DeepSR a Deep Learning Neural Network for Speech Reading

DeepSR

A Deep Learning Neural Network for Speech Reading

Basuki Winoto

A Thesis in the Field of Software Engineering

for the Degree of Master of Liberal Arts in Extension Studies

Harvard University

May 2018

Acknowledgement

I would like to thank my beautiful wife, Dika Wijayanti Hapsari, for letting me spend so many hours and nights working through this challenging research.

I would also like to thank my research advisor, Dr. Peter Vaughan Henstock, for the guidance in completing the research.

I would also like to thank Dr. Jeff Parker, Dr. Sylvain Jaume, and Nada El-Newahy for the endless support.

To my beloved daughter, Gwyneth Larasati Ayuko Winoto, who is in her learning phase: Keep learning and never let the machine outlearn us human.

Abstract

Speech reading or lip reading is the understanding of spoken language while watching the speaker. It is a natural extension to increase comprehension when hearing becomes challenging. One attentively observes the speaker's mouth movement as it forms the word. Speech reading is a skill, in addition to hand sign language, that helps with comprehension. It is an interesting skill yet difficult to learn.

This thesis describes the development of a deep learning model that translates mouth movement into words. The model stacks a Convolutional Neural Network (CNN) and Long Short Term Memory (LSTM) for learning the sequence of mouth images.

The model is trained with a dataset from the GRID corpus. This corpus contains video recordings of 34 speakers with each speaks 1000 sentences in English. The trained model is capable of predicting words from the mouth video frames with a word accuracy of 52.25%, far exceeding human accuracy of 14.47%.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

A person could be hearing challenged due to a physical audio sensory problem or due to exposure to a noisy environment. People who are hard of hearing often seek alternative communication methods. One such method to overcome the challenge is speech reading. It is a natural extension to increase comprehension when hearing becomes challenging. Bunger [1] defined speech reading as the understanding of spoken language while attentively watching the speaker, with or without help of hearing.

Speech reading is considered as a trainable skill, that one can acquire through practice. One of the well-documented speech reading training approaches is the Jena Method [1]. This method trains a person by observing the mouth shapes and movement as the speaker is forming a word. The observer draws comparisons between an audible speech and an inaudible speech. The observer starts with watching and listening to a speaker uttering one sentence. After that, the speaker will step inside a soundproof glass room and say the same sentence. The observer will try to recognize the words by watching the speaker's mouth without hearing the voice. This training is done repeatedly on a particular word until the observer is able to recognize the word without hearing the sound. Hilder et. al. [26] reported human participants achieved an average of 14.47% accuracy on a monosyllabic words. The accuracy improved to an average of 18.42% with one hour training.

This thesis stands on the premise that speech reading can be learned through training. The hypothesis is that we can develop a deep learning model that will recognize a video clip of mouth movement and determine the spoken word. This thesis selects deep learning as the approach to solve the problem because it has the capability to extract the features from the dataset during the training. Deep learning is capable of discovering intricate structures in high dimensional data [2]. This attribute, in particular, is required to attack problem with no clear set of features such as in speech reading. Recent research results in the area of computer vision and machine learning suggest that the approach is plausible.

This thesis is an attempt to build a model for solving the speech reading problem with a deep learning technique. Chapter 2 describes the background of the problem in speech reading, the approach taken in the prior research, and the approach used to solve the problem. Chapter 3 reviews the prior research that lays the foundation to this thesis. Chapter 4 presents an overview of the components, technologies and tools to build the solution. Chapter 5 presents an overview and design of the model, dataset and testing criteria. Chapter 6 presents the model implementation, the training process, the testing results and the model tuning. Chapter 7 summarizes the results and findings along with the future work.

Chapter 2 Background

Speech reading, in the context of this thesis, is defined as observing a speaker's mouth movement and determining the spoken word. It is an algorithm that takes video frames as input and translates them into a word as the output. The video frames capture the speaker's mouth movement. The translated word is the spoken word from the speaker.

One approach to accomplish this task is to use a set of rules to determine the correct translation of the video frames. Hassanat tried to create a set of rules based on the lips width and height [9]. The research suggested the ratio of mouth's inner width and height could be used to determine the spoken word. The research utilized a K-Nearest Neighbor [9] technique to group the rules based on the lips width and height dataset. With a limited set of vocabulary, they achieved an accuracy of 33%. The result showed a significant improvement when localized to the dataset of the same speaker only. It is clear that manually building the dictionary that is required in such set of rules is impractical. One would need to measure the ratio in each frame for every single word with the variation of speaker's lips shape and contour. The result leaves room for improvement. It is not generalizing well across speakers and the rules are proving to be less accurate. However, this research set the cornerstone for further research.

The speech reading problem poses three challenges, which we will need to address:

1. There is no clear set of rules that can determine the spoken word from a video frames.

2. There is no clear set of features that can distinguish one video frames from another.
3. There is no established dictionary that can translate a sequence of video frames to a word.

Given these challenges, we propose to utilize deep learning to solve this speech reading problem. Deep learning can extract features from the dataset during training, and develop set of rules based on the dataset to automate the dictionary building process which otherwise would have to be done manually. Deep learning can automate the process with an extensive training dataset.

In this thesis, we describe a deep learning model using a combination of Convolution Neural Network (CNN) and Recurrent Neural Networks (RNN). The CNN component is required to extract the visual feature of each frame in the video clip. It identifies the mouth shape using convolutional combined with a max pooling operation on the image. The RNN component is required to extract the temporal feature of the frame sequence. The combination of a CNN and an RNN formed a model that was trained with a labeled set of mouth video clips. The resulting model is capable of translating the mouth clip into words.

Chapter 3 Review of Prior Work

A number of past research papers have created the foundation for this thesis. This chapter reviews the notable research tackling this problem.

A Spatial Temporal technique of Viseme Extraction: Application in Speech Recognition

This paper confirms that visemes can be differentiated using the spatial temporal features. They extracted the lips feature using edge detection. The main features are the vertical opening (DV) and horizontal opening (DH) of the lips. They developed a model that differentiates visemes based on the image features. The model was tested on an in-house dataset and confirmed the difference between visemes [10].

From this paper, we borrowed the idea of focusing on the lips region showing vertical and horizontal opening to predict the word.

Method: Feature extraction

Approach: Lips feature extraction

Dataset: In-house dataset

Result: Confirms different visemes have different features

Scope: Closed domain problem

Visual Speech Recognition

The Visual Speech Recognition (VSR) model takes the approach of predicting the viseme from lips features [9]. Viseme is the visual equivalence to phoneme. This model uses K-Nearest Neighbor technique to predict the viseme from a lips movement. The lips feature was extracted with edge detection using a discrete wavelet transformation [25]. The paper explained that the point of interest is mainly the inner lips width and height, the appearance of tongue and the appearance of teeth.

The classification process employs dynamic time warping to compensate for the length difference, and K-Nearest Neighbor classifier to predict the word.
This model is trained against an in-house dataset with female and male speakers. The evaluation results show that the model achieved 76.38% word recognition rate if the model is trained with the same speaker. The result declines significantly to 33% word recognition rate when the training data is mixed with different speakers.

From this paper, we borrowed the idea of focusing to the inner lips width and height, and the appearance of tongue and teeth.

Method: K-Nearest Neighbor

Approach: Lips feature extraction and K-Nearest Neighbor

Dataset: In-house dataset

Result: 76.38% (same speaker), 33% (mixed speakers)

Scope: Closed domain problem

LipNet: End-to-end Sentence-level Lipreading

LipNet claims to be the first end-to-end sentence-level deep learning model. The model gives 95.2% word accuracy in sentence-level surpassing the average human lip reader [5]. The LipNet architecture is composed of a Spatio-Temporal Convolution

Neural Network (STCNN) and Recurrent Neural Network (RNN) with Connectionist Temporal Classification (CTC) algorithm to approximate the word segmentation [23]. The paper showed that combining STCNN and RNN is effective in extracting the spatio-temporal features. The CTC algorithm produces approximation of maximum-probability word segments in a sentence. The model was trained using the GRID corpus [24], which currently is the largest public sentence-level dataset available. The performance was evaluated by measuring the Character Error Rate (CER) and Word Error Rate (WER) for both unseen and overlapped speakers. The model gives a CER of 6.4% and WER of 11.4% on unseen speakers, and a CER of 1.9% and WER of 4.8% on speakers that were included in the training set. The high accuracy performance is attributed to the complexity of the network topology. It is capable of extracting the information in high granularity. The input layer takes 3 seconds colored mouth clips with total dimension of 75 frames x 100 pixels wide x 50 pixels high x 3 channels per clip. The model was trained on a machine with 8 high end GPUs (28,672 cores), which enable training for a complex model. The researchers suggested the overall performance could only be improved with a larger dataset.

From this paper, we took the same logical approach developing a model from the spatial and temporal features of the dataset.

Method: Deep learning

Approach: Spatio-temporal visual features and sequences model

Dataset: GRID corpus

Result: 95.2% accuracy (overlapped speakers), 89.6% (unseen speakers)

Scope: Closed domain problem

Lip Reading Sentence In the Wild

The Lip Reading in the Wild (LRW) model combines visual and audio cues to predict the word [8]. In making the word prediction, the model uses the notion of the speaker's lips and the speaker's voice. The LRW architecture combines the two channels of visual and audio cues. The visual model is a convolutional network that extracts the features and passes them as input to a LSTM network. The audio model is a sequence-by-sequence LSTM network. It was trained using Lip Reading Sentence (LRS) corpus, which was produced from BBC broadcasting. The dataset contains facial expressions from various speakers, hence the term "in the wild". This model aimed at tackling the speech reading problem in an open domain.

The LRW was evaluated with watch-only, watch-listen, and watch-listen in noisy environment. The watch-only model achieved 53.2% Word Error Rate (WER). In other words, it can predict the word with 46.8% word accuracy from reading the lips. The accuracy improves with a watch-listen combination. On recordings with no noise, the watch-listen model achieved 22.8% WER, or 77.2% word accuracy. These results drop to 35.1% WER, or 64.9% word accuracy, when the dataset is added with a 10db Signal to Noise Ratio (SNR) to simulate a noisy environment. As comparisons, a professional lip reader achieved 73.8% WER, or 26.2% word accuracy, on a subset of data.

From this paper, we borrowed the data pre-processing technique that localizes the problem to word segments from a full sentence.

Method: Deep learning

Approach: Watch, listen, attend and spell

Dataset: LRS corpus

Result: 46.8% accuracy (watch only), 77.2% (watch and listen)

Scope: Open domain problem

Chapter 4 Technology Choice

Neural Network

A neural network models a calculation with the use of layered perceptrons. In a simple model, a perceptron is a node that takes inputs, applies weights to each input with a bias value and produces an output based on the activation function. The output is passed through an activation function to introduce non-linearity in the model. Multiple perceptrons can be stacked in a layer to model a rather complex equation. A layer of multiple perceptrons takes a number of inputs, which will be directed to each node. Each of the nodes will apply weights, bias and activation function, which in turn will produce outputs. Neural network is a multiple layer of perceptrons. Neural networks are particularly good at modeling complex calculations. A model built in a neural network can be trained with a feed-forward and back-propagation computation to optimize weights for each node in the model. During the training process, the model learns the best weights for each node to achieve the defined goal. The goal could be defined as achieving highest accuracy score or minimizing the loss.

Convolutional Neural Network

Convolutional Neural Network or CNN is a neural network architecture designed specifically for solving the image recognition problem. It utilizes matrix convolution and max pooling techniques to detect patterns in an image [15]. Our work takes the advantage of the CNN in extracting the mouth opening pattern in each frame in the lips video sequence.

Recurrent Neural Networks

Recurrent Neural Network or RNN is a neural networks architecture that adds an internal memory in the node. The memory allows RNN to pass a value from a node to another node in the same layer, which can represents a system state. It is particularly useful for time series computation.

Long Short Term Memory

Long Short Term Memory or LSTM is a type of recurrent neural networks that is specifically designed for sequential data [16]. It is widely used in speech recognition and natural language processing. It considers the sequence when making an inference. LSTM has the capability of holding longer-term memory. It prevents the vanishing value problem, which occurs in RNN [16].

Tensorflow

Tensorflow [17] is an open source software language for building mathematical models and can be used for implementing neural networks. It was originally developed by Google's engineers for machine learning research. The latest iteration, Tensorflow 1.5.0, includes the Keras library, a wrapper library for developing neural network model [17].

Scikit Learn

Scikit learn is an open source software library in Python that has set of functionalities for data mining and analysis [18]. It is built on NumPy, SciPy and matplotlib.

Pandas

Pandas are open source software library for Python for data wrangling [19]. They provide functionalities for handling datasets in a tabular representation, which makes them intuitive to utilize when processing the dataset.

MessagePack

MessagePack or msgpack is a software library for Python for serializing data to a binary file format [20]. It helps with saving the data from memory to file.

Ffmpeg

Ffmpeg is an open source software package for audio and video processing [21]. The current research uses ffmpeg for cropping and trimming the video clips.

Dlib

Dlib is an open source C++ software library for machine learning and data analysis [22]. Our work uses the dlib face detection module for detecting the face and lips region in the images.

Chapter 5 Model Design

The contribution of this thesis is a deep learning model called DeepSR for speech reading. From a high level perspective, the DeepSR model stacks a Convolutional Neural Network (CNN) and a Long Short Term Memory (LSTM). The CNN's role is extracting the image features from each frame of the mouth clip. The LSTM's role is learning the sequence of the features, which were the output of the CNN.

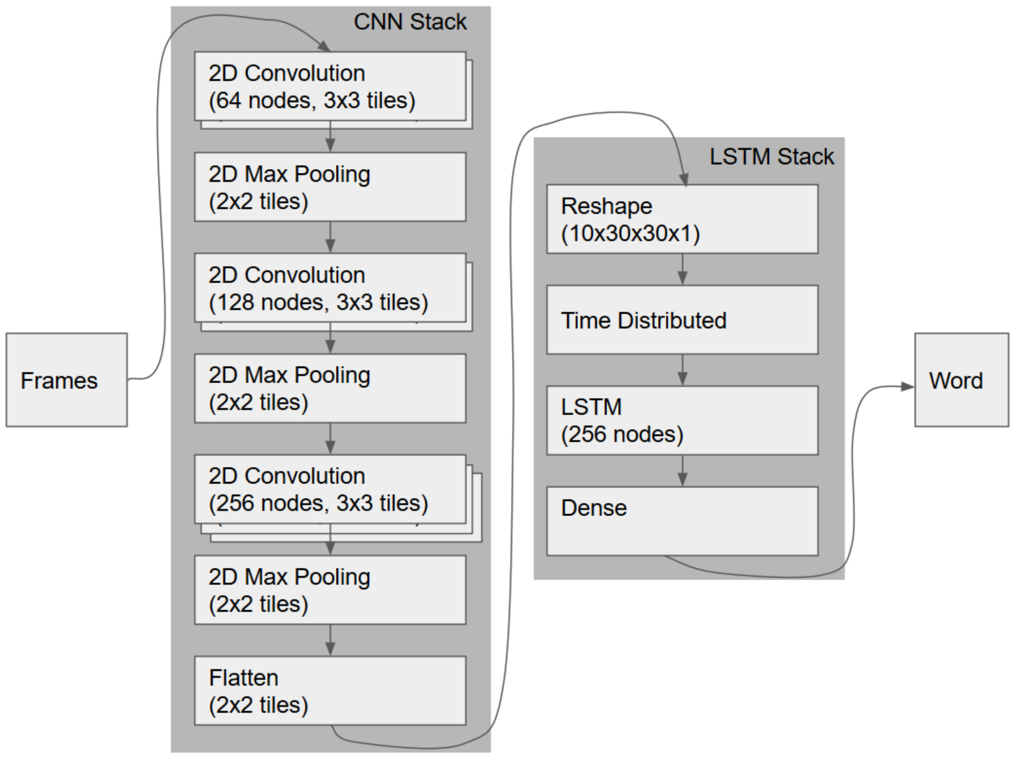


Figure 1. DeepSR architecture stacks CNN and LSTM.

