How to use the Data Generator:

The data generator requires you to have your caches set up (see “Cache Instructions.pdf” for more details), as well as having continual access to the accompanying spreadsheet “Final Dataset.xlsx”, that allows the data generator to ensure that fall videos actually contain the fall events and not the frames before and after.
To ensure the spreadsheet is accessed correctly, download it from the github, and change line 25 in the “Data Generator.py” code to the path of the downloaded spreadsheet.

After defining and compiling your model (named “model” in this case), you can train the model on the cached data using the following command, assuming the caches are set up as described in the “Cache Instructions.pdf” file, you define your *initlal_steps_per_epoch* and your *initial_validation_steps* values to the percentage of the dataset you would like to use for each epoch of training,

```
"model.fit(train_dataset,
          epochs=100,
          steps_per_epoch=initial_steps_per_epoch,
          validation_data=validation_dataset,
          validation_steps=initial_validation_steps,
          verbose=2,
          callbacks=[early_stopping, checkpoint])"
```

How to Toggle Subsets:

Change any of the following values from “False” to “True” to turn them on, meaning the data generator will only generate videos if the videos match the subset turned to true. Please be aware there is no checking to ensure only one subset is toggled on at a time, so ensure you are selective and turning off old subsets when experimenting with new subsets:
In “DataGenerator.py”:

Subset	Line Number to Change
Eight Feet (Ceiling Height)	72
Nine Feet (Ceiling Height)	73

Ten Feet (Ceiling Height)	74
Senior	75
Hospital	76

How to Change Length of Input Videos (In “DataGenerator.py”):

1. Change the “numFrames” attribute found on line 19 to desired value.
2. Also change the “num_frames” attribute found in the data generator initializer on line 84.

Remember that the dataset was collected at 4fps. Minimal error checking is implemented to ensure excessively large values don’t break the system.

How to Ensure a Balanced Generated Dataset:

In the paper, it was mentioned how the dataset would automatically be balanced based on the class of the created video. In the existing dataset generator posted to the github, this feature is inactive in case users do not want this feature. In order to use this feature, the following changes must be made to the data generator:

1. Initialization method of the data generator:
    a. The parameter list must include “max_fall_samples”, keeping track of the maximum number of fall and non-fall samples for each batch
    b. The following attributes must be included in the body of the method:

```
self.max_fall_samples = max_fall_samples
        self.num_fall_samples = 0
        self.num_nonfall_samples = 0
        self.on_epoch_end()
```

2. The following method must be included:

```
def on_epoch_end(self):
        print("on_epoch_end: Shuffling filepaths and resetting counters")
        random.shuffle(self.filepaths)
        self.num_fall_samples = 0
        self.num_nonfall_samples = 0
        print("on_epoch_end: Completed")
```

3. The __next__ method:
    a. Should now start like this:

```
def __next__(self):
        global non_fall_count
        global fall_count
        X_batch = []
        y_batch = []
        #print("__next__: Generating batch")
        while (len(X_batch) < self.batch_size):
        #while (self.num_fall_samples < self.max_fall_samples &&
self.num_nonfall_samples >= self.max_fall_samples):
            #print ("self.num_nonfall_samples is ",
str(self.num_nonfall_samples))
            #print ("self.num_fall_samples is ",
str(self.num_fall_samples))
            if ((self.num_nonfall_samples >= self.max_fall_samples) and
(self.num_fall_samples >= self.max_fall_samples)):
                self.num_fall_samples = 0
                self.num_nonfall_samples = 0
                print("Both max fall and non-fall sample limits reached,
stopping iteration.")
                raise StopIteration

            if len(self.filepaths) == 0:
                print("No more filepaths left to process.")
                raise StopIteration

            random_file_path = random.choice(self.filepaths)
            path_parts = random_file_path.split(os.sep)
            video_name = path_parts[-2]
            video_name_key = video_name.replace('.avi', '')
```

b. When examining the type of video, and selecting the “start_min” and “start_max” values, the code should be updated to the following:

```
if '-NonFall-' in video_name:
                if self.num_nonfall_samples >= self.max_fall_samples:
                    continue
                start_min = 0
                start_max = max(0, totalFrames - self.num_frames)
                y_batch.append(0)
                self.num_nonfall_samples += 1
                non_fall_count += 1
            else:
                if self.num_fall_samples >= self.max_fall_samples:
                    continue
```

```
                y_batch.append(1)
                self.num_fall_samples += 1
                fall_count += 1
                start_min = max(0, firstFallFrameOfVideo - self.num_frames)
                start_max = min(firstFallFrameOfVideo - (self.num_frames //
2), totalFrames - self.num_frames)
                if start_min > start_max:
                    continue

            start_frame = random.randint(start_min, start_max)
```

4. Defining the following callback

```
class ResetCountersCallback(Callback):
    def on_epoch_end(self, epoch, logs=None):
        global non_fall_count, fall_count
        non_fall_count = 0
        fall_count = 0
        cache_train_generator.on_epoch_end()  # Call your generator's
on_epoch_end to shuffle
        cache_validation_generator.on_epoch_end()
        print(f"Epoch {epoch+1}: Counters reset - non_fall_count:
{non_fall_count}, fall_count: {fall_count}")

# Use this callback in your model.fit() call
reset_counters_callback = ResetCountersCallback()
```

   a. Don’t forget to include this callback when training your model.