Sign Language ASL Recognition using Bag of Visual Words and Support Vector Machines

Samuel Ortega Cuadra[a] **and Laia Alexandra Sjöberg Cerezo**[b]

[a] *1669776*
[b] *1667894*

Abstract—This project focuses on developing a system for recognizing American Sign Language (ASL) alphabet and numbers. The proposed approach combines image preprocessing, feature extraction, and prediction models to accurately classify hand gestures. The dataset consists of images of hands representing ASL letters and numbers, which are processed by isolating the hand region, converting it in grayscale and applying a mask for feature extraction. Dense SIFT and a Bag of Words model are used to create feature representations, and a Support Vector Machine (SVM) classifier is trained and optimized through grid search for maximum accuracy.
The finished system can recognize hands by retrieving frames from video input, and translating ASL gesture sequences into coherent text.
This work lays a foundation for future developments in sign language recognition and improving accessibility for the deaf community by providing a tool for real-time translation of sign language to text.

Contents

1. Introduction

This document presents the development of a system for recognizing American Sign Language (ASL) alphabet and numbers. It outlines the steps taken to achieve this goal, from the **dataset collection** and preprocessing to the implementation of a **Bag of Visual Words** to help builiding a **SVM model** for classification. The system is then extended to **process videos**, extract frames, identify ASL symbols, and **reconstruct** the words being spelled.

2. Finding The Dataset

When looking for a good ASL dataset, we found the American Sign Language Dataset dataset on Kaggle. This dataset contains 2515 images of 400x400 pixels, each showing a hand making an ASL letter or number. The dataset is divided into 36 folders, one for each letter of the alphabet and one for each of the numbers from 0 to 9. The images are in JPEG format and have a black background with a white hand. The dataset is well-organized and easy to use, making it a good choice for our project.

2.1. Dataset Preprocessing

The preprocessing of the dataset invloves the following steps:

1. **Dataset Organization**: The dataset is organized into folders, one for each letter of the alphabet and one for each number from 0 to 9. Each folder contains 65 to 70 images of hands making the corresponding ASL symbol. A *categories* dictionary was created to map each label to its corresponding folder name. The filenames of all images were collected and combined with their respective category labels to construct a DataFrame.
2. **Shuffiling the Dataset**: To avoid any bias in the training and testing data, the dataset was shuffled. This ensures that the data is randomly distributed and that the model will not be trained on any specific order of images.
3. **Image Loading**: Each image was loaded from its respective path and converted into a NumPy array. Providing a new variable with a structured representation of pixels called *pixel_ data*.
4. **Grayscale conversion**: The images were converted to grayscale to reduce the complexity of the model and improve training speed.

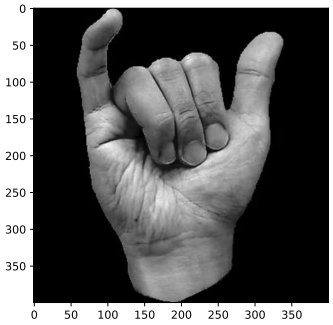


Figure 1. Grayscale original image

2.2. Masking the Hand

In order to find the best features we tried to mask the hand in the image using differents methods. The optimal resulting method was surprisingly the easiest. We took a look at the first 20 pixels of the image before converting it to grayscale and we found the most frequent color. Sequently, we applied a standard deviance and removed all of the background with the same colours, which will be really important when extracting the keypoints in the next chapter.

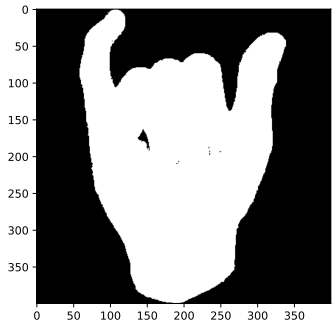


Figure 2. Mask of the image

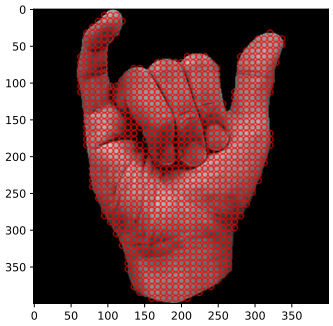


Figure 4. Mask applied Dense SIFT

> **Information**
>
> We tried other methods such as brightness threshold, gaussian blur and several others from the computer vision library. They all ended up being less efficient than the one we finally used.

3. Feature extraction

3.1. Keypoints

In the implementation of Bag of Visual Words there are two main steps to follow once you have your images: Detecting the keypoints and extracting the descriptors.

We used the SIFT algorithm to detect the keypoints in the image. This algorithm is one of the most implemented in the field of computer vision because of how robust it is. We first tried to use `detectandcompute()` (see 3a) but this created a huge problem, when taking the background into consideration. This process is non-positional, it does not matter it finds a hole in the middle of the image, (see 3b), it will be treated the same as a space in the left.

After this first approach we tried to implement **Dense SIFT** (3c), which improved the results considerably (increasing the final accuracy by almost 15%) but in the end the game changer was the mask we applied in the previous chapter. This allowed us to remove the background and focus on the hand, which allowed us to detect only the keypoints we were most interested in.

3.2. The Codebook

Once we have all the descriptors from our images, we can create a codebook. The idea behind this process is to create a "dictionary of visual words" that will represent the most common features in our dataset. This will help us find the meaning behind the keypoints we have detected.

In order to achieve this we implemented the KMeans algorithm. This will group all the descriptors found into 200 clusters which will define our codebook. This will later be used to create the histograms that will represent our images in the prediction step.

> **Information**
>
> The first number of clusters used was 36, which is the number of classes in the dataset. This was a mistake, as the codebook was not able to differentiate between the different classes. We then tried with increasing the number of clusters, which gave much better results.

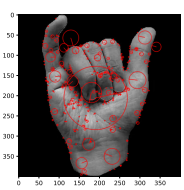
4. Predicting using the codebook

4.1. From Descriptors to Histograms

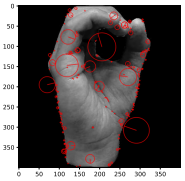
Using the KMeans algorithm mentioned before, we can convert each image into an histogram using the codebook. To do this we will predict to which cluster each descriptor belongs to and then we will count how many descriptors belong to each of them.

The main objective of this process is to predict which class the image belongs by its histogram based on the fact that similar images will present similar histograms.

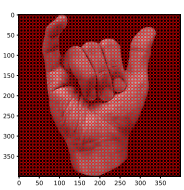
We can observe in Figure 5 that the first two histograms, corresponding to two different images categorised as the letter "a", are very similar. The main features (the peaks) are in the same position, while the histogram of the letter "p" is completely different. This is the main idea behind the idea of the codebook.



(a) Simple detect and compute

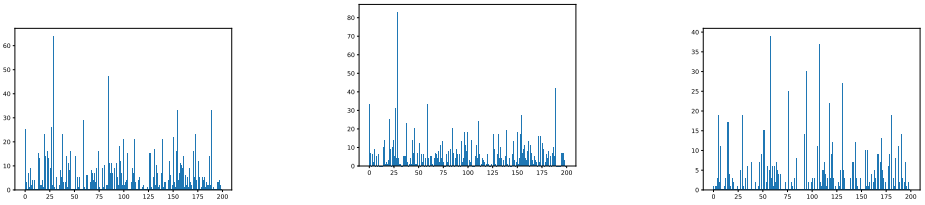


(b) Background mistaken keypoints



(c) Dense SIFT

Figure 3. Several keypoints detection methods considered



(a) Histogram of the letter a

(b) Histogram of a different photo of the letter a

(c) Histogram of the letter p

Figure 5. Comparison of histograms of different images

4.2. SVM

There are lots of different models that can be used to predict the categories of these histograms, the one we ended up choosing is SVM (Support Vector Machines). After doing a split between the train and the test we tried a starting model with a linear kernel. After this first approach, we tried to improve the model by using a grid search to find the best hyperparameters. After looking at the classification reports, this are the results:

	Precision	Recall	F1-score
Simple model	0.96	0.96	0.96
Grid search	0.97	0.97	0.97

Once we have the complete process, we compressed all the operations in one unique function `predict_image`

```
def predict_image(img_path,normalizer,km,
    best_model):
    # Load the image from a path
    img = Image.open(img_path)
    numpydata = asarray(img)

    # Get the mask of the image
    mask = get_mask(numpydata)

    #Convert to grayscale
    if len(numpydata.shape)==3:
        gray_data = numpydata.mean(axis=2)
        gray_data = gray_data.astype(np.uint8)
    else:
        gray_data = numpydata.astype(np.uint8)

    # Get dense keypoints
    dense_kp = generate_dense_keypoints(
    gray_data,mask)

    # Compute descriptors
    _, des = sift.compute(gray_data, dense_kp,
    mask)


    # Create the histogram of the image
    im_features = np.zeros(k, dtype=np.float32)

    if des is not None:
        for feature in des:
            feature = feature.reshape(1, -1)
            idx = kmeans.predict(feature)
            im_features[idx] += 1

    # Normalize the histogram
    im_features = normalizer.transform(
    im_features.reshape(1,-1))

    # Predict the category using the svm model
    y_pred = best_model.predict(im_features)

    return categories[y_pred[0]]
```

Code 1. Predict Image Function.

