Real-time Sign Language Detection

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

Sanjana Rayarala
Artificial Intelligence
Illinois Institute of Technology
srayarala@hawk.iit.edu

PROBLEM STATEMENT

Our main aim in this research is to create a smart computer program that can quickly and accurately understand American Sign Language (ASL) when people use it in real life. ASL is a vital language for the Deaf and Hard-of-Hearing community in the U.S., and we believe that technology can make it even more accessible. Imagine a computer that can 'read' sign language on a live video and understand what's being said. This could be a game-changer for the Deaf and Hard-of-Hearing community, making communication easier and more inclusive.

Lot of difficulties could be posed in the process of identifying ASL signs from video footage, such as camera angles, changes in lighting, and hand gestures that can affect how the signs look. Moreover, the meaning of ASL signs can be greatly influenced by the context in which they are used, underscoring the significance of taking and the surrounding signs and background into account when categorizing individual signs. ASL sign recognition is made more complex when the use of body language and facial expressions is taken into consideration. Tackling these challenges, the model to be developed requires a large, diverse dataset that includes data of wide variety styles of signs, by many people.

Mitigating these challenges necessitates the utilization of an extensive and diversified dataset, encapsulating a spectrum of signers, signing styles, and environmental conditions for effective model training. The model should leverage state-of-the-art deep learning techniques, mainly there are convolutional neural networks (CNNs) and recurrent neural networks (RNNs), to capture both spatial and temporal attributes within the video data.

The success of this research project hinges on the model's capacity to achieve high accuracy and rapid recognition of ASL signs in real-time. Prospective applications of this technology could conceivably be integrated into a myriad of virtual assistants, communication tools, and educational resources, ushering in a more inclusive and accessible era for the DHH community.

Notwithstanding the substantial strides made in ASL recognition in recent times, significant shortcomings persist in the precision and resilience of existing methodologies. Present models often falter in the face of ASL's intricacies, particularly in real-world settings characterized by ambient noise or idiosyncrasies in signers' styles.

This research initiative aspires to surmount these limitations by crafting a more sophisticated and robust model, characterized by its adaptability across different signers, signing styles, and environmental conditions. The proposed model will not only harness the spatial and temporal nuances of the video data but also discern the underlying context and meaning of signs within a broader linguistic context.

The potential ramifications of this research project transcend the realm of assistive technology for the DHH community. The capacity to interpret ASL signs from video data holds significance across multiple domains, including human-computer interaction, robotics, and surveillance. Furthermore, the insights gleaned from this research endeavor may provide the blueprint for analogous recognition models for other sign languages, thus extending the reach of this technology to Deaf communities worldwide.

KEYWORDS

Real-Time Recognition, Transfer Learning, Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Classification model, Image Processing.

DATA SET

The Sign Language MNIST dataset serves as a drop-in replacement for MNIST, featuring a CSV format that includes labels and pixel values in single rows. It is tailored to represent the American Sign Language (ASL) alphabet, excluding letters J and Z due to their reliance on motion. With a total of 27,455 training cases and 7,172 test cases, this dataset maps each case to a label ranging from 0 to 25, corresponding to the alphabetic letters A to Z. Each case includes a 28x28 pixel grayscale image with values spanning from 0 to 255.

The creation of this dataset entailed a meticulous process to ensure its richness and diversity. The original data consisted of hand gesture images with multiple users performing gestures against various backgrounds. To augment the dataset's size and diversity, a comprehensive image processing pipeline was implemented. The process began with precise cropping, focusing on isolating the hand region to provide a close-up perspective of the gestural element.

Next, the images were converted to grayscale, ensuring uniformity and allowing algorithms to concentrate on shape and form rather than color. Subsequently, the images were resized to a standard 28x28 pixel resolution, aligning with the MNIST convention and ensuring compatibility with various machine learning methods, including Convolutional Neural Networks (CNNs).

