Detecting Student Engagement through live Video Stream

Chris Sexton

12/2/2020

Abstract

Detecting student engagement from video is a difficult task. This paper describes in detail the methods and processes to train deep learning models to predict engagement with CNNs and RNNs using cloud infrastructure, and the practical deployment of these models to an NVIDIA Jetson device. Although the models tend to overfit to training data, the overall system is designed in such a way that as alternative models are developed they can be integrated. Some attempts are made for code reuse, usability and the centralized storage of data and results.

Introduction

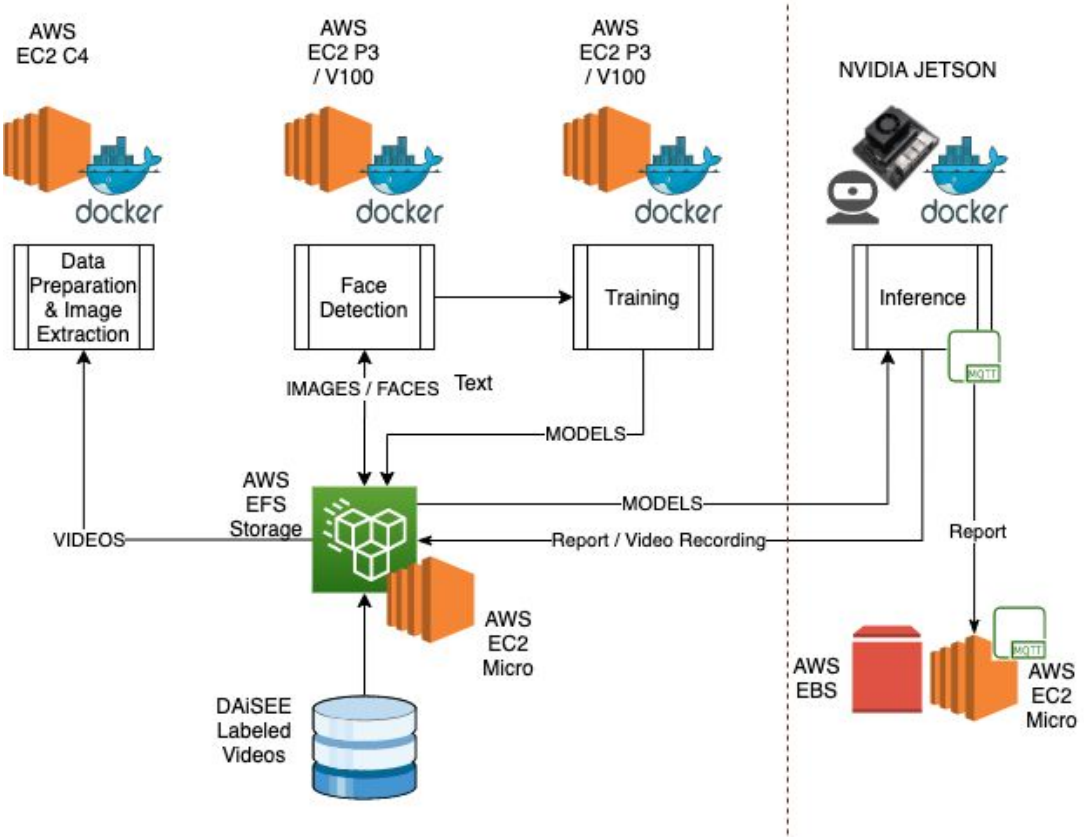
More and more students are engaged in online learning, either as a necessity due to the recent COVID-19 pandemic or voluntarily through remote learning courses. For some students, the shift from in classroom attendance to online attendance can be difficult, and communicating over video less immersive. Teachers also have to contend with ensuring students are engaged in the learning process, which can be difficult when the primary access to student response is through a small video representation in an online meeting window.

This work uses video capture to identify when students become bored and provide real time and offline feedback to the teacher. This is a proof of concept, examining the possibility of creating models to classify engagement from video and then applying these models to low powered edge devices.

High Level Solution

The solution combines cloud and edge technology to train deep learning neural net models from previously labeled videos. Individual frames are captured from the videos and utilised in various models, trained to predict boredom levels from 0 (not bored) to 3 (most bored). Models are trained against full images and also images of just the students faces, to reduce the influence of background data. Images are captured from webcam video streams.

