50.039 Deep Learning Project Report

Yap Zhan Hao, Sean (1005153) Jolin Teo Hui Lyn (1005344)
Jyotit Kaushal (1005245) Teng Chen Chang Gabriel (1005027)

April 15, 2024

1 Introduction

Lack of knowledge, denial, fear of cost, or hesitation in witness situations – all of these are just some of the factors that have been tied heavily with first-aid-response teams often arriving just a bit too late to help save someone's life. This is hence an issue that we recognized was lacking a problem in the software space that would help give someone suffering from conditions like a heart attack, seizure, etc. Image classification is a tool that can be used to tackle several problems and help towards several solutions in different medical fields, from a macroscopic view like identifying abnormal patient behaviour in video streams, to the microscopic such as detecting anomalous regions in medical scans. For this project, we want to focus on detecting fall events from images, to aid in promptly calling for help in contexts like hospice care without wasting any time or risking human error.

1.1 Datasets and Features

The dataset we recognized for this project will be the Human-Fall-Detection Dataset.

Specifically, we will use the v5 of the dataset, comprising of 4497 annotated images. This dataset is auto-oriented, with images stretched to 640x640 pixels.

This dataset has already been used in similar tasks outside of a medical setting as well, showing it's potential. Some of the details of this model are listed as follows:

- Size of dataset: 4497 annotated RGB images. Split into 2690 train, 904 validation, 903 test images.
- Features: $640 \cdot 640 \cdot 3$ pixel data per image
- Outputs: Bounding box for regions containing a person who has fallen.
- Target Classes: Fall-Detected, Sitting, Walking

We realized that a real-time fall detection system might have certain cases where a person might intentionally sit/kneel down, or something that would be half-way between standing or sitting. Due to this we chose this dataset that uses three different classes to allow the model to differentiate between potential falling events while making sure we are reducing the number of wrongly classified instances.

Figure 1 shows a few sample images from the dataset. This visualization was generated from running YOLOv8.