A substantial portion of dataset expansion was achieved by creating over 50 variations of each image. These variations were introduced systematically to enrich the dataset, simulating real-world challenges. This included applying various image filters

such as 'Mitchell,' 'Robidoux,' 'Catrom,' 'Spline,' and 'Hermite' to create distinctive visual effects. To emulate image distortion commonly seen in different lighting conditions or varying camera qualities, 5% random pixelation was introduced. Additionally, brightness and contrast adjustments within a ±15% range were made to introduce variations in image exposure. To account for different hand orientations during signing, rotations of up to 3 degrees were applied.

This deliberate dataset expansion strategy enhanced data diversity and complexity, providing real-world challenges for machine learning models. The Sign Language MNIST dataset offers a more challenging and practical counterpart to MNIST, suitable for a wide range of applications.
Inspiration:
The Sign Language MNIST dataset takes inspiration from the Fashion-MNIST and the machine learning pipeline for gestures by Sreehari.

Overall, developing a robust visual recognition algorithm using this dataset holds great promise. It not only challenges modern Deep learning techniques, including Convolutional Neural Networks but also has the potential to significantly improve computer vision applications for the deaf and hard-of-hearing community. American Sign Language (ASL) is a tough and vital language, serving as the primary means of communication for many deaf North Americans. ASL ranks as the fifth most widely used language in the U.S., following Spanish, Italian, German, and French. By implementing computer vision on affordable hardware like the Raspberry Pi with OpenCV and Text-to-Speech, we can enable improved and automated translation applications.

COMPLETED WORK

The completion of this project has been marked by a series of substantial accomplishments in line with our project proposal's timeline. Here is an overview of the key milestones achieved thus far:

1. Reviewing Existing Research

Our project embarked with a comprehensive review of existing research on American Sign Language (ASL) recognition, where deep learning techniques played a pivotal role. We delved into a diverse array of topics, encompassing image and video Analyzing, extracting features, and employing classification algorithms. Recent strides in ASL recognition have been notably underpinned by deep learning methodologies, particularly leveraging the capabilities of convolutional neural networks (CNNs) and recurrent neural networks (RNNs). These advanced techniques have shown remarkable potential in elevating the accuracy and precision of ASL recognition systems.

Our exploration extended to the study of diverse datasets developed for ASL recognition. Among these, the ASL Alphabet dataset stands out, as it houses images of hand gestures, each representing a letter of the ASL alphabet. Furthermore, we delved into datasets featuring video recordings, showcasing signers performing phrases or sentences in ASL. These datasets offered profound insights into the intricate temporal dynamics of sign language, shedding light on the efficacy of deep learning techniques in capturing the subtleties and nuances of ASL.

2. Dataset Collection and Pre-processing

The foundation of our work was laid by selecting a diverse dataset comprising a substantial number of training and testing images. This dataset was meticulously curated to facilitate the development of our classification model. The diversity within the dataset aligns with our goal of building a robust and versatile model for ASL recognition.

3. Model Development and Analysis

A critical aspect of our project involved building an initial model to recognize American Sign Language (ASL) gestures. We diligently assessed the model's performance by rigorously tracking accuracy and validation loss for each training iteration. Achieving an impressive accuracy of 98.63%, we conducted our training across 10 epochs with a Total batch size of 128. This iterative analysis helped us monitor the model's progress and identify potential concerns, such as overfitting or underfitting. The ongoing refinement process was based on this detailed evaluation, ensuring the model's accuracy and readiness for practical applications in ASL recognition.

