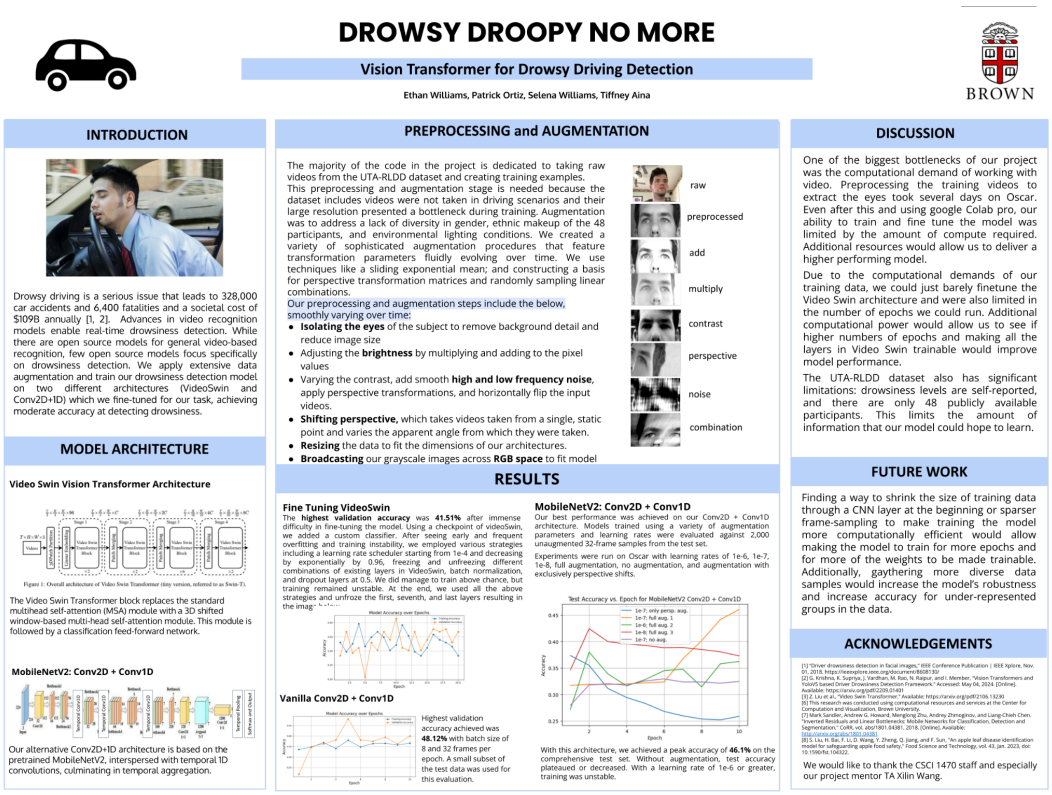
Drowsy Droopy No More - Drowsiness Detection For Safe Driving



Group Members:
Ethan Williams, Patrick Ortiz, Selena Williams, Tiffney Aina

Link to the Write-up's Google Doc (for images):
https://docs.google.com/document/d/14Yd4BN8G7mwARUi9HM7mwlE_-sxprvEqRwuEEjwOqis/edit?usp=sharing

Github:
You can access our repo here: https://github.com/EtomicBomb/deep-learning-final-project

DevPost:
https://devpost.com/software/deep-learning-final-project-zsci50

Introduction
Drowsy driving is a serious issue that leads to 328,000 car accidents and 6,400 fatalities annually according to the AAA Foundation for Traffic Safety. Additionally, the NHTSA estimates that drowsy driving related crashes cost society $109 billion annually, not including property damage. Recent advances in the accuracy and efficiency of video recognition models enable real-time drowsiness detection to be technologically possible. However, while there are open source models for general video-based recognition, few open source models exist that focus specifically on drowsiness detection. Additionally, a major bottleneck in the development of open

source drowsiness detection models is lack of data. We apply extensive data augmentation techniques that vary the brightness, contrast, perspective, and reflection of data samples in a smooth way, thereby increasing the robustness and amount of data our model can train on. With this augmented data, we trained our drowsiness detection model on two architectures: a state-of-the-art video action recognition model (VideoSwin) with a classifier on top and the Conv2D+1 architecture, achieving 46% accuracy.

