ASSIGNMENT COVER SHEET

UTS: ENGINEERING & INFORMATION TECHNOLOGY

SUBJECT NUMBER & NAME	NAME OF STUDENT(s) (PRINT CLEARLY) SURNAME FIRST NAME	STUDENT ID(s)
31256 Image Processing and Pattern Recognition - Spring 2024	*Kusumo, Valerie* *Chen, Ching-Ting* *Chen, Kevin* *Kim, Yoonsu* *Hua, Brendan* *Le, Nam*	24870572 14421592 14263002 14470292 24477541

STUDENT EMAIL	STUDENT CONTACT NUMBER
Valerie.Kusumo@student.uts.edu.au ching-ting.chen@student.uts.edu.au yuanjun.chen@student.uts.edu.au yoonsu.kim@student.uts.edu.au brendan.hua@student.uts.edu.au nam.le-1@student.uts.edu.au	+61 431 670 566 0477150351 0470244283 0401 785 832 0411 535 881

NAME OF TUTOR	TUTORIAL GROUP	DUE DATE
Stuart Perry	Group 13	1 Nov by 23:59

ASSESSMENT ITEM NUMBER & TITLE
41181 Image Processing and Pattern Recognition Assignment 2

- ☑ I acknowledge that if AI or another nonrecoverable source was used to generate materials for background research and self-study in producing this assignment, I have checked and verified the accuracy and integrity of the information used.
- ☑ I confirm that I have read, understood and followed the guidelines for assignment submission and presentation on page 2 of this cover sheet.
- ☑ I confirm that I have read, understood and followed the advice in the Subject Outline about assessment requirements.
- ☑ I understand that if this assignment is submitted after the due date it may incur a penalty for lateness unless I have previously had an extension of time approved and have attached the written confirmation of this extension.

Declaration of originality: The work contained in this assignment, other than that specifically attributed to another source, is that of the author(s) and has not been previously submitted for assessment. I have rewritten any material provided by AI or other nonrecoverable sources and where appropriate acknowledged their contribution. I understand that, should this declaration be found to be false, disciplinary action could be taken and penalties imposed in accordance with University policy and rules. In the statement below, I have indicated the extent to which I have collaborated with others, whom I have named.

No content generated by AI technologies or other sources has been presented as my own work and I have rewritten any text provided by AI or other sources in my own words.

Statement of collaboration: we agree to collaborate with one another fair and square
Signature of student(s)
Kevin Chen **Date**

Signature of student(s) Nam Le Date

Signature of student(s) Ching-Ting, Chen Date

Signature of student(s) Date

Signature of student(s) Brendan Hua, Date

Signature of student(s) yoonsu KIM Date

ASSIGNMENT RECEIPT

To be completed by the student if a receipt is required

SUBJECT NUMBER & NAME	NAME OF TUTOR
31256 Image Processing and Pattern Recognition - Spring 2024	Stuart Perry
SIGNATURE OF TUTOR	**RECEIVED DATE**

Assessment Task 2: Image Processing and Pattern Recognition

Project Implementation

(Pose Recognition)

By: **Group 13**

Member:
Ching-Ting Chen 14421592
Valerie Kusumo 24870572
Nam Le 25089665
Kevin Chen 14263002
Yoonsu Kim 14470292
Brendan Hua 24477541

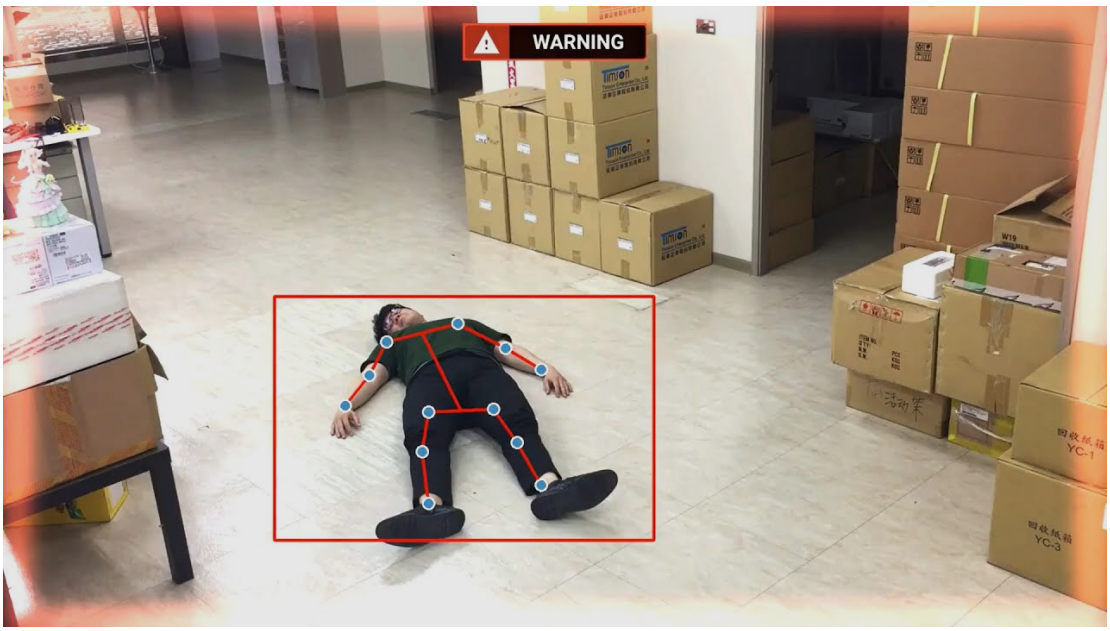

Table of Content

1. Introduction

Falls are a significant health risk among elderly individuals, with one in four Australians aged 65 and older experiencing at least one fall annually (Healthdirect, n.d.). These incidents can result in serious injuries and long-term health complications if not promptly detected and addressed. However, current fall detection systems, primarily reliant on wearable devices or surveillance cameras, have significant limitations that reduce their effectiveness. Wearable devices, for instance, require consistent usage and regular maintenance, which can be challenging for elderly users who may forget to wear them or may not keep them charged. Additionally, these devices often produce false positives by detecting sudden movements that are not necessarily falls (Stone & Skubic, 2015).

In contrast, surveillance systems provide continuous monitoring but face challenges in complex environments with multiple people or noisy backgrounds. Factors such as occlusion, variations in lighting, and high computational demands can reduce real-time detection accuracy, especially in scenarios involving multiple subjects (Del Rosario et al., 2015). These limitations in existing fall detection systems underscore the need for a more adaptable and non-intrusive approach. This project aims to address these gaps by using an image-based pose detection model, such as YOLOPose, for identifying key body postures, and a classifier like Random Forest to categorise detected poses into categories such as "standing," "sitting," or "lying down." By focusing on pose-specific detection rather than abrupt movements, this approach aims to enhance accuracy and minimise false positives across various settings.

2. Project Overview and Methodologies

2.1 Techniques Used

2.1.1 Random Forest (Classifier)

Random forest is a machine learning algorithm that creates multiple decision trees which uses different random subsets of the data and features. The predictions are made by calculating the prediction for each tree and then takes the most voted/popular result (majority voting). Through majority voting, it can effectively reduce error and achieve the highest accurate prediction.

It is particularly useful for this project as it’s the most efficient (comparing to models such as Neural Networks) and effective (comparing to models such as decision tree) algorithms that can be used to classify objects into different poses. Due to time constraints, random forests require less computational power and time to train, which are suitable for real-time applications with limited hardware resources. Additionally as the team’s dataset wasn’t large, it can work effectively with smaller datasets in comparison to Neural Networks, which generally perform better with large amounts of labelled data.

Data is split into training and testing sets the model trains on the training data, learning the relationships between landmark features and the pose labels. During training, each decision tree in the forest uses random sampling and learns a unique view of the data, which helps the forest capture diverse patterns and reduce the risk of overfitting. After training, the Random Forest model makes predictions on the test data.

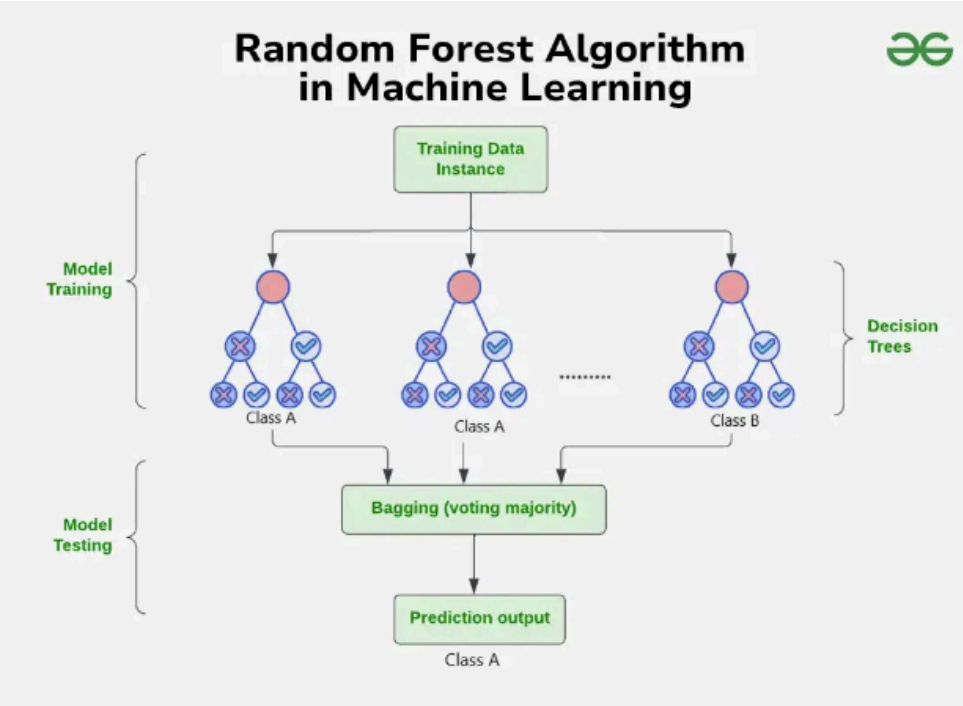

Figure 1: Random Forest Architecture (GeeksforGeeks, 2024)

2.1.2 Feedforward Neural Network (Classifier)

A Feedforward Neural Network (FNN) is a type of artificial neural network in which information flows in one direction without any cycles or feedback loops. The input layer receives the raw data in this case, the pose landmarks, hidden layers process the information, and the output layer generates the final predictions.

The FNN in this implementation consists of four layers.

1. Input layer with 128 neurons, corresponding to the number of input features (landmarks). This layer uses the ReLU activation function to introduce non-linearity, helping the network learn complex patterns in the data.
2. Hidden layer with 64 neurons, also using ReLU activation to process the data further.
3. Second hidden layer with 32 neurons continues to refine the information through ReLU activation.
4. Output layer has 3 neurons equal to the number of pose classes "standing", "sitting", and "lying down". The softmax activation function is applied at this stage, converting the output into class probabilities, making it suitable for multi-class classification.

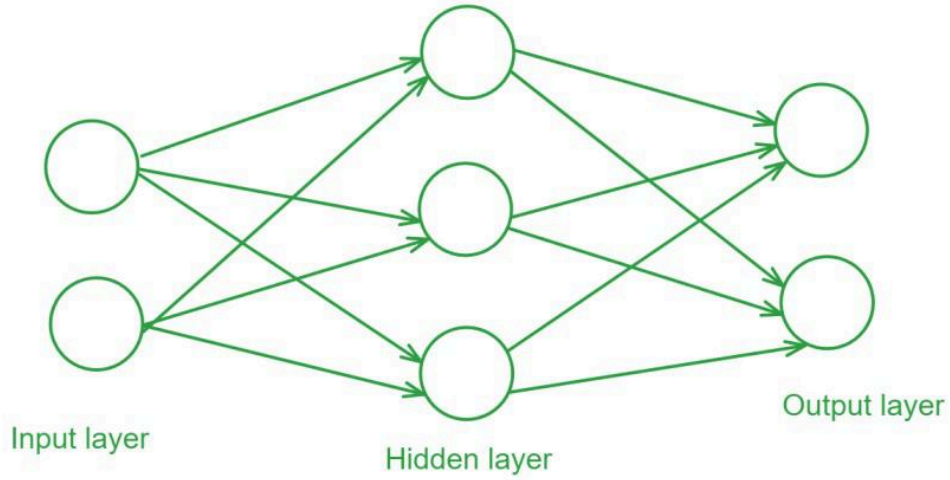

Figure 2: Feedforward Neural Network (GeeksforGeeks, 2024)

2.1.3 YoloPose (Detector)

YoloPose is a pose-estimation model that utilises YOLO (you only look once) as a backbone to perform estimating pose (*Refer to figure 1).* YoloPose uses the regression method to directly predict keypoint that already associate with the bounding box that are generated by the YOLO model. The YOLO model is great due to its speed and limited use of resources, making it one of the best pose estimations to use in real-life scenarios.

YoloPose is particularly suitable for the purpose of the project because of its efficiency in processing limited resources, allowing it to run effectively in real-life scenarios like homes or care facilities where timely detection is crucial. Unlike other models that use heatmaps and require post-processing, it can directly predict key points associated with objects which speeds up detection and reduces computational complexity.