The CNN stack consists of a mix of 2D convolutional layers and 2D max pooling layers. The 2D convolutional layers take 30x30 pixel grayscale images. This layer applies a 2D convolutional operation with a 3x3 pixel tile. This layer is designed with 64 hidden nodes and is activated with ReLU activation function. The 2D max pooling layer applies pooling operation on the convolutional output using a 2x2 pixels tile.

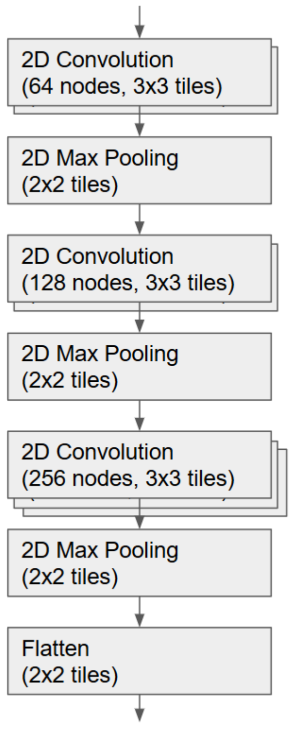


Figure 2. CNN stack uses 2D convolution, 2D max pooling and flatten operations.

Figure 2 depicts the detailed design of the CNN stack. The input is fed to two 2D convolution layers with 64 hidden nodes and 3x3 convolution tiles. These layers are utilizing a rectified linear unit (ReLU) activation function. Next in the layer is a 2D max pooling with 2x2 tiles. This combination of 2D convolution layers and max pooling is repeated three times with an expanding number of hidden nodes. The stack is capped with a layer that flatten the output, which then fed to the LSTM stack.

The LSTM stack takes the output from the CNN stack. This stack learns the series of CNN stack outputs as a sequence. The LSTM layer is designed with 256 hidden nodes and is activated with a softmax activation function (Figure 3). The output of the LSTM stack is connected with a regular dense layer to classify the result.

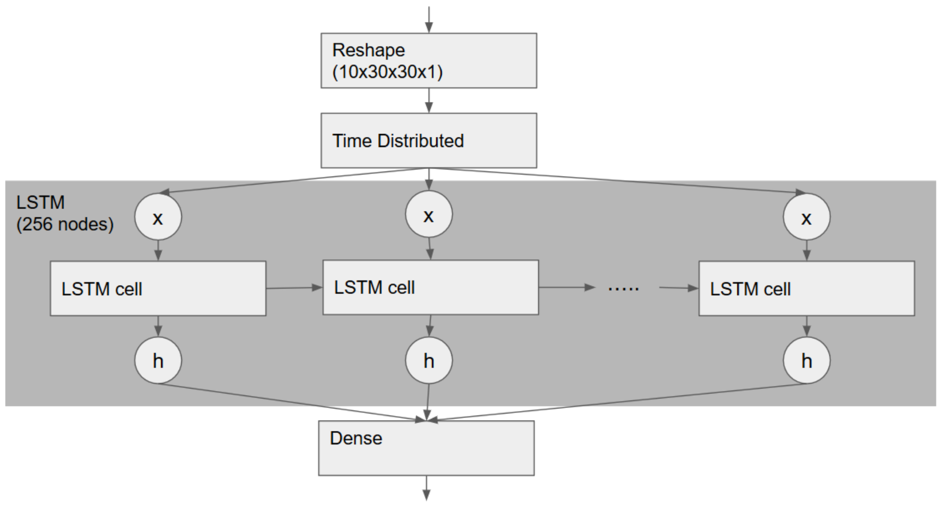


Figure 3. LSTM cells pass value to the subsequent cells.

Another consideration for the model is modifying the LSTM layer to a Bidirectional LSTM layer. Such a layer can be created from two separate LSTM layers where each layer reads the sequence in forward and backward direction.

The input data for this model is designed as a 4D array. The array has the dimension of 10x30x30x1 which represents Time x Width x Height x Channel. The output is a 1D array with the length of 51 to capture the 51 defined classes in one hot encoding. The classes are the distinct spoken words in the dataset. The words in this case could be a word, a number or an alphabetical character.

The model is trained with 90% of the dataset and validated with 5% of the dataset. The remaining 5% dataset is held out as the blind test dataset. During the implementation phase, it was found that loading the entire dataset in memory was impractical due to hardware limitation. Therefore, the dataset was redesigned with consideration to read in subsets. The testing is designed to measure three criteria: accuracy score, the confusion matrix and the ROC curve. These three criteria will be applied to the blind test dataset.

Chapter 6 Model & Results

The implementation follows five steps:

- Data preparation
- Model development
- Model training, validation and tuning
- Blind testing
- Results interpretation and further tuning

Data preparation

The data used for this research comes from the GRID corpus, which was chosen because it has a large corpus and a variety of speakers. The corpus is in full color video format at 720x526 resolution and 25 fps, with 34 different speakers, each recording 1000 sentences with 51 different words in total. The corpus is available for public download at their website [24].

The first step in data preparation is converting the video to grayscale. The conversion reduces the complexity by reducing the number of color channels from three (RGB) to one (grayscale). This makes the dataset smaller without sacrificing the quality, because we are only interested in the mouth shape.

After the conversion to grayscale, the video is segmented into word segments. The original corpus provided a metadata file, which contains the exact time frame of each word in each video clip. The video segmentation is based on the information provided in the metadata file.

The next step is cropping the video to the mouth region. The cropping is performed with facial detection using the dlib library for identifying the mouth region of the speaker face. The cropping step is aligned with the model design, in that only the mouth shape and movement are important to identifying the spoken word. The cropped video is then rescaled to a uniform resolution of 60 pixels width by 30 pixels height. The video scaling step ensures each of the data points have the same resolution

Next, the scaled video is split to half mouth: left and right. This step reduces the data size to half without losing any of the important information, due to the symmetry that is the left and right sides of the mouth creates the equivalent shape for the same word. The resulting split video is stacked together as dataset. The model will learn the features from both left and right half the mouth.

As the last step in data preprocessing, the video frames are trimmed to 10 frames in length to reduce the data size. The length is considered to be short enough yet remains to have the important information for two syllables words. The dataset has 51 different words. Each word represents one class that is encoded with one hot encoding.

The preprocessed dataset is split into three parts with a random sampling method. The split is 90% for training data, 5% for validation test and the remaining 5% for blind test. The words were sampled from each class to ensure equal distribution to the three dataset.

In summary, the dataset has close to ~400,000 clips, with total 51 classes. Each clip has 10 frames that are 30 pixels wide x 30 pixels high with 1 channel.

Model development

The model uses the Tensorflow library with the Keras wrapper included as part of Tensorflow version 1.5. In Keras terms, the model consists of two sequential models. The first one is the convolution neural network (CNN) stack. This stack is defined as a mix of Conv2D, MaxPooling2D and Flatten objects. The Conv2D object applies the convolution operation with 3x3 kernel size, 1x1 strides, same padding, and uses ReLU activation function in the output. The MaxPooling2D object applies the max pooling operation to the data with 2x2 pool size. The Flatten object transforms the data to a one-dimensional array before passing it on to the next layer.

The second sequential model represents the recurrent neural network (RNN) stack. This stack consists of Reshape, TimeDistributed and LSTM objects. Reshape transforms the data into a time series. In this case, we define the data structure as time x width x height x channel. TimeDistributed object applies the CNN model to each element from the time series data. We applied the CNN stack to each frame input. LSTM object applies the recurrent cell operation with the 256 cell units. Finally, Dense object fully connects the output from the LSTM stack and transforms the output to match 51 target classes. Appendix A depicts the implementation using a Tensorboard graph.

In the model, we defined three hyperparameters: Bidirectional layer, Regularizer and Dropout layer for tuning the model.

The model is using categorical cross entropy as the loss parameter. The model learns with Adadelta learning algorithm, which dynamically adjust the learning rate thus accelerate the learning time [27]. A single epoch test with our model showed that the loss value declines faster than using gradient descent algorithm.

Model training, validation and tuning

The model was trained using the training dataset and validated using the validation test dataset. It was trained for 10 epochs with a batch size of 128. The training was done on an Intel core i5 machine with a single NVIDIA GTX 1050 GPU, which has 640 cores. It took close to 8 hours to complete 10 epochs per session. The total training included the following four sessions: training the baseline, training with a bidirectional layer, training with a regularizer and training with a dropout layer. The result of each of the training session is a trained model in h5 format and Tensorboard logs. The training sessions are summarized in the Table 1.

Table 1. Model training and validation results.

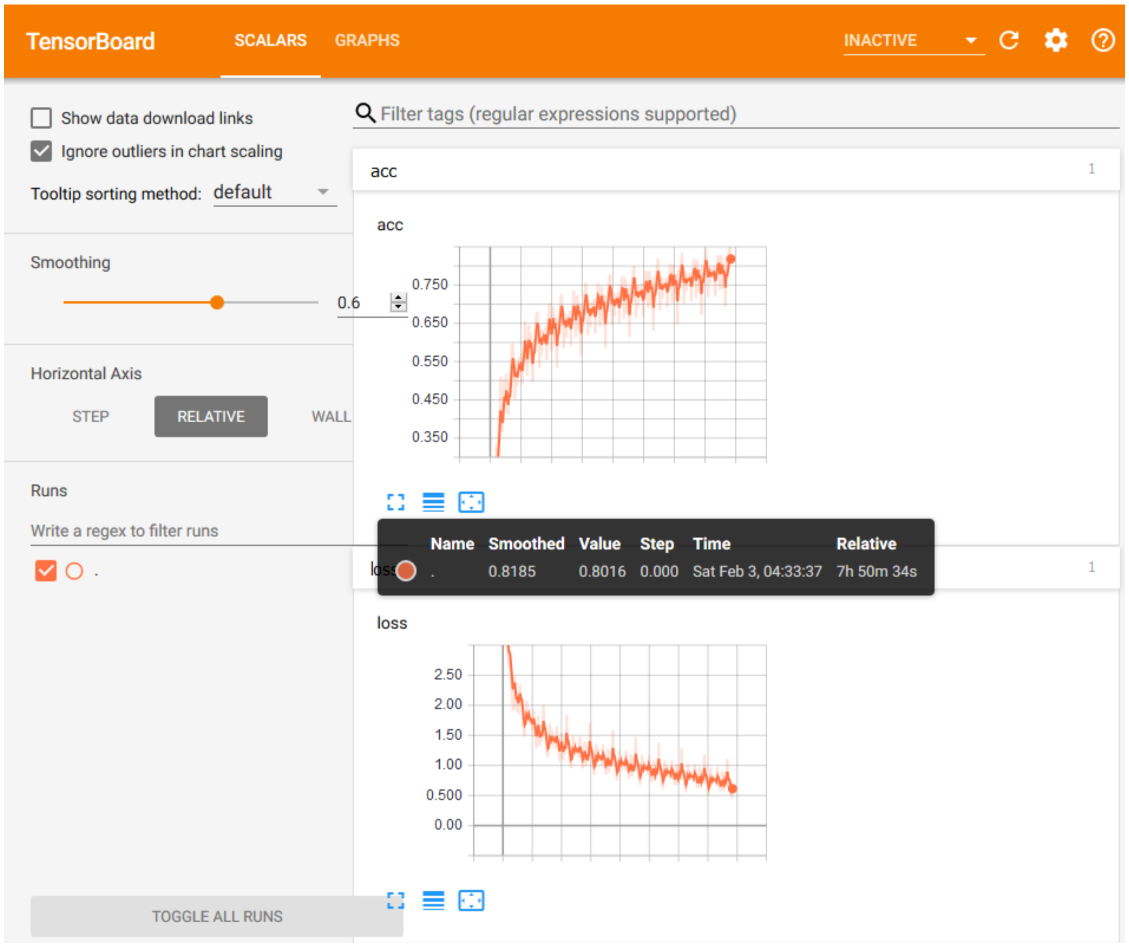
	Time	**Train Accuracy**	**Train Loss**	**Validation Accuracy**	**Validation Loss**
Baseline	7hrs 50mins	80.16%	0.6702	45.74%	2.232
Bidirectional	8hrs 10mins	80.55%	0.6539	42.67%	2.422
Regularizer	8hrs 1mins	76.71%	1.054	40.31%	2.523
Dropout	7hrs 50mins	76.83%	0.8039	45.85%	2.108

The following is a more detailed view of the training sessions as reported in the Tensorboard.

Baseline

This session trains the model with the bidirectional, regularizer and dropout options are turned off. The result of this training serves as the baseline.

Figure 4 and 5 are screen captures of the Tensorboard report on the baseline run. The training took 7 hours and 50 minutes in total to complete 10 epochs. We observed the training accuracy hit 80.16% with loss value at 0.6702 (Figure 4). The validation test showed an accuracy of 45.74% with loss value at 2.232 (Figure 5).



