Head Tilt Angle Classification Using CNNs

Sacheth Sathyanarayanan
sacheths@princeton.edu

Hari Ramakrishnan
harir@princeton.edu

Ish Kaul
ikaul@princeton.edu

May 12, 2020

Abstract

This report aims to improve upon the existing accuracies of Neural Networks applied to the detection and classification of the head tilt angle from a human image. To do so we analysed a dataset which contained images and their tilt angles and applied pretrained torchvision CNNs like ResNet and VGG to classify head tilt angles into buckets as needed. We conducted a variety of experiments to determine the optimal architecture and parameters for this task. Training and validation in all experiments were done for a constant number of epochs. The final results were compared, showing that ResNet had the best accuracy of 89% for 3 buckets, 78% for 5 buckets and 69% for 9 buckets with finetuning. In addition, the trained network can evaluate this on 200 images in under a second. This is an improvement on existing literature, and could be converted to an API/extension in the future with more experimentation with other new networks.

1 Introduction

Recently, many companies have adopted automatic scrolling features into laptops and cellphones by using the face image of the user to detect the tilt angle or using eye tracking[2]. In a controlled experiment, where readers were exposed to manual as well as automatic scrolling for reading text on screen, there was a clear preference for automatic scrolling as it increased reader speeds and decreased fixation times [3]. Additionally, increased speeds of reading also correlate with high reading quality from low text-error identification[4], clearly demonstrating its superiority over manual scrolling. While scrolling is one way to implement this feature, numerous other applications could be realised, including shifting browser tabs, gaming controls and accessibility equipment. Indeed, if performed almost instantaneously in real time, there all of these applications are highly practical and beneficial. One such application/API has been implemented, albeit using eye gaze path tracking, and has extensions for browsers, in addition to games [5]. The constraint however is to get a high accuracy in detecting or classifying the tilt angle in real time. Current techniques for auto-scrolling involve using a gimbaled sensor system with screen gaze concentration mappings [6], and detecting barycentric coordinates of nostrils from darkness variations using a USB camera[7]. In fact, the Swift programming language by Apple has an ARKit library using a 3D/2D mesh to get coordinates of image feature geometrically[8]. Another potentially useful way is to use Neural Networks to make them learn how to classify or detect tilt angles from images[9]. The use of Neural Networks, particularly Convolutional Neural Networks (CNNs), is well documented in image classification, and so we could try to use them for tilt classification for a range of angles.

2 Design And Implementation

All code [1] was written in Python in Google Colaboratory, utilising its GPU functionality. We used the Head Pose Image Database [10], which contains 2790 images of the faces of 15 different individuals along with the orientation, for example in Fig.1. It contains the vertical (tilt) and horizontal (pan) orientation angles of each image. We only operated on the tilt angles, which were present in 9 categories: -90, -60, -30, -15, 0, 15, 30, 60, and 90 degrees. This information was extracted and 300 images (10.1%) were used for validation while the rest were used for training. The images were all 288×384 pixels in dimensions. The torchvision library models ResNet-18 [11] and VGG-11 [12] were used, which have been pretrained on the ImageNet database. The VGG architecture is depicted in Fig. 2, ResNet blocks in Fig.3a[13] and the ResNet-18 architecture in Fig3b [13].

