Automated Lip Reading using Delta Feature Preprocessing and LSTMs

Andy Au, Adam Heins
Department of Mechanical and Mechatronics Engineering
University of Waterloo
Waterloo, Ontario, Canada
ch3au@uwaterloo.ca, awheins@uwaterloo.ca

***Abstract*—This paper describes a method for performing automated lip reading. There are many existing solutions to automated lip reading that involve deep learning approaches, with the most recent being the LipNet project [1] by the University of Oxford and Google DeepMind. LipNet is also the most accurate (95.2%) work that uses the publicly available GRID corpus dataset [2]. All of the existing solutions create their input by extracting faces, or portions thereof, from the still frames of videos of people speaking words. In this work, a different feature extraction approach was used. Instead of features from video frames themselves being used as input, the difference (delta) between the features of neighboring frames was used. Using the delta between features captures the movement information of the speaker, which is directly related to the words being spoken. Features were extracted from each video frame using a pre-trained convolutional neural network model, VGG-Face [3]. Neighboring features were then subtracted from each other and normalized to generate delta features. The delta features were used as input to an LSTM network, the output of which was the predicted word embedding. The network was evaluated on the GRID corpus dataset with an average accuracy of 86.30% for speaker-dependent tests and 57.83% for speaker-independent tests. The use of delta features improved both accuracy and training time in both cases.**

Keywords*—*lip reading; long short-term memory; convolutional neural networks; delta features

I. Introduction

Lip reading refers to the task of determining the words spoken by a person based only on visual input of the speaker's mouth with no accompanying audio data. Humans generally display a poor ability to lip read, with correct identification of compound words occurring only 21 ± 11% of the time [1]. Potential applications of automated lip reading include silent dictation in public places, speech recognition in noisy environments, and the captioning of silent films and videos [1]. It would thus be beneficial to automate the task of lip reading.

Automatic lip-reading generally consists of two stages, feature extraction and classification. Feature extraction often involves identifying and extracting the region around the speaker's mouth, as in [1] and [4]. In contrast, this work describes a method in which features are extracted from the entire face, rather than just the mouth region. This method preserves visual cues that may be present in other facial regions. Delta features are the differences between the features of adjacent frames. Facial features that do not contribute information to lip reading are automatically eliminated when the input to the classifier consists of delta features, because the delta of a feature that does not change is zero.

This work is focused on comparing the accuracy and training speed of classification with and without delta features. The model was evaluated in two ways. First, the model was tested in a speaker-dependent manner, in which testing and training was done on a single speaker at a time. Second, the model was tested in a speaker-independent manner. It was trained on a large set of speakers and tested using a completely separate speaker that was held out from the training set.

The paper is organized into the following sections. Section II discusses background and related research on automated lip-reading. Section III provides detail about the dataset used to test and train the model. Section IV describes the experimental methodology. Sections V and VI present the results of the work and related discussion. Sections VII and VIII discuss conclusions and future topics of research. Finally, a sample of the source code used for this work is presented in Appendix A.

II. Background Review

Research on automated systems for reading lips has received interest for many years, beginning with the PhD thesis of B. Petajan [5]. Petajan's work proposed using lip-reading to augment existing audio recognition systems, but more recent work has eliminated audio input entirely. Common methods for automated lip-reading include Hidden Markov Models (HMMs) and Support Vector Machines (SVMs). One of the first visual-only lip reading systems was done using HMMs using visemes and trisemes, but only on a limited dataset [6]. Gergen et al. in [7] used a larger dataset (GRID corpus) and developed a HMM/GMM system where mouth regions of images are trained on an LDA-transformed version of the Discrete Cosine Transform. This approach resulted in accuracy of 86.4%. However, it was only trained at the word-level, meaning only one word at a time was analyzed. In [8], optical flow was used for feature extraction which were then trained using SVM.