5. Video Processing

Now we have a model that can correctly predict the category of an image. We can use this model to predict the category of each frame of a video[1]. To do this we will use the `cv2.VideoCapture()` function together with a ratio variable to take 1 of every few frames. This will allow us to process the video faster and will help us to avoid the transition frames between signs.

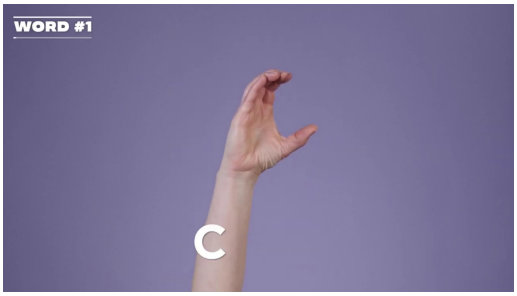


Figure 6. Frame of the spelling video

Once we have extracted some frames from the video, we need to process it into images we can predict.

6. Hand Detection

As it seems, there is a lot of noise in the frames extracted (text, coloured background, etc.). In order to have a better accuracy when predicting the ASL symbols, we first need to detect the hand in the frame.

To do this we used the `hands` library from `MediaPipe` in order to detect the hand in the frame.

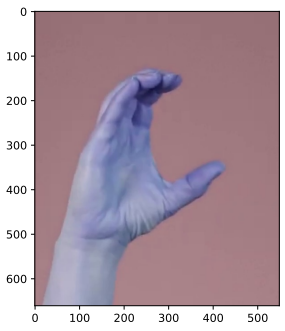
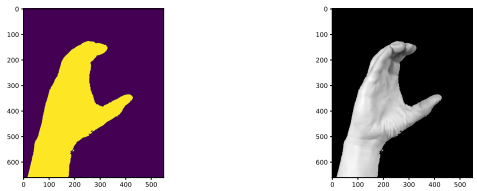


Figure 7. Hand detected in the frame

Once we have the cropped image we apply the same mask as before

[1]The video used corresponds to ASL Fingerspelling Exercise from *Learn How to Sign*

to remove the background and focus on the hand, in order to predict the result with better accuracy.



(a) Mask applied to the cropped image

(b) Result after applying the mask

Figure 8. Applying the mask to the cropped image

Once we have removed the background, just as way to standarize our photos, the images will be resized into 400x400px, the size of the dataset images.

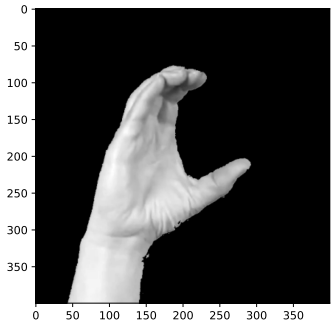


Figure 9. Final preprocessed hand

7. Final Results

After doing a prediction of all the images the results presented some problems. One of the main ones was the transition and "no sign" frames that appeared between the different signs. This derived in a lot of noise in the final prediction. In order to adjust this, one solution was to adjust the ratio of frames, or take a majority voting of several sections of the prediction.

However, a second problem appeared. Some signs, such as the "A" and the "T" or the "I" and the "J" are really similar which created confusion in the prediction on the model.

The predictor is able to detect and classify particularly selected frames, nevertheless, the solution to the noise and confusion problems is still under development. Here are some of the results of the prediction (see Figure 10).

8. Future Improvements

8.1. Category Coherence

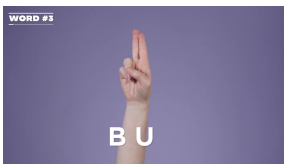
One of the main problems we have found when predicting is the different ways that exist to sign the same letter. This is a problem that can be solved by adding more variability to the images that train the model. We can appreciate this problem if we compare the letter "G" in the video and the dataset.

8.2. Transition Frames

The problem with the transition frames has many solutions, but none of them has ended working for the purpouse of the project. A future upgrade to the frame extraction step will be selecting only the frames that present a really small variance with the previous one. The idea behind this solution is to have into account that when signing, the hand will not move a lot between the frames corresponding to the same letter.

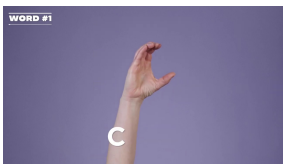
8.3. Model Improvement

The model can be improve by fine tuning the model even more or trying different classifiers. The solution we have in mind is implementing a Neural Network to predict the sign, but that solution will be applied in future approaches to this problem.



(a) Original frame of the letter U

(b) Processed frame predicted as "U"



(c) Original frame of the letter C

(d) Processed frame predicted as "C"

Figure 10. Image predictions of the video