**Report draft**

**February 1, 2024**

**Say Meng Toh**

1. **Introduction**
   1. **Objective**

This report is a replication and 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. This replication study focuses on reproducing the original model’s methodology and architecture, assessing its performance, and interpreting the potential applications in our project. In addition to the replication, we have also undertaken further experimentation. This includes exploring alternative methods of processing optical flow and experimenting with 3D Convolutional Neural Networks to potentially enhance model accuracy and robustness.

1. **Methodology**
   1. **Dataset**

The UP-Fall detection dataset was used in this replication. It contains a collection of videos and sensor data that depict various activities, including both fall and non-fall incidents. The dataset comprises of 17 subjects performing 11 activities. Each activity was repeated 3 times. Each activity was also taken from two cameras from different perspectives. This combination gives a total of 1122 videos.

The dataset provides labels from 1-11, each describing a different action.

A white paper with black text

Description automatically generated

* 1. **Optical Flow**

Optical flow is a technique used to estimate motion between two consecutive frames in a video sequence. It takes two frames in a video and compares the pixels. The result is a motion image of the difference between the pixels of two frames. Using this technique, we can detect the motion that the subject takes over time.



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.

The Farneback method 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.

* 1. **Preprocessing and training**

The dataset provided already had the frames separated in folders, thus there was no need to perform frame extraction. Following the approach in the original research, we implemented a windowing technique. In this case, a 1 second window with a 0.5 second overlap is implemented. This resulted in multiple 1 second window length series of images to be used in 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 give 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:

For each window, the optical flow was determined using the Farneback method from the OpenCV library. The optical flow was then resized to a resolution of 38 x 51 pixels to standardize input dimensions. 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 fie. Each entry in the file consisted of a timestamp followed by the average magnitudes for that window. Finally, referencing the optical flow file provided by the dataset, appropriate labels were appended to each window’s data.

Before the pre-processed dataset goes through the CNN for training, it is split into 60% training, 20% validation, and 20% training. Adam optimizer was used with a learning rate of 0.0001 and binary cross-entropy as the loss function to optimize the model’s weights during training, over a maximum of 50 epochs.

1. **Neural Network Architecture**
   1. **2D CNN Model**

Following the research paper, a 2D Convolutional Neural Network was created, which can be described as follows:

* **2D Convolutional Layers for Feature Extraction:**

Layer 1 consisting of 128 convolution filters with a kernel size of 3 x 3.

Layer 2 consisting of 128 convolution filters with a kernel size of 3 x 3.

Layer 3 consisting 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 consisting of 64 ReLU units.

Layer 2 consisting of 128 ReLU units.

Layer 3 consisting of 254 ReLU units.

* **Output Layer:**

2D SoftMax layer with a single binary output:

* Fall (represented as 1)
* No-fall (represented as 0)

In the context of the dataset, activity IDs 1-5 are classified as falls, while the others are considered as no-falls.

* 1. **Transition to 3DCNN**

To improve model performance and robustness, we considered transitioning from 2D CNNs to 3D CNNs. In applications like fall detection, temporal dynamics play a crucial role, and this is where 3D CNNs have an advantage over 2D CNNs. 3D CNNs capture both spatial and temporal features, making them particularly suited for video data. With this in mind, we decided to experiment with 3D CNNs to explore whether they could offer enhanced accuracy and better capture nuances of fall patterns over time.

As we transitioned to 3D CNNs, the approach to optical flow computation and representation 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.

* 1. **3D CNN Model**

The architecture of the 3D CNN model 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 consisting of 128 convolution filters with a kernel size of 3 x 3 x 3.

Layer 2 consisting of 128 convolution filters with a kernel size of 3 x 3 x 3.

Layer 3 consisting 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 consisting of 64 ReLU units.

Layer 2 consisting of 128 ReLU units.

Layer 3 consisting of 254 ReLU units.

* **Output Layer:**

2D SoftMax layer with a single binary output:

* Fall (represented as 1)
* No fall (represented as 0)

Due to the results and simplicity of the model architecture when converted to a 3D CNN, we decided to experiment with different numbers and types of layers to improve the performance of the 3DCNN. The next iteration has the following architecture.

* **3D Convolutional Layers for Feature Extraction:**

Layer 1 consisting of 64 convolution filters with a kernel size of 3 x 3 x 3. Followed by a Batch Normalization layer.

Layer 2 consisting of 128 convolution filters with a kernel size of 3 x 3 x 3. Followed by a Batch Normalization layer.

Layer 3 consisting of 256 convolution filters with a kernel size of 3 x 3 x 3. Followed by a Batch Normalization layer.

Layer 4 consisting of 256 convolution filters with a kernel size of 3 x 3 x 3. Followed by a Batch Normalization layer.

* **Max-Pooling Layers:**

3D max-pooling layers are used after the first three convolutional layers to reduce spatial dimensions.

* **Global Average Pooling Layer:**

A global average pooling layer is utilized after the last convolutional layer to reduce the dimensions and extract the most relevant features.

* **Fully Connected Layers for Fall Detection:**

Layer 1 consisting of 128 ReLU units. Followed by a dropout layer with a rate of 0.5 for regularization.

Layer 2 consisting of 64 ReLU units. Followed by another dropout layer with a rate of 0.5.

* **Output Layer:**

A linear layer with 2 units, representing the two classes (fall and no-fall)

1. **Updates and Optimizations in Model Development**
   1. **Adding additional preprocessing**

Using the observations made regarding the noise of the optical flow, we experimented with other preprocessing techniques to reduce the noise and enhance the optical flow calculation.

Before any preprocessing techniques are applied, normalization is performed on each frame to adjust the contrast, ensuring that pixel intensities are properly scaled. The first technique that follows is background subtraction using OpenCV’s MOG2 background subtractor. The Mixture-of-Gaussian-based Background/Foreground Segmentation Algorithm is a method used in video analysis to distinguish moving objects from the static background. At its core, MOG2 models each pixel as a mixture of Gaussians and uses an approximation to continuously update the model. In simpler terms, for each pixel in the video, MOG2 keeps track of its variations over a period and tries to decide if the pixel is part of the background or a foreground object. Pixels that remain relatively constant over time are determined to be part of the background, whereas pixels that change frequently are identified as part of moving objects.

A unique aspect of MOG2 is its ability to adapt to changing lightning conditions, ensuring it remains effective even in challenging scenarios. By applying this method, we can effectively filter out static scenes and emphasize dynamic elements, which proves invaluable in applications like motion detection and object tracking.

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. These operations help in refining the mask, ensuring the foreground object of interest is well-isolated. To further enhance the quality and reduce the noise, median blurring was applied. Median blur is especially effective in removing salt-and-pepper noise. Finally, a Gaussian blur was added to smoothen the frames, reducing high-frequency noise and making them ready for optical flow computation.

* 1. **Handling of Labels and Retraining**

During a detailed review of the labelled data, discrepancies were identified in the labels associated with the numpy arrays. The source of these inconsistencies were traced back to an error in the file handling code, leading to incorrect labelling for some of the files. The code was then corrected to ensure the accuracy of the dataset labels.

This discovery necessitated a retraining of the models. It was important to ensure that the models were trained on accurately labelled data to ensure that the results were correct.

* 1. **Down sampling due to imbalance**

Upon analysing the dataset, a significant imbalance was observed between the number of fall and non-fall instances. Specifically, the dataset contained 955 instances of falls, each represented by 2 videos, amounting to a total of 1910 1-second windows of falls. In contrast, the number of non-fall instances were 31281, each represented by 2 videos, amounting to a total of 62562 1-second windows of non-falls.

To address this imbalance, we implemented a down sampling strategy. This involved reducing the number of non-fall samples to match the 1910 fall instances. Down sampling is a technique in data preprocessing to mitigate the impact of class imbalance on model performance. This approach aimed to create a more balanced dataset, which helps in preventing the model from being biased towards the majority class, thereby improving its ability to generalize and accurately predict unseen data.

* 1. **Experiments with Batch Size, Learning Rate and Epochs**

We conducted a series of experiments to further optimize the model’s performance. The three hyperparameters we focused on were: batch size, learning rate, and the number of training epochs. Each hyperparameter was varied independently to isolate its effect on model performance. While adjusting one hyperparameter, the others were held constant to ensure comparability of results. For instance, if the learning rate was modified, the batch size remained at 32 and the number of epochs was fixed at 50. Similarly, when testing a batch size of 16, we maintained the learning rate at 0.0001 and the epoch count at 50.

Initially our models were trained using a standard batch size. We experimented with reducing the batch size by half. This change was intended to provide more frequent updates to the model, potentially leading to a more nuanced learning process. Smaller batch sizes can offer a regularizing effect and often lead to better generalization of the model.

The learning rate was initially set to 0.0001. To explore the effects of a more aggressive learning approach, we increased the learning rate to 0.001. This adjustment was made to examine whether a higher learning rate could accelerate the convergence process, enabling the model to learn faster.

Lastly, we increased the number of training epochs to 100. This was done with the intention of providing the model with more iteration to learn from the data. More epochs could lead to a better-trained model but also carries the risk of overfitting.

1. **Results**
   1. **Metrics**

The metrics used to evaluate the model were Accuracy, Precision, Recall, Specificity, and F1-Score as shown in the descriptions below.

* **Accuracy**

The ratio of correctly predicted instances to the total instances

* **Precision**

The ratio of correctly predicted positives observations to the total predicted positives

* **Recall (Sensitivity)**

The ratio of correctly predicted positives observations to all the actual positives

* **Specificity**

The ratio of correctly predicted negative observations to all the actual negatives.

