Egypt IOT and AI Challenge 2020

Graduation Projects Track

Detailed Project Progress Report GP (20-57)

Arabic Sign Language (ARSL) Interpreter

Submitted By

Mahmoud Aboud Nada MennatuAllah Ahmed El-Zahaby Hesham Abdullah Al-Kholey	Asmaa Hasan Dardarah Muhammad Ali Shokair Omar Ibraheem Shaheen

Supervisor

Professor/ Wessam Fekrey

DEPARTMENT OF ELECTRICAL ENGINEERING

KAFRELSHEIKH UNIVERSITY

2020

Methodology, Implementation and Results:

To build a system that can translate full Arabic sign language sentences and turn it from signs to spoken words and turn this idea to a product that can be used by everyone in the deaf and mute community, we had to go through a lot of different phases and follow many steps.

Step #1

First, we had to gather our own customed dataset that consists of a set of words that we believe are most common in daily life according to sign language instructors.

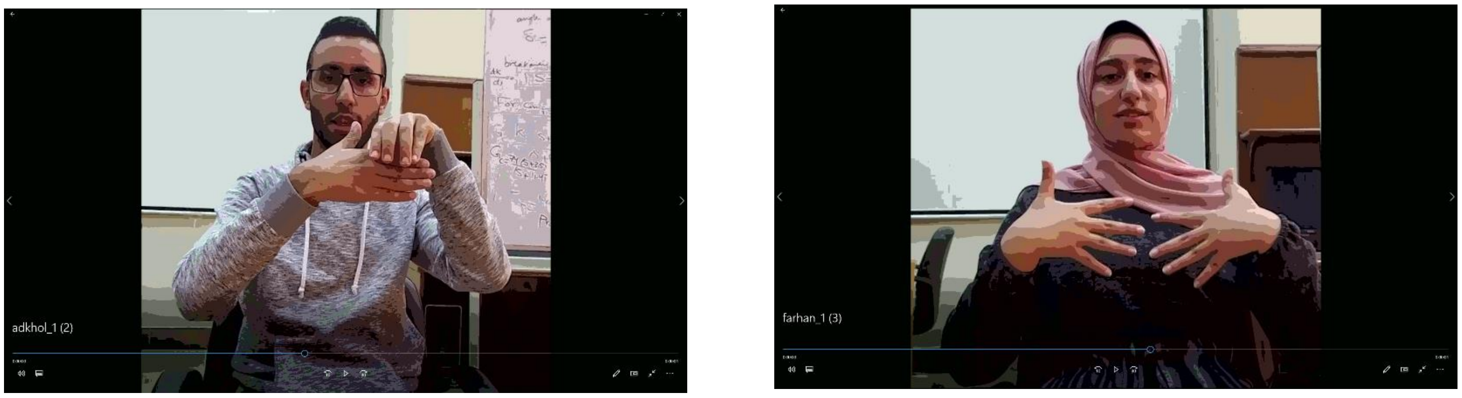


Figure 1: Snapshots from the ARSL video dataset

we started with 28 words from 4 different signers each word is repeated 10 times so this makes our initial dataset with 1120 videos.

As we were collecting the dataset from local sign language specialists, we started to prepare the code that deals with any video dataset, we used an experimental dataset called UCF101 that is used in action recognition it has 101 classes within each class videos belongs to this class.

