threshold for
edge detectors
        equalized_curr= cv2.equalizeHist(current_frame)
        equalized_prev= cv2.equalizeHist(prev_frame)
        blurred = cv2.GaussianBlur(equalized_curr , (5, 5), 0)
        blurred_prev = cv2.GaussianBlur(equalized_prev , (5, 5), 0)
        med_val = np.median(blurred)
        lower = int(max(0 ,0.5*med_val))
        upper = int(min(255,1.5*med_val))

        #Edge detection
        edges_new=cv2.Canny(blurred,lower, upper)
        edges_new_prev=cv2.Canny(blurred_prev,lower, upper)

        #Computing the threshold
        ret, thresh = cv2.threshold(edges_new, 150, 255, cv2.THRESH_BINARY)
        ret_prev, thresh_prev = cv2.threshold(edges_new_prev, 150, 255,
cv2.THRESH_BINARY)

        #Identifying contours
        contours, hierarchy = cv2.findContours(image=thresh,
mode=cv2.RETR_EXTERNAL, method=cv2.CHAIN_APPROX_SIMPLE)
        contours_prev , hierarchy_prev = cv2.findContours(image=thresh_prev,
mode=cv2.RETR_EXTERNAL, method=cv2.CHAIN_APPROX_SIMPLE)

        image = current_frame.copy()
        image_prev = prev_frame.copy()

        #Drawing contour on original image
        cv2.drawContours(image=image, contours=contours, contourIdx=-1,
color=(0, 255, 0), thickness=2, lineType=cv2.LINE_AA)
        cv2.drawContours(image=image_prev, contours=contours_prev,
contourIdx=-1, color=(0, 255, 0), thickness=2, lineType=cv2.LINE_AA)

        #Performing background subtraction
```

```
        fgmask_prev = self.fgbg.apply(image)
        fgmask_curr = self.fgbg.apply(image_prev)

        kernel = np.ones((5,5), np.uint8)

        #Performing morphological operations
        fgmask_prev = cv2.morphologyEx(fgmask_prev, cv2.MORPH_OPEN, kernel)
        fgmask_prev = cv2.morphologyEx(fgmask_prev, cv2.MORPH_CLOSE, kernel)
        fgmask_curr = cv2.morphologyEx(fgmask_curr, cv2.MORPH_OPEN, kernel)
        fgmask_curr = cv2.morphologyEx(fgmask_curr, cv2.MORPH_CLOSE, kernel)

        #Applying the mask
        prev_frame_masked = cv2.bitwise_and(prev_frame, prev_frame,
mask=fgmask_prev)
        current_frame_masked = cv2.bitwise_and(current_frame, current_frame,
mask=fgmask_curr)

        #Refining the output to mitigate noise
        prev_frame = cv2.medianBlur(prev_frame_masked, 5)
        current_frame = cv2.medianBlur(current_frame_masked, 5)
        prev_blurred = cv2.GaussianBlur(prev_frame, (5, 5), 0)
        curr_blurred = cv2.GaussianBlur(current_frame, (5, 5), 0)

        #Computing Optical Flow
        flow = cv2.calcOpticalFlowFarneback(prev_blurred, curr_blurred, None,
0.5, 3, 15, 3, 5, 1.2, 0)
        u_component = flow[..., 0]
        v_component = flow[..., 1]

        #Resizing the image to desired output
        resized_u = cv2.resize(u_component, (51, 38))
        resized_v = cv2.resize(v_component, (51, 38))

        return resized_u, resized_v
```

9.7. Appendix G

Code for training the 3D CNN

```
if __name__ == "__main__":
    try:
        script_path = os.path.abspath(__file__)
         # Extract the file name from the path
        script_name = os.path.basename(script_path)
        name_only=os.path.splitext(script_name)
        model_folder=name_only[0]

        #Defining the path for balanced and umnbalanced data
        features_path =
f'C:\\Users\\bt22aak\\GPU_DS\\Exp_1\\{model_folder}\\Balanced'
        test_path =
f'C:\\Users\\bt22aak\\GPU_DS\\Exp_1\\{model_folder}\\Unbalanced'

        #Defining paramters
        batch_size=32
        num_epochs=50
        learning_rate=0.0001
        str_model_type=f'{model_folder}_b{batch_size}e{num_epochs}L{learning_r
ate}'

        if not os.path.exists(f'./Results/{model_folder}'):
            try:
                # Create the directory and its parents if they don't exist
                os.makedirs(f'./Results/{model_folder}')
            except OSError as e:
                print(f"Error creating directory ./Results/{model_folder}:
{e}")
        #Defining the logs
        log_path=f'./Results/{model_folder}/Trainlog.log'
        print(f"Path identified is {features_path} and {test_path}")
        logging_output(f"Path identified is {features_path} and
{test_path}",log_path)
        print(f"Starting the processing of {str_model_type}")
        logging_output(f"Starting the processing of
{str_model_type}",log_path)

        code_start=timer()

