Partita: A Gaze-Controlled PDF Page Turner

Anonymous Author(s)
Affiliation
Address
email

Abstract

Turning pages presents a challenge for many musicians, and the currently widely-used solutions are expensive or require another person to assist. We present a gaze-controlled page turner for PDFs, written in Python and built on top of the ProxylessGaze gaze detection model.

1 Background

For musicians, turning the pages of sheet music can present difficulties. Often music is not written such that there is a convenient pause where the page is meant to be turned, so musicians have come up with various solutions.

Some have other people turn their pages for them, but this is impractical on many occasions for obvious reasons. If the designated page turner is not familiar with the music being played, they might turn at the wrong time; and there is always the possibility of human error, accidentally turning two pages at once, etc.

A somewhat-widely adopted modern technological solution is to display sheet music on a tablet and turn pages via a Bluetooth-controlled foot pedal. This is expensive (prices currently range from $20 to $100 for commericallly available foot pedal page turners), and necessitates carrying around additional hardware, which is already a burden for many equipment-laden musicians who travel regularly to perform. Furthermore, some musicians such as drummers or pianists use their feet to play their instruments, rendering the foot pedal inviable or at difficult to adjust to as a performer.

Finally, some software solutions auto-scroll through the page, making assumptions (or accepting user input) about how fast the musician will play. A more nuanced form of this would be to use audio input to match up what is currently being played with the notes in the sheet music, and adjust the speed accordingly; but this becomes difficult if there is any improvisation, or other deviation from the exact notes written on the screen.

Musicians crave a solution which is inexpensive and seamlessly fits into established practice and performance habits.

We now briefly outline the technologies that we built our solution on top of:

1.1 ProxylessNAS

ProxylessNAS, Cai et al. [2019], is an efficient and precise form of NAS (Neural Architecture Search), normally an extremely computationally expensive process to optimize the architecture of a neural network.

The efficiency gains were achieved by training one large overparameterized network rather than many normal-sized ones, and performance was improved by training directly on the target task, rather than an easier or cheaper "proxy" task. Further gains were realized by training using hardware metrics like

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

latency directly as optimization targets, yielding a model that is optimized for the given task on the given hardware, a new result.

1.2 ProxylessGaze

ProxylessGaze is a gaze-detection model where the architecture backbone is chosen via ProxylessNAS. The resulting model is quite small, and runs quickly even on a consumer laptop not meant for intense graphics processing (Lenovo Thinkpad from 2020 runs the demo at 60 frames per second).

We chose to use this model to perform gaze detection for us, and believe it to be a huge step up from the capabilities of previous works, as outlined below.

2 Related Work

There have been multiple other attempts at creating a gaze-controlled PDF page turner.

Our work is similar to the proposed work of Chen et al., who wanted to designate "hot corners" of the screen at which users would stare for a specified period of time. They planned to train their own eye-tracking model, which we can't compare to ProxylessGaze as we were unable to find the results of their work online.

Tabone et al. [2020], instead of turning pages using discrete gestures, continuously track the musician's progress throughout the music, assuming that they will read the sheet music from left to right, top to bottom. This approach assumes that the musician is performing their music, start to end, without repeats or deviations from what is written on the page. So, for example, if a musician is simply practicing the music on a single page, the program would erroneously turn the page once they got to the end, when the musician only wanted to go back to the top of the same page to practice it again. Furthermore, their solution, while quite robust and accurate requires an infrared eye tracking device hooked up to a workstation, rather than the simple tablet that most actual musicians use.

Nixon also created a gaze-controlled page turner, additionally implementing a continuous "scrolling" mode rather than discrete page turning Nixon [2022]. The solution was met with mixed reviews from users, with the main problem being erroneous page turns: the rolling average of gaze position would sometimes inadvertently meet the corner of the screen even though the user was not intending to turn the page. This could happen if, for example, a pianist glanced at their right hand, overshooting the corner in reality but causing the slow-reacting gaze detection system to think they meant to turn the page. the author also complained of low accuracy in the gaze detection mechanism itself due to a lack of person-specific calibration.

3 Our Work

We aimed to address most of the issues from previous works above: our solution would ideally be deployed on a tablet with high accuracy and low memory footprint, and with robust gesture detection for turning pages.

PDF Reader

We implemented a simple Python PDF reader, since we could not find an existing one to suit our purposes.

3.1 Gaze Gestures for Page Turning

On top of the PDF reader, we run ProxylessGaze to detect facial landmarks and approximate gaze direction. We designate the top right and top left corners to be gaze targets for turning the the next and previous pages, respectively. During calibration, the user looks at each corner in turn, and the program records their gaze position for each target.

After calibration, the program designates a tolerance around the calibrated target; if a user's gaze falls within the tolerance for 0.5 seconds, we register a gaze gesture and turn the page appropriately.

