Sign Language Recognition Report

Hira Mahmood
HM26732
Software Engineering
University of Texas
Austin, USA

Shaharyar Mahmood
SM78856
Software Engineering
University of Texas
Austin, USA

Danish Mahmood
DM55444
Software Engineering
University of Texas
Austin, USA

Project Code URL: https://github.com/Hira-Mahmood/ScalableMachineLearning.git

I. Introduction

Technology has made tremendous strides in helping people who are deaf or hard of hearing to communicate more effectively. One approach that has gained a lot of attention in recent years is using deep learning and computer vision to interpret sign language gestures.

One example of this technology is sign language recognition systems, which use cameras and computer algorithms to analyze a person's hand movements and translate them into text or speech. This can be incredibly helpful for people who use sign language as their primary mode of communication, but may struggle to find an interpreter in certain situations. For example, a deaf person who needs to speak with a doctor or a government official may find it difficult to convey their message without an interpreter present. With sign language recognition technology, however, they could communicate more easily and independently.

Another area where deep learning and computer vision could be particularly helpful is in developing automatic editors that can recognize and interpret hand gestures. For example, someone who is deaf or hard of hearing might use a system that tracks their hand movements and translates them into text on a screen. This could be particularly useful in situations where it's not practical to use sign language - for example, in a noisy environment where it might be hard to see or hear the other person clearly.

Of course, there are still some challenges to overcome when it comes to using deep learning and computer vision for sign language recognition. One of the biggest obstacles is creating algorithms that are accurate and reliable enough to be useful in real-world situations. This requires a lot of data and testing, as well as careful attention to factors like lighting, hand position, and background noise.

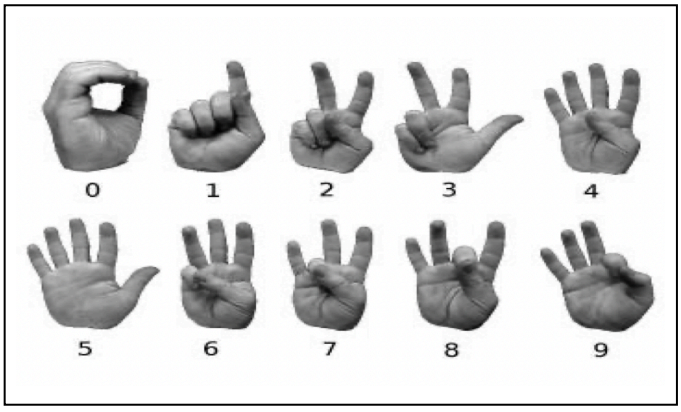
Despite these challenges, the potential benefits of this technology for people who are deaf or hard of hearing are enormous. By enabling more effective communication, these systems can help to break down barriers and promote greater inclusion and accessibility for everyone.

II. Objective

The goal of this project is to develop a real-time sign detector that can recognize numbers from 0 to 9 in sign language. However, the technology used in this project can be easily extended to recognize other signs and hand gestures, including the alphabets. The project is developed using two main modules of Python - OpenCV and Keras.

OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. It is used in this project to capture video frames from a webcam and to preprocess the images before feeding them into the deep learning model. The captured video frames are then converted to grayscale, resized, and normalized before being fed into the deep learning model.

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow. It is used in this project to build a Convolutional Neural Network (CNN) model to classify the signs. The CNN model is trained on a dataset of sign language images consisting of 10 classes (numbers from 0 to 9).



1. digits sign language. (*figure 1*)

The dataset used for training the model consists of 15,000 images of sign language digits. The dataset is divided into two parts - a training set consisting of 14,650 images and a validation set consisting of 350 images. The training set is used to train the CNN model, while the validation set is used to evaluate the performance of the model during training.

The CNN model consists of four main layers - a convolutional layer, a pooling layer, a flatten layer, and a fully connected layer. The convolutional layer is used to extract features from the input images. The pooling layer is used to downsample the output of the convolutional layer, reducing the number of parameters in the model. The flatten layer is used to convert the output of the pooling layer into a one-dimensional vector. The fully connected layer is used to classify the input images into one of the 10 classes.

Once the CNN model is trained, it is used to predict the class of the sign language digits in real-time. The sign detector captures the video frames from the webcam and preprocesses them before passing them to the CNN model for classification. The output of the model is then displayed on the screen, indicating the predicted sign language digit.

III. PROJECT MODULE DETAILS

This Project is breakdown into three modules.

1. Creating the dataset
2. Training a CNN on the captured dataset
3. Predicting the data

A. Creating the dataset

Creating a dataset on our own was more challenging than using pre-existing datasets, but it allows us to customize the dataset to our specific needs and improve the accuracy of the sign language recognition system. For this project some of the dataset we used was pre-existing and some we created by ourself using **create_dataset.ipynb**

To create a dataset for sign language recognition, we used OpenCV to capture live video data from a webcam and save the frames that detect a hand in the ROI to a directory containing training and testing folders. Here are the steps involved in creating the dataset:

1. Use OpenCV to capture live video data from a webcam.
2. Define the ROI, which is the region of interest in the frame where we want to detect the hand gestures.
3. Calculate the accumulated weighted average for the background by capturing a certain number of frames (e.g., 60 frames) before detecting any objects or hands in the ROI.
4. Subtract the accumulated weighted average for the background from each frame that contains an object or hand in the ROI to extract the foreground.
5. Use cv2.putText to display a message instructing the user to not put any objects or hands in the ROI while detecting the background.
6. Save the extracted foreground frames to a directory containing training and testing folders for further preprocessing and training.