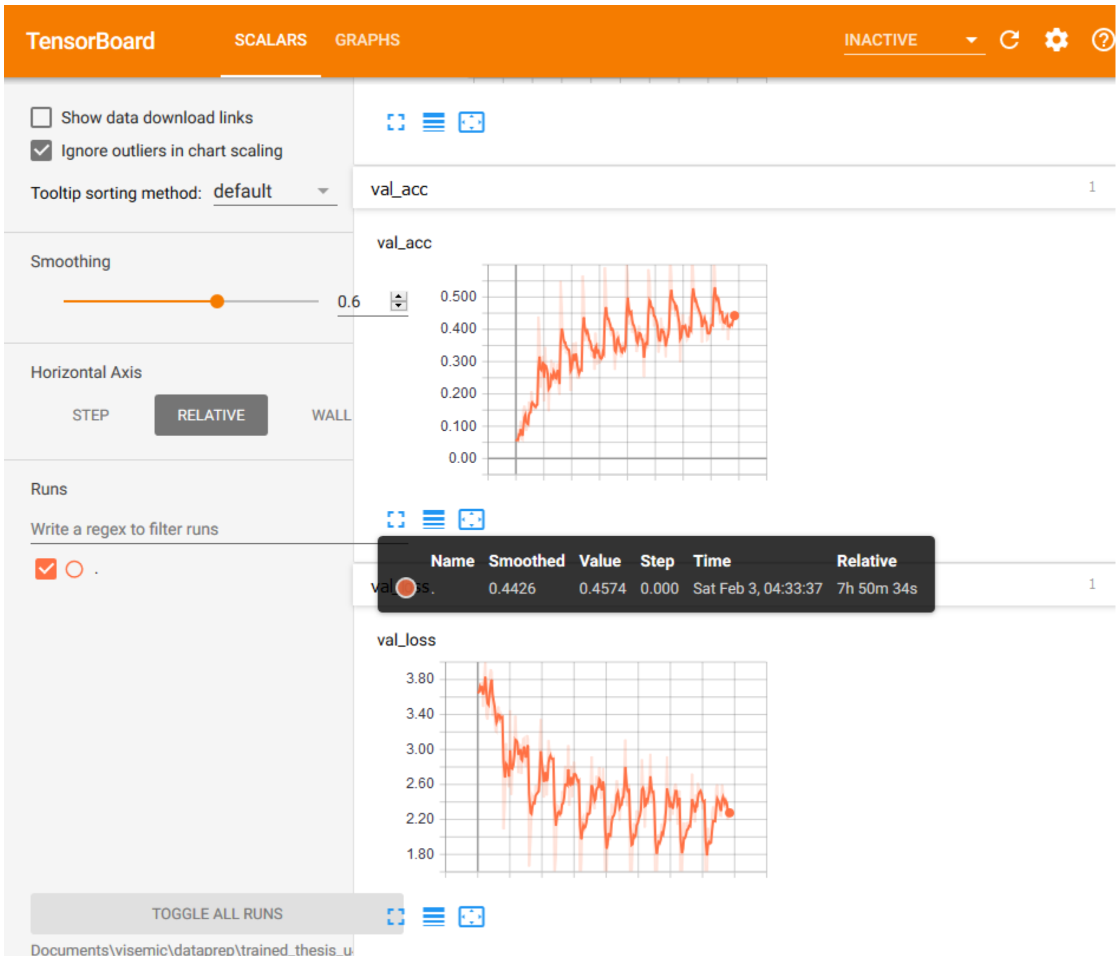
☐ Show data download links
☑ Ignore outliers in chart scaling
Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: STEP [✓ RELATIVE] WALL

Name	Smoothed	Value	Step	Time	Relative
.	0.8185	0.8016	0.000	Sat Feb 3, 04:33:37	7h 50m 34s

Figure 4. Training accuracy and loss on the baseline run.



☐ Show data download links
☑ Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing 0.6
Horizontal Axis: [✓ RELATIVE]

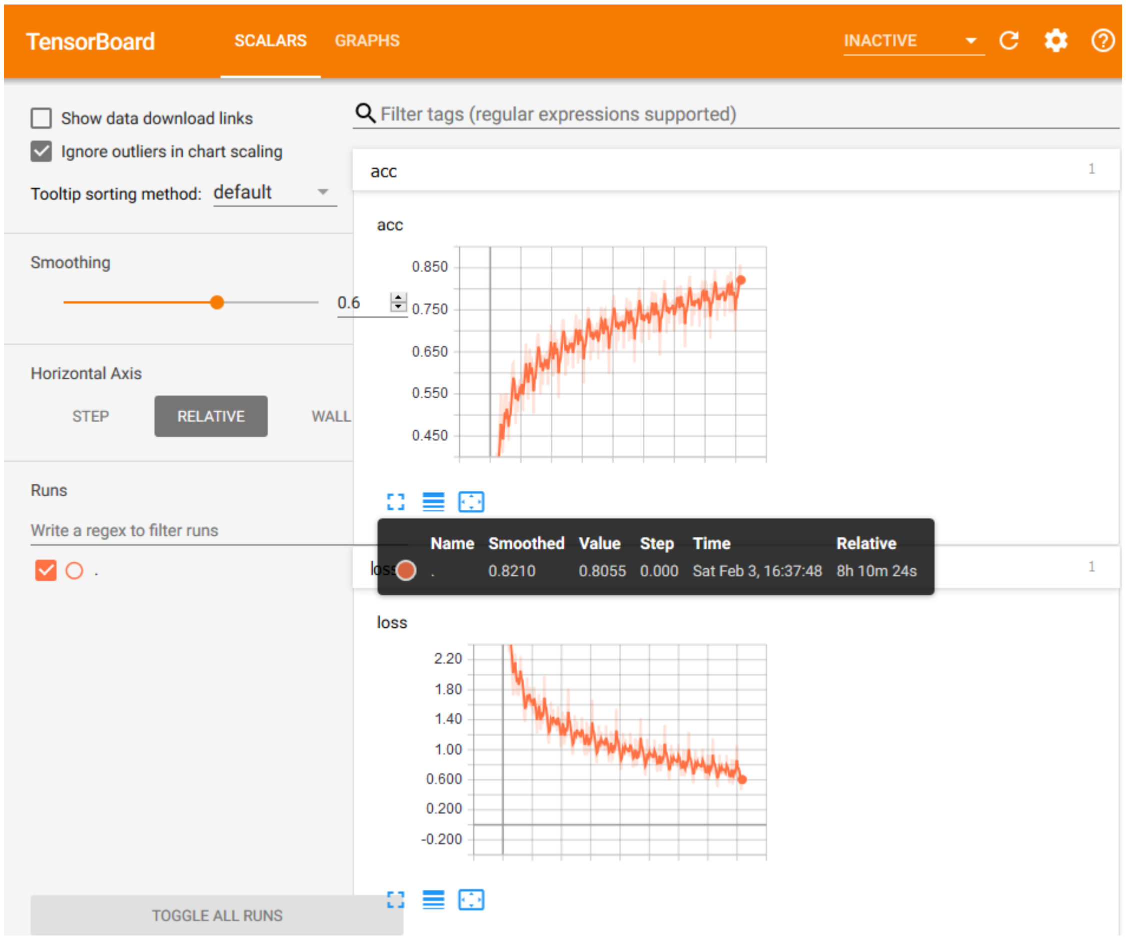
Name	Smoothed	Value	Step	Time	Relative
val ss.	0.4426	0.4574	0.000	Sat Feb 3, 04:33:37	7h 50m 34s

Documents\visemic\dataprep\trained_thesis_u

Figure 5. Validation accuracy and loss on the baseline run.

Bidirectional

This session trains the model with a bidirectional layer turned on. The training took more time and indicates a worse over fitting result than the baseline. Figure 6 and 7 are screen captures of the Tensorboard report on the model run. The training took 8 hours and 10 minutes in total to complete 10 epochs. We observed the training accuracy hit 80.55% with loss value at 0.6702 (Figure 6). The validation test showed an accuracy of 42.67% with loss value at 2.232 (Figure 7). This run, while showing a higher training accuracy score than the baseline, has a lower validation accuracy score. This indicates a worse over fitting problem with the bidirectional option turned on.



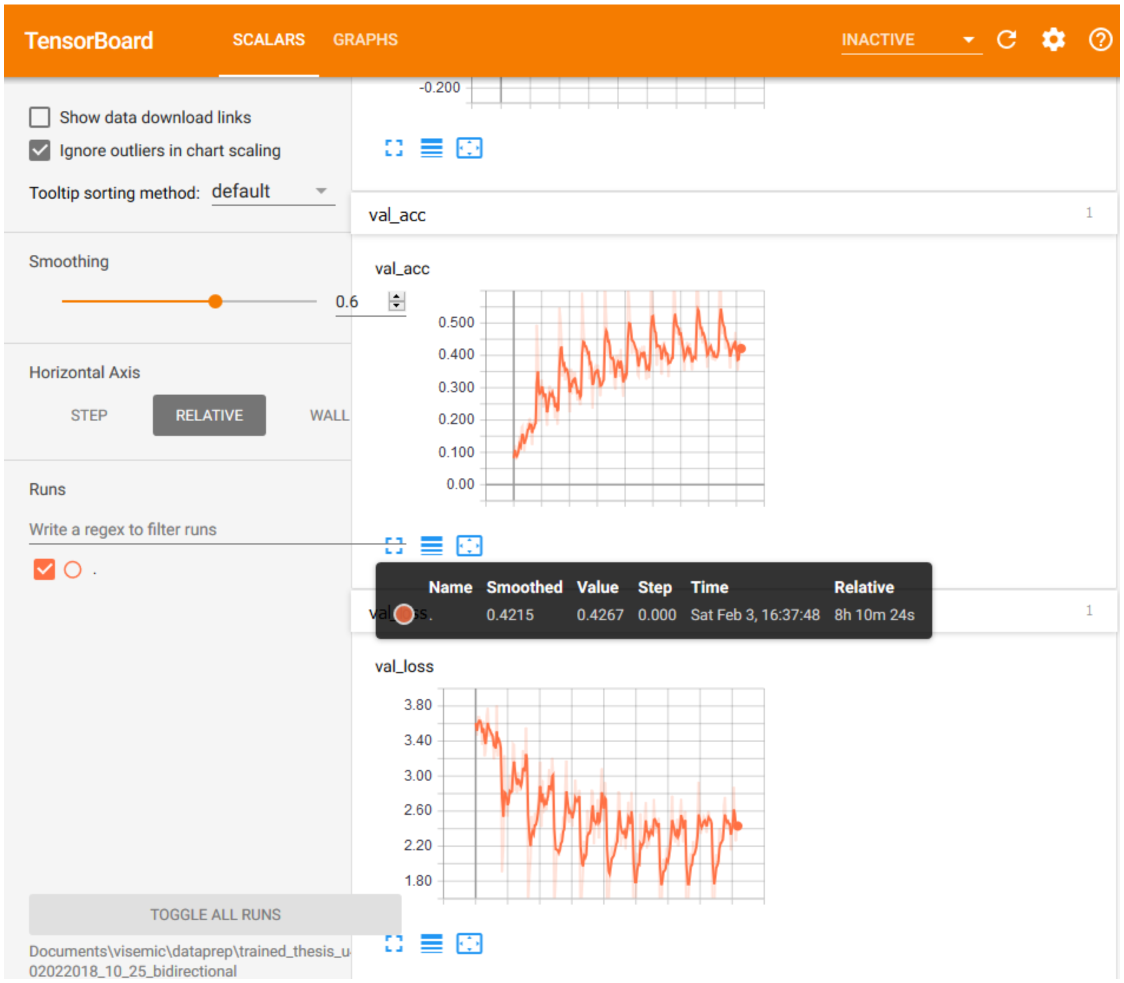
☐ Show data download links
☑ Ignore outliers in chart scaling
Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: STEP [✓ RELATIVE] WALL

Name	Smoothed	Value	Step	Time	Relative
.	0.8210	0.8055	0.000	Sat Feb 3, 16:37:48	8h 10m 24s

Figure 6. Training accuracy and loss on the model with bidirectional option on.



Show data download links
✓ Ignore outliers in chart scaling
Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: STEP [✓ RELATIVE] WALL

Runs

Write a regex to filter runs

✓ .

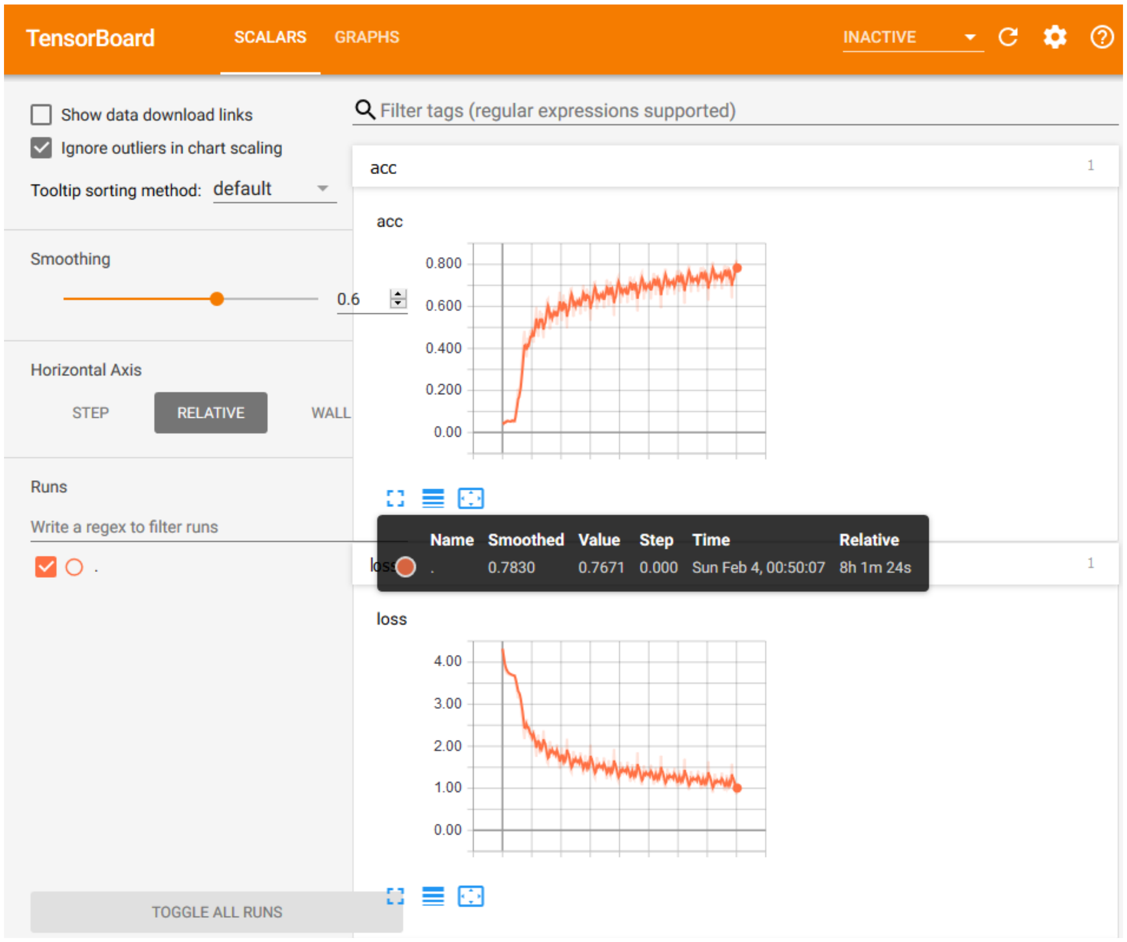
Name	Smoothed	Value	Step	Time	Relative
.	0.4215	0.4267	0.000	Sat Feb 3, 16:37:48	8h 10m 24s

Documents\visemic\dataprep\trained_thesis_u
02022018_10_25_bidirectional

Figure 7. Validation accuracy and loss on the model with bidirectional option on.

Regularizer

This session trains the baseline model with the regularizer parameter turned on. Figure 8 and 9 are screen captures of the Tensorboard report on the model run with regularizer option turned on. The training took 8 hours and 1 minute in total to complete 10 epochs. We observed the training accuracy hit 76.71% with loss value at 1.054 (Figure 8). The validation test showed an accuracy of 40.31% with loss value at 2.523 (Figure 9). This run is showing a lower training accuracy score than the baseline and a lower validation accuracy score. This indicates a less over fit model than the baseline. However the overall accuracy is also declining.



