ABSTRACT

LIP READING PROJECT WITH PYTHON

Pınar Başak ÇALIŞKAN and Alp ERKEN

There are many methods and tools to communicate in today's world. However, the most effective way to communicate is to speak. How important acoustic information is still visible by everyone. However, in these communication steps, acoustic information may not always be transmitted properly to the receiver. The main purpose of this project is to make predictions on the acoustic information by using visual information in the videos where such situations occur. So, it's intended to predict words that people who don't have any acoustic components in the video just say from their mouth and lip movements.

A number of algorithms were used to make these predictions. The system consists of a combination of these algorithms, which are completely different tasks and functions. First, the system receives the visual input as the video by the user. It captures the captured video from the video at certain time intervals by looking at its properties such as size, duration, resolution. However, these images are not full images that have been cut off from the video. These images are only images that show the mouth and its environment. These saved images are merged in the background and saved as a larger image file. This system, which was previously trained with a large dataset with machine learning, examines this large image and begins to compare it. Finally, after this comparison phase, the system tries to guess the word that is said in this input video depending on the human lip movements. Returns the highest probability word output, although there are many results in estimates. All the studies and algorithms were realized in Anaconda and Python environment. Lip reading was examined in detail by learning the machine based on basic ways such as lip detection, lip tracking.

ÖZET

PYTHON İLE DUDAK OKUMA PROJESİ

Pınar Başak ÇALIŞKAN ve Alp ERKEN

Günümüz dünyasında iletişim kurmak için birçok yöntem ve araç vardır. Ancak hala en etkili iletişim kurma yolu konuşmaktır. Sessel bilginin ne kadar önemli olduğu hala gözle görülebilmektedir. Ancak bu iletişim adımlarında sessel veri her zaman alıcıya düzgün bir şekilde iletilmeyebilir. Bu tip durumların yaşandığı videolarda görsel veriyi kullanarak sessel veriyi tahmin etmek bu projenin ana amacıdır. Yani, videoda herhangi bir ses komponentı olmayan insanların sadece ağız ve dudak hareketlerinden söyledikleri sözcükleri tahmin etmek amaçlanmıştır.

Bu tahminlerin yapılabilmesi için bir takım algoritmalar kullanılmıştır. sistem tamamen farklı görevleri ve işlevsellikleri olan bu algoritmaların bir araya getirilmesinden oluşmaktadır. İlk olarak, sistem kullanıcı tarafından görsel girdiyi video olarak alır. Sonrasında alınan bu videoyu boyut, süre, çözünürlük gibi özelliklerine bakarak belirli zaman aralıklarında videodan görüntüler alarak kaydeder. Ancak bu görüntüler videodan kesilmiş tam görüntüler değil. Bu görüntüler sadece insanın ağız ve çevresini gösteren görüntülerdir. Kaydedilen bu görüntüler arka planda birleştirilerek daha büyük bir görüntü dosyası olarak kaydeder. Daha öncesinde makine öğrenmesi ile büyük bir very tabanı ile eğitilmiş bu sistem, bu büyük boyutlu görüntüyü inceler ve karşılaştırmaya başlar. Son olarak bu karşılaştırma safhasından sonra sistem bu girdi video içindeki insanın dudak hareketlerine bağlı olarak söylenen sözcüğü tahmin etmeye çalışır. Tahminlerinde birçok sonuç olmasına rağmen en yüksek olasılıklı olan sözcüğü çıktı olarak verir. Tüm çalışmalar ve algoritmalar Anaconda ve Python ortamında gerçekleşmiştir. Dudak

algılama, takip etme gibi temel yollardan yola çıkarak makinesi öğrenmesi ile dudak okuma detaylı bir şekilde araştırılmıştır.

TABLE OF CONTENTS

ABBREVIATIONS

Adam : Adaptive Moment Estimation
ANN : Artificial Neural Network
App : Appendix
BGR : Blue Green Red Model
CNN : Convolutional Neural Network
cv2 : OpenCV
Etc : Et Cetera
FPS : Frame per Second
json : Java Script Object Notation
ML : Machine Learning
DL : Deep Learning
np : Numpy
npy : NumPy Array
plt : Plot
RNN : Recurrent Neural Network
txt : Unformatted text
tar.gz : Packed Type (Tape Archive)
val_acc : Validation Accuracy
val_loss : Validation Loss

SYMBOLS

C : Capacitance
H : The amount of heat
$\mathbf{M_x, M_y, M_{xy}}$: Torque Components
$\mathbf{N_x, N_y, N_{xy}}$: Normal Power Components
q : Phase load
t : Time
u,v : Displacement Vector Components
w : Angular velocity
XC : Capacitive reactance
XL : Inductive reactance
α : Angle of deviation from the direction of the principal stresses
ρ : Density
$\sigma_x, \sigma_y, \sigma_{xy}$: Shell internal stresses

LIST OF FIGURES

1. INTRODUCTION

1.1. Problem Statement

Wherever we go in the world of today, all we need to do is communicate. Communication is the most important way that all living creatures interact with each other. For people, the most important way of communication is to speak. This communication can also be provided in a technological and virtual environment. Both visual and audio data are present in the videos where this communication takes place. Under the leadership of these data, people can perceive and make sense of the video. However, each video may not be as it should be. For example, a video has only visual data, but what happens if there is no audio data? The aim of this project is to provide a complete understanding of the dialogue in the video despite the lack of audio data in the video. This is a project based on estimating the words spoken by people in the video by adhering to their lip movements. So, it is intended to fill the gap in communication by predicting what the person in the video says in the video from the visual data alone. Some noises in the background of the person speaking in the video also cause problems in the transmission of audio data. In addition, the person's voice data in the video may not be transmitted to the recipient due to system-sourced errors. With this project, it is aimed to minimize the impact of such problems on communication.

This project can be used by many people. Because these problems can be encountered in any environment or platform. And it would be a very good solution to use this project to eliminate or minimize the effects of these problems. In particular, there can be many different groups of people, institutions and organizations that will use this project. First, it is expected that this project will be used by people who do not have the most hearing skills (hearing impaired people or old people). With this program, people who are hearing impaired person can have the necessary information from videos that do not have any sound content. In addition, hearing impaired people can use this program to communicate with each other in a virtual environment with live video. And if this project is further developed, they can make their lives easier by using this project in many parts of their daily lives. In addition, this project is expected to be used by institutions that provide security for the country and the public. For example, a guilty person has a video. However, the video took far away from the guilty person and their speaks do not reach to the device or there are deficiencies or noises in the audio data of the video. Thanks to this project, lip reading can reveal what that person is saying. That way, before a worse action takes place, that man can be captured based on what he says. When this project

is further developed, more customer and customer groups are expected to emerge. The use of this project will affect the lives of many people in a good way. And the use of this project will be able to see in many different areas and platforms.

1.2. Background Research

There are several reasons to choose this topic. Before starting to the project, we made a comprehensive and detailed article researches for this project. At the end of these researches, not only articles about lip-reading and tracking also the face recognition articles were beneficial to understanding the principle of deep learning. The main process and working disciplines of face recognition and lip detection are similar. For example, the way of how to recognize lip and face alike each other. The articles that are examined, published in recent years. The problems or problem-solving ways not different from each other. The main problem of the articles that were worked on, is imitating human behaviours. Because every person cannot do read lips with eyes, but in this project, the end of the project, the system can complete the process with the help of deep learning and machine learning tools.

One of the most beneficial articles is **"Lip Reading in the Wild"** by Joon Son and Andrew Zisserman, written in 2017. This article is the main article because they just try to do the lip reading mechanism as we want to do. Also, they have the biggest dataset and they achieved the highest success rate. Also, the software developers who are worked with this project are the founder of Theano. Theano is a deep learning library developed for training deep neural network algorithms in 2007. However, Theano did not improve because the founders started to work with Google and they changed this library's name as TensorFlow. Then, people always use new technology and they use TensorFlow. Therefore, we believe that the best lip-reading project is created with this team and it shows the result of this project's success.

This article is written in 2017 by Amirsina Torfi, Seyed Mehdi Iranmanesh, Nasser M. Nasrabadi and Jeremy Dawson named as **"3D Convolutional Neural Networks for Cross Audio-Visual Matching Recognition"**. The main purpose of this article is completing the missing information for necessary cases. However, the approach of this is similar to the first article and in this article, the 3D convolutional neural network is used. This method helps to specify the solution way of this project.

The last article is written in 2009 by Rainer Stiefelhagen, Uwe Meier, Jie Yang and named as **"Lips detection for audio-visual speech recognition system"**. In this article, Siew Wen Chin, Li-Minn Ang and Kah Phooi Seng. Their motivation is "Hearing lips and seeing voices" [6]. Therefore, their main aim is presenting an easy life for humans. This article has a

difference from the others because they use supervised learning models to train. This model is used for machine learning and the only focus is not only deep learning. If the algorithm of a solving problem can change, the success rate can change and the main aim of this article is detecting lips and categorize these. Then, this article can separable from others.

Deep learning and machine learning algorithms are different and they can be seen each other as a rival. Therefore, this is a challenging part of this project because two of them are well known to choose one of them.

