Developing a Classification Model for American Sign Language

Pratik Narendra Borse
Computer Science
Illinois Institute of Technology
pborse1@hawk.iit.edu

Ashlesh Khajbage
Data Science
Illinois Institute of Technology
akhajbage@hawk.iit.edu

Nikhil Singh Thakur
Artificial Intelligence
Illinois Institute of Technology
nthakur4@hawk.iit.edu

PROBLEM STATEMENT

The objective of this research project is to design and develop a highly accurate real-time classification model capable of recognizing and interpreting American Sign Language (ASL) gestures from video data. ASL, a visual language used by the deaf and hard-of-hearing community in the United States, can be better understood by a machine learning model, thereby improving communication accessibility for this community.

However, recognizing ASL signs from video data poses significant challenges, including variations in lighting, camera angles, and hand movements, which can cause variations in the appearance of signs. Furthermore, the context in which ASL signs are used can significantly impact their meaning, emphasizing the importance of considering the surrounding signs and context when classifying individual signs. Additionally, the complexities of ASL, including the use of facial expressions and body language in conveying meaning, further complicates the task of ASL sign recognition.

To address these challenges, a large and diverse dataset representing a range of signers, signing styles, and environments must be utilized to train the model effectively. The model should leverage deep learning techniques such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs) to capture both spatial and temporal features in the video data.

The success of this research project will be evaluated based on the accuracy and speed of the proposed model in recognizing ASL signs in real-time. Future applications of this technology could potentially be integrated into various virtual assistants, communication tools, and educational resources, providing a more inclusive and accessible experience for the deaf and hard-of-hearing community.

Despite the significant progress in ASL recognition in recent years, there still exists a considerable gap in the accuracy and robustness of the existing methods. Existing models are often limited by their inability to capture the complex and dynamic nature of ASL signs and the context in which they are used. As a result, the current state-of-the-art models may struggle in real-world applications, particularly in noisy environments or when signers exhibit variations in their signing styles.

This research project aims to address these limitations by developing a more sophisticated and robust model that can generalize well to different signers, signing styles, and environments. The proposed model will be designed to capture not only the spatial and temporal features of the video data but also the context and meaning of the signs within a broader linguistic context.

The potential impact of this research project extends beyond the realm of assistive technology for the deaf and hard-of-hearing community. The ability to recognize and interpret ASL signs from video data has implications for a wide range of fields, including human-computer interaction, robotics, and surveillance. Additionally, the insights gained from this research could potentially inform the development of similar recognition models for other sign languages, expanding the reach of this technology to other communities worldwide.

KEYWORDS

Classification Model, Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Image Processing, Spatial Features, Temporal Features.

DATASET

The ASL Alphabet dataset contains over 87,000 labeled images of hand gestures representing each letter of the American Sign Language (ASL) alphabet. The dataset is split into two parts: a training set containing 64,800 images and a test set containing 26,400 images. The images were captured using a Leap Motion Controller, which is a small, USB-powered device that uses two cameras and three infrared LEDs to track hand and finger movements in 3D space. The images are in the RGB format and have a resolution of 200x200 pixels. The images were captured from a diverse set of signers, including both genders, a range of ages, and different skin colors, to ensure a wide variety of signing styles and hand shapes were captured. Each image was labeled with the corresponding letter of the ASL alphabet using a hand-labeled ground truth, making it possible to train and test machine learning models on this dataset. This dataset is suitable for a range of computer vision tasks, including image classification, object recognition, and hand gesture recognition. The dataset is well-suited for training and evaluating deep learning models, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), which are widely used for image classification and sequence modeling tasks.

The ASL Alphabet dataset has numerous potential applications in real-world scenarios, such as virtual assistants, sign language translation tools, and assistive technology for the deaf and hard-of-hearing community. The dataset can also be used to better understand the complexities of ASL and inform the development of more advanced recognition models for sign language. The dataset consists of 29 classes, one for each letter of the American Sign Language alphabet, represented in the images as hand gestures. The images in the dataset were captured in a variety of lighting conditions, and the backgrounds include both plain and textured surfaces. In addition to the training and testing sets, the dataset also includes a validation set containing 8,100 images. This can be used to tune the hyperparameters of machine learning models or to perform early stopping during training to prevent overfitting. The dataset is balanced, with each class containing approximately the same number of images. This ensures that machine learning models are trained on an equal number of examples from each class, which can help prevent the model from being biased towards one particular class.
The dataset was collected and labeled by a team of researchers from the National Institute of Standards and Technology (NIST) and the American Sign Language Linguistic Research Project (ASLLRP). The team used a standardized protocol for data collection and labeling, which ensures consistency across different signers and labeling styles.