Recently, techniques have been developed using artificial neural networks such as recurrent neural networks (RNNs) and convolutional neural networks (CNNs). A recent collaboration between University of Oxford and Google DeepMind, known as LipNet, used a model that was trained at the sentence-level rather than the word-level. On the GRID corpus, LipNet achieved the highest state-of-the-art sentence-level accuracy to date, 95.2% [1]. LipNet has three main building blocks. First, spatiotemporal convolutional neural networks (STCNNs) are used to process the mouth region of video frames. The extracted feature data is then sent to Gated Recurrent Units (GRUs), which are a type of RNN with similarities to Long-Short Term Memory (LSTM) networks. The output from the GRUs is processed by a linear transformation and a *softmax* function. As with many LSTM models, this model was also trained with Connectionist Temporal Classification (CTC) [1].

III. The GRID Data Corpus

The GRID Corpus[1] dataset was used because it contains videos of full sentences being spoken, allowing for the possibility of sentence-level classifications in future work. The dataset is publicly available and contains video recordings of 1000 sentences spoken by 34 speakers. Each sentence consists of exactly six words, resulting in a sample size of 60,000 words per speaker. Leaving out Speaker 21, the videos of whom are missing, the dataset contains a total of 1,980,000 spoken words [2]. Each sentence follows the consistent structure of:

command → *color* → *preposition* → *letter* → *digit* → *adverb*

The vocabulary consists of a total of 51 words [2], which are identified in Table I. The letter W was excluded due to the fact that it takes much longer to say than all other letters [2].

The number of occurrences of each word in a speaker's dataset is not equal due to the fixed sentence format. Letters appear the least frequently, followed by numbers, and finally all other words.

TABLE I.
Description of vocabulary used by the speakers in the GRID corpus. The number of occurrences for each word of a given type, per speaker, is also shown.

Word Type	**Words**	**Number of Occurrences per Word**
Command	bin, lay, place, set	248 - 256
Colour	blue, green, red, white	248 – 256
Preposition	at, by, in, with	248 – 256
Letter	A-Z, excluding W	40
Digit	0-9	100
Adverb	again, now, please, soon	248 - 256

It should be noted that the sentences do not follow English language grammar. An example sentence is "Lay blue at A seven please". Each video has a fixed length of 3 seconds and frame rate of 25 frames per second, giving a fixed length of 75 frames. Each video also comes with an "alignment" file that records the interval of frames over which each word is spoken [2].

IV. Experimental Methodology

A. Feature Extraction and Pre-processing

Facial features were extracted from the raw video frames using a convolutional neural network (CNN). Due to the fact that it is impractical to train a CNN from scratch due to time and data limitations, the pre-trained "VGG-Face"[2] CNN by the Visual Geometry Group from University of Oxford was used instead. VGG-Face was based on the University of Oxford's VGG-16 architecture, which contains a sequence of convolution, pooling and fully connected layers. VGG-Face was trained with 2,622 identities for a total of 2.5 million face images [3]. Figure 1 shows the architecture of VGG-Face. Using a pre-trained model with a large dataset like that of VGG-Face provided high quality features for transfer learning. The VGG-Face model takes in individual images as input and outputs a flattened feature vector of length 512. Feature vectors were extracted by taking the output of the model (without the fully connected layers). The feature vectors of each video were then grouped per word and saved to an output binary file which could be loaded for training at any time. The feature extraction process was done separately from the training of the model due to the fact that feature extraction was the step that took the largest amount of time.

[1] Publicly available at *http://spandh.dcs.shef.ac.uk/gridcorpus/*.

[2] Source code available at *https://github.com/rcmalli/keras-vggface*. Based on *http://www.robots.ox.ac.uk/~vgg/software/vgg_face/*.

