FINAL CAPSTONE REPORT

Course Coordinator: Marcos Bittencourt

Group Members

Shrey Raval - 100730265
Jigar Patel - 100730261
Tarunjot Singh - 100766653
Nikhil Singh - 100766644

Capstone Term 2

Executive Summary:

Our main idea revolves around helping deaf people to understand sign language without learning it. Simply by performing such signs and being interpreted and translated to letters and words can help people understand it with high ease. The main population this technology can help are the people suffering difficulty in learning sign language.

This solution can be applied almost everywhere including educational institutions, government offices to commercial places. This can help in easy learning and working activities. Also, including future possible developments during seminars and other public speaking events, real-time prediction systems can provide a very efficient method of delivery.

Hand signs in general, follow the following structure based on which, our model will be trained, and outputs will be predicted

Model Structure:

For obtaining accurate results, we decided to go with 2 final models after all the testing and modelling. They are CNN and LeNet-5. Both models have their own structure and procedure for data fitting and testing. Model Structure are Discussed below.

CNN:

A Convolutional Neural Network (CNN) is a deep learning algorithm that can identify and distinguish features in computer vision videos. This is a multi-layer neural network built to process visual signals and perform tasks such as image recognition, segmentation and object detection.

Architecture

CNN's architecture is influenced by the structure and flexibility of the visual cortex and is designed to mimic the process of neuron interaction within the human brain.

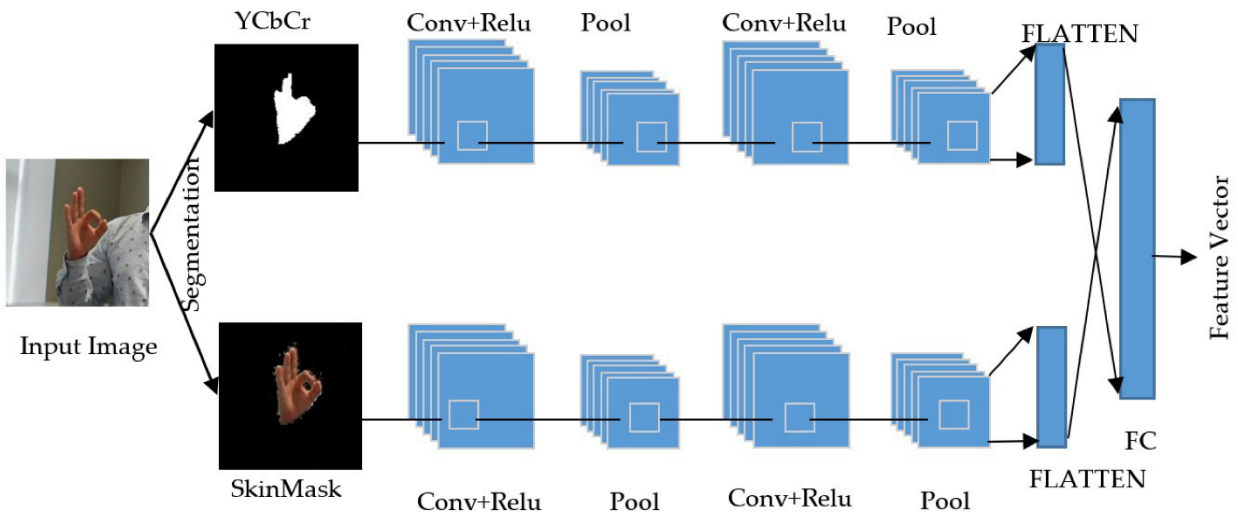
The neurons inside CNN are divided into a three-dimensional structure, with each group of neurons examining a small region or function of the picture. CNNs use layer projections to generate a final output that provides a probability score vector to reflect the possibility that a given function belongs to a certain class.

CNN is made up of several types of bricks.

Convolutional layer: It creates a function map to predict the class probabilities for each element by adding a philtre that scans the entire image, few pixels at a time.

Pooling layer: It helps to Scale down the amount of knowledge produced by the convolutionary layer for each function and preserving the most essential information.

