Artificial Intelligence and Robotics

KF6007

"An Intelligent Interactive System"

British Sign Language gesture recognition for hearing impaired users.

Sandra Czernik

W19009505

Word Count: 2177

INTRODUCTION

1 in 6 people currently in the UK are found to have several types of hearing loss, with 900,000 persons being categorized as having 'severe' hearing loss. Current technology trying to tackle gesture recognition systems require specialised equipment such as gloves or depth cameras, however these are not widely available to the population.

The system allows a user to perform gestures which are output in a text format on the screen on a live camera. The LSTM product achieved an accuracy of 95% with loss of 0.0203 with a test size of 25%

A study conducted by Ahmed (KASAPBAŞI et al., 2022), used the Convolutional Neural Network to recognise American sign language with a "99.38% accuracy and a loss of 0.025", using a customised dataset which consisted of 26 letters; however the letters Z and J were represented by a still image.

Another study used the SVM algorithm to "recognise BSL with an accuracy of 99% (Quinn & Olszewska, 2019). This study consisted of a custom dataset of 2,600 images of signed letters from A to Z, with the letters Z and J also being taken at the last endpoint of the gesture. The HOG (Histogram of Oriented Gradients) technique was used which counts different occurrences of gradient orientations, however, it is very sensitive to image rotation and noise (Alhindi et al., n.d.).

NOVEL AI TECHNIQUES AND APPROACHES

Current studies have been found to use algorithms such as CNN and SVM to recognise still images of sign language, however there were some disadvantages with those solutions implemented as "small movements" weren't possible for letters such as J and Z, which is where the inspiration for the following final product was implemented. A study conducted by Ahmed (Gomaa & Gla Elrayes, n.d.), used CNN and LSTM to recognise Egyptian sign language with an accuracy of 90% and 72%, however a solution for British Sign Language with the combination of Media Pipe holistic key points and LSTM layers has not yet been developed, for a conference call solution.

PRODUCT DEVELOPMENT STAGES

The final product is a British Sign Language gesture recognition software which is able to recognize different words the user is signing and prints what it thinks the word is on the screen for other users to see. This tool would be useful in a conference call as it bridges the gap in communication between signing users who may be visually impaired and users without a disability.

The RNN architecture is known to take into consideration the previous output into the new input and also allows for hidden states. A typical RNN neural network usually has an input, a neural network to look at data, and an output, which can be seen in Figure 1.

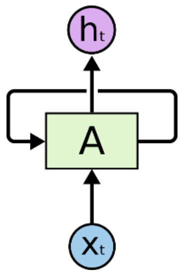


*Figure 1 RNN Neural Network Diagram (*Understanding LSTM Networks -- Colah's Blog*, n.d.).*

The main neural network used in the RNN architecture to accomplish this is the LSTM neural network, which is more standardized for many tasks, such as using previous frames in a live camera to predict what the current frame is, which in this case is essential for a product which uses gesture recognition with small movements to translate a word being signed into plain text on the screen. An example of why RNN architecture is not enough in the case of the final system because there are too many frames for RNN to learn from .A solution which can bridge a large gap like this was needed, with LSTM being the most appropriate solution.

The LSTM network was first introduced by (S. Hochreiter & Schmidhuber, 1997), which was developed to solve the problem of RNN being unable to learn information that may be too far back. In Figure 3, instead of just having one simple structure layer in a module, an LSTM will have four, which some are able to interact with each other.

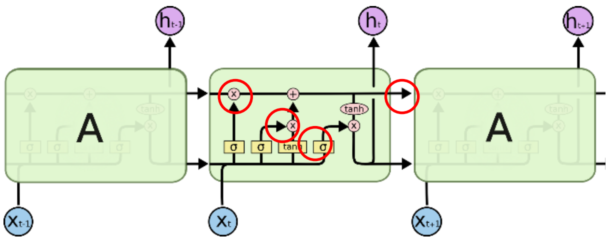


*Figure 2 LSTM layers and repeating modules (*Understanding LSTM Networks -- Colah's Blog*, n.d.).*

The main part of LSTM's is the current cell state, which can be seen as the top horizontal line in Figure 3. The module is able to remove or add information to the current state of that specific module, for example adding information (also known as letting information through to the next module) as a simple operation of multiplication.