* **F1-Score**

The weighted average of Precision and Recall

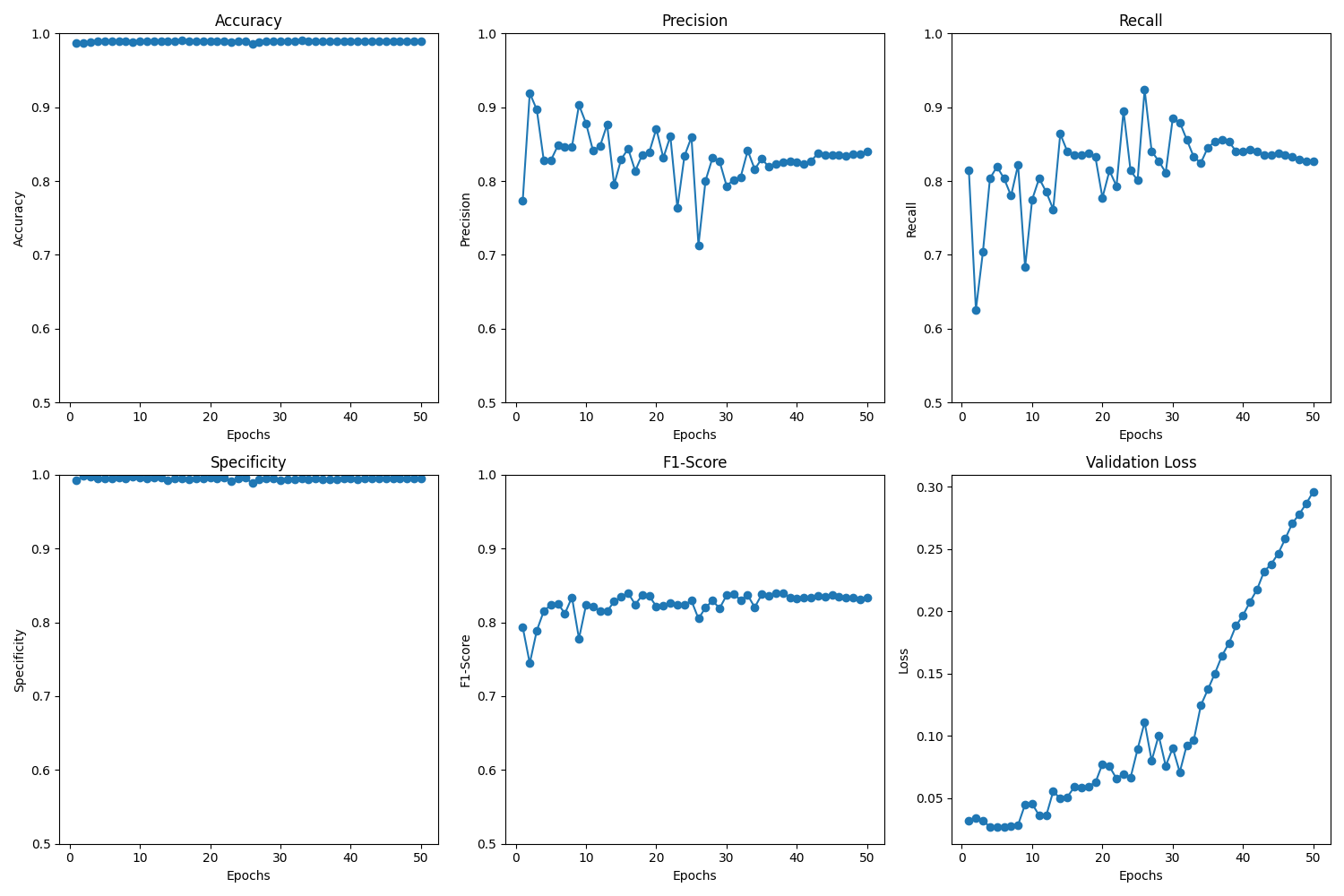
* 1. **2D CNN Results**

**A group of graphs with blue lines

Description automatically generated**

Over the 50 epochs, the 2D CNN model showed a consistent trend. The model achieved high accuracy, reaching a peak of 98.98%. The precision and recall, while oscillating, remained in the 80th percentile. The specificity was consistently high, showcasing the model’s robustness in identifying non-falls. The F1-score fluctuates between 79% and 84.8%.

* 1. **3D CNN Results**

****

Over the 50 epochs, the 3D CNN model displayed a trend with its loss decreasing over 8 epochs before increasing for the rest of the epochs. Despite this, the accuracy remained stable, fluctuating with a narrow margin and achieving a peak of 99.16%. The precision and recall metrics oscillated remaining in 87.2% and 85.16% respectively. The F1-score also varied between 75% and 86%.

* 1. **3D CNN with additional preprocess and updated architecture**

**A graph of a graph

Description automatically generated with medium confidence**

Over the 50 epochs, the second iteration of the 3D CNN model also displays a high amount of loss over the number of epochs. The accuracy fluctuates between 98.3% and 98.8%. The precision and recall metrics also fluctuated remaining in the 80%. The F1-score varied between 73% and 80%.

**A blue squares with white text

Description automatically generated**

* 1. **3D CNN with balanced dataset**

A graph of a graph

Description automatically generated with medium confidence

Over the 50 epochs, the 3D CNN on the balanced dataset showed a big improvement over the previous iterations. The accuracy starts off around 64% and goes up to 96% through 50 epochs. The precision and specificity start off at 100% but drops and stays around 96% as well. The recall and F1-score also hover around 96% throughout the training.

A blue squares with white text

Description automatically generated

* 1. **3D CNN with balanced dataset, batch size 16**

A graph of data on a white background

Description automatically generated with medium confidence

Over the 50 epochs, the accuracy hovers around 97% with a large drop around epoch 27 and 40. The precision also hovers around 97% with large drops around epoch 19. The recall also has large drops at epoch 28 and 40, ending at 97%. Specificity achieved the same result with large drops at epoch 19. F1-score also achieved the same score with large drops at epoch 28 and 40. Validation loss remained constant, initially decreasing for a few epochs before rising and dropping again. The largest increase in validation loss can be seen at epoch 28 and 38.

A blue squares with white text

Description automatically generated

* 1. **3D CNN with balanced dataset, learning rate 0.001.**

**A graph of a graph

Description automatically generated with medium confidence**

For the training with a learning rate of 0.001, the accuracy hovers around 95%. Precision, recall, and specificity stayed constant outside of a couple bigger drops, oscillating between 85% and 98%. F1-score hovered between 92% and 97%. Validation loss was more erratic but with a general increasing trend.

**A blue squares with white text

Description automatically generated**

* 1. **3D CNN with balanced dataset, 100 epochs**

A graph of blue lines

Description automatically generated with medium confidence

When training over 100 epochs, the accuracy mostly hovers in the 90% but there were 3 large drops. The rest of the metrics also hovered around 96% with a couple large drops in some epochs. Validation loss stayed low throughout the 100 epochs.

A blue squares with white text

Description automatically generated

1. **Evaluation** 
   1. **Comparative Analysis (Update)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Data** | **Accuracy** | **Precision** | **Recall** | **Specificity** | **F1-Score** |
| **Research** | **95.64** | **96.91** | **97.95** | **83.08** | **97.43** |
| **Replication** | **98.98** | **84.14** | **81.94** | **99.51** | **83.02** |
| **3D CNN** | **99.16** | **87.20** | **85.16** | **99.60** | **86.17** |
| **3D CNN with preprocess and updated architecture** | **99.04** | **84.92** | **83.59** | **99.53** | **84.25** |
| **3D CNN with balanced dataset** | **96.48** | **96.85** | **96.09** | **96.87** | **96.47** |
| **3D CNN with balanced dataset batch size 16** | **96.74** | **97.11** | **96.35** | **97.13** | **96.73** |
| **3D CNN with dataset learning rate 0.001** | **93.87** | **97.73** | **89.84** | **97.91** | **93.62** |
| **3D CNN with dataset 100 epochs** | **96.61** | **96.13** | **97.14** | **96.08** | **96.63** |

The table above highlights the performance of each model. The replication performs better in accuracy and specificity but underperforms in the rest of the metrics. Transitioning to a 3D CNN with the same architecture as the 2D CNN increased the overall performance. While accuracy and specificity stayed the same, precision, recall and f1-score all increased by 3%.

After adding more preprocessing techniques to further enhance the optical flow, the model did not improve and performed 2% worse in precision, recall, and f1-score. After performing down sampling, the performance increased drastically. Precision, recall, and specificity all reached 96%, while accuracy and specificity dropped to around 96.5%.

Decreasing the batch size by half did not improve the model by much. Roughly a 0.3% increase in each metric. Increasing the learning rate to 0.001 decreased the accuracy and recall by around 6% each. F1-score also decreased by 3%. Finally, training the model over 100 epochs did not yield significant changes as well.