Figure 1: Example Head Pose Images

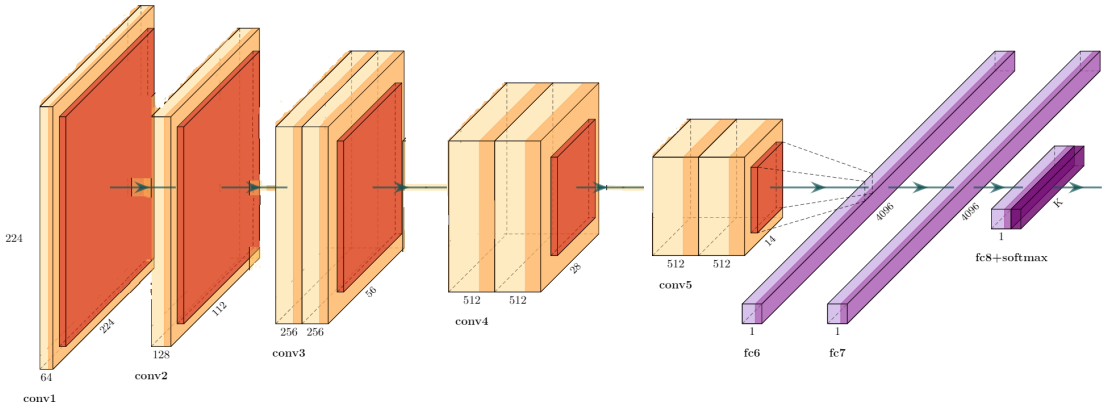
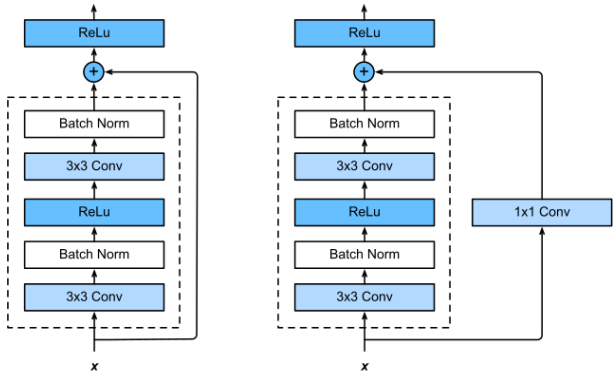
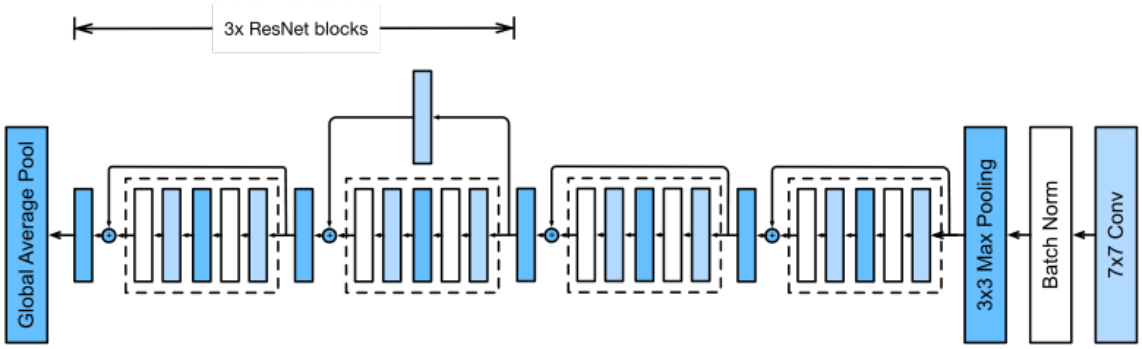


Figure 2: VGG-11 Architecture

These models were then finetuned, allowing the entire model to update weights, as well as feature extracted, allowing only the last layer to update weights. The cross entropy loss function was used to calculate the loss. Torch.CUDA() was used to implement the GPU functionality to speed-up the training process. All experiments were conducted using a batchsize of 30 and training proceeded for 30 epochs. All of this was done as described in the pytorch documentation[14]. The specific experiments are detailed below:



(a) ResNet Blocks



(b) ResNet-18 Architecture

Figure 3: ResNet-18 [13]

1. **Learning Rate Parameter (η) Variation** To investigate the effect of variation in the learning rate hyperparameter on the training process and final accuracy results, we started from a baseline of ResNet-18, using a momentum parameter of .9 for all experimental trials, classifying our images into 9 output buckets. We conducted trials with η taking on the following values: .00005, .0005, .0001, .001, .005, .01, .02. For each trial, we collected the best validation accuracy. After each of the 30 epochs of training, the model was tested against the validation set, and the best of these 30 accuracies was recorded. We also obtained visualizations for the training and validation loss curves throughout the training process. This allows us to qualitatively and quantitatively assess the training process for each value of η, and ultimately select the best value for our final models.

