CS230

Read My Lips

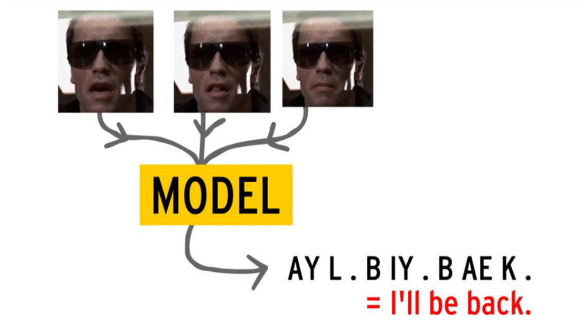
Cary K. Huang and James Y. WoMa
carykh@stanford.edu
jaywoma@stanford.edu
https://github.com/carykh/videoToVoice

Abstract

Reading lips (i.e., extracting phonemes from lip visuals) can be difficult, especially in noisy videos. To solve this problem, we created a deep-learning algorithm to read lips. We chose to use a convoluational neural network on the video frames themselves, due to the success of CNNs as image classifiers in the past. In the end, we created a model that can identify the correct phoneme spoken 48% of the time by looking at only images, which is close to peak human performance.

1 Introduction

There are countless examples of videos where a person's lips are visible, but the sound of their voice can't be picked up. This ranges from people conversing at a loud party, shouting spectators at a football game, or gossipers sharing secrets under their breath. In these cases, an algorithm that analyzes the visuals and outputs the most-likely spoken transcript is incredibly useful. A real-time implementation of this algorithm could help people converse across long distances, or help the deaf navigate through life more easily.



Our neural network converts image sequences of people speaking into the phonemes they said. Here are the steps from start to finish:

1. Split a video into an image sequence using the Python library videosequence [6]. (30 frames per second in our example)
2. Use the Python library face_recognition [2] to crop each image around the speaker's lips.
3. Extract audio (.wav format) from the input video using ffmpeg [double check this].
4. Use the Python library Gentle [4] to convert the audio into a text file that lists what phoneme is spoken per frame.
5. Use our trained model to input a timestamp in the video, and get out the predicted phoneme being spoken.

5a. When we input a timestamp, our algorithm takes the 29 consecutive frames (from 14 frames in the past to 14 frames in the future).

5b. These 29 frames each have 3 color channels and all fit within 250x250 pixels, so we can turn them into a 250x250x87-shaped vector. (See Dataset and Features)

5c. Feed this vector into a convolutional neural network (described in detail below). This will output a 41-element softmax layer, with each element corresponding to the predicted spoken phoneme (41 phonemes types). The phoneme with the highest output is the model's prediction for that frame.

2 Related work

In an article written by Noda et al, they also use a Convolutional Neural Network (CNN). [7] Their dataset includes 6 different speakers speaking, with the appropriate phoneme labels corresponding to each frame of the video, and attained a result of 58% accuracy. One key difference between our dataset and their dataset was that we used English words instead of Japanese words, and had a total of 41 phonemes instead of 40 phonemes (since we also would detect silence). Another key difference was the architecture of the model.

Our model resembled that of Manish Chablani's autoencoder, which has an encoder and decoder half. [3] First, we attempted to convert image sequences to spectrogram images to generate audio, but that model failed. To create the phoneme classifier, we removed the decoder half and replaced it with a softmax layer. We also changed the sizes of the layers to fit our task accordingly.

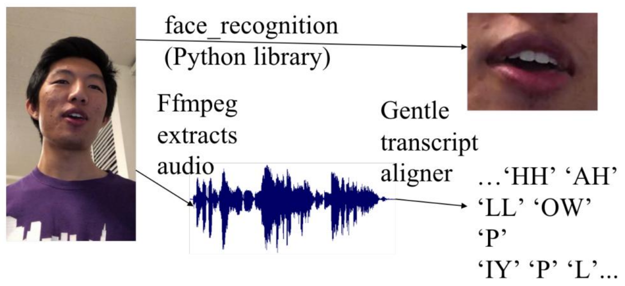
Garg et al proposed using both a CNN and the more conventional LSTM, and compared the results. [8] They had expected the LSTM to be more successful since having language context could improve performance, however they found that the LSTM model trained much slower and performed much worse than their CNN model.