* 1. **New results**

For this set of results, batch size was kept at 32 and epoch number at 50.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Learning Rate** | **Accuracy** | **Precision** | **Recall** | **Specificity** | **F1-Score** |
| **0.001** | **97.32** | **19.25** | **95.31** | **97.34** | **32.04** |
| **0.0001** | **94.61** | **10.80** | **98.44** | **94.58** | **19.46** |
| **0.00001** | **98.20** | **26.02** | **93.23** | **98.23** | **40.68** |

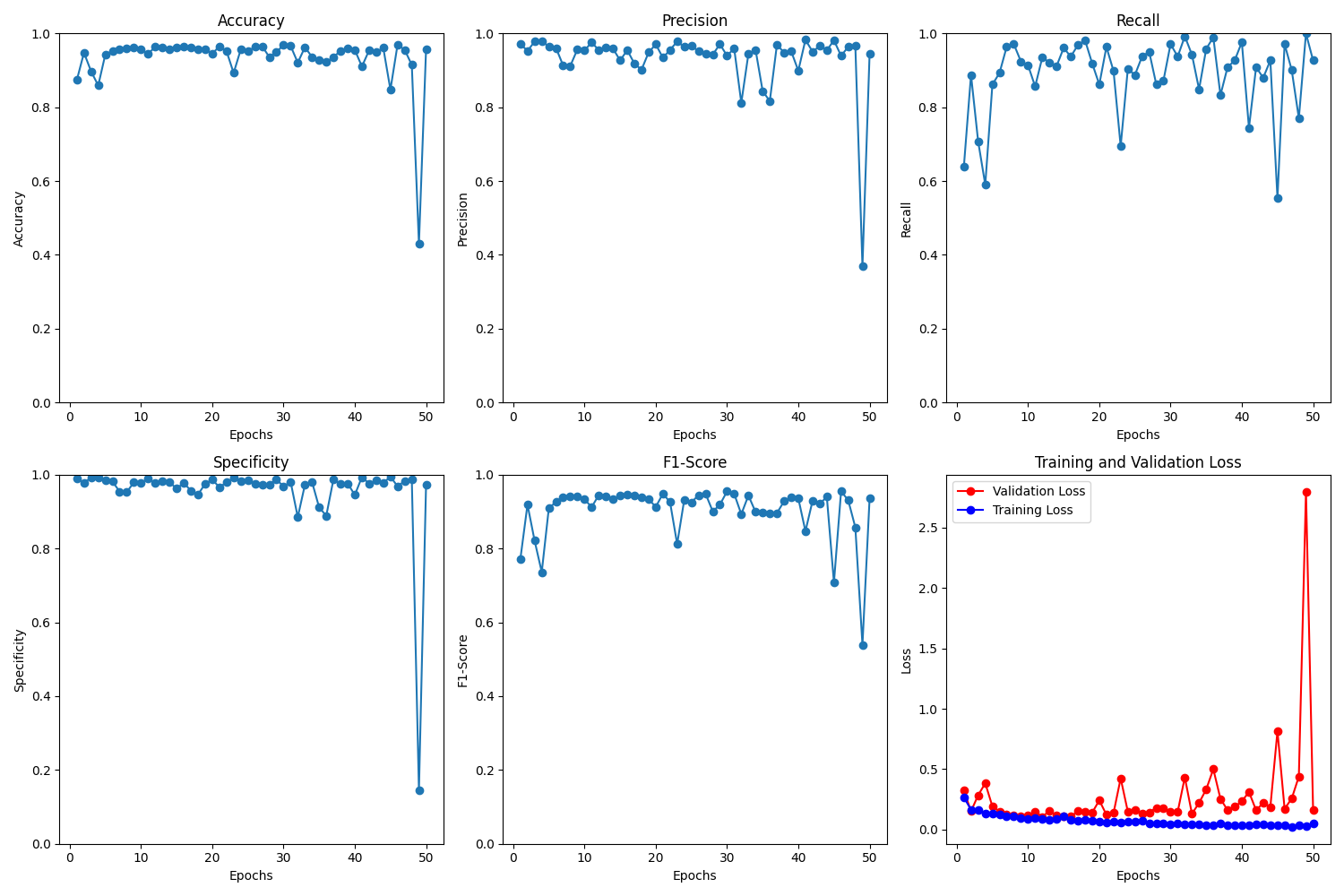


Figure Learning Rate 0.001

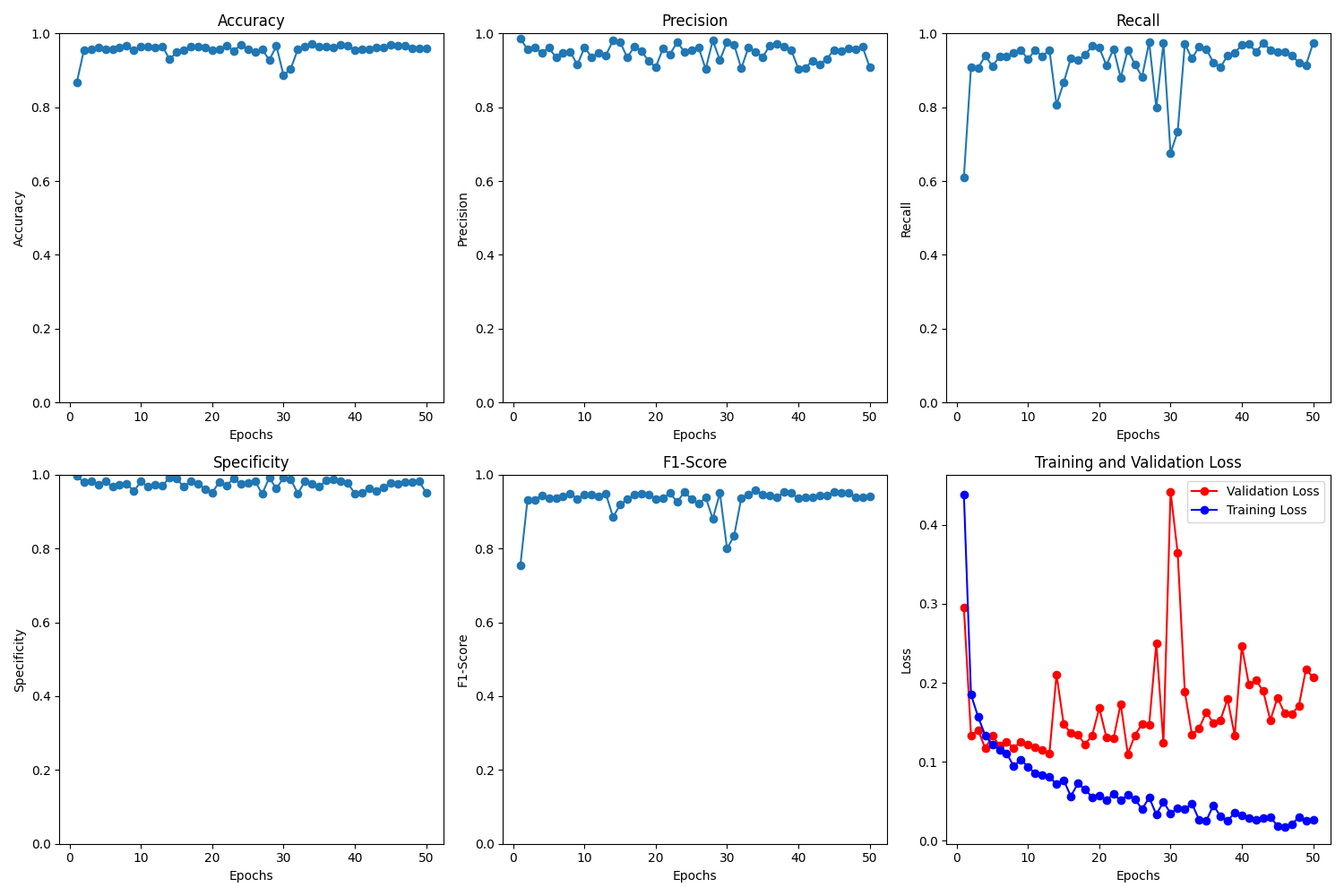


Figure Learning Rate 0.0001

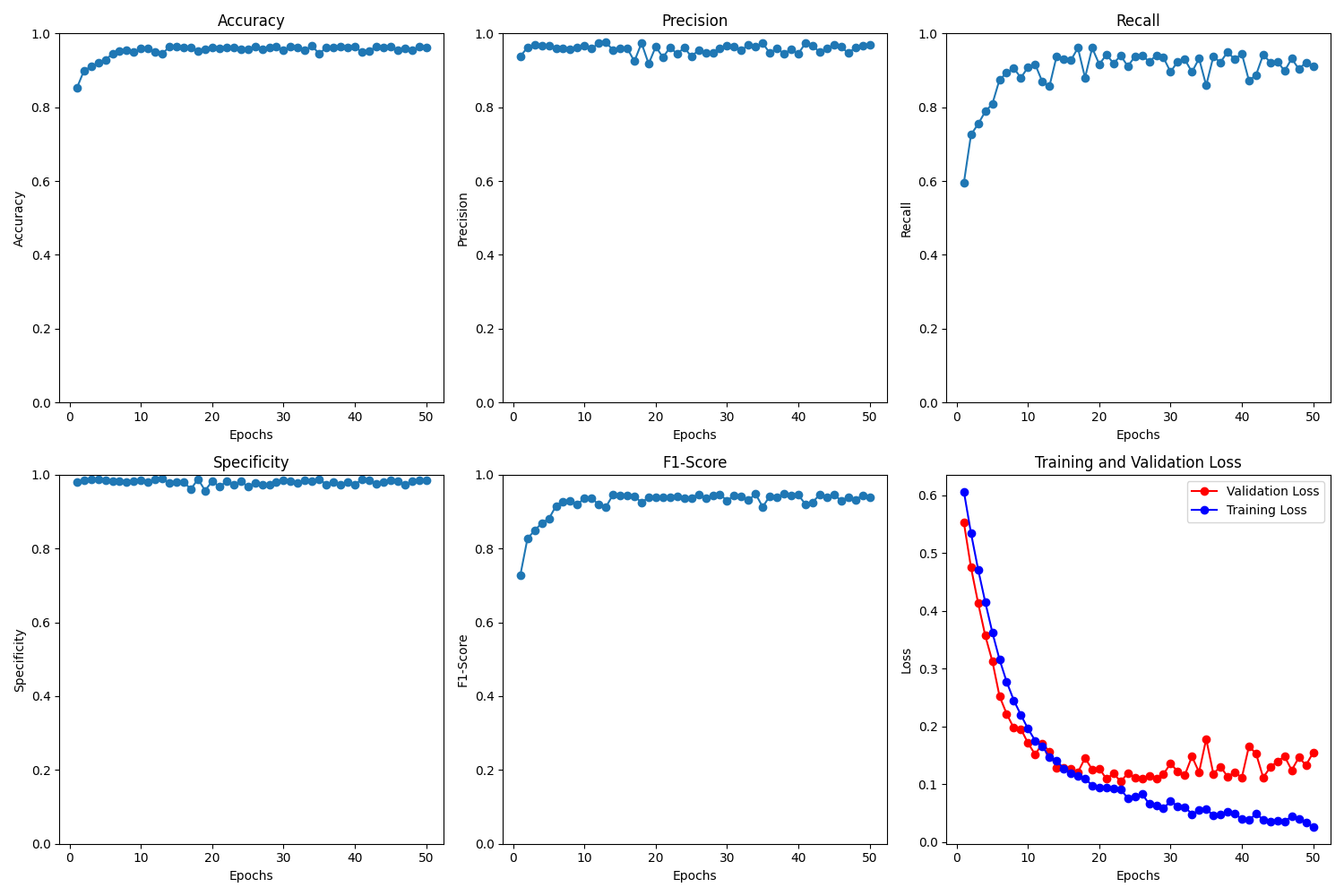


Figure Learning Rate 0.00001

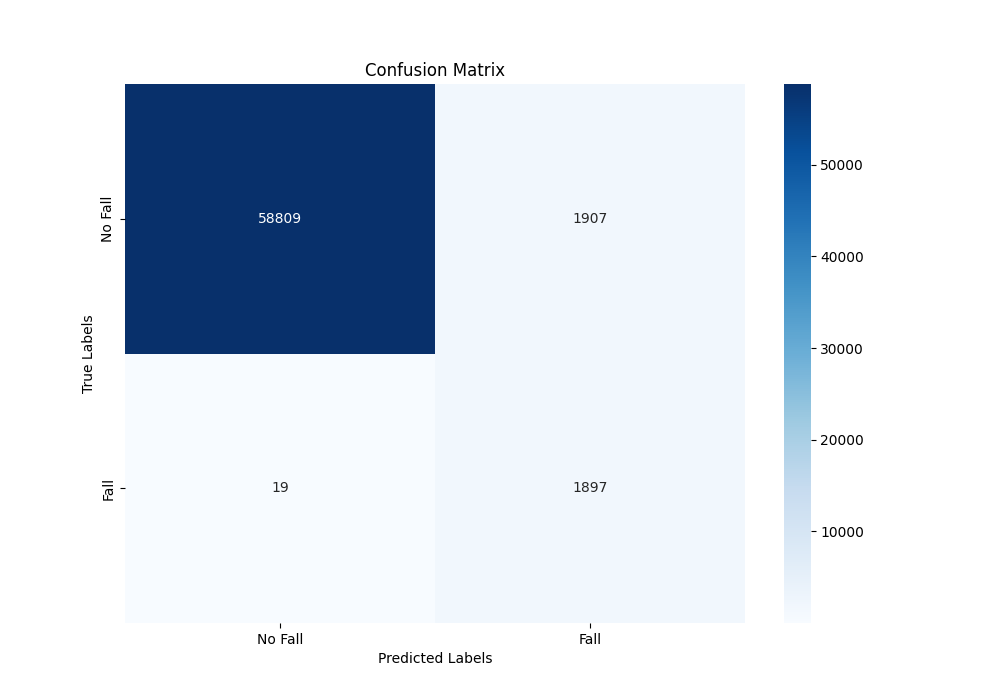


Figure Learning Rate 0.001

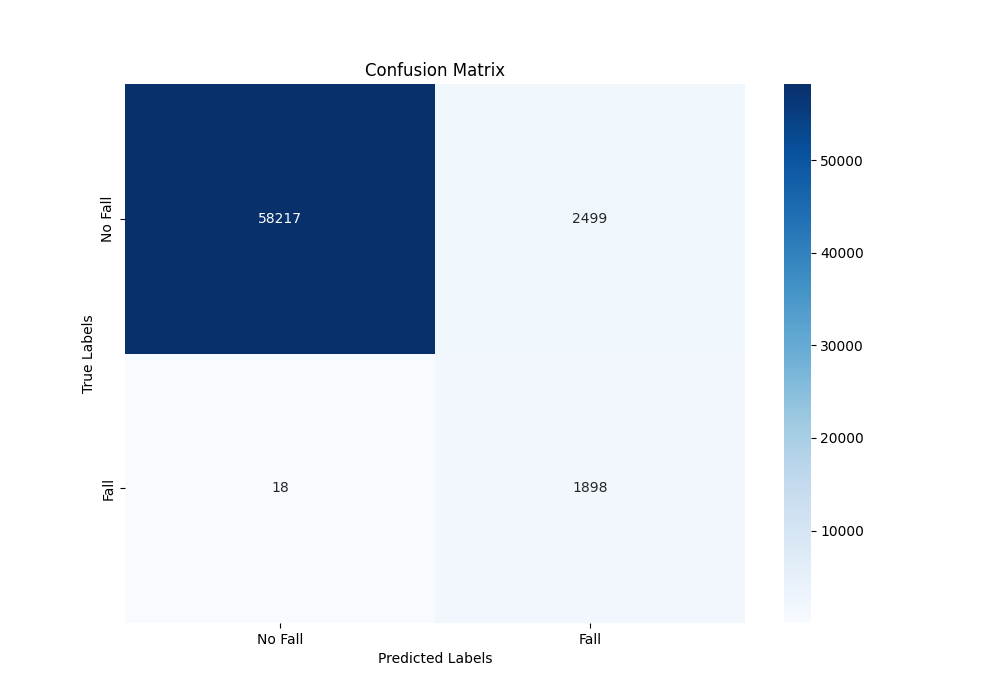