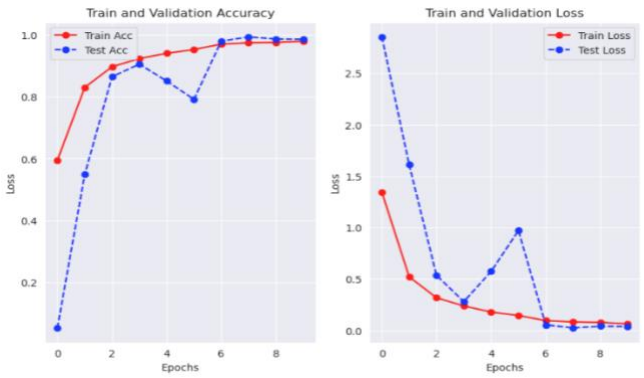
```
Epoch 1/10
204/204 [==============================] - 37s 173ms/step - loss: 1.3439 - accuracy: 0.5950 - val_loss: 2.8541 - val_accuracy: 0.0523 - lr: 0.0010
Epoch 2/10
204/204 [==============================] - 35s 170ms/step - loss: 0.5204 - accuracy: 0.8305 - val_loss: 1.6098 - val_accuracy: 0.5487 - lr: 0.0010
Epoch 3/10
204/204 [==============================] - 35s 172ms/step - loss: 0.3177 - accuracy: 0.8974 - val_loss: 0.5355 - val_accuracy: 0.8653 - lr: 0.0010
Epoch 4/10
204/204 [==============================] - 35s 172ms/step - loss: 0.2381 - accuracy: 0.9238 - val_loss: 0.2795 - val_accuracy: 0.9059 - lr: 0.0010
Epoch 5/10
204/204 [==============================] - 35s 171ms/step - loss: 0.1786 - accuracy: 0.9409 - val_loss: 0.5741 - val_accuracy: 0.8512 - lr: 0.0010
Epoch 6/10
204/204 [==============================] - ETA: 0s - loss: 0.1454 - accuracy: 0.9532
Epoch 6: ReduceLROnPlateau reducing learning rate to 0.000500000023748727.
204/204 [==============================] - 36s 174ms/step - loss: 0.1454 - accuracy: 0.9532 - val_loss: 0.9707 - val_accuracy: 0.7920 - lr: 0.0010
Epoch 7/10
204/204 [==============================] - 35s 172ms/step - loss: 0.0958 - accuracy: 0.9706 - val_loss: 0.0538 - val_accuracy: 0.9803 - lr: 5.0000e-04
Epoch 8/10
204/204 [==============================] - 35s 171ms/step - loss: 0.0828 - accuracy: 0.9750 - val_loss: 0.0256 - val_accuracy: 0.9941 - lr: 5.0000e-04
Epoch 9/10
204/204 [==============================] - 35s 172ms/step - loss: 0.0776 - accuracy: 0.9763 - val_loss: 0.0406 - val_accuracy: 0.9873 - lr: 5.0000e-04
Epoch 10/10
204/204 [==============================] - ETA: 0s - loss: 0.0649 - accuracy: 0.9799
Epoch 10: ReduceLROnPlateau reducing learning rate to 0.000250000011874363.
204/204 [==============================] - 35s 171ms/step - loss: 0.0649 - accuracy: 0.9799 - val_loss: 0.0390 - val_accuracy: 0.9863 - lr: 5.0000e-04
```

```
print("Accuracy of the model is - " , model.evaluate(x_test,y_test)[1]*100 , "%")
```

```
225/225 [==============================] - 3s 11ms/step - loss: 0.0390 - accuracy: 0.9863
Accuracy of the model is -  98.63357543945312 %
```

4. Model Evaluation

The primary objective was to achieve high accuracy and low loss for both the training and validation sets, signifying the model's capacity to generalize effectively to new examples. Discrepancies between training and validation metrics, particularly when training metrics significantly outperformed validation metrics, were indicative of potential overfitting. In response, we implemented regularization techniques, including dropout and weight decay, to prevent overfitting. The in-depth analysis of training and validation metrics played a pivotal role in model optimization. This iterative assessment not only aids in identifying potential issues but also provides insights for fine-tuning the model to optimize its performance and generalization capabilities.



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

5. Testing Dataset Validation

To ensure the model's correctness and robustness, we subjected it to rigorous testing using a dedicated dataset designed for validation.