Methodology

In our literature review, we couldn't find any video-based drowsiness detection models that used publicly available datasets for training. The only drowsiness detection models that were available used images, but we wanted to leverage the temporal relationship between image frames in video-based models as opposed to drowsiness detection from images alone.

Because of this, we instead took inspiration from 'Driver Drowsiness Detection in Facial Images' which was domain specific but used a private dataset and image based approach. We used the UTA-RLDD dataset, the largest publicly available dataset which is also used in 'Vision Transformers and YoloV5 based Driver Drowsiness Detection Framework'. We considered implementing a model similar in architecture to the Vision Transformers and YoloV5 paper, but instead decided to use the Video Swin Transformer architecture which was more computationally efficient, making it better suited for real-time drowsiness detection scenarios.

The Video Swin Transformer model achieves state of the art accuracy on Kinetics-400 and Kinetics-600 datasets for action recognition tasks, but was not specified to our task or our dataset. Additionally, while the dataset we used is the largest publicly available drowsiness dataset, it is still very limited, containing only 48 participants with limited gender and ethnic diversity. Because of this, one of the core contributions of our project was doing significant data augmentation and pre-processing to make our data more robust and compatible with the Video Swin architecture. From there, we fine-tuned the pre-trained model and added a classifier to specialize the architecture for our task for our experiments with the Video Swin architecture.

For the Conv2D+1 architecture, we ran a series of experiments with various epochs and varying amounts of pre-trained v.s. fine-tuned layers. To evaluate the impact of the data augmentation on model performance, we also tested with fully augmented data, partially augmented data, and non-augmented data for each of the architectures.

Data Augmentation & Pre-Processing

One of the most significant efforts in this project centered around preprocessing and augmentation that took several days of additional computation time and used multiple third-party models. The raw images from the UTA-RLDD dataset could not be fed directly into a model for training. There was a need for significant preprocessing and data augmentation. In this process, we developed several sophisticated tools to turn smartphone and webcam video into robust training examples.

The need for preprocessing and augmentation came from a variety of issues with the training data. The videos in the dataset are videos that were recorded by amateurs from smartphones

and webcams. Very few of the videos were taken in driving scenarios. These smartphone videos did not have isolated subjects, which would present opportunities for a model to learn false relationships between drowsiness and irrelevant details in the background.
There was a lack of diversity among participants in the dataset, and the variety of angles and lighting associated with a real-world driving scenario. This is due to there only being 48 participants included in the publicly available portion of the UTA-RLDD dataset.
The first stage of data preprocessing involved isolating the eyes of the subjects. This removed background noise and other details, allowing a model to focus on the variables that we believe can be examined to determine whether a subject is drowsy. To perform this isolation, we first recorded the eye positions in the source videos using `dlib.cnn_face_detection_model_v1`, with the pretrained model `mmod_human_face_detector`. The face positions were extracted using `dlib.shape_predictor`, with the pretrained model `shape_predictor_5_face_landmarks`. Applying these models to the source videos took several days using our available resources on the batch partition of Brown University's OSCAR supercomputer cluster. Additional effort was needed to create the transformations for the prerecorded eye positions using scikit-image. Next, 224x112, greyscale, eye-centered videos were rendered using PyAV. Rendering these videos took several hours on the same partition.
Because of the lack of diversity described above, an augmentation pipeline was necessary. Several custom video augmentation layers were developed as part of this assignment. We decided to develop an augmentation procedure that generates videos that more accurately match the kind of videos that are taken during driving. For example, as a driver passes under an overpass, a video may transition from bright to dark. As a driver shifts their head, the perspective of the video may change. These effects necessitate a more sophisticated augmentation procedure where the parameters of the augmentation change over the course of a video.