Meanwhile, Neeru Rathee created a traditional feed-forward neural network. [9] Instead of inputting the entire image of a mouth, the input consisted of keypoints that were extracted from the mouth images. The neural network was also trained to identify 1 of 200 different possible words spoken. This type of model would not generalize to larger dictionaries of possible words, so we did not choose this model.

Although Garge et al's LSTM failed, Chung et al also used an LSTM model which was able to correctly identify words with a word error rate (WER) of 3.0%. [1] Considering we were creating our own dataset (Chung used recorded BBC programs) and little computational power, we did not follow the same strategy as Chung. In the future, we would also like to incorporate an LSTM model, but for the purposes of this project, we did not think we had enough time to implement the model successfully.

3 Dataset and Features

Our dataset includes Cary reading the Bee Movie Script ($\approx$ 70 minutes) into an iPhone X camera.

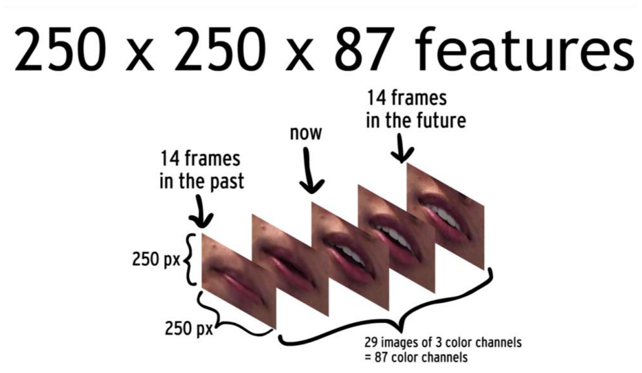


3.1 Getting the Inputs (Images)

We separated the video into 131,000 color images, cropped them around his mouth using a face recognizer [1], and then grouped the images into "frame neighborhoods". A frame neighborhood consists of one frame that the user wants to find the phoneme for (the "central" frame), as well as 14 frames from the past, and 14 frames into the future. This gives the model information about the lips' velocities and surrounding phonetic patterns.

Each frame is 250x250 pixels (enough to contain the mouth) and has 3 colors. If the mouth image was too small, it was centered on the 250x250 grid, and the margins were filled with black.

Imagine layering the 29 images on top of each other. Then, at every pixel location, we have 29 sets of 3 color channels, so we essentially have $29 \times 3 = 87$ color channels. That means the feature shape of each element was 250x250x87. (Since our task is image classification, we do not use other features besides the sequences of images.)



3.2 Getting the Ground-truth Labels (Phonemes)

We also converted the audio of 70-minute video into a list of phonemes using the Gentle Transcript Aligner [3]. This gave us the ground-truth for each element: a phoneme out of 41 options.

silence	k	ao	r	d	ih	ng	t	ah	l	n	ow	z	v
ey	iy	sh	dh	eh	w	b	uh	f	ay	s	aa	uw	m
g	ae	aw	hh	y	th	p	oov	er	jh	ch	oy	jz	

Table 1: The 41 possible phonemes, extracted by Gentle [4] using the Carnegie Mellon University Pronouncing Dictionary