6. Model Summary

Below, we present a concise yet comprehensive model summary that encapsulates the architectural and operational characteristics of the Real time sign language detection recognition model. This summary serves as a reference point for the model's structure and configuration.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 45)        450

 batch_normalization (BatchN  (None, 28, 28, 45)       180
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 14, 14, 45)       0
 )

 conv2d_1 (Conv2D)           (None, 14, 14, 55)        22330

 dropout (Dropout)           (None, 14, 14, 55)        0

 batch_normalization_1 (Batc  (None, 14, 14, 55)       220
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 7, 7, 55)         0
 2D)

 flatten (Flatten)           (None, 2695)              0

 dense (Dense)               (None, 24)                64704

=================================================================
Total params: 87,884
Trainable params: 87,684
Non-trainable params: 200
```

7. Developing Deep learning Model

CNN- Convolutional Neural Network, a deep learning model, is developed with hyperparameters consisting of 29 classes that represents 29 different signs, a batch size of 32 where these 32 samples from the dataset are used to calculate the error and update the model parameters during one iteration of the training process, 15 epochs such that entire dataset will be passed through the neural network 15 times and learning rate of 0.001 which is a moderate rate that balances the speed of learning with the risk of overshooting the minimum of the loss function.

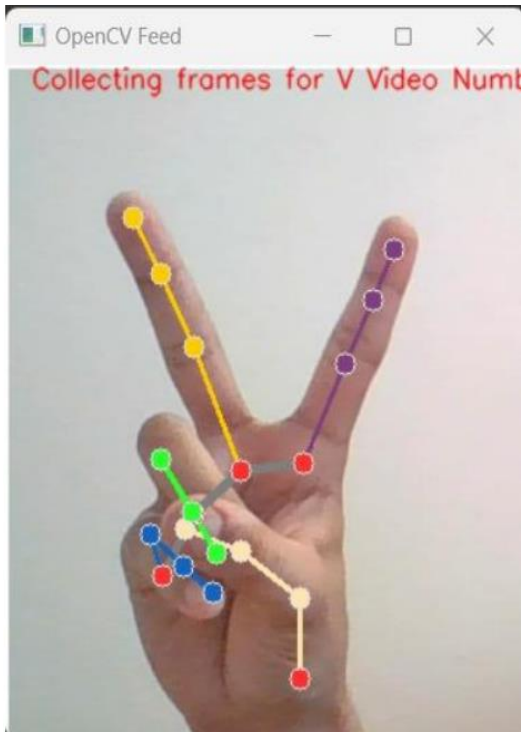
```
classes = 29
batch = 32
epochs = 15
learning_rate = 0.001
```

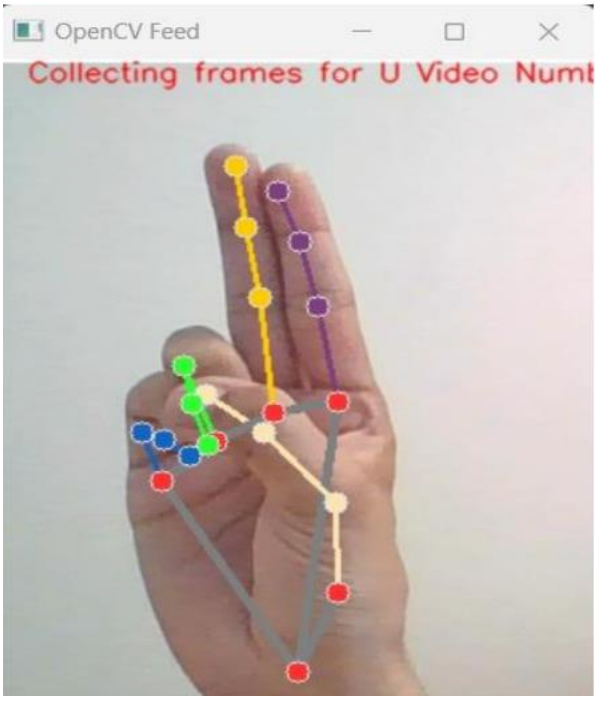
The architecture defined consists of sequential model with convolutional layers with 64, 128, 256 filters of size 3x3, using ReLU activation and 'same' padding. The increasing number of filters in convolutional layers allows the network to extract more complex features at each level. Followed by pooling layers, with 2x2 window to reduce spatial dimensions. Batch normalization after these layers to stabilize and speed up the training. A dropout layer with a rate of 0.2 to prevent overfitting by randomly setting input units to 0 during training. A layer to flatten the 3D output of the previous layers to 1D, making it suitable for input into the dense layers. Fully connected layers, with the first having 1024 neurons and ReLU activation, and the last having a number of neurons equal to the number of classes with softmax activation for multi-class classification to output probabilities for each of the classes.

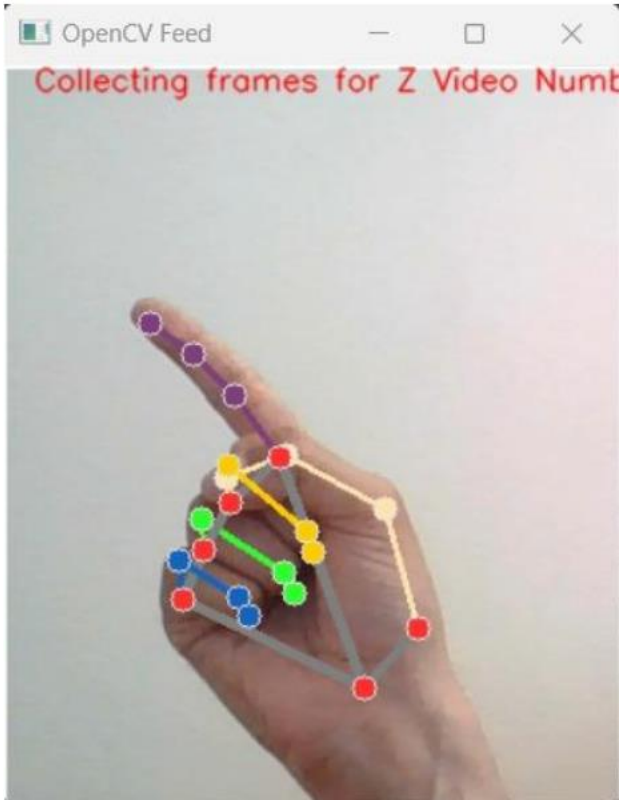
```
model = Sequential()