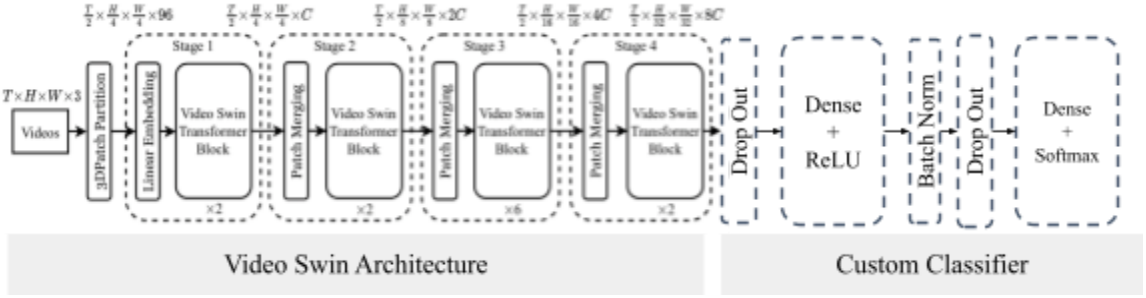
Specifically, we vary the contrast, multiply and add to the pixel values, add smooth high and low frequency noise, apply perspective transformations, and horizontally flip the input videos. These transformations are parameterized by constants that vary over time. For example, when we modify the contrast, there is a constant that controls the strength of the change. Where applicable, these parameters are sampled from a random normal distribution that varies over time, and smoothed with a sliding exponential mean. The high and low frequency noise is created from a procedure that applies a fast fourier transform to random data to generate a noise pattern that smoothly varies over space and time. This can simulate changes created by shadows and video compression artifacts.
The shifting perspective is one of the most interesting augmentations because it takes videos taken from a single, static point and varied the perspective angle from which they were

taken. This made the data more robust because it did a better job of accounting for jolts while driving, various driver heights, and various camera-to-driver angles. Four perspective shift matrices were found that form a basis for our transformation, and the actual perspective transform is found by applying a varying linear combination of these matrices to every frame. Our procedure is especially interesting because we only generate perspective transformations that are in-bounds, without any background fill needed.
Additional preprocessing was needed to make our data conform to the model's expected format. We cropped, resized, and padded the videos; and broadcasted the gray-scale pixel values across 3 RGB color channels.

Results with VideoSwin Model
Our highest accuracy with the VideoSwin architecture was 41.51%. We consider this a fair result given the difficulties our group encountered in the training process. Despite VideoSwin being well-documented and featuring useful model checkpoints to work from, it proved very difficult to fine-tune.

Despite our efforts in preparing an elaborate data augmentation pipeline, fine-tuning VideoSwin introduced a major computational bottleneck where we were forced to use a batch size of 2 since any increase in batch size caused memory issues, even with the A100 's 40GB of VRAM on Google Colab Pro. We tried many strategies to get the model to learn, making full use of what we learned in class and more.

For our experiments using Video Swin, we used the following architecture:



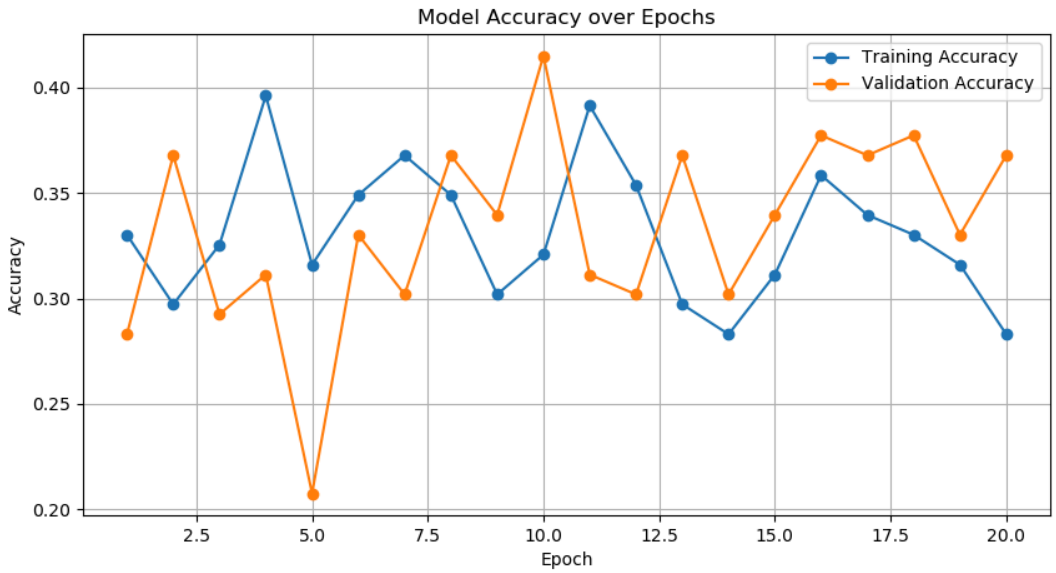
Many strategies were used to get the VideoSwin model to be fine-tuned while our custom classifier trained on top. First, we experimented with unfreezing specific layers in the architecture that would make the most impact. TFPatchEMmbed3D, the first layer, manages how embeddings are first formed. When unfrozen, we began getting some improvement in training above chance. Next, as expected, the final two layers like TFBasicLayer3 and TFBasicLayer4 - the stage 3 and 4 blocks above - when unfrozen, also helped the model to train.