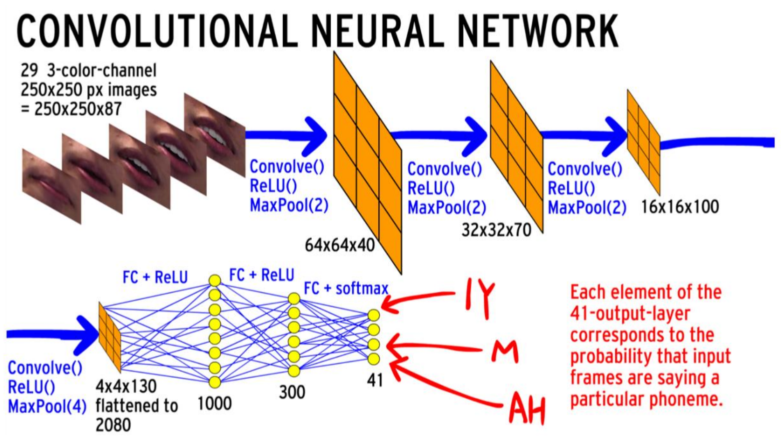
3.3 Creating the datasets

With 131,000 frames, we felt like our dataset was big enough and didn't need any data augmentation. We used the first 120,000 frames for our training set and the last 10,000 frames for our test set.

4 Methods

4.1 Model

Our main model was a convolutional neural network based on Chablani's auto-encoder [2]. Its structure is depicted below:


CONVOLUTIONAL NEURAL NETWORK
29 3-color-channel 250x250 px images = 250x250x87
Convolve() ReLU() MaxPool(2)
64x64x40
Convolve() ReLU() MaxPool(2)
32x32x70
Convolve() ReLU() MaxPool(2)
16x16x100
Convolve() ReLU() MaxPool(4)
4x4x130 flattened to 2080
FC + ReLU
1000
FC + ReLU
300
FC + softmax
41
IY
M
AH
Each element of the 41-output-layer corresponds to the probability that input frames are saying a particular phoneme.

The general idea of these layers is to continually decrease the resolution of the images while increasing the number of filters, so that by the end of the fourth layer, we have images that are low-res enough to flatten into a manageable 1-D layer (in this case, a 4x4x130 vector is flatterned into a 2080 vector). This flattened layer can then be fed through three fully-connected layers to get to the softmax layer, which will output probabilities each corresponding phonemes (by the nature of the softmax, these will add to 1).
The formulas we used are pretty standard:

$$\text{ReLU}(x) = \max(0, x) \qquad\qquad \text{softmax}(x) = \frac{e^x}{\sum(e^x)}$$

4.2 Loss function

Since our project is an image classification problem, the loss function is pretty standard: we used cross-entropy:

$$H(p, q) = \sum_x p(x) \log(q(x))$$

The built-in TensorFlow function

tf.losses.sparse_softmax_cross_entropy(labels=labels_, logits=logits_)

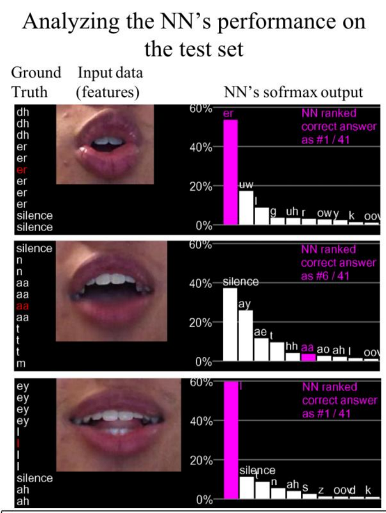
performs this equation, where logits_ are the values of the final layer before being pushed through the softmax. This function is the lowest when the model's prediction is 1 with the correct output and 0 with all the others. This means the model is incentivized to output high probabilities for correct phonemes.

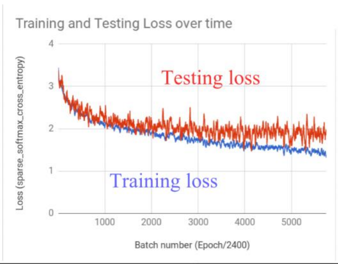
5 Experiments/Results/Discussion

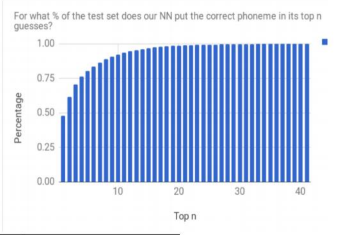
Because each input element was pretty large (250x250x87 = 5.4 million values), our batch size could not be too big before running out of memory on our GPU.

