Lip Reading
ECE 228 Final Project Track #1

Group #22: **Zachary, Chao** zachao@ucsd.edu **Vignesh, Jayananth** vjayanan@ucsd.edu

Brandon, Liu bsliu@ucsd.edu **Achyut, Pillai** apillai@ucsd.edu

Github Repository
I certify that I [have] filled the evaluation.

Abstract

In this paper, we experiment with a variety of models including 2D CNNs, 3D CNNs, LSTMs, and TCNs to predict words spoken from video clips using solely visual components. We are using a combination of these spatial and temporal architectures as well as the pre-trained networks, ResNet-18 and VGGNet-16, to make our models. These models have proven to perform well on video classification tasks because of their ability to work with time sequences and make long term connections. In our experiments, we combine these models to balance the performance and complexity. From our findings, the VGGNet16 + LSTM model performed the best with a test accuracy of 78%.

1 Introduction

So, for our task, we decided to tackle the problem of lip reading. Lip reading is a difficult task involving taking as an input, a short video clip that lacks audio, and using this to either classify individual words that are being said, or entire sentences. The applications for lip reading range from uses in subtitle generation, to uses in the hearing-impaired industry, to uses in security and surveillance. In this project, we use the Lip Reading in the Wild (LRW) dataset Chung and Zisserman [2017]. This is a dataset with 500 different words, each word with up to 1000 clips for training, 50 clips for validation, and 50 clips for testing. Each clip is a roughly 29 frame, 256x256 mp4 file. The clips were extracted from BBC broadcasts. In this report we detail our attempts to implement and improve upon existing lip reading literature.

2 Related work

Since the release of the LRW dataset in 2016, along with the corresponding model architectures detailed in the paper, many new architectures and papers have come out that tremendously improve upon the LRW performance. This initial LRW paper introduced the dataset and their best model, a multi-tower 2D convolutional net with a backend similar to VGG-M, ultimately achieving a result of 61.1% Top-1 accuracy. Stafylakis et al. combined Residual Nets with LSTMS to achieve 83.0% Top-1 accuracy Stafylakis and Tzimiropoulos [2017]. Then, in 2018, Bai et al introduced Temporal Convolutional Networks as a potential architecture for sequence learning Bai et al. [2018]. They showed that TCN's were able to encode the causal connections that more sequence learning tasks require. Ma et al turned these TCN's to lip reading, introducing ResNet plus Multiple Scale TCNs or ResNet plus Dense TCN's, achieving 85.3% and 88.4% Top-1 Accuracy on LRW Martinez et al.

Preprint. Under review.

[2020]. Some additional resources we looked at involved lip reading, but not directly trained with the LRW dataset. Garg et al used an SVM architecture on the Miracle VC1 dataset and achieved 76% validation accuracy. Their LSTM models including a VGG + LSTM model had low accuracies under 40% validation accuracy Amit et al. [2016]. One additional resource we looked at was the C3D architecture introduced by Tran et al Tran et al. [2015]. Though not introduced specifically for lip reading, the C3D model used 3D convolutional layers to process videos and describe them. Tran et al showed that using these 3D convolutional layers were able to effectively learn spatiotemporal features. Currently, the majority of the State of the Art architectures include a frontend with either 3D Conv or ResNet or both, and a backend of a TCN, Bi-GRU, or LSTM.

3 Methodology

We decided to test out a few of the architectures commonly used for the lip reading task after applying our own version of preprocessing.

3.1 Preprocessing

Figure 1: Video Frame

Figure 2: Processed Video Frame

First, we had to preprocess the video clips. The video clips come in color, 256x256 dimensions, 25 fps, with 29 frames each.

The preprocessing pipeline we used consisted of horizontally aligning, scaling, and centering the mouth, cropping the size to 64x64, and grayscaling the image. We used the dlib facial landmark detector to identify the regions of interest for the mouth. This gave us the location of landmarks on the mouth that allowed us to calculate the tilt and size of the mouth to horizontally align and scale all the mouths to the same general size. Then, in order to keep only the most important regions, we cropped to 64x64. This is different from most of the reference papers, as they seem to crop to around 80x80 or 96x96.

