CS-GY 9223 - Deep Learning

Final Project Report

Vishnu Thakral (vvt223@nyu.edu)   **Darshan Solanki** (das968@nyu.edu)

Problem Statement:

As per current stats, over 5% of the world population i.e. around 470 Million people have hearing problems with 34 million of them being children, this number is expected to rise to 900 Million by 2050.

Our project targets making communication much simpler, faster and accurate for these people. Our project focused towards lip reading will be able to help people, both disabled and otherwise (as language translators) thus ensuring that communication does not become a barrier.

Literature Survey:

Recent deep learning approaches were implemented for LipReading as a classification problem. Wand et al. [1] introduced LSTMs for lip reading at the word level, which we have taken as an inspiration in moving forward with CNN-RNN architecture. Another work by Chung & Zisserman [2] made use of spatiotemporal CNNs for word classification on the BBC TV dataset. Then for the first time, Assael et al. [3] created LipNET, a phrase predictor that uses spatiotemporal convolutions(STCNN) and bidirectional GRUs which does sentence level prediction and introduced performance metric like Word Error Rate(WER), Character Error Rate which we have used in our work. In Addition to that Connectionist temporal Loss (CTC) developed by Graves et al[4] which became a critical advancement in measuring STCNN, work of Assael et al. uses that as well.

Our model is primarily inspired by Abiel Gutierrez and Zoe-Alanah Rober[5] in which Transfer Learning VGG19 along with BiLSTM-CNN architecture is used. This architecture was applied against the MIRACLE-V1 dataset which is smaller in size. We are using Grid Corpus dataset[6] publicly available and have much richer data than MIRACLE-V1.

Dataset:

We are using GRID corpus[6] produced by Google DeepMind, consisting of 34 speakers, each narrating 1000 video responses. The vocabulary is categorized into 6 groups, i.e. {bin, lay, place, set}, {blue, green, red, white}, {at, by, in, with}, {a, . . . , z}\{W}, {zero, . . . , nine}, and {again, now, please, soon}, yielding 64000 possible sentences. For example, two sentences in the data are “set blue by A four please” and “place red at C zero again”. The videos for speaker 21 are missing, and a few others are empty or corrupt, leaving 32746 usable videos. Total size of the data is around 34 GB.

Proposed Methodology:

We chose our architecture for a couple of reasons:

1) **Transfer Learning:**

As videos in discrete sense are individual frames aligned consecutively, in our case we are training our model on 22000 videos, comprising ~2.8 million image frames, 6000 for validation and 4000 for testing. Extracting features with help of transfer learning decreases the number of trainable parameters. We fine tune the feature encoding with a combination of CNN-RNN sequence architecture.

2) Sequence to SequencePrediction:

As sentences spoken by various speakers are sequences and thus have an interdependency. To capture those semantic relationships, sequence prediction became a necessity.