Figure Learning Rate 0.0001

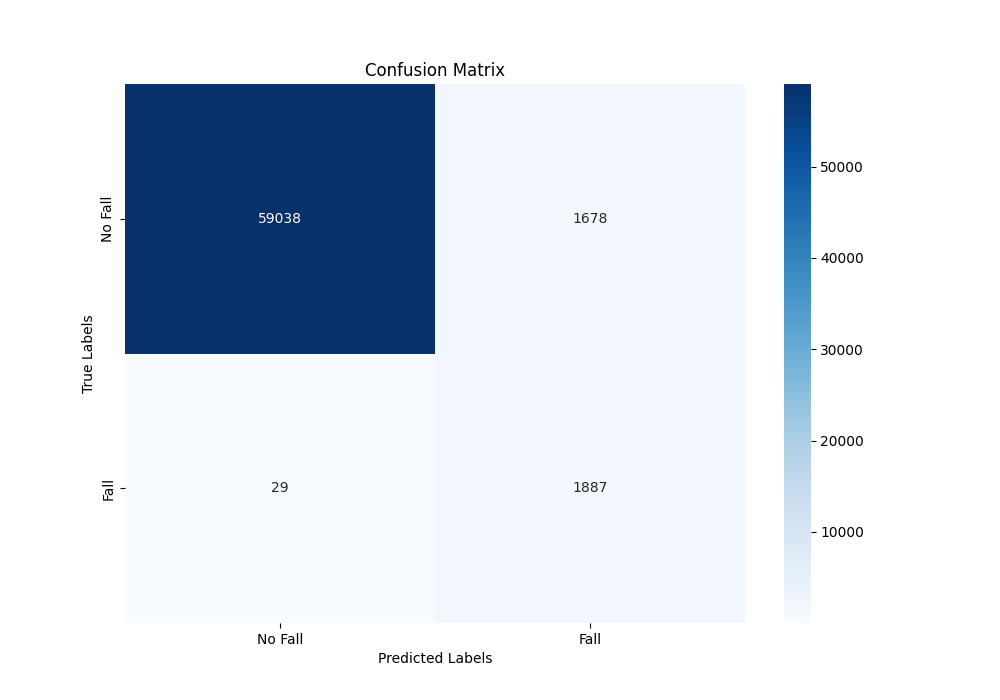


Figure Learning Rate 0.00001

For the learning rate, the graph for validation loss was a big indicator of its performance. The learning rate at 0.00001 had a graph that showed the algorithm was learning at a stable rate.

For this set of results, learning rate was kept at 0.00001 and epoch number at 50.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Batch Size** | **Accuracy** | **Precision** | **Recall** | **Specificity** | **F1-Score** |
| **8** | **97.82** | **22.58** | **94.27** | **97.85** | **36.44** |
| **16** | **97.99** | **24.11** | **95.05** | **98.01** | **38.46** |
| **32** | **98.20** | **26.02** | **93.23** | **98.23** | **40.68** |
| **48** | **96.75** | **16.40** | **95.57** | **96.75** | **27.99** |

