GET-UP: SLEEP MONITORING USING FILTERING AND DEEP LEARNING

Anthony Vento, Jason Zhang

The University of Texas at Austin
Department of Electrical and Computer Engineering

ABSTRACT

In this work, we present Get-Up[1]: a multi-stage sleep monitoring application that detects if a user is oversleeping and produces a compact video summary of a user's sleeping behaviors. The application's decision-making engine has four key components: an audio fingerprinter that matches real-time audio with an alarm database, a motion detector that applies Gaussian filtering, an optional object detection module based on a pre-trained Residual Neural Network, and finally, a custom-trained Convolutional Neural Network (CNN) that predicts if a user is in his or her bed. Each component performs reliably, and the full pipeline results in an effective sleep monitoring system.

***Index Terms*—** Sleep monitoring, Gaussian filtering, CNN, object detection, ResNet

1. INTRODUCTION

Sleep analysis has a variety of interesting image processing applications, and of particular importance to this project are recognizing when a person is oversleeping and efficiently capturing a person's movements when sleeping. Such problems enable practical application of image processing techniques such as filtering and deep learning.

Oversleeping is a universal problem, especially for undergraduate students. Often, alarms by themselves are not sufficient to guarantee that an individual will wake up on time as they can easily be turned off or ignored. According to a recent study from Mattress-Inquirer of more than 1,000 people across all ages, approximately 55% of people oversleep and of those people, 75% miss work as a result [1]. Therefore, this is not only a problem for students but also for people of all ages.

Additionally, information about individual sleeping habits such as positioning and movement frequency can be useful for both medical and personal purposes, but such information is not always easy to obtain. For example, Restless Leg Syndrome (RLS) is a nervous disorder that causes uncomfortable sensations in the legs, affecting around 10% of the U.S. population. One method of diagnosing RLS is checking for restless sleeping movements [2]. This can be inconvenient to determine without any external aid, so some method of capturing an individual's sleeping behaviors would be beneficial to diagnosing RLS. Furthermore, awareness of one's sleeping habits can also be personally useful if for instance one wishes to avoid certain postures.

We aim to address both of these problems with our project. First, we develop a reliable way to detect if a person is asleep after their alarm has gone off. From our experience, having a friend or roommate check for oversleeping is the best way to ensure getting up on time. Therefore, if a person is still asleep, our system will text and notify a designated friend or roommate to wake that person up. This provides people with a secondary means to waking up on time rather than having to solely rely on alarms to get up. Second, we create a method to concisely record a person's sleeping patterns, thereby allowing people to playback their sleep to see how often they move around, see what position(s) they fell asleep in, and determine how long they took to fall asleep.

2. METHODS

The sleep monitoring system pipelines the outputs of several processing modules to produce a sleep summary and to check for oversleeping. To setup the system, a USB webcam is mounted to capture images from an aerial view of the user's room. When the system launches, a frame processor reads images from the camera and constructs the sleep summary while another thread interprets audio frames. If the audio listener detects an alarm, the sleep summary halts, and the system must decide whether to send any alerts.

Get-Up sends alerts for two classes of events: detecting multiple people on the user's bed and oversleeping. If the optional parental mode is enabled, the frame processor sends the current frame to an object detector that alerts the user's parents if multiple people are detected in the user's bed. Afterwards, we wait 15 seconds and then send a batch of 51 frames is sent to a trained CNN which alerts the designated friends if the CNN predicts oversleeping for the majority of the frames. This whole process can be summarized by the pseudocode shown in Algorithm 1 in the following page.

The following subsections describe in detail the four core components contained in Get-Up: the audio listener, sleep

[1]Code for this project is available upon request.

summary, multiple people (MP) detector, and oversleep detector modules.

Algorithm 1 System Pseudocode

```
while audio listener does not detect any alarms do
    frame processor builds video summary
    audio listener reads new audio samples
end while

video summary halts
if parental mode enabled then
    pass current image frame to MP detector
    if MP detector finds multiple people on the bed then
        send an alert to parents
    end if
end if

wait 15 seconds
pass next 51 image frames to oversleep detector
if oversleep detector predicts oversleeping for majority of
frames then
    send an alert to designated friend
end if
```

2.1. Audio Listener Methods

The audio listener uses a Python package called PyDejavu for detecting if an alarm goes off. The Audio Listener "fingerprints" a set of alarms, encoded as ".wav" files, and stores these fingerprints in a database. Then, when a new ".wav" file is read, it compares the alarms in the database to the new ".wav" file to see if there is a match.