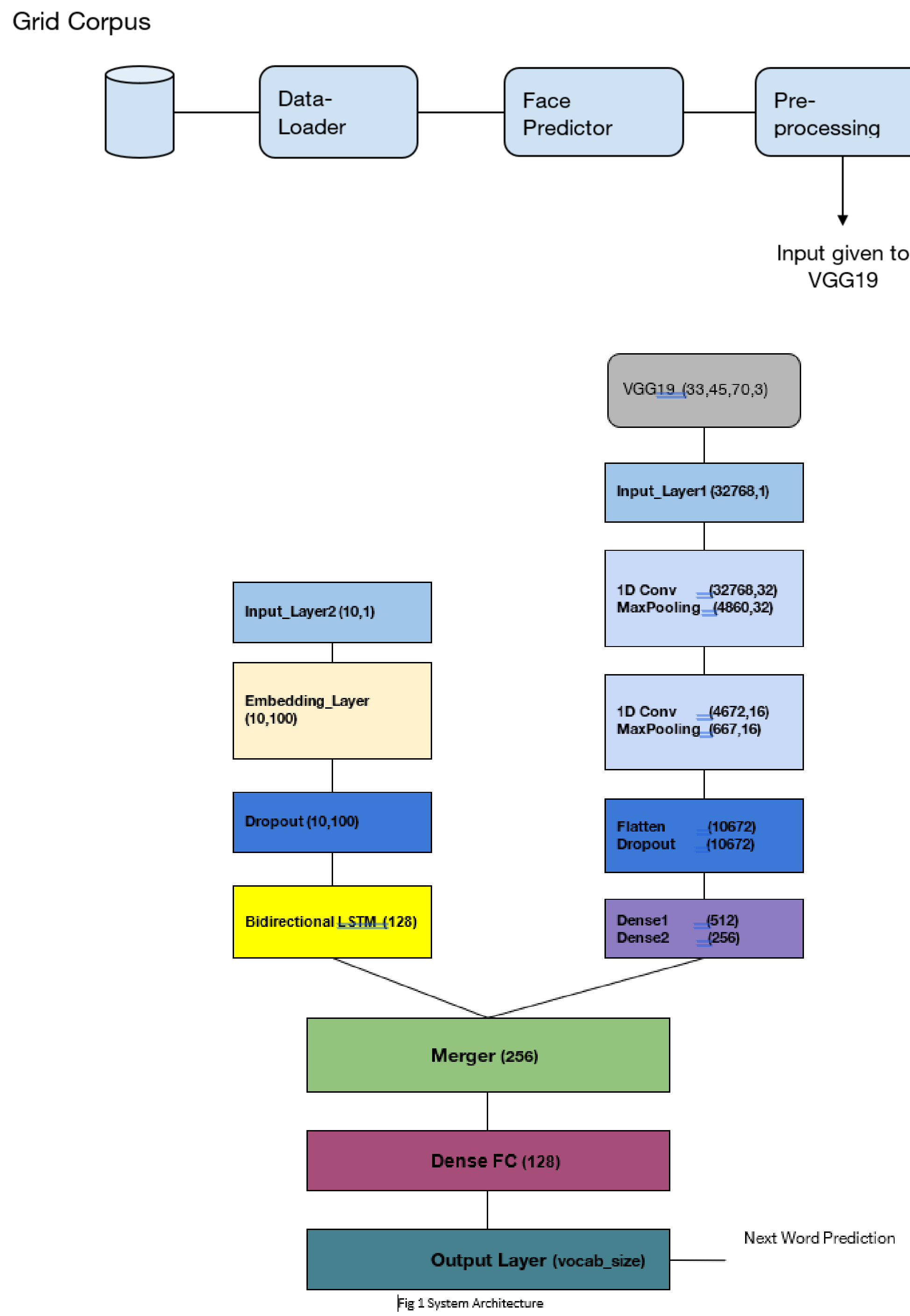


Fig 1 System Architecture

Working:

Phase1: Preprocessing:

Each speaker has produced 1000 videos and each video has a corresponding alignment file which encodes the sentence spoken in the video with their start and end timestamp in milliseconds. Each video is 3s long with 25fps. The videos are processed with Dlib face detector shape_68_landmark points [7], which predicts the face bounding box with 68 points as face points in a given frame. The pseudo code is:

```
1) Download the dataset and split into seen and unseen speakers
2) Make train, val, and test files containing location of each videos and respective alignments
3) For each video:
    a) Read the video frames and detected the mouth region via shape_68_landmark_detector
       pretrained model
    b) Capture the frames in correspondence of each alignment words given start and end
       time
    c) Add max_length padding to make all video frames consistent in shape
4) Store it as key-value pair in dictionary and pass to VGG19
```

So final word video frames are of 4 dim → (max_time_frame,width,height,channel) → (33,45,70,3) which is stored as a pickle file.