Overall, the ASL Alphabet dataset is a valuable resource for researchers and practitioners working in the field of computer vision and machine learning. Its high quality, diversity, and balance make it well-suited for developing and evaluating algorithms for image classification, object recognition, and hand gesture recognition tasks, particularly those related to American Sign Language.

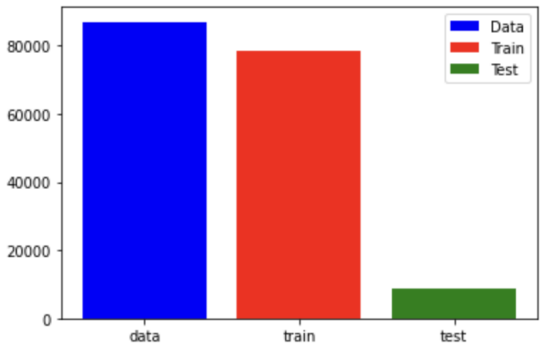


Fig.1 Distribution of Dataset into Training and Testing sets.

PREVIOUS WORK & HISTORY

During the 1990s, researchers commenced investigating the potential of computer vision and pattern recognition methodologies to recognize sign language gestures. A system developed in 1995 by Starner et al. employed a combination of neural networks and hidden Markov models to identify isolated sign language gestures. In the 1990s, the primary research focus was on identifying isolated sign language gestures rather than continuous sentences.

During the late 1990s and early 2000s, researchers began to explore statistical models such as hidden Markov models (HMMs) for recognizing continuous sign language sentences. Garg et al. developed the Sign Language Recognition and Translation (SLRT) system in 2001, which utilized HMMs to recognize Indian Sign Language sentences. Towards the early 2000s, researchers shifted their focus to developing systems that could recognize uninterrupted sign language sentences. A system created in 2004 by Vogler et al. utilized a blend of hidden Markov models and dynamic time warping to identify British Sign Language sentences.

In the mid-2000s, researchers explored the use of depth sensors, such as the Microsoft Kinect, to enhance sign language recognition. Depth sensors allowed for better tracking of hand and body movements, thus improving accuracy. In 2012, Microsoft researchers developed the Kinect Sign Language Translator as an example of such a system.

Recently, deep learning techniques, like recurrent neural networks and convolutional neural networks, have emerged as popular methods for sign language recognition. These techniques have demonstrated exceptional performance on various sign language datasets, such as the ASL-RGB-D and RWTH-BOSTON-104 datasets. In conclusion, machine learning has made significant progress in sign language recognition over the past few decades, with researchers employing diverse techniques and models for this task. Advancements in sensor technology, computing power, and the availability of large-scale sign language datasets have driven this progress.

COMPLETED WORK

As was mentioned in the Project plan section of our Project Proposal, we have completed a series of tasks so far that follow the timeline of the entire project as a whole.

1. Gather and review existing research on ASL recognition using machine learning techniques. - The existing research on ASL recognition using machine learning techniques covers a broad range of topics, including image and video processing, feature extraction, and classification algorithms. Some of the recent approaches in this area have employed deep learning techniques, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), to improve the accuracy of ASL recognition. Several datasets have also been developed for ASL recognition, including the ASL Alphabet dataset that contains images of hand gestures representing each letter of the ASL alphabet. Other

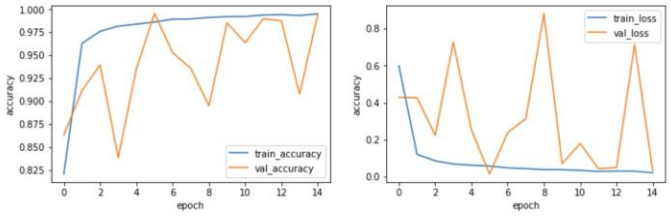
datasets include video recordings of signers performing phrases or sentences in ASL, which can be used to study the temporal dynamics of sign language.

2. Collect a diverse dataset of ASL gestures and pre-process the data. – As mentioned previously, we have selected a dataset that is diverse in nature and has a considerable amount of training and testing images in order to build a classification model.

3. Executing a baseline model and computing the accuracy and validation loss for each iteration – During the development of a classification model for American Sign Language (ASL), it is important to analyze the training and validation accuracy and loss for each epoch of training. The training accuracy measures the percentage of correctly classified training examples in each epoch, while the validation accuracy measures the percentage of correctly classified examples in the validation set. By analyzing the training and validation accuracy for each epoch, it is possible to track the progress of the model over time and identify any overfitting or underfitting issues. Similarly, the training loss measures the error rate in the training set, while the validation loss measures the error rate in the validation set. By analyzing the training and validation loss for each epoch, it is possible to determine whether the model is converging or diverging and adjust the model's architecture or hyperparameters as needed.

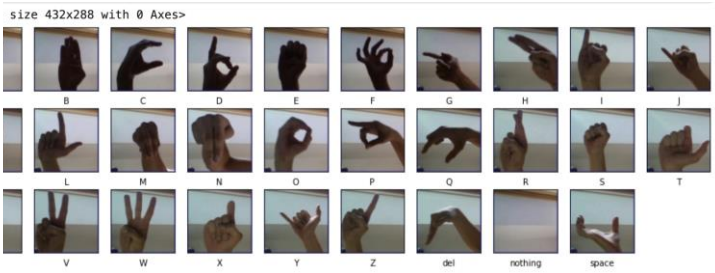
```
Epoch 1/15
1958/1958 [==============================] - 18s 5ms/step - loss: 0.5979 - accuracy: 0.8208 - val_loss: 0.4278 - val_accur
Epoch 2/15
1958/1958 [==============================] - 10s 5ms/step - loss: 0.1206 - accuracy: 0.9629 - val_loss: 0.4262 - val_accur
Epoch 3/15
1958/1958 [==============================] - 11s 5ms/step - loss: 0.0845 - accuracy: 0.9761 - val_loss: 0.2227 - val_accur
Epoch 4/15
1958/1958 [==============================] - 10s 5ms/step - loss: 0.0674 - accuracy: 0.9817 - val_loss: 0.7261 - val_accur
Epoch 5/15
1958/1958 [==============================] - 10s 5ms/step - loss: 0.0612 - accuracy: 0.9840 - val_loss: 0.2522 - val_accur
Epoch 6/15
1958/1958 [==============================] - 10s 5ms/step - loss: 0.0571 - accuracy: 0.9862 - val_loss: 0.0140 - val_accur
Epoch 7/15
1958/1958 [==============================] - 10s 5ms/step - loss: 0.0467 - accuracy: 0.9894 - val_loss: 0.2372 - val_accur
Epoch 8/15
1958/1958 [==============================] - 10s 5ms/step - loss: 0.0426 - accuracy: 0.9896 - val_loss: 0.3131 - val_accur
Epoch 9/15
1958/1958 [==============================] - 10s 5ms/step - loss: 0.0375 - accuracy: 0.9912 - val_loss: 0.8819 - val_accur
Epoch 10/15
1958/1958 [==============================] - 10s 5ms/step - loss: 0.0372 - accuracy: 0.9920 - val_loss: 0.0690 - val_accur
Epoch 11/15
1958/1958 [==============================] - 10s 5ms/step - loss: 0.0335 - accuracy: 0.9923 - val_loss: 0.1787 - val_accur
Epoch 12/15
1958/1958 [==============================] - 10s 5ms/step - loss: 0.0276 - accuracy: 0.9938 - val_loss: 0.0425 - val_accur
Epoch 13/15
1958/1958 [==============================] - 10s 5ms/step - loss: 0.0295 - accuracy: 0.9942 - val_loss: 0.0485 - val_accur
Epoch 14/15
1958/1958 [==============================] - 10s 5ms/step - loss: 0.0286 - accuracy: 0.9934 - val_loss: 0.7126 - val_accur
Epoch 15/15
1958/1958 [==============================] - 10s 5ms/step - loss: 0.0202 - accuracy: 0.9951 - val_loss: 0.0318 - val_accur
```