1.3. Motivations

Before choosing this project, we interested in deep learning and projects. We saw some projects about object detection and we wanted to try to find mouth. Therefore, we observed lots of projects and we recognized a challenge of these projects. This challenge was tracking the lips so, in this project we not only found lips also tracked the lips. After that part, we wanted to improve the project one more step and we add lip-reading. This challenge is solved by using deep learning and it is not totally resolvable. However, people try to achieve a success rate with different datasets. In BBC's PDF, they have the biggest dataset and they achieved the best prediction success, so, this ever-developing challenge is impressive for us.

1.1. Project Timeline

No#	Work to be Done	MONTHS									
		08/ 2018	09/ 2018	10/ 2018	11/ 2018	12/ 2018	01/ 2019	02/ 2019	03/ 2019	04/ 2019	05/ 2019
1	Determining the Subject	X									
2	Literature Review		X	X	X						
3	Specifying libraries which will be used			X	X	X	X	X			
4	Learning Deep Learning algorithms				X	X	X	X			
5	Improving the code					X	X	X	X	X	X
6	Senior Project Report										X
7	Senior Project Presentation										X

2. PROPOSED METHODOLOGY

2.1. Architecture and Environment

Python programming language was used in this project. Python version 3.7 was used in Anaconda. In addition, during this project, the coding part was completed and executed using Jupiter-Lab. Many libraries were needed to run different phases of this project. The libraries we use are as follows.

- Keras
- OpenCV
- Tensorflow
- Pip
- Dlib
- Imultils
- Numpy
- Matplotlib
- Graphviz

We downloaded all these libraries by using code;

- pip install …(library name)
- conda install -c anaconda …(library name)

No extra hardware was used in this project. The only hardware used is the laptop's camera.

2.2. Requirements Specification

The system in the project works completely with the inputs entered by the user. These entries must be in the video form. This video capture is the first step of the system, and this is the only step that needs to be taken by the user. After receiving this input (video), the video is divided into equal time intervals with the necessary algorithms. These split sections are saved as frames in the background after the program is run. These saved frames are specific but remain in a temporary folder until the next input's frames arrive. When a new input's frames arrive, frames of the new input are overwritten and lost on these old frames. However, these frames

are not all images of the video. These frames are only images where the lips and the surrounding area are cut off from the video. In other words, it is impossible to see the background, the entire face and the other components in the video in these frames. Thanks to the dlib library, lip tracking is done and frames are taken during this tracking. During frame recording in this system, frames .jpg or .png are not saved but are saved as .npy (numpy array). Numpy array is a 3-dimensional array that contains 3 different components.

These frames are then compared with previously trained data. This is the most difficult and long-lasting part of the system. In this lip reading project, The Oxford-BBC Lip Reading in the Wild (LRW) Dataset was used for the train section. The dataset was created by taking 500 words from 1000 different conversations. For example, the word 'expected' consists of videos created by cutting out 1000 different utterances. And there are 500 different words, such as the word "expected" in this Dataset. However, the use of this database is not publicly available. Making contact the BBC Research & Development Department to use this dataset with permission. The necessary confidentiality and security agreement was signed for the use of the data.

Because the size of the dataset was too big like (70 GB) for the train part, all the words contained in the data are not used. It took a long time to complete this training part with limited computer components (CPU, lack of GPU). That's why a system has been set up to work on some of the randomly selected words from the dataset. For the train part, it was used 200 videos instead of 1000 video. In addition to the train part, a video set was created for validation and test part. However, this validation and test part was created to be %10 of the video number of the train part. So, 200 train for just one word, 20 validation, and 20 test videos were created and clustered as needed.

In addition, the videos recorded before running this training part were taken only by cutting around the lip and lip. Before starting the train part of the algorithm, all of the videos are recorded with the lips and lips cut off. After a long training session, the algorithm is ready to make a prediction. Also, the library of Graphviz added to take keras architecture plotting part.

The prediction was created by adhering to the previous training section and we tried to model it. It uses the model created in the training section for prediction in this section. This model routes the forecast according to the dataset being trained. Thanks to this model, it compares the images taken from previous datasets with the last input and gives us the resulting

estimate as output. Thanks to the previously prepared model, this prediction takes much faster than the training part.

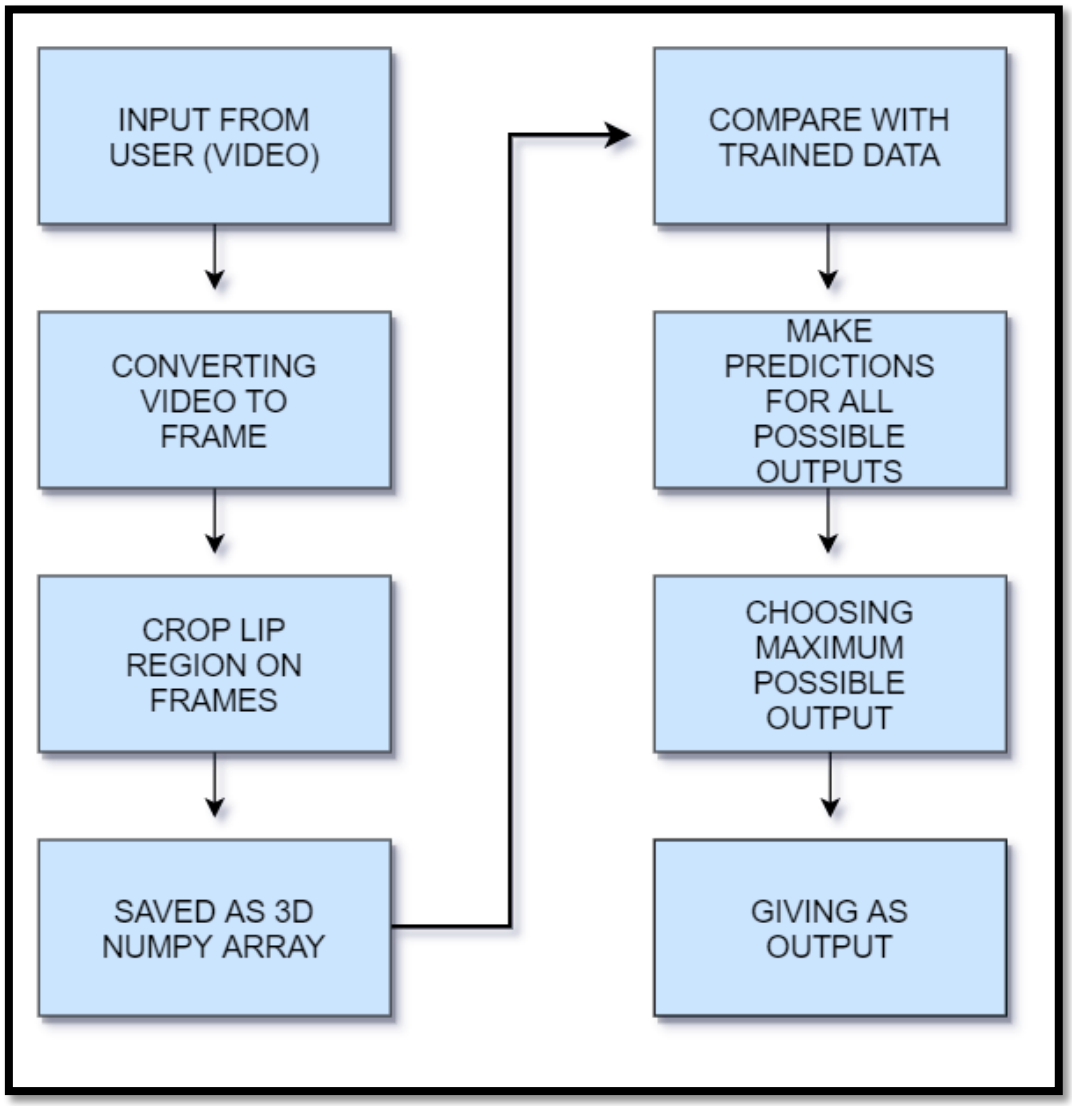

Figure 1: Diagram of the System

2.3. System Design

The first step of solving this problem is training model. In this part, videos which are selected in LRW Dataset in BBC are used. All videos could not use in this project because the training step takes a long time so, some of them were used. Beginning of the code, libraries were imported which are given at 2.1 and 2.2. Then, the videos are imported the source folder 'c:\\lrw_dataset_video'. Also, these videos had to convert frames for storing as NumPy array to take a small space. These frames include frames of lips and in this project, lips were detected with using dlib library. After that, the selected words were given with array as 'selected = []'.

The first part is ‘Data Preprocessing’. In this part, with the help of ‘convert_video_to_3d_array’ function, the videos were imported from the folder. Also, the function is defined as def () in Python. Then, an empty array was created as lipsFrames = [] to store the frames after while loop. Creating a loop with ‘while' provides to capture frame by frame. Also, these frames are stored as grey with ‘gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)’ to provides gain from space with using OpenCV. After that, we used dlib to find face region and lip region.

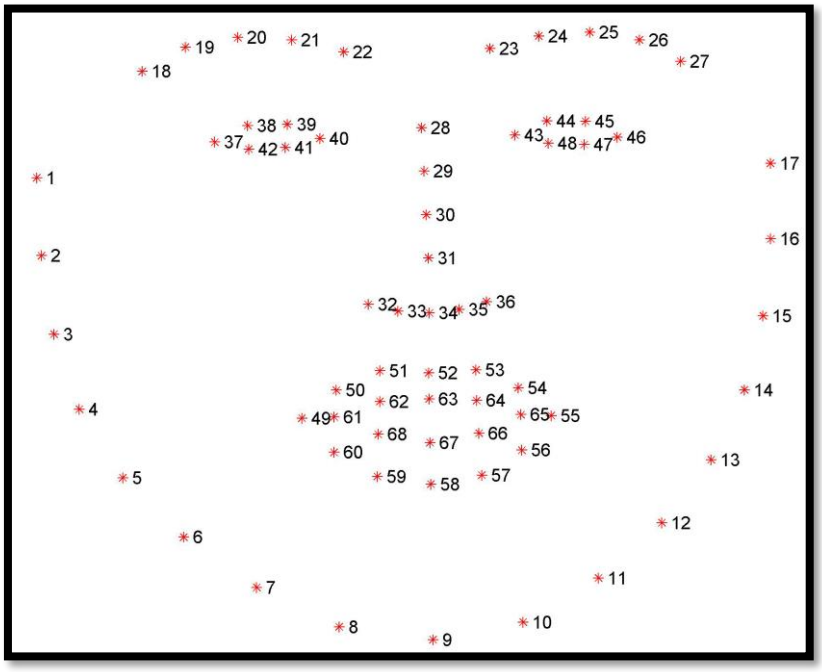

Figure 2: The Facial Landmark Coordinates with dlib

According to the figure, the cropping shape was determined. This is the important and critical part because if the lips were not determined, the code does not work properly and the video is not usable. Therefore, shape was determined to use with array format and the numbers represents the upper left corner ‘shape[14][1]’ as x coordinate, ‘shape[6][0]’ as y coordinate, and lower right corner ‘shape[11][1]’ as x coordinate, ‘shape[10][0]' as y coordinate, of the shape. The intersection of x and y coordinates gave the shape. After that part, a new function was defined as ‘crop_frames_and_save’ with calling the 1st function. In this part, cropped frames were converted NumPy array and they were saved in the target folders which were created before as ‘train’, ‘val’ and ‘test’.

```
Word: AGREEMENT, saved/skipped: 400/0
Word: BENEFIT, saved/skipped: 400/0
Word: EXPECTED, saved/skipped: 400/0
Word: CUSTOMERS, saved/skipped: 400/0
Word: CONSERVATIVE, saved/skipped: 400/0
Word: AGREEMENT, saved/skipped: 40/0
Word: BENEFIT, saved/skipped: 40/0
Word: EXPECTED, saved/skipped: 40/0
Word: CUSTOMERS, saved/skipped: 40/0
Word: CONSERVATIVE, saved/skipped: 40/0
Word: AGREEMENT, saved/skipped: 40/0
Word: BENEFIT, saved/skipped: 40/0
Word: EXPECTED, saved/skipped: 40/0
Word: CUSTOMERS, saved/skipped: 40/0
Word: CONSERVATIVE, saved/skipped: 40/0
```

Figure 3: Crop and Saving Images Reading Part of Training

The 2nd part is ‘Data Loading and Labels'. In this part, generator function was created for iterating. It provides new examples with horizontal/vertical flips, rotations etc. from using saved data to increase the success rate. In this project, the huge dataset is used and with the help of this function, the train will be more efficiency.

The 3rd part ‘Check Shape of Generated Samples' is not necessary and it controls the shape of generated samples. Also, the size should be 5, in this part, new dimensions were added with this part ‘X = X.reshape(X.shape + (1,))’.

The 4th part is ‘Train Model’. There was a new function as ‘get_model()’. This model is sequential and it is a linear stack of layers. There are many parameters, operators used and executed inside the ‘get_model()’ function. The most important of these is the 'Convolution3D' operator in which the layers are created in the training model. Convolution3D filters frames and images as three-dimension inputs. It makes these filters with the help of many parameters. One of these parameters is 'padding='same''; this parameter allows the output to be the same size as the input while the convolution process continues. 'activation='Relu' is another function of the Convolution3D operator. This function is the most widely used activation function in today's world. It is usually used in machine learning and deep learning. The Relu function can usually be used as output if a single element is requested. However, there was a sigmoid function that could use this function instead. Since sigmoid is often used in multiple output types, the Relu function was used in this part. BatchNormalization, see below for required training in a way that is in the background smaller in size and faster in a way that is used to hold the values of the function. The main purpose of BatchNormalization is to scale the values in the layers to make the training part run faster. The MaxPooling3D function increases the number of matrices

of hidden layers in the background to make the Convolution3D operator work more efficiently. In this way, the output layer ensures that it is more acceptable. This section also used the 'Dropout' function. The main purpose of the ‘Dropout’ function in neural network projects is to minimize overfitting problem. Finally, another function used in this section is 'Flatten'. The "Flatten" function is responsible for redirecting the data passing through the last layer to the output by inserting the layer in a more stable and straight state.

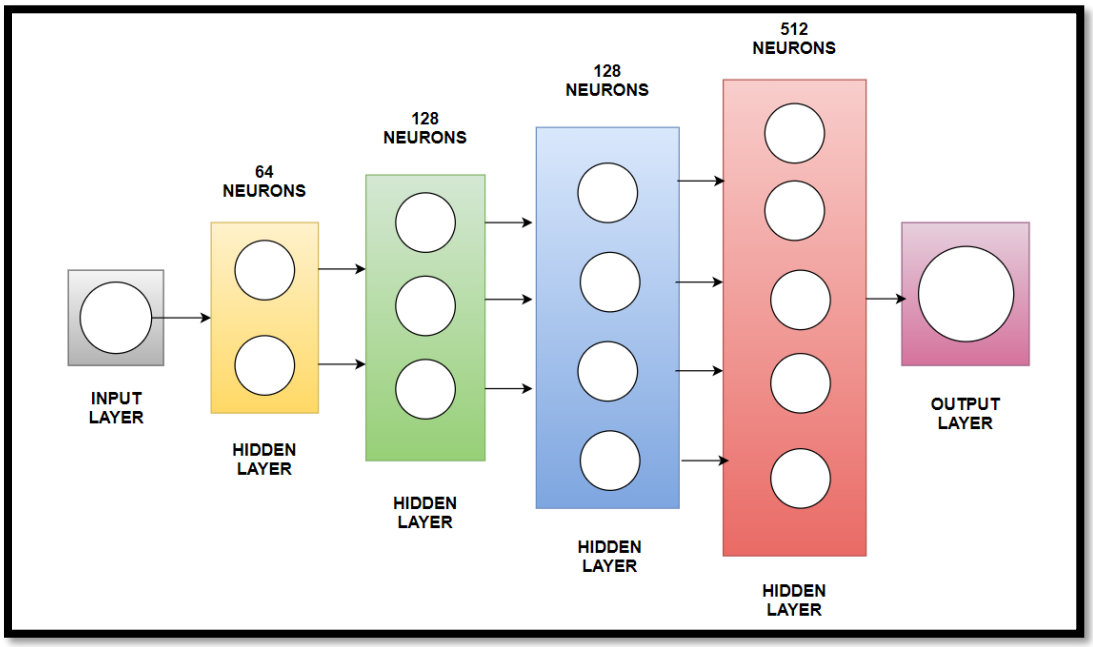

Figure 4: Visualize of Convolutional Neural Network System

The next steps are ‘Compile Model’, ‘Saving and Plotting Model Architecture’ and ‘Keras Callbacks Initialization’. Compiling model configures the model for training. ‘categorical_crossentropy' is used for loss, because in this project, there are target classes and there are 5 classes and there will be 5 dimensions, so, the next part of the code, extra dimensions will be added for using the prediction. The frames have 3 dimensions which are a row, column and depth. Also, an optimization algorithm is used to obtain better results.

```
# sizes of mouth region -> input shape
IMG_ROWS = 35
IMG_COLS = 50
VID_DEPTH = 28
```

Figure 5: The Dimensions of Mouth Region

After that part, model architecture is saved as json. The main aim of json is a data exchange that has small-size. Also, with the command of a plot, the architecture of the model

can be seen. Also, in part ‘Keras Callbacks Initialization’, Tensorboard is used. It is a new tool which is presented in 2017 by Google. With the help of this, the structure of the neural network and parameters can be shown. Also, it creates interactive graphs. In this project, Tensorboard is used only to achieve the scheme.