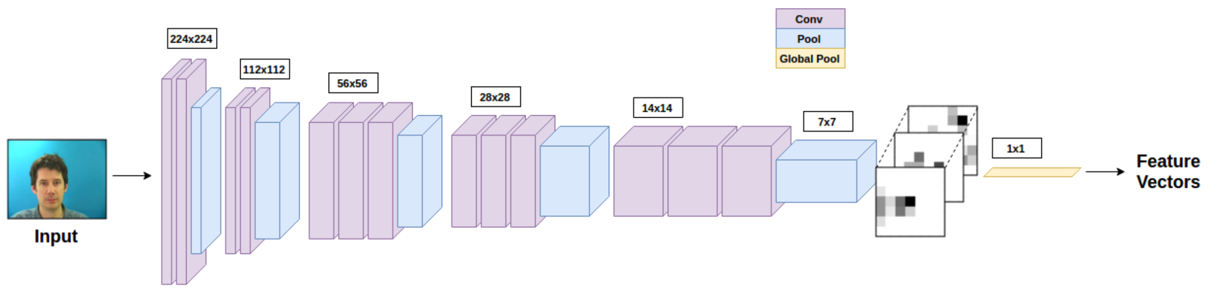

Fig. 1. Architecture of the VGG-Face CNN used for facial feature identification in each video frame.

Prior to training, the feature vectors were loaded from the binary files and preprocessed. First, the number of feature vectors for each word were truncated or extended so that each word had the same number of frames. It was decided that six frames should be used because, on average, six frames was enough to cover many of the shortest words in the dataset (such as the letters), and increasing the number did not show any increase in accuracy [4]. Words that had too few frames were padded with repetitions of the features of the last frame. Words with more than six frames were condensed by averaging the features of neighboring frames. Next, deltas of the feature vectors for each word were generated by subtracting neighboring feature vectors. Finally, the feature deltas were normalized and split into separate training and testing datasets for cross validation.

For output, each word is converted into a numerical vector using one-hot encoding where the length of the vector is equal to the size of the vocabulary. Word embedding techniques such as Word2vec were considered [9], but because the vocabulary size of GRID corpus is only 51 words, one-hot encoding is more efficient. Word2vec would be more efficient if the vocabulary size was a lot larger (i.e. greater than 200 words).

B. *Classification*

Classification was done using a similar architecture to [4], constructed with the Keras[3] framework. The model was a RNN. The first two layers were LSTMs with a size of 128 nodes each. LSTMs address the issue of vanishing and exploding gradients in traditional RNNs, as LSTM blocks are able to store representations of inputs for much larger time steps [10]. The final layer was a fully connected layer using *tanh* as the activation function. The network used mean squared error (MSE) as the loss function and Adam [11] as the optimizer.

The output of the network was a vector of length 51, representing the probability that the input matched each of the 51 possible words in the vocabulary. Figure 2 illustrates the prediction process for one word.

[3] Documentation can be found at *https://keras.io*.

C. *Validation*

The model was tested in two ways. The first way was to train and test on only a single speaker's dataset at once, referred to as speaker-dependent testing. The model was tested using k-fold cross-validation with k=5. The results for each speaker used were then averaged to calculate the final accuracy of the model.

The second way in which the model was tested was to train on multiple speakers and then test on a completely different speaker that was not part of the training set, referred to as speaker-independent testing. Twenty-five speakers from the data set were used for this task, for a total of 150,000 spoken words. During each test, 24 speakers were used for training, and the remaining speaker was used for testing.

In both scenarios, the model was trained and tested separately with normal features and delta features, so that the impact of using delta features could be examined.

V. Results

The first set of tests evaluated the performance of the model when trained and tested speaker-dependently. Speakers 1-5 from the dataset were used. The accuracies for each speaker were averaged to arrive at a final overall accuracy. The optimal number of training epochs was also recorded. The results are summarized in Table II.

TABLE II.
Results for Speaker-Dependent Model with and without Delta Features. Results were averaged over Speakers 1 - 5.

	With Delta Features	**Without Delta Features**
Mean Accuracy	86.30%	84.66%
Accuracy Std. Dev.	3.02%	2.06%
Optimal Training Epochs	29	74