We actually ran into a few issues with this preprocessing due to the dlib facial detector not working as optimally as we would have liked. Occasionally, the detector would not detect the face, or detect multiple faces, making it a pain to have consistent inputs, as when no faces were detected, our video would simply lose frames. Then, when multiple faces were detected, programmatically it's pretty hard to truly figure out which face is supposed to be the designated subject of our video. As a result, when multiple faces were detected, we simply chose the face that was closest to the center of the video, figuring that the face that is centered would be the subject. This has flaws however, as many videos the subject is not directly centered. Potentially a better way to have chosen these faces would have been to choose the largest face, as the largest face should be the closest face, and therefore the subject of the video. Ultimately, in order to solve the issue of videos missing frames (or sometimes even with extra frames from the dataset), we simply discarded the videos that contained less than 29 frames, and discarded the end frames from videos with more than 29 frames. In total, around 5 percent of the videos had to be discarded due to these issues.

Compared to most of the state of the art architectures, our preprocessing is relatively simple. Most papers incorporate data augmentation into the preprocessing to have larger and more robust amounts

of training data. In addition, Ma et al. performs a transformation to a mean face shape so that the faces in each video are more consistent Ma et al. [2021].

In addition, each clip is accompanied by metadata that tells us exactly which frames the word occurs within. The word duration is given, and each word is centered at the middle of the video, so it is simple to extract the frames. Some papers used this metadata to input a word boundary to each clip as additional information. We decided not to do this, however, models that did include word boundaries seemed to achieve a boost of around 5% accuracy Feng et al. [2020]

Then, with this preprocessing, in the interest of time, we randomly chose 25 words to experiment with, taking 200 training videos out of the 1000 total to do our training on. We keep the same 50 videos from the testing set to test on. The words we chose are given in the below table

AGAIN	COMMUNITY	EDUCATION	FAMILIES	FOUND
GETTING	HAPPENING	HOUSING	INQUIRY	LEAVE
LIVES	LIVING	LOOKING	PERSON	PLACE
SENIOR	SERIOUS	SHOULD	SIDES	STAGE
TALKING	TONIGHT	WEEKS	WHILE	WITHIN

Table 1: Table of Chosen Words

3.2 Models

In this section, we describe the architectures we decided to train with. All of the models were trained and evaluated using CrossEntropyLoss(), as this loss is suitable for classification

1. 2D CNN + LSTM
2. ResNet-18 + LSTM
3. 3D CNN
4. 3D CNN + LSTM
5. ResNet + TCN
6. VGG-16 + LSTM

3.2.1 2D CNN + LSTM

The first model we tested was two 2d convolutional layers followed by an LSTM model from scratch. In this model, the architecture was conv layer,max pool, conv layer, max pool, LSTM,fully connected layer. The LSTM takes an input of 64*32*32 and outputs 256 values. This then goes through a fully connected layer so that the output of the model has the size of the number of possible classes. The idea of this model was to see how much the model can learn with little parameters (but still have a temporal network like an LSTM).

3.2.2 ResNet-18 + LSTM

Then we experimented with the default ResNet-18 model with pre-trained weights as a feature extractor that feeds into a two layer bidirectional LSTM for classification. The ResNet-18 model has 18 layers consisting of convolutional layers, ReLU activation layers, batch normalization, average pooling, and a fully connected layer. The first convolutional layer was adjusted to take the grayscale image with only one channel and the last two layers were removed to extract the features to feed into the LSTM. Additionally, a multihead Attention layer is added after the LSTM layers to put more weight on important sections of the data. Since it is a deep CNN model, dropout and weight decay was added to the architecture to reduce overfitting.

3.2.3 3D CNN

The next model we experimented with consisted of multiple layers of 3d convolutions, batch norms, and pooling and ending with two fully connected layers. We believed that with convolutional layers being good for extracting spatial information, using 3D convolutional layers would allow our model to extract the spatial features from each individual frame, while also incorporating the temporal dependencies between frames. This model ultimately ended up with $\sim$50 million parameters.

3.2.4 3D CNN + LSTM

Finally, we tested 3D CNN as a feature extractor feeding into a two layer bidirectional LSTM. The 3D CNN consists of three 3D convolutional layers with batch normalization and ReLU activation after each one. A max pooling and dropout layer is added after the first two convolutional layers to reduce dimensionality and overfitting.