The fingerprinting algorithm from PyDejavu starts out by creating a spectogram of the ".wav" file it receives. A spectogram essentially maps a given audio frequency f and time t to the strength of f at time t in an audio sample. Next, the algorithm finds the strongest points over all frequencies and times and combines them into a unique hash. These hashes are stored in a fingerprinting table, so that when a new ".wav" file is received, it determines the similarity between the new hash and all songs in the database to return a likelihood for each file [3].

However, we discovered that there were significant bugs in the PyDejavu package including a lack of support for Python3 and a few faulty methods. Therefore, we had to debug the code and modify some of the source code to get this the package to work.

2.2. Sleep Summary Methods

The sleep summary produces a concise video capturing the user's motions by applying an algorithm making use of Gaussian filtering to eliminate unnecessary frames. This design ensures that only frame changes indicative of significant motion are preserved in the video summary.

In an ideal camera model, if there is no motion between two captured frames I_1 and I_2, then $|I_1 - I_2|$ should simply be the 0 matrix. However this is not the case in practice due to noise from the camera and slight perturbations in the camera's position. In fact, $\sum_{i,j} |I_1(i,j) - I_2(i,j)|$ may become quite large if there is high spatial frequency and I_1, I_2 align slightly differently. Thus, $I_1 - I_2$ by itself is not a good indicator of motion between image captures.

To address noise between consecutive image captures, when the last image added was I_1 the sleep summary determines whether to add image I_2 based on $I_1 \circledast G - I_2 \circledast G$ where G is a 2-D Gaussian kernel. Since the Gaussian filter acts as a low-pass filter, the high-frequency content that would have been observed in $|I_1 - I_2|$ gets smoothed, so that $|I_1 \circledast G - I_2 \circledast G|$ is more closely correlated with motion between the image captures. In fact, the sleep summary chooses to add frame I_2 when the last added frame is I_1 if $\sum_{i,j} |I_1(i,j) \circledast G(i,j) - I_2(i,j) \circledast G(i,j)| > t$ where t is some threshold. The Gaussian kernel was chosen since it avoids leakage in spatial frequency and ringing in space. In Get-Up, G defaults to a circular kernel with radius 11 and chooses a threshold of 10^6 based on the size of the camera's images.

To give some indicator of time spent in each sleeping position, the sleep summary module amends each frame with a time stamp. With this amendment, still motion can be inferred by observing skips in time stamps in the video summary, which gets saved as a ".avi" file in Get-Up's directory.

2.3. Multiple People (MP) Detector Methods

The MP Detector functions in the optional parental mode, alerting the user's parents via email if more than one person is detected on the user's bed. This module is implemented with object detection using a ResNet-50 model pre-trained on the Common Objects in Context (COCO) dataset. As is, the ResNet-50 COCO model has already been trained to detect an arbitrary number of people in images to high accuracy, so detecting multiple people is a direct application of the model [4].

Get-Up uses the ResNet-50 model because it demonstrated better performance on the images collected in our set up than other object detection models such as You Only Look Once (YOLO) [5]. Models based on the YOLO architecture were not able to detect people in our collected images, but ResNet-50 was able to detect people consistently. We suspect that the performance difference between the two classes of object detectors is that the images in Get-Up's use case do not fully expose users on their beds. Most of the time, only the upper body can be seen, and this may not be sufficient for the YOLO architecture.

2.4. Oversleep Detector Methods

The oversleep detector aims to predict if a person is still in bed after their alarm has gone off. We initially tried the object detector and found that it took a long time to detect and that it missed a few cases. However, we did not want to miss cases in our oversleep detector since the oversleep detector is the crux of our system. After researching different approaches, we discovered that CNN's are a state of the art tool in image processing for binary classification. Therefore, we decided to experiment with CNN's using what we learned in class as background.

In order to use a CNN, we had to construct a dataset to use for training. We ended up collecting a dataset of around 700 images of a person sleeping in a bed and 700 images of a person away from their bed.

Next, we had to experiment with different architectures for our CNN. In class we learned a little background about what each of the layers do, but in the end, we had to make decisions about what layers we wanted to include where to balance training accuracy and testing accuracy to avoid overfitting and improve model performance. Our first architectural models were inspired from a Kaggle tutorial on CNN's, and from there we incrementally introduced design changes [6]. When determining the best architecture, we performed cross-validation with three folds. During cross-validation we trained on two of the folds and then tested and determined the testing accuracy on the other fold for all combinations of folds. Then, we determined the average testing accuracy across all folds. We did this for a set of different architectures to find which had the high average testing accuracy score.

After training a CNN on all 1400 images, we tested with several live images. We discovered that our CNN overfit the training images and the live performance degraded. Therefore, we decided to augment our input images to cover different cases which greatly helped our testing performance.

In the end, we went with the following architecture where the input is a 320x240 image:

Conv2D(num-filters = 64, size = (5, 5))
Conv2D(64, (5, 5))
MaxPooling2D(size = (2, 2), Stride = 1)
Dropout(probability = 0.25)
Conv2D(32, (5, 5)
Conv2D(32, (5, 5)
MaxPooling2D((2, 2), 2)
Dropout(0.25)
Flatten()
Dense(256)
Dropout(0.5)
Dense(1, activation = sigmoid))

Note that the activation for all layers is ReLu, unless specified. For this architecture, we trained on all input images for a different number of epochs and determined which model performed best. In our case the model that performed best was trained for 80 epochs.

Ultimately, in our system, we send a sequence of 51 straight frames to our trained CNN. If a majority, 26, of the frames, have an output probability of greater than 0.5, then we send a text to a friend letting them know to go "get-up" their friend.

3. RESULTS AND DISCUSSION

With all components pipelined together, the full system performed well especially under standard conditions. In this section, we discuss results and takeaways for each of the individual components.

3.1. Audio Listener Results

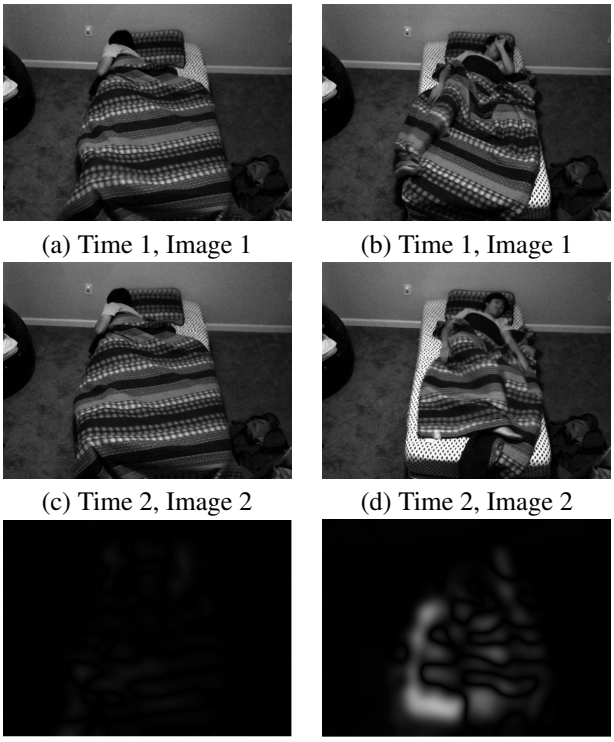
For the audio listener, we began by experimenting with auto-correlation, cross-correlation, and other techniques we learned in our signal processing courses. However, we did not receive much success with these approaches. Because this is an image processing project, we did not want to get too caught up on the audio processing part of the project so we began to look for Python tools to help us which led us to PyDejavu. Although the PyDejavu software has been tested and trained before on songs, we were unsure if it would apply to alarms as well. However, after integrating our system and testing, we found that as long as the alarm file that we pass to the system is greater than or equal to four seconds, the software will detect that an alarm is going off and no false positives will ever result. Note that we did have to write and implement a method that listens to frames from a microphone to create a ".wav" file and debug the PyDejavu package in order to get this system to fully function.

3.2. Sleep Summary Results

The sleep summary module effectively condenses video of one's sleep into a concise video summary. To see a full demonstration of the sleep summary, visit the following link `https://bit.ly/34TKUCx` for the original video and `https://bit.ly/2rY2TJc` for the sleep summary.

The effect of Gaussian filtering is specifically demonstrated in Figure 1 on the next page. Images 1a. and 1c. are consecutive image captures, and so are images 1b. and 1d. There is little noticeable movement between images 1a. and 1c., but images 1b. and 1d. are quite different. Thus, the sleep summary module would be expected to image 1d. based on 1b. but omit 1c. based on 1a.

The bottom row shows images 1e. and 1f. resulting from the magnitude difference between the Gaussian-filtered images of each column. As expected, image 1e. is mostly empty while image 1f. has many non-zero pixels. Since the absolute



(a) Time 1, Image 1 (b) Time 1, Image 1

(c) Time 2, Image 2 (d) Time 2, Image 2

(e) Gaussian Diff. Image 1 (f) Gaussian Diff. Image 2

Fig. 1: Demonstration of Sleep Summary Processing

pixel sum for image 1e. is below the threshold, the sleep summary module would not add the image to the video. On the other hand, image 1f. would be added given an appropriate threshold level.

Similar to the audio listener, we did have to write our own method to interface with the camera in order to capture frames. We found that there are many ways to configure the threshold and the Gaussian kernel, and this follows intuition as increasing the blurring effect can be compensated by decreasing the threshold. From empirical observation, choosing a filter radius of 11 and a threshold of 10^6 produced reasonable sleep summaries.

3.3. MP Detector Results

The performance of the MP Detector module is quite successful as the target objective in Get-Up is the same objective as the ResNet-50 model was trained for. As shown in Figure 2, the ResNet-50 model is able to detect when multiple people are on the bed, and can accurately draw bounding boxes around the people it identifies. However, since all the model does is count the number people in the entire image, a key assumption of our module is that if the model detects a person, that person is on the bed. This is a fair assumption since for most use cases, the bed will occupy a majority of the image frame. However, the model may fail in cases when this assumption is violated such as when people are near the bed but not on it.

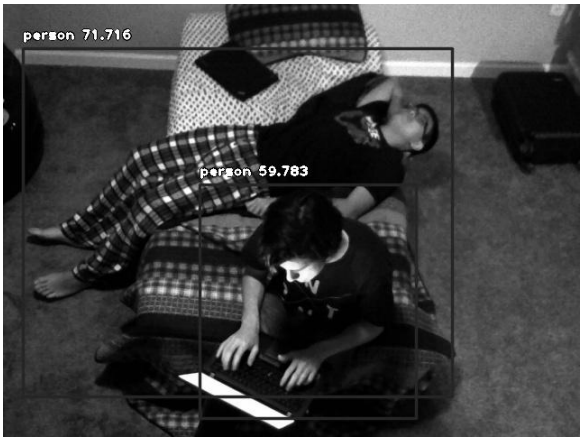


Fig. 2: Object Detection Bounding Boxes

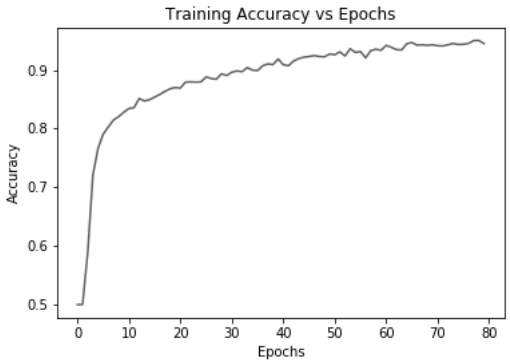


Fig. 3: Training Accuracy over Epochs

After observing the performance of the object detection module, the ResNet-50 model seemed like it would be a potential candidate in predicting for oversleeping. Under the assumption that a person in the frame will most likely be in the bed, predicting for oversleeping is the same problem as human detection. However, such a model has limitations since the premise is that it must detect a person to predict oversleeping. In many cases, an oversleeping person may be completely covered by sheets, allowing the ResNet-50 model to pass over such cases. To address these scenarios and accounting for the complex computations required of the ResNet-50 model, we decided to approach oversleep detection using a custom CNN, trained specifically for this classification problem.

3.4. Oversleep Detection Results

The CNN described in section 2.4 managed to accurately predict whether someone is oversleeping in an image. Figure 3 shows a plot of the CNN's training accuracy over 80 epochs on all images, revealing a final training accuracy close to 0.95.

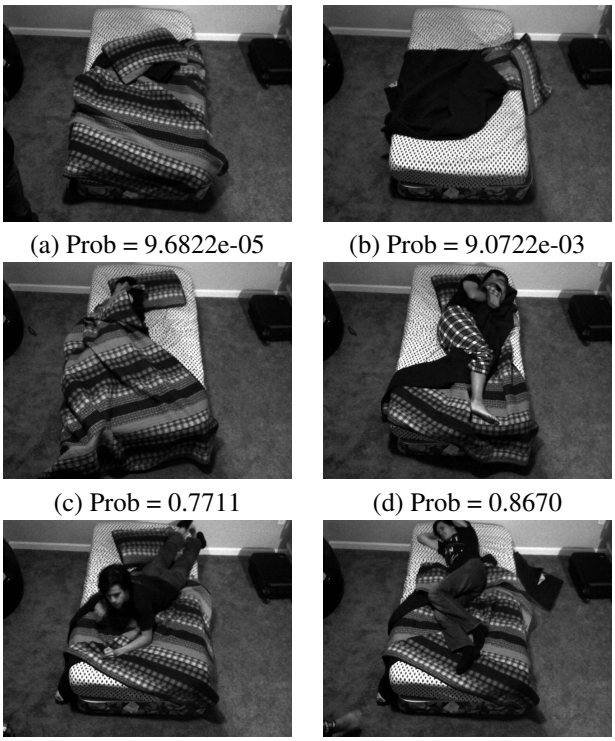
This accuracy appears to be justified as we have verified our CNN predictions on images outside its training set. Figure 4 shows a set of six sample images that the CNN predicts correctly. Images 4a and 4b have a ground truth probability of oversleeping of 0 while 4c - 4f have a ground truth probability of 1. The captioned values under each image shows the probability of oversleeping outputted by the CNN model, which all agree with the ground truth.

Training a comprehensive model to predict for oversleeping was the most difficult aspect of the system, and the changes that most impacted the model's performance dealt with making a comprehensive training dataset. In the earlier stages of Get-Up's development, most of the models suffered from overfitting to the extent that under cross-validation, both training and validation accuracy consistently converged to 1.0. From our experimentation, we identified two primary causes of overfitting: insufficiently varied training samples and an invariant camera angle.

Lacking sufficient variation in the training dataset was the first major obstacle to addressing overfitting. In early models, we observed that the CNN produced consistent and accurate predictions for scenarios somewhat similar to scenarios adequately represented in the training set. However, slight changes in a user's positioning would result in large drops in predicted probability. To address this problem, we integrated more cases within our training set to build a more comprehensive dataset. Such cases include extreme scenarios where the user is completely covered by the sheets and where the sheets are tossed onto the ground. We incrementally expanded the dataset upon every iteration of development, and consequently, the CNN's performance also incrementally improved.

Another major cause of overfitting was constructing the dataset from an invariant camera angle. For convenience of amassing thousands of training images for the CNN, the camera angle was assumed to be fixed throughout the whole process. The CNN would learn to master predictions for images at a particular orientation. However, such a model is extremely sensitive to even slight perturbations in the camera's orientation. To address this issue, we applied image augmentation to artificially generate more training samples mimicking different perspectives. The augmentation process involved applying a random combination of shear, zoom, rotation, and cropping, thereby augmenting the existing dataset with information from alternative orientations and perspectives. Figure 5 shows four examples of augmented images which demonstrate change in orientation and perspective when compared with the fixed-angle images in Figure 4. Image augmentation led to a large boost in the CNN's performance and simultaneously improved its robustness.

The final CNN model despite having been trained for one specific room should be universally applicable to all rooms provided that the bed is in the center of a new frame. While such arrangements would recreate circumstances for optimal

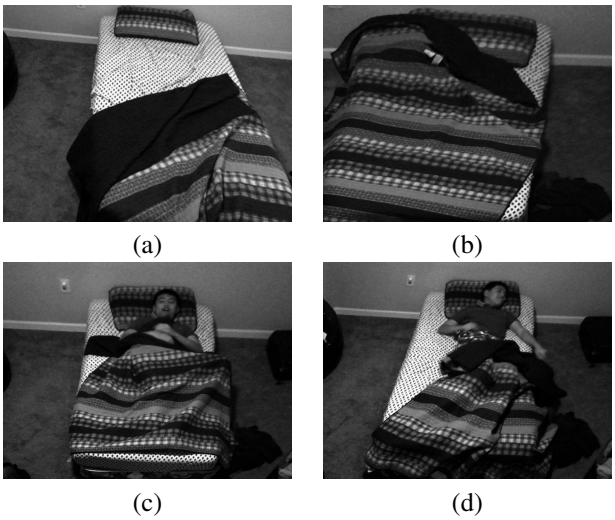


(a) Prob = 9.6822e-05 (b) Prob = 9.0722e-03

(c) Prob = 0.7711 (d) Prob = 0.8670

(e) Prob = 0.9198 (f) Prob = 0.9472

Fig. 4: CNN predicted probabilities of oversleeping. 3a, 3b have ground truth probabilities of 0. Likewise, 3c - 3f have ground truth probabilities of 1.



(a) (b)

(c) (d)

Fig. 5: Examples of Augmented Images

performance, we recommend that for new users, the CNN model should be further trained on 50-100 images of the new rooms to further reduce the risk of overfitting to room-specific features.

4. WHAT WE LEARNED

In creating Get-Up, we learned practical applications of image processing and deep learning. Firstly, we learned how to create a full system that integrates image processing that is applicable in day-to-day life. Fully connecting our various modules into a functional system was a fulfilling experience that provided us insight into how components of image processing fit together.

While linking together the various project components presented many challenges such as threading and synchronization, training our CNN model was the most valuable learning experience in terms of image processing. Developing the oversleep detector allowed us to implement the entire timeline of a CNN: creating a comprehensive database, designing the CNN architecture, and training the CNN.

To create a comprehensive database for the CNN, we learned how to automate generation of training images. Streamlining this process by taking advantage of image sampling enabled us to manually generate over 1,000 training images in reasonable time. Furthermore, we learned to make our dataset more robust by applying image augmentation to account for perturbations in camera orientation. Image augmentation was mentioned in class from a theoretical standpoint, but using this technique to strengthen our dataset allowed us to fully appreciate its impact.

By designing the CNN architecture, we learned about the building blocks behind CNN's in detail and witnessed the tradeoffs of various design choices. In class, we learned some theory about the effects of basic CNN structures such as feature maps, pooling, and dense layers, but building a full network from these structures by hand solidified our understanding. Some of our key realizations include the tendency for softmax to saturate output and the increased risk of overfitting by further deepening the CNN. Sometimes, our attempts to add more complexity to the CNN gave degraded performance when compared to simpler models.

Additionally, training the CNN by hand allowed us to explore training techniques such as k-fold cross validation. This was especially useful in the early stages of our model's development to give a realistic indicator of the model's performance. For example in the training process, we learned about the dying ReLu problem by observing stagnant validation accuracy after numerous training epochs, and reduced its presence for our CNN by adjusting the learning rate and depth of the architecture.

Outside the scope of CNN's, we also learned about the ResNet-50 and YOLO architectures for object detection and determined how Gaussian filtering can be applied to quantify similarity between image frames through the sleep summary module. It was neat to apply such a seemingly simple topic that we learned about in class, filtering, to develop a very neat tool, the sleep summary. Overall, this project enabled us to aggregate many of the image processing techniques discussed in class and provided us a valuable learning opportunity.

5. INDIVIDUAL CONTRIBUTIONS

In general, we ensured that work was split evenly between the two of us. Both of us worked together to create the system pipeline and develop the oversleep detector CNN, in particular, dataset construction with image augmentation, architecture design, and training. Individually, Anthony integrated audio fingerprinting and worked to figure out how to send texts from Python, and Jason designed the Gaussian filtering scheme for sleep summary.

6. FUTURE WORK

Given more time to complete the project, we would work on expanding two features: the MP detector and the oversleep detector.

For the MP detector, we would work on developing a custom trained model using transfer learning in order to capture a few more of the cases which it currently misses, including if two people are completely under the sheets.

For the oversleep detector, we would continue to work on testing our CNN on new settings and beds to generalize the model. We want the user to not have to add images to the training dataset in order to receive highly optimal performance. Therefore, we would work to avoid this case by adding more training images ourselves and by continuing to modify and train the architecture.

Finally, we would work on converting this entire system into an app so the user does not have to launch the system from their computer each time.

7. CONCLUSION

Overall, we developed a successful sleep monitoring tool that is able to detect and take preventative measures for oversleeping and concisely capture an individual's sleeping behaviors. Our system leverages deep learning and filtering techniques in practical applications and demonstrates reliable performance. Get-Up can apply to everyday life, and this in conjunction with aggregating our toolkit of image processing techniques made this project a valuable learning experience.

8. ACKNOWLEDGEMENTS

This work was completed for the final project for EE 371R: Digital Image Processing at The University of Texas at

Austin. We would like to thank our professor, Dr. Al Bovik for teaching this course and giving us insights into our project.

9. REFERENCES

[1] Mattress Inquirer, "Squandered by sleeping in: How oversleeping affects people," *Mattress Inquirer*, October 2019.

[2] M. Khatri, "Restless legs syndrome (rls)," *WebMD*, April 2017.

[3] W. Drevo, "dejavu," May 2019.

[4] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 779–788.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 770–778.

[6] Yassine Ghouzam, "Introduction to cnn keras," *Kaggle*, July 2017.