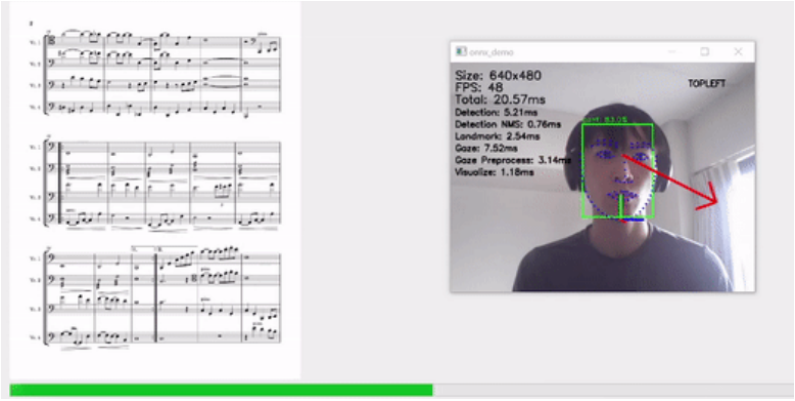


Figure 1: An example of our demo, with the bar at bottom indicating progress towards gesture detection.

We found this to be quite effective out of the box, but it needs some work to become more robust, especially to the angle of the user's face, which can very well change in the course of playing.

3.2 Quantization

The Proxyless Gaze model is already quite small, with the onnx models for face, gaze, and landmark detection totaling around 15 MB in size, but we wanted to push the boundaries of quantization such that size and inference speeds could be improved. Therefore, we attempted to apply two quantization methods, kmeans and linear quantization, specifically on the gaze-detection model. All fine-tuning was done on the MPIIFaceGaze dataset, a dataset of the faces of 15 participants across varying angles Zhang et al. [2019]. Eye features were extracted as a preprocessing step before training Cai et al. [2019].

3.2.1 Kmeans Quantization

K-means quantization is based on the idea of weight sharing: by reducing the number of bits required to represent each weight, we limit the number weights required by having multiple neurons share the same weight Han et al. [2016].

Weights are first quantized into N clusters, where weights in each bin share the same value. Therefore, we only need to store a table of bins and values for each original weight in the neural network. After each training epoch, the table is updated to reflect the gradient updates.

The bins are typically calculated using k-means clustering. In our experiments, we attempted 8-bit quantization using k-means clustering and trained over 200 epochs with a learning rate of 0.005. Our training results are shown in Figure 2, where fine-tuning on k-means led to the model being unable to converge properly on the face gaze dataset. We attribute this to the already small nature of the model found through NAS, therefore reducing the number of bits to represent the weights would most likely decrease accuracy instead of maintaining it.

Indeed, in live testing with the exported ONNX model, we found that the gaze detector was unable to accurately predict the gaze vector, failing to calibrate between different corners of the viewport.

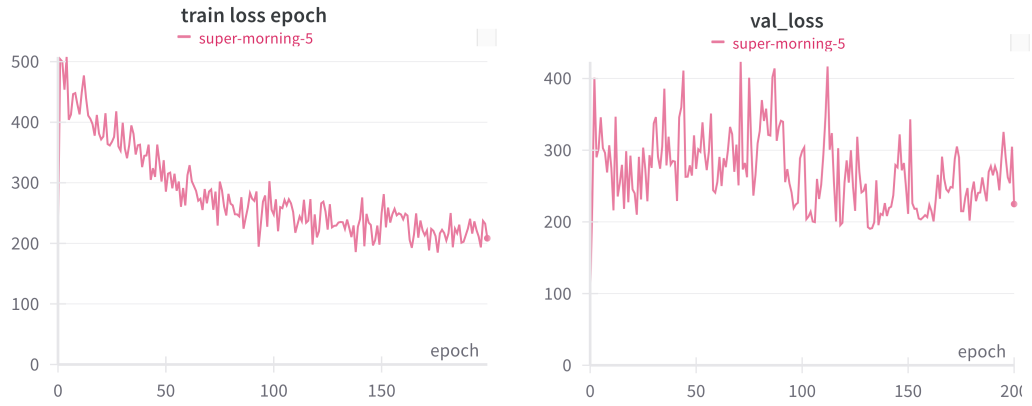
3.2.2 Linear Quantization

Linear Quantization essentially applies an affine transformation of quantized integers to real numbers of the form:

$$r = S(q - Z) \tag{1}$$

, for some constants S and Z, the quantization parameters Jacob et al. [2017]. q is the quantized weight in this case. S, or scale factor, is calculated by dividing the range of a tensor by the quantized bit range of the target quantization value:

$$S = \frac{r_{max} - r_{min}}{q_{max} - q_{min}} \tag{2}$$



(a) Training loss on MPIIFaceGaze over 200 epochs on 8-bit k-means quantization

(b) Evaluation loss on MPIIFaceGaze over 200 epochs on 8-bit k-means quantization