☐ Show data download links

☑ Ignore outliers in chart scaling

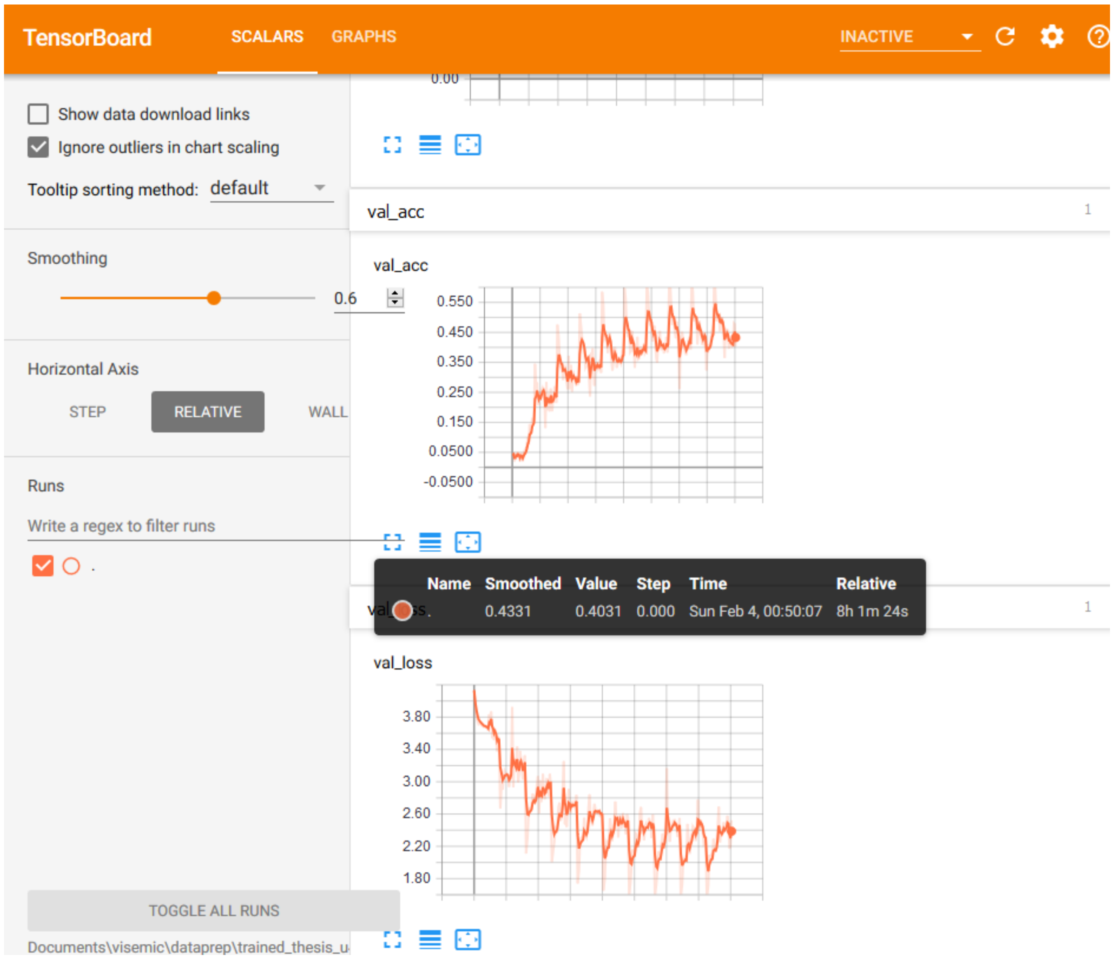
Tooltip sorting method: default

Smoothing: 0.6

Horizontal Axis: STEP [✓ RELATIVE] WALL

	Name	Smoothed	Value	Step	Time	Relative
●	.	0.7830	0.7671	0.000	Sun Feb 4, 00:50:07	8h 1m 24s

Figure 8. Training accuracy and loss on the model with regularizer option on.

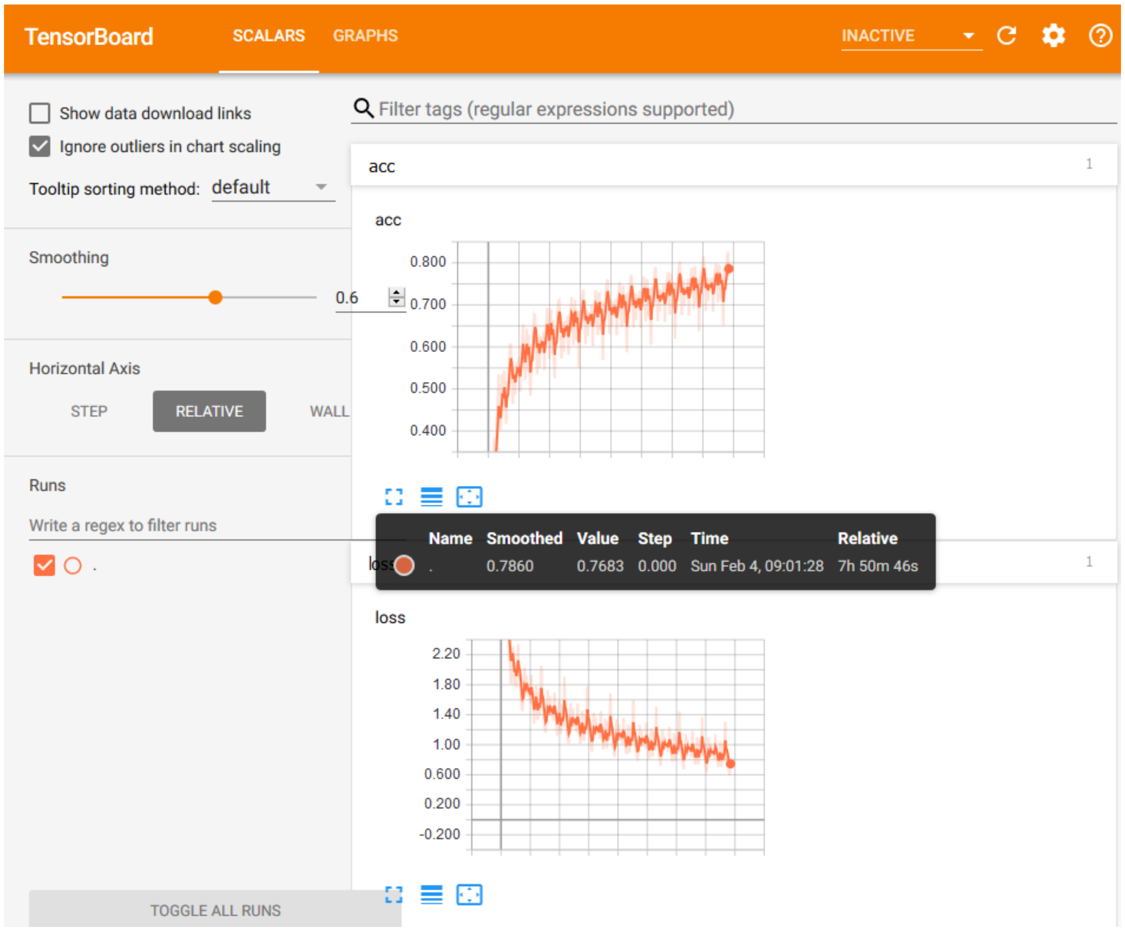

TensorBoard
SCALARS
GRAPHS
INACTIVE
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing
0.6
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
TOGGLE ALL RUNS
Documents\visemic\dataprep\trained_thesis_u
val_acc
val_loss

Name	Smoothed	Value	Step	Time	Relative
.	0.4331	0.4031	0.000	Sun Feb 4, 00:50:07	8h 1m 24s

Figure 9. Validation accuracy and loss on the model with regularizer option on.

Dropout

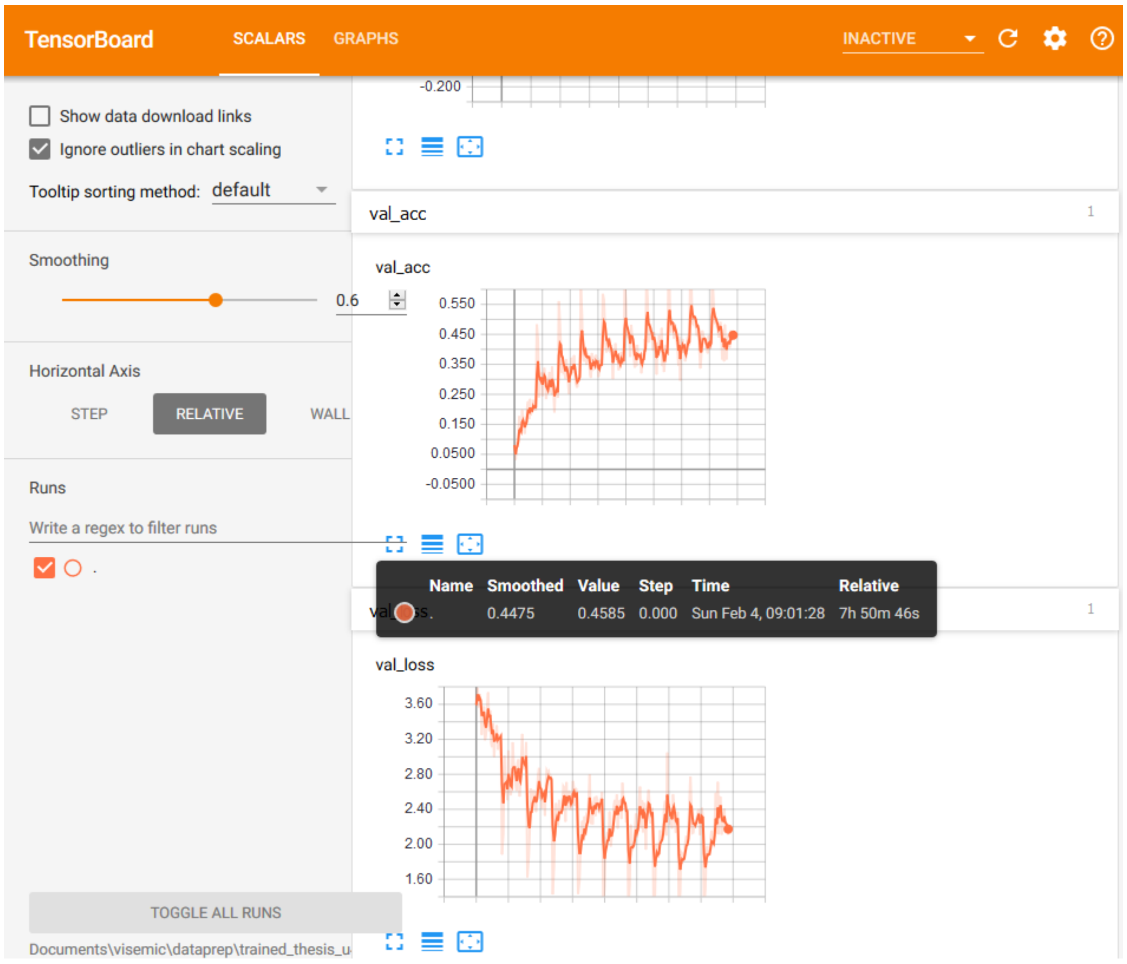
This session trains the baseline model with dropout turned on. Figure 10 and 11 are screen captures of the Tensorboard report on the model run with dropout option turned on. The training took 7 hours and 50 minutes in total to complete 10 epochs. We observed the training accuracy hit 76.83% with loss value at 0.8039 (Figure 10). The validation test showed an accuracy of 45.85% with loss value at 2.108 (Figure 11). This run shows a lower training accuracy score than the baseline and a higher validation accuracy score. This indicates a less over fit model than the baseline and also the overall accuracy improves.


TensorBoard
SCALARS
GRAPHS
INACTIVE
Filter tags (regular expressions supported)
TOGGLE ALL RUNS
acc
loss
Runs
Write a regex to filter runs

Show data download links
☑ Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing 0.6
Horizontal Axis: STEP [✓ RELATIVE] WALL

Name	Smoothed	Value	Step	Time	Relative
.	0.7860	0.7683	0.000	Sun Feb 4, 09:01:28	7h 50m 46s

Figure 10. Training accuracy and loss on the model with dropout option on.



Show data download links
✓ Ignore outliers in chart scaling
Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: RELATIVE

Runs

Write a regex to filter runs

Documents\visemic\dataprep\trained_thesis_u

Name	Smoothed	Value	Step	Time	Relative
● .	0.4475	0.4585	0.000	Sun Feb 4, 09:01:28	7h 50m 46s

Figure 11. Validation accuracy and loss on the model with dropout option on.

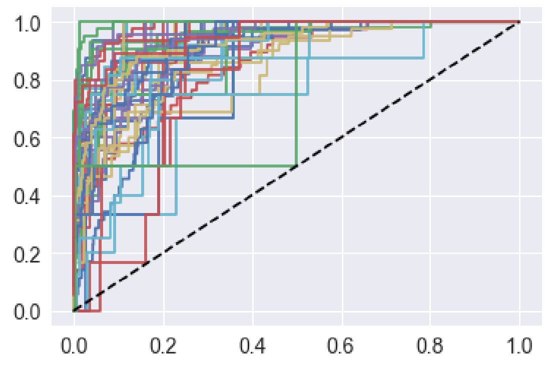
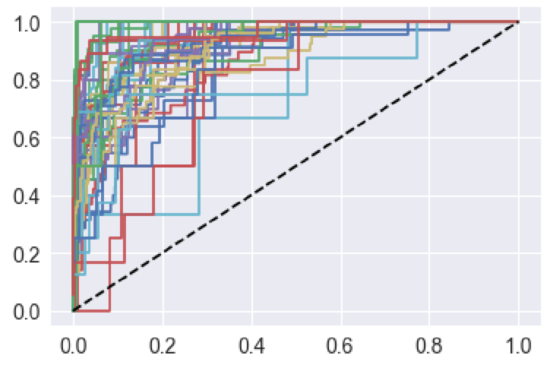
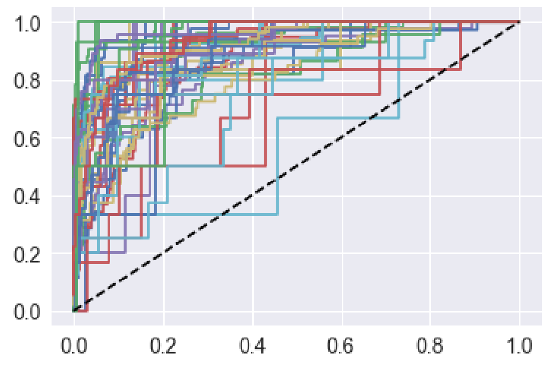
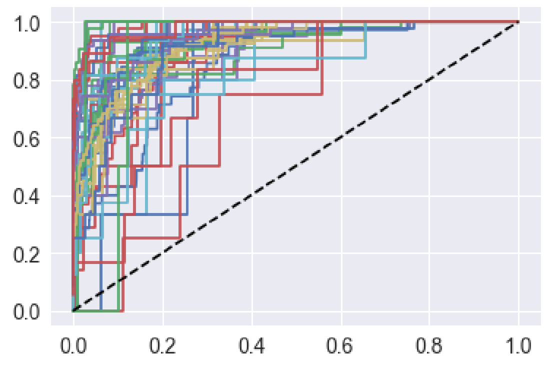
Blind test

The resulting models are tested with the blind dataset with three test criteria: accuracy score, confusion matrix and ROC curve. The accuracy score is calculated using accuracy_score function in the sklearn library. The confusion matrix is calculated using the confusion_matrix function in the sklearn library and plotted using seaborn library. The ROC curve is calculated using roc_curve function in the scikit-learn library and plotted using matplotlib library.