3.2.5 Resnet + TCN

The next model we tested was a ResNet + TCN model trained from scratch. In this model, the frontend ResNet consisted of 4 layers of ResNet Blocks. Each of these blocks consisted of two 2D convolutional layers with batch normalization and ReLU activations, adding on the residual connection at the end. At the end of the ResNet block, we averagePool over each frame to create a feature vector for each frame. This feature vector serves as the input to our TCN backend.

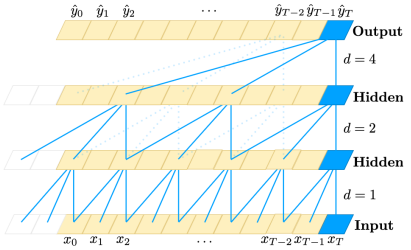

Figure 3: Illustration of TCN Bai et al. [2018]

A TCN is essentially multiple 1d convolution layers placed after each other. At each subsequent 1d convolution, dilation and padding are adjusted such that the later layers are able to take in information from larger sections of the initial inputs. Each input and output of the TCN are the same size.

Finally, we pass our final TCN output through a single Fully Connected layer to receive probability scores for each of our classes. The full ResNet + TCN model has 5 million parameters in total. Most of these parameters, around 4.9 million, are concentrated on the ResNet model.

One thing to note is that our TCN was still set up with causal convolutions. This means that each output from each TCN layer only considers the frames that have occurred before the frames we are calculating a value for. For lip reading, this causal condition is not necessary, as typically in applications, we will have access to the full clip before making predictions.

3.2.6 VGG16 + LSTM

The final model we tested was a VGGNet16 + LSTM model using the pre-trained weights from VGG16. In this model, the VGG16 architecture with 13 convolutional layers, 5 pooling layers, and 3 dense layers were used, but the final classification layer was removed so that the LSTM could be paired with it. After the VGG a batch normalization layer with ReLU activation was applied before a two layer bidirectional LSTM with dropout was applied. Finally, a fully connected layer with the number of classes was added so we could classify by word. The model ended up with around 21M parameters.

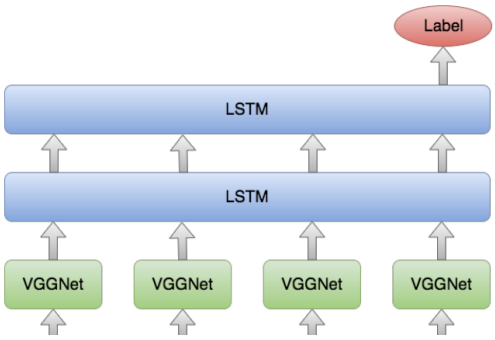

Figure 4: VGG + LSTM diagram Amit et al. [2016]

The VGG typically takes 224x224x3 as input with the option of a batching dimension as well, but any multiple of 32 for the height and width of the image will work in the model. To improve computation time we used an input size of 64x64x3 with the batching dimension as the product of batch size and the number of frames in the clip. Additionally, as we were using grayscale videos, every frame was replicated three times to 64x64x3 from 64x64x1 to keep the VGG mostly unmodified.

Since the VGG pre-trained weights were found from training on the ImageNet dataset, it may not have added much more to the model than using randomly initialized weights, but it appears to have slightly helped the training of the model. Gradient clipping was also implemented during the training

to prevent any gradient explosions from adversely affecting the weights and causing the network to diverge.

4 Experiments

4.1 2D CNN + LSTM

The 2D convolution model with LSTM was trained with a learning rate of 1e-4 with the Adam optimizer and Cosine Annealing learning rate scheduler. Only 200 clips were used for training. After 20 epochs, the model achieved a accuracy of 13.8%. Such a drastic difference between Training and Validation Accuracy could indicate overfitting as a result of not having enough data.