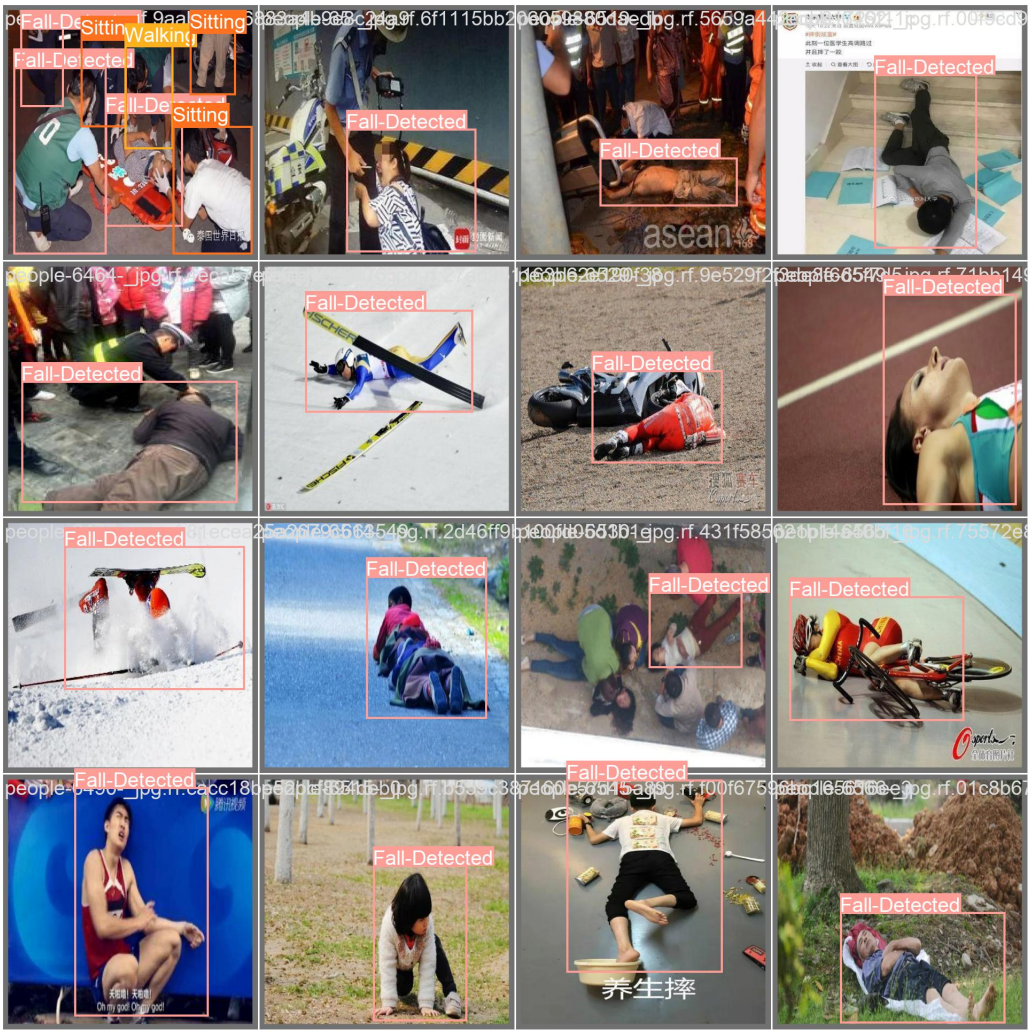

Figure 1: Sample visualization of images and labels

2 Architecture

The base architecture we explore in this project is adapted from that of YOLOv8

For our project, we will implement and adapt various Region-based CNN architectures, to explore and compare their performance at object detection.

Some of the features that we intend to include in our final model are as follows:

- **Backbone Network:** We will first train a CNN on a more varied image dataset to learn non-task-specific feature maps from input images.
- **Non-Maximum Suppression:** To suppress duplicate and highly overlapping bounding box predictions.
- **Post-Processing:** After inference, use post-processing techniques to filter and refine the final set of detected bounding boxes. This can include additional filtering based on confidence scores or further refinement of bounding box coordinates.

2.1 Loss Function

The loss function used for this project is adapted from the detection loss used in YOLOv8. In particular it includes terms for Classification, Localization, and Distribution losses.

2.1.1 Classification Loss: Cross Entropy

Cross Entropy Loss is a standard loss function used in classification tasks, including object detection. As our data is multi-categorical, we adopt Multi-class Cross Entropy to quantify the performance of our model in assigning the correct classes to predicted objects.

2.1.2 Localization Loss: Jaccard Similarity

Also known as the Intersection over Union (IoU) metric, this term in the loss function measures the overlap between the predicted bounding boxes and the ground truth bounding boxes. Object detection involves localizing objects within an image by predicting bounding boxes that tightly enclose them.

IoU loss calculates the similarity between predicted and ground truth bounding boxes by computing the ratio of their intersection to their union. It encourages the model to generate accurate bounding boxes that closely match the ground truth locations of objects.

2.1.3 Distributional Focal Loss (DFL)

Distributional Focal Loss is a variant of the focal loss function [1], which is commonly used in object detection tasks, especially when dealing with imbalanced datasets. Focal Loss addresses the problem of class imbalance by down-weighting the loss assigned to well-classified examples. This helps the model to focus more on hard, misclassified examples during training.

DFL generalizes this idea by incorporating the distributional information of the training data into the loss function. It does this by estimating the empirical distribution of the training data and using this distribution to modulate the loss function accordingly. This modulation helps to further mitigate the effects of class imbalance and improve the overall performance of the object detection model, especially in scenarios where the class distribution is highly skewed.

This is useful in the context of our project as our goal is to correctly identify people who have fallen, differentiating them from people who are in different poses, like walking, or sitting.