The dataset used for training the model consists of 15,000 images of sign language digits. for each number sign the training set consisting of 1,465 images and test dataset consisting of 35 images.

B. Training a Convolutional Neural Network (CNN)

After creating the dataset, the next step is to train a convolutional neural network (CNN) on the captured dataset. This process is preformed in training_CNN.ipynb.

The first step in this process is to load the data using the ImageDataGenerator module of Keras, which is used to generate batches of image data with real-time data augmentation. The flow_from_directory function of the ImageDataGenerator module is used to load the train and test set data, and the name of the number folders in the directory will be the class names for the images loaded.

plotImages function is used to plot sample images from the loaded dataset. This function takes in the dataset and the number of images to be plotted as arguments. It then selects random images from the dataset and plots them in a grid format using Matplotlib library. The purpose of this function is to visually inspect the dataset and verify that the images are loaded correctly

and represent the intended gestures. It can also help in identifying any inconsistencies or issues with the dataset such as incorrect labeling or variations in lighting or background.

After loading the data, the next step is to design the CNN architecture. In this project, a simple CNN architecture is used, which consists of two convolutional layers, followed by two max-pooling layers, and finally two dense layers. The CNN architecture can be modified depending on the specific requirements of the project.

After designing the CNN architecture, the model is compiled and fit on the train batches for a certain number of epochs (in this project, 10 epochs were used). The

callbacks Reduce LR on plateau and early stopping are used during the training process. The Reduce LR on plateau callback is used to reduce the learning rate of the model if the validation loss is not decreasing, while the early stopping algorithm is used to stop the training process if the validation accuracy keeps on decreasing for some epochs.

After fitting the model, the next step is to evaluate the model on the test set and print the accuracy and loss scores. This step is important to check the performance of the trained model on unseen data.

Finally, a small test is done on the trained model to check if everything is working as expected while detecting on the live cam feed. The **word_dict** is used to map the predicted labels to their

corresponding names. The predicted labels are displayed on the live cam feed to show the performance of the model in real-time.

```
2023-04-09 12:52:01.403439: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin o
 to scroll output; double click to hide PU is enabled.
1468/1468 [==============================] - 12s 8ms/step - loss: 1.3358 - accuracy: 0.5619 - val_loss: 1.2491 - v
al_accuracy: 0.6038 - lr: 0.0010
Epoch 2/10
1468/1468 [==============================] - 11s 8ms/step - loss: 0.5932 - accuracy: 0.8079 - val_loss: 0.9256 - v
al_accuracy: 0.7232 - lr: 0.0010
Epoch 3/10
1468/1468 [==============================] - 11s 8ms/step - loss: 0.3582 - accuracy: 0.8882 - val_loss: 0.7047 - v
al_accuracy: 0.8091 - lr: 0.0010
Epoch 4/10
1468/1468 [==============================] - 12s 8ms/step - loss: 0.2324 - accuracy: 0.9265 - val_loss: 0.5372 - v
al_accuracy: 0.8473 - lr: 0.0010
Epoch 5/10
1468/1468 [==============================] - 11s 8ms/step - loss: 0.1597 - accuracy: 0.9503 - val_loss: 0.5321 - v
al_accuracy: 0.8353 - lr: 0.0010
Epoch 6/10
1468/1468 [==============================] - 11s 8ms/step - loss: 0.1111 - accuracy: 0.9656 - val_loss: 0.4627 - v
al_accuracy: 0.8902 - lr: 0.0010
Epoch 7/10
1468/1468 [==============================] - 11s 8ms/step - loss: 0.0680 - accuracy: 0.9802 - val_loss: 0.7823 - v
al_accuracy: 0.8162 - lr: 0.0010
Epoch 8/10
1468/1468 [==============================] - 11s 8ms/step - loss: 0.0291 - accuracy: 0.9934 - val_loss: 0.4330 - v
al_accuracy: 0.8902 - lr: 5.0000e-04
Epoch 9/10
1468/1468 [==============================] - 12s 8ms/step - loss: 0.0157 - accuracy: 0.9975 - val_loss: 0.3799 - v
al_accuracy: 0.9093 - lr: 5.0000e-04
Epoch 10/10
1468/1468 [==============================] - 12s 8ms/step - loss: 0.0106 - accuracy: 0.9988 - val_loss: 0.4117 - v
al_accuracy: 0.9165 - lr: 5.0000e-04
loss of 0.031777918338775635; accuracy of 100.0%
```

```
Result of trained model on a small set of test data

Predictions
-----------
Three   Three   Two   Zero   Two   Five   Seven   Seven   Two   Nine

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
Actual labels
-------------
Three   Three   Two   Zero   Two   Five   Seven   Seven   Two   Nine   (10, 64, 64, 3)
```

C. Predict the Sign Language

The third module for sign language recognition involves using a previously trained model to

predict the gesture made by a hand in a live video feed. (**model_for_gesture.ipynb**).