Second, we tuned the learning rate being used with AdamW, the optimizer suggested by the paper. At first, we saw early and frequent overfitting, seen as dramatic oscillations in test accuracy. After experimenting with training rates like 1e-5, 1e-6, and 2e-5, we saw the overfitting

occur much later in the training process; in effect, we were really just pushing off inevitable overfitting to later epochs and slowing our training. So, we added a learning rate scheduler that employed exponential decay where we exponentially decreased the learning rate based on the number of steps. Starting from 1e-4, we updated every 100 steps and multiplied it by 0.96. After adding this, we saw much less oscillation in our loss. To help us avoid wasting paid compute from colab, we also do early stopping where we stop if, after 5 epochs, no decrease in validation loss is seen.

Third, we tried smaller changes to our custom classifier that could potentially help improve our model. We added batch normalization to normalize the activations of our dense layer. We thought it might help support the depth of our network and reduce how much the weights we started with - which were for another task - may impact our task. Additionally, we added dropout layers with 0.5 as their value since the model was prone to overfitting early on.

In the end, we managed to achieve a high accuracy of about 44% combining all of these together. However, this was very early in the training process of a particular run, so we do not consider this as a valid result. On a later run that was saved to our Github, we managed to get an accuracy of 41.51%. Here is a visualization of the training and validation accuracies.



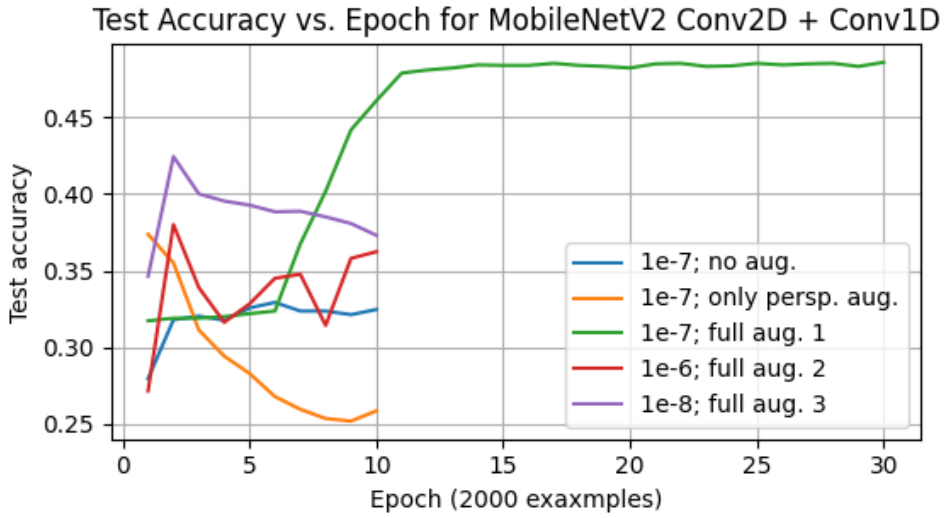
In the end, the chief issues were a lack of compute and a mismatch between the VideoSwin's task and ours. Our rationale was to use a model pre-trained on general video data and fine-tune it to our task. However, VideoSwin turned out to be hard to fine-tune without significant compute as video data with batch sizes higher than 2 would exhaust even the larger 40GB VRAM budgets with higher end Colab Pro GPUs. Our data, despite being reshaped and prepared for VideoSwin, did not seem to train properly, despite our best efforts.

We investigated other models after our difficulties with VideoSwin, which is described below.