Table 2. Blind accuracy score comparisons.

	Validation accuracy	Blind accuracy
Baseline	45.74%	47.23%
Bidirectional	42.67%	43.85%
Regularizer	40.31%	43.75%
Dropout	45.85%	47.64%

Table 3. ROC curve comparisons.

 a. ROC curve on the baseline run.	 b. ROC curve on the model run with bidirectional option on.
 c. ROC curve on the model run with regularizer option on.	 d. ROC curve on the model run with dropout option on.

Number of nodes

The number of nodes has a direct relationship to accuracy and training time. Adding nodes to the network improves the accuracy by capturing more features from the video clips. However, it requires a longer training time because the number of parameters is increasing. Inversely, a reduced number of nodes reduce the footprint and the training time. We ran tests on the baseline model with modified number of nodes.

Table 4. Training time and accuracy comparisons over different footprint.

	Number of nodes	Number of trainable parameters	Training Time	Validation Accuracy
Baseline	1x Baseline	2,272,755	7hrs 50mins	45.74%
Half-nodes model	1/2x Baseline	572,179	3hrs 34mins	33.85%
Quarter-nodes model	1/4x Baseline	145,059	2hrs 14mins	31.28%

Table 4 shows the direct relationship between training time, accuracy and footprint. Capping the number of nodes reduce the footprint, training time and accuracy. Reducing the nodes to half of the baseline model cuts the training time in half at the cost of ~12% accuracy. The larger model has the benefit from the ability to make an approximation with more parameters, hence the higher accuracy, at the cost of larger footprint and training time.

Result interpretation and further tuning

The results indicate the model is capable of properly classifying the classes. The dropout layer and regularizer helps reduce the over fitting. Adding dropout layer and regularizer improve the model through better generalization as expected. The loss value is still declining at the last epoch. It suggests the accuracy can improve by adding with more epochs.

Based on these conclusions, a final training session was run on the baseline with regularizer and dropout. The regularizer value was set at 0.005 or half regularizer value of the previous run. The dropout value was set at 0.75 or one and a half dropout value of the previous run. The session was run in 20 epochs or and twice of the previous run. The final model gave an accuracy score of 52.25% on the blind set testing. The ROC curve and confusion matrix are shown in Figure 12 and 13.

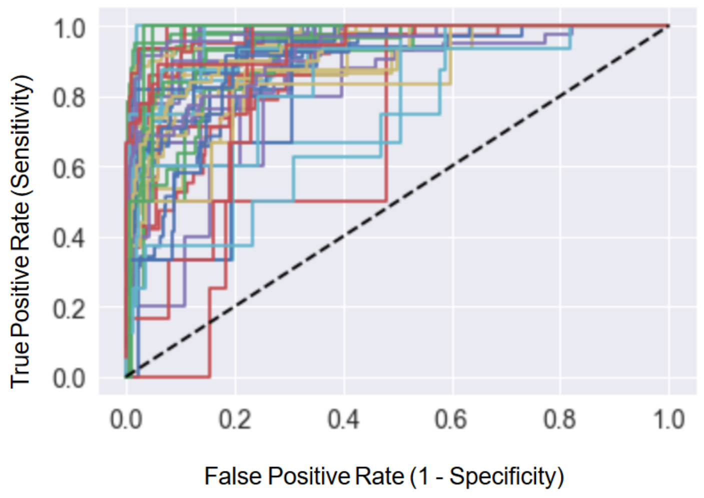


Figure 12. ROC curve on the final model run with regularizer and dropout.

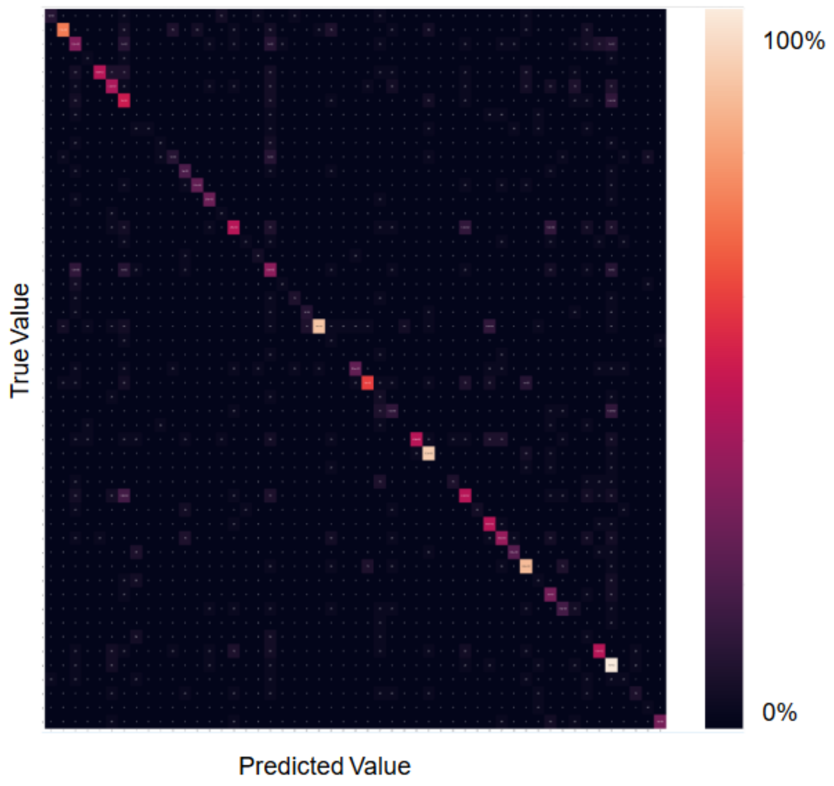


Figure 13. Confusion matrix on the final model run with regularizer and dropout.

Chapter 7 Conclusion

This thesis shows that deep learning can be used to develop an accurate model for speech reading even with modest computing power. Our deep learning network consisting of a stack of a CNN followed by an LSTM proved very effective. The CNN processed the spatial features of image data. The LSTM processed the temporal features of time series data.

Table 5. Accuracy comparisons.

	Accuracy score	**Clip resolution**	**Notes**
Human	14.47%	1920x1080 full color	Participants are non-professional.
Visual Speech Recognition (VSR)	33%	320x240 full color	Accuracy achieved on mixed speakers. In speaker dependent scenario, the accuracy improves to 76.38%.
LipNet: End-to-end Sentence-level Lipreading	95.2%	100x50 full color	Accuracy is measured on overlapped speakers. The accuracy drops to 89.6% on unseen speakers.
Lip Reading in the Wild (LRW)	46.8%	120x120 grayscale	Accuracy from visual recognition. The accuracy improves to 77.2% with audiovisual combination.
DeepSR	52.25%	30x30 grayscale	Accuracy on unseen words with mixed speakers.

Since our baseline model developed in this thesis still suffered from some over fitting, we explored parameter tuning as an opportunity for improvement. Two methods to improve the model are regularizer and dropout. Applying these methods successfully improved the model accuracy over the baseline model. The bidirectional layer in the experiment did not add value to the model.

The model implemented in this thesis achieved 52.25% accuracy score in a closed domain word level speech reading. It means that our approach outperforms human that achieves only 14.47% accuracy. Table 5 compares the results of other researchers. DeepSR is performing better than VSR and LRW on unseen mixed speakers. While the performance is lagging from LipNet, DeepSR succeeded in attaining good performance on a single GPU machine, whereas LipNet leveraged a much larger server of 8 GPUs. DeepSR performs relatively well with much smaller input dimension (30x30x1) per frame, compared to LipNet (100x50x3).

The experiments conducted throughout this thesis open opportunities for future work. Two areas that could be improved are the training time and the accuracy score. A significant amount of time was spent on loading the dataset from disk to memory. The memory capacity limit in the GPU requires the training process to load the data segments in time for training. While larger memory could solve this problem, a better data layout in contiguous columnar format is preferred to reduce the read time during the data-sampling step. In terms of the accuracy score, the final model can benefit from a larger neural network structure. Furthermore, adding more nodes in the CNN stack along with a higher resolution of the dataset can improve the result.

References

[1] Bunger, Anna Mae. “Speech Reading, Jena Method”. Danville, Illinois. The Interstate. 1961.

[2] LeCun, Y., Bengio, Y., Hinton, G. “Deep Learning”. Nature Reviews vol. 521. Macmillan Publishers. 2015.

[3] Altieri, N., Pisoni, D., Townsend, J., “Some Normatif Data on Reading Lips Skill”. The Journal of the Acoustical Society of America. 2011 Jul. 130(1): 1–4. PMCID: PMC3155585. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3155585/>

[4] -, “Paying Lip Service: The State of Lip Reading Classes in England and Wales”. Council on Deafness, ATLA, Hearing Concern LINK, RIND. 2010. <https://www.actiononhearingloss.org.uk/~/media/Documents/Campaigns/Paying_Lip_Service_PDF.ashx>

[5] Assael, Y., Shillingford, B., Whiteson, S., Freitas, N. “LipNet: End-to-end Sentence-level Lipreading”. V2. 16 Dec 2016. <https://arxiv.org/abs/1611.01599v2>

[6] -, “LipNet Reads Lips with Accuracy 93.4%”. Reddit.com. 2016. <https://www.reddit.com/r/MachineLearning/comments/5blc1z/p_lipnet_reads_lips_with_934_accuracy/>

[7] Hodson, H., “Google’s DeepMind AI can Lipread TV Shows Better than a Pro”. Newscientist.com. 2016. <https://www.newscientist.com/article/2113299-googles-deepmind-ai-can-lip-read-tv-shows-better-than-a-pro/>

[8] Chung, J., Senior, A., Vinyals, O., Zisserman, A., “Lip Reading in the Wild”. V2. 30 Jan 2017. <https://arxiv.org/abs/1611.05358v2>

[9] Hassanat, A. “Visual Speech Recognition”. V1. 3 Sep 2014. <https://arxiv.org/abs/1409.1411v1>

[10] Werda, S., Mahdi, W., Hamadou, A., “A Spatial-Temporal Technique of Viseme Extraction: Application in Speech Recognition”. IEEE SITIS 2005. <https://www.researchgate.net/publication/250869167_A_Spatial-Temporal_technique_of_Viseme_Extraction_Application>

[11] Cooke, M., Barker, J. “An audio-visual corpus for speech perception and automatic speech recognition (L)”. Acoustical Society of America. 2006. <http://laslab.org/upload/an_audio-visual_corpus_for_speech_perception_and_automatic_speech_recognition.pdf>

[12] -, “Professional Lip-reader Consuelo Gonzalez’s Cool Job”. The Seattle Times. 2015 Nov. <http://www.seattletimes.com/nwshowcase/careers/professional-lip-reader-consuelo-gonzalezs-cool-job/>

[13] McGregor, G., “Lip Reading Practice: M-P-B Sounds”. <http://www.lipreadingpractice.co.uk/Lip-Reading-Exercises/Consonants/m-p-b-sounds/>

[14] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, Dragomir, Erhan, D., Vanhoucke, V., Rabinovich, A., “Going Deeper With Convolutions”. 17 Sep 2014. <https://arxiv.org/pdf/1409.4842.pdf>

[15] LeCun, Y., Haffner, P., Bottou, L., Bengio, Y., “Object Recognition with Gradient-Based Learning”. Red Bank, New Jersey. 1999.

[16] Graves, A., Schmiduber, J., “Framewise phoneme classification with bidirectional LSTM and other neural network architectures”. Neural Networks vol. 18.

July-August 2005.

[17] _, “Tensorflow”. 2018.<http://www.tensorflow.org/>

[18] _, “Scikit-learn: Machine Learning in Python”. 2018. <http://www.scikit-learn.org/stable/>

[19] _, “Python Data Analysis Library”. 2018. <https://pandas.pydata.org/>

[20] _, “MessagePack: It’s like JSON but Fast and Small”. 2018. <https://msgpack.org/index.html>

[21] _, “Ffmpeg”. 2018. <https://www.ffmpeg.org/>

[22] _, “dlib C++ Library”. 2018. <http://dlib.net>

[23] Graves A., Fernandez S., Gomez F., Schmiduber J., “Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks”. ICML page 369-376. 2006.

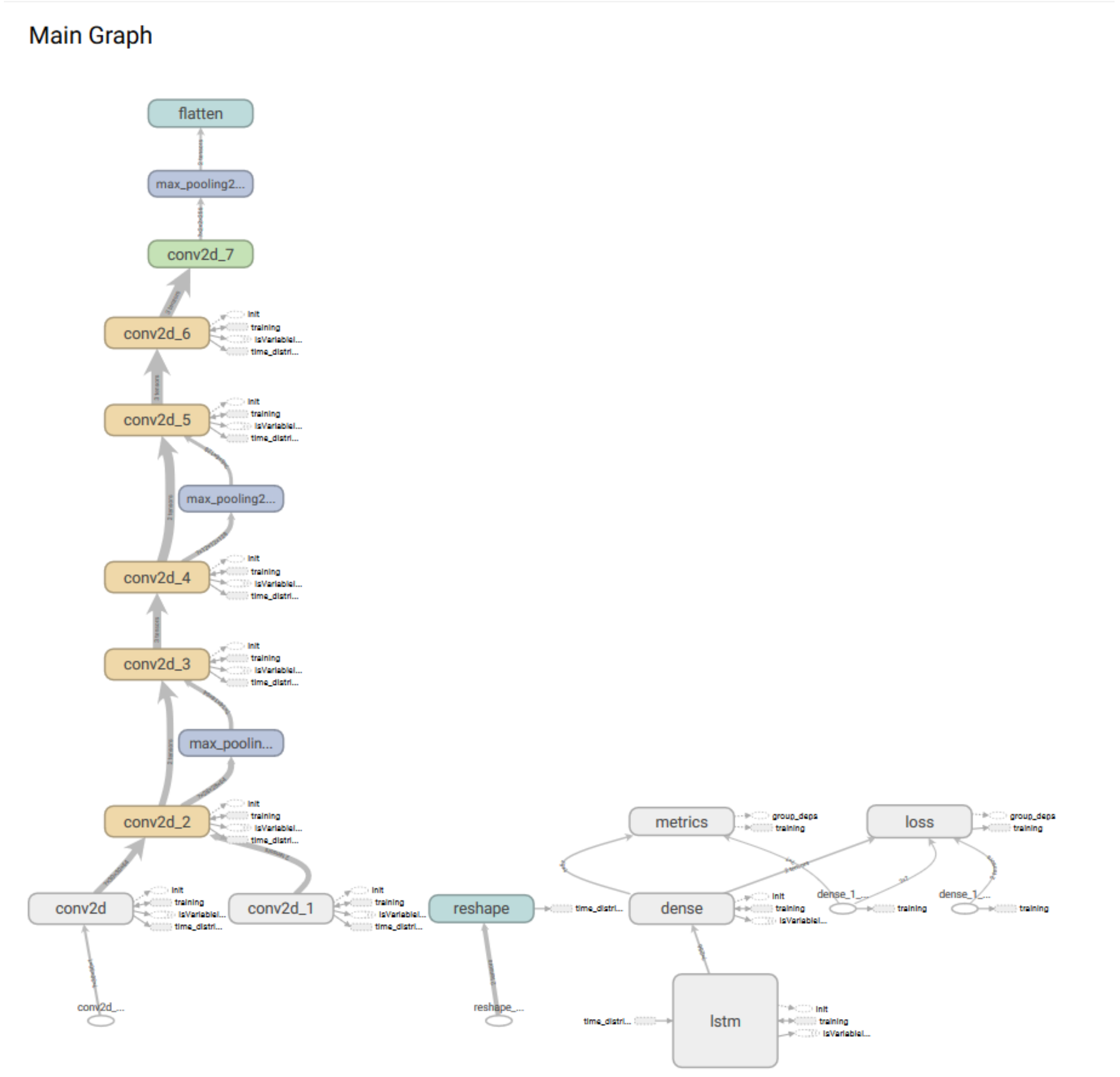
[24] Barker, J., Cooke M., Cunningham S., Shao X., “The GRID Audiovisual Sentence Corpus”. 2013. <http://spandh.dcs.shef.ac.uk/gridcorpus/>

[25] Garcia, C., Tziritas, G., “Face Detection Using Quantized Skin Color Regions Merging and Wavelet Packet Analysis”. IEEE Transactions On Multimedia. Vol.1 No. 3. 1999.