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'))
```

8. Train and test the model on the collected dataset

We trained CNN model using diverse set of input images. Following are some of the training images:







The images depict a system that establishes a reference framework by identifying and pinpointing specific reference points on each segment of the hand. Moreover, the system is trained to recognize signs by determining the location of each reference point and correlating it with other points within the same class.

9. Real-time video input feed

Using OpenCV to accept video feed through the camera where a particular region within the camera's visual range was predetermined to capture the hand gesture inputs. The trained CNN model is now used to identify the object as hand and predict the gesture made by the hand. Thereafter, the alphabet along with accuracy of the prediction are shown as the output. Few images of the sign detection done by CNN model are shown below:

The images illustrate that hand gestures are identified using the OpenCV stream, and the corresponding alphabet signified by the gesture is shown, along with the accuracy of the prediction.

We also implemented traditional algorithms like KNN, Logistic Regression and Decision Tree; that requires manual feature extraction unlike the CNN model that has been implemented and draw a comparison between these models.

MODEL 1: K NEIGHBOR CLASSIFIER

K-Nearest Neighbors (KNN) is a simple, non-parametric, and lazy learning algorithm that can be used for classification and regression. KNN can be chosen for sign language detection because it is straightforward to understand and implement as it doesn't require any complex mathematical assumptions, it exhibits Non-Parametric nature as it makes no underlying assumptions about the distribution of the data and it's ability to work well with small datasets.

Implementation of KNN:

The classifier is initialized with 6 neighbors which indicates that the classifier should consider the 6 closest neighbors to determine the class of a given sample, then KNN classifier is trained using the training data. x_train contains the feature vectors of the training samples, and y_train contains the corresponding labels or classes. The fit method adjusts the model parameters to the data, so after calling this method, the classifier is ready to make predictions on new, unseen data.

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

Now the trained classifier is used to make predictions on test data.

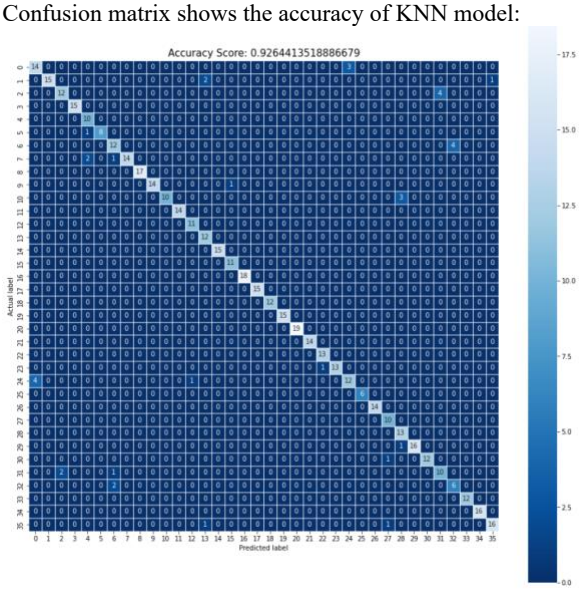
```
predicted=[]
for line in x_test:

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

So we got the accuracy by:

```
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]
```

Confusion matrix shows the accuracy of KNN model:



MODEL 2: MULTINOMIAL LOGISTIC REGRESSION

Multinomial logistic regression is a classification method that generalizes logistic regression to multiclass problems, i.e., with more than two possible discrete outcomes. It is used for sign detection and other classification tasks where the outcomes are discrete for several reasons like recognizing multiple signs, where each sign is a separate class; decision boundaries between classes can be approximated by a linear function; can be more computationally efficient than some other algorithms when dealing with large datasets; provide insights into which features are most important for sign detection; can be regularized to avoid overfitting.

Implementation of Logistic Regression:

Logistic Regression classifier is initialized with parameters set to handle multiple classes rather than just binary classification. The 'lbfgs' solver is an optimization algorithm that performs well on

moderately sized datasets for multiclass problems. Finally, 'max_iter=2000' sets the maximum number of iterations for the solver to converge, suggesting an expectation of a relatively complex optimization process or a larger dataset. This configuration is chosen to effectively model a situation where the decision boundaries between classes are linear and can be distinguished using logistic regression.

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

The model is then trained using x_train and y_train; which are the feature set of the training data and corresponding labels for the training data.

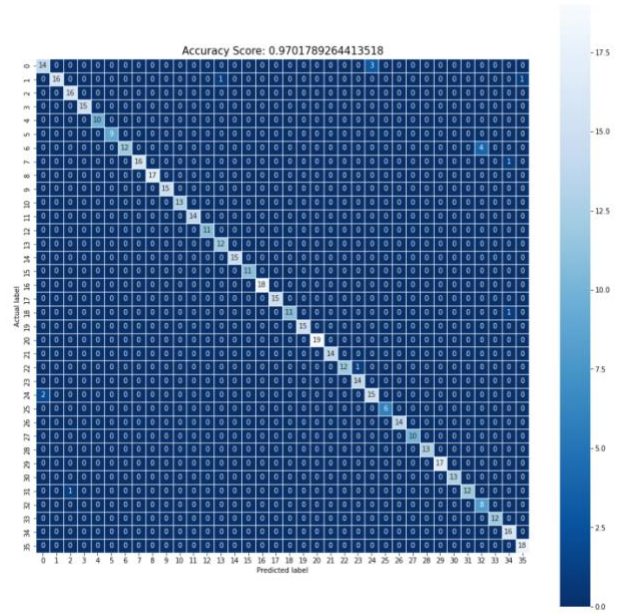
```
LogReg.fit(x_train,y_train)
```

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

The model is currently employed to generate predictions on the test dataset.

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

The confusion matrix represents performance accuracy of the Multinomial Logistic Regression Model.



MODEL 3: DECISION TREE CLASSIFIER(DTC)

The Decision Tree Classifier stands out as a widely used algorithm in the realm of machine learning that can be used for sign detection as Decision trees are easy to understand and interpret and as it allows for understanding which features are important for the classification and how the decisions are being made, it can handle non-linear relationships between features effectively. Decision trees provide a clear indication of feature importance which can help in understanding which parts of the hand gestures are more important for detecting the sign and they do not require feature scaling to be performed as they do not compute distances between features.

Implementing Decision Tree Classifier

Decision Tree Classifier is defined that uses entropy to decide on splitting nodes, is limited to a depth of 12, requires at least 8 samples to form a leaf, and has a set randomness state for reproducibility. These settings are aimed at reducing overfitting and making the model's performance more robust.

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

The model is then trained.

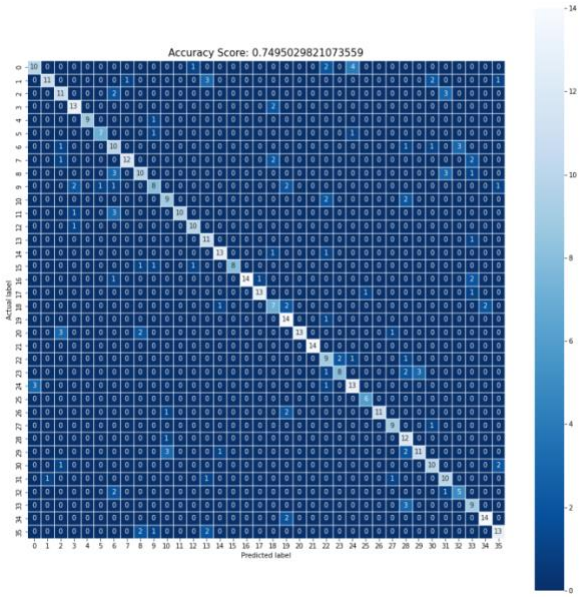
```
DecTree.fit(x_train,y_train)
```

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

Apply the trained model to make anticipations on the test set.