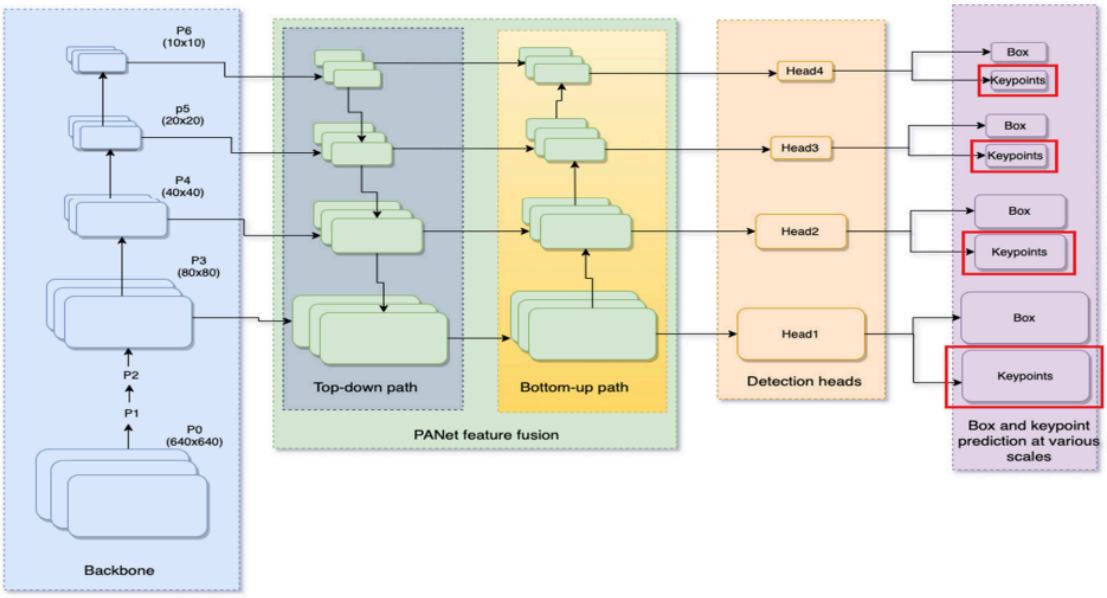

Figure 1: YOLO-Pose model architecture (Debapriya Maji et al., 2022)

Figure 2: YOLO-Pose Example

2.1.4 MediaPipe (Detector)

Mediapipe's pose landmark detection annotates the dataset, which is then used to train the Random Forest classifier. After the classifier is trained we then use it in real-time video capture to predict a person's stance, resulting in a system that can determine if an individual is standing, sitting, or lying down.
Mediapipe's pose landmark detection plays a crucial role in annotating the dataset. When creating the dataset, the system uses it to extract 33 key body landmarks from each image, where each landmark is represented by x and y coordinates. These detected landmarks are saved to a CSV file along with a manual label like "standing", "sitting", or “lying down”

The Random Forest Classifier is trained using the annotated dataset, labelled poses like standing, sitting, and lying down. After training, during the real-time video capture, MediaPipe Pose extracts landmarks from each video frame, which are fed into the RandomForest model to predict the current pose.

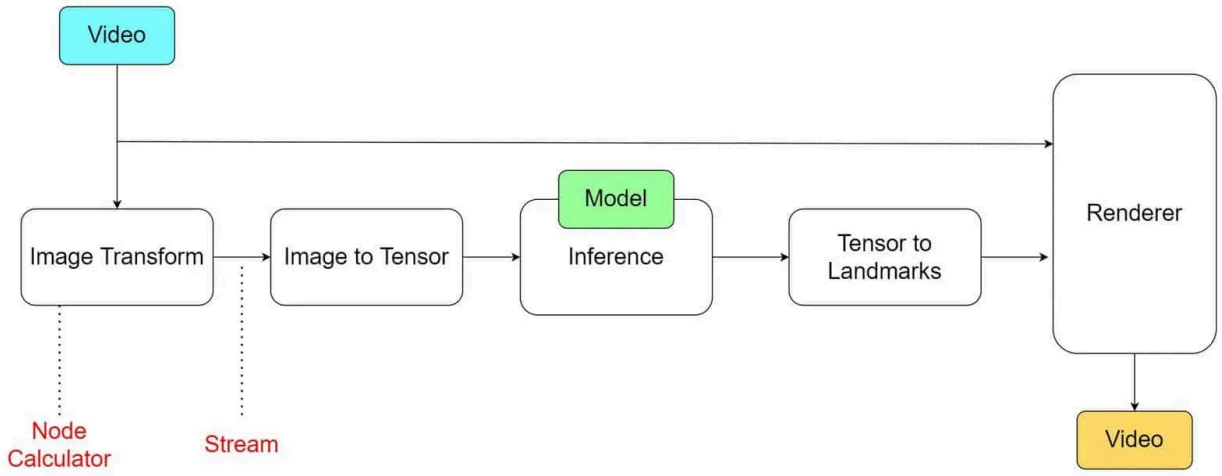

Figure 2: MediaPipe Solution Graph(LearnOpenCV, 2022)

Figure 3: MediaPipe Example

2.1.5 OpenPose (Detector)

Developed by the Carnegie Mellon University, OpenPose is a framework of real-time pose estimation that uses a complex architecture based on Convolutional Neural Networks (CNNs) to detect human body, hand, facial and foot key points from images and videos (GeeksforGeeks, 2024). The images are processed by a set of 2D key points that correspond to specific body parts, connecting these key points into a skeleton of full body poses (GeeksforGeeks, 2024) . A high level accuracy can be assumed due to its ability in detecting spatial relationships between different body parts.

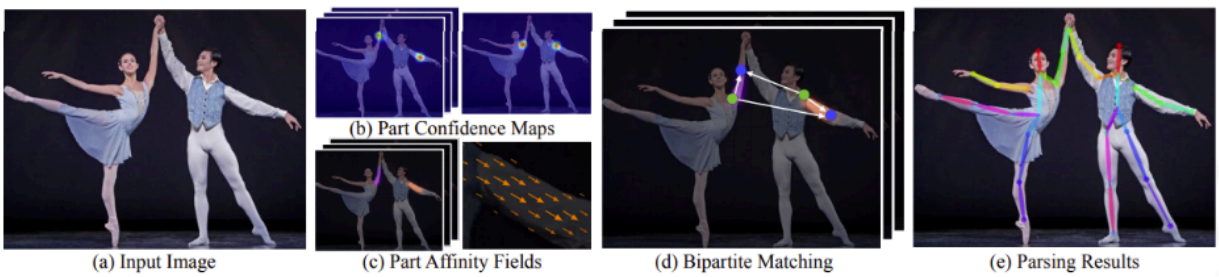

Figure 4: OpenPose Model Architecture (GeeksforGeeks, 2024)

As mentioned by Cao et al.(2019), the input image is first passed through the baseline CNN network which is initialised by the first 10 layers of VGG-19. Then, the feature map is utilised as the input for generating the Part Confidence Maps and Part Affinity Field.Finally, the Confidence Maps and Part Affinity Fields that are generated above are processed by a greedy bipartite matching algorithm to get the poses for each person in the image.

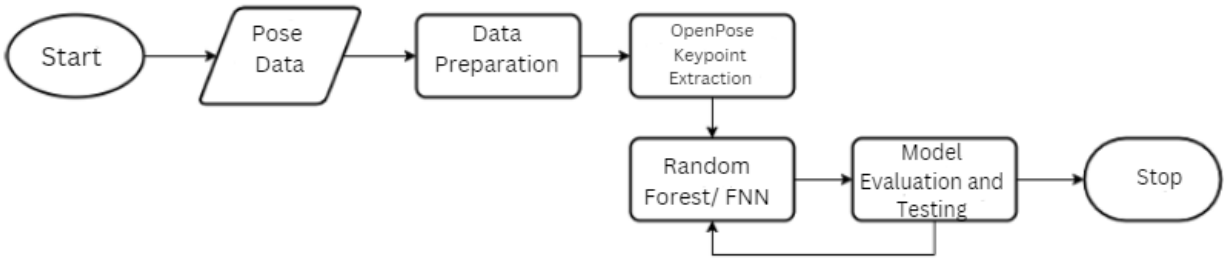

Figure 5: OpenPose Code Implementation Flow Chart

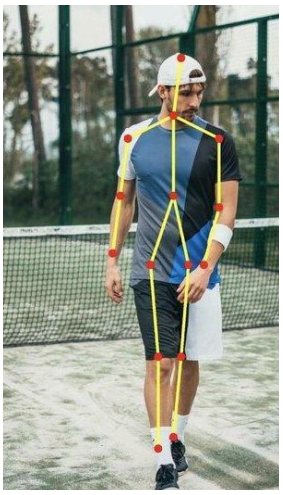

FIgure 6: OpenPose Expected Output

Figure 5 illustrates a machine learning process using OpenPose with Random Forest and Forward Neural Network machine learning. First, as illustrated in Figure 6, every image in the dataset is iterated to retrieve coordinates consisting of 135 keypoints. Then, the coordinates are saved as a csv file and is pre-processed by splitting the data into training (70%), validation (15%), and testing (15%) datasets. When the data is trained, both the model's performances are compared and refined by grid search. The model can be refined according to the initial model evaluation deliverables.

2.1.6 PoseNet (Detector)

The architecture of PoseNet is built upon convolutional neural networks (CNNs), it is a class of deep learning models that specialise in image analysis. How Posenet operates is that the input image initially undergoes a feature extraction phase, where the CNN processes the visual data to identify relevant patterns and structures so that it is able to effectively analyse pose estimation.

In the next phase, PoseNet generates a set of heatmaps, one for each of the 17 key points it's designed to detect. These heatmaps are probability distributions indicating the likelihood of a particular point being present at different locations in the image. In conjunction with the heatmap, a network also predicts part offsets. These offsets serve to refine the keypoint locations, allowing for more precise positioning within the broader grid of the heatmap.

With these techniques PoseNet is able to perform in real-time across a variety of devices. This efficiency is achieved through careful optimisation of the model architecture and implementation.

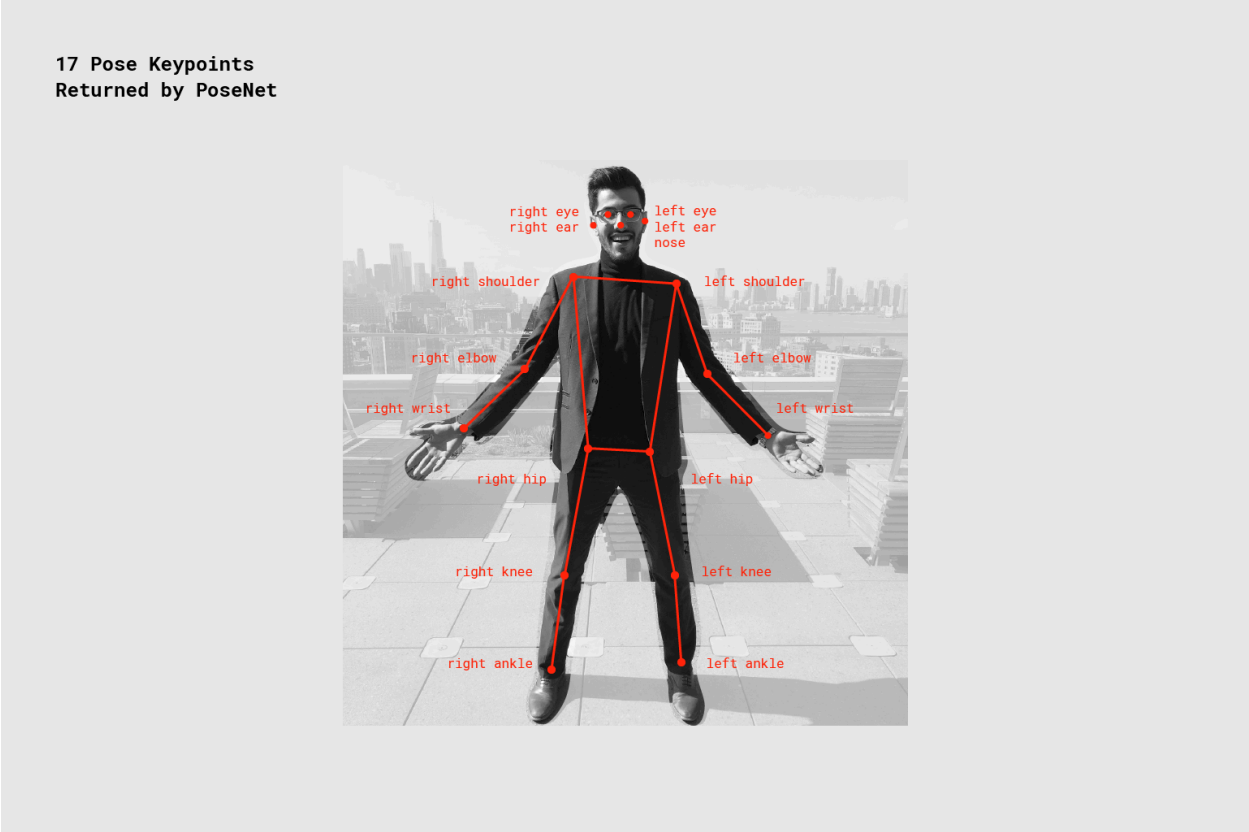

Figure 7: PoseNet Output (*Real-Time Human Pose Estimation in the Browser with TensorFlow.js*, n.d.)

2.2 Data Collection

Initially, the team have decided to collect the dataset themselves as opposed to using pre-existed datasets. As the team wanted to particularly focus on certain poses, the pre-existing datasets were thought to not be suitable since the poses are fixed. Initial approach was to take the pictures of different poses by themselves and incorporate this into a dataset to be used for training and testing the model. However, due to time constraint, the team have decided to download datasets that can be found from Roboflow and NWU.