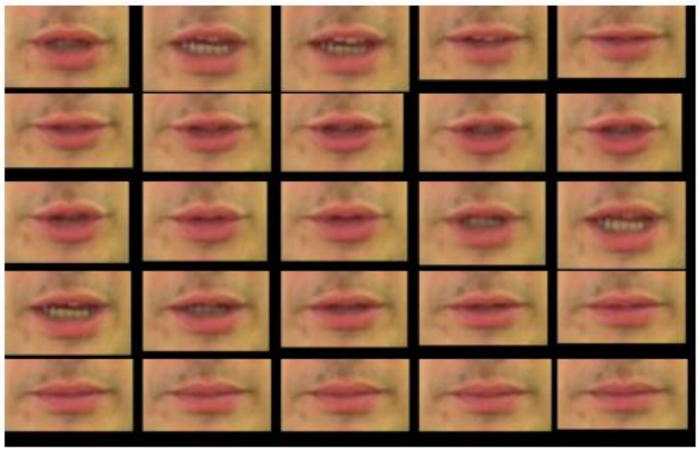


Fig 2. Cropped Image Frames for random speaker

Phase 2: Transfer Learning

We have chosen VGG19[9], trained on 1.2 billion images by google deepmind. One of the reasons for using VGG19 is flexible input parameters unlike in InceptionV3 where the input size is fixed. If we inflate the input_size to (33,229,229,3) the image might fail to capture the intrinsic details of lip movements due to blurring effect.

Phase 3: Model Creation for fine-tuning

The output vector consists of (32768,1) flatten vectors which are sparse due to padding. From our analysis, most of the words spoken lie in frame length duration of 12-20, but due to max padding of length 33, vectors become sparse. To reduce the dimension of the output we send it through 1D Conv, as studied in[8], and it shows remarkable results in capturing temporal features. With 2 stacks of 1D ConV applied (includes MaxPooling and Dropout), we process the data using a Flatten Layer and 2 FC layers to reduce the size to 256, as shown in Fig 1.

On the other Hand, we have a sequence with a max length of 10 words where special tokens [‘start’] and [‘end’] are appended to track the beginning and end of sentences. These words are fed in customized embedding, trained on train_corpus vocabulary of 55 words in total. We have used the gensim python package[10] to make an embedding matrix of size (55,100) where 55 is vocab_size and 100 is hidden dimension embeddings. Interesting results were seen when we performed PCA on word embedding to visualize the words connection. As seen in Fig 3, words that are used in similar context are clustered near to each other, which is, colors, numbers, single characters, nouns, prepositions.

Finally this sequence is fed into Bidirectional LSTM of 128 cells which takes in the input sequence and uses two LSTM one with same order of sequence and other reversal of input sequence to capture both forward and backward context of the sentence. Then we merge the model to pass through FC layers and predict the next word with Softmax at the end. Fig 4 illustrates the training process.

We have used cross-entropy loss, as we are performing the Next Word Prediction NWP from the distribution among all words in the corpus. Formula would be: $-\Sigma_i \log((\exp(Wx_i) / \Sigma_j \exp(Wx_j))$ where i,j range from 1→ vocab_size. The optimizer used is Adam with tuning parameters like decay rate set to 1e-7,1e-6.

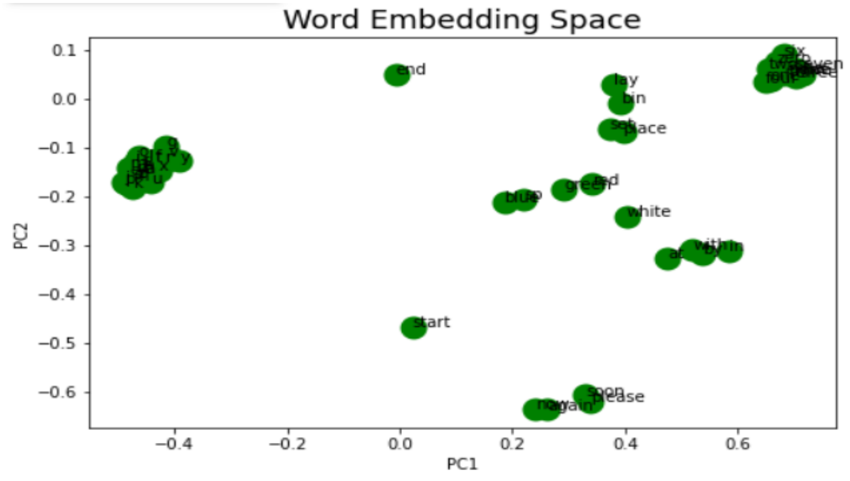


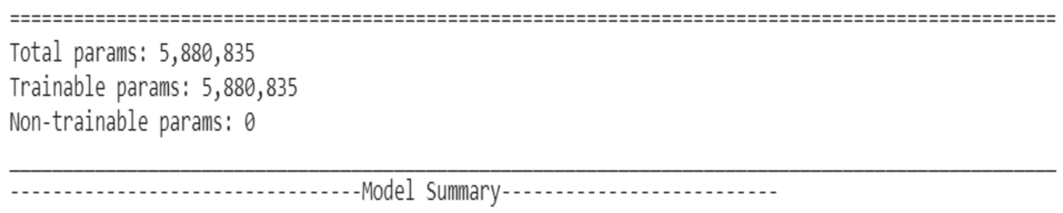
Fig 3 PCA for Word Embedding Visualization

Tokens	Encodings	Next Word Prediction
start	Word 1	Word 1
Start word 1	Word 2	Word 2
Word 2 word 3	Word 3	Word 3
Word n-2 word n-1	Word n	End

Table 1. Model Prototype Working

In this case, we used window size 2 to get the seq 2 timestamp back, mainly because LSTM are difficult to train and we need to keep the word importance relevant looking back a few timestamps away vocab size is small and has specific structure format. This is easily changeable in our code to train upon the entire sequence.

Internal Training Workflow:



```
==========================================================================================
Total params: 5,880,835
Trainable params: 5,880,835
Non-trainable params: 0
__________________________________________________________________________________________
--------------------------------Model Summary-------------------------
```

Fig 4. Model Trainable Parameters

As seen in Fig 4, the model summary has been printed with ~5.9 million trainable parameters. Let's see the internal working of each step-in detail.

Dataset Formation:

As we had 32GB max memory GPUs Tesla V100 on HPC. We made use of 2 such gpus in distributed fashion for training. To make dataset in trainable format we stored as a key-value pair for entire sequence, where key is representing img flatten features coming from VGG19 for each word given <speaker> and <video> in sequence, value being actual image feature vector for instance here is prototyped sample.
s10_v1_w1 : vector (32768,1)
s10_v1_w2: vector (32768,1)

Advantage of storing in a dictionary:

- Constant Lookups
- No consecutive space consumed in memory
- easy to perform batch training.

Our labels, Y being the word to be predicted in next sequence iteration, which is one-hot encoding at the output. Loss used is categorical_crossentropy/softmax loss.
So, at the output we get probability distribution among all words in the corpus.

Measures to Prevent Overfitting:

- **Generalization**: This is one of the important aspects we focused on to prevent overfitting. We used batch-wise training with customized data generator functions to generate the correct format as discussed in Fig 4.
- **Dropouts**, **Regularizer** : As, the network is dense hence we used dropouts (0.3,0.1,0.5) between the layers. This would help the model to pass on the information via different sets on neurons. We have also used L1, L2 norms to attain complex weights in the model.

- **Early Stopping**: We included the callback function for early stopping monitored on val_accurracy, this helped us when val_accuracy did not update for consecutive 10 epochs. It was suited for our case as we were training for a max of 500 epochs and early stopping ensured to stop the training when the model was no longer learning before the actual 500 epoch count was reached.

Model CheckPoint and Distributed Trainings:

As seen in tensorflow documentation of distributed training[11] . Each batch is split into small sub-batches and independently processes the information updating the weights in sync at the end of each epoch. With the HPC cluster, we leverage the distributed training with mirrored variables which distributes the load to different nodes in a cluster and communicates for weight synchronization updates at the end of each epoch. It has also shown training compute per epoch is 2x faster than sequential batch training as we assigned 8 tasks per node. Due to Communication failure chances, we save the model for best val_loss seen till that instance with Model Checkpoint. ModelCheck point is performed on monitoring the val_loss and saving the weights only when better minimum losses are observed.

Model Evaluation and Results:

Hyper Parameterization:

We experiment with a bunch of hyper-parameters, to see results on training and testing. Here are the statistics/results for each experiment with each combination.