First we created a bounding box to detect the Region of Interest (ROI) in the video feed where the hand gesture will be made. This is done using computer vision techniques such as background subtraction or frame differencing to identify areas of motion in the video feed.

The accumulated average of the ROI is calculated, as was done during the creation of the dataset, for identifying any foreground object. This helps to identify the hand in the video feed and separate it from the background.

then we find the max contour and if contour is detected, that means a hand is detected, so the threshold of the ROI is treated as a test image. This step involves finding the largest contour in the ROI, which corresponds to the hand in the video feed. If a contour is detected, it means that a hand is present in the ROI and the threshold of the ROI is treated as a test image.

Load the previously saved model using **keras.models. load_model** and feed the threshold image of the ROI consisting of the hand as an input to the model for prediction. The saved model is loaded using Keras and the thresholded image of the ROI is fed as input to the model for prediction.

After detecting the hand in the ROI, the hand is now detected in the live cam feed using the same technique. and segmented i.e., the max contours and the thresholded image of the hand detected. This step involves getting the max contours and the thresholded image of the hand detected

If the hand is not detected, the program will continue to run until a hand is detected.else if the hand is detected, draw contours around the hand segment and show the thresholded hand image. This step involves drawing contours around the hand segment and displaying the thresholded hand image.

The thresholded image is resized, converted to grayscale, and reshaped to match the input shape of the trained model.

The gesture is predicted using the loaded model and the thresholded image of the hand. The predicted gesture is drawn on the video feed using OpenCV. The number of frames is incremented to keep track of the video feed and the segmented hand is displayed. finally the windows are closed with Esc key.

IV. CHALLENGES AND RECOMMENDATION

A. Large and diverse dataset:

To train a CNN model for sign language recognition, a large and diverse dataset is crucial. The dataset should include a variety of sign languages, signs, and gestures to ensure that the model learns to recognize a wide range of signs accurately. Additionally, the dataset should be well-balanced, with a sufficient number of samples for each class. A well-balanced dataset helps to avoid biased models, where the model may perform well on some signs and poorly on others due to a lack of sufficient training data.

B. Tuning hyperparameters:

Hyperparameters are parameters that are set before the model training begins and cannot be learned by the model. Tuning hyperparameters such as learning rate, batch size, and number of epochs can significantly impact the model's performance and training time. The learning rate controls how much the model adjusts its weights during training, the batch size determines how many samples are used in each training iteration, and the number of epochs defines how many times the model will see the entire dataset during training.

C. Incorporating other modalities:

Incorporating other modalities like audio or text can provide additional context to the model, improving recognition accuracy. For example, incorporating audio can help the model distinguish between signs that have similar hand shapes but different meanings based on the accompanying audio. Similarly, incorporating text can help the model understand the context of the sign and its meaning.

D. Insufficient data or poor quality data:

Insufficient data or poor quality data can lead to biased models and negatively impact the system's accuracy. It is essential to ensure that the dataset used for training is of high quality and that there are enough samples for each class. Additionally, it is essential to carefully preprocess and clean the data to remove noise and artifacts that may affect the model's performance.

E. Limitations in the camera's resolution or frame rate:

The camera's resolution and frame rate can affect the system's ability to detect and recognize signs accurately. Low-resolution cameras may not capture enough detail to distinguish between similar signs, while a low frame rate may not capture the sign's motion accurately, leading to misclassification.

F. Challenges in interpreting sign language gestures:

Interpreting sign language gestures that may be similar in appearance but have different meanings is a significant challenge for sign language recognition systems. It requires the model to understand the context and semantics of the gesture, which can be challenging, especially for complex signs. Researchers are working on developing more advanced models that can incorporate additional context to improve recognition accuracy for complex signs.

REFERENCES

1. Real-Time Hand Gesture Detection and Recognition Using OpenCV" by M. P. Thapliyal and M. M. Gaurav. This paper provides an overview of hand gesture recognition techniques using OpenCV.
2. Sign Language Recognition System: A Review" by S. S. Rathod, S. B. Nimbekar, and R. M. Dhage. This paper reviews different approaches and techniques for sign language recognition systems.
3. Sign Language Recognition: From Historical Perspective to the State of the Art" by M. Mustafa and M. A. Malik. This paper provides a comprehensive review of the history, challenges, and recent advancements in sign language recognition.
4. Sign Language Recognition using Machine Learning Techniques: A Review" by S. Kumar and S. K. Singh. This paper reviews different machine learning techniques and approaches for sign language recognition systems.
5. American Sign Language Recognition using Neural Networks" by R. Sharma and M. M. Gaurav. This paper presents a sign language recognition system using convolutional neural networks.