Technical Architecture



The project is split into two sections, the data preparation and model training shown on the left of the dotted line, and then the live video capture and inference on the Jetson device on the right.. Docker is used extensively throughout.

Training

Purpose	Type	GPU	vCPUs	Mem (GB)
Image Extraction	c4.8xlarge	None	36	60
Face Detection	p3.2xlarge	1 * NVIDIA Tesla V100 (16GB)	8	61
Storage	EFS			

AWS is used for cloud infrastructure to support training. EFS is used as the back end storage to

provide persistence between environments for data and saved models. Docker is used to containerize the code, using the `tensorflow:latest-gpu-jupyter` base image. Tensorflow v2 is used for the deep learning platform, along with Keras and openCV for image processing.

Data Preparation and image extraction is performed on a compute optimized c4.2xlarge. FFMPEG is used to extract images from training video streams. Two extractions were used, 1FPS for CNN based models and 2FPS for RNN based models.

Face detection is performed using a pre-trained DNN model, with weights trained using caffe. DNN is available with OpenCV.

The model training scripts are designed to be flexible and modular, and can run with a variety of model inputs:

- Model: e.g. XceptionV3, MobileNetv2, EfficientNet, Inception
- Image Size, e.g. 224, 299
- Data Type: e.g whole images, faces only, augmented data
- Weights, e.g. imagenet
- epochs
- batch_size
- Learning rate
- Learning rate decay
- Inference

Inference

Purpose	Type	GPU	vCPUs	Mem (GB)
Inference	NVIDIA Jetson Xavier	512-Core Volta	8-Core ARM v8.2 64-Bit	32

Inference is performed on an NVIDIA Jetson with an attached webcam. Model files are copied to the Jetson with SCP..

The Jetson uses ARM computing architecture and so some compilation of code is required for inference to work, specifically newer versions of OpenCV (>3.2). For inference the docker base image `nvcr.io/nvidia/l4t-ml:r32.4.3-py3` is used. Tenroflow, Keras and OpenCV are installed.

Inference is done in real time against a video stream, with individual and running totals of engagement presented on screen. At the end of the meeting a report is saved and optionally sent to AWS S3 using MQTT for offline analysis. The client demo scripts provide some options when

run:

-r --record: whether to record the video
-m --messaging, whether to send report to AWS using MQTT
-p --path, directory to record the video, default is home
-f --filename, name of the video recording file, default is infer<datetime>.avi
-c --codec, video codec, default is MJPG
-fps --fps, frames per second, default is 2
-hg --height, height of video to record default is 480
-w --width, width of video to record default is 640

Data Acquisition and Processing

DAiSEE Data for Affective States in e-Environments is used to provide ground truth for the classification models, publicly available on request. It contains 8925 multi label 10 second video snippets captured from 112 users. Each snippet has been scored from 0 to 3 for boredom, confusion, engagement and frustration.

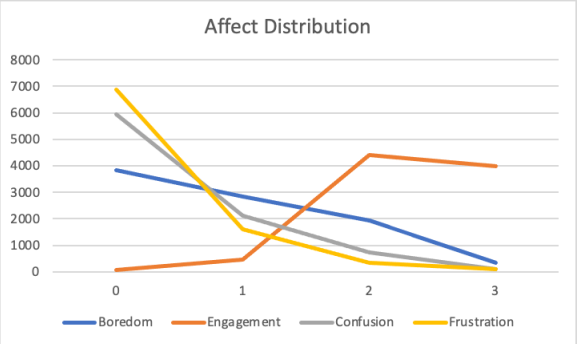
The DAiSEE data has already been organized into train, validation and test datasets. Each video is structured into a deep directory structure, i.e. Purpose -> Person -> Snippet -> Video file. A separate label CSV file contains the file path and label for each 10 second video snippet. All videos associated with one person are contained within one purpose, i.e. there cannot be videos from person A in Train and also in Validation (or Test).