A group of graphs showing different types of data

Description automatically generated with medium confidence

Figure Batch Size 8

A graph of a graph

Description automatically generated with medium confidence

Figure Batch Size 16

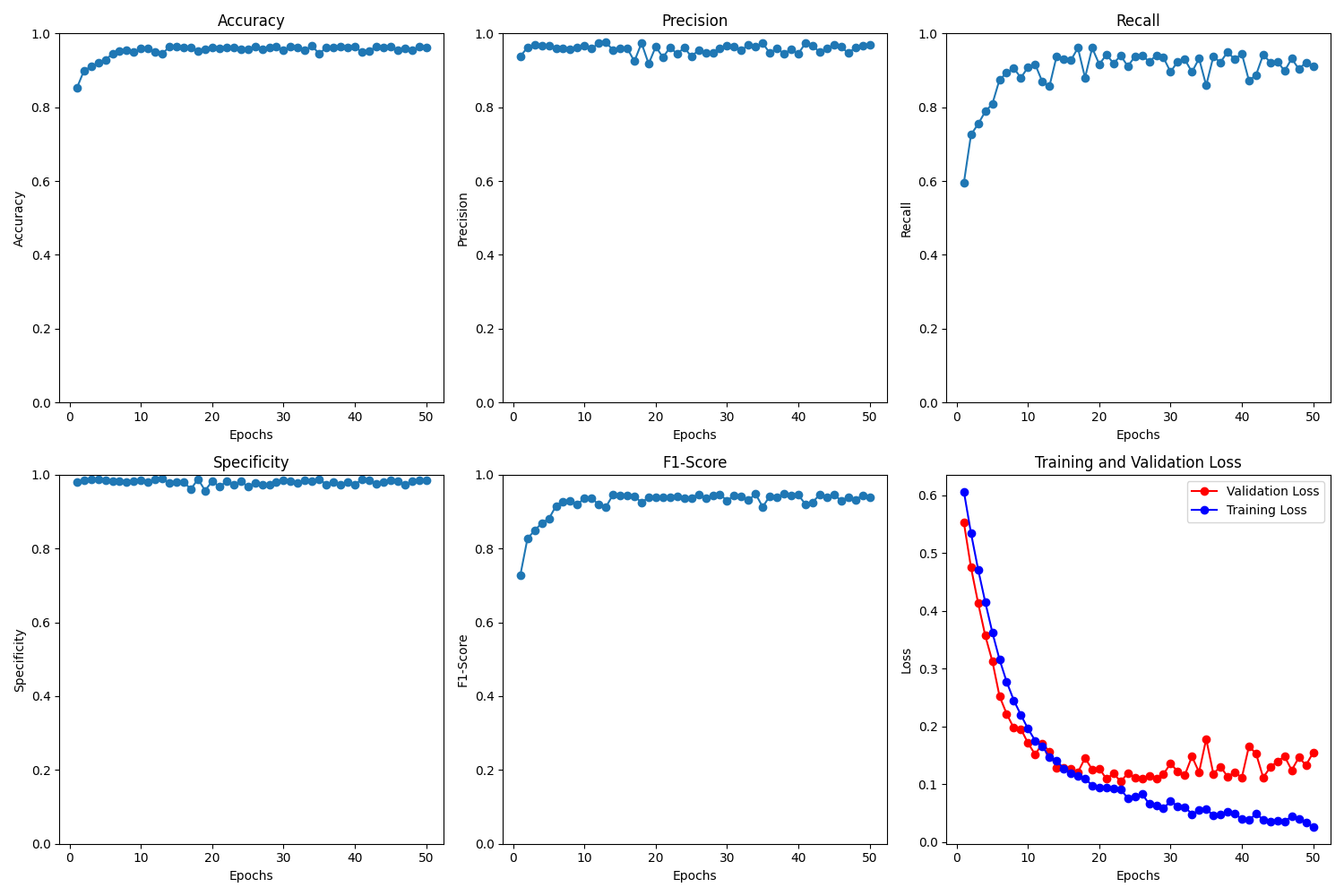


Figure Batch Size 32

A graph of a graph

Description automatically generated with medium confidence

Figure Batch Size 48

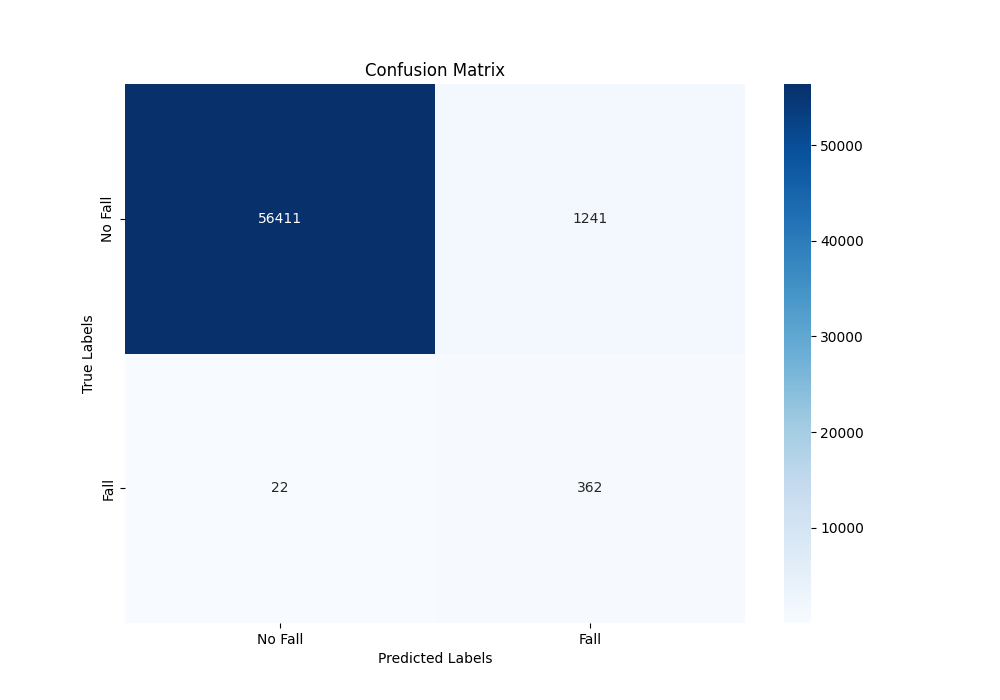


Figure Confusion Matrix Batch Size 8

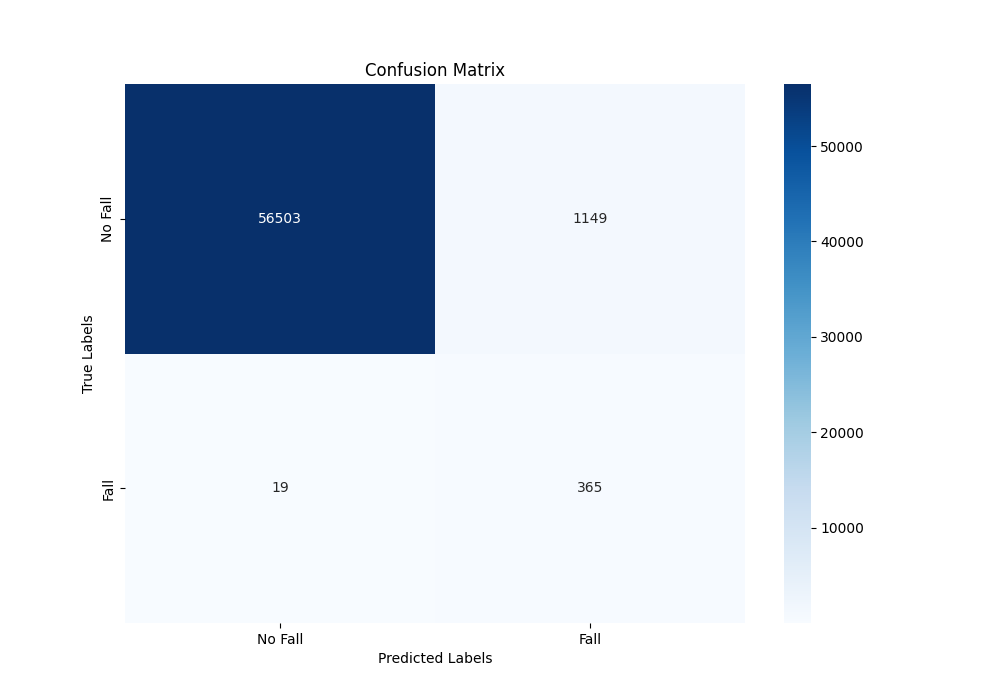


Figure Confusion Matrix Batch Size 16

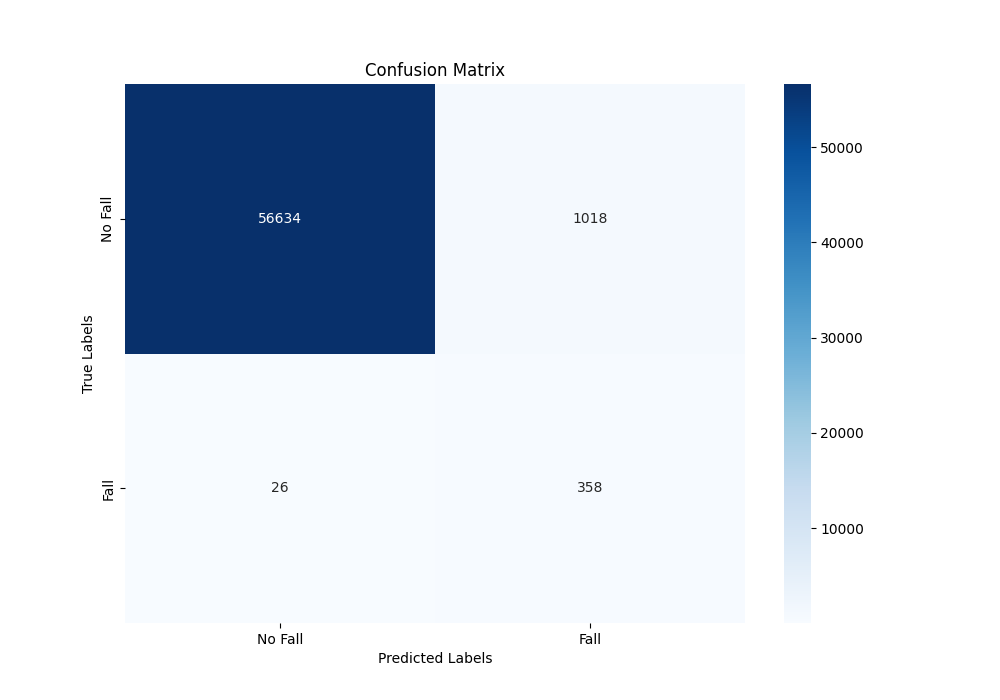


Figure Confusion Matrix Batch Size 32

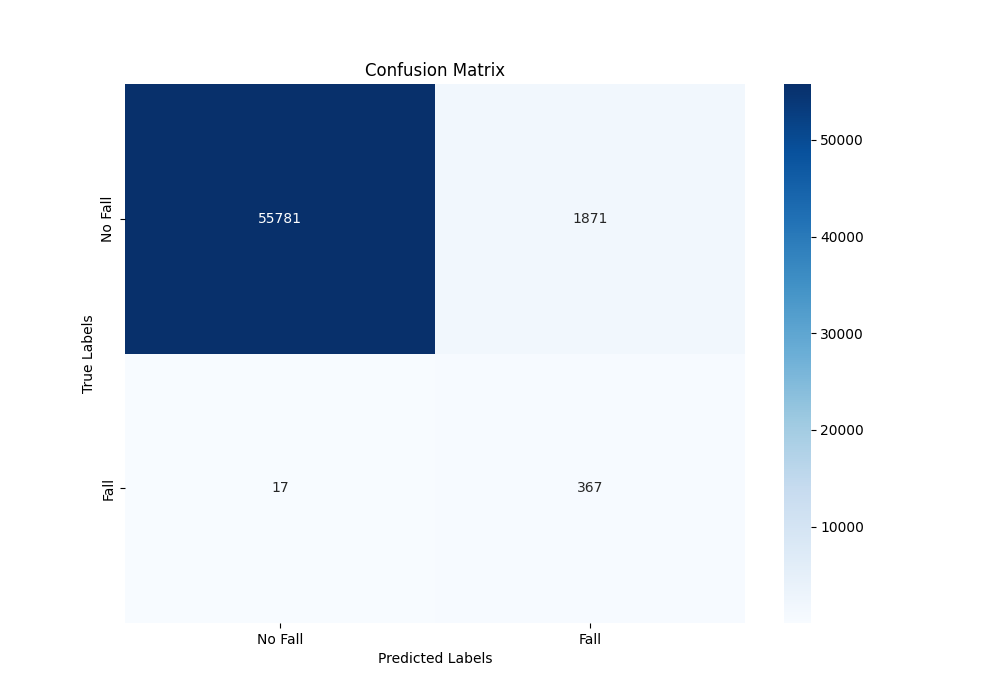


Figure Confusion Matrix Batch Size 48

Based on the above graphs and confusion matrices, it seems that a batch size of 32 offers the right balance between precision and recall. This can also be seen in the graphs where the validation loss is less erratic for a batch size of 32.

We further experimented with different resolutions. The original pipeline resized it to 58x31, we multiplied that by 1.5 and 2 to compare its performance.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Resolution** | **Accuracy** | **Precision** | **Recall** | **Specificity** | **F1-Score** |
| **58x31** | **98.20** | **26.02** | **93.23** | **98.23** | **40.68** |
| **76x57** | **98.20** | **25.71** | **91.41** | **98.24** | **40.14** |
| **102x76** | **97.22** | **18.70** | **95.57** | **97.23** | **31.27** |

As seen in the graph and table above, there are not many changes in the performance when increasing the resolution.

