Human Fall Detection System

Yinzhou Lu, Hang Yu

Abstract

Deep learning has significantly advanced how it solve complex problems in various areas like image recognition, language processing, and healthcare. Particularly, convolutional neural networks and recurrent neural networks can analyze large amounts of data to recognize patterns and make decisions, improving tasks like diagnosing diseases or translating languages.

Our project applies these advanced technologies to help elderly people stay safe in their homes. The elderly is at risk of falling, and if the falling situations are not detected on time, they can lead to serious health problems or even be fatal. By creating a system that uses deep learning to monitor and detect when an elderly person falls, it can alert family members or emergency services right away, potentially saving lives.

1 Introduction

The inspiration for this project comes from recent incidents where elderly individuals have suffered due to falls that were not promptly attended to, highlighting a critical area where artificial intelligence can make a substantial difference. By leveraging what we have learned and the deep modeling frameworks developed by researchers, we aim to create a system that not only enhances the safety of the elderly in their homes but also pushes the boundaries of how deep learning technologies can be applied to improve quality of life and care in our society.

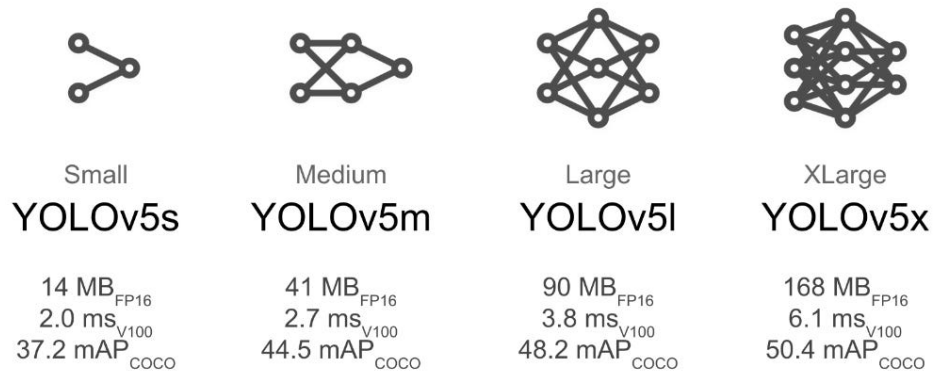

Figure 1: The image shows four pre-trained models of YOLOv5, with each model's icon complexity increasing from left to right, representing an increase in the size and complexity of the model.

```python
import cv2
# Example of IP address
url = 'http://admin:31214@10.0.0.27:8081/video'
# Capture a video stream using OpenCV's VideoCapture function
cap = cv2.VideoCapture(url)
while True:
    # Reads frames in a video stream
    ret, frame = cap.read()
    # If the frame is read correctly, it is displayed
    if ret:
        cv2.imshow('Video Stream', frame)
    # Press 'q' to exit
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
# Release the catcher and close all OpenCV Windows
cap.release()
cv2.destroyAllWindows()
```

Figure 2: Python code to call mobile equipment camera.

The objective of this project is to develop a human motion detection system that significantly contributes to eldercare by ensuring their safety and well-being. With deep learning experiencing rapid innovation across various fields such as image and speech recognition, natural language processing, and medical image analysis, we see a potent opportunity to apply these technologies to create a practical solution that addresses a pressing societal issue—the risk of elderly people suffering from falls without timely assistance.

Some previous articles explored the application of deep learning in human fall detection. In the paper "Improved YOLOv5 Method for FallDetection"[3], the author chose YOLOv5 and introduced the ECA mechanism to achieve cross-channel information interaction of feature maps. The accuracy can be improved through some parameters. PANet for the neck is modified to weighted BiFPN to obtain better fusion features. Finally, the effectiveness of the improved algorithm was verified on the Le2i fall detection data set, and the multi-camera fall data set was used to test the versatility of the improved algorithm. In the article "Accelerometer-Based Human Fall Detection Using Convolutional Neural Networks"[4], the author considers applying convolutional neural networks in this field and performing deep learning for fall detection in IoT and fog computing environments.

