What was the problem?

- Visual speech data
  - Close perspective
  - Using a webcam
- Use deep learning
  - TensorFlow
- Classify the current mouth position (i.e. viseme)
- The predictions should be suitable for recreating the mouth positions using a mesh
- i.e. How can deep learning technology be leveraged to detect the mouth positions?

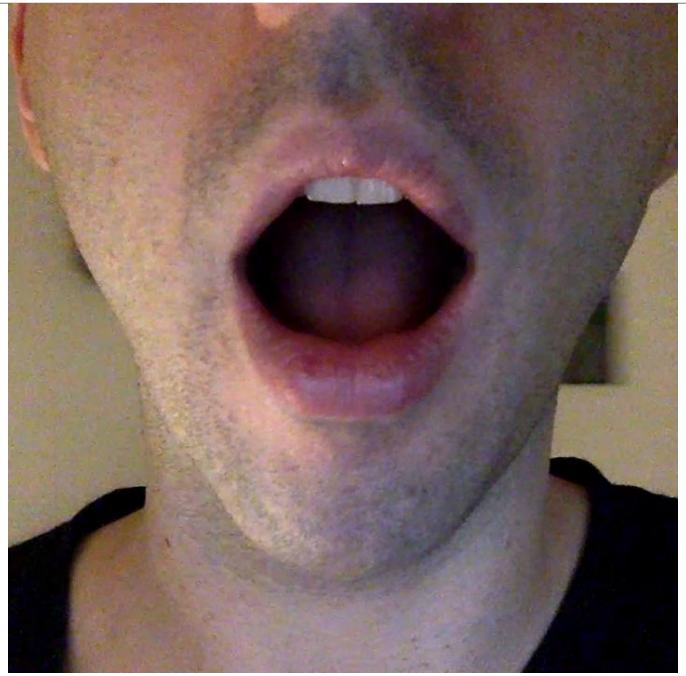
What is my solution?

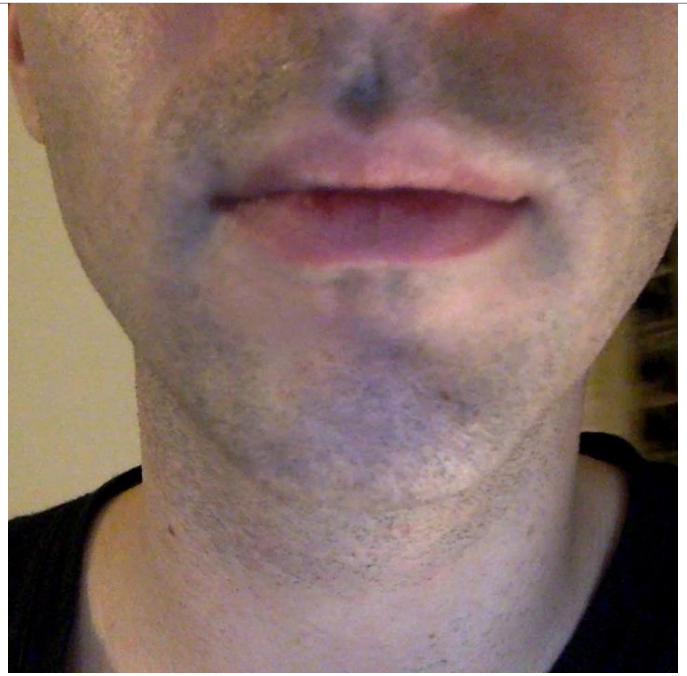
- Created a dataset of labeled images for each class
- Trained a neural network using the images and exported the frozen model
- Created a Go program to feed each frame of the webcam stream to the model and output the prediction

Image Data

- Roughly 4000 images
- Different individuals
- Some individual screenshots, some extracted from video clips using ffmpeg
  - ffmpeg -i [location of source video] -filter:v “crop=720:720:180:0” [location of saved images with naming template]
    - naming template example: “/somelocation/ope.%05d.png”
      - save each frame to “somelocation” using the prefix and 5 decimal padding with automatic incrementation
- Each image filename contains the label
  - e.g. ope.12345.png
  - e.g. clo.12345.png

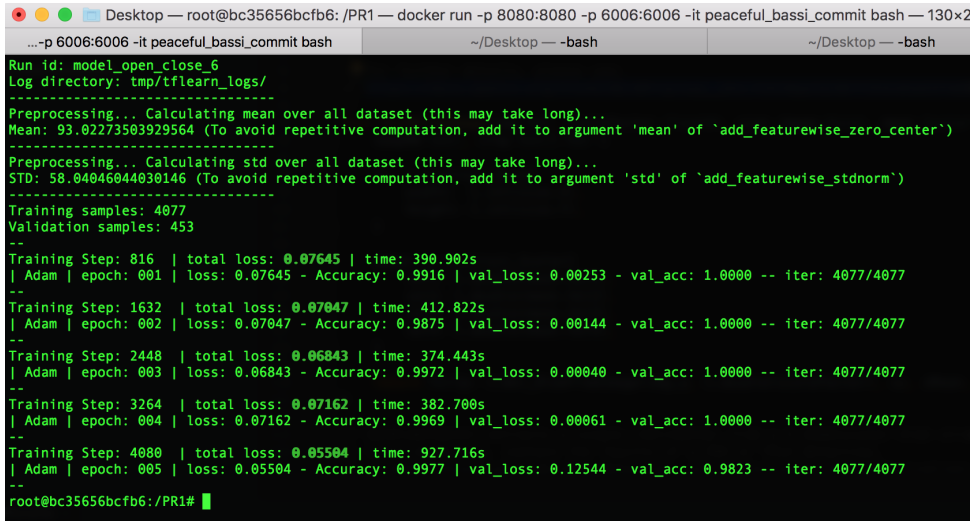
Open and Closed Classes





NN Details

- Input image size: 64x64
- Output activation function: Softmax
  - squishes output between 0 and 1
- Optimizer: Adam
- Loss function: Categorical Crossentropy
  - increases as prediction further from actual value
- Training epochs: 5
- Learning rate: 0.0005
- Batch size: 5
- Data augmentation



- Input -> Conv2D -> Pooling -> Conv2D -> Conv2D -> Pooling -> Fully Connected -> Fully Connected

How good is the result?

- 2 classes/visemes covered, Open and Closed
- Sensitive to lighting and background
- Sensitive to positioning
- Functioning but not very robustly

What could be improved

- Make it more robust against lighting and positioning
- Add more classes (e.g. 4 visemes in total)
- Would need smoothing when used in another context (to prevent repeated jumping between classes on false predictions)
- Link it to a mesh model in Unity

What I Used

- Components involved:
  - nn.py creates and trains the TF model
  - main.go processes each
- Technologies involved:
  - TensorFlow, Tflearn
  - Docker for TensorFlow environment deployment
  - GOCV for openCV bindings in Golang
  - Freeze script from https://github.com/tflearn/tflearn/issues/964

A Few Problems I faced

- Exporting the tflearn model as .pb
- Reading the model from gocv
  - openCV does not support reading models containing Dropout layers
- Needed to commit and restart container in order to use Tensorboard
  - Not sure why the only way to bind a port to a container is at runtime…