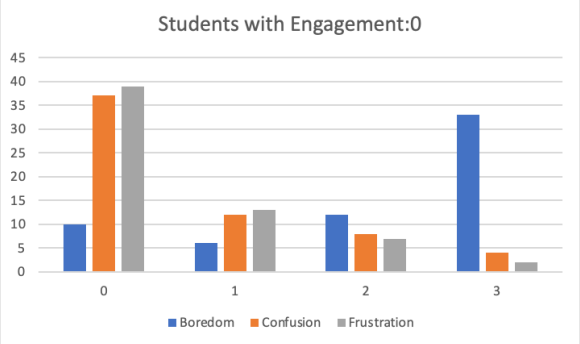
```
|__Test
        |__500044
        |__500067
        |__500095
                |__5000441001
                |__5000441002
                |__5000441003
                        |__5000441003.avi
|__Train …
|__Validation …
```

The distribution of data between the different classes (0, 1, 2, 3) is shown in the following table, for the different engagement types (Boredom, Engagement, Confusion, Frustration).

	Boredom	Engagement	Confusion	Frustration
0	3822	61	5951	6887
1	2850	455	2133	1613
2	1923	4422	741	338
3	330	3987	100	87
total	8925	8925	8925	8925
average	0.86	2.38	0.44	0.29
% labeled 0	43%	1%	67%	77%

The distribution for Boredom is slightly more even than other engagement types, and stands as a good substitute for engagement. When a student is Engagement = 0 they are more likely to be Boredom = 3





Individual images are extracted from the video files and stored in a structured format suitable for boredom classification, depending on the FPS used. Additional storage areas are created for additional image manipulation, i.e. face detection and augmentation:

Directory Structure

data / DAISEE /	Contains all the data based on DAiSEE provided data
… DataSet	The original videos stored as described above
… Labels	CSVs containing labels for videos
… 1FPS … 2FPS	Separate directories for images captured at 1 or 2 Frames per Second

… … data … … … Test, Train, Validation … … … … b0, b1, b2, b3	Files containing whole images extracted at parent FPS, organized by Test/TrainValidation and Boredom Class (b0, b1, b2, b3)
… … dataFaces … … … Test, Train, Validation … … … … b0, b1, b2, b3	Files containing just faces, extracted from data, organized by Test/TrainValidation and Boredom Class
… … dataAug … … … Test, Train, Validation … … … … b0, b1, b2, b3	Files containing augmented images, extracted from data, organized by Test/TrainValidation and Boredom Class
… … dataImages … … … Test, Train, Validation	Files containing whole images extracted at parent FPS, organized by Test/Train/Validation but containing all images for all engagement types (for multi-task modelling)

Augmented Images are created using keras ImageDataGenerator

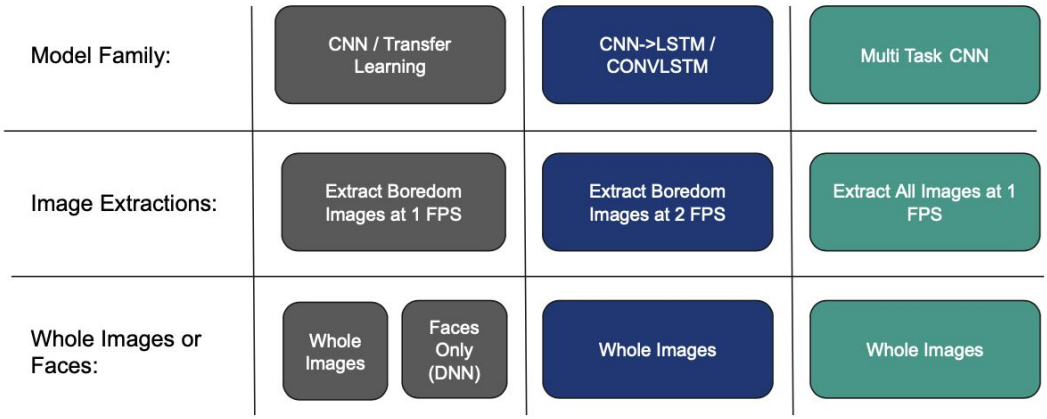
Face Capture

Haarscascade and DNN were tested, with DNN capturing faces from more image snippets than haarscascade. Examining images extracted at 1FPS:

	Original	haarcascade	dnn	dnn difference
Test	17844	17443	17830	387
Validation	14294	14062	14289	227
Train	53584	53352	53566	214
Total	85722	84857	85685	828

Modeling

Over 30 experiments were performed, divided into the following 3 general areas:



The best results from the three methods are given below:

Common parameters:

- Learning Rate: 0.0001
- Learning Rate Decay: 1.00E-06
- Weights: Imagenet
- Optimizer: Adam (apart from Multi Task which uses SparseCategoricalCrossentropy)