But we found that the data sets used for training in these projects are relatively small, and since most of the image acquisition methods come from cameras, the images in the database are also relatively single. So, the idea is to get various images from the internet as training and test sets and label these images. We will utilize a cutting-edge deep learning model (YOLOv5), known for its ability to efficiently process and analyze complex data. These models will form the backbone of our system, enabling it to analyze images or videos from smart home devices such as cameras to accurately

Figure 3: Here are some of the images included in the database we made. Pictures of people sitting down, people standing up, people falling down and people moving.

detect and differentiate human poses. During the process of adjusting parameters of our model, we continuously trained it to improve accuracy. In addition, a call to the mobile device's camera allows our model utilize detection on the mobile device.

2 Related work

2.1 Deep learning model (YOLOv5)

YOLOv5 (You Only Look Once version 5) is a popular object detection algorithm, an unofficial version of the YOLO family of algorithms, maintained and updated by an open source community. The core idea of the YOLO algorithm is to simultaneously predict multiple bounding boxes and class probabilities in an image through a single forward propagation, enabling efficient and fast object detection. YOLOv5 continuously develops this philosophy, offering several improvements and optimizations to improve the accuracy and speed of detection.[1]

The pre-trained model of YOLOv5 is trained for data sets of various sizes and can be used for detection tasks of various objects. These pre-trained models have been trained on a wide range of data sets, such as the COCO dataset, so they can recognize and locate a wide variety of common objects. Figure 1 shows four kinds of pre-training models, from small YOLOv5s to large YOLOv5x. Users can choose the right model according to their needs and resources. These models allow users to quickly start projects without having to train from scratch, especially when computing resources are

limited. The information corresponding to the text below the figure 1 shows the size (in MB) of each model, the inference time (in milliseconds) on the V100 GPU, and the mAP (mean accuracy) performance on the COCO dataset. This indicates a trade-off between model size and performance: larger models generally have better performance, but take longer to infer and have larger model files. Since we needed to use our own computer for parameter adjustment most of the time, we decided to use the YOLOv5s model due to the limited performance of private computers.

2.2 Human fall dataset

Over the past few years, a number of methods have been proposed for human fall detection. Most existing methods are evaluated based on pruned data sets. However, some researchers have come up with and produced very advanced databases on human falls. for example, the author of "A multi-modal multi-view dataset for human fall analysis and preliminary investigation on modality"[6] produced a dataset by capturing 50 Built from the activity of the subjects, it includes seven overlapping Kinect sensors and two wearable accelerometers. In the paper "Dataset for human fall recognition in an uncontrolled environment"[2], the authors created a database that simulated five types of falls and five types of activities of daily living. The data include front fall, back fall, left fall, right fall, sitting up fall and so on.

However, these databases only do a lot of image processing for a single scene. While, in real life, the camera needs to be able to effectively identify human body posture in different scenes. Therefore, the above database scenes are too simple, lack of rich scene transformation. Therefore, the database we make needs to contain a large number of rich scenes and different forms of different human poses.

3 Proposed Work

3.1 Call Camera

Due to equipment limitations, we did not use a real outdoor camera for this project. We used cameras on our mobile devices. The advantage of the mobile camera is its convenience, we can carry the device with us at all times, and do not have to be affected by the cables and network.

We used IP adress method to call the camera. The process implemented in pyton code is shown in the Fig 2. However, it is important to note that the two devices, namely the mobile device and the computer, must be under the same network.

3.2 Datasets

We used the database from Fall Detection dataset. The database contains a training dataset of 374 images and a validation dataset of 111 images, as well as corresponding coordinates and labels. In order to expand our database, we obtained a large number of pictures of human poses from the Internet. The images included pictures of people standing, people walking, people sitting, people moving, and people falling (Fig.3). We increased the number of images used for training to 2430 and the number of images used for verification to 854. This volume of databases met both our requirements for model

training and the hardware requirements of our equipment. We divided these images into three categories: up, bend, fall. For those who remained upright or walking, we classified them into up states. For pictures where the body was bent, such as sitting, crouching, or lying on one's side, we classified them as bending. Finally lying down, or losing center of gravity, was classified as the fall state.