There are sigmoid layers which are able to pass information through either as a 0 or 1, where 0 means no information should be passed through, and 1 means to let all of the information pass through, there are three of those gates which can be seen in Figure 4, represented by an 'x'(*Understanding LSTM Networks -- Colah's Blog*, n.d.).

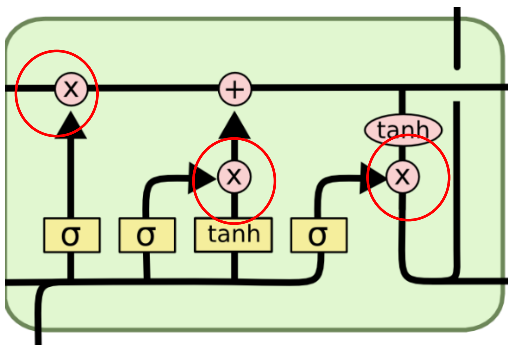


Figure 3 Cropped image of Figure 3.

TKinter was considered during the development stage of the product, however was not deemed necessary as CV2 provided all the required elements such as texts and rectangles that could be displayed on the live webcam video to print prediction results.

VSCode was the primary platform used to program the software, with the use of Anaconda and the Jupyter Notebook to separate code into blocks which made debugging the code much more efficient. Those were chosen because the language used in the product was python, which required an environment to be set up using Anaconda, which was later used to set up the kernel in the Jupyter Notebook.

PREPROCESSING DATA, LABELS AND FEATURES

The dataset used was a customized data set, which was collected using a live camera along with a package called Media Pipe, which allowed for hand and face tracking. Figure 5 displays an example of this, where the different "keypoints" seen on the figure were extracted and placed into an array, where each array would correspond to different parts of the body, such as the left and right hand. The x, y and z coordinates were each placed in those arrays so they can be stored for later use.

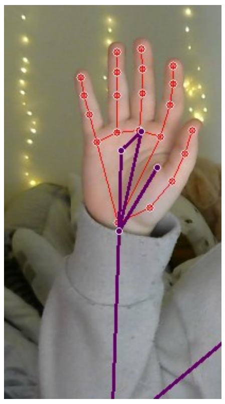
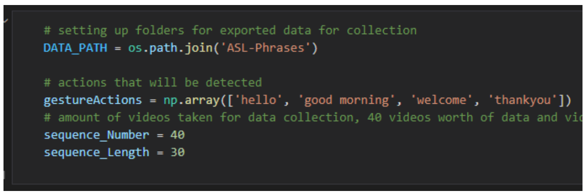


Figure 4 Own image. Holistic key point demonstration on live camera 21 key points can be seen on the hand.

Folders were then created for each gesture with 40 videos worth 30 frames each (for each gesture), as seen in Figure 6.



```
# setting up folders for exported data for collection
DATA_PATH = os.path.join('ASL-Phrases')

# actions that will be detected
gestureActions = np.array(['hello', 'good morning', 'welcome', 'thankyou'])
# amount of videos taken for data collection, 40 videos worth of data and vi
sequence_Number = 40
sequence_Length = 30
```

Figure 5 Folder creation for key point data collection.

These folders with the collected dataset were then preprocessed, to create labels and features, ending with a .npy file format. This data would eventually be categorized into an array with 4 elements, where hello would be [1,0,0,0], good morning would be [0,1,0,0] and so on..

While unpacking the train_test_split function, the final test size was set to 0.25 (25%), as it achieved the highest accuracy out of all the values tested, which was the final step in preprocessing the data captured.

IMPLEMENTATION

The use of CV2 elements allowed for an interface to be developed, with a simple text output which can be seen at the bottom of the live camera view as seen in Figure 7.