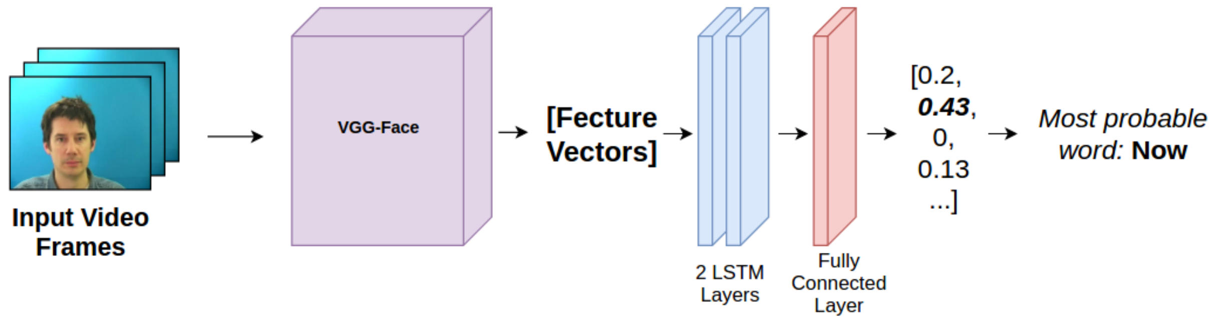

Fig. 2. End-to-end architecture used to classify a set of video frames as a spoken word, with example output.

Speaker-independent tests were also performed, in which the model was trained on all speakers except one, whose data was used for the test set. Twenty-five speakers (Speakers 1 - 15, 17, 18, 27 - 34) were used for this task. Three tests were done, in which Speaker 1, 2, and 3 were held out for testing, respectively. Testing was done both with and without delta features. The results for each test were averaged and are summarized in Table III.

TABLE III.
RESULTS FOR SPEAKER-INDEPENDENT TESTS IN WHICH A DATA SET OF 25 SPEAKERS WAS USED, AND A SINGLE SPEAKER WAS HELD OUT FOR TESTING AT A TIME. THIS WAS REPEATED FOR SPEAKER 1, 2, AND 3 BOTH WITH AND WITHOUT DELTA FEATURES.

	With Delta Features	**Without Delta Features**
Mean Accuracy	57.83%	40.05%
Accuracy Std. Dev.	5.01%	5.37%
Optimal Training Epochs	6	11

VI. DISCUSSION

In the speaker-dependent tests, the usage of delta features provided a slight accuracy increase of 1.63%. Notably, training to this level of accuracy without delta features required over 2.5 times more training epochs, considerably increased the training time required. One of the possible reasons for this result is that the speaker's facial movement is the most relevant data for predicting the words spoken. Delta features already emphasize movement, since they are the difference between the features of neighboring video frames. This provides a “shortcut” for the network, such that the model doesn't have to learn to isolate the movement itself and thus training time is reduced considerably.

The accuracy of 86.30% achieved using delta features is comparable to the best accuracy achieved by word-level classification, 86.4% [7]. However, it falls short of the state-of-the-art sentence-level classification performed by [1], which achieved an accuracy of 95.2%.

The speaker-independent tests resulted in a considerable difference in accuracy when delta features were used. The usage of delta features improved accuracy by 17.78%. Like the speaker-dependent tests, more epochs were required to train without delta features. In this case, training without delta features required 1.8 times as many training epochs.

The large increase in accuracy when delta features were used suggests that the movement a person makes when saying a word, captured by delta features, is more generalizable than the actual sequence of facial features. In other words, the movements people make when saying a given word are more generalizable than the overall appearance of people when saying the word. Thus, a model trained with delta features was more accurate when applied to as-yet-unseen speakers for testing.

The architecture used in this model is not competitive with the current state-of-the-art sentence-level classifier [1], which achieves an accuracy of 88.6% when used for a speaker-independent test. However, it is considered likely that including the remaining speakers in the GRID dataset would improve accuracy further.

VII. CONCLUSION

The use of delta features in preprocessing shows benefits in both accuracy and training time for both speaker-dependent and speaker-independent tests. In speaker-dependent tests, the use of delta features increased the accuracy of the model by 1.63% and reduced training time by a factor of 2.5. Further, in speaker-independent tests, the use of delta features increased accuracy by 17.78% and reduced training time by a factor of 1.8.

The accuracy of the model was below the state-of-the-art for both the speaker-dependent and speaker-independent tests; however, the concept of delta features is independent of the model and could be used as a preprocessing step in other work to improve results. The concept of delta features can be extended to the delta between raw image inputs, as well as other input types. It is recommended that the use of delta features be investigated in any work where change in features, or movement, is considered to be important for classification.

VIII. Future Work

The VGG-Face model was trained for faces and not just the mouth region [3]. As such, some of the features of the face might not have provided useful context for the LSTM network and could have reduced the accuracy of the model. The VGG-Face model could potentially be more effective if output from a different layer, that had a higher emphasis on the mouth region, were used instead.

In order to determine which layer would be best for lip recognition, one could prepare two sets of the same input face images, but one set has the mouth removed (setting the value of the mouth region pixels to zero). The two inputs could then be passed through the VGG-Face network, generating two sets of feature vectors for each layer (with and without the mouth). For each layer, a distance metric could be used to evaluate the features with the mouth against the features without the mouth. The layer that recognizes the mouth most effectively would be the layer that yields the highest distance metric between its two sets of feature vectors. Some possible distance metrics include the average Euclidean distance between each set of corresponding feature vectors or the variance of the feature vectors for each layer. One could also visualize the difference between the sets of feature vectors by creating a t-SNE visualization, such that the distance could be visualized on a two-dimensional plane [12].

Aside from determining the best layer of the VGG-Face model to use, a CNN trained just to classify the mouth region could provide improvements. Such a model would provide feature data on the part of the face that offers the most context for lip reading. However, similar to the VGG-Face project, a very large dataset and considerable computational resources would be required to train such a model [3].

A further improvement to the work would be to incorporate sentence-level classification, as in [1]. This would allow the model to learn about the speakers' transitions between words, in addition to each word individually. The additional context of sentence-level classification would provide robustness against differences between the accents of speakers that may slur words together in different ways. Sentence-level classification can be done with temporal classifiers, such as the CTC used in [1], which represents the outputs as a probability distribution over all of the possible labeled sentences. CTC also eliminates both the need to pre-segment video frames into sets representing individual words, and the need to post-process network outputs (words) into labeled sentences [13]. Using CTC would greatly decrease the complexity of pre-processing in this work.

References

[1] Y. M. Assael, B. Shillingford, S. Whiteson, and N. de Freitas. "Lipnet: End-To-End Sentence-Level Lipreading", arXiv preprint arXiv:1611.01599, 2016.

[2] M. Cooke, J. Barker, S. Cunningham, and X. Shao. "An audio-visual corpus for speech perception and automatic speech recognition," The Journal of the Acoustical Society of America 120(5):2421–2424, 2006.

[3] O. M. Parkhi et al. "Deep face recognition," Proceedings of the British Machine Vision, 2015.

[4] M. Wand, J. Koutnik, and J. Schmidhuber. "Lipreading With Long Short-Term Memory," IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 6115–6119, 2016.

[5] E. D. Petajan. "Automatic Lipreading to Enhance Speech Recognition (Speech Reading)," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 1984.

[6] J. Goldschen, O. N. Garcia, and E. D. Petajan. "Continuous automatic speech recognition by lipreading," Motion-Based recognition, pp. 321–343. Springer, 1997.

[7] S. Gergen et al. "Dynamic Stream Weighting for Turbo-Decoding-Based Audiovisual ASR," Interspeech, pp. 2135–2139, 2016.

[8] A. Shaikh, D. Kumar, and W. Yau. "Lip Reading using Optical Flow and Support Vector Machines," IEEE International Congress on Image and Signal Processing, 1:327–330, 2010.

[9] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. "Distributed Representations of Words and Phrases and their Compositionality," Advances in Neural Information Processing Systems 26, pages 3111–3119, 2013.

[10] S. Hochreiter and J. Schmidhuber. "Long short-term memory," Neural computation, 9(8):1735–1780, 1997.

[11] D. Kingma and J. Ba. "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.

[12] L. van der Maaten and G. E. Hinton. "Visualizing Data using t-SNE," Journal of Machine Learning Research, vol. 9, pp. 2579–2605, 2008.

[13] A. Graves, S. Fernandez, F. Gomez, and J. Schmidhuber. "Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks," ICML, 2006.