2. **Architecture Variation** We trained our network with the ResNet-18 and VGG-11 models that were pre-trained on the ImageNet dataset. We fine-tuned the networks by replacing the last layer (which is a fully connected layer with an output size of 1000, since ImageNet has 1000 classes) with another fully connected layer with an output size of 9.

3. **Feature Extraction** In addition to the fine-tuning explained above, we also performed feature extraction, wherein we still replaced the final layer, but we only updated the parameters of this new final layer.

4. **Number of Buckets Variation** For the purpose of our application of mobile/web scrolling, we don't necessarily need 9 buckets; we can make do with 3 buckets (one bucket corresponds to scroll up, one to scroll down and another to no scroll) or 5 buckets (where we have two speeds of scrolling in both directions in addition to one no-scroll bucket). We varied this parameter and trained and tested our network.

3 Results

3.1 Learning Rate Parameter Variation

Table 1 contains our Best Validation Accuracies for each value of η that we tried in our Learning Rate Experiments.

Table 1: Best Validation Accuracies from Learning Rate Variation

η	Best Validation Accuracy
.00005	47.00%
.0001	46.70%
.0005	61.33%
.001	69.67%
.005	66.00%
.01	59.00%
.02	57.00%

To represent the data another way, we can plot the best validation accuracy against the log of the learning rate parameter. This is represented in Fig. 4.