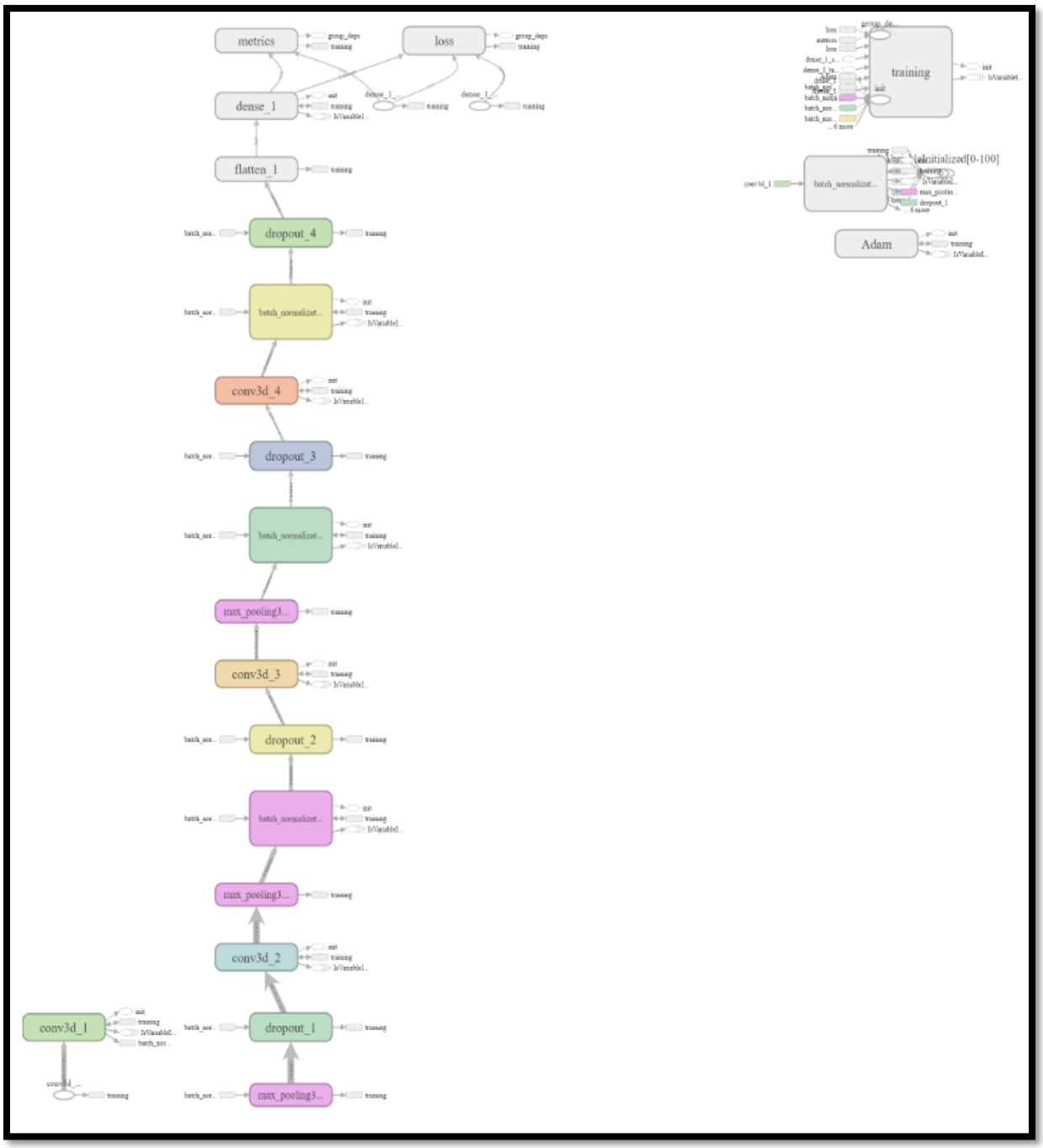

Figure 6: Tensorboard Scheme

(More detailed Tensorboard Scheme uploaded at https://drive.google.com/file/d/1ugDaefKxmfAgOx51fnNAP3l_PmakSdbs/view?usp=sharing)

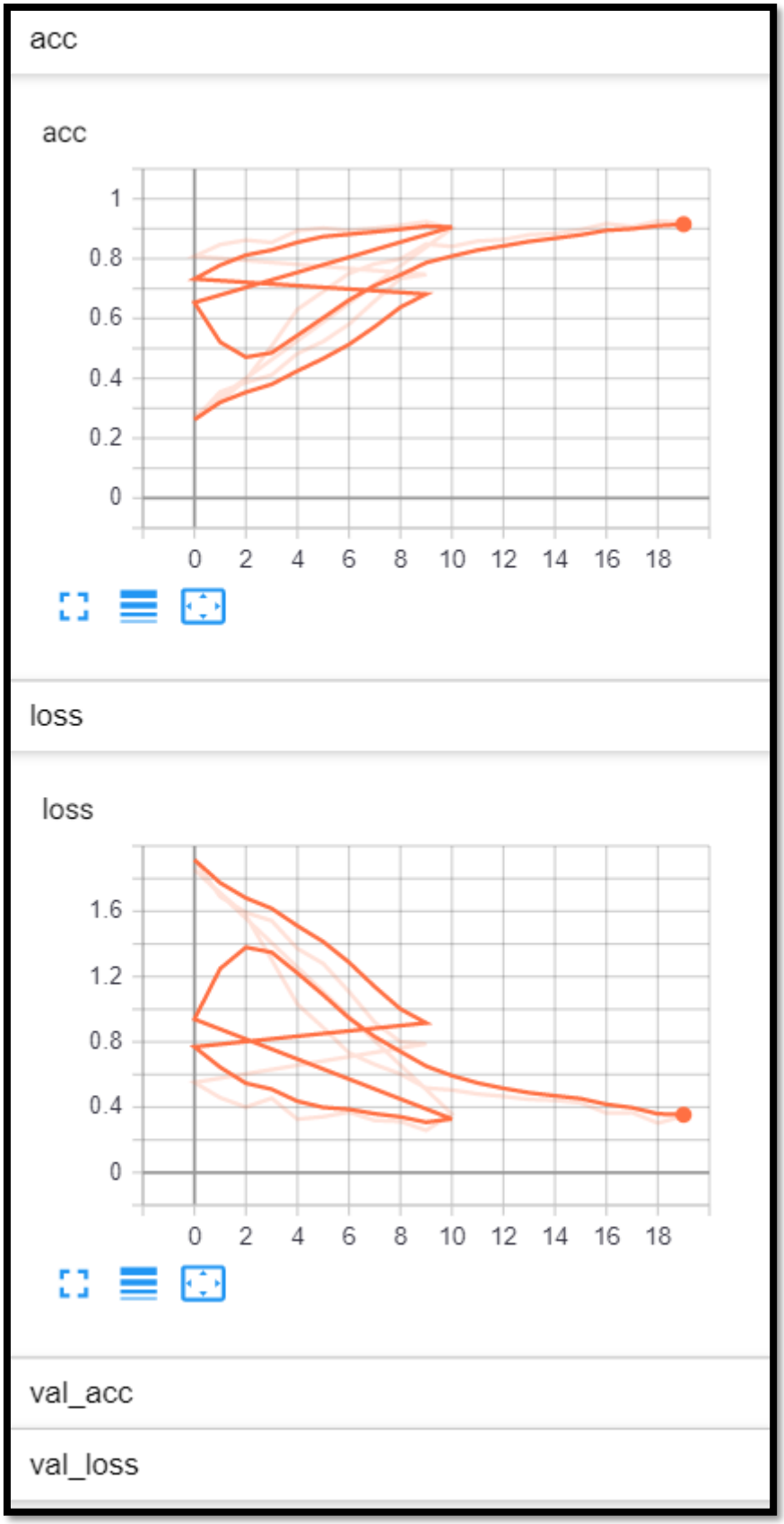

Figure 7: Tensorboard Accuracy and Loss Graphs

The other part is ‘Run Training'. In this part, the train was started with 20 epochs and batch size is the specified number of photos which are taken from RAM. It is specified according to the computers. Also, epoch is a function in which everything happens. When the epoch starts, neurons activate and they calculate loss and take partial derivatives of loss function. Then, every epoch repeats the same steps. In this part, the model was developing step by step and also the success rate is given with every epoch. In this part took nearly 12 hours to train the model because every epoch took an average 40 minutes. Then, the progress can be shown like this;

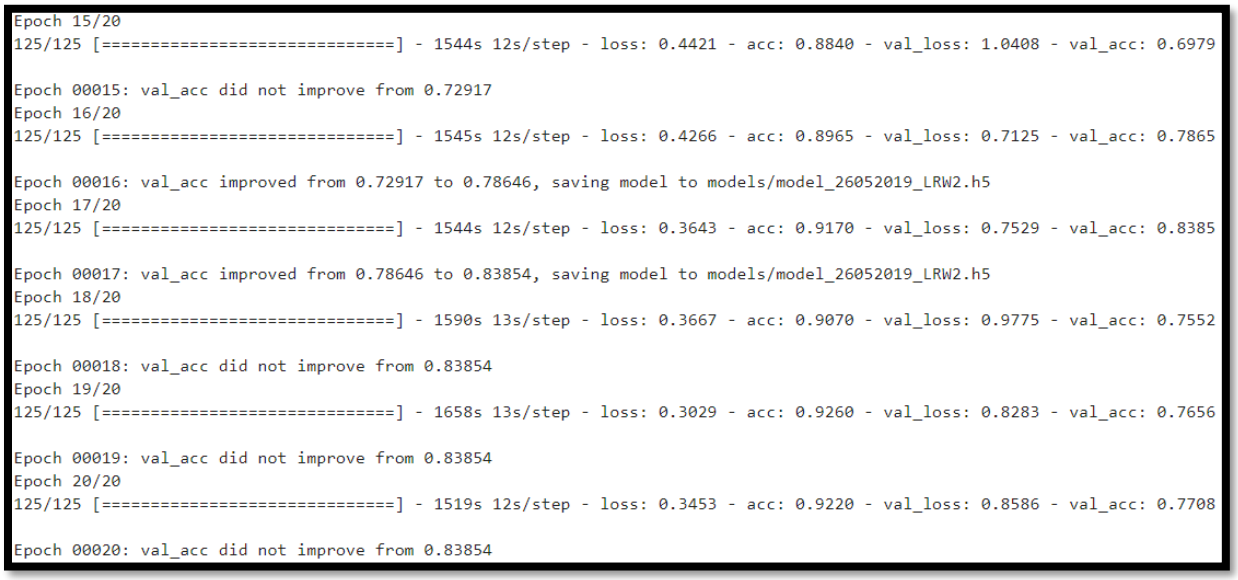

Figure 8: Shows Last 5 Epochs While Training 400 Video for Each Word

According to the figure, there are the success of every step, after the completing epoch, the system shows the next accuracy level and if the model does not learn correctly, the step gives an error like this; ‘val_acc did not improve from (value)'. Therefore, the last accuracy value is 0.83854 and it means that the model achieved 83% success rate.