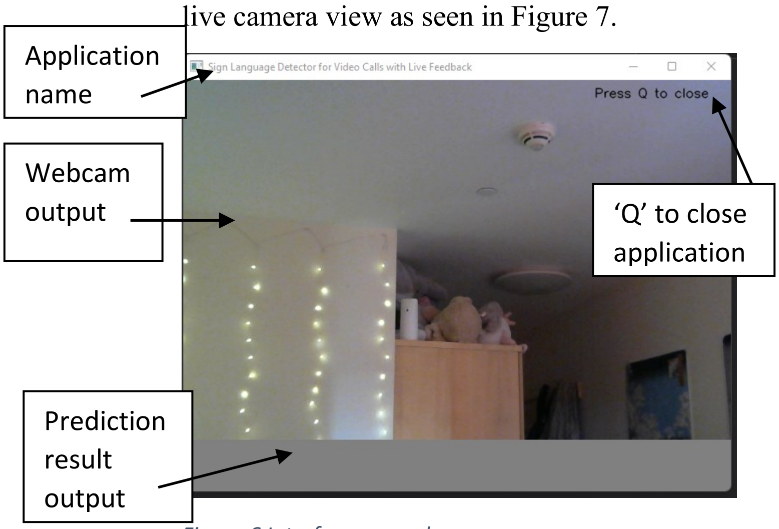


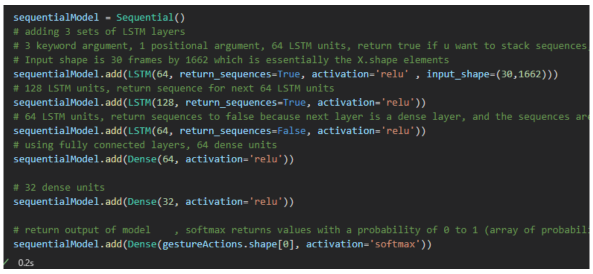
Figure 6 Interface example.

The basis of the text outputting the predicted result can be seen below with some pseudocode.

```
IF SEQUENCE = 30
  PREDICT RESULT BASED ON SEQUENCE
   PRINT GESTURE ACTION
    IF RESULT IS WITHIN THRESHOLD
     IF SENTENCE ARRAY > 0
     IF GESTURE ACTION != CURRENT SENTENCE    ARRAY
     PRINT PREDICTED SENTENCE
    ELSE PRINT PREDICTED SENTENCE
   IF SETENCE LENGTH > 3
    SET SENTENCE ARRAY == 0
```

This would print out a prediction if it is above the threshold, which is a variable than can be set from 0.0 to 1.0 and then print it in an array and concatenate words together. However if the array is over 3 words, then it would be set to 0 to stop cluttering from occurring and to stop a large array of words being created which could impact the performance of the product.

A sequential model was used, with 3 sets of LSTM layers and two dense layers as seen in Figure 8.



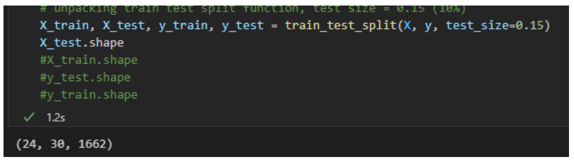
```
sequentialModel = Sequential()
# adding 3 sets of LSTM layers
# 3 keyword argument, 1 positional argument, 64 LSTM units, return true if u want to stack sequences
# Input shape is 30 frames by 1662 which is essentially the X.shape elements
sequentialModel.add(LSTM(64, return_sequences=True, activation='relu' , input_shape=(30,1662)))
# 128 LSTM units, return sequence for next 64 LSTM units
sequentialModel.add(LSTM(128, return_sequences=True, activation='relu'))
# 64 LSTM units, return sequences to false because next layer is a dense layer, and the sequences are
sequentialModel.add(LSTM(64, return_sequences=False, activation='relu'))
# using fully connected layers, 64 dense units
sequentialModel.add(Dense(64, activation='relu'))

# 32 dense units
sequentialModel.add(Dense(32, activation='relu'))

# return output of model    , softmax returns values with a probability of 0 to 1 (array of probabil
sequentialModel.add(Dense(gestureActions.shape[0], activation='softmax'))
✓ 0.2s
```