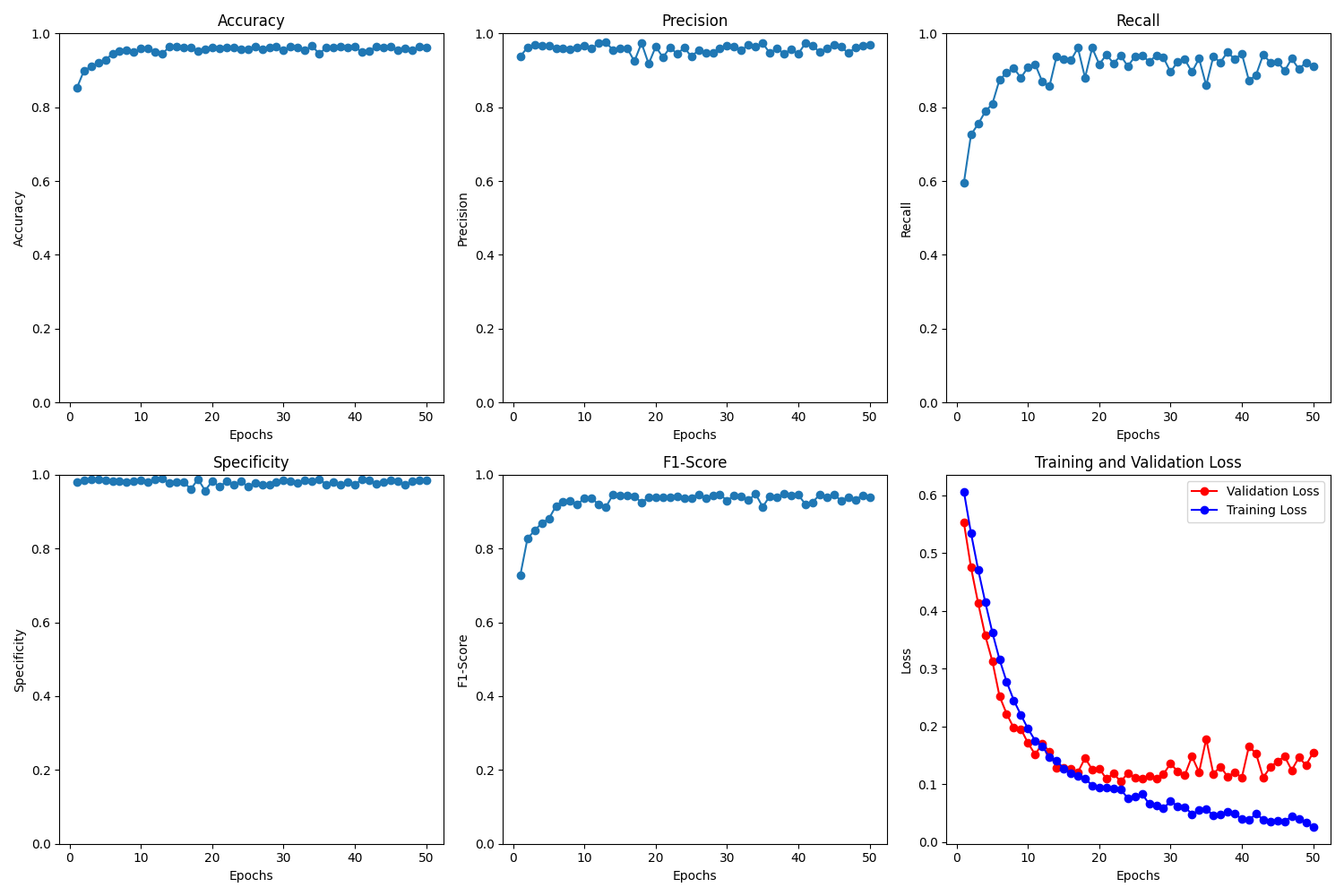


Figure Resolution 58x31

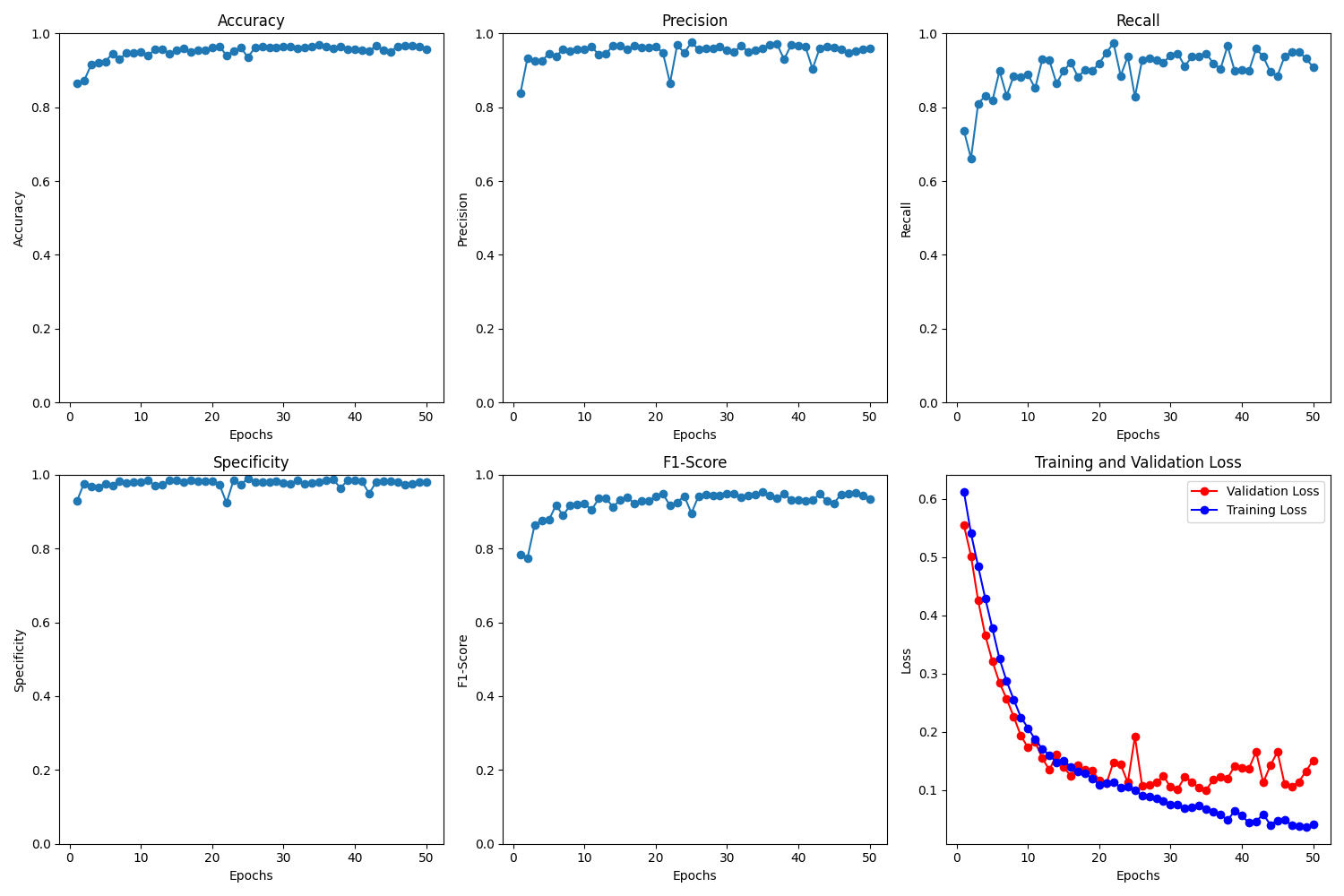


Figure Resolution 76x57

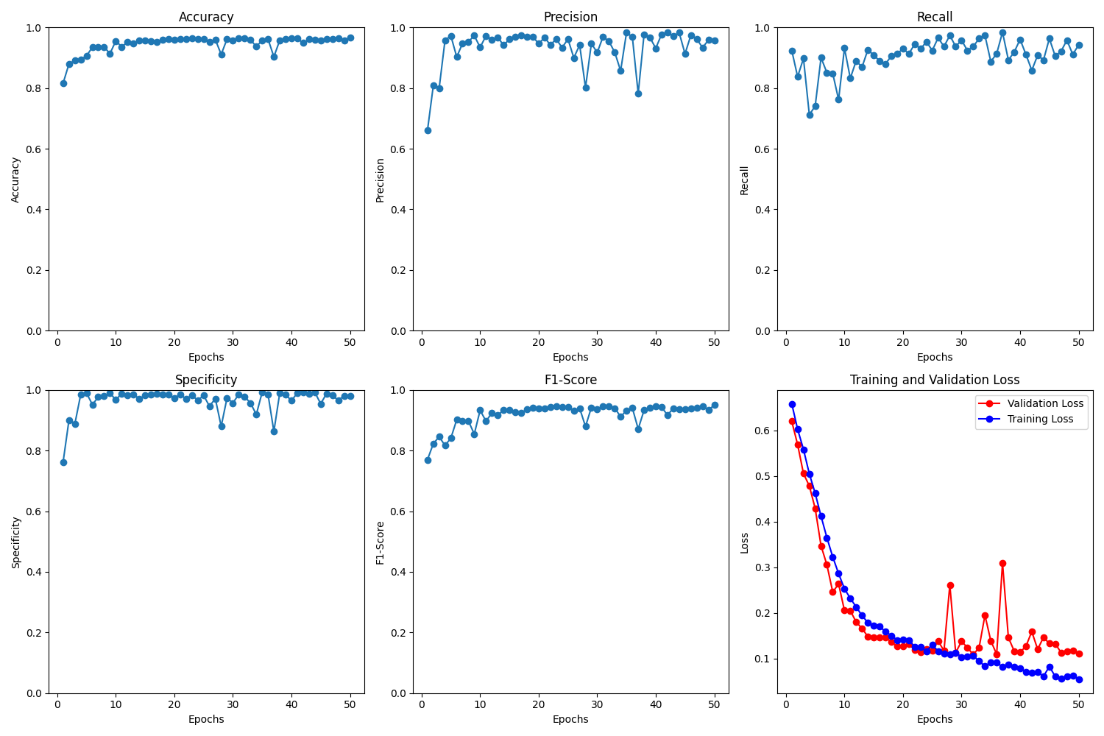


Figure Resolution 102x76

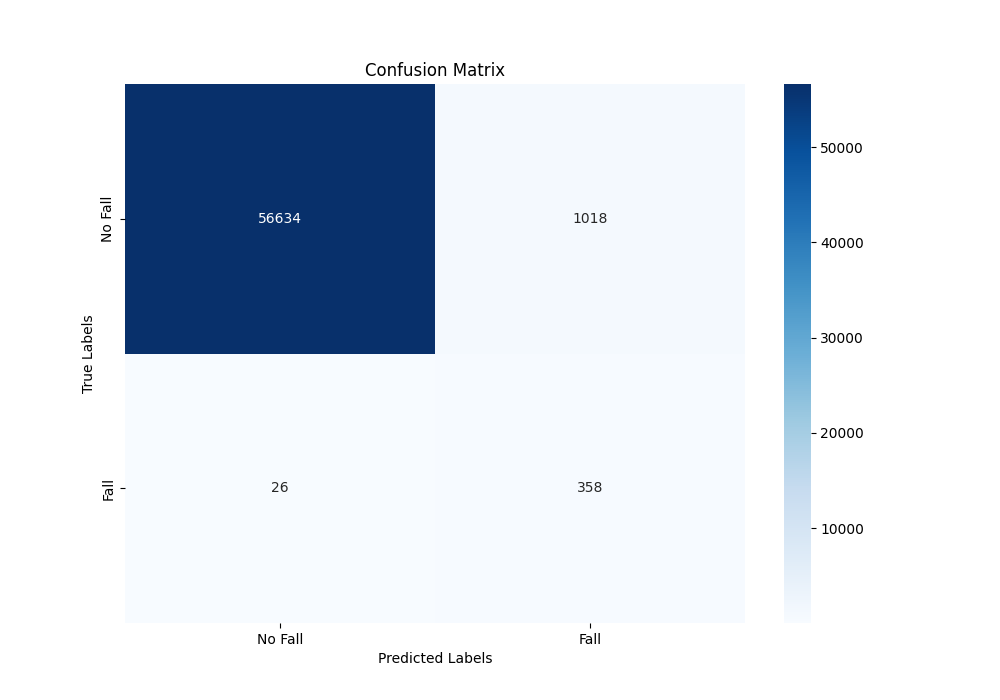


Figure Confusion Matrix 58x31