4. Ideally, the model should achieve high accuracy and low loss on both the training and validation sets, indicating that it is able to generalize well to new examples. However, if the training accuracy and loss are much better than the validation accuracy and loss, it may indicate overfitting, where the model is fitting too closely to the training data and not able to generalize well to new examples. In this case, regularization techniques, such as dropout or weight decay, may be used to prevent overfitting. Overall, analyzing the training and validation accuracy and loss for each epoch is a crucial step in the development of a classification model for ASL, as it allows for the identification of potential issues and optimization of the model's performance.



The above image shows the training accuracy, validation accuracy, training loss and validation loss of the classification model that we implemented.

5. In order to ensure that the model was working correctly, we used the below testing dataset.



6. Below is the model summary that we obtained. –

```
[13]: model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 64)        1792
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 64)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 16, 16, 64)        256
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 128)       73856
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 128)         0
_________________________________________________________________
batch_normalization_1 (Batch (None, 8, 8, 128)         512
_________________________________________________________________
dropout (Dropout)            (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 256)         295168
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 256)         0
_________________________________________________________________
batch_normalization_2 (Batch (None, 4, 4, 256)         1024
_________________________________________________________________
flatten (Flatten)            (None, 4096)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense (Dense)                (None, 1024)              4195328
_________________________________________________________________
dense_1 (Dense)              (None, 29)                29725
=================================================================
Total params: 4,597,661
Trainable params: 4,596,765
```

7. Develop a deep learning model such as Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs). –

```
classes = 29
batch = 32
epochs = 15
learning_rate = 0.001
```

```
model = Sequential()

model.add(Conv2D(64, (3, 3), padding='same', input_shape=(32, 32, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())

model.add(Conv2D(128, (3, 3), padding='same', input_shape=(32, 32, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Conv2D(256, (3, 3), padding='same', input_shape=(32, 32, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())

model.add(Flatten())
model.add(Dropout(0.2))
model.add(Dense(1024, activation='relu'))
model.add(Dense(classes, activation='softmax'))
```

The above code defines a Convolutional Neural Network (CNN) in Keras, a high-level neural networks API, for image classification.

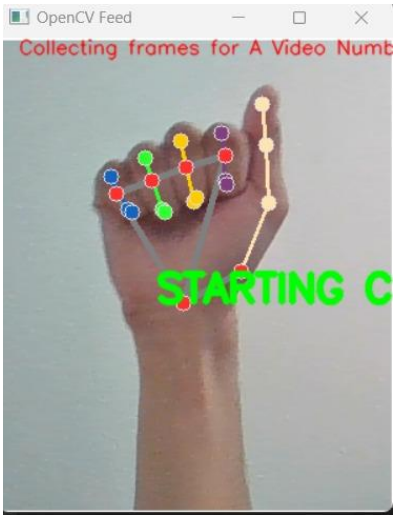
The model starts with a sequence of convolutional layers, where each layer has 64, 128, and 256 filters respectively. The filter size is (3,3) and each convolutional layer is followed by a max-pooling layer with a pool size of (2,2). The 'padding' argument is set to 'same', which means that the size of the output feature maps will be the same as the input size.

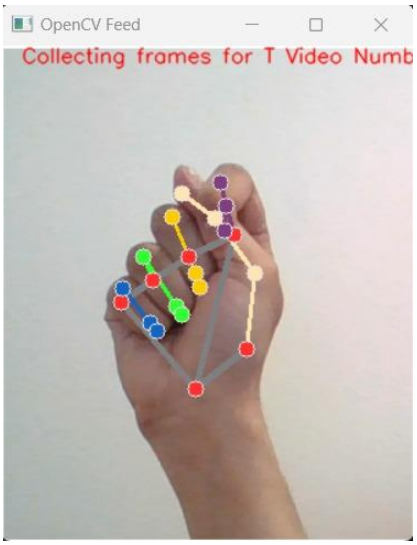
Batch normalization is applied after each convolutional layer to improve the training of the model. Dropout regularization is applied after the second and last convolutional layers to prevent overfitting.

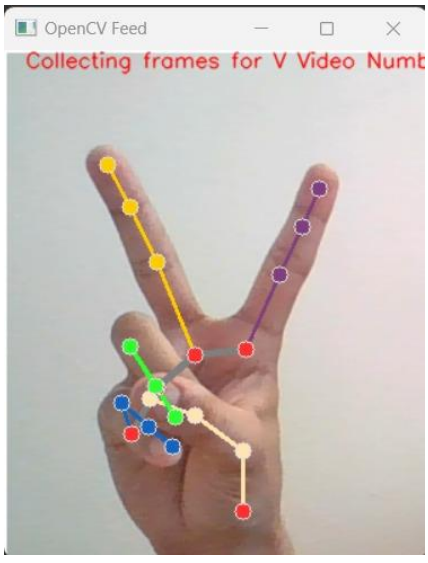
The output from the last convolutional layer is then flattened and fed into two fully connected dense layers with 1024 and 'classes' number of neurons, respectively. The 'softmax' activation function is used in the output layer to normalize the output probabilities over the 'classes'.

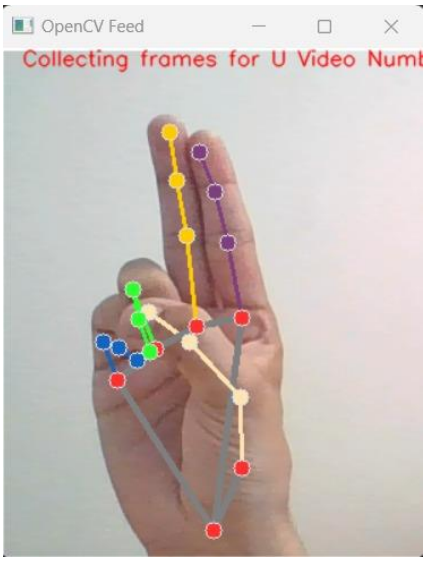
8. Train and evaluate the model on the collected dataset.

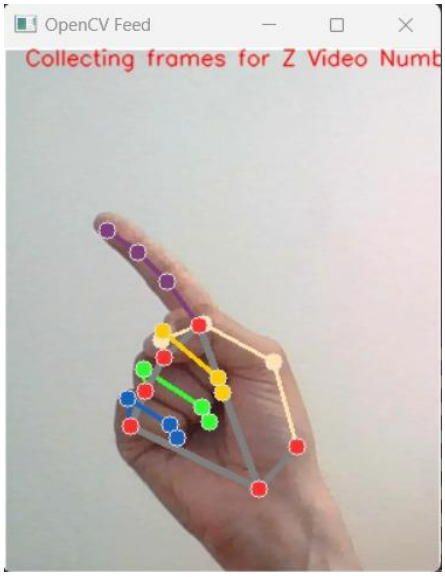
   We have trained the CNN model using a variety of input images. A few examples of the training images are shown below –











As seen from these images, the model tries to set a baseline by marking and locating the pointers for each section of the hand. In addition, the sign is trained by calculating the position of each pointer with its corresponding set of pointers of the same class or group.

9. After training the CNN Model, we used OpenCV to accept video input through the device's camera. A specific area in the camera's field of vision was pre-selected to take the input of the hand gestures.
   Once the model recognized the object as a hand, it would try to predict the gesture that the hand makes.
   As an output, the alphabet along with the accuracy of the prediction would be displayed. Below are a few images of the hand gesture recognition carried out by the CNN model. –

As seen in these images, the hand gestures are being recognized through the OpenCV feed and the alphabet that the gesture depicts is displayed along with its accuracy.

In addition to the CNN model, we also decided to implement a few other models and compare their accuracies with the existing CNN model. The first model that we trained was the KNN model.