However, such a large amount of picture data still could not meet our training requirements. In the process of training, we still needed detailed information about the relative image, including the coordinate position of the person in the picture and the category of the human posture in the picture. To make it easier for us to take the coordinates and labels and convert them into a format that YOLOv5 can read, we used a tool named makesense.ai. It was a free to use online image annotation tool that was ideal for preparing datasets in computer vision deep learning projects. Because it was browser-based, there was no need for complicated installation steps, and users simply visit the website to get started. It aimed to save users time in photo tagging by integrating the latest artificial intelligence models to provide label recommendations and automate repetitive tagging work. We ended up with a text file (in txt format) that corresponds to the number of images, and each line in the file represents a marked object. From left to right is the index of the object class (we labeled fall 0, up 1, bend 2), the X-coordinate of the object center, the Y-coordinate of the object center, the width of the object, and the height of the object. All coordinates and dimensions were provided in scale form, rather than pixel values, so that our model could accommodate input images of different sizes.

3.3 Parameter changed

In the process of experiment, we needed to continuously optimize the trained model. In order to improve the overall accuracy of our model and reduce the loss in the training process, we adjusted the following parameters.

- Batch size: It refers to the number of samples that are passed through the network and processed at one time when training a neural network. This parameter directly affects the training process of the model, including the learning speed, memory requirements, and training stability. Large batch sizes require more memory resources because more samples are propagated both forward and backward-across the network, while also making more efficient use of the parallel processing capabilities of modern computing platforms (such as Gpus and Tpus), which can lead to significantly shorter training times.[5] However, too large a batch can also result in a reduction in the number of steps per epoch, which can be detrimental to the network's exploration in parameter space. (Fig 4) The choice of batch size therefore requires a balance between training efficiency and model performance. The default batch size is 16. After our analysis, we believe that a higher batch size will be more conducive to improving the performance of our model. We tried 20,24,28, and 32. Finally, we decided to change the parameter size to 32 to meet our requirements for the model.

- Learning rate: It is an important concept in the field of machine learning and deep learning, and is often used in gradient descent and its variant optimization

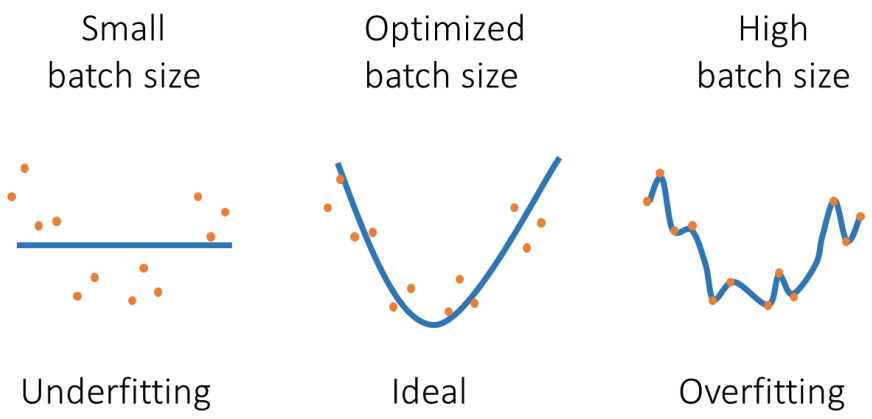

Figure 4: The effect of different batch sizes on the model.

algorithms. The learning rate defines the size of the compensation for updating parameters during optimization. When training a model, the learning rate represents how much the model parameters change with each iteration. If the learning rate is too high, the model may not be able to find the minimum value of the loss function. If the learning rate is too low, the model will converge very slowly.[7] However, since we are using the pre-trained model of YOLOv5, in order to retain the features learned by the pre-trained model and avoid a series of problems such as overfitting, we need to use a small learning rate to fine-tune. The default learning rate is 0.01, after our adjustment, we think that the most appropriate learning rate is set to 0.005.

- Epoch: An epoch refers to one forward pass and one backward pass of the entire data set during training. It is closely related to the adequacy of model learning and the risk of overfitting. By training multiple epochs, the model is able to learn the data multiple times, each time further adjusting its parameters based on what it learned the previous time. However, too many epochs may result in an overfit, and too few epochs may result in an underfit. The default epoch value is 20, and we need to dynamically adjust the results to prevent overfitting and underfitting. We tried 25,30,35,40,45. But we overfit when we tried 45. So we end up with an epoch of 40.