Figure 7 Sequential model layers

The input shape in the first layer translates to the X_test elements, which was created using the train_test_split function, as seen in Figure 9. 30 is the number of frames for each gesture, and 1662 in the total amount of key points from the left and right hand, as well as the face and the pose connections using the holistic pipeline model. 'Relu' will output the input directly if it is positive, otherwise it will output 0.

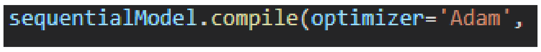


```
    # unpacking train test split function, test size = 0.15 (10%)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15)
    X_test.shape
    #X_train.shape
    #y_test.shape
    #y_train.shape
  ✓ 1.2s

(24, 30, 1662)
```

Figure 8 Example of the shape of X_test data.

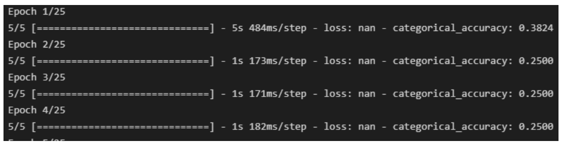
The second LSTM layer is made up of 128 units and will return the sequences of those to the next layer which has been condensed to 64 units. This layer will not return any sequences to the next. The next few dense layers will use the fully connected units from LSTM to create fully connected neural network neurons which are then added to an array where 'softmax' will normalize the output to a probability distribution.



```
sequentialModel.compile(optimizer='Adam',
```

Figure 9 Optimizer choice.

Adam was the final choice as the optimizer for the model, as options such as SGD produced inaccurate results compared to the chosen optimizer, as seen in Figure 11.



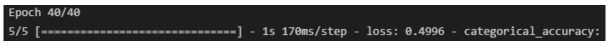
```
Epoch 1/25
5/5 [==============================] - 5s 484ms/step - loss: nan - categorical_accuracy: 0.3824
Epoch 2/25
5/5 [==============================] - 1s 173ms/step - loss: nan - categorical_accuracy: 0.2500
Epoch 3/25
5/5 [==============================] - 1s 171ms/step - loss: nan - categorical_accuracy: 0.2500
Epoch 4/25
5/5 [==============================] - 1s 182ms/step - loss: nan - categorical_accuracy: 0.2500
```

Figure 10 SGD output result.

EXPERIMENTAL PARAMETERS

Before the final percentage of the testing size being implemented, several other values were tested, such as 30%, 10% and 5% for contrast.

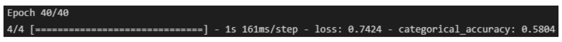
A test size of 5% came at around 77% accuracy as seen in Figure 12, however, was was unable to predict most of the gestures once tested with a live camera.



```
Epoch 40/40
5/5 [==============================] - 1s 170ms/step - loss: 0.4996 - categorical_accuracy:
```

Figure 12 10% test size

With a test size of 10% it was noticeable that the loss every epoch seemed to be very unstable, jumping from 3.6 up to 84.07 and then back to 1.4 after 40 epochs. The accuracy for this test size was 38% as overfitting may have occurred. 30% test size also had a slightly increased accuracy rate of 58%, as seen in Figure 13. The loss for this test size was 0.74.



```
Epoch 40/40
4/4 [==============================] - 1s 161ms/step - loss: 0.7424 - categorical_accuracy: 0.5804
```

Figure 13 30%

CRITICAL EVALUATION

The Adam optimizer proved to be very accurate with an accuracy of 0.95 (95%) and a loss of 0.0203, with just 50 epochs, as seen in Figures 14 and 15 respectively.