Model	data type	FPS	Base Model	frozen layers	epochs	batch size	accuracy	val accuracy	test accuracy
CNN	Augmented Faces	1	MobileNetV2	[:126]	100	32	0.88	0.34	0.36
CONVLSTM	Whole Images	2	ConvLSTM2D	All	100	16	0.96	0.32	0.34
CNN -> LSTM	Whole Images	2	MobileNetV2 -> LSTM	All	100	32	0.88	0.31	0.38
Multi Task (CNN)	Whole Images	1	Xception	All	10	16	0.59, 0.65, 0.70, 0.59	0.39, 0.46, 0.64, 0.39	42.48, 56.01, 66.92, 42.77

Multi Task Results are presented in order for Boredom, Engagement, Confusion, Frustration.

All of the models had a tendency to overfit, this is likely due to the nuanced nature of the problem set. The facial image differences between four levels of boredom is slight and difficult for a mode to distinguish.

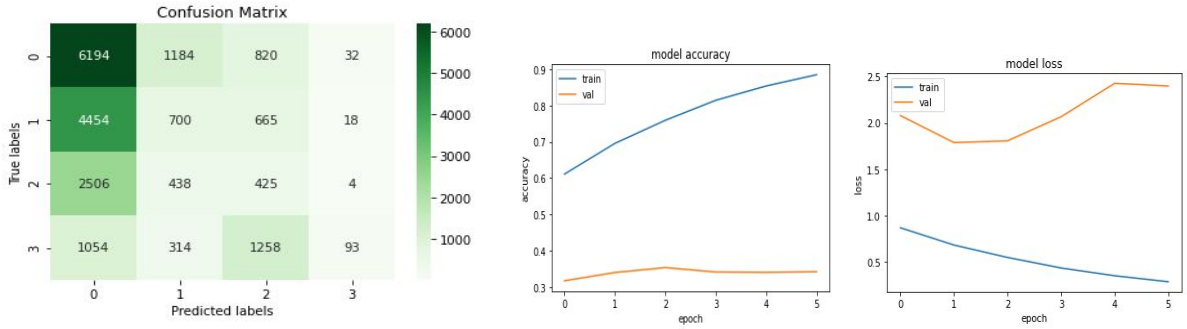
CNN Models

For the models CNN, the classification layer is removed and 2 Fully Connected Dense Layers added with a dropout of 0.2 before the output layer.

```
x = layers.Flatten()(base_model.output)
x = layers.Dense(1024, activation='relu')(x)
x = layers.Dropout(0.2)(x)
x = layers.Dense(512, activation='relu')(x)
x = layers.Dense(4, activation='softmax')(x)
```

The best CNN results all used a frame rate of 1 FPS: the videos do not contain a lot of action and using a higher frame rate does not result in performance improvement. Faces rather than whole images are used for classification input, and the addition of augmented images for the under represented boredom 3 class helped the model to find more instances of boredom 3.

The model is overfitting and over predicting for class B0. This still gives the best results during inference (better than LSTM and multi-task) but more work needs to be done.



Muti task Model

For the multi task model there are four classification heads, one for each class:

```
x = GlobalAveragePooling2D()(base_model.output)
x = Dense(128, activation="relu", name="fc1")(x)
x = Dense(64, activation="relu", name="fc2")(x)
boredom = Dense(4, activation='softmax', name="y1")(x)
engagement = Dense(4, activation='softmax', name="y2")(x)
confusion = Dense(4, activation='softmax', name="y3")(x)
frustration = Dense(4, activation='softmax', name="y4")(x)
```

CNN -> LSTM Model

First features are extracted from the base model (e.g MobileNetV2) at the global_average_pooling2d layer. The features are reshaped for input into the LSTM model. Whole images are used, captured at 2 FPS to provide a sequence of 20 frames.

```
model = Sequential()
model.add(LSTM(units=2048, input_shape=(20,1280),
               return_sequences=False,
               dropout=0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(4, activation='softmax'))
```

CONVLSTM Model

Whole images are used, captured at 2 FPS to provide a sequence of 20 frames.

```
model = Sequential()
model.add(ConvLSTM2D(filters = 16, kernel_size = (3, 3),
return_sequences = False, data_format = "channels_last", input_shape
= (seq_len, img_height, img_width, 3)))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(256, activation="relu"))
model.add(Dropout(0.3))
model.add(Dense(4, activation = "softmax"))
```