Model 1: K Neighbor Classifier

Initializing The model with 6 neighbors

```
knnClss=KNeighborsClassifier(n_neighbors=6)
knnClss.fit(x_train,y_train)
```

```
KNeighborsClassifier(n_neighbors=6)
```

Predicting for the test data:

```
predicted=[]
for line in x_test:

  predicted.append(knnClss.predict([line]))
```

We implemented the KNN model and acquired the below accuracy results –

Predicting for the test data:

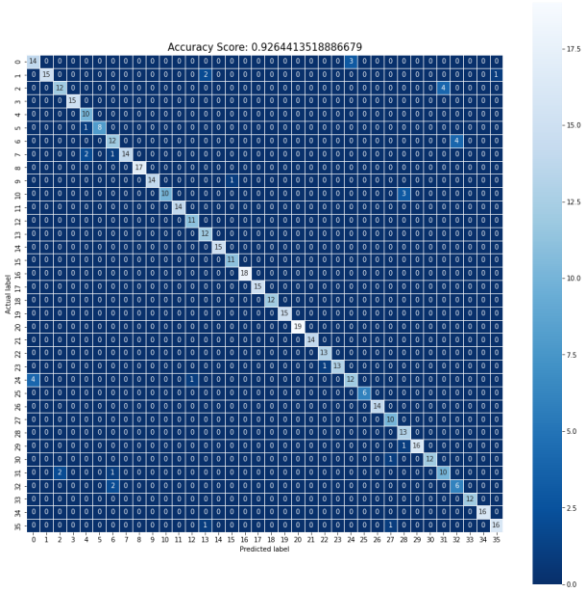
```
predicted=[]
for line in x_test:

  predicted.append(knnClss.predict([line]))
```

Accuracy of the model and F1 accuracy for each Class

```
print(f"Accuracy of the model: {accuracy_score(predicted,y_test)}")
print(f"F1 Score of the model:\n{f1_score(y_test,predicted,average=None)}")
```

```
ccuracy of the model: 0.9264413518886679
1 Score of the model:
0.8         0.90909091 0.8         1.         0.86956522 0.94117647
0.75        0.90322581 1.          0.96551724 0.86956522 1.
0.95652174 0.88888889 1.          0.95652174 1.         1.
1.          1.         1.          1.         0.96296296 0.96296296
0.75        1.         1.          0.90909091 0.86666667 0.96969697
0.96        0.74074074 0.66666667 1.         1.         0.91428571]
```

The Confusion Matrix depicts the accuracy of the model as below –



The next model that we trained was the Multinomial Logistic Regression Model.

Model 2: Multinomial Logistic Regression Model

Initialzing the model

```
LogReg = LogisticRegression(multi_class="multinomial",solver='lbfgs', max_iter=2000)
```

Training the Model

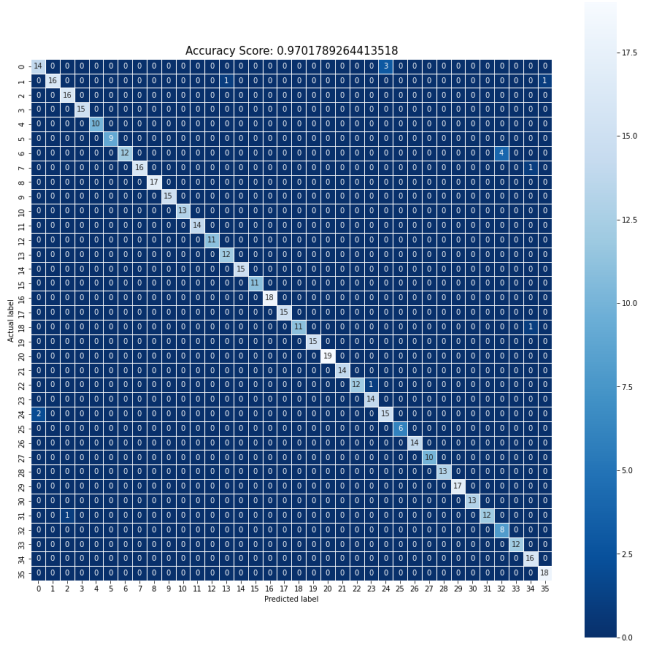
```
LogReg.fit(x_train,y_train)
```

```
LogisticRegression(max_iter=2000, multi_class='multinomial')
```

Predicting for the test cases

```
y_pred = LogReg.predict(x_test)
```

The following confusion matrix depicts the accuracy of the Multinomial Logistic Regression Model. –



Finally, we decided to train the Decision Tree Model with a max depth of 12.