[26] Hilder S., Harvey R., Theobald B., “Comparison of human and machine based lip-reading”. AVSP. 2009. < http://www.isca-speech.org/archive_open/avsp09/papers/av09_086.pdf >

[27] Zeiler, M., “Adadelta: An Adaptive Learning Rate Method”. 2012. < https://arxiv.org/pdf/1212.5701.pdf >

The diagram below is generated in Tensorboard. It depicts the DeepSR network topology in the implementation.



Appendix B Source Code

This following is the full codes and the execution result in jupyter notebook file format. Several parts of the output are truncated to save space without missing the important information.

```
{
 "cells": [
  {
   "cell_type": "code",
   "execution_count": 1,
   "metadata": {},
   "outputs": [],
   "source": [
      "# update1 - train using subset data ~13k rows. accuracy: 5-
6%\n",
      "# update2 - load dataset in chunks, train using all data.
smaller network shape 32 - 8, 64 - 16\n",
      "# update2-remodel - change implementation with keras, resume
train with chunks\n",
      "# update2-remodel1 - add tensorboard, roc, evaluate blind set
and save model\n",
      "# update2-remodel2 - refactoring\n",
      "# update3 - full run of the model (24 pickles for training and
test, 1 pickle for blind, 20 epochs)\n",
      "# update4 - update3 suffers from overfitting, tune to
generalize the model (baseline, l2 reg, dropout)\n",
      "# update4-finaltuning - a tuned version of update4-baseline 1/2
regularizer, 1.5x dropout, 2x epoch\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 2,
   "metadata": {},
   "outputs": [
      {
      "name": "stderr",
      "output_type": "stream",
      "text": [
      "c:\\users\\basuki\\appdata\\local\\programs\\python\\python36\\
lib\\site-packages\\h5py\\__init__.py:36: FutureWarning: Conversion of
the second argument of issubdtype from `float` to `np.floating` is
deprecated. In future, it will be treated as `np.float64 ==
np.dtype(float).type`.\n",
      "  from ._conv import register_converters as
_register_converters\n"
      ]
      },
      {
      "data": {
      "text/plain": [
      "'1.5.0'"
      ]
      },
      "execution_count": 2,
      "metadata": {},
      "output_type": "execute_result"
      }
   ],
   "source": [
      "# Load and check Tensorflow library version\n",
      "import tensorflow as tf\n",
      "tf.__version__\n"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
      "# Dataset\n",
      "The dataset is a collection of word clips, cropped to halflips,
with resolution 30x30 pixels, grayscale. The dataset is stored in the
disk as 25 pickle files.  \n",
      "The code below extract the dataset from pickle and produce a
pandas dataframe with the 30x30 clips."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 3,
   "metadata": {},
   "outputs": [],
   "source": [
      "# Load required libraries\n",
      "import os\n",
      "import pickle\n",
      "import pandas as pd\n",
      "import msgpack\n",
      "import msgpack_numpy as mnp\n",
      "import gc\n",
      "import numpy as np\n",
      "import sklearn\n",
      "import sklearn.model_selection\n",
      "import random"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 4,
   "metadata": {},
   "outputs": [],
   "source": [
      "# Set configuration\n",
      "datadir = \"C:/Users/Basuki/Documents/grid_dataset\"\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 5,
   "metadata": {},
   "outputs": [],
   "source": [
      "# Collection of functions definition required to process
dataset\n",
      "\n",
      "\"\"\"\n",
      "Function load_chunks reads a pickle file from the data
directory, unpacks the mouth clips,\n",
      "splits to half lips, and produces a pandas dataframe.\n",
      "Param: datadir, pickle_file\n",
      "Output: a dataframe\n",
      "\"\"\"\n",
      "def load_chunks(datadir, pickle_file):\n",
      "     #open dictionary\n",
      "     with open(os.path.join(datadir,\"dictionary.pickle\"),
'rb') as handle:\n",
      "     one_hot_dictionary = pickle.load(handle)\n",
      "     #load pickle apply one_hot\n",
      "     df_pickle =
pd.read_pickle(os.path.join(datadir,pickle_file))\n",
      "     df_pickle[\"one_hot_label\"] =
df_pickle[\"label\"].apply(lambda x: one_hot_dictionary[x])\n",
      "     #unpack lips\n",
      "     df_pickle[\"unpacked\"] =
df_pickle[\"frames\"].apply(lambda x:msgpack.unpackb(x,
object_hook=mnp.decode))\n",
      "     df_pickle[\"left\"] = df_pickle[\"unpacked\"].apply(lambda
x: x[:,20:50,:30].ravel().astype(\"float32\"))\n",
      "     df_pickle[\"right\"] =
df_pickle[\"unpacked\"].apply(lambda x:
x[:,20:50,30:].ravel().astype(\"float32\"))\n",
      "     #combine\n",
      "     df_left =
df_pickle[[\"left\",\"one_hot_label\"]].rename(columns={\"left\":\"fea
t\"})\n",
      "     df_right =
df_pickle[[\"right\",\"one_hot_label\"]].rename(columns={\"right\":\"f
eat\"})\n",
      "     df_halflips =
pd.concat([df_left,df_right]).reset_index(drop=True)\n",
      "     df_halflips[\"t\"] = [i.shape[0] for i in
df_halflips.feat]\n",
      "     df_halflips = df_halflips.sort_values(by=\"t\")\n",
      "     #preserve memory\n",
      "     df_result = df_halflips.copy()\n",
      "     del df_halflips\n",
      "     del df_left\n",
      "     del df_right\n",
      "     del df_pickle\n",
      "     gc.collect()\n",
      "     # returned the chunks as a pandas dataframe\n",
      "     return df_result\n",
      "\n",
      "\"\"\"\n",
      "Function get_dataset reads a pickle file at certain index in
the data directory\n",
      "and produces the feature set X_data and label y_data.\n",
      "Param: datadir, index\n",
      "Output: X_data, y_data\n",
      "\"\"\"\n",
      "def get_dataset(datadir, index):\n",
      "     #load data chunks\n",
      "     visemes_pickles = [ f for f in os.listdir(datadir) if
f.startswith(\"visemes_\")]\n",
      "     df_halflips = load_chunks(datadir,
visemes_pickles[index])\n",
      "     #get features, add padding\n",
      "     X_data = np.vstack(df_halflips.feat.apply(lambda x:
np.resize(x,(9000))).values)\n",
      "     #get one_hot_label\n",
      "     y_data = np.array(df_halflips.one_hot_label.tolist())\n",
      "     #preserve memory\n",
      "     del df_halflips\n",
      "     gc.collect()\n",
      "     return X_data, y_data\n",
      "\n",
      "\"\"\"\n",
      "Function get_chunks_len reads all files in the datadir and
counts the number of rows\n",
      "in each file.\n",
      "Param: datadir\n",
      "Output: list of row counts in each file\n",
      "\"\"\"\n",
      "def get_chunks_len(datadir):\n",
      "     l=[]\n",
      "     for i in range(25):\n",
      "     X_data, y_data = get_dataset(datadir, i)\n",
      "     l.append(len(X_data))\n",
      "     return l\n",
      "\n",
      "\"\"\"\n",
      "Function get_random_indices generates list of random indices
for each file in the datadir,\n",
      "and splits the indices to three parts 90% train, 5% test and 5%
blind.\n",
      "Param: datadir\n",
      "Output: list of random indices\n",
      "\"\"\"\n",
      "def get_random_indices(datadir):\n",
      "     #use get_chunks len if working with different dataset\n",
      "     #chunks_len = get_chunks_len(datadir)\n",
      "     chunks_len = [19510, 24162, 15654, 14634, 15630, 16634,
16370, 18654, 16892, 15928, 15720, 16080, 17934, 14526, 12738, 13364,
15852, 15548, 11870, 14196, 13264, 15976, 15056, 15126, 13422]\n",
      "     l = chunks_len[0]\n",
      "     random.seed(7)\n",
      "     indices = random.sample([e for e in range(l)], l)\n",
      "     train_indices = [indices[:round(.9*len(indices))]]\n",
      "     test_indices =
[indices[round(.9*len(indices)):round(.95*len(indices))]]\n",
      "     blind_indices = [indices[round(.95*len(indices)):]]\n",
      "     for l in chunks_len[1:]:\n",
      "     random.seed(7)\n",
      "     indices = random.sample([e for e in range(l)], l)\n",
      "     train_indices.append(indices[:round(.9*len(indices))])\n",
      "
      test_indices.append(indices[round(.9*len(indices)):round(.95*len
(indices))])\n",
      "     blind_indices.append(indices[round(.95*len(indices)):])
      \n",
      "     return train_indices, test_indices, blind_indices\n",
      "\n",
      "\"\"\"\n",
      "Function get_train_by_indices reads a pickle file at certain
index in the data directory\n",
      "and produces the train feature set based on the provided
indices.\n",
      "Param: datadir, index, train_indices\n",
      "Output: X_train, y_train\n",
      "\"\"\"\n",
      "def get_train_by_indices(datadir, index, train_indices):\n",
      "     X_data, y_data = get_dataset(datadir, index)\n",
      "     X_train = X_data.take(train_indices, axis=0)\n",
      "     y_train = y_data.take(train_indices, axis=0)\n",
      "     return X_train, y_train\n",
      "\n",
      "\"\"\"\n",
      "Function get_test_by_indices reads a pickle file at certain
index in the data directory\n",
      "and produces the test feature set, and test label based on the
provided indices.\n",
      "Param: datadir, index, test_indices\n",
      "Output: X_test, y_test\n",
      "\"\"\"\n",
      "def get_test_by_indices(datadir, index, test_indices):\n",
      "     X_data, y_data = get_dataset(datadir, index)\n",
      "     X_test = X_data.take(test_indices, axis=0)\n",
      "     y_test = y_data.take(test_indices, axis=0)\n",
      "     return X_test, y_test\n",
      "\n",
      "\"\"\"\n",
      "Function get_blind_indices reads a pickle file at certain index
in the data directory\n",
      "and produces the blind feature set, and blind label.\n",
      "Param: datadir, index, blind_indices\n",
      "Output: X_blind, y_blind\n",
      "\"\"\"\n",
      "def get_blind_by_indices(datadir, index, blind_indices):\n",
      "     X_data, y_data = get_dataset(datadir, index)\n",
      "     X_blind = X_data.take(blind_indices, axis=0)\n",
      "     y_blind = y_data.take(blind_indices, axis=0)\n",
      "     return X_blind, y_blind"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
      "\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": []
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": []
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
      "# Model\n",
      "The model is a series of ConvNets in a LSTM network. The
ConvNets acquires the image features for each frame in the mouth
clips. The LSTM recognizes the sequences of ConvNets output and
classifies the sequence to a word label/class."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 6,
   "metadata": {},
   "outputs": [],
   "source": [
      "# Set configuration\n",
      "regularize = True\n",
      "dropout = True\n",
      "bidirectional = False"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 7,
   "metadata": {
      "scrolled": true
   },
   "outputs": [
      {
      "name": "stdout",
      "output_type": "stream",
      "text": [
      "WARNING:tensorflow:From
c:\\users\\basuki\\appdata\\local\\programs\\python\\python36\\lib\\si
te-packages\\tensorflow\\python\\keras\\_impl\\keras\\backend.py:1456:
calling reduce_sum (from tensorflow.python.ops.math_ops) with
keep_dims is deprecated and will be removed in a future version.\n",
      "Instructions for updating:\n",
      "keep_dims is deprecated, use keepdims instead\n",
      "WARNING:tensorflow:From
c:\\users\\basuki\\appdata\\local\\programs\\python\\python36\\lib\\si
te-packages\\tensorflow\\python\\keras\\_impl\\keras\\backend.py:1557:
calling reduce_mean (from tensorflow.python.ops.math_ops) with
keep_dims is deprecated and will be removed in a future version.\n",
      "Instructions for updating:\n",
      "keep_dims is deprecated, use keepdims instead\n",

      "_______________________________________________________________
__\n",
      "Layer (type)                 Output Shape              Param #
\n",

      "===============================================================
==\n",
      "conv2d_1 (Conv2D)            (None, 30, 30, 64)        640
      \n",

      "_______________________________________________________________
__\n",
      "conv2d_2 (Conv2D)            (None, 28, 28, 64)        36928
      \n",

      "_______________________________________________________________
__\n",
      "max_pooling2d_1 (MaxPooling2 (None, 14, 14, 64)        0
      \n",

      "_______________________________________________________________
__\n",
      "conv2d_3 (Conv2D)            (None, 14, 14, 128)       73856
      \n",

      "_______________________________________________________________
__\n",
      "conv2d_4 (Conv2D)            (None, 12, 12, 128)       147584
      \n",
      "_________________________________________________________________
__\n",
      "max_pooling2d_2 (MaxPooling2 (None, 6, 6, 128)       0         
      \n",
      
      "_________________________________________________________________
__\n",
      "conv2d_5 (Conv2D)            (None, 6, 6, 256)       295168    
      \n",
      
      "_________________________________________________________________
__\n",
      "conv2d_6 (Conv2D)            (None, 4, 4, 256)       590080    
      \n",
      
      "_________________________________________________________________
__\n",
      "conv2d_7 (Conv2D)            (None, 2, 2, 256)       590080    
      \n",
      
      "_________________________________________________________________
__\n",
      "max_pooling2d_3 (MaxPooling2 (None, 1, 1, 256)       0         
      \n",
      
      "_________________________________________________________________
__\n",
      "flatten_1 (Flatten)          (None, 256)             0         
      \n",
      
      "=================================================================
==\n",
      "Total params: 1,734,336\n",
      "Trainable params: 1,734,336\n",
      "Non-trainable params: 0\n",
      
      "_________________________________________________________________
__\n",
      
      "_________________________________________________________________
__\n",
      "Layer (type)                 Output Shape            Param #   
\n",
      
      "=================================================================
==\n",
      "reshape_1 (Reshape)          (None, 10, 30, 30, 1)   0         
      \n",

      "_______________________________________________________________
__\n",
      "time_distributed_1 (TimeDist (None, 10, 256)         1734336
\n",

      "_______________________________________________________________
__\n",
      "lstm_1 (LSTM)                (None, 256)              525312
      \n",

      "_______________________________________________________________
__\n",
      "dropout_1 (Dropout)          (None, 256)              0