CNN Architecture:



Detail summary of the Model Layers:

```
model = Sequential()

model.add(Conv2D(64, kernel_size = (3, 3), activation = 'relu', input_shape = (28, 28, 1)))
model.add(MaxPooling2D(pool_size = (2, 2)))

model.add(Conv2D(64, kernel_size = (3, 3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2, 2)))

model.add(Conv2D(64, kernel_size = (3, 3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2, 2)))

model.add(Flatten())
model.add(Dense(128, activation = 'relu'))
model.add(Dropout(0.2))

model.add(Dense(num_classes, activation = 'softmax'))
```

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 26, 26, 64)	640
max_pooling2d_1 (MaxPooling2	(None, 13, 13, 64)	0
conv2d_2 (Conv2D)	(None, 11, 11, 64)	36928
max_pooling2d_2 (MaxPooling2	(None, 5, 5, 64)	0
conv2d_3 (Conv2D)	(None, 3, 3, 64)	36928
max_pooling2d_3 (MaxPooling2	(None, 1, 1, 64)	0
flatten_1 (Flatten)	(None, 64)	0
dense_1 (Dense)	(None, 128)	8320
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 24)	3096

Total params: 85,912
Trainable params: 85,912
Non-trainable params: 0

Model includes 3 levels of Convolution layers with the output shape of (26, 26, 64) and total number of input parameters are 640. Further we have used max pooling to reduce the dimension of the image dataset which result in low computational cost for further processing. On top of that this layer also helps to extract the important piece of information from images. Moving further, this model has the second convolution layer followed by the third convolution layers. Coming towards the end there is a dropout feature which help to overcome the problem of model overfitting and then output is obtained.

Model Fitting:

```
model.compile(loss = keras.losses.categorical_crossentropy, optimizer = keras.optimizers.Adam(), metrics = ['accuracy'])
```

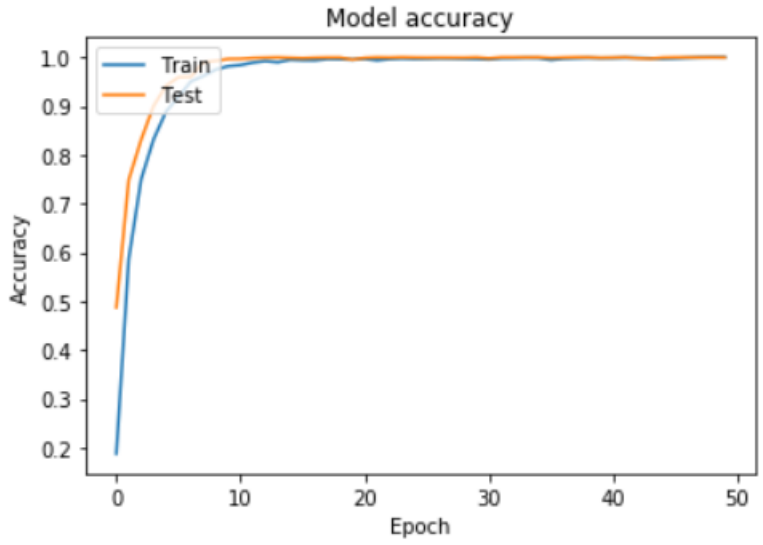
```
history = model.fit(x_train, y_train, validation_data = (x_test, y_test), epochs = epochs,
                    batch_size = batch_size)
```

Model Output:

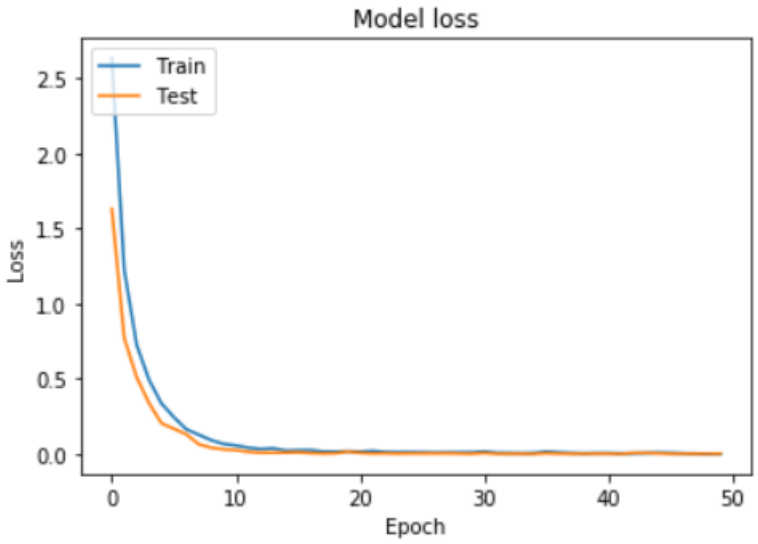
```
Epoch 49/50
19218/19218 [==============================] - 12s 612us/step - loss: 0.0014 - accuracy: 0.9996 - val_loss: 2.6242e-
accuracy: 0.9999
Epoch 50/50
19218/19218 [==============================] - 12s 610us/step - loss: 0.0012 - accuracy: 0.9997 - val_loss: 0.0023 -
```

Train Accuracy: 99.97%

Test Accuracy: 84.64%



Loss: 0.023



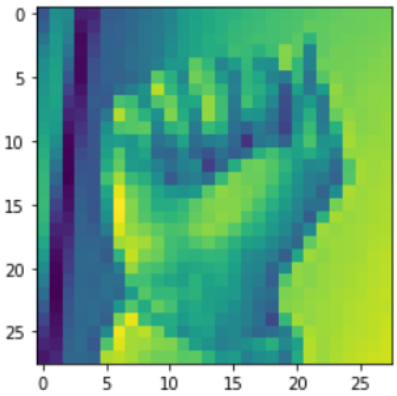
Prototype Predicted Output:

```
In [31]: image_0  = x_train[0]
         image_0 = image_0.reshape((1, 28, 28, 1))
         z = model.predict(image_0)
         alphabets = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'k', 'l', 'm', 'n', 'o', 'p',
         max_index = np.argmax(z[0])
         final_label = alphabets[max_index]

         print('Predicted alphabet: ', final_label)
         plt.imshow(x_train[0].reshape(28, 28))
```

```
Predicted alphabet:  a
```

```
Out[31]: <matplotlib.image.AxesImage at 0x1e888507ba8>
```



Pros:

- High accuracy in terms of machine learning models working with images and computer vision
- After training, predictions are comparatively faster
- Works better with more data points
- Flexible in terms of regression and classification problems

Cons:

- Very high in terms of computational costs
- If strong hardware is not present for computation, CNNs are slow to train
- Requires high amount of training data
- Consumes lot of time, hence comparatively worse in terms of time complexity

LeNET-5

Lenet-5 is a type of simple CNN which consists of 7 Layers. Out of these 7 Layers, 3 are the convolutional layers, 2 are pooling layers (sub-sampling), 1 Fully connected and lastly 1 output layer. We will be implementing the Le-net 5 architecture because the le-net architecture was initially developed for the recognition of handwritten digits and alphabets and is used by majority of U.S banks till Date to read cheques.

Following are the specifications of the layers:

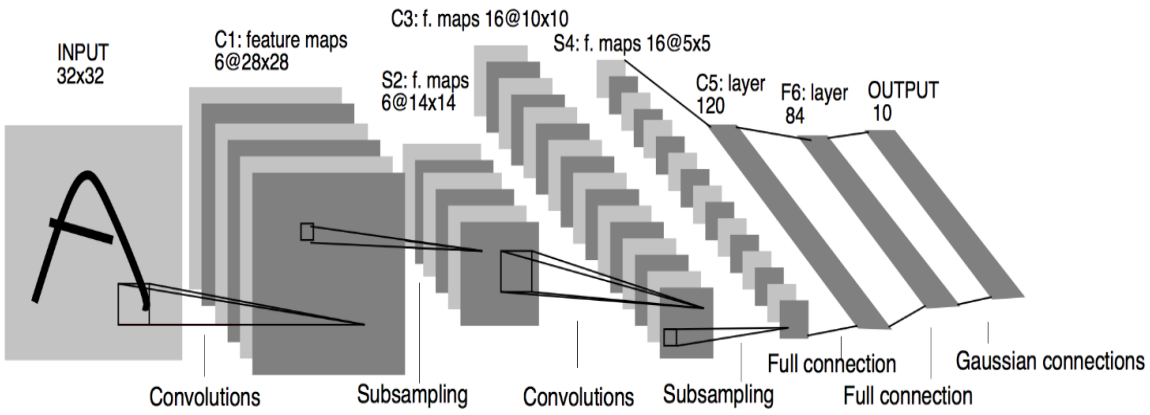
Convolutional layer will be using the 5x5 convolutions with stride.