The dataset was done by combining a total of 619 images that have different lightings, angles and visibility, to ensure the noises are recognised and overcome by the model when training. By doing such, it can effectively eliminate different circumstances in reality that could have potentially caused the camera to be blurred, thereby affecting its ability to detect. For all three poses, they will be labelled with their own class labels into ‘standing’, ‘sitting’ and ‘lying down’ and includes pose-related features (e.g. body angles, joint positions) to ensure it’s suitable for random forest as well as tested for Forward Neural Networks. For ‘sitting’ and ‘standing’ data, the coordinates are predefined on the images. For ‘sitting’ and ‘standing’ data, the coordinates are predefined on the images.

3. Project Management and Teamwork

3.1 Project Plan

Week 9	- Separating the task among team member in a meeting on Teams - Start researching on how to implement the technique with python
Week 10	- Start working on the report - Model building for MediaPipe, OpenPose, Random Forest
Week 11	- More information added to Report - Model building for YoloPose, PoseNet, FNN - Testing each Detector with classifier - Meeting to decide on presentation task division.
Week 12	- Finishing up on the report, check for errors in report and code - Video record for the presentation, editing

3.2 Work Contribution Table:

Project Report	**Group Member Name**	**Contribution/Task**
1. Introduction	Kevin Chen, Yoonsu Kim	Introduction

2. Project Overview and Methodologies	Kevin Chen	2.1.1 Random Forest
	Ching-Ting Chen	2.1.2 Feedforward Neural Network (Classifier)
	Kevin Chen	2.1.3 YoloPose (Detector)
	Ching-Ting Chen	2.1.4 MediaPipe (Detector)
	Valerie Kusumo	2.1.5 OpenPose (Detector)
	Brendan Hua	2.1.6 PoseNet (Detector)
	Kevin Chen, Valerie Kusumo	2.2 Data Collection
3. Project Management and TeamWork	Ching-Ting Chen, Valerie Kusumo	3.1 Project Plan
	Kevin Chen, Valerie Kusumo	3.2 Work Contribution Table
4. Result	Ching-Ting Chen	4.1 MediaPipe with Random Forest
		4.2 MediaPipe with FNN
		4.3 YoloPose with Random Forest
		4.4 YoloPose with FNN
	Valerie Kusumo	4.5 OpenPose with Random Forest
		4.6 OpenPose with FNN
		4.7 PoseNet with Random Forest
		4.8 PoseNet with FNN
5. Discussion and Challenges	Yoonsu Kim	5.1 Goals and Accomplishment
	Ching Ting Chen	5.1.1 MediaPipe
		5.1.2 YoloPose
	Brendan Hua / Valerie Kusumo	5.2 Challenges
	Brendan Hua	5.3 Insights

6. Conclusion	Yoonsu Kim	6.Conclusion
Appendix	Valerie Kusumo & Ching Ting Chen	Codes
	Yoonsu Kim	Reference

4. Result

4.1 MediaPipe with Random Forest

```
standing      191
lying down    174
sitting       153
Name: count, dtype: int64
Random Forest Accuracy: 0.83
              precision    recall  f1-score   support

  lying down       0.95      0.88      0.91        40
     sitting       0.71      0.81      0.76        31
    standing       0.81      0.79      0.80        33

    accuracy                           0.83       104
   macro avg       0.82      0.82      0.82       104
weighted avg       0.83      0.83      0.83       104

Confusion Matrix:
 [[35  3  2]
 [ 2 25  4]
 [ 0  7 26]]
```

Figure 8: MediaPipe with Random Forest Result

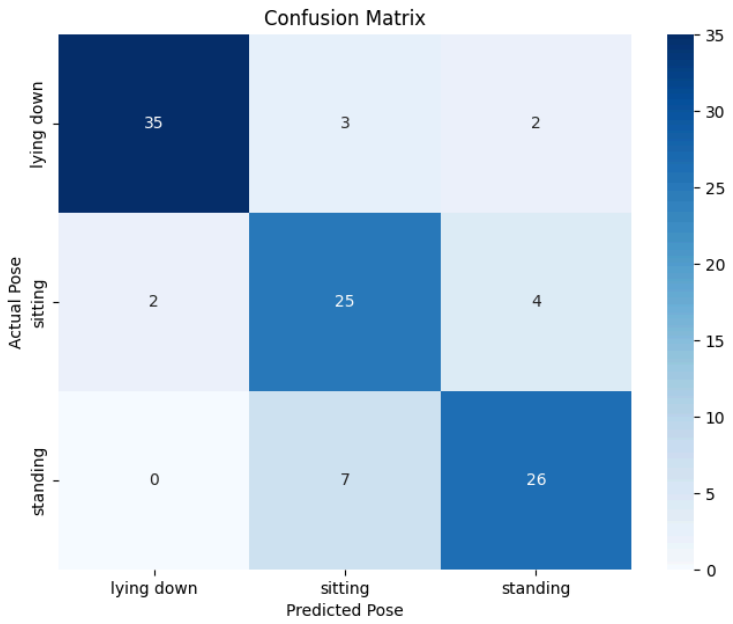

Figure 9: Confusion Matrix for MediaPipe with Random Forest

The overall accuracy for MediaPipe with Random Forest is 83% which is the highest out of all the models we have. And a consistent recall score around 80% and with high recall suggests that the model misses fewer true positives. F1 score for "sitting" is 0.76, which is slightly lower than the other classes, indicating that the model struggles more with this pose compared to "standing" and "lying down." The confusion matrix provides a detailed look at where the model makes correct and incorrect predictions.

4.2 MediaPipe with FNN

```
              precision    recall  f1-score   support

    standing       0.84      0.80      0.82        40
     sitting       0.64      0.81      0.71        31
  lying down       0.67      0.55      0.60        33

    accuracy                           0.72       104
   macro avg       0.72      0.72      0.71       104
weighted avg       0.73      0.72      0.72       104

Confusion Matrix:
 [[32  3  5]
 [ 2 25  4]
 [ 4 11 18]]
```

Figure 10: MediaPipe with FNN Result

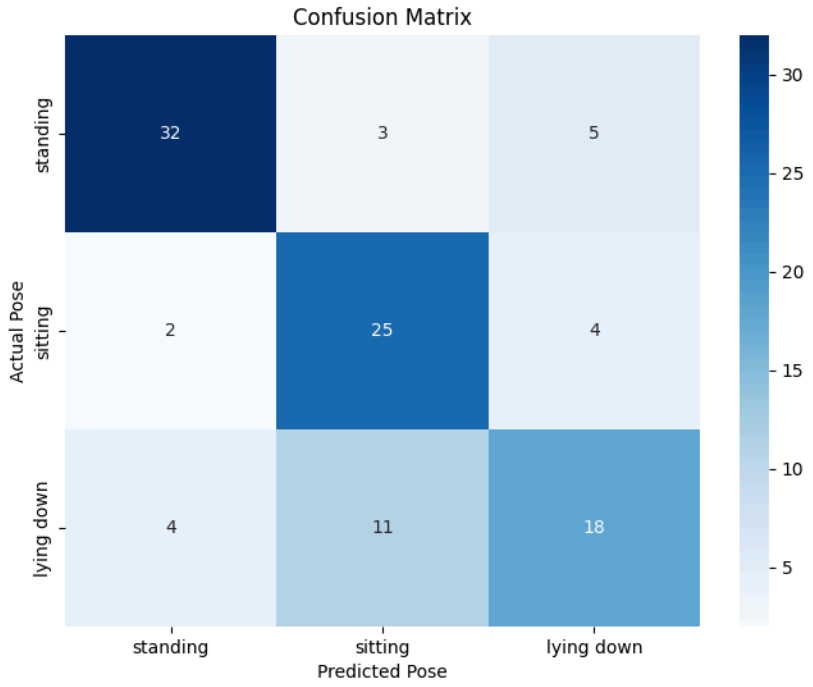

Figure 11: Confusion Matrix for MediaPipe with FNN

The FNN model shows lower overall accuracy in comparison to the Random Forest model, achieving 72% accuracy. And if we take a closer look at the results we can see that FNN performs well for the "standing" pose with high precision, the precision for the "sitting" and "lying down" poses is noticeably lower. For recall, the model shows a significant drop for the "lying down" pose, with only 55%, half of the predictions were wrong. The F1 score shows the same trend, scoring 60%.
This indicates that the model is better at identifying "standing" poses but struggles for "sitting" and "lying down". This could be due to the limited dataset we use, neural network performed better with large dataset and enough practice.

4.3 YoloPose with Random Forest

```
label            object
dtype: object
Random Forest Accuracy: 0.76

Classification Report:
              precision    recall  f1-score   support

    standing       0.86      0.79      0.82        38
     sitting       0.62      0.72      0.67        29
  lying down       0.79      0.76      0.77        49

    accuracy                           0.76       116
   macro avg       0.75      0.76      0.75       116
weighted avg       0.77      0.76      0.76       116


Confusion Matrix:
[[30  3  5]
 [ 3 21  5]
 [ 2 10 37]]
```

Figure 12: YoloPose with Random Forest Result

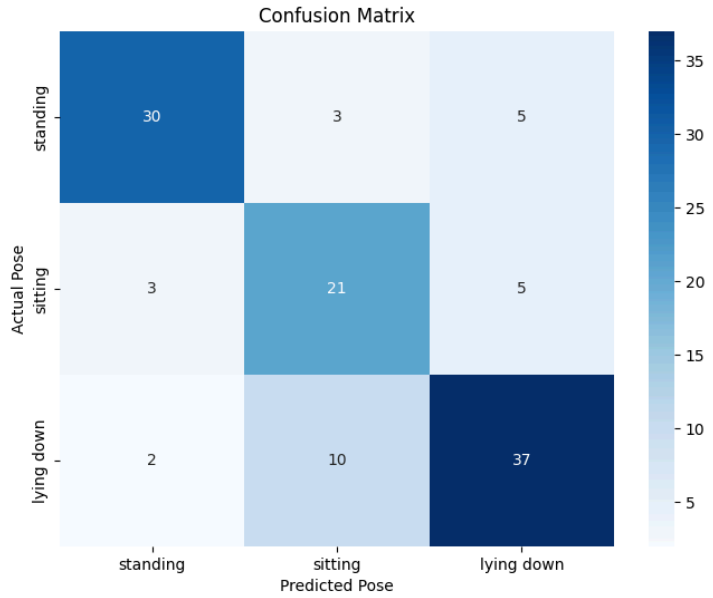

Figure 13: Confusion Matrix for YoloPose with Random Forest

The overall accuracy of the model is 76% with the recall scores being consistent across the poses, around 75%, meaning the model is fairly effective in identifying the true instances of each pose. However, when it comes to precision and F1 score, the model struggles with "sitting" pose. The precision for "sitting" is 62% and 67% for F1 score which is way lower than the other two poses. This means that the model has a significant number of incorrect "sitting" predictions and the drop in F1 score shows the model's difficulty in maintaining a good balance between

precision and recall for the "sitting" pose, likely due to the overlapping characteristics between "sitting" and "lying down" as shown by the confusion matrix.

4.4 YoloPose with FNN

	precision	recall	f1-score	support
standing	0.82	0.74	0.78	38
sitting	0.64	0.55	0.59	29
lying down	0.65	0.76	0.70	49
accuracy			0.70	116
macro avg	0.70	0.68	0.69	116
weighted avg	0.70	0.70	0.70	116

Confusion Matrix:
[[28 1 9]
[2 16 11]
[4 8 37]]

Figure 14: YoloPose with FNN Result

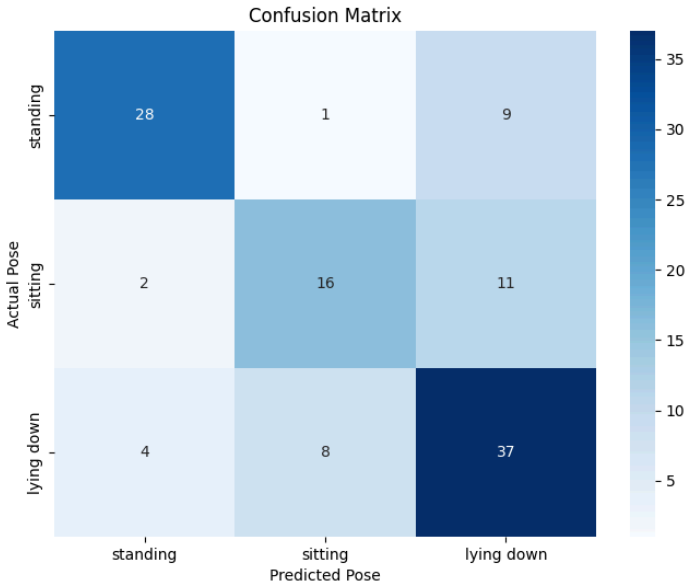

Figure 15: Confusion Matrix for YoloPose with FNN

This FNN with YoloPose achieves a 70% overall accuracy. The performances are relatively good for "standing" and "lying down", with a balanced precision, recall, and F1 score. But the model really struggles with "sitting", as shown by the figure with lower recall (55%) and F1 score (59%). With the confusion matrix it can be seen that almost half of "sitting" instances are misclassified as "lying down," suggesting that the model has difficulty differentiating between the two poses. Improving the differentiation between "sitting" and "lying down" could lead to a better model accuracy and overall performance.

4.5 OpenPose with Random Forest

The OpenPose model, though having a built-in Convolutional Neural Network, will be tested using Random Forest Classifiers as well as Forward Neural Network Classifiers with pre and post hyperparameter tuning using grid search. This is used to compare and contrast the results as well as model performance to observe if it fits initial model deliverables.