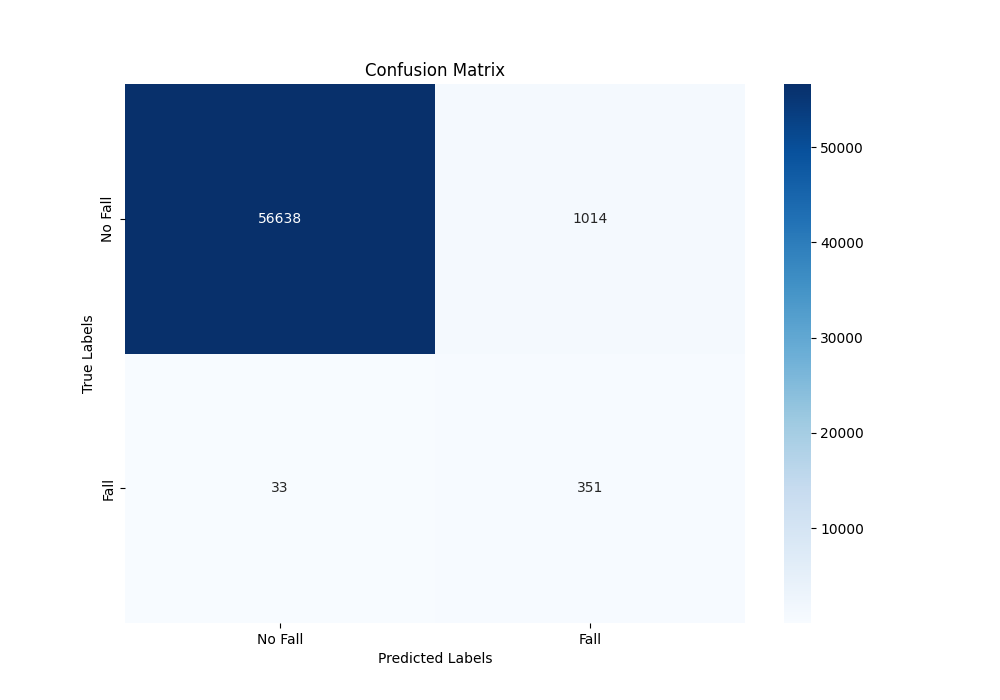


Figure Resolution 76x57

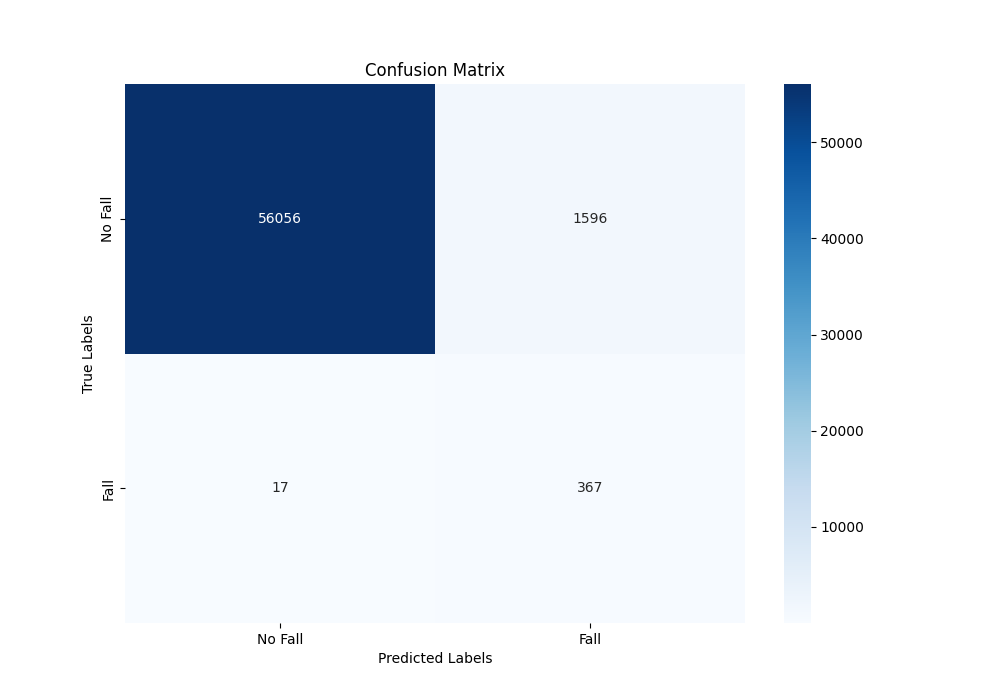


Figure Resolution 102x76

While increasing resolution could offer more details that the algorithm could learn from, it seems like the model does not improve much as the resolution increases.