The next step is ‘Plot Results after Training’ and ‘Evaluate Model on Test Data’. With using these parts, the graphs were achieved with given data such as label, title etc. There are 2 different graphs which are ‘model accuracy’ and ‘model loss’.

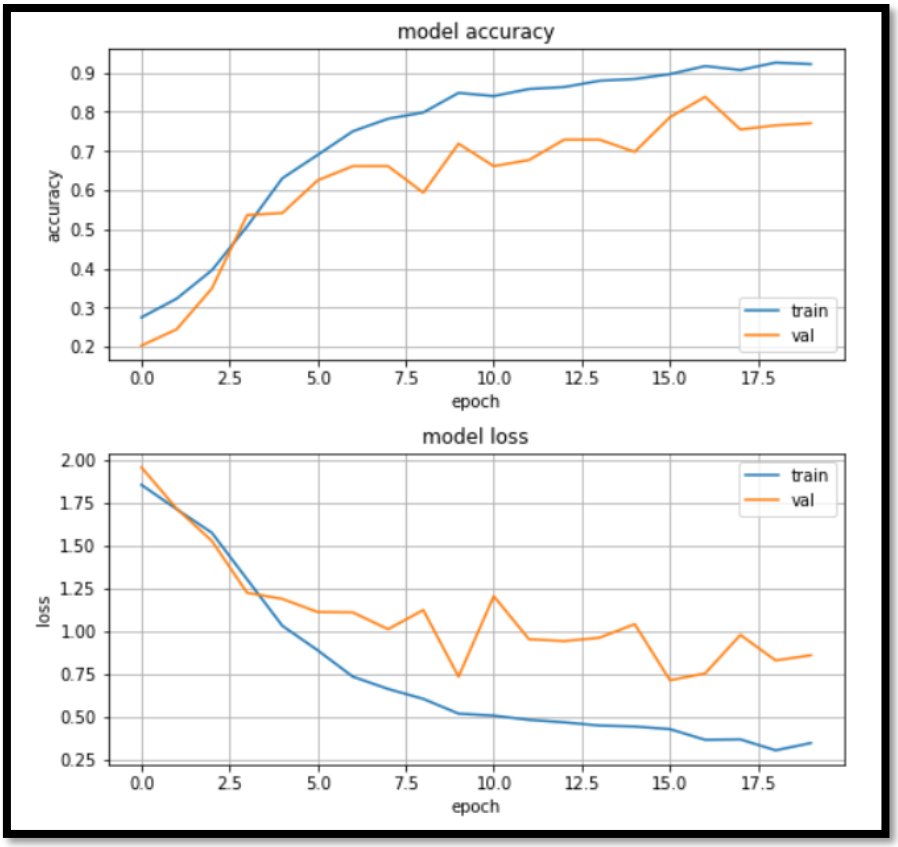

Figure 9: Plot of Model Accuracy and Loss

After obtaining the graphs and trained model, the model is saved with a name and its path also have to be given with ‘models_dir'. The model file is saved as ‘.h5' file and model weight is saved as ‘.json' file. Then, the model is loaded for training test data.

```
scores = model.evaluate(X_test, y_test)

print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
203/203 [==============================] - 27s 133ms/step

acc: 75.86%
```

Figure 10: Accuracy Rate of Test

As a result, the model used the test data and it achieved 75.86% accuracy. The code of ‘Training Model’ is determined in Appendix A.

After that, the system was tested with ‘Predict Word' part of the code. The model tested with other parts of the train videos which were not used for training. Then, the system worked as expected. The code of this part is determined in Appendix B.

Also, there is an extra step which is ‘Record Video' to create own dataset with taking videos. This step takes videos and they are converted grayscale. With the help of the dlib, lips and mouth are detected. After that, these videos can be used with the 3rd step to create a new dataset. The code is determined in Appendix C.

3. VERIFICATION AND VALIDATION

The first and the main problem was achieving the dataset which is taken from the BBC. The dataset is downloaded from the website of BBC as divided 7 parts. The extension of the files was shown ‘file' and all parts had to be combined as ‘tar.gz'. It is compressed file for Linux system. After achieved ‘tar.gz' file, the videos were extracted. These steps were done with the Linux subsystem because the computers have windows system.

The second problem was tracking the lips. The first challenge was detecting the face and it was solved with haar cascade classifiers but it was not tracking the mouth. Therefore, if it does not track the lips, the frames cannot obtain correctly because, in this project, just the lip region was used to decrease the size of frames. Thus, the frames won't use storage space as much as videos. In the first part of the project, haar cascade was used to take frames from videos. Dataset was used correctly but the last part, haar cascade did not work correctly, the system took frames nose, eyes or chin. Therefore, it is not proper to detect the lips. Then, haar cascade filters were changed with dlib. This step was the challenging part because the total system works with haarcascade and it had to be changed for work. After changing the part of the taking frames, the code worked correctly and dlib not only detect face and lips but also track the lips when people talks.

The third problem was storing the frames. In these projects, videos were 2 seconds and for every video, 29 frames were taken and these frames were combined side by side. One video has a 256x256 size and when 29 frames were combined, they created a new size 7424x256. It was a problem because if every frame has this size, the train time will take more than 20 hours. Therefore, it was changed with numpy array and the frames were stored as 3D numpy array. Also, the videos have BGR model and BGR model has 3 dimensions; and the videos have 5 dimensions like width, depth and 3 colour dimensions. However, in this project, the videos were converted BGR to grey. It means that the videos were grayscaled. Then, this process was needed some extra steps to the project. Grayscaled videos have 1 dimension and the next part of the project, extra dimensions were added to test the other videos.

The fourth problem was ‘number of used video'. Initially, 50 videos were used to train the model but it did not achieve adequate accuracy. Then, 200 videos were used for training and it took nearly 7 hours.

```
Epoch 00004: val_acc did not improve from 0.73958
Epoch 5/20
62/62 [==============================] - 732s 12s/step - loss: 0.3266 - acc: 0.8931 - val_loss: 1.1212 - val_acc: 0.6875

Epoch 00005: val_acc did not improve from 0.73958
Epoch 6/20
62/62 [==============================] - 736s 12s/step - loss: 0.3418 - acc: 0.9022 - val_loss: 1.1042 - val_acc: 0.7188

Epoch 00006: val_acc did not improve from 0.73958
Epoch 7/20
62/62 [==============================] - 731s 12s/step - loss: 0.3698 - acc: 0.8931 - val_loss: 1.1600 - val_acc: 0.7917

Epoch 00007: val_acc improved from 0.73958 to 0.79167, saving model to models/model_26052019_LRW2.h5
Epoch 8/20
62/62 [==============================] - 738s 12s/step - loss: 0.3191 - acc: 0.9012 - val_loss: 0.9623 - val_acc: 0.6875

Epoch 00008: val_acc did not improve from 0.79167
Epoch 9/20
62/62 [==============================] - 743s 12s/step - loss: 0.3128 - acc: 0.9113 - val_loss: 0.8072 - val_acc: 0.6875

Epoch 00009: val_acc did not improve from 0.79167
Epoch 10/20
62/62 [==============================] - 756s 12s/step - loss: 0.2581 - acc: 0.9234 - val_loss: 1.4127 - val_acc: 0.6771

Epoch 00010: val_acc did not improve from 0.79167
Epoch 11/20
62/62 [==============================] - 750s 12s/step - loss: 0.3587 - acc: 0.9012 - val_loss: 1.3286 - val_acc: 0.6667

Epoch 00011: val_acc did not improve from 0.79167
Epoch 00011: early stopping
```

Figure 11: Shows Result of Training with 200 Videos Each Word

```
scores = model.evaluate(X_test, y_test)

print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
100/100 [==============================] - 13s 127ms/step

acc: 66.00%
```

Figure 12: Accuracy of Test

According to the figure, the model achieved 79% accuracy and the test part achieved 66% accuracy. In the dataset, there are 1000 videos for each vocabulary and all videos for chosen vocabularies were tried to use for training. However, this process took more than 24 hours and the computers encountered blue screen error. Then, 400 videos were used to improve the model and it took 12 hours. However, the model achieved 83% accuracy. The number of videos is limited to use with standard computers.

Another problem was testing new faces which are not in the dataset. In this code, there is a part to record video and it was prepared to test and creating own dataset. However, the system's success rate decreases with unknown videos which are not included in the dataset. This problem could not solve because if the system works properly, this model will be trained with more than 400 videos. Also, taken videos' FPS should match the dataset videos' FPS. Also, this problem is just for new videos. For the people who were used for the dataset, this model works according to the accuracy rate. When the system tests with the 600 videos which are not used for train, the system works as expected.

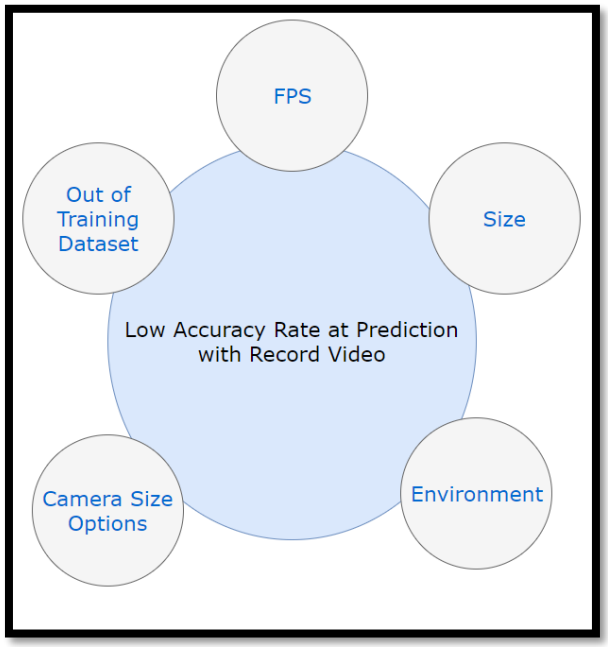

Figure 13: Low Accuracy Rate at Prediction with Record Video

4. IMPLEMENTATION

The system started with the training the model with the chosen videos for 5 categories which were the chosen words; ‘agreement', ‘benefit', ‘conservative', ‘customers' and ‘expected'. The frames were taken from the videos and they were grayscaled and stored as 3D NumPy array. Therefore, if the system cannot use the videos, skipped videos can be seen in the code and this problem can be solved with changing the video. However, the videos which are skipped cannot be seen with the name of the file. Thus, every video has to control one by one. After generated samples, the train was started. However, there was a tricky step to stop the train. This is ‘early stopping'. This can be changed according to the claim. This step provides to control the model by stopping the code. If the model encounters to ‘do not improve' feedback ‘x' times in a row.

```
earlyStopping = EarlyStopping(monitor='val_acc', patience=3, verbose=1, mode='max')
```

Figure 14: Early Stopping Code

In this code, this is 3 because every epoch takes a long time and if the model cannot improve itself, this system gets involved and the train is stopped.

After the training model, the model's accuracy can be seen as 83%. However, with using the confusion matrix which is taken from Kaggle, the success rate can be seen for each word.

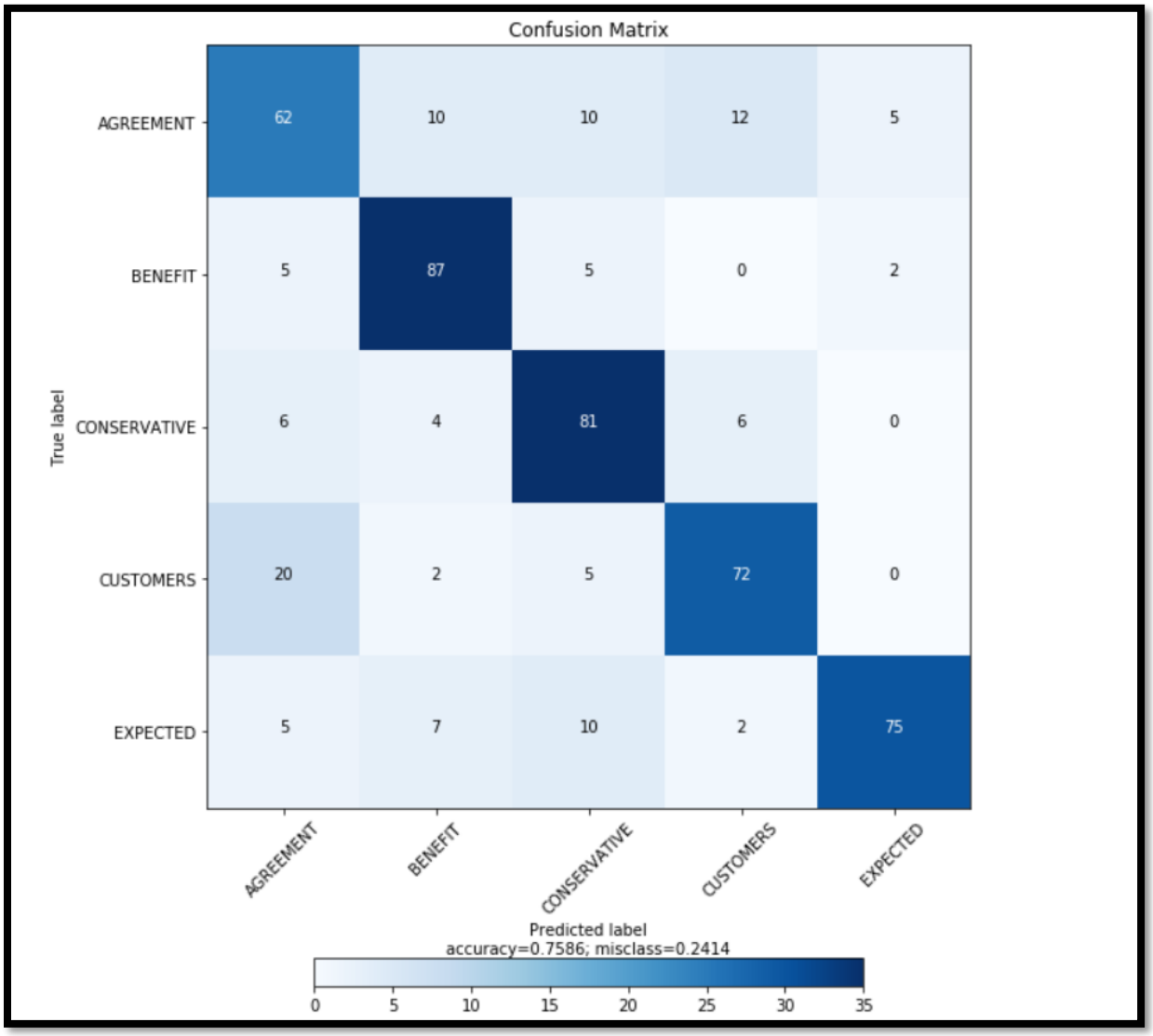

Figure 15: Confusion Matrix

According to the confusion matrix, ‘agreement’ has 62%, ‘benefit’ has 87%, ‘conservative’ has 81%, ‘customers’ has 72% and ‘expected’ has 75% accuracy according to the test data.

After the created model, the prediction part is operated with the dataset's videos. Therefore, the results are nearly expected for the model. However, all dataset does not include the frontal face of people. In this project, the videos which were chosen include a frontal face of people. Then, the test data does not give the expected results with if the videos, which includes the profile of people, are used.

5. CONCLUSIONS

Lip reading is a necessary process to improve communication in the lives of people who cannot receive any sound data. In this project, lip reading was processed in the video environment. In the video, it was aimed to improve communication among people by making word prediction by adhering to the gestures of the people's lips. Machine learning (ML) and deep learning (DL) methods were used during these procedures. With the use of a combination of functions and operators with many different backgrounds, we were able to complete our work. We have encountered many problems and difficulties during the completion of this project.

First of all, we did general research before we started the project. Theoretically, we knew what we were going to do, how we were going to do it. However, we realized that there was a significant difference between theoretical and practical, and we used many different ways and methods to complete the project. We have decided to see with our eyes and test even the question of which library, function or operator is better. In addition, we have experienced the first hand that such a large database and a large project requires a lot of system requirements. We have encountered the Blue Screen error type quite many times during both training and data delivery due to overloaded computers during this project. And this resulted in the subsequent work going on the computers that were restarted after the error. In such projects, we did not know that the system requirements were so important, but we learned. With a better system, faster and accurate results could be achieved. However, we are very pleased that we are doing projects in areas such as Machine Learning (ML), Deep Learning (DL). We believe that the experience we have gained during the completion of this project will definitely work in our future career and life.

Finally, we have experienced what professional program developers do. And we developed a program that could be the best solution to a problem that we choose. We think we've learned a lot more when we develop this program than we need to teach us. We have a better understanding and learning about machine learning, deep learning, convolutional neural

network, recurrent neural network, classifiers, functions and operators logic and working principles through our research and writing codes. During this project design an architecture, coding, Python Software language use in the field of skills developed.

REFERENCES

[1] Chung J. S.,Zisserman A. “Lip Reading in the Wild”, in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* pp. 3444 - 3453, July 2017.

[2] Torfi A., Iranmanesh S. M., Nasrabadi N., Dawson J. “3D Convolutional Neural Networks for Cross Audio-Visual Matching Recognition”, in *IEEE Access*, vol. 5, no.3, pp. 22081-22091, October 2017.

[3] Mroueh Y., Marcheret E., Goel V. “Deep multimodal learning for Audio-Visual Speech Recognition”, in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),* pp. 2130-2134, April 2015.

[4] WenJuan Y., YaLing L., MingHui D. “A real-time lip localization and tacking for lip reading”, in *2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE),* pp. 123-127, August 2010.

[5] Pingxian Y., Rong G., Peng G., Zhaoju F. “Research on lip detection based on OpenCV”, in *Proceedings 2011 International Conference on Transportation, Mechanical, and Electrical Engineering (TMEE),* pp. 1465-1468, December 2011.

[6] Chin S. W., Ang L., Seng K. P. “Lips detection for audio-visual speech recognition system”, in *2008 International Symposium on Intelligent Signal Processing and Communications Systems*, pp. 1-4, Febuary 2009.

[6] Huang Y.,Liang J.,Pan B., Fan X. “A new lip-automatic detection and location algorithm in lip-reading system”, in *2010 IEEE International Conference on Systems, Man and Cybernetics*, pp. 2402-2405, October 2010.

[7] Jun H., Hua Z. “A Real Time Lip Detection Method in Lipreading”, in *2007 Chinese Control Conference,* pp. 516-520, July 2007.

[8] Saitoh T., Konishi R. “Profile Lip Reading for Vowel and Word Recognition”, in *2010 20th International Conference on Pattern Recognition,* pp. 1356-1359, August 2010.

[9] Wang B., Zhang J. “Lip detection and adaptive tracking”, in *2017 2nd Asia-Pacific Conference on Intelligent Robot Systems (ACIRS)*, pp. 66-72, June 2017.

[10] Seyedarabi H., Wonsook L., Aghagolzadeh A. “Automatic Lip Tracking and Action Units Classification using Two-Step Active Contours and Probabilistic Neural Networks”, in *2006 Canadian Conference on Electrical and Computer Engineering*, pp. 2021-2024, January 2007.

[11] Smilefie: how you can auto-capture selfies by detecting a smile. (2018, August 7). Retrieved from https://www.freecodecamp.org/news/smilfie-auto-capture-selfies-by-detecting-a-smile-using-opencv-and-python-8c5cfb6ec197/

[12] Joshi P., Garrido G. (January 2007). OpenCV 3.x with Python By Example - Second Edition. Packt Publishing.

[13] Plot a Confusion Matrix. (2017). Retrieved from https://www.kaggle.com/grfiv4/plot-a-confusion-matrix

APPENDIX A

```
import glob

import numpy as np
import cv2
import matplotlib
import matplotlib.pyplot as plt
from random import shuffle
import imutils
import os
import time
import timeit
import itertools
import dlib

from imutils import face_utils

from sklearn.metrics import classification_report, confusion_matrix

from keras.models import Sequential
from keras.layers import Dense, Convolution3D, ZeroPadding3D, Activation,
MaxPooling3D, Flatten, Dropout, BatchNormalization
from keras.callbacks import TensorBoard, ModelCheckpoint, EarlyStopping,
CSVLogger, ReduceLROnPlateau
from keras.optimizers import SGD, rmsprop, Adam
from keras.models import model_from_json
from keras.utils import to_categorical, plot_model

np.random.seed(7)
np.set_printoptions(formatter={'float': '{: 0.3f}'.format})

# add GraphViz library for keras architecure plotting, if on Windows
if os.name == 'nt':
    os.environ["PATH"] += os.pathsep + 'C:/Program Files
(x86)/Graphviz2.38/bin/'



# source folder of video files
source_folder = 'c:\\lrw_dataset_video'
# destination folder of frames
target_folder = 'c:\\lrw_dataset_frame'

# sizes of mouth region -> input shape
IMG_ROWS = 35
IMG_COLS = 50
VID_DEPTH = 28
```

```
# initialize dlib's face detector
p = "shape_predictor_68_face_landmarks.dat"
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor(p)



# selected subset of words for training
selected = ["AGREEMENT","BENEFIT", "CONSERVATIVE", "CUSTOMERS", "EXPECTED"]
print(len(selected))



def convert_video_to_3d_array(video):
    cap = cv2.VideoCapture(video)

    cnt = 0

    lipsFrames = []

    current_video_array = None

    while cap.isOpened():
        # Capture frame-by-frame
        ret, frame = cap.read()

        if ret == True:

            # convert frame to grayscale
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

            # detect faces in the grayscale image
            rects = detector(gray, 0)

            # loop over the face detections
            for (i, rect) in enumerate(rects):
                # determine the facial landmarks for the face region, then
                # convert the facial landmark (x, y)-coordinates to a npy
                global shape
                shape = predictor(gray, rect)
                shape = face_utils.shape_to_np(shape)

            # crop_img = frame[top:bottom, left:right]
            crop = gray[ shape[14][1]:shape[11][1], shape[6][0]:shape[10][0]]
            crop = cv2.resize(crop, (IMG_COLS,IMG_ROWS))
```

```
            lips_resized = cv2.resize(crop, (IMG_COLS, IMG_ROWS),
interpolation=cv2.INTER_AREA)
            lipsFrames.append(lips_resized)


        # if video is empty, skip the sample
        else:
            break

        cnt += 1
        if cnt > VID_DEPTH:
            current_video_array = np.array(lipsFrames, dtype="uint8")
            break

    cap.release()
    return current_video_array



    # cycle over words of subset(train / val / test)
def crop_frames_and_save(set_type, words):

    # create target_folder if it doesnt exists
    if not os.path.exists(target_folder):
        os.makedirs(target_folder)

    # create TYPE folder if it doest exists
    if not os.path.exists(os.path.join(target_folder, set_type)):
        os.makedirs(os.path.join(target_folder, set_type))

    for current_word in words:

        word_files = glob.glob(os.path.join(source_folder, current_word,
set_type, '') + "*.mp4")
        saved = 0
        skipped = 0

        # cycle through word samples
        for i, video in enumerate(word_files):
            np_frame_array = convert_video_to_3d_array(video)

            if np_frame_array is not None:
                if not os.path.exists(os.path.join(target_folder, set_type,
current_word)):
                    os.makedirs(os.path.join(target_folder, set_type,
current_word))
```

```
                np.save(os.path.join(target_folder, set_type, current_word,
str(i)), np_frame_array)
                saved += 1
            else:
                skipped += 1

        print("Word: {}, saved/skipped: {}/{}".format(current_word, saved,
skipped))



crop_frames_and_save("train", selected)
crop_frames_and_save("val", selected)
crop_frames_and_save("test", selected)



nb_classes = len(selected)

labels = selected
labels_num = [i for i in range(0, len(labels))]
one_hot_labels = to_categorical(labels_num)

labels_num_dict = dict(zip(labels, labels_num))
labels_hot_dict = dict(zip(labels, one_hot_labels))



def sample_generator(basedir, set_type, batch_size):

    # Directory from which to load samples
    directory = os.path.join(basedir, set_type)

    # Placeholder NumPy arrays for features and labels
    batch_features = np.zeros((batch_size, VID_DEPTH+1, IMG_ROWS, IMG_COLS,
1))
    batch_labels = np.zeros((batch_size, nb_classes), dtype="uint8")

    file_list = []
    # Populate with file paths and labels
    for word_folder in labels:
        file_list.extend((word_folder, os.path.join(directory, word_folder,
word_name)) \
                         for word_name in os.listdir(os.path.join(directory,
word_folder)))

    while True:
        for b in range(batch_size):
```

```
            i = np.random.choice(len(file_list), 1)[0]

            sample = np.load(file_list[i][1])  # get random sample

            # Normalize to [-1; 1]
            sample = (sample.astype("float16") - 128) / 128
            sample = sample.reshape(sample.shape + (1,))

            batch_features[b] = sample
            batch_labels[b] = labels_hot_dict[file_list[i][0]]

        yield (batch_features, batch_labels)



a = sample_generator(target_folder, "val", 2)
s = next(a)
print(s[0].dtype, s[1].dtype)
print(s[0].shape, s[1].shape)



# load all data at once into array - e.g when evaluating the test subset
def get_all_data(basedir, set_type):

    directory = os.path.join(basedir, set_type)
    file_list = []
    for word_folder in labels:
        file_list.extend((word_folder, os.path.join(directory, word_folder,
word_name)) \
                         for word_name in os.listdir(os.path.join(directory,
word_folder)))

    shuffle(file_list)

    X = []
    y = []

    for f in file_list:

        sample = np.load(f[1])
        sample = (sample.astype("float16") - 128) / 128  # normalize to 0 - 1
        X.append(sample)

        y.append(labels_hot_dict[f[0]])

    X = np.array(X)
    X = X.reshape(X.shape + (1,))
```

```
    return (X, np.array(y))



model_name = "model"
models_dir = 'models/'



def get_model():


    model = Sequential()
    model.add(Convolution3D(64, (5, 5, 5), padding='same', activation='relu',
                            input_shape=(VID_DEPTH+1, IMG_ROWS, IMG_COLS, 1)))
    model.add(BatchNormalization())
    model.add(MaxPooling3D(pool_size=(3, 3, 3)))
    model.add(Dropout(0.2))

    model.add(Convolution3D(128, (5, 5, 5), padding='same',
activation='relu'))
    model.add(MaxPooling3D(pool_size=(3, 3, 3)))
    model.add(BatchNormalization())
    model.add(Dropout(0.2))

    model.add(Convolution3D(128, (5, 5, 5), padding='same',
activation='relu'))
    model.add(MaxPooling3D(pool_size=(3, 3, 3)))
    model.add(BatchNormalization())
    model.add(Dropout(0.2))

    model.add(Convolution3D(512, (5, 5, 5), padding='same',
activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.2))

    model.add(Flatten())
    model.add(Dense(nb_classes, activation='softmax'))


    return model



opt = Adam(lr=1e-4)

model = get_model()
model.compile(loss='categorical_crossentropy', optimizer=opt,
metrics=['accuracy'])
```

```
# save model architecture to json
model_json = model.to_json()
with open(models_dir + model_name + ".json", "w") as json_file:
    json_file.write(model_json)

# print / plot model architecture
model.summary()
plot_model(model, to_file='outputs/architecture_{}.pdf'.format(model_name),
show_shapes=True, show_layer_names=False)



tensorboard = TensorBoard(log_dir="logs/{}".format(model_name),
                          write_graph=True, write_images=True)

filepath = models_dir + model_name + ".h5"
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1,
                             save_best_only=True, save_weights_only=True,
mode='max')

earlyStopping = EarlyStopping(monitor='val_acc', patience=4, verbose=1,
mode='max')

csv_logger = CSVLogger('outputs/log_{}.csv'.format(model_name), append=True,
separator=';')

learning_rate_reduction = ReduceLROnPlateau(monitor='val_acc', patience=3,
verbose=1, factor=0.5, min_lr=0.0001)



nb_epoch = 20
batch_size = 16

num_examples = nb_classes * 400
num_val_examples = nb_classes * 40

history = model.fit_generator(
          generator=sample_generator(target_folder, "train", batch_size),
          epochs=nb_epoch,
          steps_per_epoch=num_examples // batch_size,
          validation_data=sample_generator(target_folder, "val", batch_size),
          validation_steps=num_val_examples // batch_size,
          verbose=True,
          callbacks = [tensorboard, checkpoint, earlyStopping, csv_logger]
)
```

```
def plot_and_save_training():
    plt.figure(1, figsize=(8,8))
    # summarize history for accuracy
    plt.subplot(211)
    plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='lower right')
    plt.grid()

    # summarize history for loss
    plt.subplot(212)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper right')
    plt.grid()

    plt.subplots_adjust(left=None, bottom=None, right=None, top=None,
wspace=None, hspace=0.3)

    plt.savefig('outputs/train_{}.pdf'.format(model_name))
    plt.show()



plot_and_save_training()



model_name = "model"
models_dir = 'models/'



def load_model(model_path, model_weights_path):

    json_file = open(model_path, 'r')
    loaded_model_json = json_file.read()
    json_file.close()
    loaded_model = model_from_json(loaded_model_json)
```

```
    # load weights into new model
    loaded_model.load_weights(model_weights_path)
    print("Loaded Model from disk")

    # compile and evaluate loaded model
    loaded_model.compile(loss='categorical_crossentropy', optimizer='sgd',
metrics=['accuracy'])
    print("Loaded Model Weights from disk")

    return loaded_model



model = load_model(models_dir + model_name + '.json', models_dir + model_name
+ '.h5')



X_test, y_test = get_all_data("C:\\lrw_dataset_frame\\", "test")



scores = model.evaluate(X_test, y_test)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))



predictions = model.predict_classes(X_test)
lab = np.argmax(y_test, axis=1)

cm = confusion_matrix(lab, predictions)



def plot_and_save_confusion_matrix(cm, target_names, title='Confusion matrix',
cmap=None, normalize=True):

    # Source: https://www.kaggle.com/grfiv4/plot-a-confusion-matrix

    accuracy = np.trace(cm) / float(np.sum(cm))
    misclass = 1 - accuracy

    if cmap is None:
        cmap = plt.get_cmap('Blues')

    plt.figure(figsize=(10, 10))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar(orientation="horizontal", pad=0.14, shrink=0.5)
```

```
    if target_names is not None:
        tick_marks = np.arange(len(target_names))
        plt.xticks(tick_marks, target_names, rotation=45)
        plt.yticks(tick_marks, target_names)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, "{:d}".format(int(cm[i, j] * 100)),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, "{:d}".format(int(cm[i, j] * 100)),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label', color="black")
    plt.xlabel('Predicted label\naccuracy={:0.4f};
misclass={:0.4f}'.format(accuracy, misclass), color="black")

    plt.savefig('outputs/conf_matrix_{}.pdf'.format(model_name))
    plt.show()



plot_and_save_confusion_matrix(cm, labels, title="Confusion Matrix")
```

APPENDIX B

```
import glob

import numpy as np
import cv2
import matplotlib
import matplotlib.pyplot as plt
from random import shuffle
import imutils
import os
import time
import timeit
import itertools
import dlib

from imutils import face_utils

from sklearn.metrics import classification_report, confusion_matrix

from keras.models import Sequential
from keras.layers import Dense, Convolution3D, ZeroPadding3D, Activation,
MaxPooling3D, Flatten, Dropout, BatchNormalization
from keras.callbacks import TensorBoard, ModelCheckpoint, EarlyStopping,
CSVLogger, ReduceLROnPlateau
from keras.optimizers import SGD, rmsprop, Adam
from keras.models import model_from_json
from keras.utils import to_categorical, plot_model

np.random.seed(7)
np.set_printoptions(formatter={'float': '{: 0.3f}'.format})

# sizes of mouth region -> input shape
IMG_ROWS = 35
IMG_COLS = 50
VID_DEPTH = 28


# initialize dlib's face detector
p = "shape_predictor_68_face_landmarks.dat"
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor(p)


# selected subset of words for training
selected = ["AGREEMENT", "BENEFIT", "CONSERVATIVE", "CUSTOMERS", "EXPECTED"]
print(len(selected))
```

```
model_name = "model"
models_dir = 'models/'


def load_model(model_path, model_weights_path):

    json_file = open(model_path, 'r')
    loaded_model_json = json_file.read()
    json_file.close()
    loaded_model = model_from_json(loaded_model_json)

    # load weights into new model
    loaded_model.load_weights(model_weights_path)
    print("Loaded Model from disk")

    # compile and evaluate loaded model
    loaded_model.compile(loss='categorical_crossentropy', optimizer='sgd',
metrics=['accuracy'])
    print("Loaded Model Weights from disk")

    return loaded_model


model = load_model(models_dir + model_name + '.json', models_dir + model_name
+ '.h5')


def convert_video_to_3d_array(video):
    cap = cv2.VideoCapture(video)

    cnt = 0

    lipsFrames = []

    current_video_array = None

    while cap.isOpened():
        # Capture frame-by-frame
        ret, frame = cap.read()

        if ret == True:

            # convert frame to grayscale
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

            # detect faces in the grayscale image
            rects = detector(gray, 0)

            # loop over the face detections
```

```
            for (i, rect) in enumerate(rects):
                # determine the facial landmarks for the face region, then
                # convert the facial landmark (x, y)-coordinates to a npy

                global shape
                shape = predictor(gray, rect)
                shape = face_utils.shape_to_np(shape)

            # crop_img = frame[top:bottom, left:right]
            crop = gray[ shape[14][1]:shape[11][1], shape[6][0]:shape[10][0]]
            crop = cv2.resize(crop, (IMG_COLS,IMG_ROWS))

            lips_resized = cv2.resize(crop, (IMG_COLS, IMG_ROWS),
interpolation=cv2.INTER_AREA)
            lipsFrames.append(lips_resized)


        # if video is empty, skip the sample
        else:
            break

        cnt += 1
        if cnt > VID_DEPTH:
            current_video_array = np.array(lipsFrames, dtype="uint8")
            break

    cap.release()
    return current_video_array


np_frame_array = convert_video_to_3d_array('predict_word.mp4')
np.save('c:\odev\predict_word.npy', np_frame_array)


sample = np.load('c:\odev\predict_word.npy')
sample = (sample.astype("float16") - 128) / 128  # normalize to 0 - 1
sample = np.array(sample)
sample = np.expand_dims(sample, axis=0)
sample = sample.reshape(sample.shape + (1,))



prediction = model.predict(sample)
print(prediction)

prediction_class = np.argmax(prediction, axis=1)
print(prediction_class)
print(selected[prediction_class[0]])
print(prediction_class[0])
```

APPENDIX C

```
# import the necessary packages
from imutils import face_utils
import dlib
import cv2
import time
import numpy as np

# This is our python code to record video when saying words
# It records 2 seconds and then stops.
# Saves to the same folder with this script run.


# initialize dlib's face detector
p = "shape_predictor_68_face_landmarks.dat"
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor(p)

cap = cv2.VideoCapture(0)
count = 0

# Define the codec and create VideoWriter object
#fourcc = cv2.VideoWriter_fourcc(*'XVID')
#out = cv2.VideoWriter('predict_word.avi',fourcc, 13.0 (FPS of cam),
(1024,768))


fourcc = cv2.VideoWriter_fourcc(*'MP4V')
out = cv2.VideoWriter('predict_word.mp4', fourcc, 13.0, (640,480))

while count < 29:
    # load the input image and convert it to grayscale
    _, image = cap.read()
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # detect faces in the grayscale image
    rects = detector(gray, 0)

    # loop over the face detections
    for (i, rect) in enumerate(rects):
        # determine the facial landmarks for the face region, then
        # convert the facial landmark (x, y)-coordinates to a npy
        shape = predictor(gray, rect)
        shape = face_utils.shape_to_np(shape)


        cv2.rectangle(image,(shape[6][0], shape[14][1]),
(shape[10][0],shape[11][1]), (0,0,255), 2)
```

```
    # show the output image with the face detections + facial landmarks

    # crop_img = frame[top:bottom, left:right]
    # crop = image[ shape[52][1]:shape[57][1], shape[48][0]:shape[54][0]]


    out.write(image)
    cv2.imshow("Output", image)
    count += 1
    k = cv2.waitKey(5) & 0xFF
    if k == 27:
        break

cv2.destroyAllWindows()
out.release()
cap.release()
```