- Optimizer: The official optimizer for yolov5 is SGD, but after extensive testing we found that it did not live up to our expectations. This was because SGD tended to oscillate near minimum values rather than converge directly, and in standard SGD, all parameters were updated at the same learning rate, which might not work for all parameters. So we replaced it with Adam optimizer. Compared with SGD, Adam automatically adjusts the learning rate of each parameter, calculates the adaptive learning rate based on the update history of the parameter, and requires less manual adjustment of the learning rate.

4 Evaluation

In this section, we visualize the training process and the results. Are represented by figure 5 and 6.

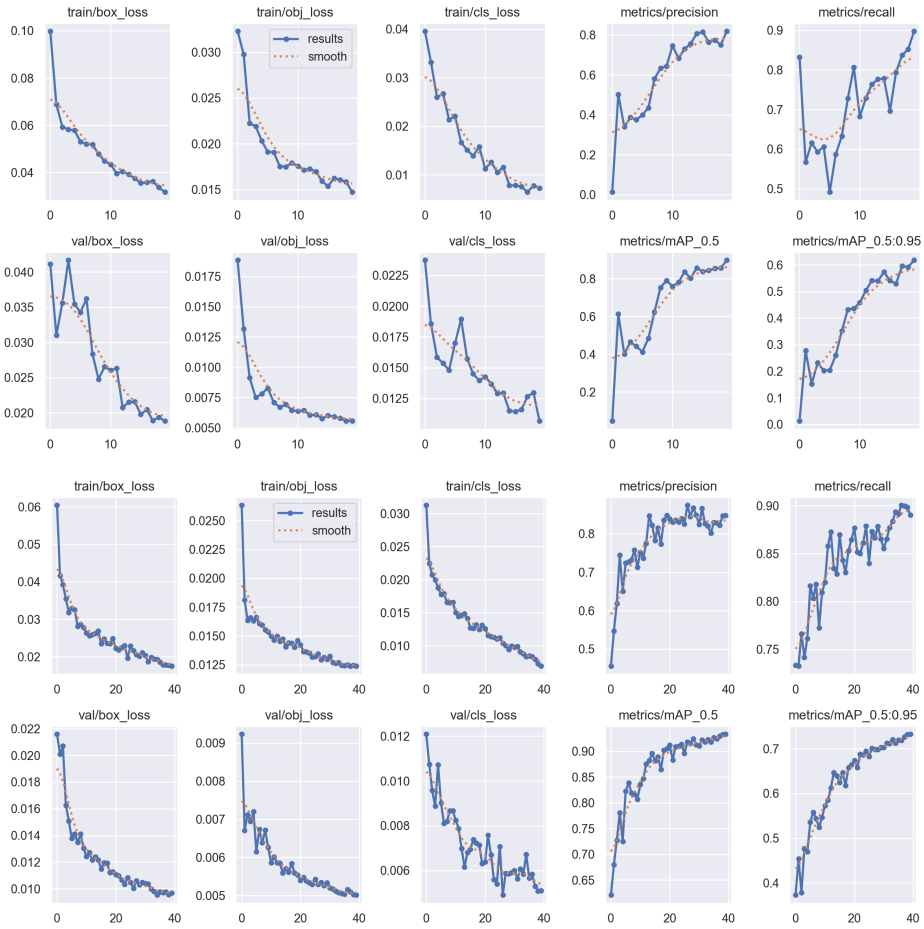

Figure 5: Performance Metrics dashboard when training a neural network, which shows changes in various metrics during training and validation. The above two lines represent the results of training our pre-trained model with default parameters. The following two lines are the results we get after adjusting the training parameters.