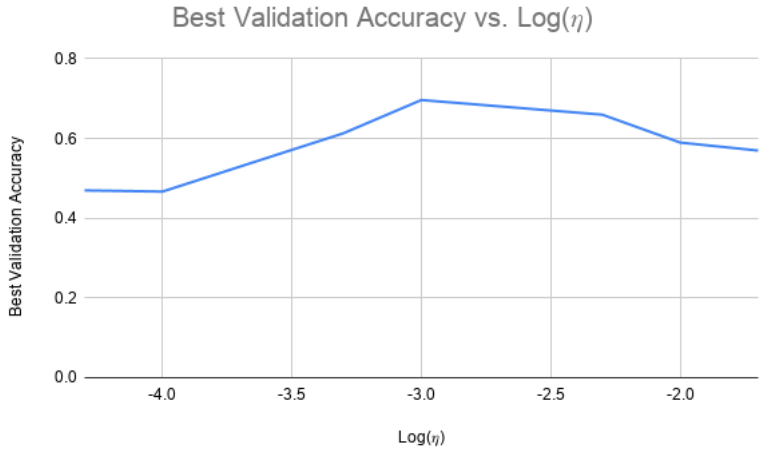


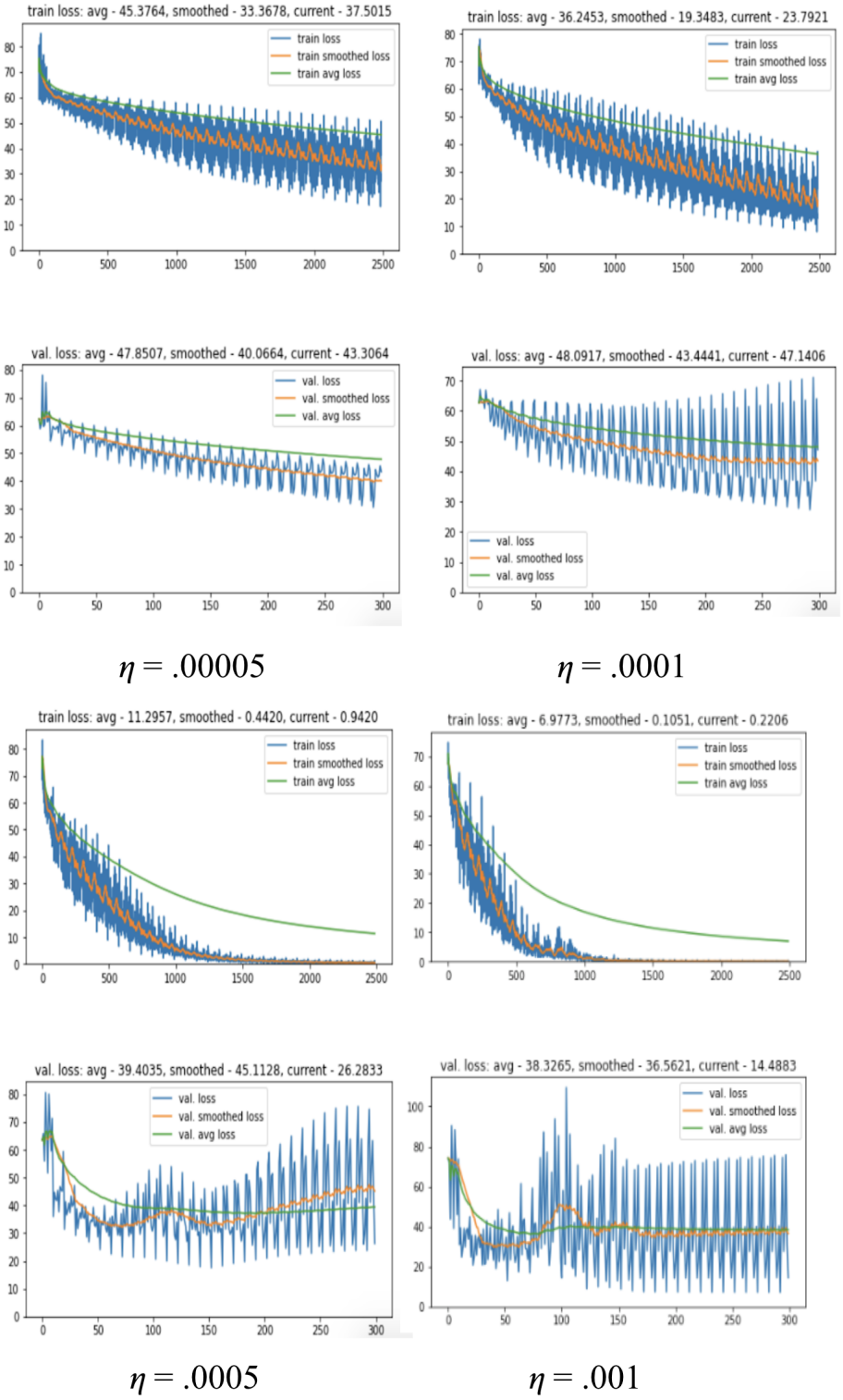
Figure 4: Log(η vs Best Validation Accuracy)

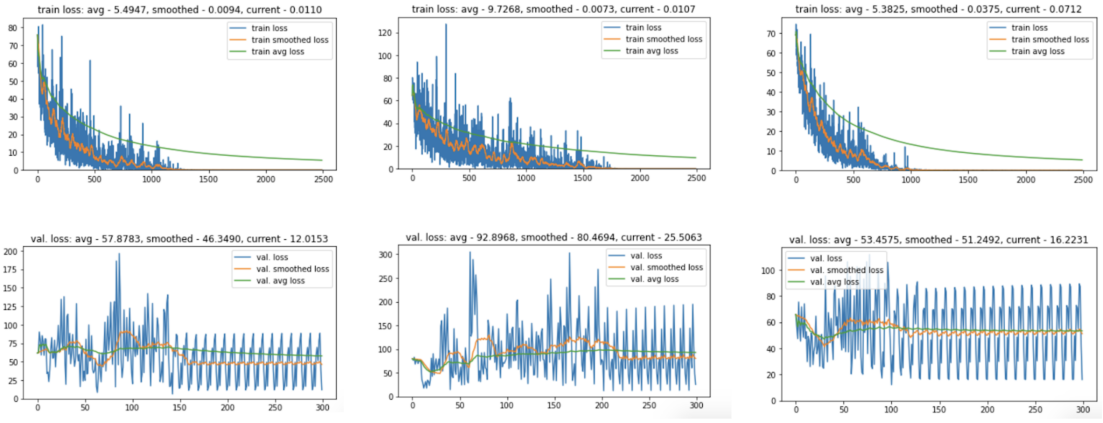
We can see that the peak of the graph in Fig. 4 corresponds to the trial where we set $\eta = .001$. This is thus selected as the value that we proceed with for training in the rest of our experiments.