      \n",

      "_______________________________________________________________
__\n",
      "dense_1 (Dense)              (None, 51)               13107
      \n",

      "===============================================================
==\n",
      "Total params: 2,272,755\n",
      "Trainable params: 2,272,755\n",
      "Non-trainable params: 0\n",

      "_______________________________________________________________
__\n"
      ]
      }
   ],
   "source": [
      "# Create Neural Networks model\n",
      "from tensorflow.python.keras.layers import Conv2D,
MaxPooling2D, Flatten, LSTM, Dense, Dropout, TimeDistributed, Input,
InputLayer, Reshape, Bidirectional\n",
      "from tensorflow.python.keras.models import Sequential\n",
      "from tensorflow.python.keras import losses, regularizers,
optimizers, callbacks\n",
      "\n",
      "\"\"\"\n",
      "Instance vision_model is a Convolution Networks for learning
the features in each image within a mouth clips.\n",
      "This code follows the same code in Keras Documentation with a
change in input shape.\n",
      "\"\"\"\n",
      "vision_model = Sequential()\n",
      "vision_model.add(Conv2D(64, (3, 3), activation='relu',
padding='same', input_shape=(30, 30, 1)))\n",
      "vision_model.add(Conv2D(64, (3, 3), activation='relu'))\n",
      "vision_model.add(MaxPooling2D((2, 2)))\n",
      "vision_model.add(Conv2D(128, (3, 3), activation='relu',
padding='same'))\n",
      "vision_model.add(Conv2D(128, (3, 3), activation='relu'))\n",
      "vision_model.add(MaxPooling2D((2, 2)))\n",
      "vision_model.add(Conv2D(256, (3, 3), activation='relu',
padding='same'))\n",
      "vision_model.add(Conv2D(256, (3, 3), activation='relu'))\n",
      "vision_model.add(Conv2D(256, (3, 3), activation='relu'))\n",
      "vision_model.add(MaxPooling2D((2, 2)))\n",
      "vision_model.add(Flatten())\n",
      "\n",
      "\"\"\"\n",
      "Instance video_model is a TimeDistributed/LSTM Networks for
learning the sequence of a mouth clips.\n",
      "This code follows the same code in Keras Documentation with a
change in the input shape.\n",
      "\"\"\"\n",
      "video_model = Sequential()\n",
      "video_model.add(Reshape((10,30,30,1),input_shape=(9000,)))\n",
      "video_model.add(TimeDistributed(vision_model))\n",
      "if bidirectional:\n",
      "     video_model.add(Bidirectional(LSTM(256)))\n",
      "else:\n",
      "     video_model.add(LSTM(256))    \n",
      "if dropout:\n",
      "     video_model.add(Dropout(0.75))      \n",
      "if regularize:\n",
      "     video_model.add(Dense(51,
kernel_regularizer=regularizers.l2(0.005),
activation=\"softmax\"))\n",
      "else:\n",
      "     video_model.add(Dense(51, activation=\"softmax\"))\n",
      "\n",
      "video_model.compile(loss=losses.categorical_crossentropy,\n",
      "
      optimizer=optimizers.Adadelta(),metrics=[\"accuracy\"])\n",
      "\n",
      "\"\"\"\n",
      "Print model summary\n",
      "\"\"\"\n",
      "vision_model.summary()\n",
      "video_model.summary()"
   ]
```

```
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
      "# Training\n",
      "The training used 80% of the data and validated against the
rest 20%. Training is done per pickle file to preserve memory. Each
data is seen in 2 epochs during the training."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 8,
   "metadata": {},
   "outputs": [],
   "source": [
      "# Load required libraries\n",
      "import h5py"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 9,
   "metadata": {},
   "outputs": [],
   "source": [
      "# Set configuration\n",
      "batch_size = 128\n",
      "num_classes = 51\n",
      "epochs = 20 #est. an hour per epoch\n",
      "num_pickle_files = 25 #max 25\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 10,
   "metadata": {},
   "outputs": [],
   "source": [
      "model_name =
\"_\".join((\"./trained\",\"thesis\",\"u4\",\"02042018\",str(epochs),s
tr(num_pickle_files)))\n",
      "if regularize:\n",
      "     model_name = \"_\".join((model_name,\"regularize\"))\n",
      "if dropout:\n",
      "     model_name = \"_\".join((model_name,\"dropout\"))\n",
      "if bidirectional:\n",
      "     model_name = \"_\".join((model_name,\"bidirectional\"))\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 11,
   "metadata": {},
   "outputs": [
      {
      "name": "stdout",
      "output_type": "stream",
      "text": [
      "16789/16789 [==============================]16789/16789
[==============================] - 122s 7ms/step - loss: 1.1464 - acc:
0.7251 - val_loss: 2.3427 - val_acc: 0.4359\n",
      "\n",
      "Epoch: 18 of 20, Chunk Num: 8 of 25\n",
      "Train on 15203 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "15203/15203 [==============================]15203/15203
[==============================] - 113s 7ms/step - loss: 0.9355 - acc:
0.7831 - val_loss: 2.2300 - val_acc: 0.4646\n",
      "\n",
      "Epoch: 18 of 20, Chunk Num: 9 of 25\n",
      "Train on 14335 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "14335/14335 [==============================]14335/14335
[==============================] - 109s 8ms/step - loss: 0.8370 - acc:
0.8102 - val_loss: 2.1302 - val_acc: 0.4944\n",
      "\n",
      "Epoch: 18 of 20, Chunk Num: 10 of 25\n",
      "Train on 14148 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "14148/14148 [==============================]14148/14148
[==============================] - 106s 8ms/step - loss: 0.7627 - acc:
0.8261 - val_loss: 2.1920 - val_acc: 0.4677\n",
      "\n",
      "Epoch: 18 of 20, Chunk Num: 11 of 25\n",
      "Train on 14472 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "14472/14472 [==============================]14472/14472
[==============================] - 110s 8ms/step - loss: 0.8377 - acc:
0.8127 - val_loss: 2.4714 - val_acc: 0.4215\n",
      "\n",
      "Epoch: 18 of 20, Chunk Num: 12 of 25\n",
      "Train on 16141 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "16141/16141 [==============================]16141/16141
```

```
[==============================] - 118s 7ms/step - loss: 1.0806 - acc:
0.7443 - val_loss: 2.3749 - val_acc: 0.4369\n",
      "\n",
      "Epoch: 18 of 20, Chunk Num: 13 of 25\n",
      "Train on 13073 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "13073/13073 [==============================]13073/13073
[==============================] - 103s 8ms/step - loss: 0.7655 - acc:
0.8308 - val_loss: 2.1130 - val_acc: 0.5005\n",
      "\n",
      "Epoch: 18 of 20, Chunk Num: 14 of 25\n",
      "Train on 11464 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "11464/11464 [==============================]11464/11464
[==============================] - 95s 8ms/step - loss: 0.8070 - acc:
0.8173 - val_loss: 2.0977 - val_acc: 0.5015\n",
      "\n",
      "Epoch: 18 of 20, Chunk Num: 15 of 25\n",
      "Train on 12028 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "12028/12028 [==============================]12028/12028
[==============================] - 96s 8ms/step - loss: 1.0305 - acc:
0.7578 - val_loss: 2.2031 - val_acc: 0.4821\n",
      "\n",
      "Epoch: 18 of 20, Chunk Num: 16 of 25\n",
      "Train on 14267 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "14267/14267 [==============================]14267/14267
[==============================] - 107s 8ms/step - loss: 0.8540 - acc:
0.8006 - val_loss: 2.4403 - val_acc: 0.4277\n",
      "\n",
      "Epoch: 18 of 20, Chunk Num: 17 of 25\n",
      "Train on 13993 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "13993/13993 [==============================]13993/13993
[==============================] - 108s 8ms/step - loss: 1.0421 - acc:
0.7502 - val_loss: 2.2521 - val_acc: 0.4605\n",
      "\n",
      "Epoch: 18 of 20, Chunk Num: 18 of 25\n",
      "Train on 10683 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "10683/10683 [==============================]10683/10683
[==============================] - 91s 9ms/step - loss: 1.3713 - acc:
0.6845 - val_loss: 2.2105 - val_acc: 0.4790\n",
      "\n",
      "Epoch: 18 of 20, Chunk Num: 19 of 25\n",
      "Train on 12776 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "12776/12776 [==============================]12776/12776
[==============================] - 102s 8ms/step - loss: 0.8814 - acc:
0.8131 - val_loss: 2.2588 - val_acc: 0.4595\n",
      "\n",
      "Epoch: 18 of 20, Chunk Num: 20 of 25\n",
      "Train on 11938 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "11938/11938 [==============================]11938/11938
[==============================] - 97s 8ms/step - loss: 0.9618 - acc:
0.7750 - val_loss: 2.2829 - val_acc: 0.4482\n",
      "\n",
      "Epoch: 18 of 20, Chunk Num: 21 of 25\n",
      "Train on 14378 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "14378/14378 [==============================]14378/14378
[==============================] - 109s 8ms/step - loss: 0.8294 - acc:
0.8150 - val_loss: 2.2509 - val_acc: 0.4513\n",
      "\n",
      "Epoch: 18 of 20, Chunk Num: 22 of 25\n",
      "Train on 13550 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "13550/13550 [==============================]13550/13550
[==============================] - 105s 8ms/step - loss: 0.7465 - acc:
0.8346 - val_loss: 2.0527 - val_acc: 0.5077\n",
      "\n",
      "Epoch: 18 of 20, Chunk Num: 23 of 25\n",
      "Train on 13613 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "13613/13613 [==============================]13613/13613
[==============================] - 105s 8ms/step - loss: 0.6669 - acc:
0.8522 - val_loss: 2.0631 - val_acc: 0.5200\n",
      "\n",
      "Epoch: 18 of 20, Chunk Num: 24 of 25\n",
      "Train on 12080 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "12080/12080 [==============================]12080/12080
[==============================] - 98s 8ms/step - loss: 0.8892 - acc:
0.7917 - val_loss: 1.9752 - val_acc: 0.5364\n",
      "\n",
      "Epoch: 19 of 20, Chunk Num: 0 of 25\n",
      "Train on 17559 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "17559/17559 [==============================]17559/17559
[==============================] - 126s 7ms/step - loss: 1.0873 - acc:
0.7362 - val_loss: 1.4870 - val_acc: 0.6410\n",
      "\n",
      "Epoch: 19 of 20, Chunk Num: 1 of 25\n",
      "Train on 21746 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "21746/21746 [==============================]21746/21746
[==============================] - 145s 7ms/step - loss: 0.8668 - acc:
0.7996 - val_loss: 1.5738 - val_acc: 0.6164\n",
      "\n",
      "Epoch: 19 of 20, Chunk Num: 2 of 25\n",
      "Train on 14089 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "14089/14089 [==============================]14089/14089
[==============================] - 108s 8ms/step - loss: 0.9835 - acc:
0.7734 - val_loss: 1.8372 - val_acc: 0.5559\n",
      "\n",
      "Epoch: 19 of 20, Chunk Num: 3 of 25\n",
      "Train on 13171 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "13171/13171 [==============================]13171/13171
[==============================] - 103s 8ms/step - loss: 0.8812 - acc:
0.8035 - val_loss: 1.8790 - val_acc: 0.5518\n",
      "\n",
      "Epoch: 19 of 20, Chunk Num: 4 of 25\n",
      "Train on 14067 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "14067/14067 [==============================]14067/14067
[==============================] - 108s 8ms/step - loss: 0.9122 - acc:
0.7887 - val_loss: 1.9955 - val_acc: 0.5138\n",
      "\n",
      "Epoch: 19 of 20, Chunk Num: 5 of 25\n",
      "Train on 14971 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "14971/14971 [==============================]14971/14971
[==============================] - 113s 8ms/step - loss: 0.9275 - acc:
0.7870 - val_loss: 2.0016 - val_acc: 0.5149\n",
      "\n",
      "Epoch: 19 of 20, Chunk Num: 6 of 25\n",
      "Train on 14733 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "14733/14733 [==============================]14733/14733
[==============================] - 111s 8ms/step - loss: 0.8405 - acc:
0.8089 - val_loss: 2.0987 - val_acc: 0.4964\n",
      "\n",
      "Epoch: 19 of 20, Chunk Num: 7 of 25\n",
      "Train on 16789 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "16789/16789 [==============================]16789/16789
[==============================] - 122s 7ms/step - loss: 1.1217 - acc:
0.7342 - val_loss: 2.0972 - val_acc: 0.4738\n",
      "\n",
      "Epoch: 19 of 20, Chunk Num: 8 of 25\n",
      "Train on 15203 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "15203/15203 [==============================]15203/15203
[==============================] - 114s 7ms/step - loss: 0.9127 - acc:
0.7894 - val_loss: 2.1298 - val_acc: 0.5097\n",
      "\n",
      "Epoch: 19 of 20, Chunk Num: 9 of 25\n",
      "Train on 14335 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "14335/14335 [==============================]14335/14335
[==============================] - 109s 8ms/step - loss: 0.8177 - acc:
0.8144 - val_loss: 2.1139 - val_acc: 0.5005\n",
      "\n",
      "Epoch: 19 of 20, Chunk Num: 10 of 25\n",
      "Train on 14148 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "14148/14148 [==============================]14148/14148
[==============================] - 109s 8ms/step - loss: 0.7376 - acc:
0.8312 - val_loss: 2.2838 - val_acc: 0.4656\n",
      "\n",
      "Epoch: 19 of 20, Chunk Num: 11 of 25\n",
      "Train on 14472 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "14472/14472 [==============================]14472/14472
[==============================] - 110s 8ms/step - loss: 0.8131 - acc:
0.8177 - val_loss: 2.7184 - val_acc: 0.3713\n",
      "\n",
      "Epoch: 19 of 20, Chunk Num: 12 of 25\n",
      "Train on 16141 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "16141/16141 [==============================]16141/16141
[==============================] - 119s 7ms/step - loss: 1.0588 - acc:
0.7510 - val_loss: 2.3813 - val_acc: 0.4482\n"
      ]
      },
      {
      "name": "stdout",
      "output_type": "stream",
      "text": [
      "\n",
      "Epoch: 19 of 20, Chunk Num: 13 of 25\n",
      "Train on 13073 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "13073/13073 [==============================]13073/13073
[==============================] - 103s 8ms/step - loss: 0.7381 - acc:
0.8356 - val_loss: 2.1558 - val_acc: 0.4985\n",
      "\n",
      "Epoch: 19 of 20, Chunk Num: 14 of 25\n",
      "Train on 11464 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "11464/11464 [==============================]11464/11464
[==============================] - 95s 8ms/step - loss: 0.7978 - acc:
0.8214 - val_loss: 2.0674 - val_acc: 0.5128\n",
      "\n",
      "Epoch: 19 of 20, Chunk Num: 15 of 25\n",