Figure 5 shows the changes of various performance indicators in the training process. The two graphs in the first column on the left display bounding box losses for the training and validation datasets, quantifying the discrepancy between the model's predicted bounding boxes and the actual bounding boxes. It is observed that the loss values in both graphs demonstrate a decreasing trend; however, further testing is necessary to ascertain if the lowest point has been achieved. The graphs in the second column illustrate object losses, which assess the model's precision in predicting the presence of objects. For object detection models, this entails identifying the existence of objects of interest within an image. Similarly, parameter adjustments are required to minimize the loss values. The two graphs in the third column present the model's accuracy in distinguishing between different classes on the training and validation datasets. The fourth column's graphs depict the model's accuracy and recall rates. Accuracy is the

proportion of correctly identified samples as positive by the model, whereas recall denotes the proportion of all positive samples accurately identified by the model. The final column's graphs display the model's mean Average Precision (mAP) metrics at various Intersection over Union (IoU) thresholds. mAP is a standard metric for evaluating the performance of object detection models. mAP0.5 refers to the mAP when the IoU threshold is 0.5, and mAP0.5:0.95 represents the average mAP as the IoU varies from 0.5 to 0.95.

Comparing the two graphs, we can see that after we adjusted the parameters, the training and validation loss curve corresponding to the second graph is smoother and more stable than that of the first graph, indicating better generalization ability and less overfitting of the training data. Accuracy and recall curves are also smoother, showing less fluctuation, indicating more stable and reliable model performance. In terms of mAP score, both IoU=0.5 and IoU=0.5:0.95 are higher and show steady growth, which means that the model has better object detection ability. In particular, there was a significant improvement in Maps with IoU=0.5:0.95, indicating that the model performed well even under more stringent standards of detection accuracy. Overall, our adjusted model shows better stability and performance on all metrics.

Figure 6 is two confusion matrix illustrating the classification model's performance across different categories. In this matrix, rows correspond to the classes predicted by the model, while columns represent the actual classes. Each cell indicates the proportion of samples genuinely belonging to a real class predicted by the model to fall into a specific class. The gradient scale on the right demonstrates that darker colors signify higher accuracy.

In these two confusion matrices, the rows represent the predicted class of the model and the columns represent the actual class. There are "fall detected "," up","bending"," background". As can be seen from Figure 6, when trained with default parameters, the model performed well on "fall detected" and "up" with true-to-truth ratios of 0.92 and 1.0, respectively, meaning that there was a high probability that both actions were correctly classified. However, the classification of "background" is somewhat poor, and some of the other three types of actions are misclassified as "background". When we changed the model using parameters, we made the accuracy of the resolution of both "fall detected" and "bending" slightly increase, while the resolution of the "up" type decreased slightly. Overall, both models performed very well, both before and after the parameter changes, but the modified model performed slightly better.

5 Conclusion

At present, we have completed the establishment of the training database, the adjustment of the parameters of the training process, the call of the camera of the mobile device and the correct prediction of the real-time video using the model. The final result can be seen here:Final result. Overall, our project has achieved good results and the results are very satisfactory. However, improvements can still be made in the following areas: First of all, in the training of the model, limited by our personal equipment, we can not use large pre-trained models for training. Second, the technology still has the potential to be applied to more practical areas, such as home medical robots, when the robot detects that someone has fallen, it can move next to the person for the first time