For each of these trials, we also obtained the graphs of the training and validation loss curves over time as training progressed. These are presented in Fig. 5.

Figure 5: Training and Validation Loss Curves: η Variation

(Loss Function Value on y-axis; Number of images presented on x-axis)





$\eta = .005$ $\eta = .01$ $\eta = .02$

For all the preceding graphs, the blue curve is the actual value of the loss function after each image presented to the network. The orange curve is the value of the loss function averaged out over several images in order to remove some of the noise in our visualization. Finally, the green curve is the average cumulative value of the loss function up till that point in training.

In the graphs, we see that with an extremely low value of η, training progressed very slowly, and while the training and validation losses were decreasing, they were doing so at a very low rate. Increasing η allowed training to progress at a much faster rate, and we see that the training and validation loss curves reached a plateau by the end of the 30 epochs, indicating that more repetitions of training on the same data would not have yielded better results in these cases.

As we continue to increase the value of η, we see this trend continues, as the loss curves decrease even more sharply initially, before flattening out. We see that the plateauing of the validation curves also occurs earlier and more distinctly than with lower values of η.

In all of these trials, we observe a periodicity in the loss curves, as they have consistent sharp peaks and valleys. We hypothesized that this was due to the order in which we presented data to the model, which was identical for each iteration of training and validation phases. This was confirmed through later experimentation, as we later proceeded to train and test a network with the images being presented in a random order. When we did this, we observed that this periodicity was not present, and the peaks and valleys had disappeared, however, we still obtained the same accuracies as we had without this randomization. The resulting training and validation loss curve from this trial are presented in Fig. 6.

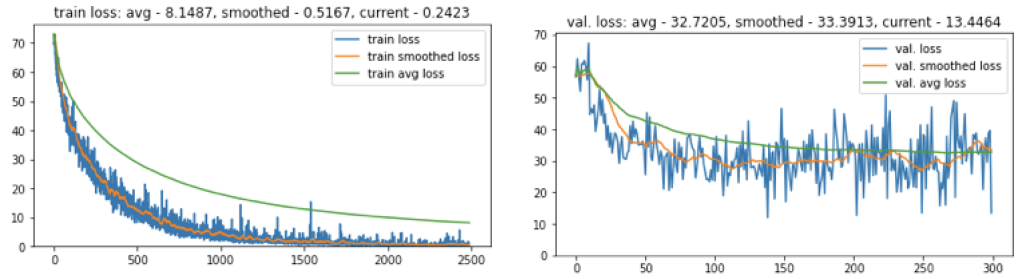
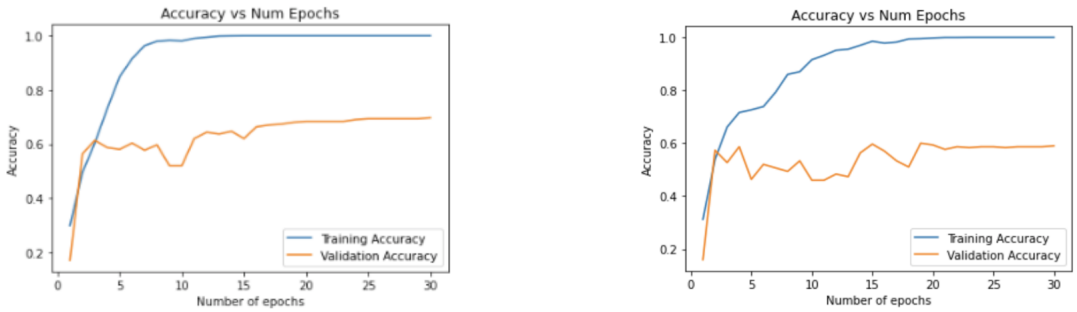