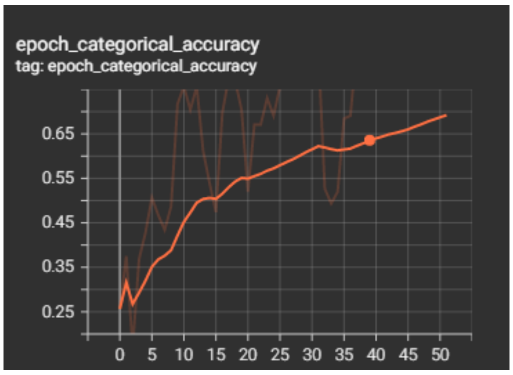


Figure 14 categorical accuracy

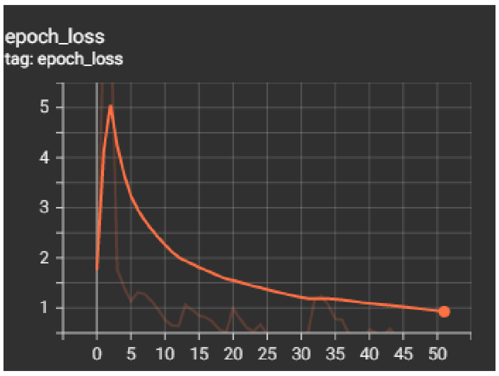
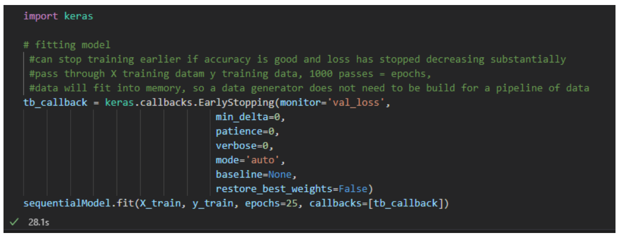


Figure 15 epoch loss.

These graphs were made using a tensor board package within anaconda, so when the model started training, the graph would update real time using the training data from the train folder, allowing to see any potential early overfitting and to overall see how the model was performing in real-time.

A callback variable was also created during the training process to account for early stopping once the loss has become negligible, as seen in Figure 16.

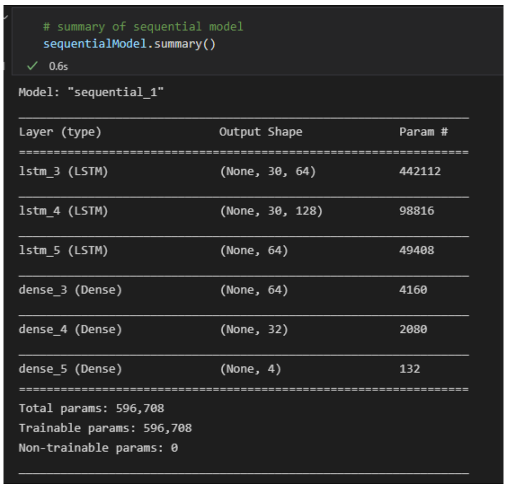


```
import keras

# fitting model
 #can stop training earlier if accuracy is good and loss has stopped decreasing substantially
 #pass through X training datam y training data, 1000 passes = epochs,
 #data will fit into memory, so a data generator does not need to be build for a pipeline of data
tb_callback = keras.callbacks.EarlyStopping(monitor='val_loss',
                                            min_delta=0,
                                            patience=0,
                                            verbose=0,
                                            mode='auto',
                                            baseline=None,
                                            restore_best_weights=False)
sequentialModel.fit(X_train, y_train, epochs=25, callbacks=[tb_callback])
✓ 28.1s
```

Figure 16 Early stopping variable.

The summary of the sequential model included just over 500,000 parameters, as seen in Figure 17.