Learning Rate	Batch Size	Regularizer	Epochs	Train | Val Acc
2e-4	64	L1 - 0.1 L2 - 0.1	100,200, 500	40.2% | 38.4%
3e-4	64,128	L1 - 0.5 L2 - 0.5	250,500	41.2% | 39.8%
1e-5,1e-4	128, 256	L1 - 0.8 L2 - 1	150,400, 500	42% | 44.9%

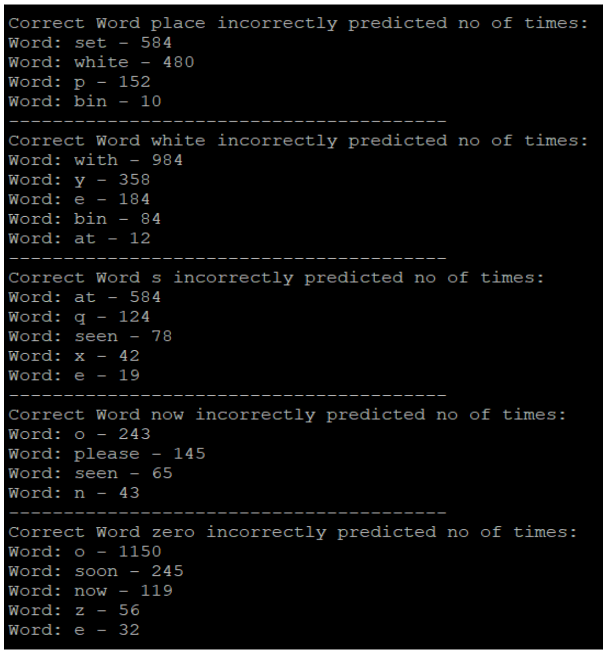
Table 2: Results with different Hyper parameters

From the experiments conducted, 1e-4 gave the best results with batch_size 256 for 500 epochs with training time of 12 hrs on Tesla V100 SMX2 GPU with 32GB Memory.
Getting image encodings from VGG19 took around 28hrs with heavy CPU resource usage. The task was achieved by developing sbatch files to submit to HPC clusters with internal commands to run inside the Singularity/Docker image with all dependencies installed.

Result Explanation and Test Visualization:

On the test sample we apply the same preprocessing steps (crop, pad, get vgg encoding, flatten) used on validation and training data. After this we pass the test video along with a 'start' token to the model for prediction. The model predicts the next word, which is again fed to model as next input to get next word prediction. The breaking condition would either be that video frames get exhausted or the model predicts the "end" word.

To check the correctness of the prediction, we implemented a Word Error Rate (WER) metric inspired from [4], where it calculates the number of times incorrect words are predicted for corresponding true labels seen in the test dataset. With this, we were able to infer which words were most predicted wrongly. Hence it would be useful to tweak the model to make it better in future work. In Fig 5, we can see the top 5 words incorrectly predicted with respective counts. The WER for our model is 58.3% which is which is majorly because the train accuracy was lesser than expected.



```
Correct Word place incorrectly predicted no of times:
Word: set - 584
Word: white - 480
Word: p - 152
Word: bin - 10
----------------------------------------
Correct Word white incorrectly predicted no of times:
Word: with - 984
Word: y - 358
Word: e - 184
Word: bin - 84
Word: at - 12
----------------------------------------
Correct Word s incorrectly predicted no of times:
Word: at - 584
Word: q - 124
Word: seen - 78
Word: x - 42
Word: e - 19
----------------------------------------
Correct Word now incorrectly predicted no of times:
Word: o - 243
Word: please - 145
Word: seen - 65
Word: n - 43
----------------------------------------
Correct Word zero incorrectly predicted no of times:
Word: o - 1150
Word: soon - 245
Word: now - 119
Word: z - 56
Word: e - 32
```