Initializing the Model

```
DecTree=DecisionTreeClassifier(criterion='entropy', max_depth=12, min_samples_leaf=8, random_state=100)
```

Trainig The Model

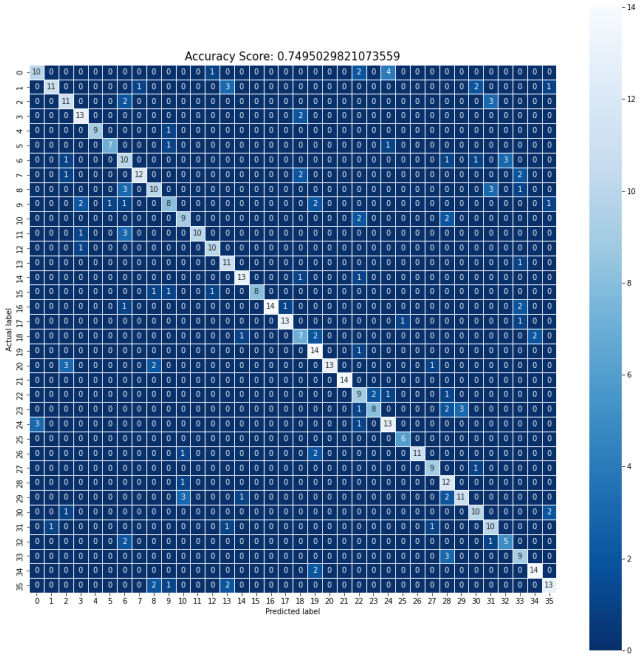
```
DecTree.fit(x_train,y_train)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=12, min_samples_leaf=8,
                       random_state=100)
```

Predicting for Test Cases

```
ypred=DecTree.predict(x_test)
```

The following confusion matrix shows us the accuracy of the Decision Tree Model. –



CONCLUSION

To create an application that could accept live video input and recognize the hand gestures by classifying them to the alphabets they represent according to the American Sign Language, we tried various Machine Learning models to create a classification system. The different types of models that we implemented were the KNN model, Multinomial Logistic Regression model, the Decision Tree model, and a CNN model.

Below are the accuracy comparisons of the first three models –

Summary of the Models

```
print(f"The Accuracy of KNN is {accuracy_score(y_test,predicted)}")
print(f"The Accuracy of Logistic Regression is {accuracy_score(y_test,y_pred)}")
print(f"The Accuracy of Decision Tree is {accuracy_score(y_test,ypred)}")
```

```
The Accuracy of KNN is 0.9264413518886679
The Accuracy of Logistic Regression is 0.9701789264413518
The Accuracy of Decision Tree is 0.7495029821073559
```

As we can see, Logistic regression had an accuracy of around 97%, followed by KNN with around 92% and Decision tree with the least amount of accuracy at 75%.

On the other hand, CNN reported an accuracy of around 99% which can be considered as a good result. Based on these results, we could conclude that CNN would be the optimal model for us to train and test the ASL data and create a classification system.

Convolutional Neural Networks (CNNs) are engineered to acquire sophisticated features from raw input data, specifically images, by utilizing convolutional layers that utilize filter sets to obtain features such as edges, corners, and textures. Conversely, KNN, Logistic regression and Decision Tree models are more straightforward and do not possess the capacity to learn complex features from raw input data.

In terms of robustness, CNNs exhibit greater resilience to noise and input data variations, which is crucial for ASL gesture recognition as hand gestures can differ in size, orientation, and lighting. Conversely, KNN, Logistic regression and Decision Tree models may be more vulnerable to these fluctuations, resulting in suboptimal performance.

CNNs are readily scalable to accommodate larger datasets and complex tasks, such as multi-classification of ASL gestures. In contrast, KNN, Logistic regression and Decision Tree models may not be as scalable, impeding performance on larger datasets.

CNNs have been demonstrated to achieve superior performance on various image recognition tasks, including ASL gesture recognition, resulting in state-of-the-art accuracy. Conversely, KNN, Logistic regression and Decision Tree models may struggle to attain the same level of accuracy due to their constraints in terms of acquiring complex features, handling noise, and scalability to larger datasets.

In summary, CNNs are a superior choice for ASL gesture recognition due to their ability to learn complex features, robustness to noise, scalability, and better accuracy compared to simpler models such as KNN, Logistic regression and Decision Tree.