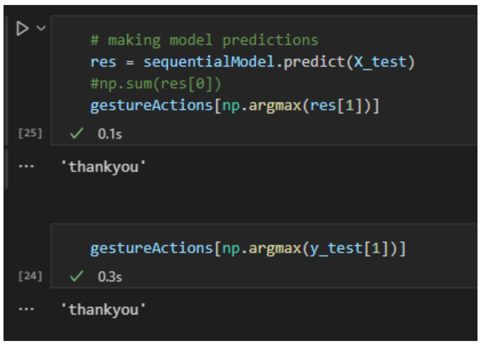


```
# summary of sequential model
sequentialModel.summary()
✓ 0.6s

Model: "sequential_1"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
lstm_3 (LSTM)               (None, 30, 64)            442112
_________________________________________________________________
lstm_4 (LSTM)               (None, 30, 128)           98816
_________________________________________________________________
lstm_5 (LSTM)               (None, 64)                49408
_________________________________________________________________
dense_3 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_4 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_5 (Dense)             (None, 4)                 132
=================================================================
Total params: 596,708
Trainable params: 596,708
Non-trainable params: 0
_________________________________________________________________
```

Figure 17 model summary.

The first test was done to see if the model weights were worth saving by predicting the X and y testing data with the prediction result which an example of can be seen below in Figure 18. This result meant that the corresponding array 1 of a specific gesture action was predicted as the same action.

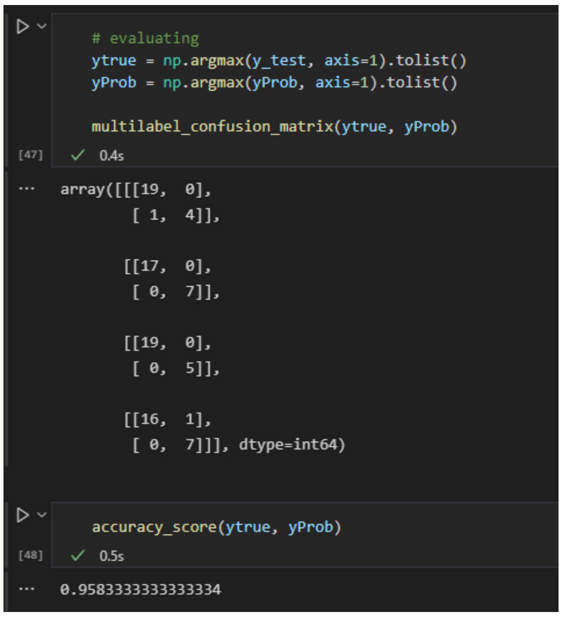


```
# making model predictions
res = sequentialModel.predict(X_test)
#np.sum(res[0])
gestureActions[np.argmax(res[1])]
[25]  ✓ 0.1s
'thankyou'

gestureActions[np.argmax(y_test[1])]
[24]  ✓ 0.3s
'thankyou'
```

Figure 18 simple test to see if weights could be saved.

A more complex evaluation was made using a multilabel confusion matrix to display false and true positives and negatives, as seen in Figure 19. The ideal result would be to have two values in the opposite corners, however as seen in the figure there is a discrepancy, which is normal for an accuracy of 95%.



```
# evaluating
ytrue = np.argmax(y_test, axis=1).tolist()
yProb = np.argmax(yProb, axis=1).tolist()

multilabel_confusion_matrix(ytrue, yProb)
[47]  ✓ 0.4s
array([[[19,  0],
        [ 1,  4]],

       [[17,  0],
        [ 0,  7]],

       [[19,  0],
        [ 0,  5]],

       [[16,  1],
        [ 0,  7]]], dtype=int64)

accuracy_score(ytrue, yProb)
[48]  ✓ 0.5s
0.9583333333333334
```

Figure 19 multilabel confusion matrix and accuracy score based on that.

This prediction result would then be evaluated against the key points on a live camera and if the last 30 frames match those key points which the model was trained on, it would output a text on the screen with the word that the model has predicted.

STRENGTHS, LIMITATIONS AND FUTURE CONSIDERATIONS

The strength of using LSTM over a non-recurring neural network is that it was able to learn information which was further back in the network and could use that information to learn, whereas a simple RNN would not be able to. This allowed for the model to have an accuracy of 95% and used 30 frames of data allowing for more complex gestures rather than still images. A possible disadvantage to this is that very similar gestures, for example hello and bye, as seen in Figure 20 and 21, would be quite difficult to interpret as they are very similar.

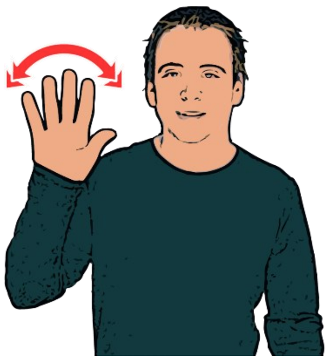


*Figure 20 'Hello' gesture (*British Sign Language Dictionary | Hello, *n.d.)*

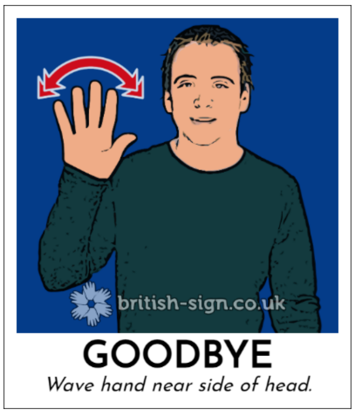


*Figure 21 'Goodbye' gesture (*British Sign Language Dictionary | Goodbye, *n.d.)*

A solution to this problem which was attempted in the final product was to use the key points of the face and to say which word you are signing, so the model could also train not just on the hand gesture, but also on the face especially the mouth key points which could aid in distinguishing between two very similar gestures.

In the future, more gestures could be added for the model to train on, to make it more accessible and appropriate for users to use in a video call setting. Another consideration would be to use different users for the data collection to allow for variety in gestures to be recorded and trained on.

REFERENCES

Alhindi, T. J., Kalra, S., Ng, K. H., Afrin, A., & Tizhoosh, H. R. (n.d.). *Comparing LBP, HOG and Deep Features for Classification of Histopathology Images*. Retrieved April 16, 2022, from http://kimia.uwaterloo.ca/

British Sign Language Dictionary | Goodbye. (n.d.). Retrieved April 17, 2022, from https://www.british-sign.co.uk/british-sign-language/how-to-sign/goodbye/

British Sign Language Dictionary | Hello. (n.d.). Retrieved April 17, 2022, from https://www.british-sign.co.uk/british-sign-language/how-to-sign/hello/

Gomaa, A. A., & Gla Elrayes, R. (n.d.). *Egyptian Sign Language Recognition Using CNN and LSTM*.

Hochreiter, J. (1991). *DIPLOMARBEIT IM FACH INFORMATIK Untersuchungen zu dynamischen neuronalen Netzen*.

Hochreiter, S., & Schmidhuber, J. (1997). *LONG SHORT-TERM MEMORY*. http://www.bioinf.jku.at/publications/older/2604.pdf

Introduction - Deaf awareness - LibGuides at University of South Wales. (n.d.). Retrieved April 16, 2022, from https://libguides.southwales.ac.uk/deaf_awareness

KASAPBAŞI, A., ELBUSHRA, A. E. A., AL-HARDANEE, O., & YILMAZ, A. (2022). DeepASLR: A CNN based human computer interface for American Sign Language recognition for hearing-impaired individuals. *Computer Methods and Programs in Biomedicine Update*, *2*, 100048. https://doi.org/10.1016/J.CMPBUP.2021.100048

Quinn, M., & Olszewska, J. I. (2019). British sign language recognition in the wild based on multi-class SVM. *Proceedings of the 2019 Federated Conference on Computer Science and Information Systems, FedCSIS 2019*, 81–86. https://doi.org/10.15439/2019F274

The Unreasonable Effectiveness of Recurrent Neural Networks. (n.d.). Retrieved April 17, 2022, from http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Understanding LSTM Networks -- colah's blog. (n.d.). Retrieved April 17, 2022, from https://colah.github.io/posts/2015-08-Understanding-LSTMs/