Our training dataset had 120,000 images, but we only trained on 50 per mini-batch. (100 made it crash.) This may seem too low, which would make it train too specifically. However, if we made our learning rate lower as well, it would train slower, then the two effects would hopefully cancel out.

So, after starting with 1e-3, we lowered our learning rate to a constant 2e-4. This worked pretty well, and gave us results that can be seen in this video (`https://www.youtube.com/watch?v=e6JXYQFad_o`) and below:







	0.01 epochs	3.00 epochs
Training loss (first 120,000 frames, approx 67 min.)	3.4416	1.3337
Testing loss (first 11,000 frames, approx 6 min.)	3.3763	1.8987

As you can see, training and testing loss both went down, with testing loss being higher, as expected. However, with a cross-entropy loss function, it's hard for a human to quantify how well a loss of "1.89" really is. As a result, we ran an accuracy test on the training set to see what percentage of predictions were correct or close to correct.

According to the graph above, our final NN guessed the correct phoneme as its top choice 48% of the time, within its top 3 choices with 70.5% of the time, and within its top 5 choices with 80.5% of the time.

Although an accuracy rate of 48% does not seem effective, this accuracy is high considering the NN chose from 41 different phonemes. Additionally, many phonemes look indistinguishable ('f' & 'v'). Our accuracy is also high compared to human level lip-reading accuracy, which have ranged from 12.4% to 52.3%.

In addition, we were surprised by our NN's performance in detecting silence. The NN even detected silence while Cary was inhaling with his mouth open. The NN also could distinguish between silence and 'm' in many cases.

We believe using a 29 frame neighborhood (from 14 frames before to 14 after) allowed the NN to be so effective. The mouth distorts differently depending on the sound produced, and the NN was likely able to use that information in its predictions.

6 Conclusion/Future Work

In the end, we were pleased with our model's performance, given how difficult lip-reading is for humans. Although using CNNs to do image classification has been repeated to death, using multiple images across a time window is quite a bit rarer, so we weren't sure if it would work. Add on to that the fact that many phonemes look like (McGurk effect), and suddenly, 48% accuracy seems pretty effective!

There are several ways we could expand this project. First off, having a mini-batch size of 50 when the training set if 120,000 is embarrassingly small, so we would like to train on larger GPUs to increase mini-batch size. Also, as of now, we only trained on videos of Cary's mouth. As an extension, we could train on other mouths so the NN generalizes. A final extension includes turning the NN's outputted phonemes into actual sound, so it can generate audio instead of a string of phonemes. This would be the ultimate end-to-end project, which could potentially add voice audio to silent videos.

7 Contributions

James found sources, references, and related work; coordinated meetings with TAs at office hours; fine-tuned the hyperparameters, and created the outlines/templates for the project milestone, poster, and paper.

Cary created the dataset (reciting a movie script), downloaded and processed the data, implemented the models in TensorFlow, tested them, and re-wrote them when necessary.

References

[1] Chung, J. S. & Senior, A. & Vinyals, O. & Zisserman, A. (2016) *Lip Reading Sentences in the Wild*, 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)

[2] Geitgey, Adam (2018) *Face Recognition*. GitHub

[3] M. Chablani (2017) *Autoencoders Introduction and Implementation in TF*. Medium

[4] lowerquality (2017) *Gentle*. GitHub

[5] Rouzic, Michael (2008) *The ARSS*. SourceForge

[6] Wareham, Rich (2016) *videosequence 1.1.0*. Python Software Foundation (US)

[7] Noda, Kuniaki & Yamaguchi, Yuki & Nakadai, Kazuhiro & Okuno, Hiroshi & Ogata, Tetsuya (2014) *Lipreading using Convolutional Neural Network*, 2014 INTERSPEECH

[8] Garg, Amit & Noyola, Jonathan & Bagadia, Sameep (2015) *Lip reading using CNN and LSTM* CS231 Stanford University

[9] Rathee, Neeru (2016) *A novel approach for lip Reading based on neural network* 2016 International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT)