# 1 Introduction

This summer training report is dedicated to demonstrate the work done on BITES - Defence & Aerospace Technologies at the department of artificial intelligence that focused on machine learning and deep learning. Machine learning and deep learning are interesting subjects that gains value around the globe that propose many job opportunities for electrical and electronics engineers. In this sense, this place was chosen for the summer training since there was a department fully focused on developing machine learning and deep learning solutions to various problems at many fields. To briefly summarize the work done, **2 different projects** were worked on. They were **Lip Reading based on Visuals** and **Avionic Predictive Maintenance**.

**In the Lip Reading Project**: There were videos of thousands of Turkish speakers saying 10 different words which were gathered from YouTube videos. The purpose was to acquire the highest accuracy rate of classification of these videos to these 10 classes (10 words) using machine learning or deep learning models. This project was conducted on python and it involved detailed research on articles for detecting facial landmarks from images, data pre-processing, feature extraction, deep learning models (CNN, LSTM, BiGRU, etc.), neural networks, overfitting, underfitting, regularization and related topics. Some models were created that can actually achieve lip reading to classify unlabeled data to 10 different classes. These models were evaluated and compared with each other to find the most powerful model that serves this task. For comparison and adjustment of hyperparameters of the models, MlFlow was used that is responsible for keeping track of the hyperparameters. The motivation behind this work was to acquire the highest accuracy of lip reading. The work done for Lip Reading Project was significant because every engineer in the project was focusing on different models, approaches, pre-processing methods etc. that could best achieve the lip-reading purpose and in this sense, my work had some unique methods and it helped them to get the results of various models that can be compared and evaluated with other engineers' approaches. This was the major project that has been dealt with during the internship.

**Predictive Maintenance Project:** Predictive maintenance refers to analyzing the condition of an equipment by using data-driven approaches and helping to predict when maintenance should be performed [1]. Avionic Predictive Maintenance project's purpose was to research for predictive maintenance ways using machine learning and deep learning approaches. These methods were analyzed and compared. A power point presentation was created and the outcomes of the research was shared with other team members, engineers and the team captain. In this project the workload was divided to every engineer (some of them were finding datasets for experimenting, some were studying the important parameters for predictive maintenance, etc.) so finding suitable methods for predictive maintenance was the sole purpose of the project and it used other engineers' findings. In this sense the work done was significant for the project. Machine learning and Deep learning researches were done and presented to others. This was a minor part of the internship through the end of it.

Rest of the report is organized as company information, work done, performance and outcome, conclusions and references with their subsections. Most of this report was made up of work done section. This section was also divided into subsections. There were 2 projects involved as explained above so 3.1 represented the first project and 3.2 represented the second project. 3.1 also divided into its subsections for different models and approaches of lip reading as 3.1.1, 3.1.2, etc. These subsections further

divided into subsections to discuss the phases that lead to creation of the models such as pre-processing, modeling, optimization of hyperparameters, etc.

# 2 Company Information

## 2.1 About the company

The summer training performed at BITES - Defence & Aerospace Technologies which is a technology-based brand located in Ankara in METU Technopolis which has a wide range of product base that will be discussed in the next section [2]. The company established in Istanbul at 2001 then continued its activities in Ankara. In 2019, ASELSAN became a shareholder with 51% of the equity [3]. Today, BITES employs hundreds of engineers and it is one of the most innovative companies in Turkey. BITES ranked among “Fastest Growing Technology Companies of Turkey” and was granted “The Deloitte Fast 50 Award” for 3 consecutive years [3]. In 2020, the company became one of the top 10 companies that make the most R&D expenditure in Turkey in the Software and Information Technologies sector and ranked among the top 50 companies in all sectors [3]. Some of the specialties of the company are Augmented Reality, Virtual Reality, Modelling and Simulation, Artificial Intelligence, Embedded Software, Image Processing, Autonomous Technologies, etc. [2], [3].

## 2.2 About the products and production systems of the company

BITES produces lots of software and hardware solutions to various problems at various fields. Some of them are listed as the following: Next-Generation Computer Based Training Systems and Synthetic Environments, 3D Virtual Maintenance Trainers, Low-Cost Synthetic Training Aids and Hardware Components, Advanced Training Management Information Systems and Mission-Planning & After Action Review-Debriefing Software Solutions and a serious game-engine driven image generator for all kinds of virtual training applications [2].

## 2.3 About your department

The summer training was performed in the artificial intelligence department of BITES. The department consisted of 8 engineers and 5 interns at the time the summer training was performed. The focus of the department was mainly on machine learning and deep learning. The department was located at METU Technopolis unlike some of the departments that are located in the Loft Office at Next Level. Engineers in the department were frequently attending to meetings with other defense companies like ASELSAN. The projects held in the department were mostly for other defense companies. There were 2 projects at the time the summer training was conducted “Lip Reading based on Visuals” and “Avionic Predictive Maintenance”.

# 3 Work Done

## 3.1 Lip Reading based on Visuals Project

Lip reading is an important concept for enhancing speech recognition in noisy or loud environments. Some of the significant aspects of the topic is that it could be used to enhance security or to help people with hearing loss to communicate [4]. Although lip reading has been performed by professionals for decades, with the improvements in Computer Vision and Deep Learning, machines have the potential to perform automated lip reading (ALR) which is the process of understanding what is being spoken based solely on the visuals [5]. This project uses machine learning to accomplish ALR by applying deep learning concepts to a non-synthetic dataset of commonly used Turkish words gathered from YouTube videos.

The goal of the project is to use implement several deep learning models for pattern recognition and classification on the given dataset that consists of frames of 10 Turkish words cut from YouTube videos that has separated to train, test and validation datasets. To see the frames been cut from one video for one word see **Appendix A**. Several deep learning approaches was experimented through project to find the method that gives the highest accuracy for classification of the frames in the test dataset to the correct class (“Afiyet Olsun”, “Günaydın”, etc.). The purpose of this project can be summarized as to find the deep learning model that give the highest test accuracy rate on the given dataset.

Initially the dataset contained 10 Turkish words: “Afiyet olsun”, “Başla”, “Bitir”, “Görüşmek Üzere”, “Günaydın”, “Hoş Geldiniz”, “Merhaba”, “Özür Dilerim”, “Selam”, “Teşekkür Ederim”. Every word contributed to a separate class. For the preparation of the dataset, 5-25 frames were cut from YouTube videos that demonstrated the pronunciation of the words (the dataset was already created and summer training did not involve creating a dataset at any point). Each word had 200-250 videos assigned and a total of 2000-3500 frames were gathered. Initially the dataset was divergent in terms of images per word since longer words like “Teşekkür Ederim” contained more images per video. Organization of the dataset further explained at **Appendix B** with visuals. After the pre-processing techniques applied this situation was solved for every deep learning model created.

Three deep learning models (or methods) were used on the dataset. They were **CNN**, **CNN + BiGRU** (pre-processing was different than just CNN), **Transfer Learning** of **VGG16**.

Before continuing with the deep learning models created, it is good to give some information about the general concepts and libraries used through the modeling process of models.

**Pre-processing:** Is the work of preparing the data in a format so that the neural network model can accept the data.

**CNN (Convolutional Neural Network) Architecture**: CNNs are neural networks that is used for recognizing and classifying features from images. They are extensively used for image classification. CNNs are used at various stages for this project [6].

**Confusion Matrix**: It is a method for summarizing the performance of a classification Algorithm in a visually appealing way.

**Overfitting:** Overfitting refers to the event that the deep learning model give good results in terms of accuracy and loss for the training data however when new data is introduced as a test dataset, the model fails [7]. Overfitting is due to the fact that the model learns the **noise** in the data as an underlying concept [7].

**Regularization:** It is the set of approaches to lessen the complexity of a neural network to hinder overfitting [7].

**Dropout:** It is the process of randomly turning off neurons in a layer with a probability P. Dropout is a regularization technique which is quite successful for preventing overfitting. Dropout achieves this success by preventing complex co-adaptations that ultimately results to overfitting [7].

**Early Stopping**: It is a form of regularization to prevent overfitting. Early stopping rules how many iterations can be run before over fitting happens by observing a desired parameter. In this project, validation accuracy was observed at all times.

**TensorFlow Library:** Google's TensorFlow is an open-source library that is extensively used for machine learning and deep learning modeling since it involves functions that ease the process of data acquisition, process of training the models, making predictions, etc. [8] TensorFlow has been used extensively in this project.

**Keras Library:** It is a deep learning library written over TensorFlow. It is frequently used since it eases the creation of neural network layers greatly [8]. Keras was used in this project to create lots of layers.

Codes of all the work done in the subsections below are given at Appendices C, D, E, F, G and H.

### 3.1.1 CNN Architecture for Modeling

#### 3.1.1.1 Pre-processing

To detect and extract the lip region from the frames, *dlib* library's *face landmark detector* was used. Figure 1 below demonstrates the facial landmarks obtained by the detector.

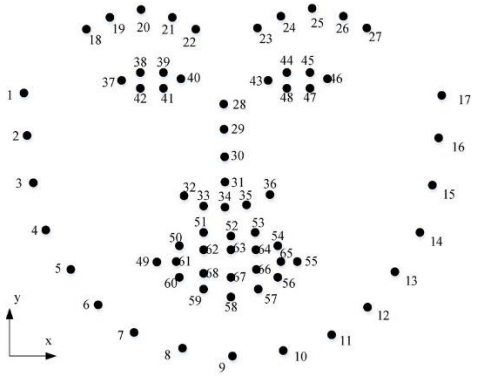


Figure 1: Facial Landmarks

Points from 49 to 68 corresponded to the lip region in Figure 1 above. Therefore, these points were extracted from the images. Figure 2 below shows one of the lip regions extracted.


Figure 2: A Lip Region Extracted

After extracting the lip regions, to be able to feed the images to a CNN network, an algorithm was created to make a collage of lip regions of every video. Every collage had some number of 50x50 lip extracted frames that is less than 25. Some of these frames used multiple times so that the collage image had exactly 25 frames. The python code served to this purpose was the following,

```
for i in range(25):
        images[i] = image_list[int(i*len(image_list)/25)]
```

This code enabled the usage of some frames multiple times depending on the number of the frames available for the specific video. ***Image_list*** was a list object and contained names of the images and ***images*** was the list created whose length is 25.

After this process, number of images dropped to approximately **2000** and they were separated to train, test and validation folders and to subfolders with their related class names to ease the modeling process. Roughly **%63** of the images was separated for **training**, **%23** of them for **testing** and **%14** of them for **validation**. Figure 2 below demonstrates a collage created. It is also worth mentioning that pre-processing method was generating corrupted images at some points and to see the effect of these corrupted images the experiment was conducted two times with and without corrupted images. Therefore, for this part, 2 datasets were used that yield different results. They were named as modified and unmodified datasets. It is important to note that these datasets are different than the original dataset as they are formed as a result of the preprocessing stage. With the help of this approach the corrupted images' effect can be distinguished. However, the real result comes from the unmodified dataset since it is impossible the delete the corrupted images for bigger dataset. Results from both of the datasets are labeled accordingly in the further sections.

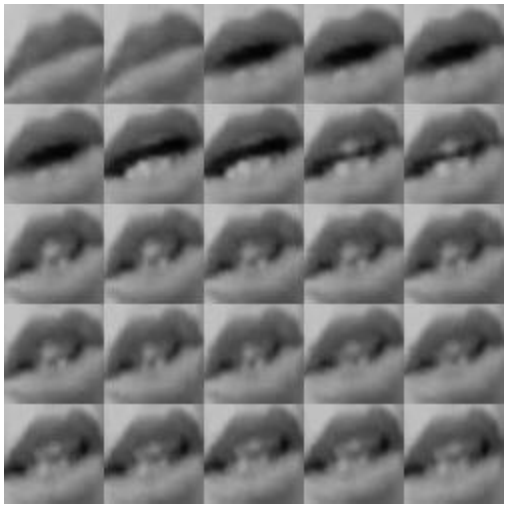


Figure 3: A Representative Collage Created

### 3.1.1.2 Modeling

The lip-reading model was trained with a CNN architecture. A 2D CNN was applied to train the pre-processed collages. The saved pre-processed image data from training set loaded in 2D CNN to train the model. The architecture was implemented using *Keras*, an open-source neural network library written in Python. Along with *Keras*, *TensorFlow* was also used [8]. *Keras* and *TensorFlow* was explained earlier at this report. To tune the hyperparameter, the **number of epochs was set to 100**, **Adam optimizer** algorithm was used. As the optimizer Adam was chosen since it showed higher accuracy rates than others at general purpose tasks. As early stopping was being used number of epochs was thought to be set to a big number and it was 100 in our case. Rather than that the kernel size was set to **5x5** since it showed greater accuracy than **3x3**. For 2D convolutional layers the activation function was chosen as ***Relu*** since it was better for simpler neural networks [9]. To be able to handle **multiple classes**, at the last layer the activation function was chosen as ***Softmax*** over ***Sigmoid*** which was better for binary classification [10].

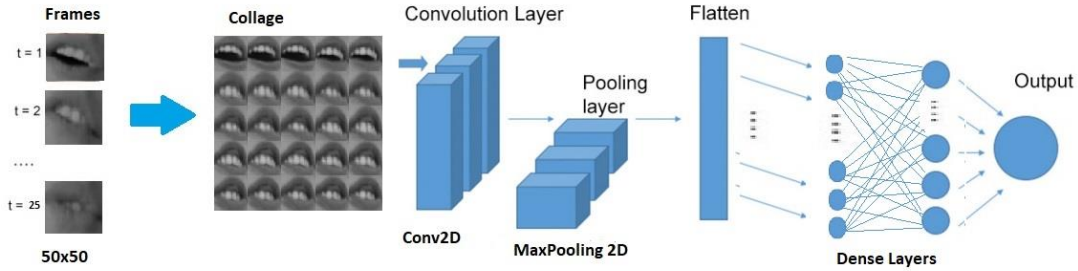



Figure 4: Model Architecture

### 3.1.1.3 Results of CNN Architecture

For the slightly modified dataset (without corrupted images):
Train accuracy: %84.74 at the last epoch 10.
Validation accuracy: %64.46 at the 7. Epoch (peak).
Test accuracy: %73 at the end of the training.

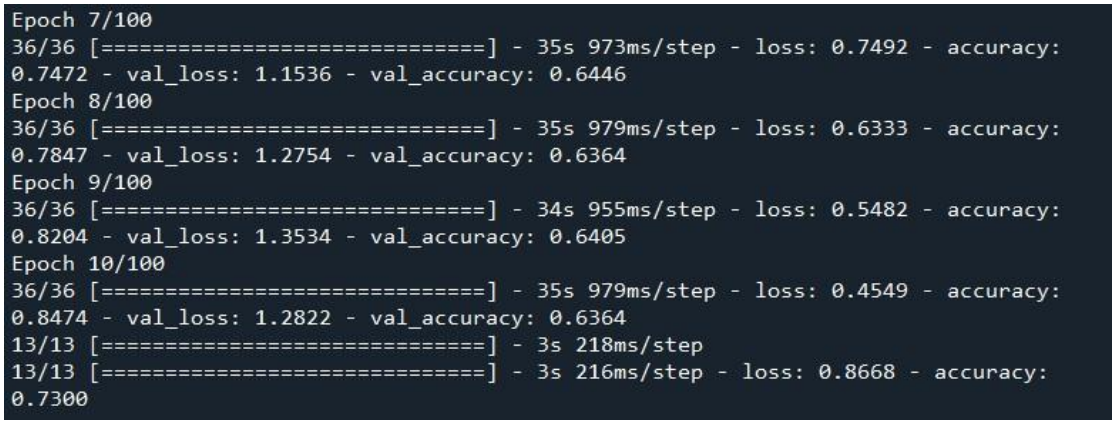



Figure 5: Results for Slightly Modified Dataset

For the unmodified dataset:
Train accuracy: %78.17 at the last epoch 16.
Validation accuracy: %60.90 (peak) at the 6. epoch.
Test accuracy: %61.65 at the end of the training.

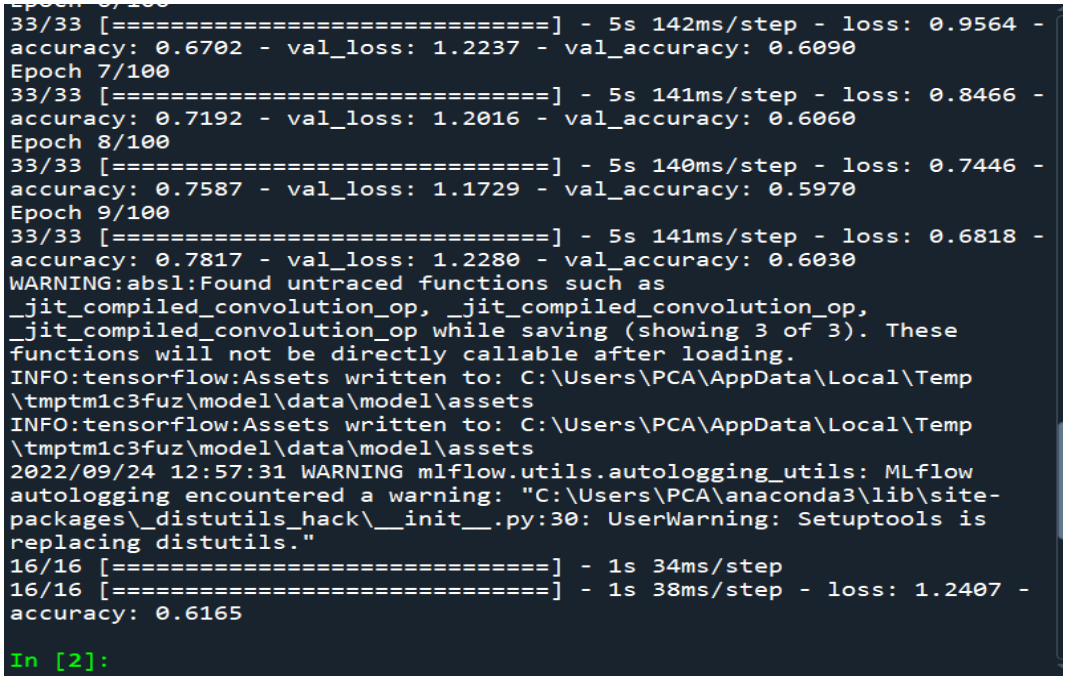



Figure 6: Results for Unmodified Dataset

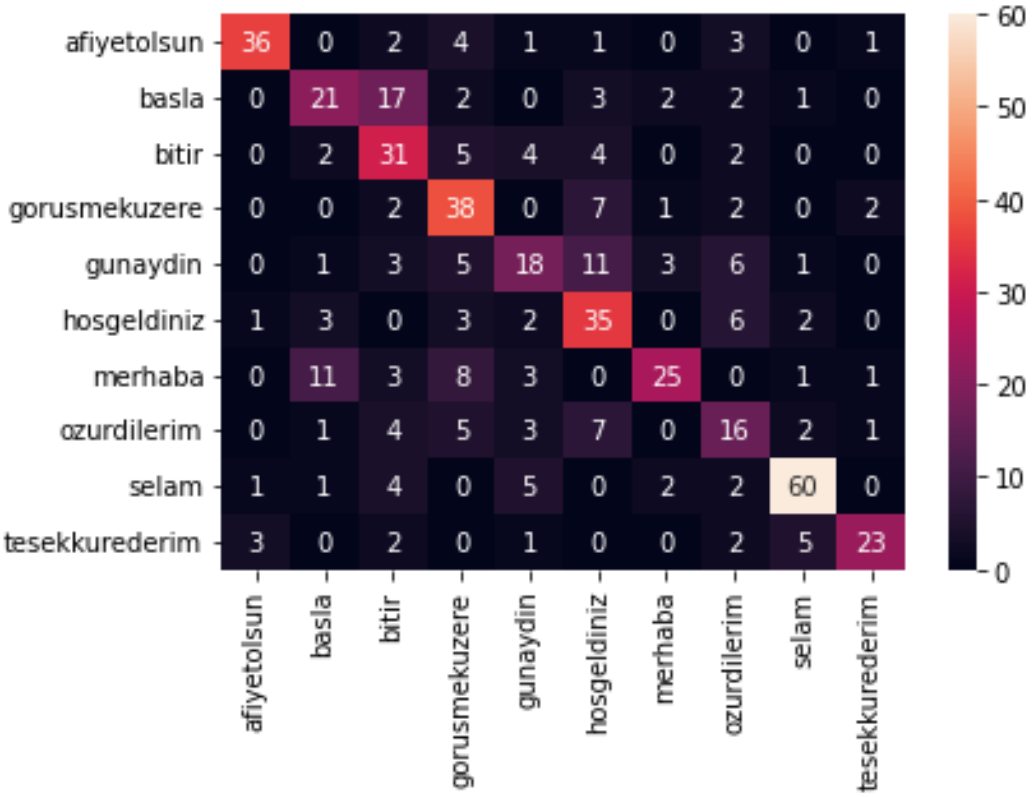



Figure 7: Confusion Matrix of Unmodified Collages

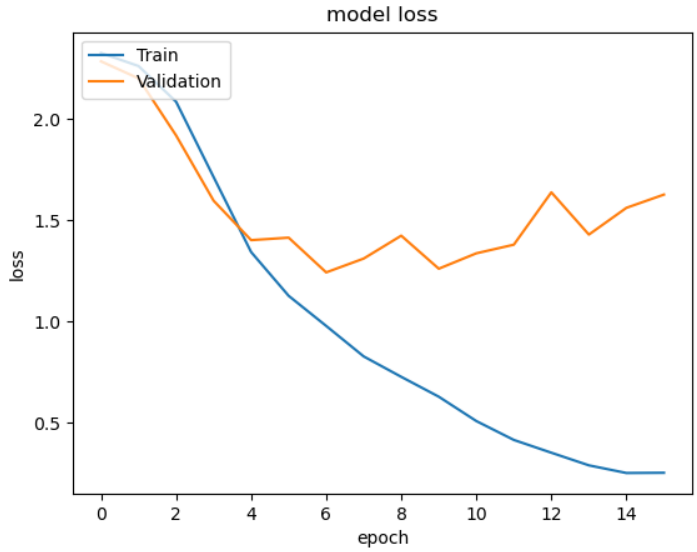



Figure 8: Unmodified Dataset Loss Graph

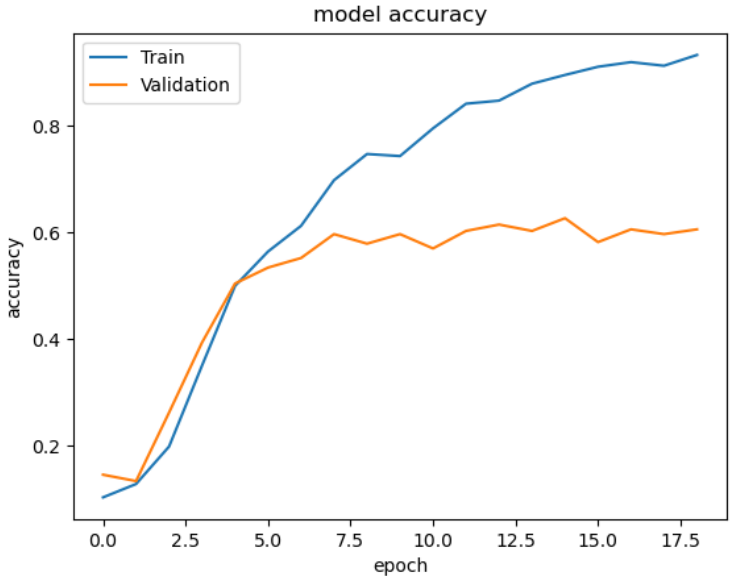



Figure 9: Unmodified Dataset Accuracy Graph

As it can be seen confusion matrix, accuracy and loss curves are included only for the unmodified dataset since modified dataset was only created to demonstrate the significant effect of corrupted images on accuracy.

When test accuracy rates of modified (73%) and unmodified (%61.65)datasets are compared it can be seen that there is a great difference which indicates that the pre-processing method could have been enhanced to increase the accuracy for this architecture.

From the confusion matrix at Figure 7, it can be seen that the model confuses the words “başla” and “bitir” with each other frequently. This result makes sense since these words’ pronunciation is quite similar. Rather than that the test accuracy is acceptable since the dataset is small. From Figure 9, it can be seen that early stopping works well as a regularization method to stop the training at the right time so that overfitting does not occur. Early-stopping was explained in the beginning of this project already.

The results at this part were before optimizing the parameters with a useful python library called “Mlflow”. In the summer training it was learnt that Mlflow is a powerful tool for machine learning and deep learning applications. During the training it was worked on how to use Mlflow. Next chapter will argue what is Mlflow and how it is a useful tool.

#### 3.1.1.4 Mlflow Integration for Unmodified Dataset to Optimize Hyperparameters

Mlflow is basically a tracking tool that allows you to track experiments to record and compare parameters and results [11]. The following figures will show the user interface and some of the experiments done with their parameters saved locally with the help of the Mlflow.

Firstly, the aim was not to optimize the hyperparameters for the modified dataset and it was worked only on the unmodified dataset since it was just to demonstrate the

remarkable effect of corrupted images coming from pre-processing stage. Thus, at this part with the help of the Mlflow, it will be tried to obtain the highest accuracy possible by making lots of trials.

| | | | | Metrics | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Start Time | Duration | Run Name | accuracy | loss | stopped_epoch | test_accuracy | test_loss | val_accuracy | val_loss |
| | 2 months ago | 7.0min | - | 0.867 | 0.401 | 12 | 0.58 | 1.601 | 0.6 | 1.635 |
| | 2 months ago | 3.0min | - | 0.84 | 0.478 | 20 | 0.606 | 1.488 | 0.633 | 1.344 |
| | 2 months ago | 1.0min | - | 0.776 | 0.736 | 28 | 0.506 | 1.614 | 0.504 | 1.609 |
| | 2 months ago | 1.9min | - | 0.933 | 0.222 | 20 | 0.627 | 1.692 | 0.63 | 1.64 |
| | 2 months ago | 2.0min | - | 0.849 | 0.439 | 21 | 0.649 | 1.368 | 0.654 | 1.38 |
| | 2 months ago | 2.6min | - | 0.805 | 0.546 | 30 | 0.606 | 1.484 | 0.633 | 1.337 |
| | 2 months ago | 1.6min | - | 0.671 | 0.962 | 16 | 0.594 | 1.26 | 0.624 | 1.174 |
| | 2 months ago | 2.0min | - | 0.845 | 0.468 | 22 | 0.594 | 1.37 | 0.651 | 1.222 |
| | 2 months ago | 42.9s | - | 0.099 | 2.461 | 4 | 0.104 | 2.303 | 0.128 | 2.296 |
| | 2 months ago | 1.2min | - | 0.879 | 0.392 | 10 | 0.651 | 1.221 | 0.618 | 1.182 |
| | 2 months ago | 2.1min | - | 0.9 | 0.345 | 9 | 0.637 | 1.26 | 0.639 | 1.222 |
| | 2 months ago | 3.1min | - | 0.979 | 0.116 | 15 | 0.631 | 1.366 | 0.624 | 1.284 |
| | 2 months ago | 2.2min | - | 0.805 | 0.578 | 10 | 0.622 | 1.203 | 0.594 | 1.168 |
| | 2 months ago | 1.5min | - | 0.83 | 0.564 | 14 | 0.645 | 1.185 | 0.639 | 1.124 |
| | 2 months ago | 1.5min | - | 0.649 | 1.088 | 13 | 0.612 | 1.211 | 0.636 | 1.163 |
| | 2 months ago | 1.3min | - | 0.836 | 0.548 | 10 | 0.647 | 1.203 | 0.651 | 1.193 |
| | 2 months ago | 1.5min | - | 0.923 | 0.308 | 11 | 0.643 | 1.184 | 0.648 | 1.147 |
| | 2 months ago | 1.2min | - | 0.988 | 0.102 | 17 | 0.643 | 1.379 | 0.648 | 1.284 |
| | 2 months ago | 56.6s | - | 0.931 | 0.318 | 17 | 0.647 | 1.197 | 0.648 | 1.177 |
| | 2 months ago | 28.0s | - | 0.691 | 0.963 | 18 | 0.558 | 1.387 | 0.558 | 1.401 |

Figure 10: Metrics Saved with Mlflow

| | | | Parameters | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Start Time | Duration | Run Name | CNN_first_units | CNN_second_un | CNN_third_units | activation functi | baseline | batch_size | class_weight | droput_cnn |
| 2 months ago | 7.0min | - | 16 | 32 | 64 | relu | None | None | None | 0.2 |
| 2 months ago | 3.0min | - | 16 | 32 | 64 | relu | None | None | None | 0.2 |
| 2 months ago | 1.0min | - | 16 | 32 | 64 | relu | None | None | None | 0.2 |
| 2 months ago | 1.9min | - | 16 | 32 | 64 | relu | None | None | None | 0.2 |
| 2 months ago | 2.0min | - | 16 | 32 | 64 | relu | None | None | None | 0.4 |
| 2 months ago | 2.6min | - | 16 | 32 | 64 | relu | None | None | None | 0.5 |
| 2 months ago | 1.6min | - | 16 | 32 | 64 | relu | None | None | None | 0.4 |
| 2 months ago | 2.0min | - | 16 | 32 | 64 | relu | None | None | None | 0.4 |
| 2 months ago | 42.9s | - | 16 | 32 | 64 | sigmoid | None | None | None | 0.4 |
| 2 months ago | 1.2min | - | 16 | 32 | 64 | tanh | None | None | None | 0.4 |
| 2 months ago | 2.1min | - | 32 | 64 | 128 | tanh | None | None | None | 0.4 |
| 2 months ago | 3.1min | - | 32 | 64 | 128 | tanh | None | None | None | 0.4 |
| 2 months ago | 2.2min | - | 32 | 64 | 128 | tanh | None | None | None | 0.6 |
| 2 months ago | 1.5min | - | 16 | 32 | 64 | tanh | None | None | None | 0.6 |
| 2 months ago | 1.5min | - | 16 | 32 | 64 | tanh | None | None | None | 0.8 |
| 2 months ago | 1.3min | - | 16 | 32 | 64 | tanh | None | None | None | 0.4 |
| 2 months ago | 1.5min | - | 16 | 32 | 64 | tanh | None | None | None | 0.2 |
| 2 months ago | 1.2min | - | 8 | 16 | 32 | tanh | None | None | None | 0.2 |
| 2 months ago | 56.6s | - | 4 | 8 | 16 | tanh | None | None | None | 0.2 |
| 2 months ago | 28.0s | - | 4 | 8 | 16 | tanh | None | None | None | 0.2 |
| 2 months ago | 1.1min | - | 16 | 32 | 64 | tanh | None | None | None | 0.4 |

Figure 11: Some Parameters of the Experiments in Figure 9, Tracked by Mlflow

After lots of experiments the best results came from the architecture with the following hyperparameters:

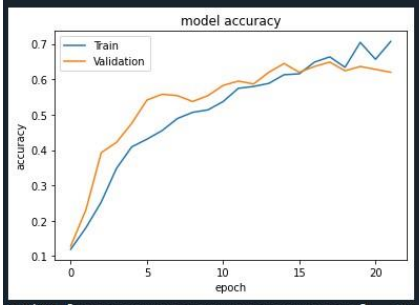
Activation function: 'tanh'
Last activation function = 'softmax'
Kernel size = (3,3)
Pooling size = (2,2)
Num of CNN Layers = 3

CNN first layer units = 16
CNN second layer units = 32
CNN third layer units = 64
CNN dense layer units = 64
Dropout layer with 0.4 dropout rate

With the help of these experiments done, the outcomes of the model changed as the following.

For the unmodified dataset:
Train accuracy: %70.44 at the 20. epoch.
Validation accuracy: %64.88 at the 18. epoch.
Test accuracy: %65.75 at the end of the training.

```
36/36 [==============================] - 34s 945ms/step - loss: 0.9672 - accuracy:
0.6486 - val_loss: 1.2486 - val_accuracy: 0.6364
Epoch 18/100
36/36 [==============================] - 33s 930ms/step - loss: 0.9057 - accuracy:
0.6635 - val_loss: 1.1646 - val_accuracy: 0.6488
Epoch 19/100
36/36 [==============================] - 34s 951ms/step - loss: 0.9399 - accuracy:
0.6338 - val_loss: 1.2766 - val_accuracy: 0.6240
Epoch 20/100
36/36 [==============================] - 34s 934ms/step - loss: 0.8358 - accuracy:
0.7044 - val_loss: 1.2300 - val_accuracy: 0.6364
Epoch 21/100
36/36 [==============================] - 34s 939ms/step - loss: 0.9423 - accuracy:
0.6565 - val_loss: 1.1998 - val_accuracy: 0.6281
Epoch 22/100
36/36 [==============================] - 34s 950ms/step - loss: 0.7797 - accuracy:
0.7071 - val_loss: 1.4029 - val_accuracy: 0.6198
13/13 [==============================] - 3s 208ms/step
```




```
13/13 [==============================] - 3s 217ms/step - loss: 1.2705 - accuracy:
0.6575
```

Figure 12: Log after Mlflow Implementation

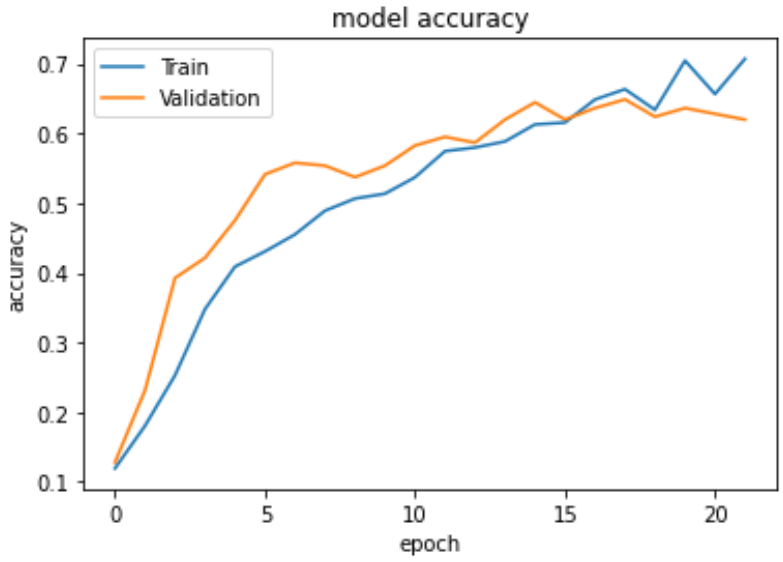


Figure 13: Accuracy of the Mlflow Implemented Model

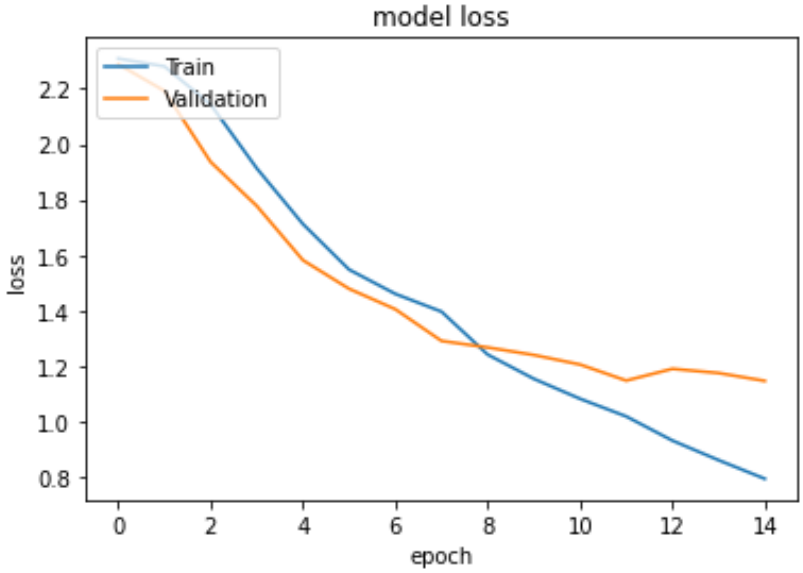



Figure 14: Loss of the Mlflow Implemented Model

Thus, the final results for this model are the following:

| Train Accuracy | Validation accuracy | Test accuracy |
|---|---|---|
| %70.44 | %64.88 | %65.75 |

Table 1: Results for CNN (Unmodified)

### 3.1.2 CNN + BiGRU Architecture

In this model or architecture, features from the images will be extracted by the CNN architecture and then they will be fed to BiGRU layers. Also, there might be additional dropout layers at some points which will be later optimized with Mlflow as it was with the bare CNN implementation before in 3.1.1.

For this architecture a detailed literature review was conducted. Understanding the purpose of RNNs and their implementation took some time and actually creating this architecture took more than days since formatting the data was challenging at some points.

To give brief information about GRU and BiGRU, GRU (Gated Recurrent Unit) aims to solve the vanishing gradient problem which comes with a standard recurrent neural network (RNN), whereas a **Bidirectional GRU**, or **BiGRU**, is a sequence processing model that is made up of 2 GRUs, one of them taking the input in a forward direction, and the other in a backwards direction [12], [13].

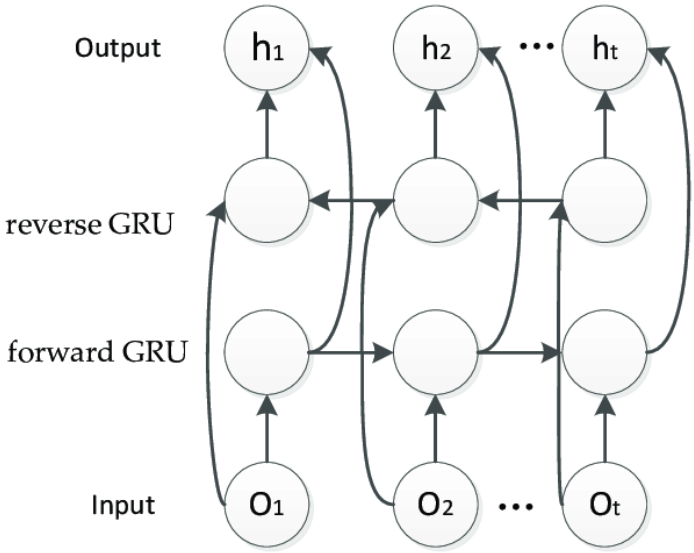



Figure 15: BiGRU Network Layer [14]

### 3.1.2.2 Pre-Processing

The pre-processing started with the same steps with the previous CNN architecture (extracting the lip regions) however collages were not created in this method. Since GRU layers operated for a fixed number of images at once, number of images for each video was restricted to be equal. For this purpose, the idea used in the creation of the collage maker algorithm was also used there. With this algorithm, number of images that a video contained were extended and fixed to 25 by replicating some of the images in the same manner as before. Rather than that no further operation took place. The figure below shows the part of the pre-processing code that is responsible for fixing the image number to 25.

```
for i in range(25):
    images[i] = image_list[int(i*len(image_list)/25)]
```

Figure 16: Python Code for Fixing the Image Number

In this code *images* and *image_list* are list type objects and it uses the idea to round the number coming from the length of the *image_list* divided by 25. With this code some of the images are randomly duplicated. Figure 17 below shows images of one video before and after pre-processing.

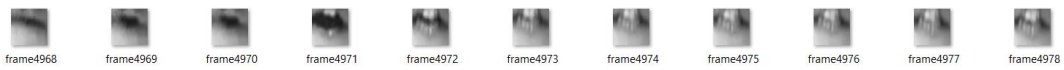



Figure 17: Before Pre-Processing

Figure 18: After Pre-Processing

### 3.1.2.3 Modeling

The feature extractor CNN layers were created using the same hyperparameters with the bare CNN architecture as it was discussed in 3.1.1.3. initially (they were optimized later) and the only difference is that the output of the last max pooling layer was fed to the BiGRU layers after flattening rather than having a dense layer. The reason for initially choosing the same hyperparameters is that as they will be optimized with Mlflow by doing lots of experiments, it is not a concern at the initialization part. Figure below shows the part of the code related to this argument.

```
model = Sequential()
model.add(tf.keras.layers.Conv2D(16,(5,5), input_shape= (50,50,1), activation='relu'))
model.add(tf.keras.layers.MaxPool2D(2,2)),
model.add(tf.keras.layers.Conv2D(32,(5,5), activation='relu')),
model.add(tf.keras.layers.MaxPool2D(2,2)),
model.add(tf.keras.layers.Conv2D(64,(5,5), activation='relu')),
model.add(tf.keras.layers.MaxPool2D(2,2)),
model.add(Flatten(name = 'FFF')),
```

Figure 19: CNN as Feature Extractor

For the BiGRU implementation to CNN, initially there were 2 sequential BiGRU layers with 64 units followed by 2 dense layers with 128 and 10 units. As it was explained earlier the reason for choosing 10 units for the last layer was due to having 10 separate classes.

```
model1 = Sequential()
model1.add(Bidirectional(GRU(64, return_sequences = True)))
model1.add(Bidirectional(GRU(64)))
model1.add(Dense(128, activation='relu'))
model1.add(Dense(10, activation='softmax'))
```

Figure 20: BiGRU Implementation After CNN

### 3.1.2.4 Results

Initially the model resulted to an accuracy of 55% however, from the previous work with the CNN architecture it is known that after optimization of hyperparameters with Mlflow, accuracy tends to increase. Early stopping was configured to 5 epochs as a regularization method as discussed earlier. Training ended after 25 epochs since overfitting was happening. From Figure 21 below it can be seen that Train accuracy is **78.94%**, validation acc. Is **52.30%** at most and test accuracy is **50%** before optimization of hyperparameters.

```
28/28 [==============================] - 1s 25ms/step - loss: 0.8228 - accuracy: 0.7123 -
val_loss: 1.5823 - val_accuracy: 0.5428
Epoch 21/50
28/28 [==============================] - 1s 26ms/step - loss: 0.8220 - accuracy: 0.7158 -
val_loss: 1.5728 - val_accuracy: 0.5132
Epoch 22/50
28/28 [==============================] - 1s 26ms/step - loss: 0.7473 - accuracy: 0.7457 -
val_loss: 1.5708 - val_accuracy: 0.5230
Epoch 23/50
28/28 [==============================] - 1s 26ms/step - loss: 0.7593 - accuracy: 0.7491 -
val_loss: 1.5377 - val_accuracy: 0.5263
Epoch 24/50
28/28 [==============================] - 1s 26ms/step - loss: 0.6702 - accuracy: 0.7802 -
val_loss: 1.6251 - val_accuracy: 0.5099
Epoch 25/50
28/28 [==============================] - 1s 26ms/step - loss: 0.6317 - accuracy: 0.7894 -
val_loss: 1.6963 - val_accuracy: 0.5000
WARNING:absl:Found untraced functions such as gru_cell_7_layer_call_fn,
gru_cell_7_layer_call_and_return_conditional_losses, gru_cell_8_layer_call_fn,
gru_cell_8_layer_call_and_return_conditional_losses, gru_cell_10_layer_call_fn while
saving (showing 5 of 8). These functions will not be directly callable after loading.
INFO:tensorflow:Assets written to: C:\Users\PCA\AppData\Local\Temp\tmpkzn1jmp_\model\data
\model\assets
INFO:tensorflow:Assets written to: C:\Users\PCA\AppData\Local\Temp\tmpkzn1jmp_\model\data
\model\assets
16/16 [==============================] - 2s 14ms/step
16/16 [==============================] - 0s 14ms/step - loss: 1.3937 - accuracy: 0.5519

In [6]:
```

Figure 21: Python Log Showing Results

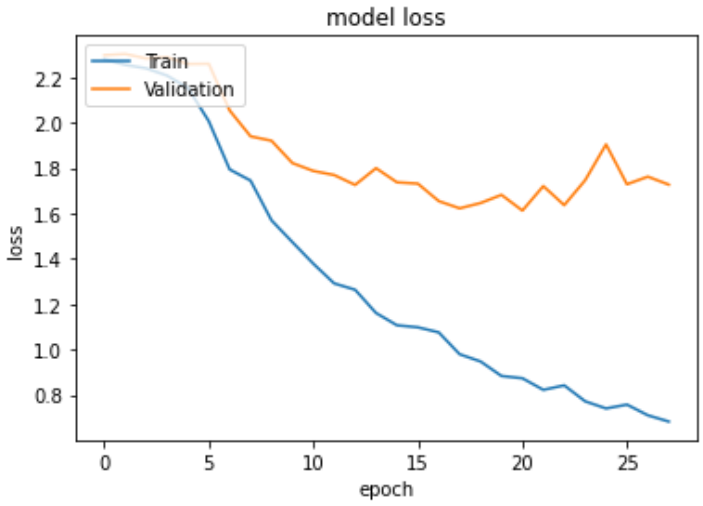



Figure 22: Loss Curves

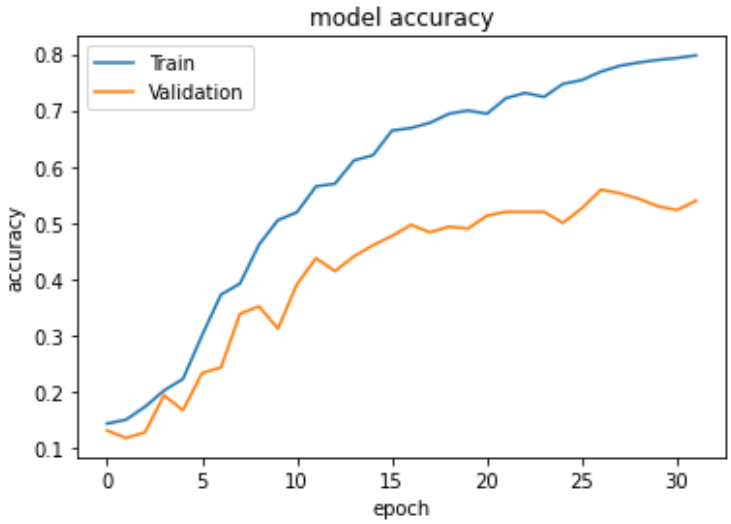



Figure 23: Accuracy Curves

### 3.1.2.5 Mlflow Implementation

**Relu** was proven to be the optimal activation function for CNN and Dense layers (not the last layer).

Best accuracy obtained with 3 sequential CNN layers that have the following number of units: **8, 16, 32** (Test Loss was relatively higher) and 2 BiGRU layers with **128** and **256** units.

Figure below is obtained from Mlflow and it demonstrates how each parameter combination result on test accuracy.

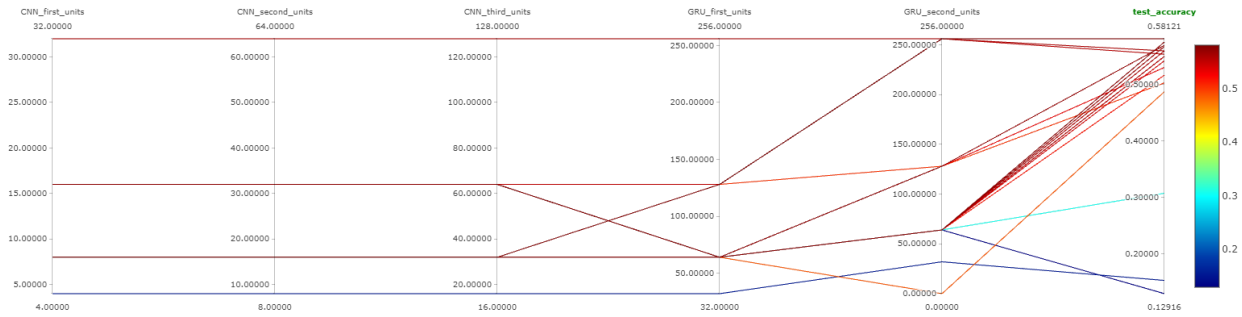



Figure 24: CNN and BiGRU Units vs Test Accuracy

As it can be seen from Figure 24 above, **the best accuracy (58.12%) obtained with no dropout layers** however the second-best experiment (57.5%) involved 2 dropout layers for CNN with 0.3 and 0.3 dropout rates and a dropout layer for BiGRU with 0.7 dropout rate. This may show that regularization is not that necessary. Actually, with dropout layers applied the best experiment also performed bad with 51% accuracy. This can be due to the fact that a few neurons might be delivering some important information about the word spoken and by adding dropout layers, these neurons might be killed that eventually results to loss of important information about the image.

Thus, at last the hyper parameters were as following:
Number of BiGRU layers = 2,
BiGRU first layer units = 128,
BiGRU second layer units = 256,
BiGRU dense layer units = 512,
One dropout layer with dropout rate 0.6.

To conclude, test accuracy was improved from 50% to **58.12%** by optimizing the hyperparameters. Train and validation accuracies are not noted since the goal is to optimize the test accuracy.

### 3.1.3 Transfer Learning

Firstly, to briefly mention about transfer learning, the reuse of a pre-trained model on a new problem is known as transfer learning in machine learning and deep learning. With this method, knowledge learned from a prior assignment can be used to predict other tasks that are closely related to the task that the model has trained [14]. Transfer learning offers some advantages, the most important of them are reduced training time and improved neural network performance for conditions that lacks of a large amount of data [15]. In our case since the dataset is small compared to other deep learning applications that performs on millions of labeled data samples, using transfer learning seems a pretty good way to increase accuracy.

Detailed Literature Review was conducted to find a suitable model to be transferred that will be used on lip reading. There were not any model that could be imported to be used on lip reading however the closest model was chosen as VGG16. VGG16 is a classification algorithm which is able to classify a thousand images of thousands of different categories with nearly 93% accuracy [16]. It is one of the most popular algorithms for classification of images [16]. It is also easier to integrate to code than other popular image classification algorithms. Since VGG16 was built to classify general images, it is thought that it could also distinguish between collages of lip images as it was in 3.1.1.2.

There is no pre-processing, modeling and Mlflow integration sections for Transfer Learning unlike the previous handmade architectures because the same pre-processing method was used to make collages which described in the 3.1.1.2. section of this report. Modeling and Mlflow integration parts were also absent since the model was imported as a whole.

#### 3.1.3.2 Results

At the end of the training (15 epochs) the train accuracy was 31% at most and the test accuracy was **24%** as it is demonstrated by the figure below.

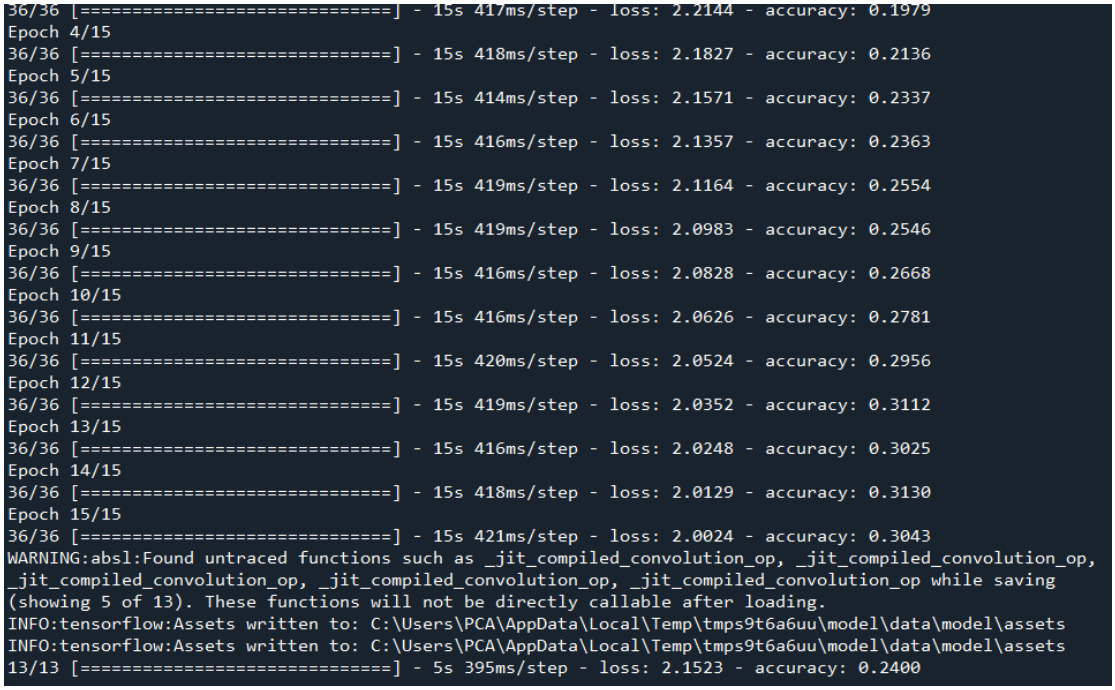



Figure 25: Accuracies Obtained

As it can be observed easily, these accuracy rates are pretty low and the possible reasons for such a bad result is that VGG16 is simply not that good for the lip-reading purpose (although it is the most suitable model to be transferred). Mlflow implementation could not be made since this was an imported model.

Thus, to give the final results obtained for each architecture:

| Model | Test Accuracies |
|---|---|
| CNN | 65.75% |
| CNN+BiGRU | 58.12% |
| Transfer Learning of VGG16 | 24.00% |

Table 2: Overall Results of the Project

Since the goal was to find the most successful deep learning model out of these 3 models, the purpose is successfully satisfied with CNN architecture having 65.75% test accuracy.

## 3.2 Avionic Predictive Maintenance Project

For this project a detailed literature review was conducted about the following questions:

- What predictive maintenance is?
- Which fields use predictive maintenance?
- What are the advantages of predictive maintenance on avionic systems?
- Which machine learning algorithms lead to accurate maintenance predictions?

And some more detailed questions related to these 4 questions. The findings from the research can be summed up as below.

To briefly talk about the outcomes of the research conducted, predictive maintenance algorithms use data analysis to estimate when an equipment might fail so that maintenance can be scheduled before the failure happens [1]. The purpose of predictive maintenance is to use an equipment's life span to its fullest. To perform

predictive maintenance there are many machine learning algorithms used at different fields. Some of them are: Support Vector Machine, Decision Tree, Random Forest, Hidden Markov Model, Linear Regression, etc. And some fields that predictive maintenance are used are: Avionic, automotive, energy, manufacturing, hard drive companies, etc.

The usage of Predictive Maintenance on Avionic Equipment decreases unscheduled maintenance events which have big-time impact on an airline's schedule performance, passenger satisfaction [17]. With predictive maintenance 10 to 50 million dollars per year can be saved by airline companies [17].
Rather than that a power point presentation was created and a presentation was delivered to other engineers working on predictive maintenance. The presentation was 20 minutes long so lots of preparation was needed. This project was a minor project after finishing the previous Lip-Reading project before the deadline. The emphasis was on the Lip-Reading project.

# 4 Performance and Outcomes

## 4.1 Solving Complex Engineering Problems

Mlflow implementation: It was hard to activate the mlflow at first because there were just a few resources to get help and the Mlflow was not compatible with the python version which was python 3.8. To overcome this issue python version was upgraded to 3.9. Rather than that since python was using in the anaconda environment, it was causing problems to open the Mlflow user interface. This problem was solved by using anaconda prompt after searching for a solution online for several hours.

Rather than that, for CNN+BiGRU architecture for Lip Reading project the pre-processing stage involved reformatting data to 25 frames for every video. However, each video in the dataset contained different number of frames. To overcome this issue a simple algorithm was written as it can be seen from Figure 15. Overcoming this issue involved a creative engineering approach.

Rather than that, CNN and CNN + BiGRU implementation of Lip Reading Project, at first there was the problem of overfitting which occurred because of training the model on a relatively small dataset for much more epochs than needed and the lip regions on the images were noisy so that the model was learning the noise as an underlying concept. This problem was searched online and some literature review was conducted. The findings indicated the usage of some regularization methods were necessary and they were chosen as early stopping and dropout layers. Usage of dropout layers solved this problem by deactivating some neurons at random so that the model did not learn the noise in the images. Thus, both of the implementations involved these regularization methods.

## 4.2 Recognizing Ethical and Professional Responsibilities

There were meetings of ours every Thursday. It was my responsibility to summarize the things I have worked on that week and present it via a power-point presentation or with a written report. Attending these meetings with the related material was one of my professional responsibilities.

Rather than that there were tasks given to me for a specific time period. There were many times that I finished my task before the deadline. At these times rather than

idling, I informed my mentor that I completed the task and he gave me another. This was an ethical responsibility of mine.

More than that, through the training it was always followed the mentor's instructions about the works and it was never worked separately. This was also a professional responsibility since working for needed parts for the team is important for the project rather than overdoing some part which was already dealt with another engineer's work.

## 4.3 Making Informed Judgments

While implementing the CNN architecture with collages as described in the work done section, at first the accuracy rate was lower than the result (61% while the project resulted to 65% accuracy.) at this point rather than accepting the result and keeping it, it was searched online and literature review was conducted. From the findings it was observed that there was a python library called Mlflow to optimize the hyperparameters and it was used accordingly. This increased accuracy to 65%. Thus, a judgment was made at the point that accuracy was lower than expected and this judgment resulted to a greater accuracy that improved the project's result.

Rather than that, at Figure 7, it was observed that the model was having a difficulty to distinguish between "başla" and "bitir" frequently although other words were successfully classified in general. Rather than trying to fix some parts of the model, a judgment was made such that these words' pronunciation was quite similar and the problem was due to this phenomenon. By making this judgment lots of time were saved.

## 4.4 Acquiring New Knowledge by Using Appropriate Learning Strategies

In the Lip-Reading project as the first step of all 3 approaches the mouth regions must be extracted from each frame. Since there were not any previous experience on computer vision or related image works, new knowledge had to be acquired. This was done through literature reviews, YouTube videos and searching for online articles. With this appropriate learning technique dlib library was discovered and it eased the pre-processing stage greatly which is discussed in the next section 4.5.

Rather than that another beneficial strategy to learn was through the supervisor engineer. He stated and taught the concepts needed to be learnt to improve the work done. At a point he suggested to learn regularization and its techniques that can be implemented on the code. Then he taught how to use early stopping and dropout layers. Rather than that he also suggested taught the usage of confusion matrices.

## 4.5 Applying New Knowledge As Needed

"face landmark detection" of dlib was used throughout the project to cut the lip regions as a stage of the pre-processing as dlib was previously learnt through literature review and YouTube videos.

In the Lip Reading Project, during the evaluation of the results of different architectures, the engineer suggested to use a confusion matrix to check which words are confused by the model. By implementing the new knowledge of confusion

matrices to the codes, it enabled to visually see the confusions and therefore make important judgements.

Rather than that, at first regularization techniques were not known and therefore the models created were facing the problem of overfitting as explained in 4.4. By applying the new knowledge obtained from the engineer, models overcame the problem of the overfitting.

# 5 Conclusions

This internship enabled to learn and practice on machine learning and mostly on deep learning with learning and implementing many useful python libraries that are frequently used for machine learning. By doing pre-processing, data handling was learnt and by creating different neural network models the power of deep learning was emphasized. A commonly used deep learning model CNN was extensively used in this internship with also learning about BiGRU architectures that can handle sequences of data which is very useful for action recognition or lip reading in our case. Regularization was another important concept that is learnt specially while working with small datasets that tend to overfit. When it comes to optimize the models for the required task MlFlow library was discovered. It turned out to be an amazing tool to track the hyperparameters of the neural network. Rather than that activation functions such as “ReLu, Sigmoid, Softmax, etc. was learnt, implemented and mathematically analyzed to understand the importance of them. Rather than that avionic predictive maintenance was searched through literature review and learnt through other engineers. Although the emphasis was on the lip reading project, avionic predictive maintenance made me realize that the scope of the machine learning and deep learning is so large and getting into other industries very fast. It was also beneficial to learn about vast of the machine learning algorithms and their working principles. The company BITES was at “METU Technopolis” so that it enabled to see other companies and engineers at some time that enhanced my view through engineering and my career plan. The Artificial Intelligence team I have worked on and my supervisor there helped me at many points when I struggled to implement some parts of the codes and they pushed me to learn further. The tasks given were successfully accomplished at the end of the training.

## References


[1] “Predictive Maintenance”. https://www.heavy.ai/technical-glossary/predictive-maintenance. [Accessed: Sep 25, 2022].

[2] “BITES”. https://www.bites.com.tr/. [Accessed: Sep 25, 2022].

[3] “BITES - Defence & Aerospace Technologies”. https://tr.linkedin.com/company/bitessavunma [Accessed: Sep 25, 2022].

[4] “Lip Reading using Neural Network and Deep Learning”. https://portfolios.cs.earlham.edu/wp-content/uploads/2019/08/CS488_Karan_Final_Paper.pdf [Accessed: Sep 25, 2022].

[5] “Automated Lip Reading: Simplified”. https://towardsdatascience.com/automated-lip-reading-simplified-c01789469dd8. [Accessed: Sep 25, 2022].

[6] “Basic CNN Architecture: Explaining 5 Layers of Convolutional Neural Network”. https://www.upgrad.com/blog/basic-cnn-architecture/#:~:text=other%20advanced%20tasks.-,What%20is%20the%20architecture%20of%20CNN%3F,the%20main%20responsibility%20for%20computation. [Accessed: Sep 25, 2022].

[7] “Regularization in Deep Learning”. https://towardsdatascience.com/regularization-in-deep-learning-l1-l2-and-dropout-377e75acc036#:~:text=Regularization%20is%20a%20set%20of,data%20from%20the%20problem%20domain. [Accessed: Sep 25, 2022].

[8] “What is TensorFlow?”. https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html. [Accessed: Sep 25, 2022].

[9] “Relu vs Sigmoid vs Softmax as Hidden Layer Neurons”. https://stats.stackexchange.com/questions/218752/relu-vs-sigmoid-vs-softmax-as-hidden-layer-neurons. [Accessed: Sep 25, 2022].

[10] “Activation Functions: Sigmoid, Tanh, ReLU, Leaky ReLU, Softmax”. https://medium.com/@cmukesh8688/activation-functions-sigmoid-tanh-relu-leaky-relu-softmax-50d3778dcea5#:~:text=As%20per%20our%20business%20requirement,use%20in%20last%20output%20layer%20. [Accessed: Sep 25, 2022].

[11] “Mlflow Guide”. https://docs.databricks.com/applications/mlflow/index.html#:~:text=MLflow%20is%20an%20open%20source,and%20compare%20parameters%20and%20results. [Accessed: Sep 25, 2022].

[12] “Bidirectional GRU”. https://paperswithcode.com/method/bigru. [Accessed: Sep 25, 2022].

[13] “Understanding GRU Networks”. https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be. [Accessed: Sep 25, 2022].

[14] “BiGRU Network Layer”. https://www.researchgate.net/figure/BiGRU-Network-Layer_fig1_359062984. [Accessed: Sep 25, 2022].

[15] “Understanding Transfer Learning for Deep Learning”. https://www.analyticsvidhya.com/blog/2021/10/understanding-transfer-learning-for-deep-learning/#:~:text=The%20reuse%20of%20a%20previously,a%20small%20amount%20of%20data. [Accessed: Sep 25, 2022].

[16] “Everything You Need to Know About VGG16”. https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918. [Accessed: Sep 25, 2022].

[17] “Moving Beyond the Hype of Predictive Maintenance”. https://aerospace.honeywell.com/us/en/about-us/blogs/moving-beyond-the-hype-of-predictive-maintenance. [Accessed: Sep 25, 2022].

# Appendices

## **Appendix A**: Frames Cut from a YouTube Video

Below figures represents the frames cut from one YouTube video for the word “Afiyet Olsun”.

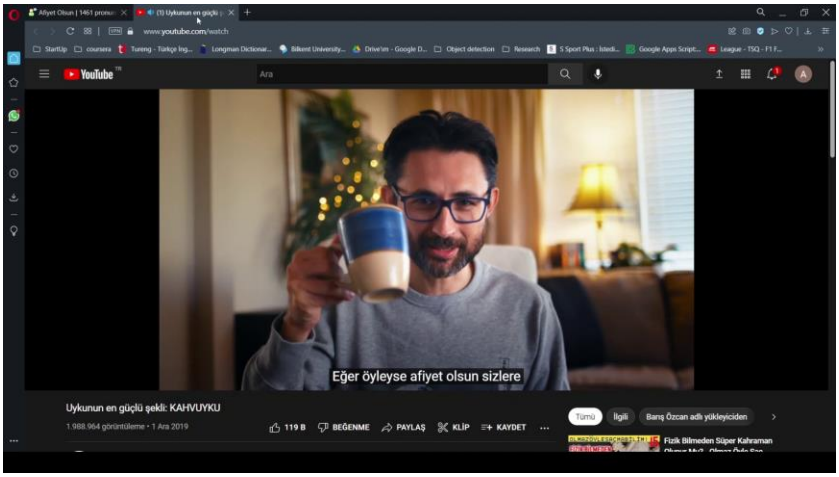



Figure 26: Frame 1

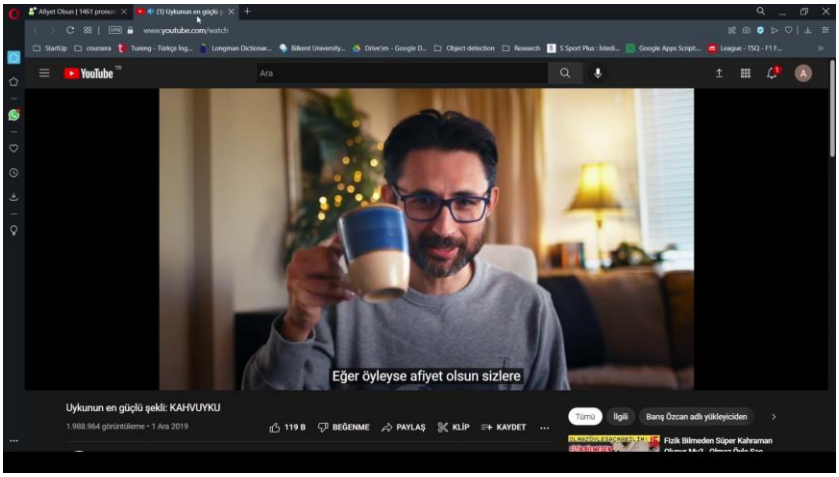



Figure 27: Frame 2

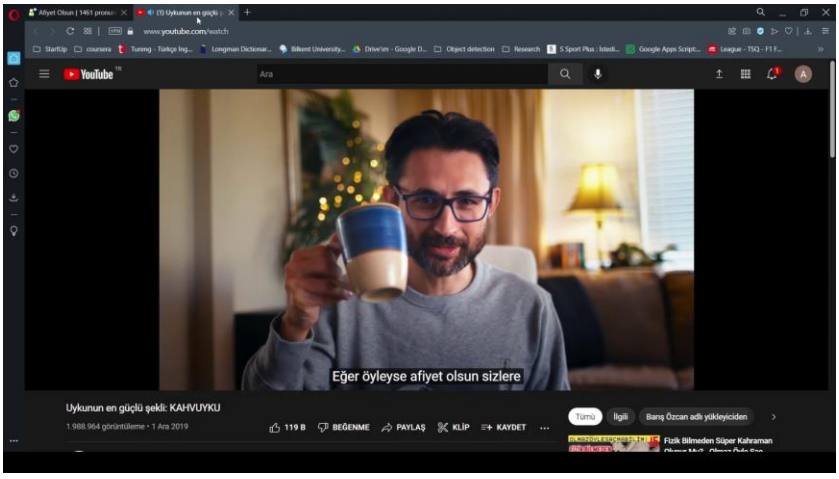



Figure 28: Frame 3

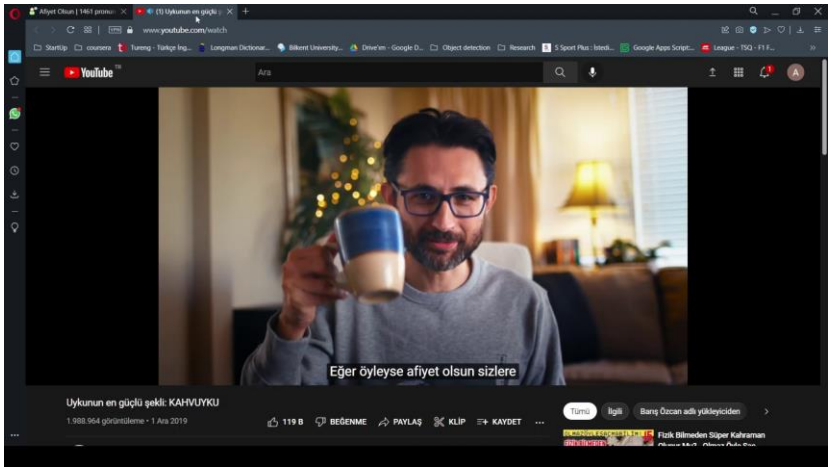



Figure 29: Frame 4

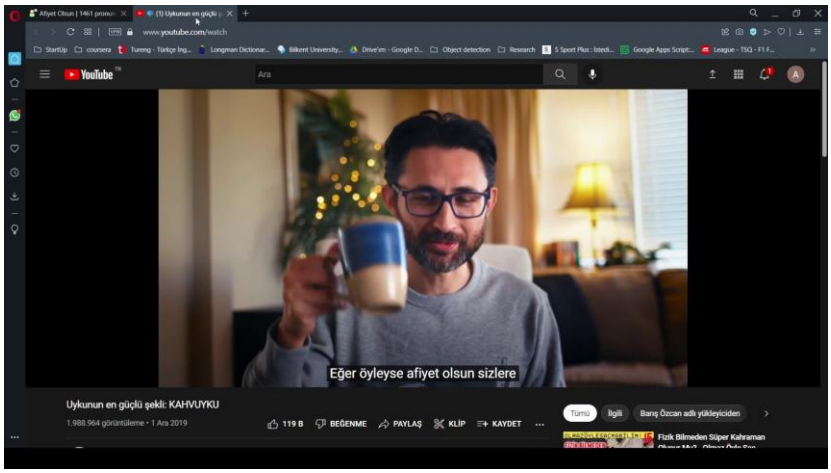



Figure 30: Frame 5

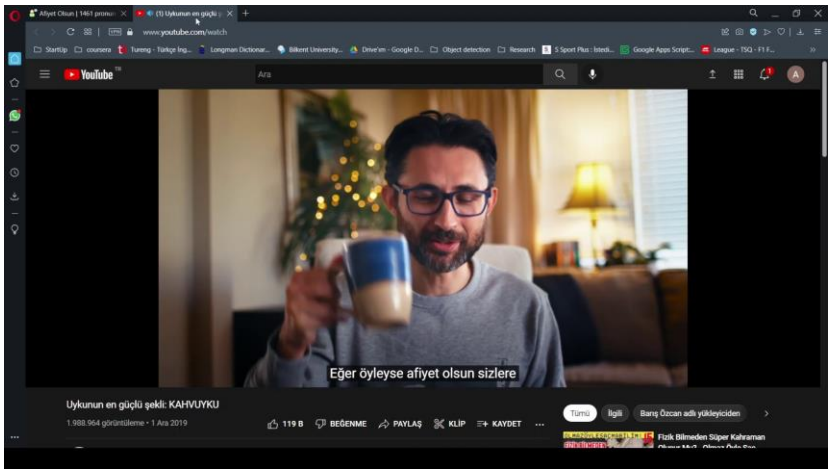



Figure 31: Frame 6

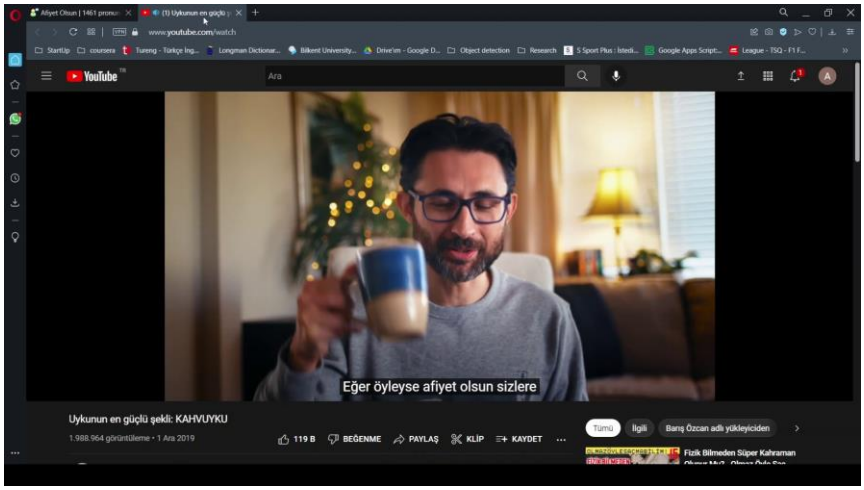



Figure 32: Frame 7

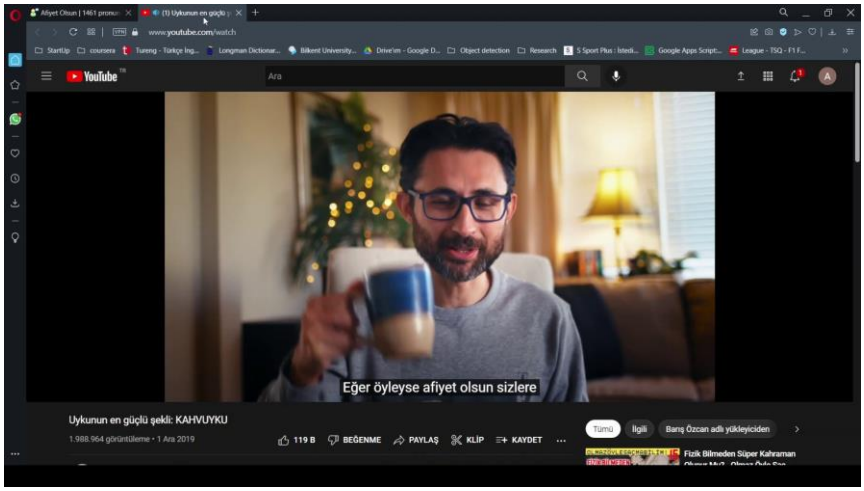



Figure 33: Frame 8

## **Appendix B:** Organization of the Dataset


Figure 34: Organization of the Dataset

**Appendix C:** Lip Region Extractor Code (A Part of Pre-Processing)

```
import numpy as np

import cv2

from imutils import face_utils

import dlib

import os


words = ['hosgeldiniz', 'merhaba', 'ozurdilerim', 'tesekkurederim', 'selam']

labels = ['Hoş geldiniz', 'Özür dilerim', 'Teşekkür ederim', 'Merhaba', 'Selam',
'Günaydın']

n_labels = len(words)

root_path = 'dataset/basla50/'

shape_pred_path = 'dataset/shape_predictor_68_face_landmarks.dat'


root_path_mouth = 'dataset/basla50mouth/'


os.mkdir(root_path_mouth)

for word in words:

    videos = os.listdir(root_path + word)

    os.mkdir(root_path_mouth + word)

    for video in videos:

        print(video) #001, 002, 003

        os.mkdir(root_path_mouth + word + '/' + video)

        sequence = []

        image_list = os.listdir(root_path + word + '/' + video)

        for im in image_list:

            image = cv2.imread(root_path + word + '/' + video + '/' + '/' + im, 0)
```

```
        face=dlib.get_frontal_face_detector()(image, 1)
        for each in face:
            face_points=dlib.shape_predictor(shape_pred_path)(image,each)
            face_points = face_utils.shape_to_np(face_points)
            (x, y, w, h) = cv2.boundingRect(np.array([face_points[49:68]])) # 48-68
mouth points
            (a,b) = face_points[49]
            mouth = image[y:y+h, x:x+w]
            try:
                mouth = cv2.resize(mouth, (50, 50))
                print(video)
                cv2.imwrite(root_path_mouth + word + '/' + video + '/' + im, mouth)
            except Exception as e:
                print(e)
```

## Appendix D: Collage Maker Code for Preprocessing of 3.1.1

```
import numpy as np

import cv2

import os


words = ['afiyetolsun', 'basla', 'bitir', 'gorusmekuzere',
'gunaydin','hosgeldiniz','ozurdilerim','tesekkurederim','selam','merhaba']

labels = ['Hoş geldiniz', 'Özür dilerim', 'Teşekkür ederim', 'Merhaba', 'Selam',
'Günaydın']

root_path = 'dataset/mouth/'

shape_pred_path = 'dataset/shape_predictor_68_face_landmarks.dat'


word_length = 15

input_dim = 50

channel = 1

stretch_seq =[]

images = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

l = 0

images_last=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]


for word in words:

    videos = os.listdir(root_path + word)

    for video in videos:

        image_list = os.listdir(root_path + word + '/' + video)

        for i in range(25):

            images[i] = image_list[int(i*len(image_list)/25)]

            if (i == 24):
```

```
            for im in images:

                images_last[l] = cv2.imread(root_path + word + '/' + video + '/'  + im, 0)

                l = l+1

            images_last = np.array(images_last)


Horizontal1=np.hstack([images_last[0],images_last[1],images_last[2],images_last[3],i
mages_last[4]])


Horizontal2=np.hstack([images_last[5],images_last[6],images_last[7],images_last[8],i
mages_last[9]])


Horizontal3=np.hstack([images_last[10],images_last[11],images_last[12],images_last
[13],images_last[14]])


Horizontal4=np.hstack([images_last[15],images_last[16],images_last[17],images_last
[18],images_last[19]])


Horizontal5=np.hstack([images_last[20],images_last[21],images_last[22],images_last
[23],images_last[24]])


Vertical_attachment=np.vstack([Horizontal1,Horizontal2,Horizontal3,Horizontal4,Hori
zontal5])

            cv2.imwrite('dataset/collage_notmodified/'+ word+ '/' + im
,Vertical_attachment)

            l = 0
```

**Appendix E:** Code for CNN Model Created for 3.1.1.2

```
import tensorflow as tf

from tensorflow import keras

from tensorflow.keras.utils import to_categorical

from tensorflow.keras.callbacks import EarlyStopping

import seaborn as sns

import numpy as np

import cv2

from sklearn.metrics import confusion_matrix

import os


words = ['afiyetolsun', 'basla', 'bitir', 'gorusmekuzere',
'gunaydin','hosgeldiniz','ozurdilerim','tesekkurederim','selam','merhaba']

labels = ['afiyetolsun', 'basla', 'bitir', 'gorusmekuzere',
'gunaydin','hosgeldiniz','merhaba','ozurdilerim','selam','tesekkurederim']

n_labels = len(words)

root_path = 'dataset/frames/'

shape_pred_path = 'dataset/shape_predictor_68_face_landmarks.dat'

root_path_mouth = 'dataset/mouth2/'


print(os.listdir("dataset"))


SIZE = 256  #Resize images


train_images = []

train_labels = []

root_path_train = 'dataset/collage_notmodified/train/'
```

```
for word in words:
    image_list = os.listdir(root_path_train + word)
    for im in image_list:
        img = cv2.imread(root_path_train + word + '/' + im,0)
        img = cv2.resize(img, (SIZE, SIZE))
        train_images.append(img)
        train_labels.append(word)

#Convert lists to arrays
train_images = np.array(train_images)
train_labels = np.array(train_labels)

test_images = []
test_labels = []
root_path_test = 'dataset/collage_notmodified/test/'
for word in words:
    image_list = os.listdir(root_path_test + word)
    for im in image_list:
        img = cv2.imread(root_path_test + word + '/' + im,0)
        img = cv2.resize(img, (SIZE, SIZE))
        test_images.append(img)
        test_labels.append(word)
#Convert lists to arrays
test_images = np.array(test_images)
test_labels = np.array(test_labels)
```

```
valid_images = []
valid_labels = []
root_path_valid = 'dataset/collage_notmodified/valid/'
for word in words:
    image_list = os.listdir(root_path_valid + word)
    for im in image_list:
        img = cv2.imread(root_path_valid + word + '/' + im,0)
        img = cv2.resize(img, (SIZE, SIZE))
        valid_images.append(img)
        valid_labels.append(word)
#Convert lists to arrays
valid_images = np.array(valid_images)
valid_labels = np.array(valid_labels)

#Encode labels to integers
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(test_labels)
test_labels_encoded = le.transform(test_labels)
le.fit(train_labels)
train_labels_encoded = le.transform(train_labels)
le.fit(valid_labels)
valid_labels_encoded = le.transform(valid_labels)

#Split data
```

```
x_train, y_train, x_test, y_test,x_valid,y_valid = train_images, train_labels_encoded, test_images, test_labels_encoded,valid_images,valid_labels_encoded


# Normalize pixel values

x_train, x_test,x_valid = x_train / 255.0, x_test / 255.0, x_valid / 255.0


#One hot encode values

y_train_one_hot = to_categorical(y_train)

y_test_one_hot = to_categorical(y_test)

y_valid_one_hot = to_categorical(y_valid)


model = tf.keras.models.Sequential([tf.keras.layers.Conv2D(16,(5,5), activation = 'relu', input_shape = (256,256,1)),

                                    tf.keras.layers.MaxPool2D(2,2),

                                    tf.keras.layers.Conv2D(32,(5,5), activation = 'relu'),

                                    tf.keras.layers.MaxPool2D(2,2),

                                    tf.keras.layers.Conv2D(64,(5,5), activation = 'relu'),

                                    tf.keras.layers.MaxPool2D(2,2),

                                    tf.keras.layers.Flatten(),

                                    tf.keras.layers.Dense(128, activation = 'relu'),

                                    keras.layers.Dropout(0.2),

                                    tf.keras.layers.Dense(10, activation = 'softmax')])


model.compile(loss = "sparse_categorical_crossentropy",

              optimizer = 'adam',

              metrics = ['accuracy'])
```

```
callbacks = [EarlyStopping(monitor='val_accuracy', patience=4)]

history = model.fit(x_train, y_train, epochs=100, validation_data = (x_valid,y_valid),
callbacks=callbacks)

y_pred = model.predict(x_test)

y_classes = [np.argmax(element) for element in y_pred]

cm = confusion_matrix(test_labels_encoded, y_classes)

cm_plot_labels = ['afiyetolsun', 'basla', 'bitir', 'gorusmekuzere',
'gunaydin','hosgeldiniz','merhaba','ozurdilerim','selam','tesekkurederim']

sns.heatmap(cm,xticklabels =['afiyetolsun', 'basla', 'bitir', 'gorusmekuzere',
'gunaydin','hosgeldiniz','merhaba','ozurdilerim','selam','tesekkurederim'],
yticklabels=['afiyetolsun', 'basla', 'bitir', 'gorusmekuzere',
'gunaydin','hosgeldiniz','merhaba','ozurdilerim','selam','tesekkurederim'], annot=True)

model.evaluate(x_test,y_test)
```

## Appendix F: Code for BiGRU Pre-processing

```
import cv2

import os


words = ['afiyetolsun', 'basla', 'bitir', 'gorusmekuzere',
'gunaydin','hosgeldiniz','ozurdilerim','tesekkurederim','selam','merhaba']

labels = ['Hoş geldiniz', 'Özür dilerim', 'Teşekkür ederim', 'Merhaba', 'Selam',
'Günaydın']

root_path = 'dataset/mouth/'

root_path_mouth = 'dataset/mouth_CNN+BiGRU/'


images = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

l = 0

images_last=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]


for word in words:
    videos = os.listdir(root_path + word)
    os.mkdir(root_path_mouth + word)
    for video in videos:
        image_list = os.listdir(root_path + word + '/' + video)
        os.mkdir(root_path_mouth + word + '/' + video)
        for i in range(25):
            images[i] = image_list[int(i*len(image_list)/25)]
            if (i == 24):
                n = 0
                for im in images:
                    n = str(n)
```

```
                visual = cv2.imread(root_path + word + '/' + video  + '/' + im, 0)

                cv2.imwrite('dataset/mouth_CNN+BiGRU/'+ word + '/' + video + '/' + n +
'.jpg' ,visual)

                n = int(n)

                n = n + 1
```

**Appendix G:** Code of the BiGru Model with MlFlow Integrated

```
import tensorflow as tf

from tensorflow.keras.layers import *

from tensorflow import keras

from tensorflow.keras import models

from tensorflow.keras.utils import to_categorical

from tensorflow.keras.callbacks import EarlyStopping

from tensorflow.keras.models import Sequential

import matplotlib.pyplot as plt

import numpy as np

import cv2

from sklearn.metrics import confusion_matrix

from keras.layers import TimeDistributed

import os

import mlflow.tensorflow

np.random.seed(3)


mlflow.start_run(experiment_id=0)

mlflow.tensorflow.autolog()

words = ['afiyetolsun', 'basla', 'bitir', 'gorusmekuzere',
'gunaydin','hosgeldiniz','ozurdilerim','tesekkurederim','selam','merhaba']

labels = ['afiyetolsun', 'basla', 'bitir', 'gorusmekuzere',
'gunaydin','hosgeldiniz','merhaba','ozurdilerim','selam','tesekkurederim']

n_labels = len(words)

root_path = 'dataset/frames/'

shape_pred_path = 'dataset/shape_predictor_68_face_landmarks.dat'
```

```
root_path_mouth = 'dataset/mouth2/'

SIZE = 50  #Resize images

#Capture training data and labels into respective lists
train_images = []
train_labels = []
root_path_train = 'dataset/mouth_lstm/train/'

for word in words:
    video_list = os.listdir(root_path_train + word)
    for video in video_list:
        im_list = os.listdir(root_path_train + word + '/' + video)
        for im in im_list:
            img = cv2.imread(root_path_train + word + '/' + video + '/' + im,0)
            img = cv2.resize(img, (SIZE, SIZE))
            train_images.append(img)
            train_labels.append(word)
#Convert lists to arrays
train_images = np.array(train_images)
train_labels = np.array(train_labels)


# Capture test/validation data and labels into respective lists

test_images = []
```

```
test_labels = []
root_path_test = 'dataset/mouth_lstm/test/'
for word in words:
    video_list = os.listdir(root_path_test + word)
    for video in video_list:
        im_list = os.listdir(root_path_test + word + '/' + video)
        for im in im_list:
            img = cv2.imread(root_path_test + word + '/' + video + '/' + im,0)
            img = cv2.resize(img, (SIZE, SIZE))
            test_images.append(img)
            test_labels.append(word)
#Convert lists to arrays
test_images = np.array(test_images)
test_labels = np.array(test_labels)

valid_images = []
valid_labels = []
root_path_valid = 'dataset/mouth_lstm/valid/'
for word in words:
    video_list = os.listdir(root_path_valid + word)
    for video in video_list:
        im_list = os.listdir(root_path_valid + word + '/' + video)
        for im in im_list:
            img = cv2.imread(root_path_valid + word + '/' + video + '/' + im,0)
            img = cv2.resize(img, (SIZE, SIZE))
            valid_images.append(img)
```

```
            valid_labels.append(word)
#Convert to arrays
valid_images = np.array(valid_images)
valid_labels = np.array(valid_labels)


#Encode labels to integer
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(test_labels)
test_labels_encoded = le.transform(test_labels)
le.fit(train_labels)
train_labels_encoded = le.transform(train_labels)
le.fit(valid_labels)
valid_labels_encoded = le.transform(valid_labels)


#Split data
x_train, y_train, x_test, y_test,x_valid,y_valid = train_images, train_labels_encoded,
test_images, test_labels_encoded,valid_images,valid_labels_encoded


# Normalize
x_train, x_test,x_valid = x_train / 255.0, x_test / 255.0, x_valid / 255.0


y_train_one_hot = to_categorical(y_train)
y_test_one_hot = to_categorical(y_test)
y_valid_one_hot = to_categorical(y_valid)


x_train = x_train.reshape(len(x_train),50,50,1)
```

```
x_test = x_test.reshape(len(x_test),50,50,1)

x_valid = x_valid.reshape(len(x_valid),50,50,1)


activation = 'relu'

last_activation = 'softmax'

kernel_size = (5,5)

pooling_size = (2,2)

num_of_CNN = 3

CNN_first_units = 8

CNN_second_units = 16

CNN_third_units = 32

CNN_dense_units = 64 #don't used

dropout1_cnn = 0.25

dropout2_cnn = 0.25



model = Sequential()

model.add(tf.keras.layers.Conv2D(CNN_first_units,kernel_size, input_shape=
(50,50,1), activation= activation))

model.add(tf.keras.layers.MaxPool2D(pooling_size)),

model.add(tf.keras.layers.Dropout(dropout1_cnn)),

model.add(tf.keras.layers.Conv2D(CNN_second_units,kernel_size, activation =
activation)),

model.add(tf.keras.layers.MaxPool2D(pooling_size)),

model.add(tf.keras.layers.Dropout(dropout2_cnn)),

model.add(tf.keras.layers.Conv2D(CNN_third_units,kernel_size, activation =
activation)),

model.add(tf.keras.layers.MaxPool2D(pooling_size)),
```

```
model.add(Flatten(name = 'FFF')),

model.add(tf.keras.layers.Dense(CNN_dense_units, activation = activation)),

model.add(tf.keras.layers.Dense(10, activation = last_activation))


model.compile(loss = "sparse_categorical_crossentropy",

              optimizer = 'adam',

              metrics = ['accuracy'])


modelFeatured = models.Model(inputs = model.input,

                             outputs = model.get_layer('FFF').output)


train_featured = modelFeatured.predict(x_train)

test_featured = modelFeatured.predict(x_test)

valid_featured = modelFeatured.predict(x_valid)


temp = train_featured[:25]

temp = temp.reshape(1,temp.shape[0],temp.shape[1])

for i in range(25,21725,25):

    temp = np.append(temp,
train_featured[i:i+25].reshape(1,25,train_featured.shape[1]), axis = 0)

trainX = temp


temp = test_featured[:25]

temp = temp.reshape(1,temp.shape[0],temp.shape[1])

for i in range(25,12775,25):

    temp = np.append(temp,
test_featured[i:i+25].reshape(1,25,test_featured.shape[1]), axis = 0)
```

```
testX = temp


temp = valid_featured[:25]

temp = temp.reshape(1,temp.shape[0],temp.shape[1])

for i in range(25,7600,25):

    temp = np.append(temp,
valid_featured[i:i+25].reshape(1,25,valid_featured.shape[1]), axis = 0)

validX = temp


temp_train = []

for i in range (869):

    n = i * 25

    temp_train.append(y_train[n])


temp_train = np.array(temp_train)


temp_test = []

for i in range (511):

    n = i * 25

    temp_test.append(y_test[n])


temp_test = np.array(temp_test)


temp_valid = []

for i in range (304):

    n = i * 25

    temp_valid.append(y_valid[n])
```

```
temp_valid = np.array(temp_valid)


n_steps = 25

n_features = 256


GRU_first_units = 128

GRU_second_units = 256

GRU_dense_units = 512

dropout_GRU = 0.6

num_of_BiGRU = 2


model1 = Sequential()

model1.add(Bidirectional(GRU(GRU_first_units, return_sequences = True)))

model1.add(Bidirectional(GRU(GRU_second_units)))

model1.add(tf.keras.layers.Dropout(dropout_GRU)),

model1.add(Dense(GRU_dense_units, activation=activation))

model1.add(Dense(10, activation=last_activation))


model1.compile(loss = ''sparse_categorical_crossentropy'',

            optimizer = 'adam',

            metrics = ['accuracy'])

callbacks = [EarlyStopping(monitor='val_accuracy', patience=5)] # Early Stopping

history = model1.fit(trainX,temp_train, epochs=50, validation_data =
(validX,temp_valid),callbacks=callbacks)
```

```
y_pred = model1.predict(testX)

y_classes = [np.argmax(element) for element in y_pred]

# cm = confusion_matrix(temp_test, y_classes)

# cm_plot_labels = ['afiyetolsun', 'basla', 'bitir', 'gorusmekuzere',
'gunaydin','hosgeldiniz','merhaba','ozurdilerim','selam','tesekkurederim']

# sns.heatmap(cm,xticklabels =['afiyetolsun', 'basla', 'bitir', 'gorusmekuzere',
'gunaydin','hosgeldiniz','merhaba','ozurdilerim','selam','tesekkurederim'],
yticklabels=['afiyetolsun', 'basla', 'bitir', 'gorusmekuzere',
'gunaydin','hosgeldiniz','merhaba','ozurdilerim','selam','tesekkurederim'], annot=True)



# plt.plot(history.history['accuracy'])

# plt.plot(history.history['val_accuracy'])

# plt.title('model accuracy')

# plt.ylabel('accuracy')

# plt.xlabel('epoch')

# plt.legend(['Train', 'Validation'], loc='upper left')

# plt.show()


plt.plot(history.history['loss'])

plt.plot(history.history['val_loss'])

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()


evaluation = model1.evaluate(testX,temp_test)

mlflow.log_metric('test_accuracy', evaluation[1])
```

```
mlflow.log_metric('test_loss', evaluation[0])
mlflow.log_param('activation function', activation)
mlflow.log_param('kernel_size', kernel_size)
mlflow.log_param('pooling_size', pooling_size)
mlflow.log_param('CNN_first_units', CNN_first_units)
mlflow.log_param('CNN_second_units', CNN_second_units)
mlflow.log_param('CNN_third_units', CNN_third_units)
mlflow.log_param('number_of_CNN_layers', num_of_CNN)
mlflow.log_param('GRU_first_units', GRU_first_units)
mlflow.log_param('GRU_second_units', GRU_second_units)
mlflow.log_param('GRU_dense_units', GRU_dense_units)
mlflow.log_param('number_of_BiGRU_layers', num_of_BiGRU)
mlflow.log_param('droput1_cnn', dropout1_cnn)
mlflow.log_param('droput2_cnn', dropout2_cnn)
mlflow.log_param('droput_GRU', dropout_GRU)
mlflow.end_run()
```

**Appendix H:** Code of Transfer Learning of VGG16

```
import numpy as np

import cv2

from keras.layers.normalization import *

from keras.models import Model

from keras.layers import Dense, Flatten

from tensorflow.keras.utils import to_categorical

import os

from keras.applications.vgg16 import VGG16



words = ['afiyetolsun', 'basla', 'bitir', 'gorusmekuzere',
'gunaydin','hosgeldiniz','ozurdilerim','tesekkurederim','selam','merhaba']

labels = ['Hoş geldiniz', 'Özür dilerim', 'Teşekkür ederim', 'Merhaba', 'Selam',
'Günaydın']

n_labels = len(words)

root_path = 'dataset/frames/'

shape_pred_path = 'dataset/shape_predictor_68_face_landmarks.dat'



root_path_mouth = 'dataset/mouth2/'



print(os.listdir("dataset"))



SIZE = 50  #Resize images



train_images = []

train_labels = []

root_path_train = 'dataset/collage/train/'
```

```
for word in words:
    image_list = os.listdir(root_path_train + word)
    for im in image_list:
        img = cv2.imread(root_path_train + word + '/' + im)
        img = cv2.resize(img, (SIZE, SIZE))
        train_images.append(img)
        train_labels.append(word)

train_images = np.array(train_images)
train_labels = np.array(train_labels)

test_images = []
test_labels = []
root_path_test = 'dataset/collage/test/'
for word in words:
    image_list = os.listdir(root_path_test + word)
    for im in image_list:
        img = cv2.imread(root_path_test + word + '/' + im)
        img = cv2.resize(img, (SIZE, SIZE))
        test_images.append(img)
        test_labels.append(word)

test_images = np.array(test_images)
test_labels = np.array(test_labels)
```

```
from sklearn import preprocessing

le = preprocessing.LabelEncoder()

le.fit(test_labels)

test_labels_encoded = le.transform(test_labels)

le.fit(train_labels)

train_labels_encoded = le.transform(train_labels)


x_train, y_train, x_test, y_test = train_images, train_labels_encoded, test_images,
test_labels_encoded


x_train, x_test = x_train / 255.0, x_test / 255.0


y_train_one_hot = to_categorical(y_train)

y_test_one_hot = to_categorical(y_test)


#Load VGG16

VGG_model = VGG16(weights='imagenet', include_top=False, input_shape=(SIZE,
SIZE, 3))


for layer in VGG_model.layers:

        layer.trainable = False


VGG_model.summary()

x = Flatten()(VGG_model.output)


prediction = Dense(10, activation='softmax')(x)
```

```
model = Model(inputs=VGG_model.input, outputs=prediction)


model.summary()


model.compile(
  loss='sparse_categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy']
)


model.fit(x_train, y_train, epochs=15)
model.evaluate(x_test,y_test)
```