4.5.1 Pre Parameter Tuning

```
Training data size: 432
Validation data size: 93
Testing data size: 93

Validation Accuracy: 77.42%
Testing Accuracy: 74.19%

Classification Report (Testing):
              precision    recall  f1-score   support

  Lying Down       0.95      0.97      0.96        39
     Sitting       0.71      0.53      0.61        32
    Standing       0.48      0.64      0.55        22

    accuracy                           0.74        93
   macro avg       0.71      0.71      0.71        93
weighted avg       0.76      0.74      0.74        93
```

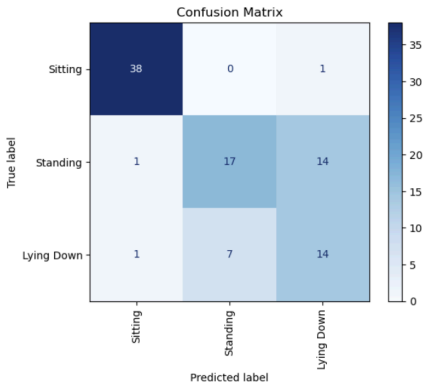

Figure 16: Classification Report and Confusion Matrix (OpenPose Random Forest Pre Parameter Tuning)

The OpenPose model with Random Forest had an overall accuracy of 74.19%, effectively recognising "Lying Down" with an F1-score of 0.96. However, it is less reliable at detecting "Sitting" and "Standing," with F1-scores of 0.61 and 0.55 respectively, indicating misclassifications. The confusion matrix exhibits great performance in identifying the "Sitting" class, properly categorising 38 samples while struggling to identify between "Standing" and "Lying Down," correctly recognising 17 and 14 samples respectively.

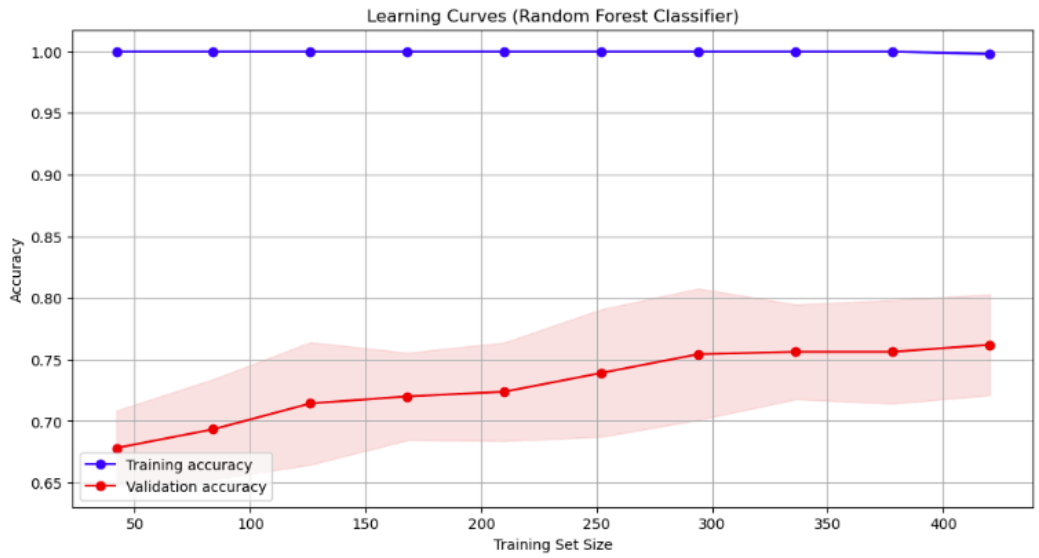

Figure 17: Learning Curve (OpenPose Random Forest Pre Parameter Tuning)

The figure above shows the learning curve of the Random Forest model. The curve indicates overfitting as the training accuracy remains near 1 while the validation accuracy gradually increases but stays around 0.75. This gap suggests the struggle to generalise unseen data while the model fits the training data well. The shaded area in the validation curve represents variability which decreases as the training set size increases, indicating that the model is stabilised with more data.

Figure 18: Classification Test Results (OpenPose Random Forest Pre Parameter Tuning)

The figure above shows the results after classification testing by uploading images one by one from the dataset within each classification. As seen in the figure above, the “Lying Down” data

was incorrectly classified as “Sitting” but the “Standing” and “Sitting” images were classified correctly. This shows that the model requires hyper parameter tuning to increase the accuracy.

4.5.2 Post Parameter Tuning

```
Starting Grid Search for Hyperparameter Tuning...
Fitting Grid Search...
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
Grid Search completed!

Best Hyperparameters (Grid Search): {'max_depth': None, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 50}
Best Training Accuracy: 0.79

Classification Report:
              precision    recall  f1-score   support

  Lying Down       0.88      0.91      0.89        46
    Standing       0.56      0.53      0.54        38
     Sitting       0.57      0.57      0.57        40

    accuracy                           0.69       124
   macro avg       0.67      0.67      0.67       124
weighted avg       0.68      0.69      0.68       124
```

Figure 19: Classification Report (OpenPose Random Forest Post Parameter Tuning)

After implementing a grid search in attempts to improve accuracy, the model’s effectiveness in detecting coordinates within the dataset is approximately 79% with the optimal parameters set for the Random Forest model. The model demonstrated a very strong accuracy in the “Lying Down” class with an F-1 score of 0.89 and a less reliable feature detection for “Sitting” with F-1 score of 0.54. Despite implementing hyperparameter tuning, the F-1 scores and Accuracy decreased instead of the expected result.

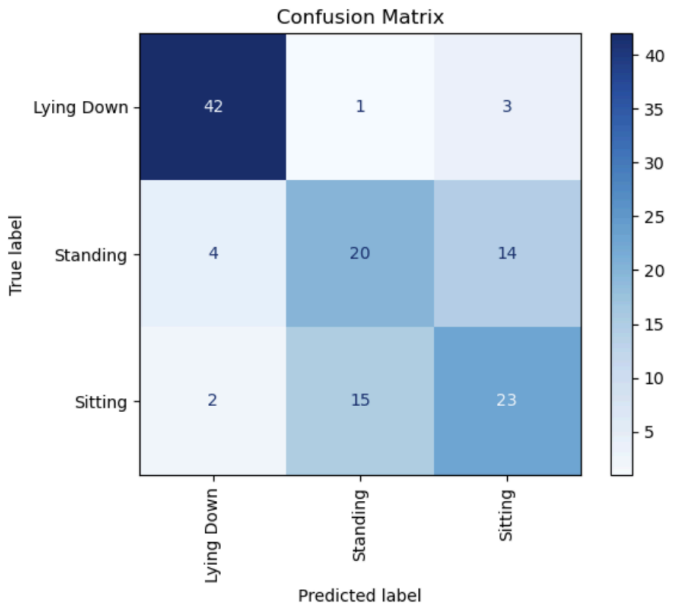

Figure 20: Confusion Matrix (OpenPose Random Forest Post Parameter Tuning)

The figure above represents the confusion matrix after hypertuning the parameters of Random Forest for this dataset. The model is said to effectively identify 42 of the “Lying Down” class with only 4 misclassifications while the “Standing” class has a balanced distribution of correct and incorrect predictions with 20 correctly classified samples but 14 misclassified as “Sitting” and 4 as “Lying Down”. Lastly, the “Sitting” class is said to have identified 23 instances correctly but misclassified 15 as “Standing” and 2 as “Lying Down”.

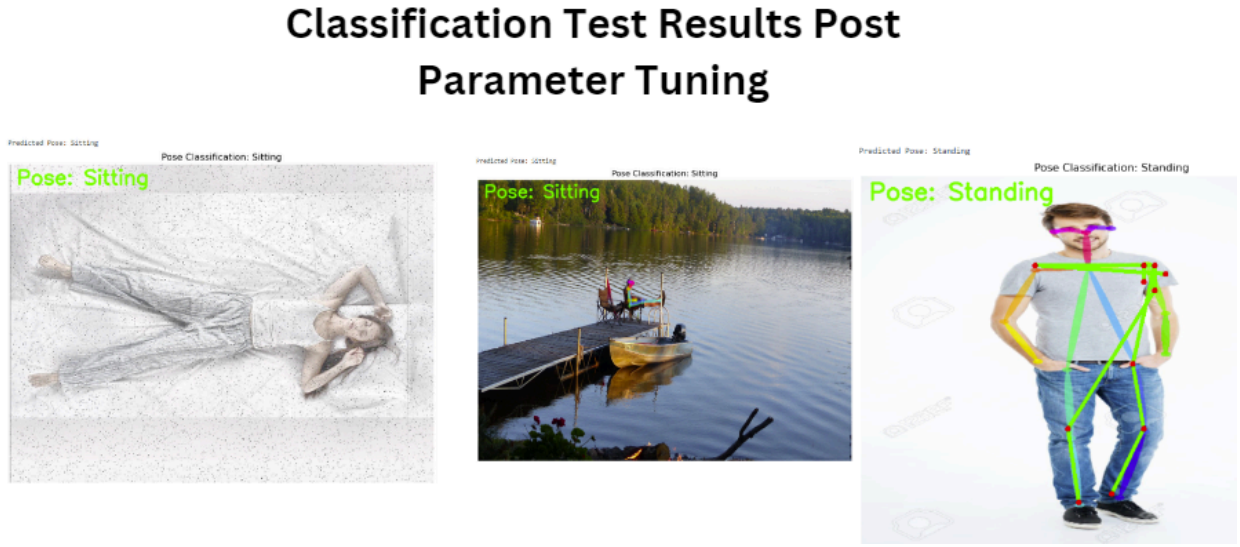

Figure 21: Classification Test Results (OpenPose Random Forest Post Parameter Tuning)

The figure above shows the results after classification testing and hyperparameter tuning by uploading images one by one from the dataset within each classification. As seen in the figure above, no improvements were seen as the “Lying Down” data was incorrectly classified as “Sitting” but the “Standing” and “Sitting” images were classified correctly. This shows the model requires other methods or further feature engineering to increase the accuracy.

4.6 OpenPose with Forward Neural Network

4.6.1 Pre Parameter Tuning

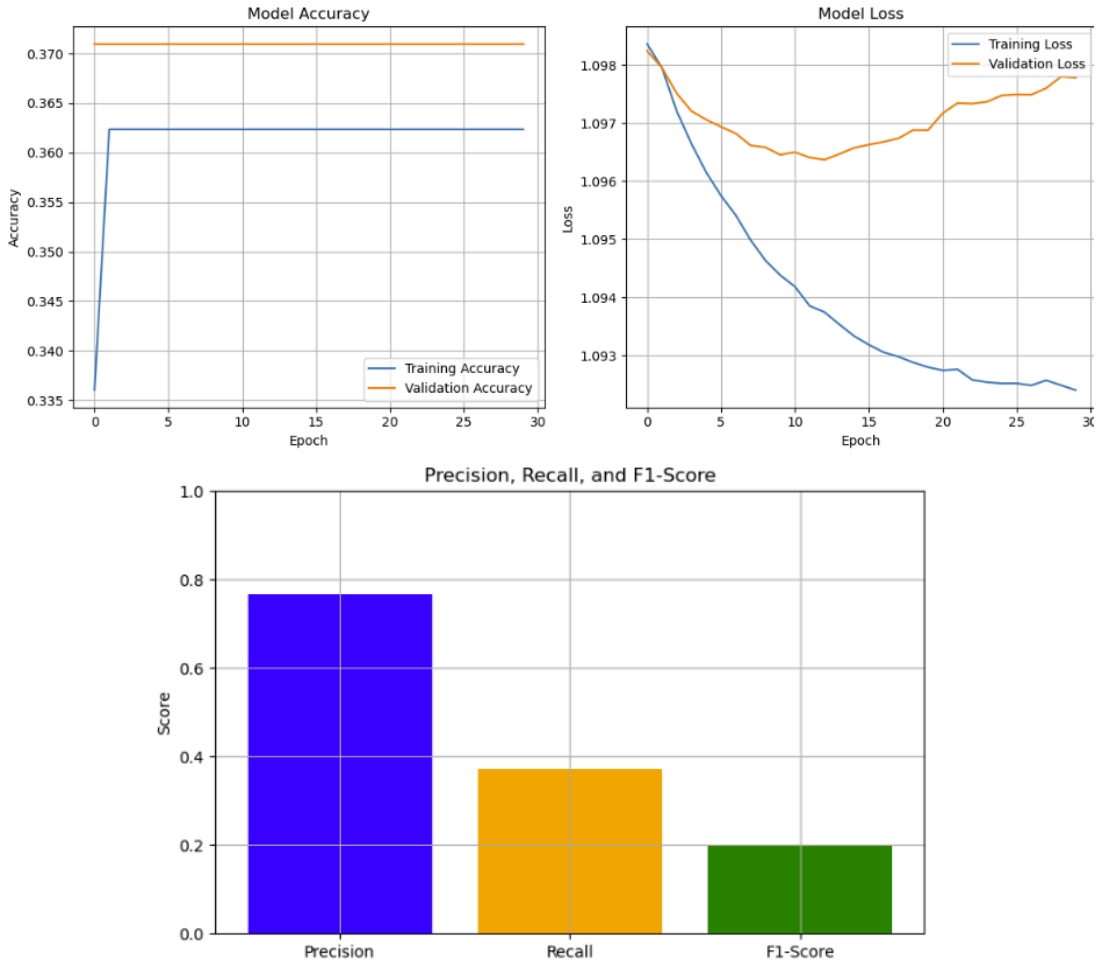

Figure 22: Learning Curves and Bar Graph for FNN (Pre Parameter Tuning)

The figure shows evidence of underfitting because the training accuracy quickly approaches 0.36 and flattens, whilst the validation accuracy stabilises slightly higher at roughly 0.37 with negligible gains across epochs. Furthermore, the loss curves indicate overfitting, with training loss gradually falling and validation loss plateauing before climbing. The bar graph demonstrates that precision is the highest of the metrics, indicating that the model makes reliable positive predictions. However, recall is low, indicating the model's inability to catch all relevant instances, resulting in a low F1 score. This imbalance implies that, while the model is exact, it lacks recall, implying that additional enhancements are required for better overall performance.