2.2 Layers

- ConvLayer: A custom convolutional layer with batch normalisation and ReLU activation. This is a simplified version of the one found in YOLOv8, also we replaced SiLU with ReLU activation.
- ConvBlock: Two ConvLayer modules put together where the first of the ConvLayers has a kernel of size 1. This is used to downsample or 'bottleneck' the number of channels.
- ResidualConv: The ConvLayer but with an additional residual connection.

2.3 Structure

The structure of our model follows somewhat of an encoder-decoder architecture, similar to YOLOv8.

The 'backbone' comprises a fixed portion which accepts images and a head adapter which will allow us to pass the multiple sets of outputs to the decoder heads.

The 'head' has two decoders, one for determining the class and the other for determining the bounding box parameters for each label detected.

2.4 Label Assignment: Task Aligned Assignment and Non-Maximum Suppression

During training, we can evaluate the performance of our model by comparing the predicted bounding boxes to the target boxes, assigning the prediction with the highest overlap and confidence score to a particular target. However, this is not possible to do during inference, since the target bounding boxes are not available to the model. As opposed to during training, where the model has access to the target boxes

3 Dataset Augmentation and Transformation

Training and validation datasets are augmented during training using the `albumentations` library, with the following transformations:

- Color Jitter, allowing the model to learn images in different levels of lighting, contrast, saturation, etc. that might be expected from in a real life setting.
- Horizontal Flip, allowing the model to learn from objects that could appear facing either direction which in turn helps maximizing the learning the model gets from each data point.
- Shift Scale Rotate, to help the model cope with small misalignments or rotations objects might have in a real world setting.
- Normalisation to help the learning less resource and time expensive

Consequently, each training/validation sample undergoes a different random augmentation every epoch, increasing data diversity while limiting the increase in training time.

Crucially, the bounding boxes must follow the image transformations. Hence, we used the albumentations package which offers support for bounding box transformations together with the image.

Additionally, images are normalized by the sample mean and variance of the training dataset, to homogenize the distribution of the data.

4 Model Checkpoints

Model checkpoints are saved every 4 iterations per epoch, storing both the current model weights, and updating the best-performing model so far. The best model is determined based on the total loss during training, and the accuracy during validation. The final model weights are also saved at the end of training and testing.

5 Evaluation

For reproducibility, we share a model checkpoint, trained with the following hyperparameters:

- No. of Epochs: 10
- Learning Rate: 0.001
- Head channels: (64, 32, 16)
- Backbone Hidden Channels: 512
- Dropout: 0.2

We will evaluate our model based on this checkpoint.

Figure 2 shows the change in the loss terms during training and validation. What is worth noting is the magnitudes of each loss term.Specifically, the classification losses are significantly higher than the other two terms. We attribute this to the choice of applying softmax to determine the class confidence scores, as opposed to the use of the sigmoid activation function in the original implementation of YOLOv8. One thing we did not manage to explore for this project was the fine-tuning and searching of the loss gain parameters, to scale each of the loss terms.

5.1 Comparison with YOLOv8

Given that our model architecture is heavily inspired from YOLOv8, we will compare the it is only fitting to evaluate our model against it.

6 Challenges

Runtime and Resource Limits

Although our architecture was inspired by YOLOv8, replicating that performance and speed has proven to be challenging. In particular, after including the computation of F1 scores as an evaluation metric, the training loop of our model is frequently terminated due to insufficient available RAM for the GPU.