        #Extraacting the data
        train_val_dataset = OpticalFlow2DDataset(features_path)
        test_dataset = OpticalFlow2DDataset(test_path)

        #Test Train Split
```

```
        train_idx, val_idx = train_test_split(range(len(train_val_dataset)),
test_size=0.25, random_state=42, stratify=train_val_dataset.labels)

        train_dataset = torch.utils.data.Subset(train_val_dataset, train_idx)
        val_dataset = torch.utils.data.Subset(train_val_dataset, val_idx)

        dataloader_train = DataLoader(train_dataset, batch_size=32,
shuffle=True)
        dataloader_val = DataLoader(val_dataset, batch_size=32, shuffle=False)
        dataloader_test = DataLoader(test_dataset, batch_size=32,
shuffle=False)

        #Checking CUDA in the system
        if torch.cuda.is_available() :
            print(f"Running on GPU")
            logging_output("Running on GPU",log_path)
        else:
            print(f"CPU Only")
            logging_output("Running on CPU Only",log_path)

        device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

        print(f"Model is {str_model_type}")
        logging_output(f"Model is {str_model_type}",log_path)

        #Loading the model to CUDA
        model = FallDetectionCNN().to(device)

        #Training the model
        model = train_model(dataloader_train,
dataloader_val,num_epochs,learning_rate)
        criterion = nn.CrossEntropyLoss().to(device)

        #Evaluating the model
        test_loss, test_accuracy, test_precision, test_recall,
test_specificity, test_f1, true_labels, predictions = evaluate_model(model,
dataloader_test, criterion, device)

        #Printing Evaluation Metrics
        print(f"Test Loss: {test_loss:.4f}, Test Accuracy:
{test_accuracy:.4f}, Test Precision: {test_precision:.4f}, Test Recall:
{test_recall:.4f}, Test Specificity: {test_specificity:.4f}, Test F1-Score:
{test_f1:.4f}")
        logging_output(f"Test Loss: {test_loss:.4f}, Test Accuracy:
{test_accuracy:.4f}, Test Precision: {test_precision:.4f}, Test Recall:
{test_recall:.4f}, Test Specificity: {test_specificity:.4f}, Test F1-Score:
{test_f1:.4f}",log_path)
        print(f"Total Time taken: {timer() -code_start } seconds")
```

```
        logging_output(f"Total Time taken: { timer() -code_start } Seconds",
log_path)

        #Plotting the Confusion Matrix
        plot_confusion_matrix(true_labels, predictions, classes = ["No Fall",
"Fall"])

        #Saving the model
        torch.save(model.state_dict(),
f'./Results/{model_folder}/{str_model_type}.pth')

    #Handling Exception
    except Exception as E :
        err_msg=f"Error occured {E}"
        print(err_msg)
        error_stack = traceback.format_exc()
        print(f"Error stack:\n{error_stack}")
        logging_output("******************************",log_path)
        logging_output(err_msg,log_path)
        logging_output(error_stack,log_path)
        logging_output("******************************",log_path)
```

9.8.Appendix H

Code for visualisation

```
#Visualising the preprocessde frame into Grids
def
visualise_experiments(self,image_paths,variable_names,grid_height=2,grid_width
=3):
    desired_size = (512, 384)
    # grid_height, grid_width = 2, 4
    grid_image = np.zeros((grid_height * desired_size[1], grid_width *
desired_size[0], 3), dtype=np.uint8)

    for i,(variable_name, image) in enumerate(zip(variable_names,
image_paths)):

        # Resize the image to a fixed size
        resized_image = cv2.resize(image, desired_size)
        if len(resized_image.shape) == 2:
                resized_image = cv2.cvtColor(resized_image,
cv2.COLOR_GRAY2RGB)
        # Calculate the row and column index for the grid
        row = i // grid_width
        col = i % grid_width
        # Place the resized image into the grid
        grid_image[row * desired_size[1]:(row + 1) * desired_size[1], col *
desired_size[0]:(col + 1) * desired_size[0], :] = resized_image
        label = variable_name
        cv2.putText(grid_image, label, (col * desired_size[0] + 10, (row + 1)
* desired_size[1] - 30),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1, cv2.LINE_AA)
    # Display the grid using cv2.imshow()
    cv2.imshow("Image Grid", grid_image)
    cv2.waitKey(1)

#Following code generates all the experiment output using code of appendix E
# and pass the experiment frames to visualise_experiments
def compute_edge_detector(self, current_frame):
    current_frame = cv2.normalize(
        src=current_frame,
        dst=None,
        alpha=0,
        beta=255,
        norm_type=cv2.NORM_MINMAX,
        dtype=cv2.CV_8U,
```