Figure 6: Training and Validation Loss Curves with Randomized Presentation of Images

3.2 Architecture Variation

We obtained a best validation accuracy of 69.67% with pretrained ResNet-18 and a best validation accuracy of 60.00% with pretrained VGG-11 as in Fig. 7. This difference makes sense since ResNet is a superior model that trains better since it has residual layers.



ResNet-18
Best val. acc. = 69.67%

VGG-11
Best val. acc. = 60.00%

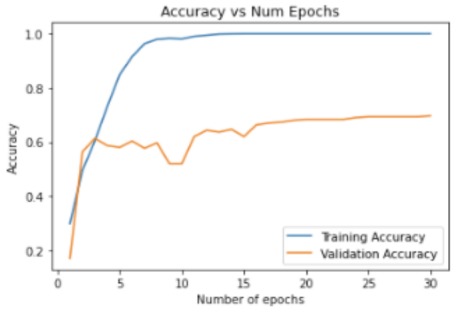
Figure 7: Architecture Variation

3.3 Feature Extraction

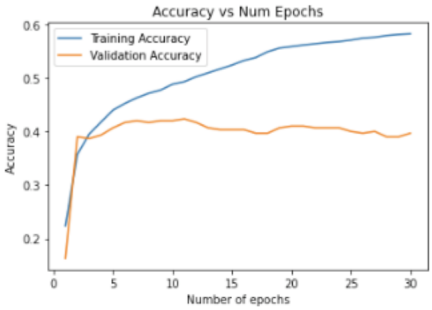
We performed feature extraction on the ResNet-18 model and compared it to finetuning. We obtained best validation accuracy of 42.33% with feature extraction and 69.67% with finetuning as in Fig.8. This makes sense since finetuning changes all the parameters in the network so the model will be more accurately trained for our purposes.

3.4 Number of Buckets

We trained and tested ResNet-18 with 3 and 5 buckets (in addition to the usual 9 buckets). We obtained best validation accuracy of 78.67% with 5 buckets and 89.00% with 3 buckets as in Fig.9. The accuracy obviously increases with fewer buckets since there are more training images of each class and less classification to perform.

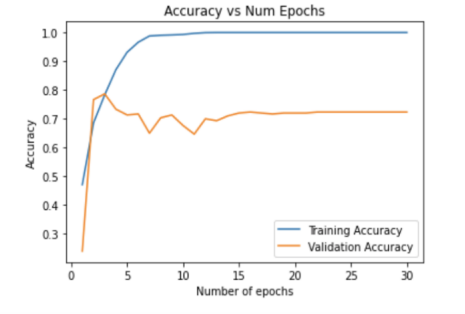


ResNet-18
Fine-tuning
Best val. acc. = 69.67%

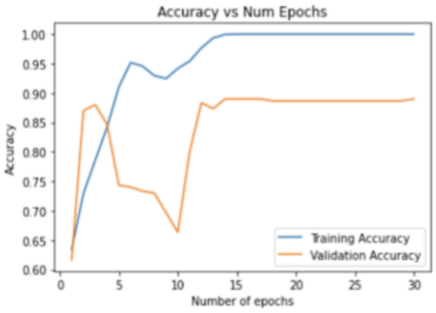


ResNet-18
Feature Extraction
Best val. acc. = 42.33%

Figure 8: Feature Extraction



ResNet-18 - 5 Buckets
Best val. acc. = 78.67%



ResNet-18 - 3 Buckets
Best val. acc. = 89.00%

Figure 9: Number of Buckets Variation

4 Discussion of Strengths and Weaknesses

We can evaluate the strengths and weaknesses of our project in the context of the possible use cases of our networks. We expect our project to have applications involving real-time scrolling on both desktop and mobile platforms. This would create a new way to interface with a device that could be helpful when navigating long webpages, articles, or menus. In order for this interaction to be the most natural, the calculation of the user's head tilt angle (and subsequent scrolling) would have to occur in close to real time.