To decrease the preprocessing time for the entire dataset and potentially increasing the performance of the system in the real world, the videos in the dataset was first resized to 102x76 then optical flow was calculated. The time it took to preprocess the entire dataset decreased from 15 hours to 1 hour and 15 minutes.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Resize** | **Accuracy** | **Precision** | **Recall** | **Specificity** | **F1-Score** |
| **After** | **97.22** | **18.70** | **95.57** | **97.23** | **31.27** |
| **Before** | **98.30** | **25.45** | **81.25** | **98.41** | **38.76** |

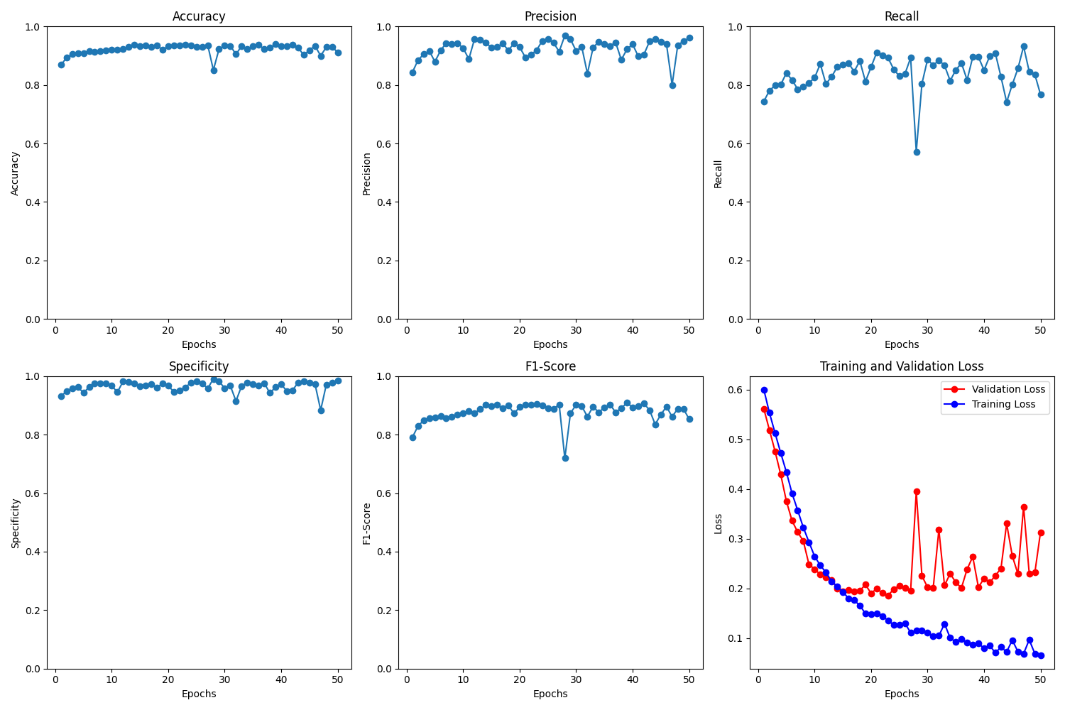


Figure Resize before optical flow

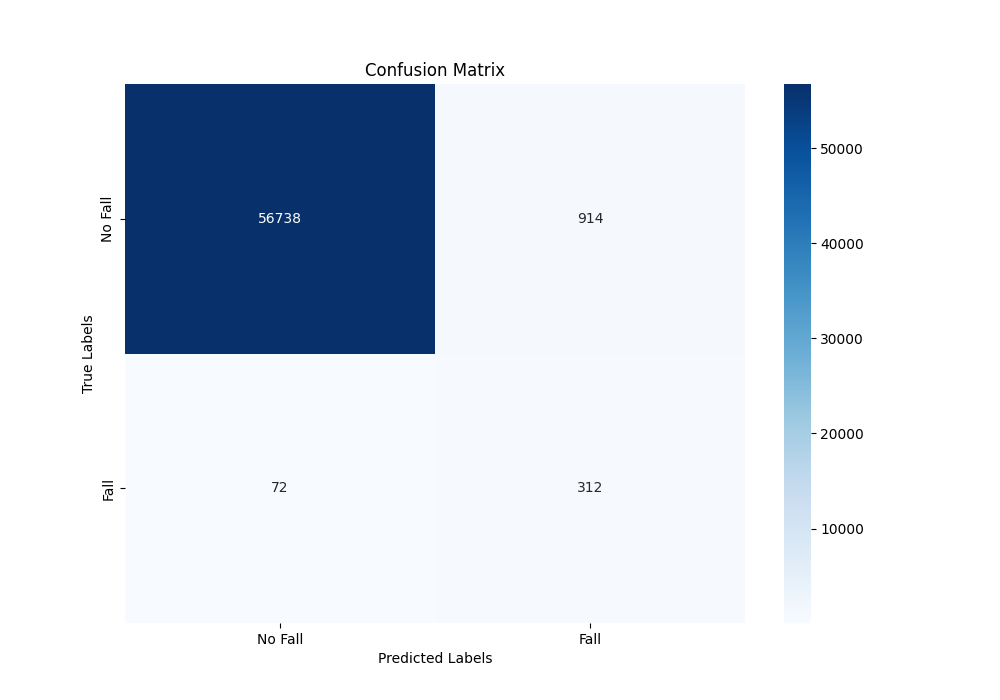


Figure Confusion matrix

As seen in the table and graphs above, resizing before calculating optical flow did not yield better results. Accuracy, Specificity, and F1-Score remained similar while Precision was slightly better. On the other hand, Recall fell by around 14%.

Based on the results, resizing before calculating optical flow may have reduced details that optical flow can calculate which could explain the slightly worse performance, however the increase in speed of preprocessing is a great improvement. Further investigation may be needed to find a good balance between increasing efficiency while having good results.

* 1. **Potential Reasons for Results**

The first reason for the difference between the research paper and the replication may be the way optical flow is calculated. We used Farneback method instead of the Horne and Schunck method which will affect the results. Secondly, the research paper does not provide information regarding the kernel size of their convolutional layers which would affect how our model performs.

After a thorough examination of the optical flows across various frames, it was observed that certain frames exhibit a high level of noise. This is likely to negatively impact the training process.

A black background with colorful spots

Description automatically generated

Additionally, a deeper dive into the dataset revealed an imbalance: there are 955 one-second windows containing falls, and 31,281 one-second windows without any falls. Such a disparity is a probable cause for the diminished precision, recall, and F1-score that were noted. The initial imbalance likely caused the model to predict the majority class more often, which explains the high accuracy. The bias was mitigated by balancing the dataset. Balancing also increased precision and recall which means that the model has become more reliable in predicting falls.

1. **Future improvements**
   1. **Model Improvements**

One area to explore in model enhancement is the depth and architecture of the neural network. By experimenting with the addition or reduction of convolutional layers, we can better understand the balance between model complexity and performance. Incorporating other types of layers could also prove beneficial. For instance, introducing dropout layers might mitigate overfitting, while batch normalization layers could expedite the training process. Within the convolutional layers itself, the kernel size is also another factor for experimentation.

The choice of activation functions is another area we can experiment with. While ReLU is widely used and recognized for its efficiency, there are alternative functions such as Leaky ReLU, Parametric ReLU, and Explonential Linear Unit that might offer subtle advantages.

Another aspect for experimentation is the selection of optimizers and learning rate strategies. The model currently uses an Adam optimization technique, but others like gradient descent, or Adagrad could affect model performance. Instead of employing a fixed learning rate, considering adaptive learning rate could facilitate faster and more stable model convergence.

Lastly, the ensemble technique is another option we could explore. Ensemble techniques offer a promising avenue for boosting model performance by combining predictions from multiple models to produce a final decision. The fundamental idea behind ensemble learning is that a group of “weak learners” can come together to form a “strong learner”. By leveraging the strengths of multiple models and minimizing individual weaknesses, ensemble methods can lead to increased accuracy, reduced overfitting, and improved generalization to unseen data.

* 1. **Expanding Dataset**

Our current methodology predominantly focuses on the UP-Fall detection dataset, with minimal preprocessing done on the video data prior to optical flow computation. While a degree of noise reduction has been applied, there’s potential to further refine the preprocessing steps. Improved video clarity can subsequently yield more accurate optical flow computations, which is pivotal for the model’s performance.

However, basing our results and conclusions solely on one dataset might not provide a holistic view of the model’s capabilities. Diversifying the data sources would allow the model to be exposed to various fall patterns, lighting conditions, and environments, thereby making It more robust. In addition to the UP-Fall dataset, we should consider incorporating other publicly available fall detection datasets or even curating our own dataset tailored to the specific scenarios we aim to address.

To truly gauge the model’s performance and generalization capabilities, it’s crucial to test it on an unseen dataset. This will provide a more realistic assessment of how the model might perform in real-world applications, away from the controlled conditions of the training data. Evaluating the model on and external dataset can also highlight areas where the model might be lacking, offering insights into potential refinements and improvements.

Furthermore, balancing the dataset by down sampling decreased the amount of data that was fed into the model. Originally it had 955 falls and 31281 non falls. After balancing, there are 955 falls and 955 non falls. Techniques such as data augmentation could be used to artificially expand the dataset, potentially improving the model’s robustness.

* 1. **Experiment with other Optical Flow Methods**

The current method of calculating optical flow is different from the paper’s original method. Switching to the Horne and Schunck method may allow us to replicate the results of the paper. Using another method may also improve the performance in 3D CNNs.