Classification Report:

	precision	recall	f1-score	support
Standing	0.00	0.00	0.00	46
Sitting	0.50	0.03	0.05	38
Lying Down	0.32	0.95	0.48	40
accuracy			0.31	124
macro avg	0.27	0.33	0.18	124
weighted avg	0.26	0.31	0.17	124

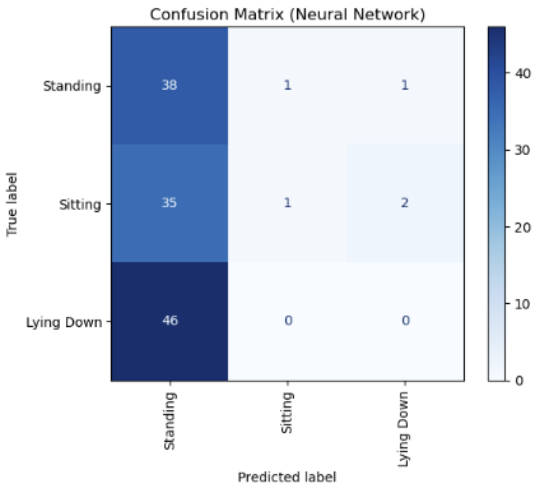

Figure 23: Classification Report and Confusion Matrix for OpenPose with FNN(Pre Parameter Tuning)

Prior to parameter adjustment, the FNN model for OpenPose has a relatively low overall accuracy of 31%, particularly with the "Standing" class, which it incorrectly classifies. The "Sitting" class has just an F1-score of 0.05, while "Lying Down" has a somewhat higher F1-score of 0.48. The poor macro and weighted averages for precision, recall, and F1-score indicate an uneven performance with significant misclassifications, particularly for "standing" and "sitting." The confusion matrix shows that, while the model correctly recognises 38 "standing" samples, it struggles with "sitting" and "lying down," misclassifying both as "standing." This shows a high model bias towards "Standing," which is most likely caused by data imbalance and inadequate feature extraction for the other classes.

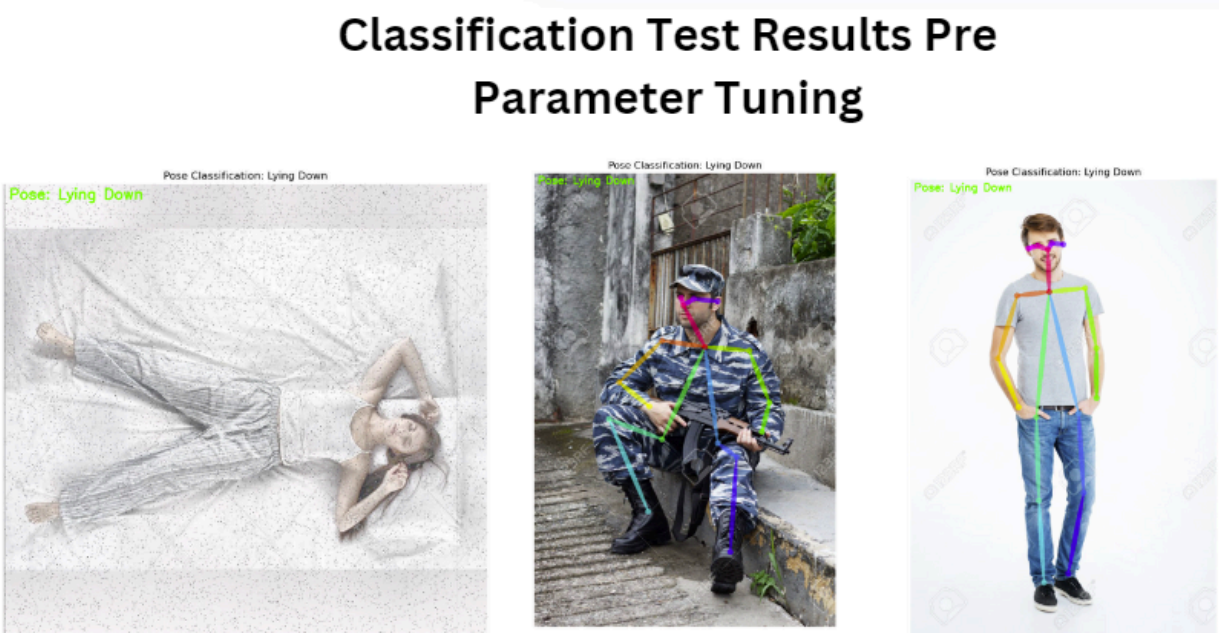

Figure 24: Classification Test Results for OpenPose with FNN(Pre Parameter Tuning)

The figure above shows the results after classification testing by uploading images one by one from the dataset within each classification. As seen in the figure above,all the different types of data were classified as “Lying Down”, defining the model’s inability to differentiate the three classes.

4.6.2 Post Parameter Tuning

```
Starting Grid Search...
Fitting 3 folds for each of 24 candidates, totalling 72 fits
Grid Search completed!

Best Hyperparameters (Grid Search): {'batch_size': 32, 'epochs': 30, 'model__dropout_rate': 0.2, 'model__optimizer': 'rmsprop'}
Best Training Accuracy: 0.72

Classification Report:
              precision    recall  f1-score   support

   Lying Down       0.80      0.93      0.86        46
      Sitting       0.56      0.47      0.51        38
     Standing       0.61      0.57      0.59        40

     accuracy                           0.68       124
    macro avg       0.65      0.66      0.65       124
 weighted avg       0.66      0.68      0.67       124
```

Figure 25: Classification Report for OpenPose with FNN(Post Parameter Tuning)

The classification report above represents the OpenPose model using a feedforward neural network (FNN) after parameter tuning shows improved performance compared to the earlier stages. The model achieved a best training accuracy of 0.72 using optimised hyperparameters, including a batch size of 32, 30 epochs, a dropout rate of 0.2, and the RMSprop optimizer. The "Lying Down" class has the highest performance, with an F1-score of 0.86, precision of 0.80, and recall of 0.93, indicating that the model can accurately identify this class. The "Sitting" and "Standing" classes show moderate improvement, with F1-scores of 0.51 and 0.59, respectively, but still have room for further enhancement.

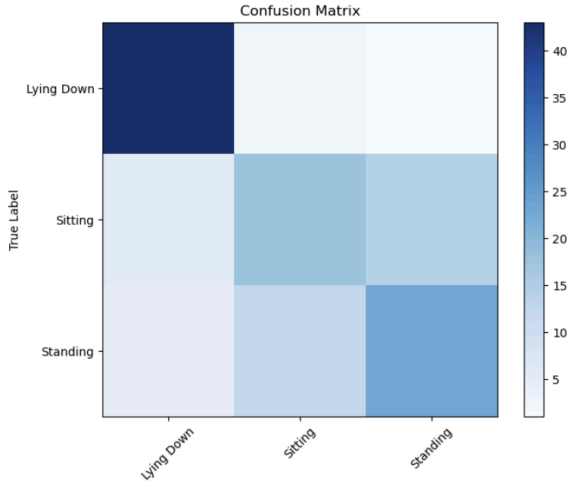

Figure 26: Confusion Matrix for OpenPose with FNN(Post Parameter Tuning)

The figure above represents the confusion matrix for the tuned FNN. This indicates enhanced detection for the "Lying Down" class, with the majority of cases correctly classified. Despite this,

the model still shows confusion between the "Sitting" and "Standing" classes, with numerous "Sitting" examples misclassified as "Standing," and vice versa. Despite improved performance, particularly for "Lying Down," the overlap between "Sitting" and "Standing" indicates these two classes may share similar features or that the model needs to be refined further. The findings indicate that, if the model improves in accuracy, other tactics, such as feature engineering, may help classify between these closely related classes.

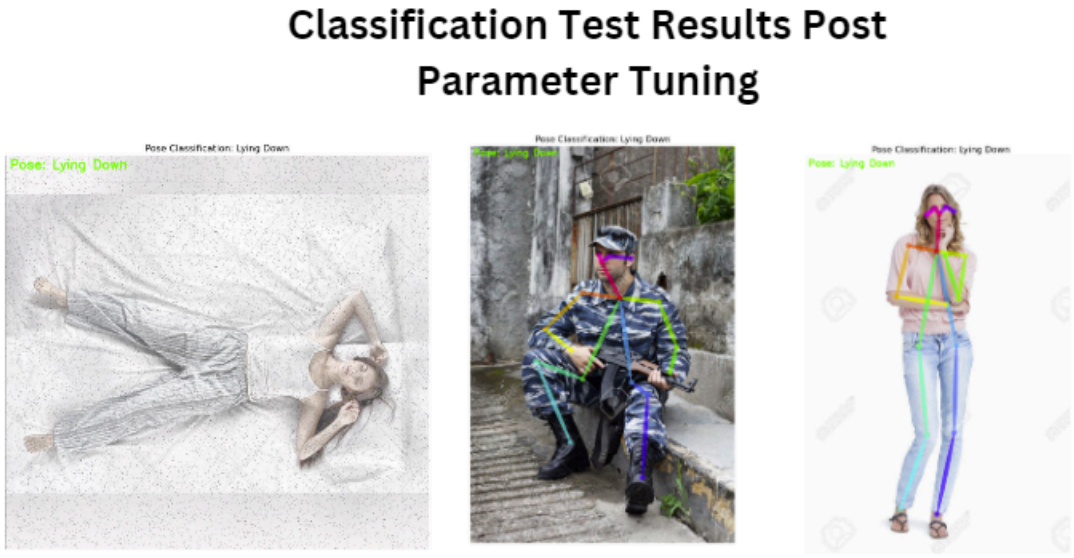

Figure 27: Classification Test Results for OpenPose with FNN(Post Parameter Tuning)

The figure above shows the results after classification testing and hyperparameter tuning by uploading images one by one from the dataset from each classification. As seen in the figure above, no improvements were seen as the “Sitting” and “Standing” data were still classified as “Lying Down”. This shows the model requires other methods or further feature engineering to increase the accuracy.

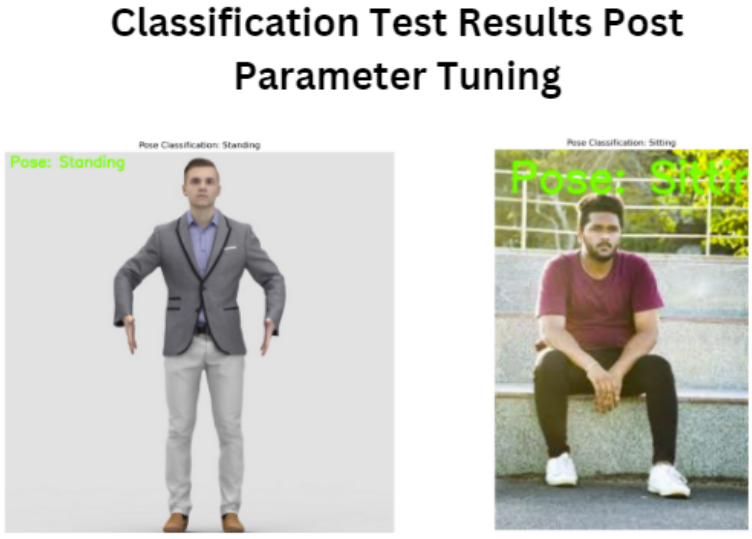

Figure 28: Experimental Classification Test Results for OpenPose with FNN(Post Parameter Tuning)

Out of curiosity, new data without pre-coordinates of “Standing” and “Sitting” was added to the model. This resulted in the correct classification of standing and sitting positions, indicating that the model learns better when it receives raw pose data without pre-defined coordinates. This suggests that the FNN benefits from processing the full set of pose information, allowing it to more accurately distinguish between similar classes like 'Standing' and 'Sitting.' The inclusion of new data without explicit pose markers seems to enhance the model's ability to generalise classes.