Pooling Layer will consist of 2x2 Average Pooling

Activation Function throughout the entire network will be consisting of only tan h and sigmoid.

During the research regarding various architecture we came to know that as LeNet works with digit and character recognition in a great way it would also work in a similar way for our problem also as our scope includes the recognition of hand gestures.

LeNet-5 Architecture



Detailed Summary of Model Layers:

```
model = Sequential()

model.add(Conv2D(filters = 6, kernel_size = (5, 5), strides = 1, input_shape = (dim_x, dim_y, 1), activation = tf.nn.tanh))
model.add(MaxPooling2D(pool_size = 2, strides = 2))
model.add(Conv2D(filters = 16, kernel_size = (5, 5), strides = 1, activation = tf.nn.tanh))
model.add(MaxPooling2D(pool_size = 2, strides = 2))
model.add(Flatten())
model.add(Dense(120, activation = tf.nn.tanh))
model.add(Dense(84, activation = tf.nn.tanh))
model.add(Dense(classes, activation = tf.nn.softmax))
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 24, 24, 6)         156
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 12, 12, 6)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 8, 8, 16)          2416
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 4, 4, 16)          0
_________________________________________________________________
flatten (Flatten)            (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 120)               30840
_________________________________________________________________
dense_1 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_2 (Dense)              (None, 24)                2040
=================================================================
Total params: 45,616
Trainable params: 45,616
Non-trainable params: 0
```

Model includes 2 levels of Convolution layers out of which the 1st has the output shape of (24, 24, 6) and the 2nd one has and output shape of (8, 8, 16) and total number of input parameters are 156 and 2416 respectively. Further we have used max pooling that reduces the dimension of the image dataset which results into saving a lot of computation power as well as processing time. Lastly, we have the dense layer which will act as a regular layer of neurons within the network.

Model Fitting:

```
In [19]: steps_per_epoch = int(len(y_train) / batch_size)
         max_epochs = 500
         history = model.fit_generator(generator = train_generator, steps_per_epoch = steps_per_epoch,
                                       validation_data = val_generator, validation_steps = 50, epochs = max_epochs,
                                       verbose = 1, callbacks = [checkpoint_callback])
```

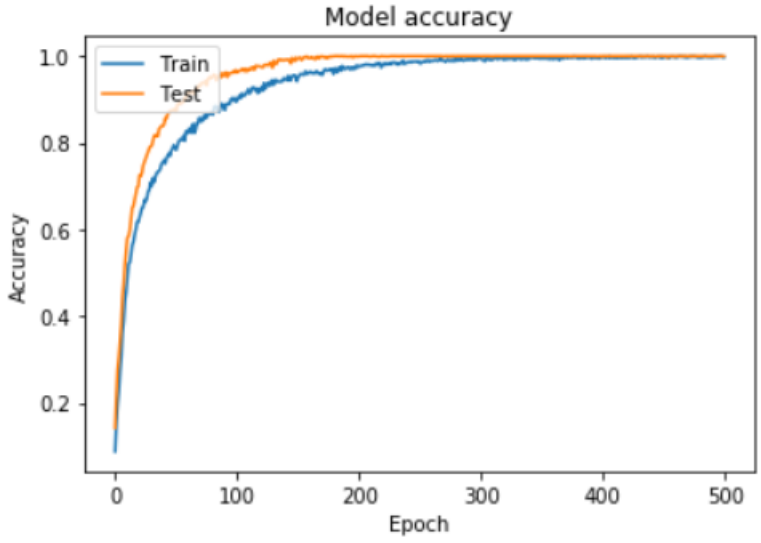
Model Output:

After training for 500 epochs, model provides the output of:

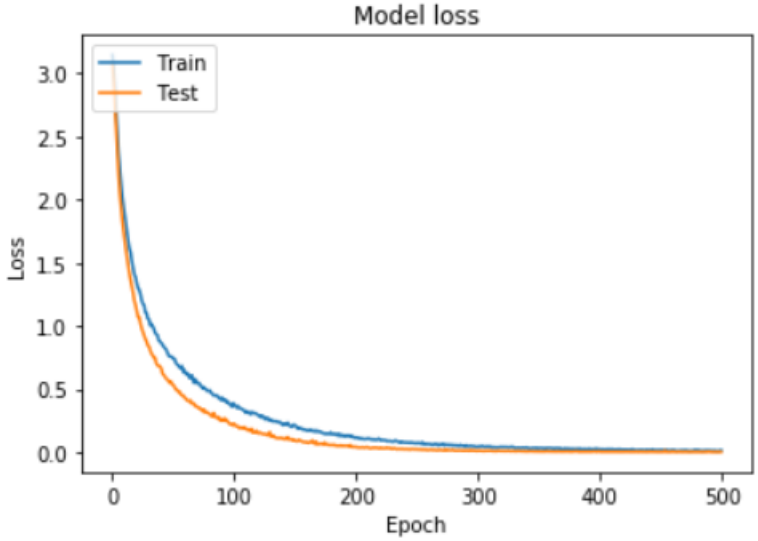
```
Epoch 499/500
85/85 [==============================] - 1s 15ms/step - loss: 0.0143 - accuracy: 0.9982 - val_loss: 0.0017 -
0000
Epoch 500/500
85/85 [==============================] - 2s 21ms/step - loss: 0.0154 - accuracy: 0.9971 - val_loss: 0.0021 -
```

Train Accuracy: 99%

Test Accuracy: 100%



Loss: 0.0143

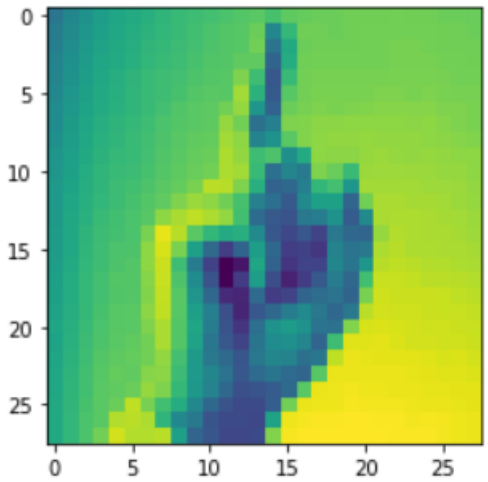


Prototype Predicted Output:

```
print('Predicted alphabet: ', final_label)
plt.imshow(x_train[0].reshape(28, 28))
```

```
Predicted alphabet:  d

<matplotlib.image.AxesImage at 0x26ad060d2b0>
```



Pros:

- When compared to CNN and VGG-16, Le-Net architecture performs very well.
- Good time complexity
- Works really well with images and real-time computer vision
- Developed by researchers at google, hence can be called dependable in terms of performance

Cons:

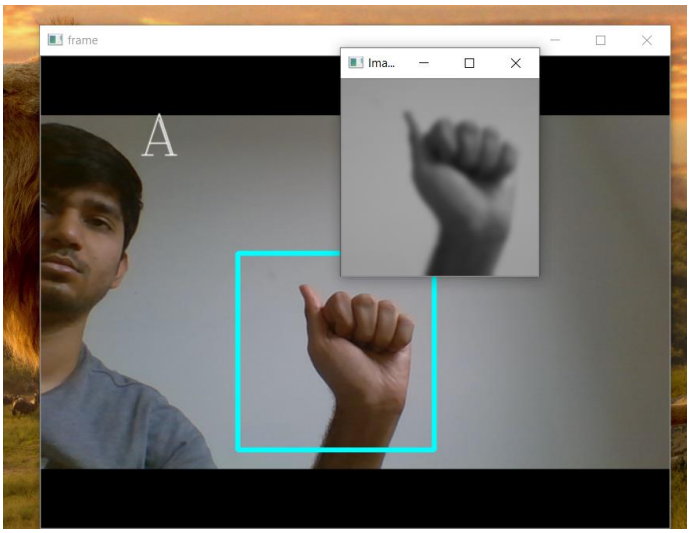
- Requires high number of epochs for accurate training and output
- If, the data is complex with more data points in correlation, the time complexity worsens
- LeNet architecture is not flexible, hence cannot be modified for accurate outputs.

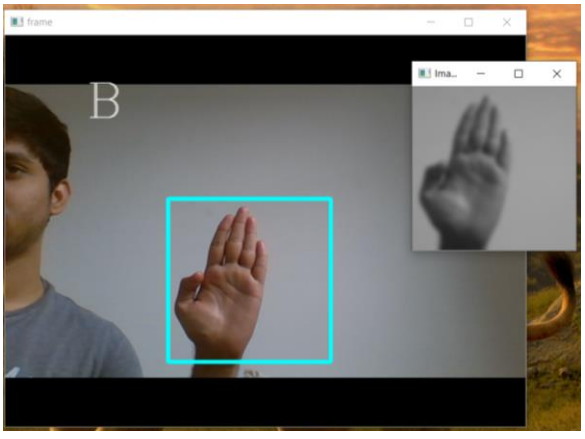
Evaluation Score Card:

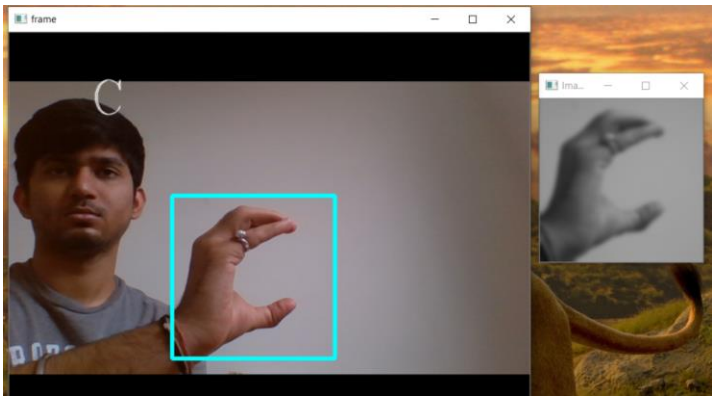
Evaluation Criteria	Convolutional Neural Network	LeNet-5
Training Accuracy	99.97%	99.85%
Test Accuracy	84.64%	100%
Loss	0.0012	0.0116
Epochs	50	500
Training Time Period	624 secs	531 secs

Real-time Implementation

After training the model and generating python environment results, our next milestone was to develop a front-end software to generate real-time results. As future potential clients would not be requiring python code to see, they would require a working software. Hence, we decided to go with a simple yml configuration to save processing power and worked out a small front-end software. Regardless of the flaws and limitations, it is just a steppingstone that we achieved and can be transformed into industrial level software for market release. Below are few outputs generated from our front end.







Even though the results may vary in real-time, it is mainly due to a few tweaks that the model might still require. The testing accuracy for the obtained model is 84.64%.

The working implementation can be made even better with time and curating the model and modifying the parameters. Regardless, the existing model performs consistently well, and a modified version can be of real help to the community

References:

1. https://www.azuredevopslabs.com/labs/vstsextend/aml/
2. https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac
3. http://deeplearning.net/tutorial/lenet.html
4. https://medium.com/@pechyonkin/key-deep-learning-architectures-lenet-5-6fc3c59e6f4