This is one strength of our model. Our empirical testing showed that a 3 bucket ResNet trained model was able to compute the head angles of 200 input images in under a second. This speed is more than adequate for all scrolling applications on mobile and desktop, and would likely be sufficient for all but the most intensive gaming applications.

The biggest improvement that needs to be made before a widespread commercial application could be considered is the accuracy of our angle detection. Even with 3 buckets which can capture scrolling down, no scroll, and scrolling up, our best validation accuracy was 89%. This would not be sufficient for widespread use. To be able to achieve the accu-

racies required for commercial use (consistent 99% or more accuracy) would require more experimentation with different network structures to pinpoint the best suited network. It would also require much more training data in order for the network to adequately learn and thus achieve the desired accuracies.

The controlled nature of our current dataset also would also pose a problem to the real world performance of our models. The Head Pose Image Dataset contains images of peoples heads centered in the frame, with a constant distance from the camera and a constant neutral background. Our validation images and the training images both came from this dataset, but in order to use this model for a more widespread application, we would need to get access to a far more varied set of data on which to train. This would also hopefully improve the generalization performance of our model and thus allow it to be used in a wider variety of settings.

5 Next Steps

The current version of our model is best suited for the mobile/web scrolling application. The next steps, as such, would be to either implement a Browser extension (such as a Google Chrome Extension or a Safari Extension) or an API with end-points that return scrolling speed and direction. Similar features could also be implemented for other applications like gaming controls or browse tab shift.

References

[1] Code: Github

[2] Etherington, D. 2013. Samsungís Galaxy S IV will scroll content based on eye movement, report says. `http://techcrunch.com/2013/03/04/samsungs-galaxy-x-iv-will-scroll-content-based-on-eye-movement-report-says/`

[3] S. Sharmin, et. al. *Reading On-Screen Text with Gaze-Based Auto- Scrolling*. Proceedings of Eye Tracking South Africa, Cape Town, 29-31 August 2013

[4] Y. Hsieh, et. al. *Reading performance of Chinese text with automatic scrolling*. Proceedings of the 12th International Conference on Human-Computer Interaction: HCIí07, 311ñ319.

[5] O. Špakov. 2012. ETU-Driver, `http://www.sis.uta.fi/~csolsp/downloads.php`

[6] J.H. Lemelson, et. al. 2003. *System and Methods for Controlling Automatic Scrolling of Information on a Display or Screen*. US 6,603,491 B2

[7] N. Nakazawa. *Recognition of Face Orientations Based on Nostril Feature*. Proceedings of International Conference on Technology and Social Science 2019 (ICTSS 2019)

[8] `https://developer.apple.com/documentation/arkit/arfacegeometry`

[9] Stiefelhagen. *Estimating Head Pose with Neural Networks*. Pointing '04 ICPR workshop (satellite workshop to International Conference on Pattern Recognition), Cambridge, UK, August 2004

[10] N. Gourier, D. Hall, J. L. Crowley. *Estimating Face Orientation from Robust Detection of Salient Facial Features*. Proceedings of Pointing 2004, ICPR, International Workshop on Visual Observation of Deictic Gestures, Cambridge, UK.

[11] K. He, et. al. *Deep Residual Learning for Image Recognition*. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770-778.

[12] Simonyan, K., and Zisserman, A. (2015). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. CoRR, abs/1409.1556.

[13] A. Zhang, Z. C. Lipton, M. Li and A. J. Smola (2019). *Dive into Deep Learning*. `http://www.d2l.ai`

[14] N. Inkawhich. `https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html`