```
    )
    fgmask_curr = self.fgbg.apply(current_frame)
    kernel = np.ones((5, 5), np.uint8)
    fgmask_curr = cv2.morphologyEx(fgmask_curr, cv2.MORPH_OPEN, kernel)
    fgmask_curr = cv2.morphologyEx(fgmask_curr, cv2.MORPH_CLOSE, kernel)
    current_frame_masked = cv2.bitwise_and(
        current_frame, current_frame, mask=fgmask_curr
    )
    exp_1 = cv2.Canny(current_frame_masked, 65, 195)


    edge_frame_curr =cv2.Canny(current_frame,65, 195)
    fgmask_curr = self.fgbg.apply(current_frame)
    kernel = np.ones((5, 5), np.uint8)
    fgmask_curr = cv2.morphologyEx(fgmask_curr, cv2.MORPH_OPEN, kernel)
    fgmask_curr = cv2.morphologyEx(fgmask_curr, cv2.MORPH_CLOSE, kernel)
    exp_2 = cv2.bitwise_and(
        edge_frame_curr, edge_frame_curr, mask=fgmask_curr
    )

    fgmask_curr = self.fgbg.apply(current_frame)
    kernel = np.ones((5, 5), np.uint8)
    fgmask_curr = cv2.morphologyEx(fgmask_curr, cv2.MORPH_OPEN, kernel)
    fgmask_curr = cv2.morphologyEx(fgmask_curr, cv2.MORPH_CLOSE, kernel)
    current_frame_masked = cv2.bitwise_and(
        current_frame, current_frame, mask=fgmask_curr
    )
    exp_3 = np.uint8(cv2.Laplacian(current_frame_masked,cv2.CV_64F))


    edge_frame_curr =np.uint8(cv2.Laplacian(current_frame,cv2.CV_64F))
    fgmask_curr = self.fgbg.apply(current_frame)
    kernel = np.ones((5, 5), np.uint8)
    fgmask_curr = cv2.morphologyEx(fgmask_curr, cv2.MORPH_OPEN, kernel)
    fgmask_curr = cv2.morphologyEx(fgmask_curr, cv2.MORPH_CLOSE, kernel)
    exp_4 = cv2.bitwise_and(
        edge_frame_curr, edge_frame_curr, mask=fgmask_curr
    )

    fgmask_curr = self.fgbg.apply(current_frame)
    kernel = np.ones((5, 5), np.uint8)
    fgmask_curr = cv2.morphologyEx(fgmask_curr, cv2.MORPH_OPEN, kernel)
    fgmask_curr = cv2.morphologyEx(fgmask_curr, cv2.MORPH_CLOSE, kernel)
    current_frame_masked = cv2.bitwise_and(
        current_frame, current_frame, mask=fgmask_curr
    )
    exp_5 =cv2.Sobel(src=current_frame_masked, ddepth=cv2.CV_64F, dx=1, dy=1,
ksize=5)
```

```
    edge_frame_curr =cv2.Sobel(src=current_frame, ddepth=cv2.CV_64F, dx=1,
dy=1, ksize=5)
    fgmask_curr = self.fgbg.apply(current_frame)
    kernel = np.ones((5, 5), np.uint8)
    fgmask_curr = cv2.morphologyEx(fgmask_curr, cv2.MORPH_OPEN, kernel)
    fgmask_curr = cv2.morphologyEx(fgmask_curr, cv2.MORPH_CLOSE, kernel)
    exp_6 = cv2.bitwise_and(
        edge_frame_curr, edge_frame_curr, mask=fgmask_curr
    )
    # Load your images (replace these file paths with your own images)
    exp_5 = exp_5.astype(np.float32)  # Convert to float32
    exp_5 = cv2.convertScaleAbs(exp_5)  # Convert to CV_8U
    exp_6 = exp_6.astype(np.float32)  # Convert to float32
    exp_6 = cv2.convertScaleAbs(exp_6)  # Convert to CV_8U
    image_paths = [current_frame,exp_1,exp_2,exp_3,exp_4,exp_5,exp_6]
    # image_paths = [cexp_1,exp_2,exp_3,exp_4,exp_5,exp_1]
    image_paths2=[exp_5,exp_6]
    variable_names = ["Original","Experiment 1.1", "Experiment 1.2",
"Experiment 1.3", "Experiment 1.4", "Experiment 1.5", "Experiment 1.6"]
    variable_names2 = ["Experiment 5.1", "Experiment 6.1"]

    self.visualise_experiments(image_paths,variable_names,grid_height=2,grid_w
idth=4)

    resized_frame = cv2.resize(exp_1, (51, 38))
    return resized_frame
```