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

The confusion matrix below depicts the accuracy of the Decision Tree model



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
```

CONCLUSION

We used multiple machine learning models like KNN, Logistic Regression, Decision Tree, CNN to classify the hand gestures from the live video input feed into alphabets they represent as per the American Sign Language.
We observed that the traditional algorithms KNN, Logistic Regression and Decision tree gave 92%, 97%, and 75% accuracies. Where as, CNN gave accuracy of 98% which is higher than the other models which have been implemented, hence it can be regarded as the optimal model to build a system to classify ASL hand-gestures.

Convolutional Neural Networks (CNNs) are crafted to capture complex attributes from unprocessed input, particularly images, through the use of convolutional layers that apply various filters to discern features like edges, corners, and textures. In contrast, models such as KNN, Logistic Regression, and Decision Trees are simpler and lack the ability to autonomously identify intricate features from raw inputs.

CNNs stand out for their exceptional ability to handle complex tasks, such as ASL gesture recognition, with state-of-the-art precision. This capability stems from their sophisticated feature learning, resilience to noise, and adaptability to scale with larger datasets. In contrast, simpler models like KNN, Logistic Regression, and Decision Trees may not achieve the same high degrees of precision. These models often have limitations in extracting complex features and may underperform when faced with data variability and larger dataset scales.

The robustness of CNNs against variations in the input data, including differences in size, orientation, and lighting, makes them particularly well-suited for recognizing ASL gestures. Where models like KNN, Logistic Regression, and Decision Trees may falter in the face of such fluctuations, CNNs maintain their performance, which is critical for accurately interpreting hand gestures.

Furthermore, the scalability of CNNs is a significant advantage. They can efficiently manage and learn from extensive datasets, a vital attribute for multi-class ASL gesture classification tasks. Simpler models, in comparison, may not scale as effectively, potentially hindering their performance as the dataset size increases.

Overall, for tasks involving ASL gesture recognition, CNNs are generally the preferred option, given their robust feature learning, resilience to input variations, and scalability, which collectively contribute to their superior accuracy over more basic models like KNN, Logistic Regression, and Decision Trees.

FUTURE WORK

To improve ASL gesture recognition models, it's vital to consider advanced deep learning techniques such as RNNs and attention-based models, which excel in grasping the sequential and nuanced characteristics of sign language. These methods can pinpoint more detailed features by concentrating on specific parts of the input image, such as hand shapes or facial expressions.

Incorporating additional modalities like facial expressions and body language can significantly bolster the model's accuracy and robustness. Since these elements are integral to ASL communication, their inclusion can prevent potential misinterpretations of signs. Utilizing multimodal frameworks that merge data from different sources, like video incorporating hand movements and facial expressions, can offer a richer, more nuanced understanding of ASL.

Exploring various approaches for classification, including RNNs and Graph Neural Networks, is another crucial step. Implementing different models allows for a thorough analysis and comparative study, enhancing our understanding of which techniques are most effective for ASL recognition.

Further expanding the dataset to encompass a broader range of signers, skin tones, hand sizes, and environmental variables will also aid in developing more generalizable systems. This diversity is key to addressing the variability in signing styles and contexts, ensuring the model remains accurate across various users and settings.

Developing real-time systems capable of recognizing ASL signs instantaneously can facilitate seamless communication between deaf and hearing individuals, supporting more fluid and natural interactions. Moreover, moving beyond isolated signs to recognize sentences and full conversations in ASL could revolutionize the field, although it would require sophisticated natural language processing capabilities.

Additionally, future work could focus on combining auditory and visual inputs to clarify signs that have similar hand configurations but different meanings. This multi-modal approach could further refine the precision of ASL recognition models.

In conclusion, advancing ASL recognition models involves a multifaceted strategy: from integrating multiple modalities and leveraging cutting-edge deep learning technologies to enriching datasets and developing real-time systems. Together, these approaches aim to create more accurate, robust, and comprehensive ASL recognition tools.

REFERENCES

[1] Computer Vision: Algorithms and Applications" by Richard Szeliski: It covers computer vision fundamentals, which are essential for real-time image and video analysis in sign language detection systems.

[2] Deep Learning" by Ian Goodfellow, Yoshua Bengio, and Aaron Courville: This book is great to understand deep learning techniques, including convolutional neural networks (CNNs) and recurrent neural networks (RNNs), which are crucial for image and sequence processing.

[3] Hands-On Computer Vision with OpenCV and Python" by Riaz Munshi, Steven L. Palmer, and Kemal Tugrul Sandeep Mudigonda:

[4] Multimodal Interaction in Image and Video Applications" edited by Cha Zhang and Zhengyou Zhang

[5] Sign Language Recognition, Translation, and Production" edited by Rami Abielmona and Onur Tuncer: This book is specifically focused on sign language recognition and can provide insights into the challenges and solutions in this field.

[6] Real-Time Sign Language Recognition from Video: A Comprehensive Overview" by Tanzeem Syeda Anjum: This research paper provides an in-depth exploration of real-time sign language recognition, making it a valuable reference for your project.

[7] Real-Time Sign Language Recognition Using a Range Camera. Authors: Ariya Rastrow, Jin Z. Zhang Published in: Proceedings of the 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, 2008
Link: https://ieeexplore.ieee.org/document/4563097