Deployment

Models are deployed to the Jetson using SCP. Different python programs are written for each of the model types

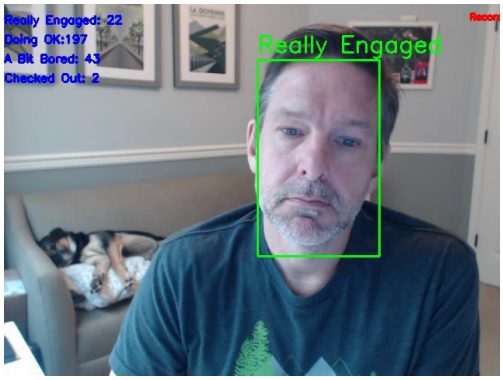
- infer_cnn_dnn.py
- Infer_muti_task.py
- Infer_cnn_lstm.py
- infer_CONVlstm.py

The infer_cnn_dnn is the best demo script and incorporates the following features:

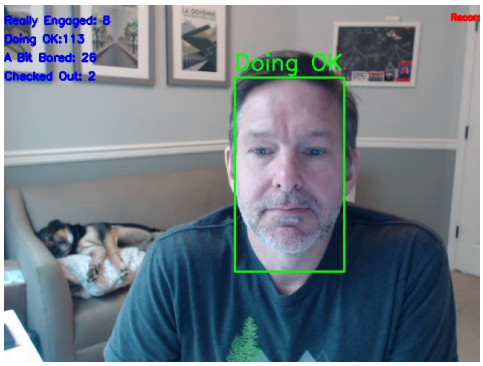
- Options for setting frame rates and codecs for image capture
- Options for recording video
- Options for using MQTT messaging for final report
- Display's running count of each class
- Display's "recording" message if video capture is being recorded
- Saves summary text report of video

Example Results from CNN

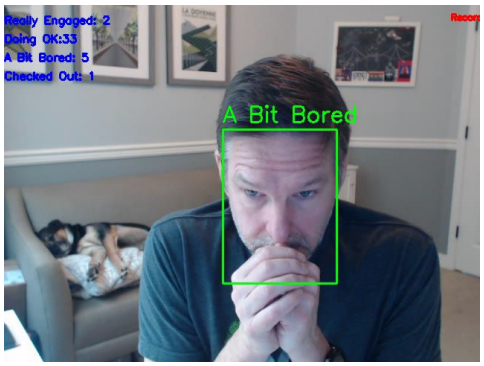
BORED = 0



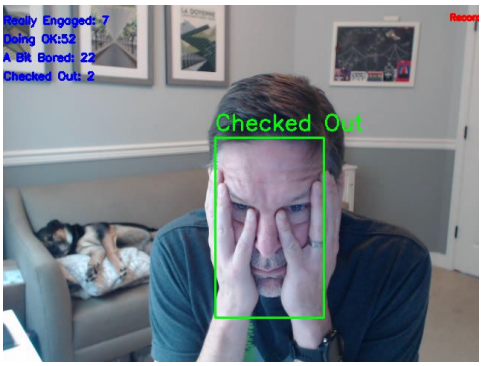
BORED = 1



BORED = 2

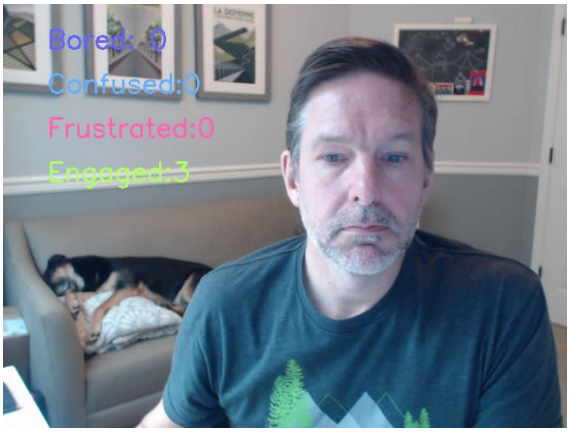


BORED = 3




```
Really Engaged: 27
Doing OK:198
A Bit Bored: 43
Checked Out: 2
```

Example Results from Multi-Class



Challenges

Challenges exist in ensuring that library versions are consistent between AWS X86 architecture and Jetson ARM architecture.