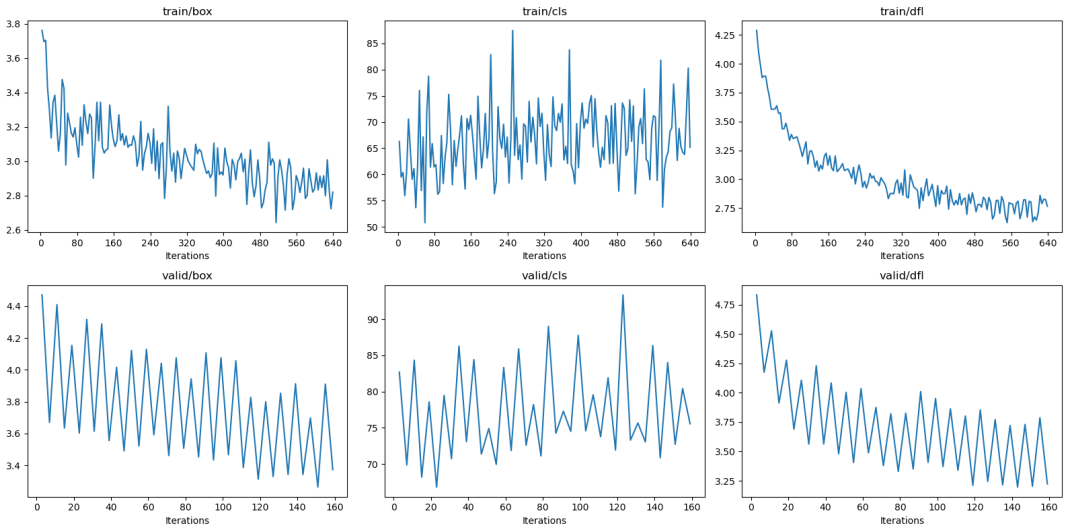

Figure 2: Evolution of loss terms across iterations

Figure 3: Evolution of F1 scores during validation

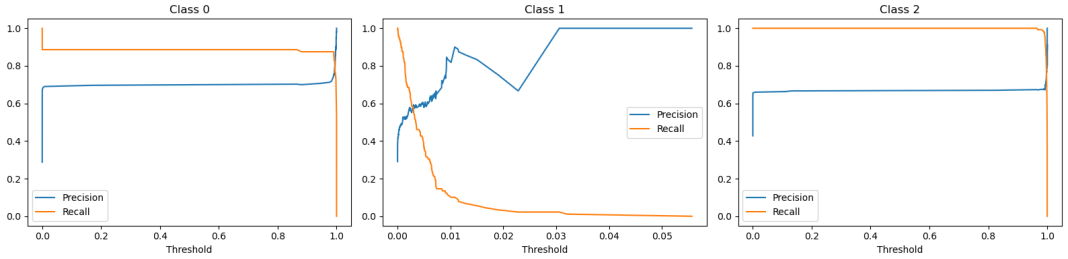

Figure 4: Precision and Recall at different thresholds, by class

To tackle this, we frequently call upon `torch.cuda.empty_cache()` to deallocate memory in the GPU assigned to no longer used tensors. However, this was not particularly effective, perhaps due to some synergistic effects of the implementation of our trainer function.

GPU Dependence

YOLOv8 relies heavily on powerful GPUs. Our inspired architecture is also very large and this was our main problem we faced while training our model, that training was extremely slow given the resources we have.

7 How to Run the Code

7.1 Prerequisites

Download the requirements needed to run the code.

- Python 3.11
- PyTorch 1.9.0
- Torchvision 0.10.0
- torcheval 0.0.7
- NumPy 1.21.2
- Matplotlib 3.4.3
- Scikit-learn 0.24.2

7.2 Getting Started

- Git clone the repository.
- Download the Dataset.

7.3 Commands required to run the Code

```
cd dl-fall-detection
```

Run the code in the `train.ipynb` notebook

8 Contributions

Everyone

- Coming up with and exploring different architectures
- Writing the report

Sean

- Coding the architecture
- Adapting the loss function
- Training the model

Gabriel

- Data Augmentation
- Training the model

Jolin

- Data Augmentation

Jyotit

- Training and Configuring Yolov Architecture

References

[1] Tsung-Yi Lin et al. *Focal Loss for Dense Object Detection*. 2018. arXiv: 1708.02002 [cs.CV].