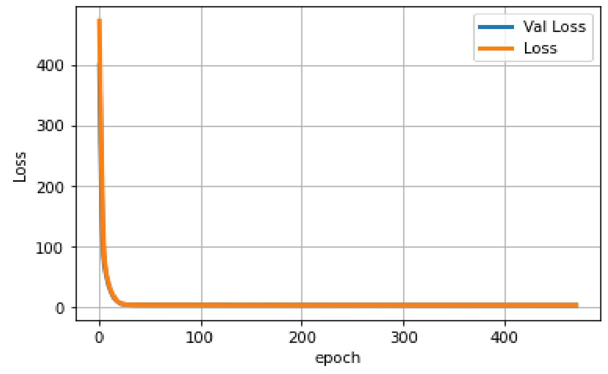
Fig 5 Word incorrectly predicted

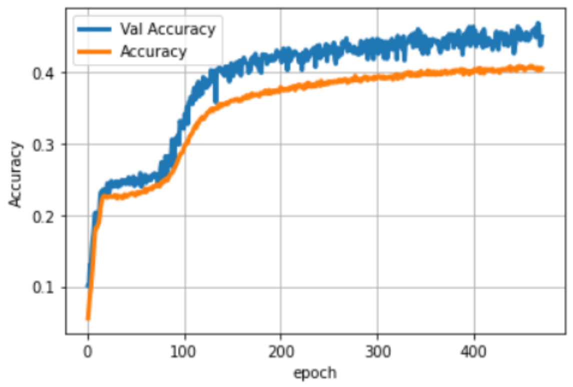
Here it can be seen that words like zero (0), which is interpreted as o due to the fact that mouth opening is nearly the same for both. Whereas words like white and with are close in the embedding structure could be the reason for incorrect prediciton.

In addition to this, shown in fig 6 are standard plots which attained close to zero loss quickly, but faced difficulty in shooting up the accuracy. Though accuracy did not rise but loss values continuously decreased in the case of our model. Based on our research we believe use of softmax or CTC loss as done in[3] is a better estimator than accuracy metric in sequence prediction models.

Test Video Link: https://youtu.be/ox4Etk49RoM

Our model was able to predict some words in the sequence but did not predict 6 words and ended up predicting 5 words before getting end sequence word in prediction.





Conclusion:

Our model achieved an accuracy of around 45%, which when compared to other benchmarks in[5], is a little better performance in terms of validation and testing for Frozen CNN-LSTM architecture. Also, we believe, accuracy might not be a good metric to evaluate as it is predicting word probability than sequence probability given ground truth text. We tried to infer the lacking capabilities of the model with WER as explained in fig 5 which indicates improvements in refining the model architecture. However, we formulate our training as near to industry exposure with distributed computing and running over the cloud network. Lastly, applied various methods to prevent overfitting, in-memory consumption and hyperparameter tuning to select the best trial experiment suited for our case.

Future Work:

1) Our model lacks in achieving good results, as LSTM are arduous to train. Our need would be supplanted with better algorithmic models like BiGRU and CTC loss functions.
2) Also, this Grid Corpus dataset, has fixed length video representation and corpus consists of only 53 words, certainly for real-world applications we would like to explore LRW dataset [13] having variable length videos and more words.
3) We believe, character-based learnings would do better than word-based learnings and it will also resolve the problems of unspoken words in the training corpus.

Acknowledgments:

We thank the Grid Corpus community [1], for providing open-sourced data for academic/research-oriented work. We also thank NYU Tandon for providing HPC clusters with edge-cutting GPU resources for computational purposes. Lastly, we deeply thank Prof Chinmay Hedge as a mentor for this project and guiding us through the coursework.

References:

[1] M. Wand, J. Koutnik, and J. Schmidhuber. Lipreading with long short-term memory. In IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 6115–6119, 2016.
[2] J. S. Chung and A. Zisserman. Lip reading in the wild. In Asian Conference on Computer Vision, 2016.
[3] Yannis M. Assael1 et. al “Lipnet: end to End sentence level lipreading”,arXiv:1611.01599v2, Dec 2016

[4] Graves, A., Fernandez, S., Gomez, F., and Schmidhuber, ´ J. Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. In ICML, Pittsburgh, USA, 2006.
[5] Abiel Gutierrez, Zoe-Alanah Robert “LipReading Word Classification”, stanford report
[6] Grid Corpus Dataset - http://spandh.dcs.shef.ac.uk/gridcorpus/
[7] https://www.pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/
[8] Karen Simonyan, Andrew Zisserman,”Very Deep Convolutional Neural Network for Large Scale Image Recognition”;arXiv:1409.1556v6 September 2014
[9] Serkan Kiranyaz et.al 1D Convolution Neural Networks and Applications; arXiv:1611.01599v2 [cs.LG] 16 Dec 2016
[10] Gensim Documentation - https://buildmedia.readthedocs.org/media/pdf/gensim/stable/gensim.pdf
[11] MirrorValued Documentation - https://keras.io/guides/distributed_training/
[12] LRW dataset - https://www.robots.ox.ac.uk/~vgg/data/lip_reading/lrw1.html

Source Code Link

https://github.com/Darshansol9/LipNet-Reader