ASL Interpreter

Alex Noble and Benjamin Salvesen

The University of West Florida, 11000 University Parkway, Pensacola, FL, USA
acn11@students.uwf.edu, bs144@students.uwf.com

1 Introduction

1.1 American Sign Language

American Sign Language (ASL) is a visual-gestural language used by millions of individuals worldwide as their primary means of communication. However, despite its prevalence, barriers to communication still exist for deaf and hard-of-hearing individuals, particularly in contexts where ASL interpreters may not be readily available. In response to this need, our project focuses on developing a real time, camera-based ASL interpreter system. This system aims to recognize and interpret ASL gestures in real-time, providing a means for seamless communication between individuals who use ASL and those who may not be fluent in the language. We were inspired by this project, ”Sign Language Recognition Using Python and OpenCV”[3].

1.2 ASL Interpreter

The problem we aimed to solve with our ASL interpreter is the fact that there are a lack of accessible and efficient communication tools for individuals who use ASL. Traditional methods of communication, such as text-based or oral communication, are not always suitable or effective for conveying ASL to someone who is not fluent in ASL. Additionally, the availability of ASL interpreters may be limited in certain situations, leading to communication barriers for deaf and hard-of-hearing individuals. A system that could be integrated into online meetings, subtitle creation, or other applications would make a world of difference to those who do not understand American Sign Language as well as those who rely on it to communicate effectively. Additionally, it could be a means of education for those who wish to learn American Sign Language.

Our solution involves using machine learning techniques, particularly Convolutional Neural Networks (CNNs), to develop a camera-based system capable of recognizing and interpreting ASL gestures. By training our model on a dataset composed of thousands of images of ASL gestures representing five letters, we aim to create an accurate and reliable interpreter that can recognize ASL fingerspelling in varying contexts. Fingerspelling is part of ASL and is used to spell out English words. In the fingerspelled alphabet, each letter corresponds to a distinct handshape[1]. Through this project, we seek to contribute to the advancement of accessibility technology and provide a means of enhanced communication for individuals who use

ASL. Additionally, we aim to help others learn ASL by seeing the fingerspelling gestures and which letters they represent in real time.

2 Methods

To implement the ASL interpreter, the model employs a sequential architecture. This architecture is a linear stack of layers, which is suitable for straightforward tasks such as image classification where each layer feeds into the next. The network begins with an input layer that defines the shape of the input images, which in this case are 160x160 pixels and a single color channel (grayscale) is applied to the images. The input layer is followed by a series of convolutional layers that are paired with max pooling layers. The three convolutional layers have 32, 64, and 128 filters respectively, and use a 3x3 kernel size as well as a ReLU (Rectified Linear Unit) activation function. ReLU was chosen because it has the ability to introduce non-linearity without affecting receptive fields. Max pooling is implemented following each convolutional layer with a 2x2 pooling window to reduce spatial dimensions. In turn, this decreases the potential for overfitting while preserving important features.

Post convolutional and pooling layers, the architecture includes flattening, which transforms the 2D feature maps into a 1D feature vector. This vector feeds into a dense layer with 512 units, also utilizing the ReLU activation function. A dropout layer with a rate of 0.5 follows, which reduces overfitting by randomly setting a portion of the input units to zero during training. This improves generalization to new and unseen data. The final part of the model is a dense output layer with 5 units corresponding to the number of classes (letters), with a softmax activation function to produce a probability distribution across the class labels.

For training the model, the ADAM optimizer is used. The model uses categorical cross entropy as the loss function, which is appropriate for multi-class classification tasks. Accuracy is monitored as a metric during training to evaluate performance throughout epochs to ensure that the model is training well.

To enhance training efficiency and model performance, two callbacks are implemented. The EarlyStopping callback monitors validation loss and stops training when there is no improvement in the model's learning in five consecutive epochs. Additionally, it restores the weights from the best epoch when stopping occurs. This prevents overfitting as well as unnecessary training time. The ModelCheckpoint callback saves the best model based on maximum validation accuracy, ensuring that the model saved is the one that performs best on the validation dataset. The ModelCheckpoint did not work effectively for us, but stayed in the final code so that we could store multiple models to test against one another in an attempt to find the better performing model out of the two, which was consistently the one saved at the end of the file.

The training process is executed over 20 epochs with the previously specified callbacks using the training and validation data. This configuration optimizes training time and model performance but also ensures robustness and generalizability by validating the models learning against unseen data, which was 20 percent of the image data contained in the training set.