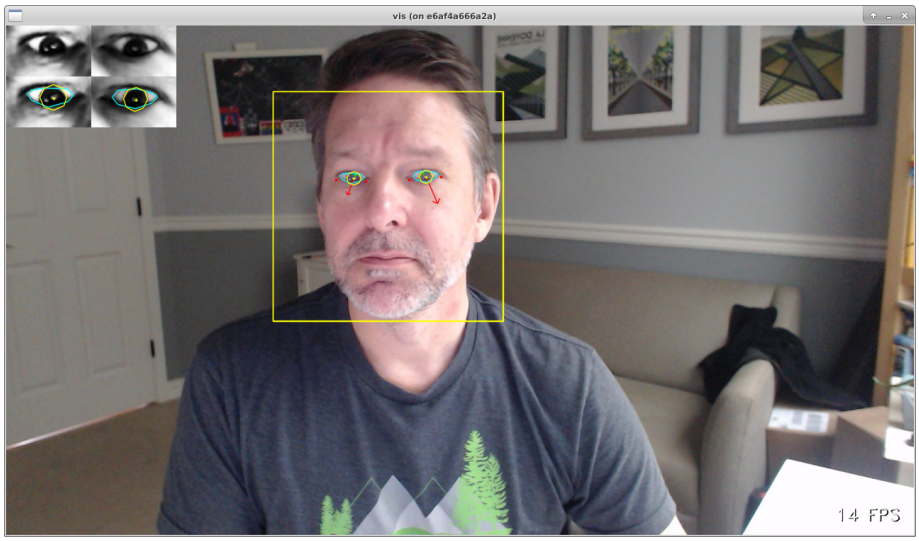
Issue	Mitigation
Overfitting	Add additional meta inputs for modelling, such as gaze direction, head angle. Add more fundamental features, eyes shut, student at desk Consider other operational approaches to problem
No docker image support for DNN	Compile and install OpenCV from source
Model Sizes too big for Jetson	Use MobileNetV2
General Memory Errors	Configure Tensorflow to grab more memory as needed
Unbalanced training set	Augment Images for class three Reduce # of images for class zero
Base models not designed for engagement	Unfreeze later layers for training

Future Improvements

- Package up the code, create modules for reuse
- Feedback Loop for improved models. A system whereby teachers or students express their engagement during classes through a simple interface.
- Incorporate Gaze detection into the training process, if students are looking away from the screen for too long, identify as losing attention.
- Identify the difference between working (looking down) and disengagement (looking away, unfocussed gaze
- Report to contain timing of boredom to align with duration of class for correlation
- Consider ethics of application, do not allow recording of video, summarize classroom
- Capture video from multiple students in online recording (e.g. from Zoom or MS Teams)

Addendum - Gaze ML

Using gaze detection may be a useful input to the modelling process. Seonwook Park et al have provided code to predict eye gaze detection from video, which has been ported to the Jetson:



https://github.com/swook/GazeML

References

[1] Automatic Recognition of Student Engagement using Deep Learning and Facial Expression, Omid Mohamad Nezami,, Mark Dras, Len Hamey, Deborah Richards, Stephen Wan, and Ce ´cile Paris, 2018, https://arxiv.org/abs/1808.02324

[2] Prediction and Localization of Student Engagement in the Wild, Amanjot kaur, Aamir Mustafa, Love Mehta, Abhinav Dhall, 2018, https://arxiv.org/abs/1804.00858

[3] DAiSEE: Towards User Engagement Recognition in the Wild, Abhay Gupta, Arjun D'Cunha, Kamal Awasthi, Vineeth Balasubramanian, 2016, https://arxiv.org/abs/1609.01885

[4] Gaze360: Physically Unconstrained Gaze Estimation in the Wild, Petr Kellnhofer, Adria` Recasens, Simon Stent2, Wojciech Matusik, and Antonio Torralb, http://gaze360.csail.mit.edu/iccv2019_gaze360.pdf

[5] Learning to Find Eye Region Landmarks for Remote Gaze Estimation in Unconstrained Settings, Seonwook Park, Xucong Zhang, Andreas Bulling, Otmar Hilliges, 2018, https://ait.ethz.ch/projects/2018/landmarks-gaze/