Figure 2: Training and Evaluation loss graphs show that k-means 8-bit quantization fails to converge properly for this specific task.

```
(21): MobileInvertedResidualBlock(
  (mobile_inverted_conv): MBInvertedConvLayer(
    (inverted_bottleneck): Sequential(
      (conv): Conv2d(64, 384, kernel_size=(1, 1), stride=(1, 1))
      (act): ReLU6(inplace=True)
    )
    (depth_conv): Sequential(
      (conv): Conv2d(384, 384, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3), groups=384)
      (act): ReLU6(inplace=True)
    )
    (point_linear): Sequential(
      (conv): Conv2d(384, 96, kernel_size=(1, 1), stride=(1, 1))
    )
  )
)
```

(a) `MobileInvertedResidualBlock` structure before linear quantization. Note how the BatchNorm layer has been fused with the convolutional layer.

```
(21): MobileInvertedResidualBlock(
  (mobile_inverted_conv): MBInvertedConvLayer(
    (inverted_bottleneck): QuantizedConv2d()
    (depth_conv): QuantizedConv2d()
    (point_linear): QuantizedConv2d()
  )
)
```

(b) `MobileInvertedResidualBlock` structure after linear quantization, where each Conv2D has been replaced by a `QuantizedConv2D` layer.

Figure 3: Before and after linear quantization of a `MobileInvertedResidualBlock`

, where r is the tensor to be quantized. Z is the zero-point, and represents the quantized value q corresponding to the real value 0. Z is calculated using the following:

$$Z = round\left(q_{min} - \frac{r_{min}}{S}\right) \tag{3}$$

Quantization of ProxylessNASNet As per Jacob et al. [2017], we also fuse the BatchNorm paper into the convolutional layer, reducing any extra multiplication during inference. Taking the `MyModelv7` module in the proxyless repository, we took each `ProxylessNASNets` backbone, mainly the `eye_channel`, `attention_branch`, and `face_channel`, and modify each of the `MobileInvertedResidualBlocks` by replacing each convolutional layer with a new `QuantizedConv2D` with the ReLU layer combined with the forward pass. Linear layers for the `leye_fc`, `reye_fc`, `leye_attention_fc`, and `reye_attention_fc` modules were then converted into `QuantizedLinear` modules. Quantization calculation code for scale and zero-point was taken from 6.5940 Lab 2, compiled into a wrapper for ease of use.

Results Unfortunately, we ran into problems deploying the quantized model to the PDF gaze viewer. ONNX natively does not support int8 nor int16 types when exporting. We attempted to get around this by simply running the quantized model directly as inference. This did reduce FPS speeds significantly. Even with the reduced speeds, however, accuracy was also significantly reduced, with the model being unable to differentiate between different sides of the viewport. As a continuation of this project, a more widespread sweep of quantizing individual layers instead of the entire model may improve accuracy.

3.3 Deployment on Tablet

We were unable to complete tablet deployment in time, due to the inflexibility of mobile app development. Model inference is helpfully within reach since there are iOS frameworks for Pytorch and ONNX, but all processing of the output, detection of face landmarks, and gaze/gesture detection would have to be rewritten in Objective C/Swift for iOS, or C++ for Android.

Although we didn't successfully finish deployment on tablet, we would be excited to continue to pursue this route in the future.

4 Conclusion

We implemented a gaze-controlled PDF page turner, using the ProxylessGaze project for gaze detection, and would like to deploy to tablets in the future. Our code (with demo) is open-source and available at `https://github.com/lithafnium/6.5940-project`.

References

H. Cai, L. Zhu, and S. Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019. URL `https://arxiv.org/pdf/1812.00332.pdf`.

Y. Chen, Z. Xia, and A. Zhumabekova. Real-time gazed-controlled digital page turning system. URL `https://stanford.edu/class/ee367/Winter2019/chen_xia_zhumabekova_ee367_win19_proposal.pdf`.

S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2016.

B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference, 2017.

C. Nixon. *EyeGaze: Automated sheet music page-turning using gaze detection.* PhD thesis, 2022.

A. Tabone, A. Bonnici, and S. Cristina. Automated page turner for musicians. *Frontiers in Artificial Intelligence*, 3, 2020. ISSN 2624-8212. doi: 10.3389/frai.2020.00057. URL `https://www.frontiersin.org/articles/10.3389/frai.2020.00057`.

X. Zhang, Y. Sugano, M. Fritz, and A. Bulling. Mpiigaze: Real-world dataset and deep appearance-based gaze estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 41(1):162–175, 2019. doi: 10.1109/TPAMI.2017.2778103.