      "Train on 12028 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "12028/12028 [==============================]12028/12028
[==============================] - 98s 8ms/step - loss: 1.0218 - acc:
0.7593 - val_loss: 2.1671 - val_acc: 0.4831\n",
      "\n",
      "Epoch: 19 of 20, Chunk Num: 16 of 25\n",
      "Train on 14267 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "14267/14267 [==============================]14267/14267
[==============================] - 108s 8ms/step - loss: 0.8165 - acc:
0.8113 - val_loss: 2.3713 - val_acc: 0.4287\n",
      "\n",
      "Epoch: 19 of 20, Chunk Num: 17 of 25\n",
      "Train on 13993 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "13993/13993 [==============================]13993/13993
[==============================] - 108s 8ms/step - loss: 1.0269 - acc:
0.7554 - val_loss: 2.3424 - val_acc: 0.4564\n",
      "\n",
      "Epoch: 19 of 20, Chunk Num: 18 of 25\n",
      "Train on 10683 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "10683/10683 [==============================]10683/10683
[==============================] - 91s 9ms/step - loss: 1.3493 - acc:
0.6890 - val_loss: 2.0945 - val_acc: 0.5087\n",
      "\n",
      "Epoch: 19 of 20, Chunk Num: 19 of 25\n",
      "Train on 12776 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "12776/12776 [==============================]12776/12776
[==============================] - 102s 8ms/step - loss: 0.8676 - acc:
0.8155 - val_loss: 2.2836 - val_acc: 0.4718\n",
      "\n",
      "Epoch: 19 of 20, Chunk Num: 20 of 25\n",
      "Train on 11938 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "11938/11938 [==============================]11938/11938
[==============================] - 97s 8ms/step - loss: 0.9316 - acc:
0.7810 - val_loss: 2.2398 - val_acc: 0.4708\n",
      "\n",
      "Epoch: 19 of 20, Chunk Num: 21 of 25\n",
      "Train on 14378 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "14378/14378 [==============================]14378/14378
[==============================] - 111s 8ms/step - loss: 0.7990 - acc:
0.8200 - val_loss: 2.1713 - val_acc: 0.4759\n",
      "\n",
      "Epoch: 19 of 20, Chunk Num: 22 of 25\n",
      "Train on 13550 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "13550/13550 [==============================]13550/13550
[==============================] - 106s 8ms/step - loss: 0.7193 - acc:
0.8406 - val_loss: 2.1042 - val_acc: 0.4985\n",
      "\n",
      "Epoch: 19 of 20, Chunk Num: 23 of 25\n",
      "Train on 13613 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "13613/13613 [==============================]13613/13613
[==============================] - 106s 8ms/step - loss: 0.6417 - acc:
0.8604 - val_loss: 2.0051 - val_acc: 0.5251\n",
      "\n",
      "Epoch: 19 of 20, Chunk Num: 24 of 25\n",
      "Train on 12080 samples, validate on 24375 samples\n",
      "Epoch 1/1\n",
      "12080/12080 [==============================]12080/12080
[==============================] - 98s 8ms/step - loss: 0.8491 - acc:
0.8054 - val_loss: 2.0998 - val_acc: 0.5159\n",
      "\n"
      ]
      }
   ],
   "source": [
      "# Run training\n",
      "\n",
      "\"\"\"\n",
      "Instance tensorboard will write the training summary of the
model.\n",
      "\"\"\"\n",
      "tensorboard = callbacks.TensorBoard(log_dir=model_name,
histogram_freq=0, write_graph=True, write_images=True)\n",
      "\n",
      "\"\"\"\n",
      "The training iterates through the number of epochs. In each
epoch, the model is exposed to the entire training dataset.\n",
      "To preserve memory, the training is done per dataset file at a
time.\n",
      "\"\"\"\n",
      "train_idxs, test_idxs, blind_idxs =
get_random_indices(datadir)\n",
      "\n",
      "X_test, y_test = get_test_by_indices(datadir, 0,
test_idxs[0])\n",
      "for i in range(1, num_pickle_files):\n",
      "     Xtest, ytest = get_test_by_indices(datadir, 0,
test_idxs[0])\n",
      "     X_test = np.vstack((X_test,Xtest))\n",
      "     y_test = np.vstack((y_test,ytest))\n",
      "\n",
      "for e in range(epochs):\n",
      "     for i in range(num_pickle_files):\n",
      "     print(\"Epoch: {} of {}, Chunk Num: {} of
{}\".format(e,epochs,i,num_pickle_files))\n",
      "     X_train, y_train = get_train_by_indices(datadir, i,
train_idxs[i])\n",
      "\n",
      "     video_model.fit(X_train, y_train, batch_size=batch_size,
epochs=1, verbose=1, validation_data=(X_test,y_test),
callbacks=[tensorboard])\n",
      "\n",
      "     #preserve memory\n",
      "     del X_train, y_train\n",
      "     gc.collect()\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
      "\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": []
  },
  {
   "cell_type": "code",
   "execution_count": 12,
   "metadata": {},
   "outputs": [],
   "source": [
      "\"\"\"\n",
      "Save the trained model\n",
      "\"\"\"\n",
      "video_model.save(\"\".join((model_name,\".h5\")))"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
      "# Evaluation - Test set\n",
      "Evaluation is done in three criteria: accuracy, confusion
matrix and roc curve. This section evaluates the model against the
test set. The accuracy is expected to be high."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 13,
   "metadata": {},
   "outputs": [],
   "source": [
      "# Load required libraries\n",
      "import sklearn\n",
      "import matplotlib.pyplot as plt\n",
      "import seaborn as sn"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 14,
   "metadata": {},
   "outputs": [],
   "source": [
      "# Collection of functions definition required to evaluate the
model\n",
      "\n",
      "\"\"\"\n",
      "Function get_accuracy measures the accuracy score of the model
given features X and label y.\n",
      "Param: model, X, y\n",
      "Output: the accuracy score\n",
      "\"\"\"\n",
      "def get_accuracy(model, X, y):\n",
      "     #evaluate\n",
      "     y_score = model.predict(X)\n",
      "     y_pred = np.argmax(y_score, axis=1)\n",
      "     y_true = np.argmax(y,axis=1)\n",
      "     return sklearn.metrics.accuracy_score(y_true,y_pred)  \n",
      "\n",
      "\"\"\"\n",
      "Function show_confusion_matrix calculates and displays the 
confusion matrix of the model given features X and label y\n",
      "Param: model, X, y\n",
      "Output: confusion matrix\n",
      "\"\"\"\n",
      "def show_confusion_matrix(model, X, y):\n",
      "     #evaluate\n",
      "     y_score = model.predict(X)\n",
      "     y_pred = np.argmax(y_score, axis=1)\n",
      "     y_true = np.argmax(y,axis=1)\n",
      "     #show matrix\n",
      "     array = sklearn.metrics.confusion_matrix(y_true,y_pred)
      \n",
      "     df_cm = pd.DataFrame(array)\n",
      "     plt.figure(figsize = (100,100))\n",
      "     sn.set(font_scale=1.4)#for label size\n",
      "     sn.heatmap(df_cm, annot=True,annot_kws={\"size\": 16})# 
font size\n",
      "     return True\n",
      "\n",
      "\"\"\"\n",
      "Function show_roc_curve calculates and displays the roc curve 
of the model given features X and label y\n",
      "Param: model, X, y\n",
      "Output: roc curve\n",
      "\"\"\"\n",
      "def show_roc_curve(model, X, y):\n",
      "     y_score = model.predict(X)\n",
      "\n",
      "     # Compute ROC curve and ROC area for each class\n",
      "     fpr = dict()\n",
      "     tpr = dict()\n",
      "     roc_auc = dict()\n",
      "     for i in range(num_classes):\n",
      "     fpr[i], tpr[i], _ = sklearn.metrics.roc_curve(y[:, i], 
y_score[:, i])\n",
      "     roc_auc[i] = sklearn.metrics.auc(fpr[i], tpr[i])\n",
      "     print(roc_auc)\n",
      "\n",
      "     ## Draw Roc chart\n",
      "     for i in range(51):\n",
      "     plt.plot(fpr[i],tpr[i])\n",
      "     plt.plot([0, 1], [0, 1], 'k--')\n",
      "     plt.show()        \n",
      "     "
```

```
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 15,
   "metadata": {},
   "outputs": [],
   "source": [
      "# Reload test data\n",
      "#X_test, y_test = get_test_by_indices(datadir, 0,
test_idxs[0])\n",
      "#for i in range(1, num_pickle_files):\n",
      "#    Xtest, ytest = get_test_by_indices(datadir, 0,
test_idxs[0])\n",
      "#    X_test = np.vstack((X_test,Xtest))\n",
      "#    y_test = np.vstack((y_test,ytest))\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 16,
   "metadata": {},
   "outputs": [
      {
      "name": "stdout",
      "output_type": "stream",
      "text": [
      "Accuracy score: 51.59%\n"
      ]
      }
   ],
   "source": [
      "# Print accuracy score\n",
      "accuracy = get_accuracy(video_model, X_test, y_test)\n",
      "print(\"Accuracy score: {:.2f}%\".format(accuracy*100))"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 17,
   "metadata": {},
   "outputs": [
      {
      "data": {
      "text/plain": [
      "True"
      ]
      },
      "execution_count": 17,
      "metadata": {},
      "output_type": "execute_result"
      },
      {
      "data": {
      "image/png": ""\n",
      "text/plain": [
      "<matplotlib.figure.Figure at 0x159ecd0a940>"
      ]
      },
      "metadata": {},
      "output_type": "display_data"
      }
   ],
   "source": [
      "# Print confusion matrix\n",
      "show_confusion_matrix(video_model, X_test, y_test)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 18,
   "metadata": {},
   "outputs": [
      {
      "name": "stdout",
      "output_type": "stream",
      "text": [
      "{0: 0.9827497425334706, 1: 0.9647058823529411, 2:
0.8372085335213999, 3: 0.8031286894923259, 4: 0.9236972972972974, 5:
0.9420055837986498, 6: 0.9125910769537877, 7: 0.9301030927835052, 8:
0.9635362917096664, 9: 0.9522659305268001, 10: 0.841306047966632, 11:
0.8933021806853582, 12: 0.9291666666666668, 13: 0.9900352345298392,
14: 0.9773429454170958, 15: 0.9278568876639616, 16:
0.9997938144329896, 17: 0.9523686920700309, 18: 0.8436177991330235,
19: 0.9379524301964839, 20: 0.9159190246146768, 21:
0.9654282765737875, 22: 0.9384008462996429, 23: 0.8624913733609385,
24: 0.8345631641086186, 25: 0.9700509134471398, 26:
0.9520660744585288, 27: 0.8816872427983539, 28: 0.931875, 29:
0.8455200823892893, 30: 0.9803647416413374, 31: 0.9953868234007655,
32: 0.7878350515463918, 33: 0.9830072090628218, 34:
0.8608656381212771, 35: 0.8861168562564633, 36: 0.9194560669456068,
37: 0.9138061109015466, 38: 0.9756916679993604, 39:
0.9821027097384925, 40: 0.6475283213182287, 41: 0.9619577185641929,
42: 0.9691736183524505, 43: 0.8465499485066941, 44:
0.9420364637082903, 45: 0.9262151911368399, 46: 0.941204888494281, 47:
0.8822680412371134, 48: 0.9106940777065259, 49: 0.9318885448916409,
50: 0.9622989070367186}\n"
      ]
      },
      {
      "data": {
      "image/png": "\n",
      "text/plain": [
      "<matplotlib.figure.Figure at 0x15aa88895c0>"
      ]
      },
      "metadata": {},
      "output_type": "display_data"
      }
   ],
   "source": [
      "# Print roc curve\n",
      "show_roc_curve(video_model, X_test, y_test)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
      "# Evaluation - Blind Set\n",
      "The last pickle file is held as blind set."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 19,
   "metadata": {},
   "outputs": [],
   "source": [
      "# Load blind dataset\n",
      "X_blind, y_blind = get_blind_by_indices(datadir, 0,
blind_idxs[0])\n",
      "for i in range(1, num_pickle_files):\n",
      "     Xblind, yblind = get_blind_by_indices(datadir, 0,
blind_idxs[0])\n",
      "     X_blind = np.vstack((X_blind,Xblind))\n",
      "     y_blind = np.vstack((y_blind,yblind))\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 20,
   "metadata": {},
   "outputs": [
    {
    "name": "stdout",
    "output_type": "stream",
    "text": [
    "Accuracy score: 52.25%\n"
    ]
    }
 ],
 "source": [
    "# Print accuracy score\n",
    "accuracy = get_accuracy(video_model, X_blind, y_blind)\n",
    "print(\"Accuracy score: {:.2f}%\".format(accuracy*100))"
 ]
},
{
 "cell_type": "code",
 "execution_count": 21,
 "metadata": {},
 "outputs": [
    {
    "data": {
    "text/plain": [
    "True"
    ]
    },
    "execution_count": 21,
    "metadata": {},
    "output_type": "execute_result"
    },
    {
    "data": {
    "image/png": "\n",
    "text/plain": [
    "<matplotlib.figure.Figure at 0x15aa60adb38>"
    ]
    },
    "metadata": {},
    "output_type": "display_data"
    }
 ],
 "source": [
    "# Print confusion matrix\n",
    "show_confusion_matrix(video_model, X_blind, y_blind)"
 ]
},
{
 "cell_type": "code",
 "execution_count": 22,
   "metadata": {},
   "outputs": [
      {
      "name": "stdout",
      "output_type": "stream",
      "text": [
      "{0: 0.91888836552049, 1: 0.9718644739449177, 2:
0.8688559084277858, 3: 0.9513532031517642, 4: 0.9510740601973273, 5:
0.9307486400138157, 6: 0.8975861545468347, 7: 0.9579814624098867, 8:
0.9737579242223204, 9: 0.8253347064881565, 10: 0.8658342004856052, 11:
0.9400111830018373, 12: 0.9804036458333334, 13: 0.959013959013959, 14:
0.7602669404517454, 15: 0.8738401142041399, 16: 0.9249594574671973,
17: 0.9935699588477366, 18: 0.8784423865189009, 19:
0.9328703703703703, 20: 0.9059413238979803, 21: 0.967712283120763, 22:
0.9725618631732168, 23: 0.7653305926687222, 24: 0.8256252141144227,
25: 0.9128549647417572, 26: 0.9629320464327638, 27:
0.8770339855818743, 28: 0.8870442708333333, 29: 0.8731204943357364,
30: 0.9359384834304793, 31: 0.9862598144182726, 32:
0.8106995884773662, 33: 0.9489648033126293, 34: 0.9175213675213675,
35: 0.9664256198347108, 36: 0.9546883933676387, 37:
0.9476361153780509, 38: 0.9889004509191813, 39: 0.980902290790089, 40:
0.8391752577319588, 41: 0.9177734375000001, 42: 0.9387842728332209,
43: 0.9784394250513347, 44: 0.8508591065292096, 45:
0.9210566266889327, 46: 0.9423110151187905, 47: 0.7213326446280992,
48: 0.8983402489626556, 49: 0.9727926078028747, 50:
0.9532591046160984}\n"
      ]
      },
      {
      "data": {
      "image/png": "\n",
      "text/plain": [
      "<matplotlib.figure.Figure at 0x159c7c42898>"
      ]
      },
      "metadata": {},
      "output_type": "display_data"
      }
   ],
   "source": [
      "# Print roc curve\n",
      "show_roc_curve(video_model, X_blind, y_blind)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": []
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": []
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": []
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": []
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Python 3",
   "language": "python",
   "name": "python3"
  },
  "language_info": {
   "codemirror_mode": {
      "name": "ipython",
      "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.6.4"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 1
}
```