4.6 PoseNet with Random Forest

The PoseNet model, though having a built-in Convolutional Neural Network and a hyper-parameter called Output Stride, will be tested using Random Forest Classifiers as well as Forward Neural Network Classifiers with pre and post hyperparameter tuning using grid search. This is used to compare and contrast the results as well as model performance to observe if it fits initial model deliverables.

```
Training data size: 432
Validation data size: 93
Testing data size: 93

Validation Accuracy: 81.72%
Testing Accuracy: 73.12%

Classification Report (Testing):
              precision    recall  f1-score   support

   Lying Down       0.86      0.79      0.83        39
     Sitting        0.68      0.78      0.72        32
    Standing        0.60      0.55      0.57        22

    accuracy                           0.73        93
   macro avg        0.71      0.71      0.71        93
weighted avg        0.74      0.73      0.73        93
```

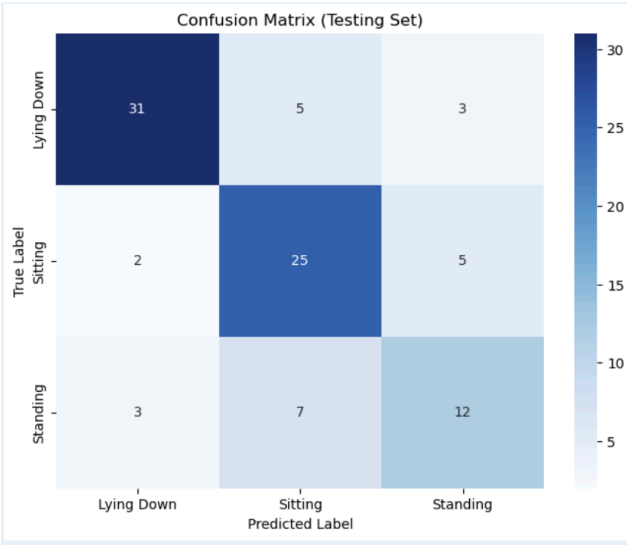

Figure 29: Classification Report for PoseNet with Random Forest(Pre Parameter Tuning)

The PoseNet with Random Forest model has a validation accuracy of 81.72% and test accuracy of 73.12%, demonstrating strong generalisation but a decline in accuracy on unobserved data. The highest classification rate is in “Lying Down”, with an F1-score of 0.83, thanks to precision and recall values of 0.86 and 0.79, respectively. The "Sitting" class obtains an acceptable F1-score of 0.72, whereas the "Standing" class has the lowest performance, with an F1-score of 0.57, indicating frequent misclassification. Overall accuracy is 73%, with F1-scores ranging from 0.71 to 0.74, indicating moderate proficiency.

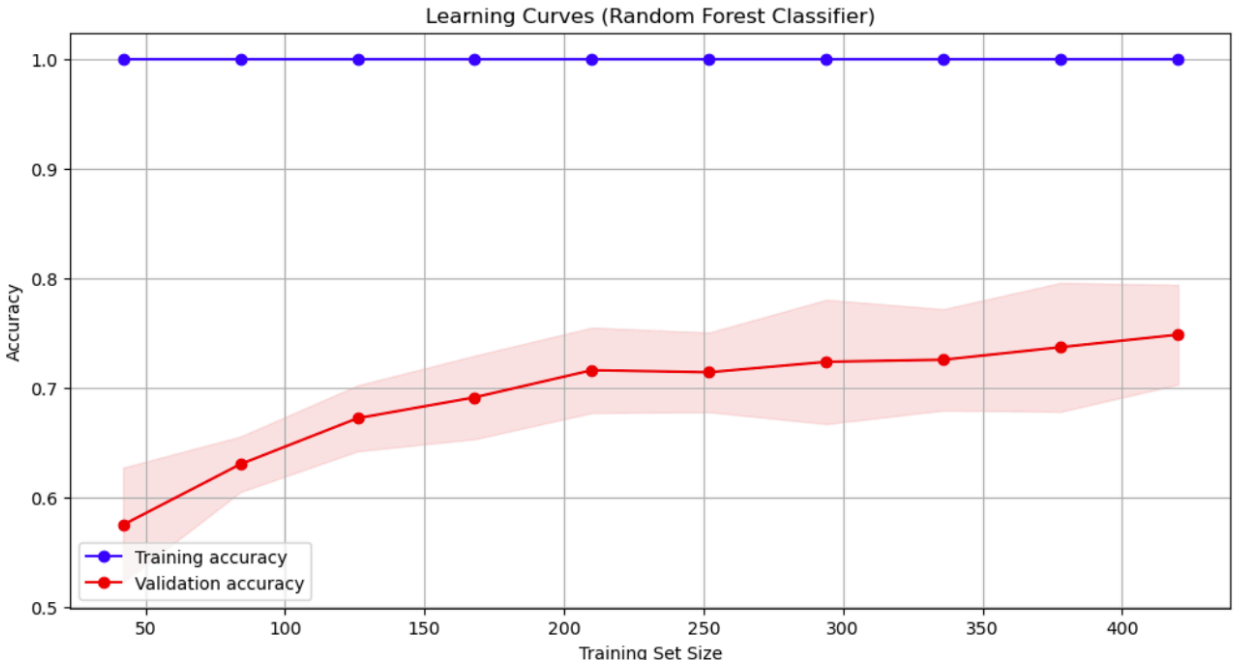

Figure 30: Learning Curve for PoseNet with Random Forest(Pre Parameter Tuning)

The figure above represents the learning curve for PoseNet using a Random Forest classifier before hyperparameter tuning. This indicates overfitting as the training accuracy remains consistently at 1.0, suggesting that the model perfectly fits the training data regardless of the training set size. The validation accuracy, however, starts around 0.55 and increases steadily to about 0.7 as the training set size increases but remains significantly lower than the training accuracy. Additionally, the shaded area around the validation curve indicates the possible fluctuation of performance with different validation subsets. This suggests that the model has difficulty generalising unseen data, requiring further methods to improve its accuracy.

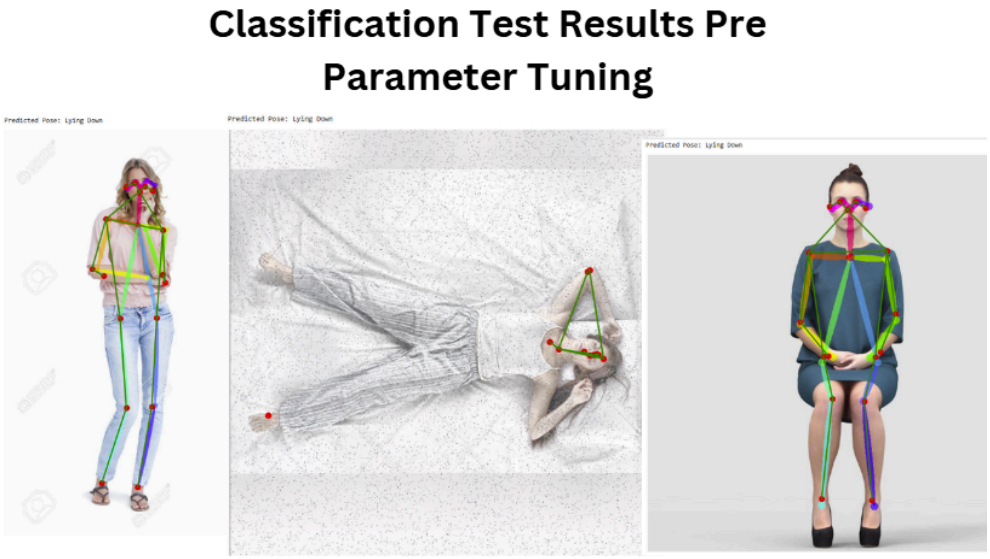

Figure 31: Classification Test Results for PoseNet with Random Forest (Pre Parameter Tuning)

The figure above shows the results after classification testing by uploading images one by one from the dataset within each classification. As seen in the figure above,all the different types of data were classified as “Lying Down”, defining the model’s inability to differentiate the three classes.

4.6.2 Post Parameter Tuning

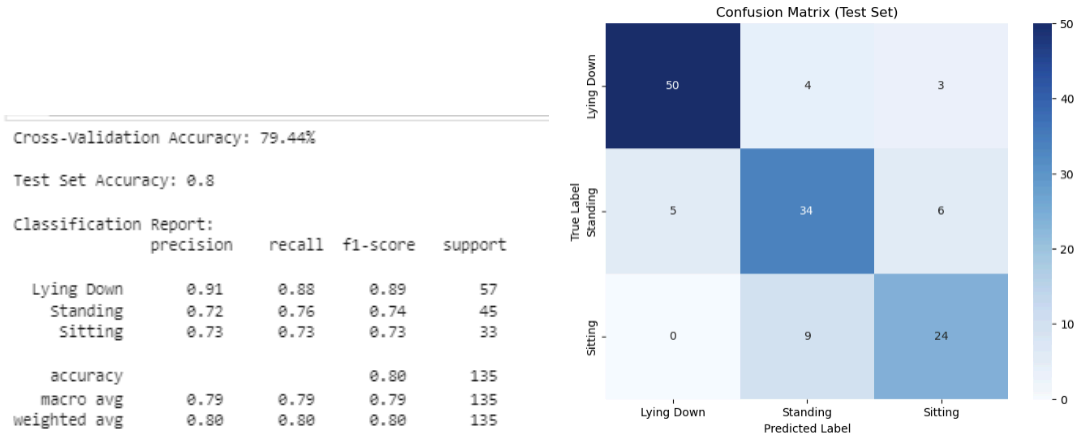

Figure 32: Testing Result for PoseNet with Random Forest(Post Parameter Tuning)

The Random Forest model performs well after hyperparameter adjustment, with an overall test set accuracy of 80% and a cross-validation accuracy of 79.44%. It surpasses in recognising the "Lying Down", with an F1-score of 0.89. The F1-scores for "Standing" and "Sitting" are slightly lower, at 0.74 and 0.73, respectively, but remain acceptable, suggesting consistent but slightly weaker projections. The confusion matrix also shows that "Lying Down" has the most correct classifications, with 50 instances, whereas "Sitting" has more misclassifications, with only 24 classes correctly predicted.

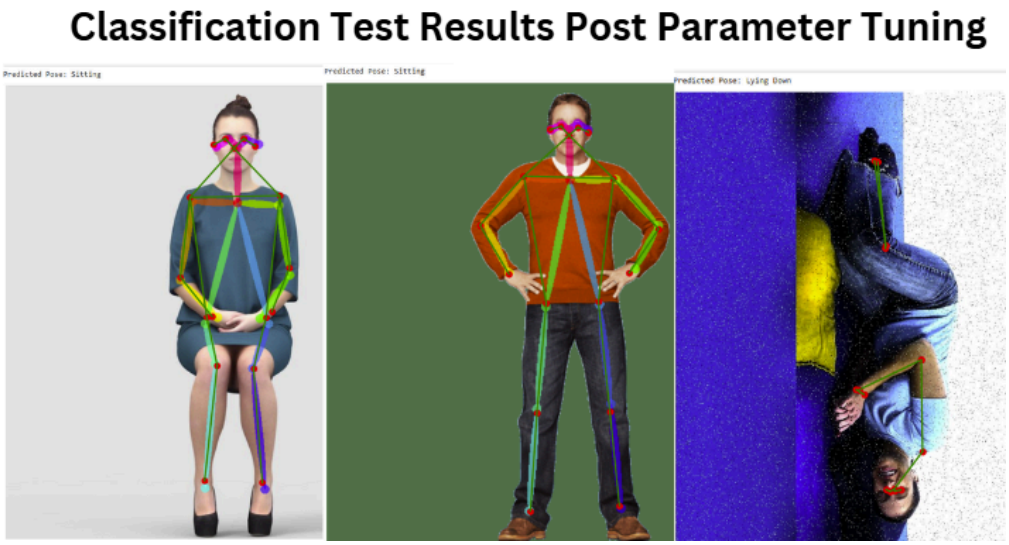

Figure 33: Testing Result for PoseNet with Random Forest(Post Parameter Tuning)

The figure above shows the result of Random forest in PoseNet post hyperparameter tuning using grid search. It classified the “Sitting” and “Lying Down” classes correctly but classified “Standing” as “Sitting”. This shows the model requires other methods or further feature engineering to increase the accuracy.

4.7 PoseNet with Forward Neural Network

4.7.1 Pre Parameter Tuning

Test Set Accuracy: 44.09%

Classification Report:

	precision	recall	f1-score	support
Lying Down	0.61	0.56	0.59	39
Sitting	0.38	0.09	0.15	32
Standing	0.33	0.73	0.45	22
accuracy			0.44	93
macro avg	0.44	0.46	0.40	93
weighted avg	0.46	0.44	0.40	93

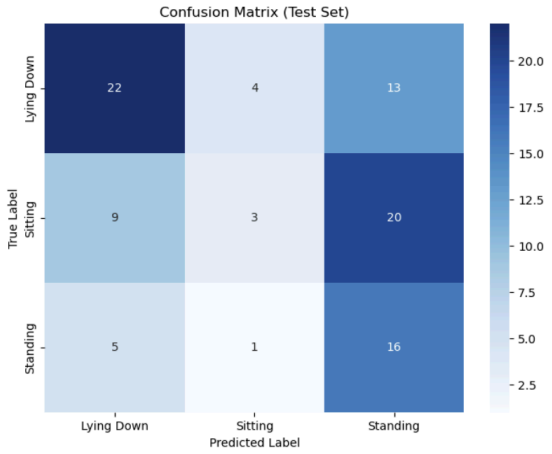

Figure 34: Confusion Matrix for PoseNet with FNN(Pre Parameter Tuning)

The FNN model in PoseNet shows a low test set accuracy of 44.09%, reflecting poor performance in classifying the three poses. The "Lying Down" class achieves moderate results with an F1-score of 0.59. However, the "Sitting" class performs poorly, with an F1-score of 0.15. The "Standing" class shows a moderate F1-score of 0.45. Both the macro and weighted average F1-scores are 0.40, indicating an unbalanced model that struggles to generalise across all classes.