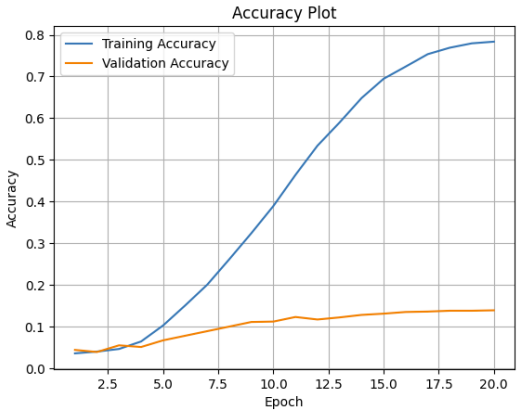

Figure 5: Plot for 2D CNN + LSTM

4.2 ResNet-18 + LSTM

Our ResNet18 + LSTM model was trained with a learning rate of 1e-4 using the AdamW optimizer with weight decay of 0.1 and a Cosine Annealing learning rate scheduler for 20 epochs. Dropout of 0.25 is added to the LSTM layers to prevent overfitting the model. Using the deeper ResNet-50 model or increasing the number of hidden dimensions did not make a significant difference in accuracy, likely due to overfitting. After fine-tuning, the model reaches a test accuracy of 60.2%.

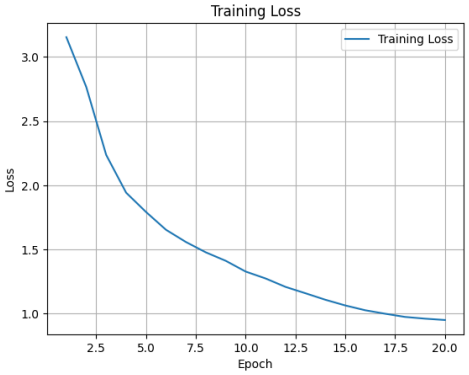

Figure 6: ResNet-18 + LSTM Loss

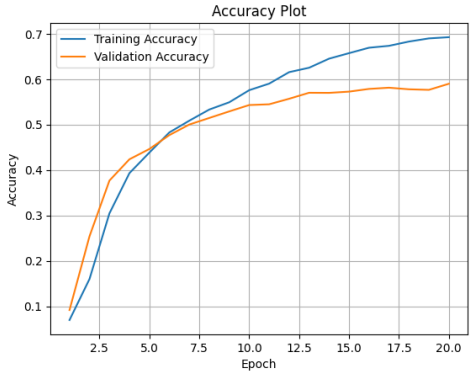

Figure 7: ResNet-18 + LSTM Accuracy

4.3 3D CNN

Our 3D convolution model was trained with a learning rate of 1e-5 with the Adam optimizer and a StepLR scheduler multiplying the learning rate by .7 every 5 epochs. After 32 epochs, our model had achieved a test accuracy of 67.6%.

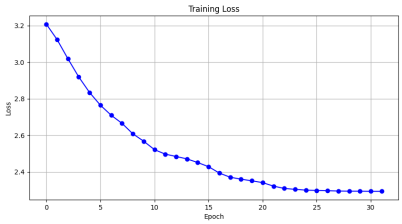

Figure 8: 3D CNN Train Loss

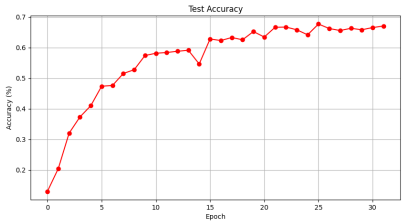

Figure 9: 3D CNN Test Accuracy

4.4 3D CNN + LSTM

Similarly, the 3D CNN + LSTM model was also trained with a learning rate of 1e-4 using the AdamW optimizer with weight decay of 0.1 and a Cosine Annealing learning rate scheduler. The 3D CNN + LSTM model overfits even with the dropout and batch normalization in the 3D CNN so dropout of 0.5 is used in the LSTM layers as well. When using only 200 clips for training, the training accuracy is significantly higher than the validation accuracy. This shows that for the more complex models, much more data is needed to match the complexity of the model and prevent overfitting. After training for 20 epochs, the model achieves a test accuracy of 69.5%.

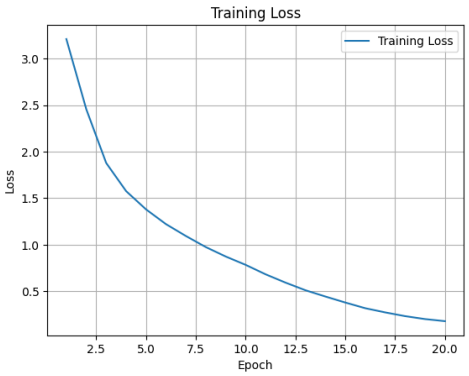

Figure 10: 3D CNN + LSTM Loss

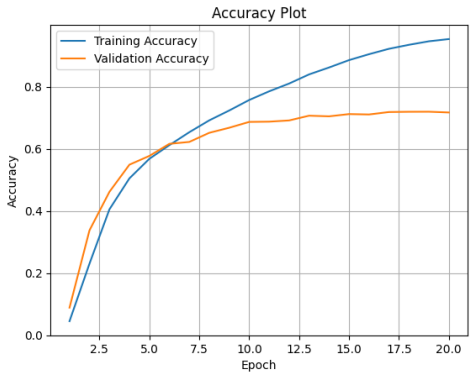

Figure 11: 3D CNN + LSTM Accuracy

4.5 ResNet + TCN

Our ResNet + TCN model was trained with a learning rate of 1e-3 with the AdamW optimizer and a StepLR scheduler multiplying the learning rate by 0.5 every 10 epochs. After 29 epochs, out model achieved a test accuracy of 77.8%. The plot for this model can be seen in figure 12. Because this model is relatively small, with only 5 million parameters, we can see that even on the 200 example training set, the model is not quick to overfit, and test loss more or less follows the train loss down.

4.6 VGG-16 + LSTM

Our VGG-16 + LSTM model was trained with a starting learning rate of 1e-4 using the Adam optimizer and a Cosine Annealing learning rate scheduler that reset every 10 epochs.

An SGD optimizer was tested, but it didn't seem to provide as much improvement as the Adam over the same amount of time. A step decay learning rate scheduler was tested as well, but it did not

provide sufficient results and seemed to get stuck at a local minima. Using only 200 clips for training resulted in overfitting of the model, so the full 1000 clips was used for the final chosen parameters.

Using a batch size of 32 after 30 epochs our model achieved an accuracy of 78.1% accuracy.

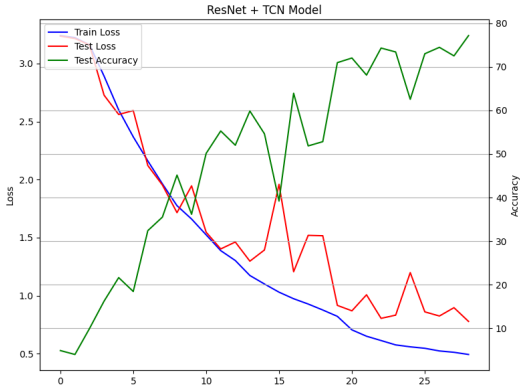

Figure 12: Plot for ResNet TCN

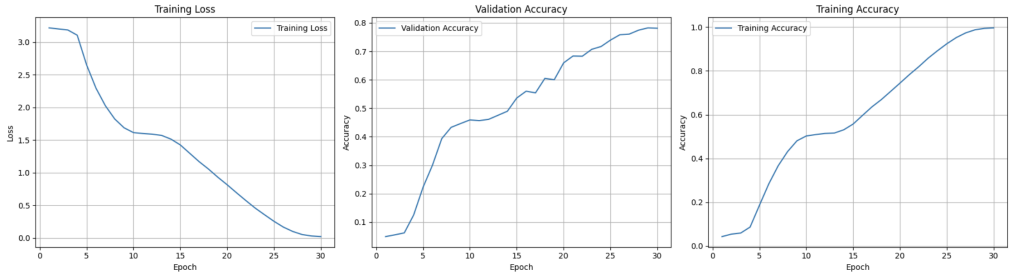

Figure 13: Loss and Accuracy for VGG16 + LSTM

5 Conclusion

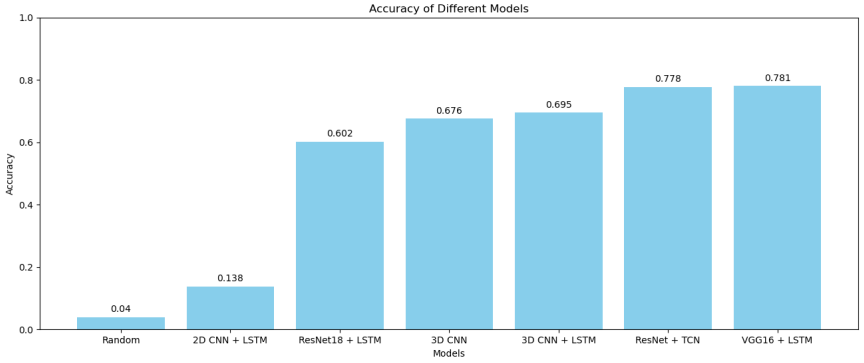

Figure 14: Accuracies of Models

In this paper, we implement multiple popular architectures for the task of lip reading. We first start with the all in one 3D conv model which extract spatiotemporal features all at the same time. Then, we move onto more recent models which consist of a frontend to extract feature vectors from individual frames, and a backend that takes in these features and performs sequence learning to extract temporal relationships.

Our top performing models were the ResNet + TCN and VGG16 + LSTM models, at 77% and 78% respectively. Below, we have the confusion matrix for the TCN model, showing which true labels were often confused with which words. Strangely enough, our most misclassified pair was classifying Weeks as Place, two seemingly unrelated words.

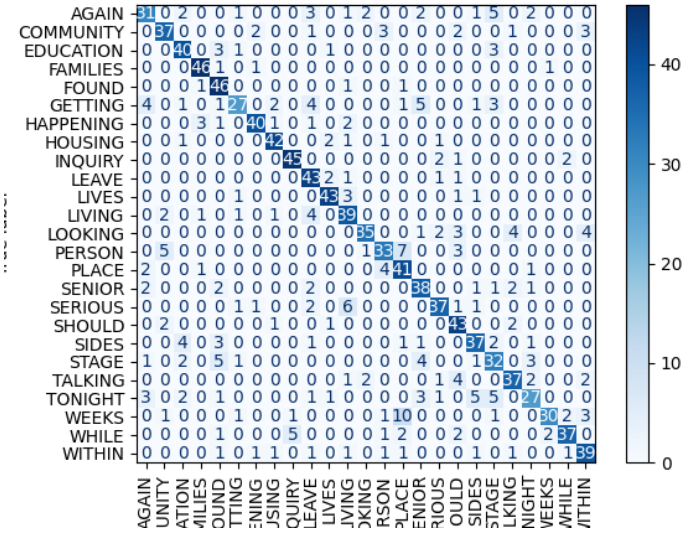

Figure 15: Confusion Matrix for ResNet TCN

Overall, the results are not spectacular, especially when compared to the state of the art models. This could possibly be the result of having a relatively simple model architecture, as well as limited data.

Because we are training on only 200 examples per word for some models, such as the ResNet + TCN, rather than the 1000 possible examples, our models are more likely to overfit and not learn more general lip reading connections. Even without changing the model architecture, our performance could be greatly improved with using all the available data and/or performing data augmentation to increase the size of the training set as can be seen by the performance of the VGG16 + LSTM model.

Ultimately, compared to the state of the art models, our models are relatively simple. Future improvements to our models could be implementing Multi-Scale TCN's or Bidirectional GRU's instead of LSTMS. In addition, the top models use a variety of smoothing methods, initialization methods, and ensembling methods to allow their models to extract the highest accuracy.

Our current accuracy leaves a lot to be desired, but with a little more data and time to train, our models' performance has a lot of potential to improve.

References

A. Amit, J. N. jnoyola, and S. B. sameepb. Lip reading using cnn and lstm. 2016. URL `https://api.semanticscholar.org/CorpusID:22889293`.

S. Bai, J. Z. Kolter, and V. Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling, 2018.

J. S. Chung and A. Zisserman. Lip reading in the wild. In *Computer Vision–ACCV 2016: 13th Asian Conference on Computer Vision, Taipei, Taiwan, November 20-24, 2016, Revised Selected Papers, Part II 13*, pages 87–103. Springer, 2017.

D. Feng, S. Yang, S. Shan, and X. Chen. Learn an effective lip reading model without pains, 2020.

P. Ma, Y. Wang, J. Shen, S. Petridis, and M. Pantic. Lip-reading with densely connected temporal convolutional networks. In *2021 IEEE Winter Conference on Applications of Computer Vision*

(WACV). IEEE, Jan. 2021. doi: 10.1109/wacv48630.2021.00290. URL http://dx.doi.org/10.1109/WACV48630.2021.00290.

B. Martinez, P. Ma, S. Petridis, and M. Pantic. Lipreading using temporal convolutional networks, 2020.

T. Stafylakis and G. Tzimiropoulos. Combining residual networks with lstms for lipreading, 2017.

D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3d convolutional networks, 2015.