Results of Experiments with MobileNetV2-based Conv2D+Conv1D Model:

Our best performance was achieved on our Conv2D + Conv1D architecture. This architecture was created in order to make use of existing individual-image models. MobileNetV2 [7] is a fast, high-performing low parameter convolutional neural network specialized for classifying ImageNet *images*. We believe that its early convolutional filters can be used in developing a video drowsiness classifier. MobileNetV2 has several downsizing blocks in which we can intersperse Conv1D blocks to synthesize information over time. The convolutional model is applied to each frame of the clip, and the results are aggregated across time using a final 3D convolution layer. After making these modifications, experiments were run on Oscar with learning rates of 1e-6, 1e-7, 1e-8, full augmentation, no augmentation, and augmentation with exclusively perspective shifts.

Models trained using a variety of augmentation parameters and learning rates were evaluated against 2,000 unaugmented 32-frame samples from the test set. None of the participants in the test set were shown to the model during the training process.



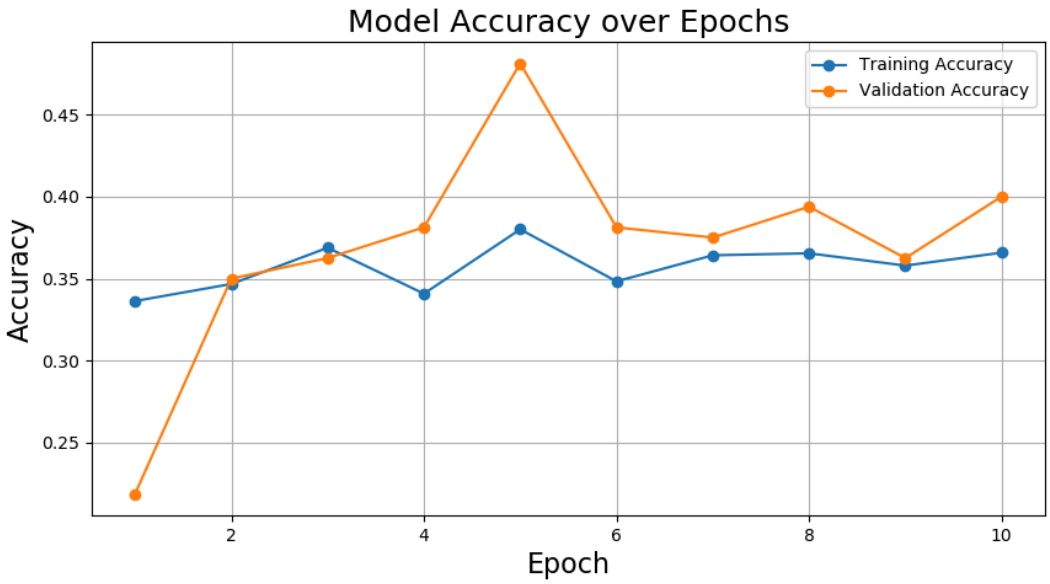
Broadly, results ranged from low 20% accuracy to high 40% accuracy. The high accuracy results are known to be statistically significant due to the large size of the test set. Our model is able to successfully predict the drowsiness level of participants, above chance (33%).

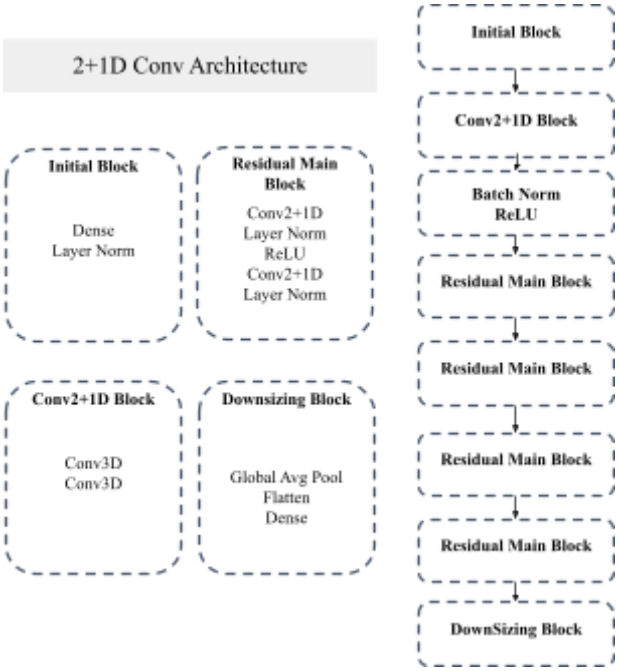
The best result, 48.6%, was achieved with every augmentation setting enabled, and a learning rate of 1e-7. The small learning rate was needed due to the fact that we were largely fine tuning an existing architecture, and the fact that our training set is highly augmented. Training was unstable with a larger learning rate. Without augmentation, performance on the test set decreased. This is likely due to overfitting. In the other configurations, performance on the test set stabilized. We trained the 1e-7; full augmentation run for an additional 20 epochs because of its early outstanding performance, which is shown in the graph.