Model #1

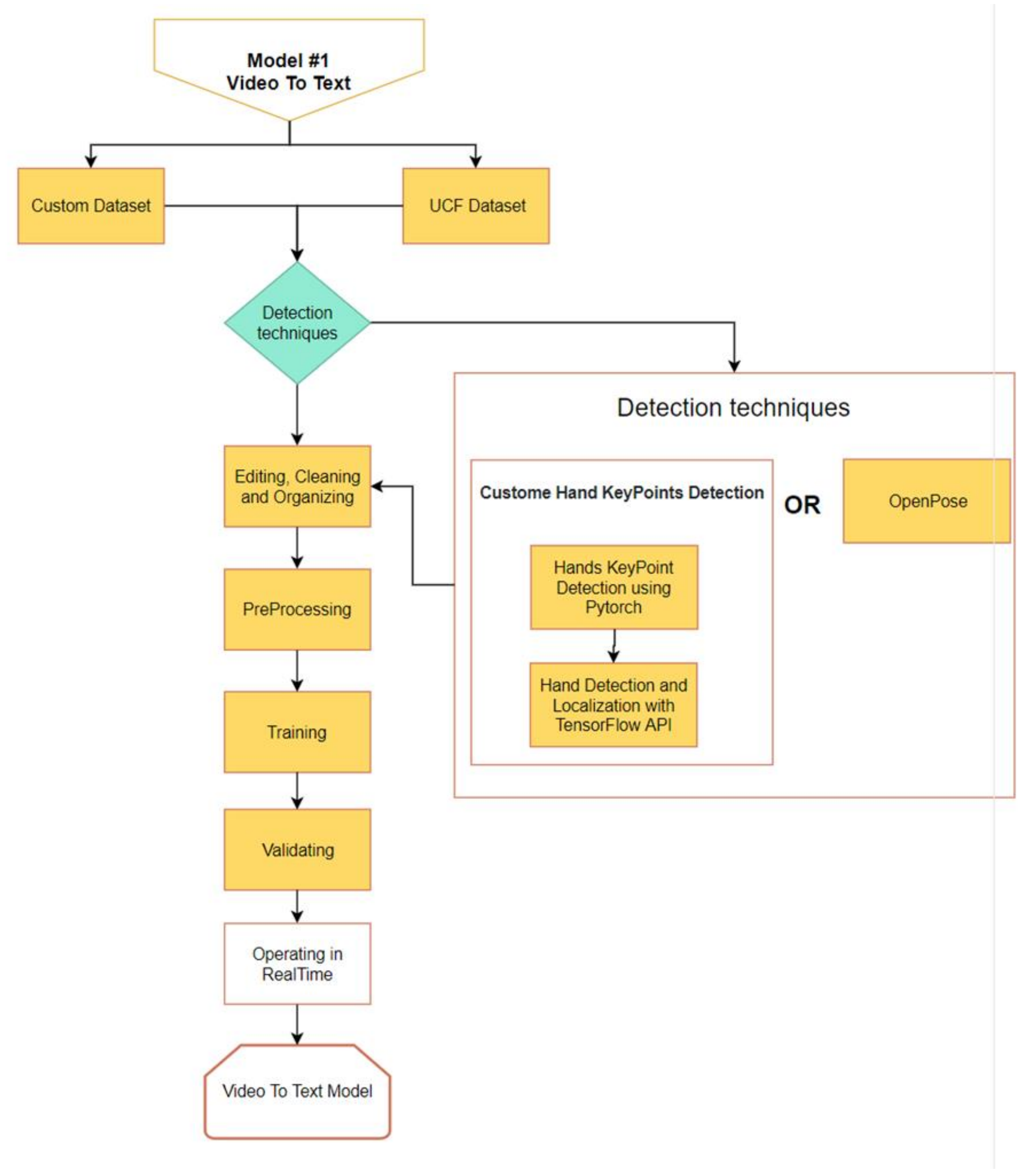
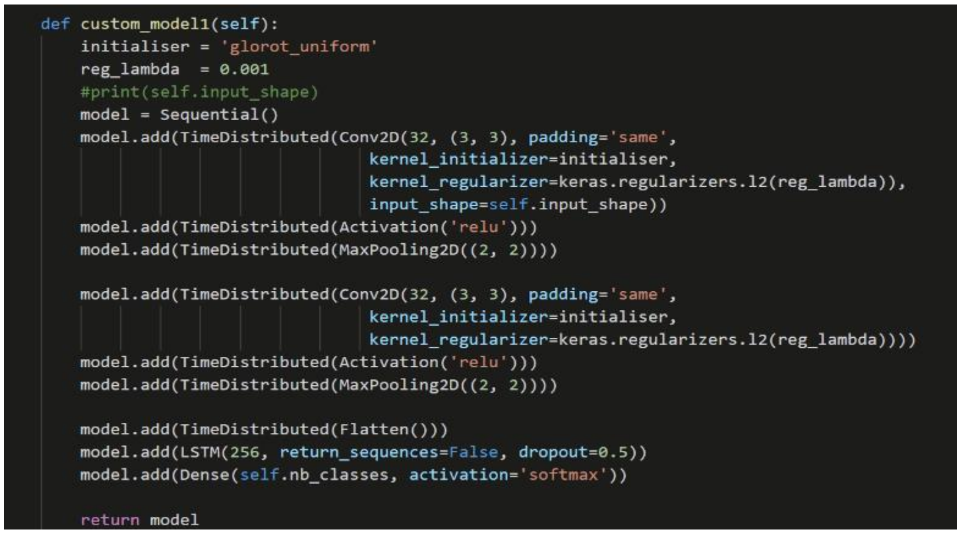


Figure 2: Model 1 graph

First, we started working on the video to text conversion without going into the detection techniques and methods.

With the help of UCF dataset, we managed to build the code that preprocess the videos (organize them and extract the frames from each video using ffmpeg tool), we also built an initial small CNN + LSTM model that we used in training.



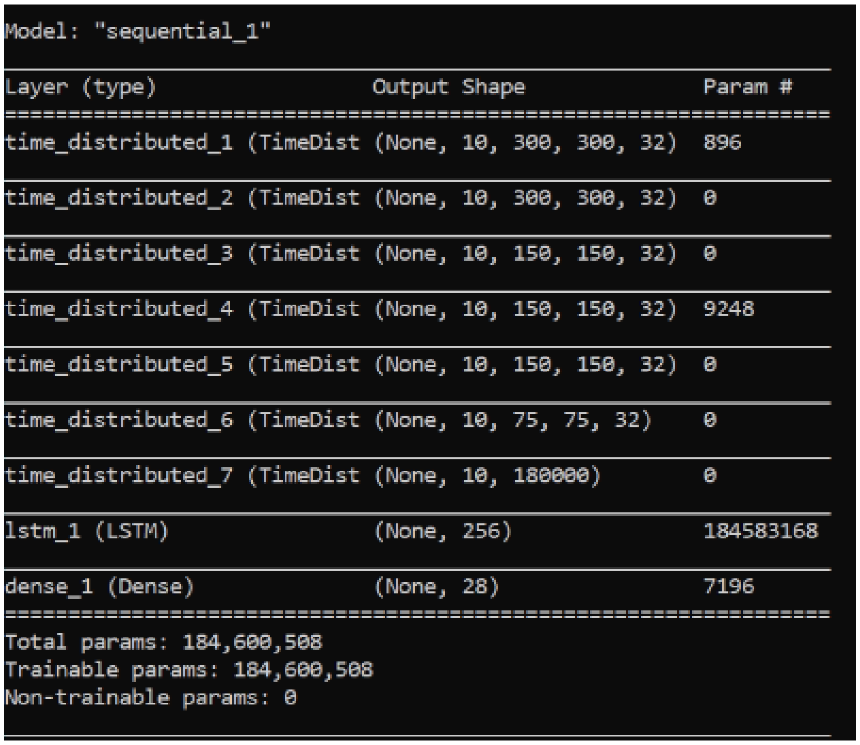
```
def custom_model1(self):
    initialiser = 'glorot_uniform'
    reg_lambda  = 0.001
    #print(self.input_shape)
    model = Sequential()
    model.add(TimeDistributed(Conv2D(32, (3, 3), padding='same',
                                     kernel_initializer=initialiser,
                                     kernel_regularizer=keras.regularizers.l2(reg_lambda)),
                                     input_shape=self.input_shape))
    model.add(TimeDistributed(Activation('relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))

    model.add(TimeDistributed(Conv2D(32, (3, 3), padding='same',
                                     kernel_initializer=initialiser,
                                     kernel_regularizer=keras.regularizers.l2(reg_lambda))))
    model.add(TimeDistributed(Activation('relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))

    model.add(TimeDistributed(Flatten()))
    model.add(LSTM(256, return_sequences=False, dropout=0.5))
    model.add(Dense(self.nb_classes, activation='softmax'))

    return model
```

Figure 3: The initial custom model for training

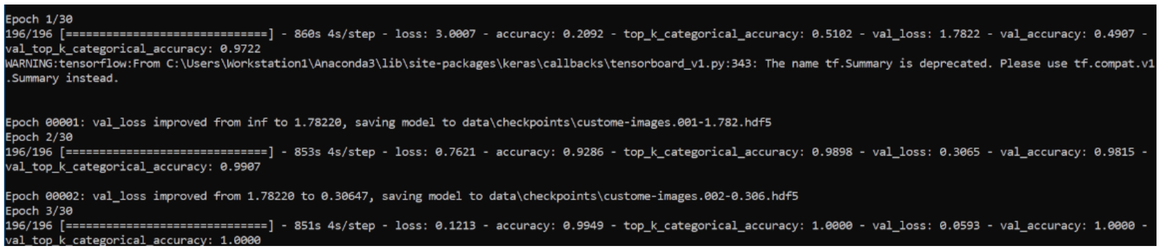


```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
time_distributed_1 (TimeDist (None, 10, 300, 300, 32)  896
_________________________________________________________________
time_distributed_2 (TimeDist (None, 10, 300, 300, 32)  0
_________________________________________________________________
time_distributed_3 (TimeDist (None, 10, 150, 150, 32)  0
_________________________________________________________________
time_distributed_4 (TimeDist (None, 10, 150, 150, 32)  9248
_________________________________________________________________
time_distributed_5 (TimeDist (None, 10, 150, 150, 32)  0
_________________________________________________________________
time_distributed_6 (TimeDist (None, 10, 75, 75, 32)    0
_________________________________________________________________
time_distributed_7 (TimeDist (None, 10, 180000)        0
_________________________________________________________________
lstm_1 (LSTM)                (None, 256)               184583168
_________________________________________________________________
dense_1 (Dense)              (None, 28)                7196
=================================================================
Total params: 184,600,508
Trainable params: 184,600,508
Non-trainable params: 0
_________________________________________________________________
```

Figure 4: The custom model architecture

To try this model we took a segment that belongs to one signer and started with it to see whether our concept is capable of learning the patterns or not; each video of the dataset has around 30 frames we decided to pick 10 frames from each video as a start also resized all frames to be (300 x 300) this makes our input shape = 10 x 300 x 300 x 3, our model performed naively well and had an accuracy of 100% at training and validation, when we tried this model on a video that was preprocessed and had its frames extracted we got a false prediction of the sign, we believe that 10 frames from each video was not a good choice so we are changed it to be near the minimum number of frames at the videos in the dataset which is 30 frames and it did perform really well.

The next step is to train the model with 4 signers' videos, enhance and tune a model to get the best results.



```
Epoch 1/30
196/196 [==============================] - 860s 4s/step - loss: 3.0007 - accuracy: 0.2092 - top_k_categorical_accuracy: 0.5102 - val_loss: 1.7822 - val_accuracy: 0.4907 -
val_top_k_categorical_accuracy: 0.9722
WARNING:tensorflow:From C:\Users\Workstation1\Anaconda3\lib\site-packages\keras\callbacks\tensorboard_v1.py:343: The name tf.Summary is deprecated. Please use tf.compat.v1
.Summary instead.


Epoch 00001: val_loss improved from inf to 1.78220, saving model to data\checkpoints\custome-images.001-1.782.hdf5
Epoch 2/30
196/196 [==============================] - 853s 4s/step - loss: 0.7621 - accuracy: 0.9286 - top_k_categorical_accuracy: 0.9898 - val_loss: 0.3065 - val_accuracy: 0.9815 -
val_top_k_categorical_accuracy: 0.9907

Epoch 00002: val_loss improved from 1.78220 to 0.30647, saving model to data\checkpoints\custome-images.002-0.306.hdf5
Epoch 3/30
196/196 [==============================] - 851s 4s/step - loss: 0.1213 - accuracy: 0.9949 - top_k_categorical_accuracy: 1.0000 - val_loss: 0.0593 - val_accuracy: 1.0000 -
val_top_k_categorical_accuracy: 1.0000
```

Figure 5: Snapshot from the training process

We added multiple Conv2D layers to extract better and more features that the LSTM can learn from and added another LSTM that takes the sequence from its prior LSTM and produces a vector to the output Dense layer.

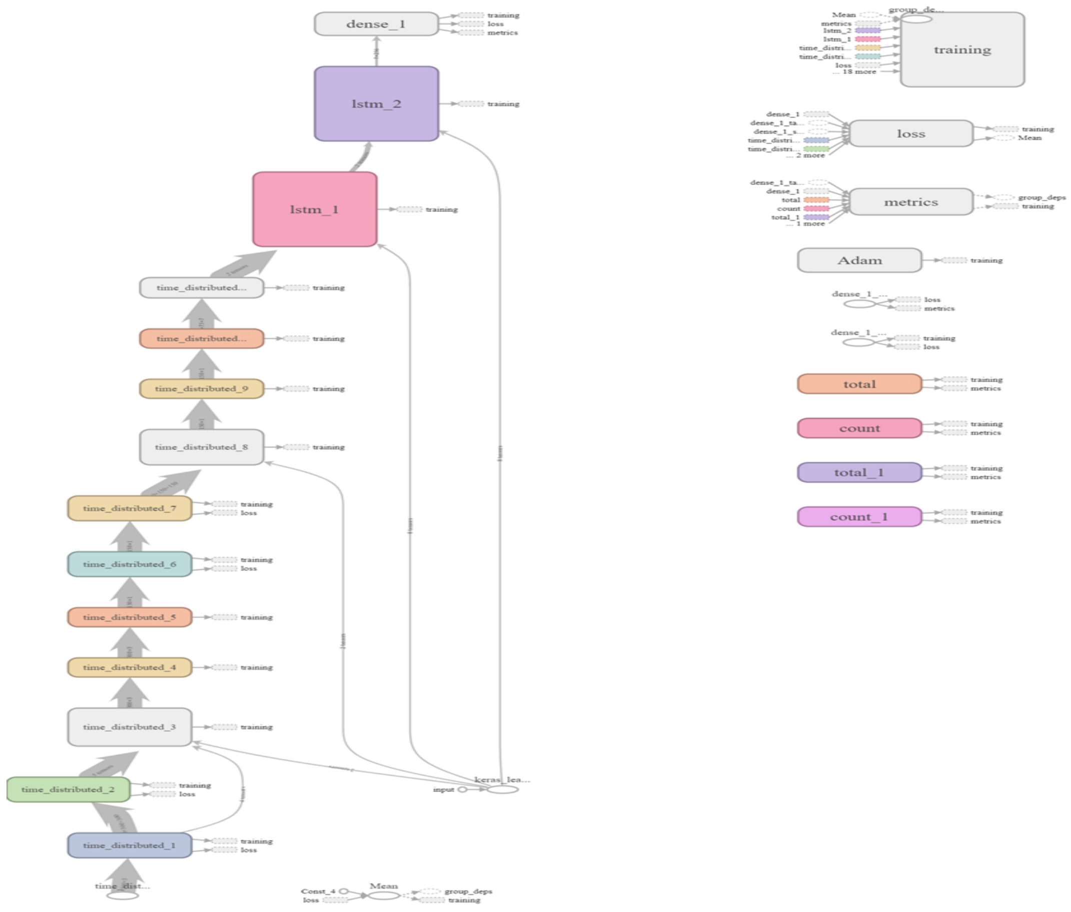


Figure 6: The modified custom model architecture graph

The Previous Model Did not Perform significantly better that than the first one so we used a similar architecture introduced in an Action Recognition Paper called Very Deep Convolutional Networks for Large-Scale Image Recognition**,** and used a Bidirectional LSTM at the end as advised by the Deep ASL Paper and this is our Final Model Architecture.

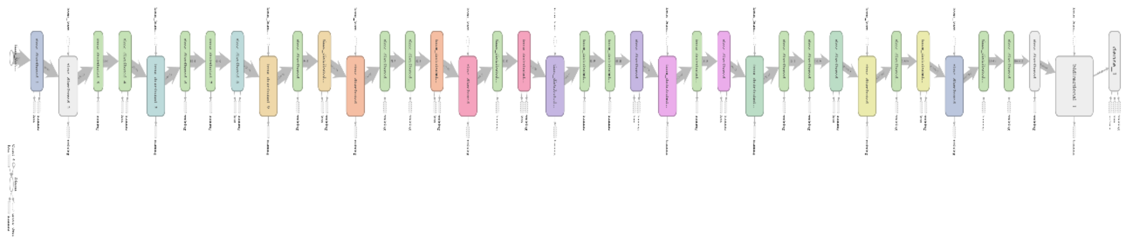
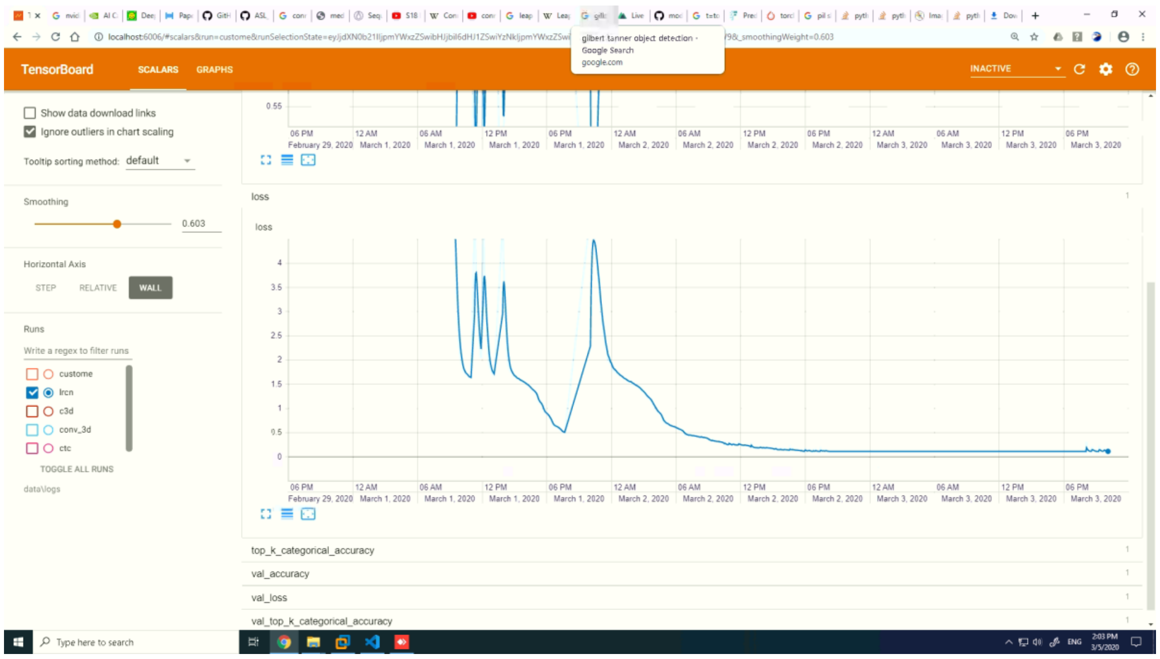


Figure 7:

Training this model took very large time, but was robust and has better results than the priors so we saved its weights to use in the next step.

The final step in this model is to build a script that takes a video as frames and pass every sequence of frames to our model to be interpreted.

The following segment shows part of the training process.


TensorBoard SCALARS GRAPHS INACTIVE

- [ ] Show data download links
- [x] Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.603

Horizontal Axis: STEP RELATIVE [✓ WALL]

Runs

Write a regex to filter runs

- [ ] custome
- [x] lrcn
- [ ] c3d
- [ ] conv_3d
- [ ] ctc

TOGGLE ALL RUNS

data\logs

loss

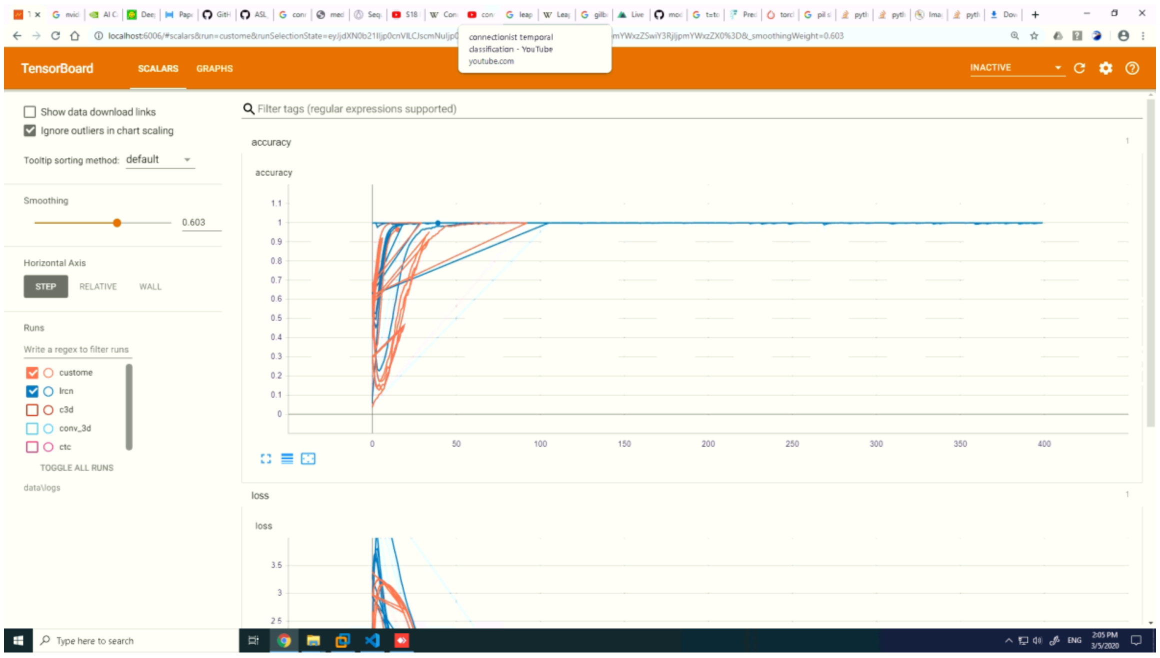
top_k_categorical_accuracy

val_accuracy

val_loss

val_top_k_categorical_accuracy

Figure 8: Training process of the enhanced model through days of fine tuning.



☐ Show data download links
☑ Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing 0.603
Horizontal Axis [✓ STEP] RELATIVE WALL
Runs
☑ custome
☑ lrcn
☐ c3d
☐ conv_3d
☐ ctc

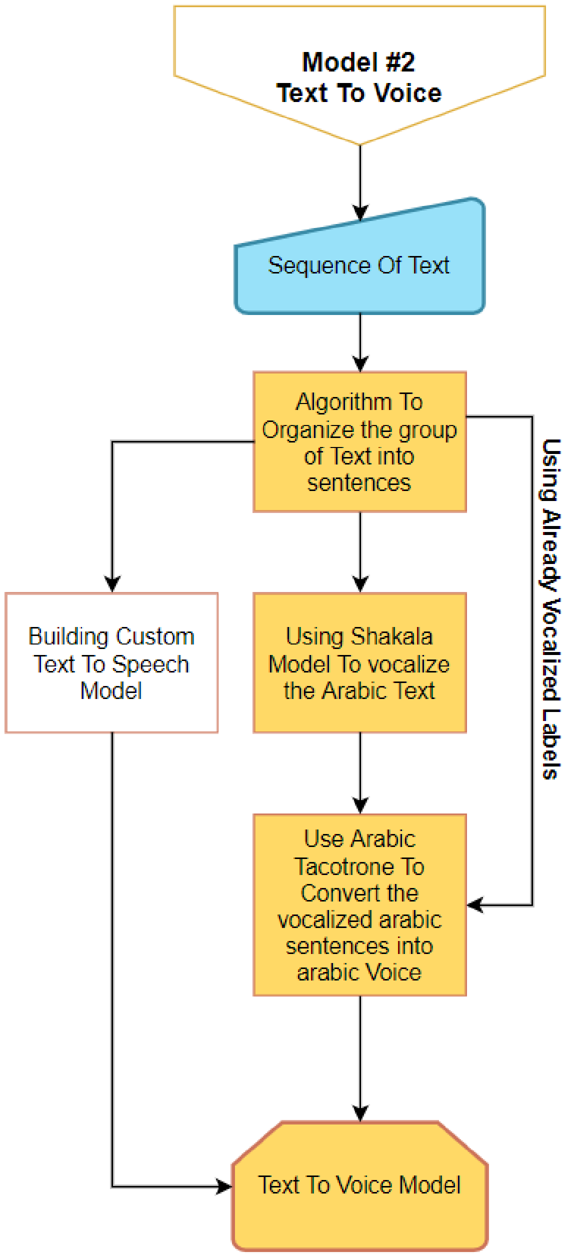
Figure 9: results of the different model we have tried

Model #2

The second model converts Arabic text to Arabic voice to do so, we used the help of two separate models:

- First, we vocalize the text using a model called Shakkala; this model takes in Arabic unvocalized text and returns the vocalized version of this text.
- Second, we used an Arabic text to voice model that is based on tacotrone model; this model takes in the vocalized text and produces generated Arabic voice as output.

We had several other theories on how to generate the voice without the need to go deep into several models, we thought of building a model that directly generates speech from the received video frames without the need for the text middle step, but we faced the problem of the dataset structure. We also thought of building a model to take the unvocalized text



Model #2
Text To Voice
Sequence Of Text
Algorithm To Organize the group of Text into sentences
Using Already Vocalized Labels
Building Custom Text To Speech Model
Using Shakala Model To vocalize the Arabic Text
Use Arabic Tacotrone To Convert the vocalized arabic sentences into arabic Voice
Text To Voice Model

Figure 10: Model 2 graph

as input and result in Arabic voice without the need for the vocalization method. In the end we decided to stick with our first approach as it has given us the best results for now; and leave the other approaches for our improvements phase.

Deployment

Deploying a Deep learning model is not an easy task, especially in our case where we have multiple models running simultaneously in addition to the fact that these models need to be running in real time as well as being portable e.g. (Phone).

Available Deployment Approach:

we buy a high computation server to deploy our model on, this way the model can be accessed by mobile applications or normal websites without being slow.

Pros:

1. Compatibility with every software that connects to internet
2. This way any lightweight app that runs on all mobile phone would be able to get the service, this way any user with a connection to internet would get the service.
3. This increases the feasibility and portability and that the core advantage

Cons:

1. Internet connection that handles streaming is not available everywhere like in our target case, Egypt.

After many researches and experiments we chose to go with this deployment method, and the following steps are to be made:

- Create a website using flask that can talk directly to the model, it sends a video feed to the model and receives the interpretations back.
- Look up a server that is affordable to us in the meantime, to deploy our system on.
- Last step is making an API that takes the video feed from the Flask website and gives us back the interpretations.

Current Results

For our prototype, we built a simple web page using HTML and JavaScript. The web page receives a video then processes it extracting the frames and saving them into a list, after that it calls our Prediction model passing in the preprocessed frames to predict the corresponding word (label) for the given video from our labels list and return the Arabic word as a message through our page with the corresponding voice for the predicted word.

Future Work

Our current results aren't perfect and we have many future plans for our project and device to be as helpful as possible to everyone in the deaf and mute community. For now, we are working towards two main aspects:

1- Build a more consistent model that receives real time signs which it can detect the key points of the signer's hands and face, extract those key points and save them as a dataset of landmarks. The model then trains on those key points landmarks and pass straight to the prediction algorithm where it can predict the correct corresponding label to each word and generate the correct voice for that word in Arabic. It should be a fast working model that's able to process in real time.

2- Deploy the new model on the hardware device using Raspberry Pi as our main controller with the needed gadgets, a camera, microphone, speaker, etc. the users will use this portable device whenever they need to talk to anyone that is not familiar with Arabic sign language easily and reliably.

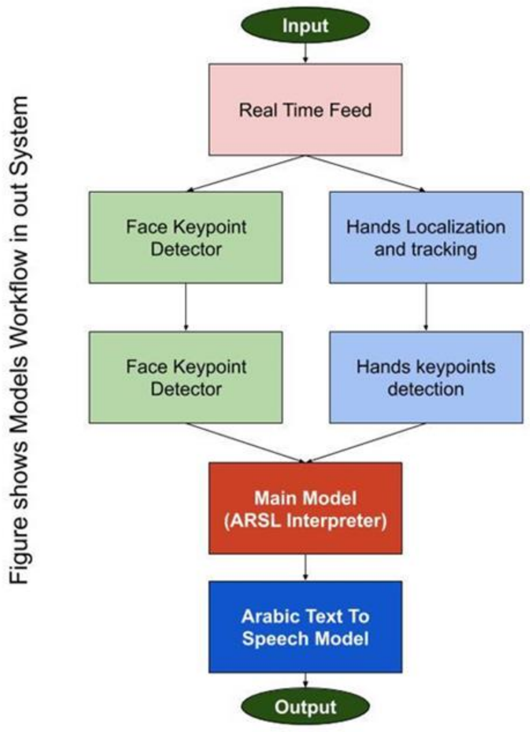


Figure shows Models Workflow in out System
Input
Real Time Feed
Face Keypoint Detector
Hands Localization and tracking
Face Keypoint Detector
Hands keypoints detection
Main Model (ARSL Interpreter)
Arabic Text To Speech Model
Output

Figure 2.11: The final project flow graph

Figure 2.11, show the desired final workflow graph that we are aiming at when we finish our work. It’s a smooth workflow and consistent operation between models, leading to accurate and fast results in real time.