FUTURE WORK

Another important step in the project is to try and explore other approaches to creating a classification model such as RNNs and Graph Neural Networks. Once we are able to implement a model using each of these approaches, we can form an in-depth analysis of each approach and perform a comparative study of those networks.

Incorporating other modalities such as facial expressions and body movements into a classification model for ASL recognition can enhance the accuracy and robustness of the model. Facial expressions and body movements play a crucial role in conveying meaning in ASL and ignoring them can lead to misinterpretations of signs.

One approach for incorporating facial expressions and body movements into the model is to use a multimodal framework that combines information from different modalities, such as video and audio. For example, the model can be trained on video sequences that include both hand gestures and facial expressions, using techniques such as 3D convolutional neural networks (CNNs) to capture spatial and temporal features from the data.

Another approach is to use an attention mechanism that focuses on relevant regions of the input, such as the face and body, while ignoring irrelevant regions. This can be achieved by training the model with an auxiliary task, such as face recognition, that encourages the model to learn relevant features from the face.

Incorporating other modalities can also help in addressing the issue of variability in signing styles and contexts. For example, different signers may use different facial expressions or body movements to convey the same sign, and incorporating these modalities can help the model to generalize better across different signers and contexts.

Overall, incorporating other modalities such as facial expressions and body movements into the model can improve the accuracy and robustness of the ASL recognition model and provide a more comprehensive and inclusive representation of the language. In order to enhance the performance of ASL gesture recognition models, advanced deep learning techniques like RNNs and attention-based models can be utilized. These models can capture the sequential nature of sign language and extract more informative features from different parts of the input image.

To further improve accuracy, the dataset can be expanded to include more diverse samples from different signers, skin tones, hand sizes, shapes, and variations in lighting, camera angles, and backgrounds. Developing real-time recognition systems that can recognize signs in real-time is also a promising direction for future work. This can improve communication between deaf and hearing individuals by allowing more natural and spontaneous interactions. In addition to recognizing individual signs, future work could focus on recognizing entire sentences or conversations in sign language, which would require more advanced natural language processing techniques. Lastly, combining multiple input modes like audio and visual could potentially enhance the accuracy of ASL gesture recognition models by disambiguating between signs with similar hand shapes but different meanings.

Overall, there are several potential avenues for future work in developing ASL gesture recognition models, including advanced deep learning techniques, dataset expansion, real-time recognition systems, expanded recognition scope, and multi-modal recognition.

GITHUB REPOSITORY

The training dataset, testing dataset, code repository and results of the project can be found on the below GitHub repository –

https://github.com/Ashleshk/Classification-Model-for-ASL

REFERENCES

[1] Necati Cihan Camgoz, Oscar Koller, Simon Hadfield, Richard Bowden, 2020. *Sign Language Transformers: Joint End-to-end Sign Language Recognition and Translation*.
DOI: https://doi.org/10.48550/arXiv.2003.13830

[2] Jie Huang, Wengang Zhou, Houqiang Li and Weiping Li, "Sign Language Recognition using 3D convolutional neural networks," 2015 IEEE International Conference on Multimedia and Expo (ICME), Turin, 2015, pp. 1-6, doi: 10.1109/ICME.2015.7177428.

[3] S. Ikram and N. Dhanda, "American Sign Language Recognition using Convolutional Neural Network," 2021 IEEE 4th International Conference on Computing, Power and Communication Technologies (GUCON), Kuala Lumpur, Malaysia, 2021, pp. 1-12, doi: 10.1109/GUCON50781.2021.9573782.

[4] de Amorim, Cleison Correia, David Macêdo, and Cleber Zanchettin. "Spatial-temporal graph convolutional networks for sign language recognition." Artificial Neural Networks and Machine Learning–ICANN 2019: Workshop and Special Sessions: 28th International Conference on Artificial Neural Networks, Munich, Germany, September 17–19, 2019, Proceedings 28. Springer International Publishing, 2019.

[5] Huang, J., Zhou, W., Zhang, Q., Li, H., & Li, W. (2018). Video-based Sign Language Recognition without Temporal Segmentation. AAAI Conference on Artificial Intelligence.