Appendix A: Sample Source Code

All source code[4] was written in Python. The following code is an excerpt from the script used to load and preprocess data.

```
# Data parameters.
K = 5
SPEAKERS = [1]
SHUFFLE = True
USE_DELTA_FRAMES = True

# Model parameters.
EPOCHS = 50
BATCH_SIZE = 32
ACTIVATION = 'tanh'
LOSS = 'mean_squared_error'
OPTIMIZER = 'adam'

VERBOSE = True

# Load the data.
x, y, kmasks, vocab = load_data(k=K, speakers=SPEAKERS, shuffle=SHUFFLE,
                                use_delta_frames=USE_DELTA_FRAMES)
accuracies = []
epochs = []

for fold in xrange(1,3):
    print('Fold: ', fold)

    # Split data based on current fold.
    train, test = split_train_test(x, y, kmasks[fold])
    x_train, y_train = train
    x_test, y_test = test

    # Build the LSTM model.
    model = Sequential()
    model.add(LSTM(128, input_shape=x_train[0].shape, return_sequences=True))
    model.add(LSTM(128))
    model.add(Dense(len(vocab), activation=ACTIVATION))
    model.compile(loss=LOSS, optimizer=OPTIMIZER, metrics=['accuracy'])

    # Train the model.
    accs = []
    for _ in xrange(EPOCHS):
        model.fit(x_train, y_train, batch_size=BATCH_SIZE, epochs=1,
                  verbose=VERBOSE, validation_data=(x_test, y_test))
        _, acc = model.evaluate(x_test, y_test, batch_size=BATCH_SIZE,
                                verbose=VERBOSE)
        accs.append(acc)

    # Find epoch with best accuracy.
    best_acc = 0
    best_epoch = 0
    for epoch, acc in enumerate(accs):
        if acc > best_acc:
            best_acc = acc
            best_epoch = epoch

    print('Best accuracy: {} at epoch {}.'.format(best_acc, best_epoch))
    accuracies.append(best_acc)
    epochs.append(best_epoch)
```

[4] Complete source code is publicly available at *https://github.com/adamheins/read-my-lips*.

The next code excerpt is part of the pre-processing pipeline to load and manipulate input data.

```
def load_data(k=5, speakers=[], shuffle=False, use_delta_frames=True):
    ''' Load facial feature data from disk. '''

    # Select data files to load. Loads data from speakers specified, or takes
    # all data is no speakers are specified.
    if len(speakers) == 0:
        data_glob = os.path.join(FEATURE_DIRECTORY_PATH, '*', '*.npy')
        data_files = glob.glob(data_glob)
    else:
        data_files = []
        for speaker in speakers:
            speaker = 's' + str(speaker)
            data_glob = os.path.join(FEATURE_DIRECTORY_PATH, speaker, '*.npy')
            data_files.extend(glob.glob(data_glob))

    # Load the data.
    x, y, empty_file_count = load_data_files(data_files, shuffle)
    print('Skipped {} empty data files.'.format(empty_file_count))

    # Build mapping of vocabulary to integers, and remap output to it.
    vocab, y = build_vocab(y)

    # Calculate the maximum number of frames any word has.
    word_max_frames = max_frames(x)

    # If we're using delta frames, we need to keep an extra frame around to
    # produce the delta.
    if use_delta_frames:
        effective_num_frames = NUM_FRAMES + 1
    else:
        effective_num_frames = NUM_FRAMES

    for i, f in enumerate(x):
        last = np.array(f[-1].reshape(1, NUM_FACIAL_FEATURES))

        # Add padding with duplicates of last frame.
        for _ in xrange(f.shape[0], effective_num_frames):
            x[i] = np.concatenate((x[i], last), axis=0)

        x[i] = condense_frames(x[i], effective_num_frames)

        if use_delta_frames:
            # Take deltas.
            for j in xrange(1, len(x[i])):
                x[i][j-1,:] = x[i][j,:] - x[i][j-1,:]

            # Remove extra frame from the end.
            x[i] = x[i][:NUM_FRAMES, ...]

        # Normalize the entire set of frames for this word.
        x[i] = normalize_word_frame(x[i])

    # Convert to numpy arrays.
    x = np.asarray(x)
    y = np.asarray(y)

    # Create masks for k test and training data sets.
    kmasks = kfold(x, y, k)

    return x, y, kmasks, vocab
```