Overall, we can conclude that the augmentation was highly effective in combating the low diversity of our data. We are able to predict the self-reported drowsiness label with a moderate level of success.

Results of Experiments with Vanilla 2D+1D Conv Model:

Vanilla 2D+1's highest validation accuracy achieved was 48.12% with batch size of 8 and 32 frames per epoch.





Discussion

One of the biggest bottlenecks of our project was the computational demand of working with video data. Even after cropping the videos to only include the eyes and nose region of the face, our ability to train and fine tune the model was limited by the amount of compute needed.

Challenges

Computational Bottleneck

Due to the computational demands of our training data, we could only finetune the last few layers of the Video Swin architecture and were also limited in the number of epochs we could run. To gain access to more compute, we used google colab pro, but even with a pro account we ran out of GPUs and often ran out of available RAM. We also utilized OSCAR to try address this issue but found that we had the best performance with google colab pro. Additional computational power would allow us to see if higher numbers of epochs and more making all the layers in Video Swin trainable would improve model performance. For the Conv2D+1 architecture, computational bottleneck was also an issue which led to slow training and limited learning. The nature of the data we used (real world, video-based) was naturally massive in size and even with the measures we took to crop and downsample the data were not enough to overcome with the limited computing resources we have. As mentioned in the results section, our max batch size of 2 caused high variance updates during training that seemed to seriously affect our ability to fine-tune VideoSwin.

Lack of Diversity Among Training Data

Although we used the largest publicly available dataset for drowsiness detection, our dataset had several limitations. These were: lack of ethnic and gender diversity (e.g. 9 women out of 48 participants and primarily white and Indo-Aryan participants). Additional limitations of our dataset were that drowsiness levels were self-reported, and thus can vary across participants, and that they were taken selfie-style and not in real-world driving conditions. We did everything we could to overcome these challenges by going through significant data augmentation to make samples more like real-world conditions. These augmentations increased the difficulty of the training task, since there was greater variance among data conditions. We saw this as a positive, important step because we wanted to tackle a real-world problem, which meant that we wanted to use data that was as close to real-world conditions as possible. However, this became a challenge while training the model due to the limited computational resources we had.

Future Work

Future work includes finding a way to shrink the size of training data (through a CNN layer at the beginning or sparser frame-sampling) to make training the model more computationally efficient and solve the computational bottleneck we faced. Gathering more diverse data samples to

increase the gender and racial diversity of training data would also be important from a model performance and ethical perspective.

Reflection

How do we feel about how our project came out?

Although we wish we could have achieved a higher accuracy, we are proud of the data augmentation techniques and the extent to which we addressed the computational challenges we faced. Our accuracy ended up being lower than our base goal, but we underestimated the magnitude of compute we would need which hindered our ability to train and improve our model.

Did our model work out the way we expected?

Our model did not work out the way we expected it to, but we learned a lot in the process and made creative, informed efforts to solve the different challenges that arose. We went into the project knowing that video data would be large to work with, and made initial decisions to address that by using architectures that were known to be more efficient. However, we quickly learned that even with extensive pre-processing and choice of architecture used, computation demands remained a massive bottleneck to model training and learning.

How did our approach change over time/what would we have done if we could do it over?