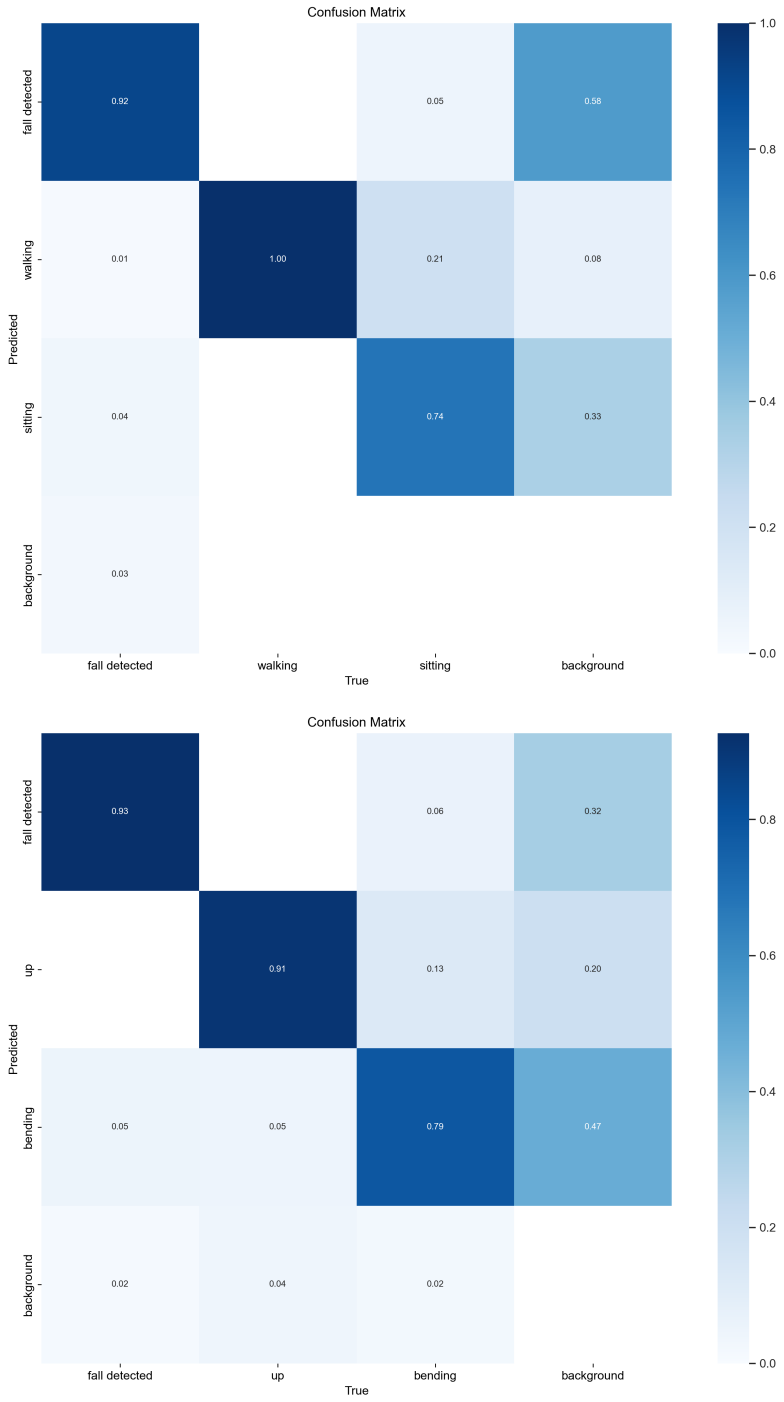

Figure 6: Confusion matrix drawn from the training model. The first matrix represents the confusion matrix trained on the pre-trained model using default parameters. The second matrix adjusts the training parameters and then trains the confusion matrix.

to implement rescue. So we still have a long way to go.

References

[1] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. 2020.

[2] José Camilo Eraso Guerrero, Elena Muñoz España, Mariela Muñoz Añasco, and Jesús Emilio Pinto Lopera. Dataset for human fall recognition in an uncontrolled environment. *Data in Brief*, 45:108610, 2022. ISSN 2352-3409. doi: https://doi.org/10.1016/j.dib.2022.108610. URL https://www.sciencedirect.com/science/article/pii/S2352340922008162.

[3] Jun Peng, Yuanmin He, Shangzhu Jin, Haojun Dai, Fei Peng, and Yuhao Zhang. Improved yolov5 method for fall detection. pages 504–509, 2022. doi: 10.1109/ICIEA54703.2022.10006129.

[4] Guto Leoni Santos, Patricia Takako Endo, Kayo Henrique de Carvalho Monteiro, Elisson da Silva Rocha, Ivanovitch Silva, and Theo Lynn. Accelerometer-based human fall detection using convolutional neural networks. *Sensors*, 19(7), 2019. ISSN 1424-8220. doi: 10.3390/s19071644. URL https://www.mdpi.com/1424-8220/19/7/1644.

[5] Samuel L. Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V. Le. Don't decay the learning rate, increase the batch size, 2018.

[6] Thanh-Hai Tran, Thi-Lan Le, Dinh-Tan Pham, Van-Nam Hoang, Van-Minh Khong, Quoc-Toan Tran, Thai-Son Nguyen, and Cuong Pham. A multi-modal multi-view dataset for human fall analysis and preliminary investigation on modality. pages 1947–1952, 2018. doi: 10.1109/ICPR.2018.8546308.

[7] Matthew D. Zeiler. Adadelta: An adaptive learning rate method, 2012.