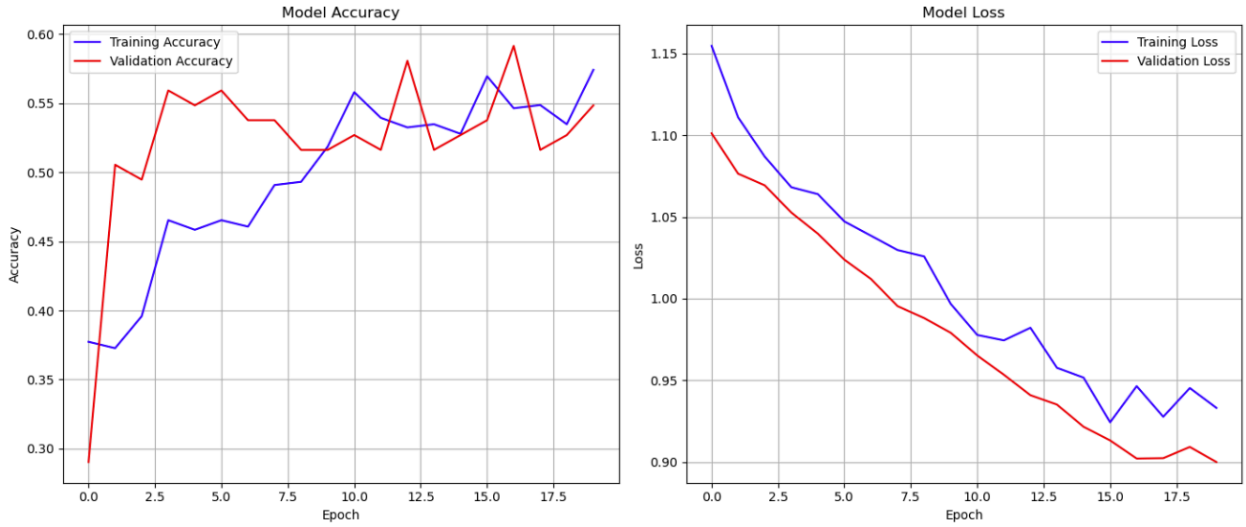

Figure 35: Confusion Matrix for PoseNet with FNN(Pre Parameter Tuning)

The learning curve of both the training and validation accuracy for PoseNet with FNNs pre hyperparameter tuning over 20 epochs has a validation accuracy that initially surpassed the training accuracy in several epochs and both accuracies converge around 0.55 at the end, suggesting a stable performance and good classification in training. In the loss curves, the validation loss consistently remains below training loss, suggesting that the model did not overfit. However, the low accuracy (peaking around 0.55) indicates that the model still has significant room for improvement.

4.7.2 Post Parameter Tuning

```
Best Hyperparameters: {'batch_size': 16, 'epochs': 20, 'model__dropout_rate': 0.3, 'model__num_neurons': 128, 'model__optimizer': 'rmsprop'}

Test Set Accuracy after GridSearch: 63.71%

Classification Report:
              precision    recall  f1-score   support

  Lying Down       0.91      0.63      0.74        46
     Sitting       0.51      0.63      0.56        38
    Standing       0.58      0.65      0.61        40

    accuracy                           0.64       124
   macro avg       0.66      0.64      0.64       124
weighted avg       0.68      0.64      0.65       124
```

Figure 36: Classification Report and Confusion Matrix for PoseNet with FNN(Post Parameter Tuning)

The model after gridsearch achieved a test set accuracy of 63.71%, with the best hyperparameters set to a batch size of 16, 20 epochs, a dropout rate of 0.3, 128 neurons, and the RMSprop optimizer. The model performs best in classifying the "Lying Down" position with a high precision of 0.91 and F1-score of 0.74. However, it struggles to classify "Sitting" and "Standing".

Classification Test Results Post Parameter Tuning

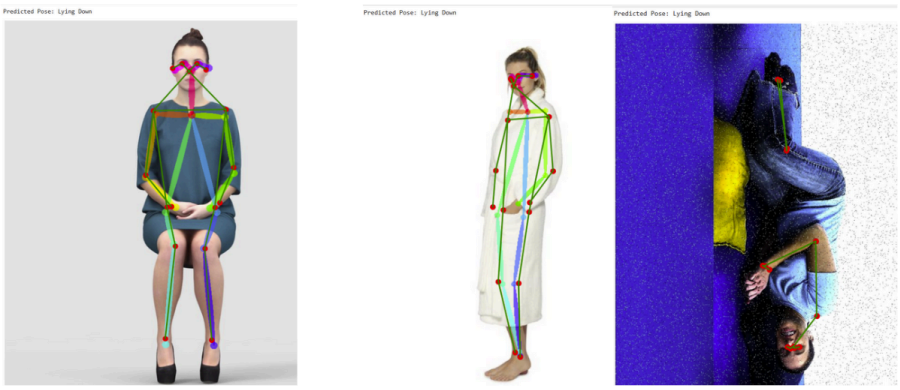

Figure 37: Testing Result for PoseNet with FNN (Post Parameter Tuning)

The figure above shows the results after classification testing and hyperparameter tuning by uploading images one by one from the dataset from each classification. As seen in the figure above, no improvements were seen as the “Sitting” and “Standing” data were still classified as “Lying Down”, similar to the pre parameter tuning of FNN. This shows the model requires other methods or further feature engineering to increase the accuracy.

5. Discussion and Challenges

5.1 Goals and Accomplishments

As mentioned in assignment 1, the team has decided to use evaluation metrics of precision, recall, F-1 score, accuracy as well as confusion matrix to determine the model's performance in each detector. Due to class imbalance, the evaluation of the model is determined by the F-1 score instead of accuracy. An F-1 score is the harmonic mean of precision and recall, essentially, transforming the values into reciprocals to retrieve their balanced average and then transforming them back, as mentioned by GeeksforGeeks (2023). An F-1 score of 0.6-0.8 will be considered acceptable. Anything higher will be considered exceptional and anything lower will be considered unacceptable. Additionally, though accuracy is the proportion of correctly classified samples out of the total number of cases examined, we still use it as an evaluation metric to determine the model's effectiveness. An accuracy above 70% is what our team will aim for and consider the model good to use while anything below 70% is not acceptable.

5.1.1 MediaPipe

MediaPipe has a rich resource base, with lots of tutorials, documentation, and community support available online making it easy during our implementation. The pre-trained models MediaPipe offers, like pose estimation, hand tracking, and face detection serve as a guide while we first started the implementation.

One of our achievements with MediaPipe was a live video capture prediction model, where it uses the trained random forest classifier and the landmark provided by the pose landmark model to predict poses in real time.

MediaPipe with Random Forest model is clearly the better model. With an overall accuracy of 0.83, it surpasses the acceptable threshold of 70% and demonstrates a high level of effectiveness. Its F1 score, exceeding 0.75 across all poses, is deemed exceptional by our standards, making this model highly suitable for use.

In comparison, while the MediaPipe with FNN model meets the accuracy requirement at 0.72, its F1 score falls to 0.60 for the lying down pose, marking it only as acceptable rather than exceptional.

5.1.2 YOLOPose

YOLOPose with Random Forest achieves an overall accuracy of 0.76, which is above our 70% threshold, and demonstrates strong F1 scores: 0.82 for standing, 0.67 for sitting, and 0.77 for lying down. Although the F1 score for sitting is only acceptable, the other scores are either exceptional or acceptable, making this model a reliable choice.

In contrast, YOLOPose with FNN barely meets the acceptable accuracy level at 0.70 and falls short on the F1 score reaching only 0.59 for sitting, which does not meet our standard.

5.2 Challenges

While we achieved many of our initial objectives, some aspects proved more challenging than anticipated. We faced unexpected technical difficulties and a steeper learning curve in certain areas, which required us to reassess our approach and timelines. To address this, we would reevaluate our strategy and redistribute tasks based on individual strengths. This approach allowed us to operate as optimally as possible.

5.2.1 Dataset

Creating a custom dataset proved to be challenging and time-consuming, with ethical and privacy concerns. We opted to use existing public datasets instead, so that more focus can be placed on model development and evaluation.

We encountered dataset bias in the "lying down" pose, leading to misclassifications during testing, as it had too many ‘sleeping’ positions within the dataset. This proved that we required dataset diversification.

Furthermore, we encountered an overfitting issue related to the standing pose dataset. The standing pose images in our dataset already contained coordinate information, which led to the model overfitting on these specific features. This overfitting resulted in poor generalisation when the model was presented with new, unseen data for the standing pose. An attempt was made to search for datasets without coordinate information. However, it proved scarce and subject to copyright concerns.

5.2.2 Time Constraints

One of the significant challenges encountered in this project was the time constraint. The complexity of the task, which according to DataCamp's classification can be categorised as advanced, necessitated extensive research and development efforts (Bex Tuychiev, 2024). The intricate nature of pose recognition, coupled with the need to implement and compare multiple detection and classification techniques, demanded a substantial time investment from all team members.

As a result of the project topic selection it presented the team with a steep learning curve, requiring team members to rapidly acquire and apply new knowledge. This complexity not only increased the time required for implementation but also for troubleshooting and optimising the various components of our system.

5.2.3 Integration of Various Software Components

The integration of various software components presented notable challenges. An example of this is the implementation of PoseNet, as it originally developed to run in JavaScript environments - specifically using TensorFlow.js. However, the project is primarily using Python to run all the classifiers and detectors. Furthermore, when executing PoseNet the coordinates of detected body parts appear incorrectly.

5.2.4 Refining Models

The process of refining our models to achieve desired accuracy and performance was inherently challenging. It required multiple iterations of model adjustments, hyperparameter tuning, and data preprocessing revisions. Furthermore, some detectors/classifiers had extended training time for certain models, due to hardware constraints, posed a significant challenge. We addressed this by running model training overnight to maximise our resources and time efficiency.

5.2.5 MediaPipe

During the implementation of the pose estimation models, MediaPipe would occasionally fail to create landmarks and keypoints on some images. Furthermore, there were compatibility issues with software dependencies as the model required strict version requirements to operate. Specifically, the use of NumPy libraries prior to version 2.0 is required.

5.2.6 YoloPose

YoloPose has been the smoothest and most reliable model to work with. However, YoloPose also encountered a similar issue to MediaPipe, in that it would also occasionally fail to create landmarks and key points on some images.

5.2.7 PoseNet

In implementing PoseNet, we initially set a low confidence threshold of 0.2 (20%) for keypoint detection, resulting in missing key points. An interpolation technique was employed to combat this issue. This technique estimates the missing key points but it presented poorly for asymmetrical poses. The interpolation method's ineffectiveness led to the use of mean keypoint values for compensation. Despite this, the model consistently misclassified images as "lying down," necessitating the use of the original dataset with missing values.

Adapting PoseNet from its native JavaScript environment to Python presented significant technical challenges, including system crashes. This experience demonstrates the complexities of cross-language model implementation and the importance of platform compatibility in selecting pose estimation frameworks. It also emphasised the need for robust error handling and resource management in non-native environments for computationally intensive machine learning models.

5.2.8 OpenPose

Real-time pose detection via camera feed revealed limitations in accurately identifying lying down and standing positions, potentially due to an overrepresentation of sitting postures in the training dataset. The Random Forest classifier excelled in interpreting coordinate data for sitting and standing postures but struggled with lying down positions. Conversely, the Feedforward Neural Network performed well in classifying images without explicit coordinate data, particularly for lying down postures, but misclassified sitting and standing positions.

Webcam-based real-time classification yielded suboptimal results, attributed to dataset imbalance affecting model generalisation. Despite hyperparameter tuning through grid search, these issues persisted, sometimes leading to reduced overall accuracy. These findings highlight the challenges in developing robust pose recognition systems and underscore the importance of balanced, representative datasets in machine learning applications.

5.2.9 Team Issues

Furthermore, the project's timeline was adversely affected by uneven distribution of workload among team members. Notably, one team member completed their assigned tasks very close to the deadline, whilst the other had made no contribution to the project. This last-minute approach to task completion introduced additional stress to the project timeline and potentially compromised the whole project.

5.3 Insights

Given additional time, our team would have approached the project topic selection with greater consideration. While we do not regret our choice, as it provided valuable learning experiences and opportunities for experimentation, we recognise that a more thorough consideration of alternatives could have been beneficial.

With extended time, we could have conducted a more in-depth comparison of models, incorporating advanced feature engineering methods such as feature splitting. This would have allowed for a more nuanced analysis of model performance under various conditions. Moreover, further experimentation and fine tuning with hyperparameter tuning and optimisation could have been applied to the models. We have considered using Bayesian optimisation or random search across a broader hyperparameter space which could have potentially led to enhanced performance. Additionally, further model evaluation metrics could have been included as we have previously considered to include Object Keypoint Similarity (OKS) which is an evaluation metric that is highly relevant to pose estimation. As OKS provides a nuanced evaluation of keypoint localisation accuracy by considering the size of the object and the relative importance of different keypoints.

Furthermore, additional time would have enabled us to implement a more efficient task distribution strategy among team members. This project served as a significant learning

experience in teamwork dynamics and highlighted the critical importance of effective communication in collaborative endeavours.

Perhaps most crucially, this project underestimated the value of individual initiative and contribution within a team setting. It provided us with the opportunity to identify and leverage our respective strengths while addressing our weaknesses, ultimately fostering a more cohesive and productive team environment.

6. Conclusion

This project aimed to develop a comprehensive, real-time pose detection system that enhances fall detection capabilities for elderly care. By leveraging image-based solutions such as YOLOPose, MediaPipe, OpenPose, and PoseNet. The project sought to address the limitations of current fall detection systems, which primarily rely on wearables or surveillance cameras. Through rigorous experimentation and evaluation, each technique’s performance was assessed in terms of accuracy, speed, and adaptability to different environments.

The results demonstrated that while MediaPipe integrated with Random Forest achieved the highest accuracy (83%), it also maintained a balanced recall rate across all poses which make it the most reliable model for real-time detection. In contrast, the performance of Feedforward Neural Networks (FNN) was less consistent, particularly in classifying "lying down" and "sitting" poses as accuracy scores falling below 75%. This underperformance highlights FNN's sensitivity to smaller datasets and limited training iterations, indicating that this model would benefit from a more extensive dataset and longer training times.