Professor Singh has mentioned throughout the semester, many machine learning decisions happen empirically, and we found this true with our project as well. While our conceptual understanding of the task guided our early decisions (like choosing a video-based architecture, cropping images to reduce resolution and decrease computation demand, choosing a relatively more efficient video-based architecture), as we trained the model we quickly realized that those efforts alone would not be enough to overcome the compute necessary to get the model to learn. When this happened, we tried a variety of changes to get better results. We utilized different levels of data augmentation in training, different batch sizes, a new architecture implemented from scratch, varying epochs and weight initializations for models we fine-tuned, and training our model on Oscar and Google Colab Pro. Some of these measures were more successful than others, but all were important parts of the learning process. If we could do the project over again, we would have chosen a task with more pre-existing literature. While there is literature for image based drowsiness detection, literature was limited for video-based approaches. This led to the decision of using a well-known general video-based action recognition architecture that was more efficient than other video-based architectures, but still presented massive constraints to training due to computational demands. Choosing an implementation with more task-specific literature would have allowed us to run into smaller, more directly problems (i.e. we made many changes to address computational bottleneck but could only do so much with the architectures we chose). Additionally, if we had known how computationally demanding the task was going to be, we would have put greater effort into downsizing the data ahead of time. As it was, we reduced image dimensions by cropping and localizing to the eyes region, but made the design decision to not run the videos through an extra CNN to maintain data compatibility with the Video Swin architecture. Another thing we may

have done if we did the project over again and still used the Video Swin architecture would be to have used a small subset of it, which would have allowed us to train with all of the weights as trainable for the subset we chose, instead of having to use some of the layers pre-trained weights due to a lack of available compute to train all the layers. That being said, we only know to make these changes because of the many difficulties we experienced with what seemed like a relatively simple task at the beginning.

What can we improve on if we had more time?
If we had more time and additional computational resources, we would like to increase the batch size and let more of the layers be trainable. If we had more time but no additional compute resources, we would have hard pivoted to an image-based approach that utilized regular images. We also would have liked to explore techniques that use classifiers that localize the eye and pupil positions like facial geometry models to see their effectiveness. We chose a video-based approach over an image-based approach initially because we thought that the temporal element of image frames would increase accuracy, but the computational demands ended up ultimately hindering model learning. Similarly, we chose a video-based approach over a facial-geometry approach because we thought that might cause the model to miss important data and felt that video-based methods were more similar to how humans naturally recognize drowsiness.

Biggest Takeaways
Our biggest takeaway from this project is that video data is really hard to work with due to the computational demands it requires. We also learned that having robust data pipelines is really important for generalization to real world scenarios. Particularly, in the research community where many datasets are created in static, lab settings, data augmentation is incredibly important to create a model that learns robust, meaningful relationships between the data and labels to make real-world performance possible. The last thing we learned is the difficulty of applying task-specific datasets to general purpose foundations models, and the difficulty of applying a dataset to an architecture that wasn't built for it for any domain and expecting it to work, even with significant alterations and fine-tuning. We felt that the learning in this project was informative and invaluable to our understanding of deep learning research and we look forward to applying these insights in future research.

References
[1] "Driver drowsiness detection in facial images," IEEE Conference Publication | IEEE Xplore, Nov. 01, 2018. https://ieeexplore.ieee.org/document/8608130/
[2] G. Krishna, K. Supriya, J. Vardhan, M. Rao, N. Raipur, and I. Member, "Vision Transformers and YoloV5 based Driver Drowsiness Detection Framework." Accessed: May 04, 2024. [Online]. Available: https://arxiv.org/pdf/2209.01401
[3] Z. Liu et al., "Video Swin Transformer." Available: https://arxiv.org/pdf/2106.13230
[4] Davis E. King, "Dlib-ml: A Machine Learning Toolkit," Journal of Machine Learning Research, vol. 10, pp. 1755-1758, 2009.

[5] PyAV. Accessed: May 06, 2024. [Online]. Available: https://github.com/PyAV-Org/PyAV
[6] This research was conducted using computational resources and services at the Center for Computation and Visualization, Brown University.
[7] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. "Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation," CoRR, vol. abs/1801.04381, 2018. [Online]. Available: http://arxiv.org/abs/1801.04381
[8] S. Liu, H. Bai, F. Li, D. Wang, Y. Zheng, Q. Jiang, and F. Sun, "An apple leaf disease identification model for safeguarding apple food safety," Food Science and Technology, vol. 43, Jan. 2023, doi: 10.1590/fst.104322.
[9] “Video classification with a 3D convolutional neural network | TensorFlow Core,” TensorFlow. https://www.tensorflow.org/tutorials/video/video_classification