3 Experiments

In our initial experimentation phase, we began by capturing sample images of fingerspelling ASL letters against a dark (black) background. However, we encountered challenges with accuracy due to the sensitivity to changes in lighting and color, resulting in an overall accuracy range of 30-40% when classifying unseen images of the ASL letters we trained on. We realized that the appearance of individuals' hands would completely throw off the accuracy if there was a change in color or lighting so we had to change our approach. The next change implemented was converting the images to grayscale. In doing this, we got much more consistent results across different lighting configurations, mostly due to the fact that the color in our hands was no longer considered by the model.

To improve on this even further, we implemented a filter, which would mask the background and highlight the hand's silhouette and contours in red. This adjustment significantly improved model performance since it no longer learned from the noise of the background. We then captured 1000 pictures with this filter applied directly into the image capturing process for each letter (A-E) and trained our model to recognize the hand gestures correctly. Training was done by reading the image files from the "image" folder, tuning the parameters described in the methods section to train/validate the model, and ultimately training a more accurate and robust model.

After the training was complete, we were able to do real-time testing using a live camera feed. The tester could provide ASL (A-E) fingerspelling gestures and the model would make a prediction every second, recognizing the letters, and prompting the tester as to which letter was being performed. Lighting is still a factor, and the model will not be able to recognize hand gestures if the image and lighting is too dark to allow for features to be extracted using the filter.

4 Results & Conclusions

In conclusion, our method utilizing Convolution Neural Networks (CNN) proved to be effective. We were able to achieve real-time ASL interpretation utilizing the camera. Using OpenCV's VideoCapture and the filter that we configured, training was improved drastically and classification was much more robust and accurate. Increasing the size of the data set (number of training pictures per letter) also helped the accuracy of interpretation by allowing the model to learn more effectively. We came to the conclusion that the effectiveness of our methods suggests that CNNs are well suited for ASL fingerspelling recognition. Our results are relatively strong because we are able to see them in real time with the camera filtering we made, but the model has its issues at times. If the data set is not relatively uniform for each letter and does not accurately capture features needed to tell them apart, misclassifications will certainly occur. Additionally, with a small data set (<500 images per letter), the model did not learn to tell the difference between the letters effectively and cared more about the position of the testers hand/wrist rather than the actual sign they held up.

With this being said, using fingerspelling letters (A-E), we can cleanly and accurately interpret the ASL gestures into their alphabetical letters. Our next steps would be including all the letters in the alphabet, including tougher letters to implement such as J and Z, which require a motion to gesture accurately. To do this, we would need to further refine our methods and take multiple frames at a time into account. To train the model to do this would provide a serious challenge, as did static ASL classification. For our scale in this project we decided to only implement training for a few letters (A-E) as it took a while to find configurations for the data itself, the model, and the filtering that we applied to tie it all together.

Figure 1. Training image for the ASL letter "A"

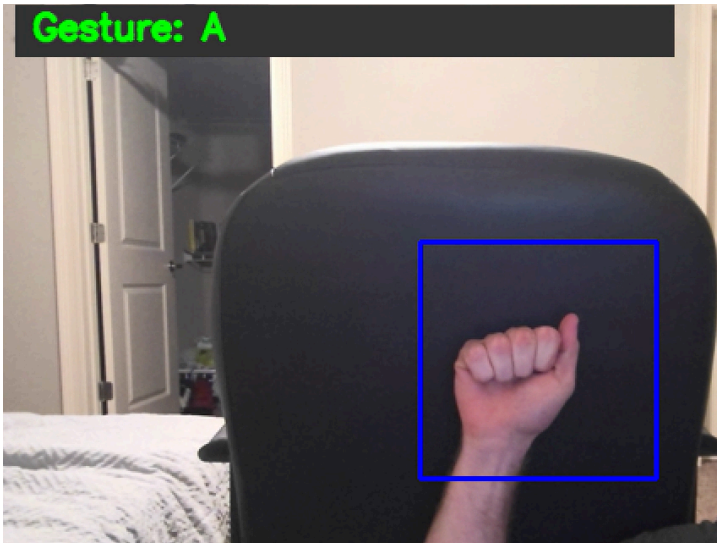

Figure 2. Accurate classification of the letter “A” in real time.

References

1. “American Sign Language.” National Institute of Deafness and Other Communication Disorders(NIDCD),NationalInstitutesofHealth,https://www.nidcd.nih.gov/health/american-sign-language. Accessed 04/21/2024
2. “What are Convolutional Neural Networks?” International Business Machines, https://www.ibm.com/topics/convolutional-neural-networks. Accessed 04/21/2024
3. “Sign Language Recognition Using Python and OpenCV” Data Flair, https://data-flair.training/blogs/sign-language-recognition-python-ml-opencv/