YOLOPose proved to be an efficient and resource-effective solution, especially suitable for environments with limited computational power, such as homes and care facilities. However, its integration with Random Forest showed that while it could identify standing and lying down poses effectively, it struggled with differentiating "sitting" from "lying down," likely due to overlapping pose characteristics. This suggests that future iterations should focus on improving model tuning and refining pose-specific data to enhance detection accuracy.

Despite its complexity and longer processing times, OpenPose showcased exceptional accuracy in detecting complex poses, particularly for multi-person scenarios. However, its higher computational requirements make it less ideal for real-time applications in low-resource settings. PoseNet, while more suitable for mobile devices, displayed lower overall accuracy than other models, reinforcing the need for optimised implementation when used alongside classifiers like Random Forest and FNN.

The project’s iterative approach allowed for continuous adjustments based on performance metrics, addressing initial challenges such as dataset bias, model overfitting, and integration issues across different detectors. It also underscored the importance of diverse datasets in improving real-world applicability, particularly in accurately detecting poses in various lighting conditions and angles.

In summary, this project demonstrates a hybrid approach combining real-time pose detectors with optimised classifiers that can significantly improve fall detection for elderly individuals. While MediaPipe with Random Forest emerged as the most balanced solution, further developments should aim at integrating advanced models like YOLOPose and OpenPose with improved training datasets and hyperparameter tuning to enhance accuracy and reliability. The insights gained from this project can serve as a foundation for future work, particularly in developing more adaptable, precise, and user-friendly fall detection systems that minimise false positives and provide timely responses in emergency situations.

Appendix

Dataset:

RoboFlow. (2023). *Sleep poses dataset*. RoboFlow Universe. https://universe.roboflow.com/sleep-pose/sleep-poses/dataset/2

North-West University. (2024). *Human pose dataset (sit, stand pose classes)*. DAYTA. https://dayta.nwu.ac.za/articles/dataset/Human_pose_dataset_sit_stand_pose_classes_/23290937

References:

Bex Tuychiev. (2024, July 24). *19 Computer Vision Projects From Beginner to Advanced*.DataCamp. https://www.datacamp.com/blog/computer-vision-projects

Cao, Z., Hidalgo, G., Simon, T., Wei, S.-E., & Sheikh, Y. (2019). *OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields.* https://arxiv.org/abs/1812.08008

Datta, S. (2020, June 3). *Introduction to MediaPipe*. LearnOpenCV. https://learnopencv.com/introduction-to-mediapipe/

Del Rosario, M. B., Wang, K., Wang, J., & Lovell, N. H. (2015). A review of the development and applications of smart wearable systems for fall detection and personal healthcare. *Frontiers in Aging Neuroscience, 7*, 145. https://www.frontiersin.org/journals/aging-neuroscience/articles/10.3389/fnagi.2015.00145/full

GeeksforGeeks. (2022). *PoseNet pose estimation.* GeeksforGeeks. https://www.geeksforgeeks.org/posenet-pose-estimation/

GeeksforGeeks. (2024). *Feedforward neural network.* GeeksforGeeks. https://www.geeksforgeeks.org/feedforward-neural-network/

GeeksforGeeks. (2024). *OpenPose : Human Pose Estimation Method.* GeeksforGeeks. https://www.geeksforgeeks.org/openpose-human-pose-estimation-method/

GeeksforGeeks. (2024). *Random forest algorithm in machine learning.* GeeksforGeeks. https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning/

Healthdirect. (n.d). *Older people and falls* https://www.healthdirect.gov.au/falls#:~:text=Falls%20are%20the%20number%20one,least%201%20fall%20per%20year

Scherer, J., & Lim, S. (2022). Pose detection for real-time human motion analysis. *Scientific Reports, 12*, Article 26951. https://link.springer.com/article/10.1038/s41598-022-26951-z

Stone, E., & Skubic, M. (2015). Fall detection in homes of older adults using the Microsoft Kinect. *IEEE Journal of Biomedical and Health Informatics, 19*(1), 290-301. https://ieeexplore.ieee.org/document/6774430

Codes

https://drive.google.com/drive/folders/1--TCxQ0OIAjBdFMRfBrVumvJ5rjeeiky?usp=sharing

MediaPipe PoseLandmark

```python
import os
import csv
import cv2
import mediapipe as mp

# Initialize MediaPipe Pose
mp_pose = mp.solutions.pose
mp_drawing = mp.solutions.drawing_utils

# Path to the folder containing images (with subfolders)
image_folder = '/Users/timchen/Desktop/MediaPipe/images/'
# Folder to save processed images
csv_file = 'pose_landmarks.csv'
output_folder = '/Users/timchen/Desktop/MediaPipe/output_images/'
csv_file = 'pose_landmarks.csv'

# Ensure output folder exists
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

# Initialize CSV file for storing landmarks
with open(csv_file, mode='w', newline='') as file:
    writer = csv.writer(file)
    header = ['filename'] + [f'landmark_{i}_x' for i in range(33)] + [f'landmark_{i}_y' for i in range(33)] + ['label']
    writer.writerow(header)

    # Initialize Pose model
    with mp_pose.Pose(static_image_mode=True) as pose:
        for root, dirs, files in os.walk(image_folder):  # Recursively go through subfolders
            for image_file in files:
                image_path = os.path.join(root, image_file)
                image = cv2.imread(image_path)

                # Skip if image could not be read
                if image is None:
                    continue

                # Convert image to RGB
                rgb_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

                # Process the image and detect landmarks
                results = pose.process(rgb_image)
```

```python
                if results.pose_landmarks:
                    landmarks = []
                    for landmark in results.pose_landmarks.landmark:
                        landmarks.extend([landmark.x, landmark.y])


                    # Label based on file naming convention
                    if 'sit' in image_file.lower():
                        label = 'sitting'
                    elif 'stand' in image_file.lower():
                        label = 'standing'
                    elif 'lying' in image_file.lower():
                        label = 'lying down'
                    else:
                        label = 'unknown'

                    # Write to CSV file
                    writer.writerow([image_file] + landmarks + [label])

                    # Draw landmarks on the image
                    mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_pose.POSE_CONNECTIONS)

                    # Save the processed image with landmarks
                    relative_path = os.path.relpath(image_path, image_folder)
                    output_image_path = os.path.join(output_folder, relative_path)

                    # Ensure the directory for the output image exists
                    os.makedirs(os.path.dirname(output_image_path), exist_ok=True)

                    cv2.imwrite(output_image_path, image)

print("Landmark extraction complete. Images and data saved.")

```

Random Forest for mediapipe

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Load the data
data = pd.read_csv('pose_landmarks.csv')

# Check for class distribution
print(data['label'].value_counts())

# Split data into features (landmarks) and target (label)
X = data.drop(columns=['filename', 'label'])
y = data['label']

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the Random Forest classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)

# Predict and evaluate
y_pred = rf_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Random Forest Accuracy: {accuracy:.2f}")

# Additional evaluation with classification report
print(classification_report(y_test, y_pred))

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", conf_matrix)

# Visualize confusion matrix using Seaborn heatmap
plt.figure(figsize=(8,6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=rf_classifier.classes_, yticklabels=rf_classifier.classes_)
plt.ylabel('Actual Pose')
plt.xlabel('Predicted Pose')
plt.title('Confusion Matrix')
plt.show()
```

YoloPose Landmark

```python
import os
import cv2
import csv
from ultralytics import YOLO

# Path to the folder containing images (with subfolders)
image_folder = '/Users/timchen/Desktop/YoloPose/images/'
# Folder to save processed images and CSV
csv_file = 'pose_landmarks.csv'
output_folder = '/Users/timchen/Desktop/YoloPose/output_images/'

# Ensure output folder exists
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

# Initialize YOLOPose model
model = YOLO('yolo11n-pose.pt')

def process_images_and_save_landmarks():
    """Processes images from the folder, detects poses, saves landmarks to a CSV, and saves processed images."""
    # Initialize CSV file for storing landmarks
    with open(csv_file, mode='w', newline='') as file:
        writer = csv.writer(file)
        header = ['filename'] + [f'landmark_{i}_x' for i in range(17)] + [f'landmark_{i}_y' for i in range(17)] + ['label']
        writer.writerow(header)

        # Iterate through the image folder
        for root, dirs, files in os.walk(image_folder):
            for image_file in files:
                image_path = os.path.join(root, image_file)
                image = cv2.imread(image_path)

                # Skip if image could not be loaded
                if image is None:
                    print(f"Failed to load image: {image_file}")
                    continue

                # Detect pose landmarks using YOLOPose
                results = model(image)

                if results and results[0].keypoints is not None:
                    keypoints = results[0].keypoints.data.cpu().numpy()[0]
                    landmarks = []
                    for point in keypoints:
                        landmarks.extend([point[0] / image.shape[1], point[1] / image.shape[0]])

                    # Label based on file naming convention
                    if 'sit' in image_file.lower():
                        label = 'sitting'
                    elif 'stand' in image_file.lower():
                        label = 'standing'
                    elif 'lying' in image_file.lower():
                        label = 'lying down'
                    else:
                        label = 'unknown'

                    # Write to CSV file
                    writer.writerow([image_file] + landmarks + [label])

                    # Draw landmarks on the image
                    for point in keypoints:
                        cv2.circle(image, (int(point[0]), int(point[1])), 5, (0, 255, 0), -1)

                    # Save the processed image with landmarks
                    relative_path = os.path.relpath(image_path, image_folder)
                    output_image_path = os.path.join(output_folder, relative_path)
                    os.makedirs(os.path.dirname(output_image_path), exist_ok=True)
                    cv2.imwrite(output_image_path, image)

    print("Pose landmarks extracted and saved to CSV.")

process_images_and_save_landmarks()
```

Random Forest for YoloPose

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
import matplotlib.pyplot as plt

# Path to the CSV file generated by YOLOPose
csv_file = 'pose_landmarks.csv'

# Manually specify the pose labels
pose_labels = ['standing', 'sitting', 'lying down']

def random_forest_classifier():
    """Train and evaluate a Random Forest Classifier using the pose landmarks from the CSV."""
    # Load the pose landmarks CSV
    data = pd.read_csv(csv_file)

    # Check if there are any non-numeric columns in the features
    print("Data types:\n", data.dtypes)

    # Filter out any rows where the label is other
    data = data[data['label'].isin(pose_labels)]

    # Encode the 'label' column (target) using LabelEncoder
    le = LabelEncoder()
    data['label'] = le.fit_transform(data['label'])

    # Drop any non-numeric columns (such as 'filename') and make sure all features are numeric
    X = data.drop(columns=['filename', 'label'])
    y = data['label']

    # Convert features to numeric data types if necessary (this should prevent string-to-float errors)
    X = X.apply(pd.to_numeric, errors='coerce')

    # Check for any missing values or NaNs in the features after conversion
    if X.isnull().values.any():
        print("Warning: Missing values detected in the features. Handling them by filling with 0.")
        X = X.fillna(0)

    # Split into training and test sets (80% training, 20% testing)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Initialize and train the Random Forest classifier
    rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
    rf_classifier.fit(X_train, y_train)

    # Predict and evaluate on the test set
    y_pred = rf_classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Random Forest Accuracy: {accuracy:.2f}")

    # Additional evaluation with classification report
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred, target_names=pose_labels))

    # Confusion matrix
    conf_matrix = confusion_matrix(y_test, y_pred)
    print("\nConfusion Matrix:")
    print(conf_matrix)

    # Visualize confusion matrix using Seaborn heatmap with pose labels
    plt.figure(figsize=(8,6))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=pose_labels, yticklabels=pose_labels)
    plt.ylabel('Actual Pose')
    plt.xlabel('Predicted Pose')
    plt.title('Confusion Matrix')
    plt.show()

random_forest_classifier()
```

FNN for MediaPipe and YoloPose

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix

# Load the data
csv_file = 'pose_landmarks.csv'
data = pd.read_csv(csv_file)

# Preprocess the labels (target)
pose_labels = ['standing', 'sitting', 'lying down']
data = data[data['label'].isin(pose_labels)]  # Filter relevant poses

# Encode labels into numeric values
le = LabelEncoder()
data['label'] = le.fit_transform(data['label'])  # Encode pose labels

# Prepare features (X) and labels (y)
X = data.drop(columns=['filename', 'label'])  # Drop unnecessary columns
y = to_categorical(data['label'])  # Convert labels to one-hot encoded format

# Convert features to numeric data types (ensures compatibility)
X = X.apply(pd.to_numeric, errors='coerce').fillna(0)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build a Feedforward Neural Network
model = Sequential([
    Dense(128, input_shape=(X_train.shape[1],), activation='relu'),
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(y_train.shape[1], activation='softmax')  # Output layer with softmax for classification
])

```

```python
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=20, batch_size=32, validation_split=0.2)

# Evaluate the model on test data
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_accuracy:.2f}")

# Predict on test data
y_pred = np.argmax(model.predict(X_test), axis=1)
y_true = np.argmax(y_test, axis=1)

# Display a classification report
print(classification_report(y_true, y_pred, target_names=pose_labels))

# Confusion matrix
conf_matrix = confusion_matrix(y_true, y_pred)
print("Confusion Matrix:\n", conf_matrix)

# Visualize the confusion matrix using Seaborn heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=pose_labels, yticklabels=pose_labels)
plt.ylabel('Actual Pose')
plt.xlabel('Predicted Pose')
plt.title('Confusion Matrix')

# Save the confusion matrix as an image
plt.savefig('confusion_matrix.png')

# Display the plot
plt.show()

```

OpenPose: