Thibaut Baguette and Mohamed Elsamadouny
MAIS 202 Bootcamp
5 November 2021

Deliverable 2

1. Problem statement

The goal of the project is to write an AI-based program that would allow a user to draw shapes on a canvas by making signs and waving their hand in front of a camera. It would essentially be like MS Paint but with a camera and hand signs instead of a mouse interaction.

2. Data preprocessing

The dataset that was used for the hand sign detection is the one that was initially proposed: ASL Alphabet | Kaggle. The dataset is composed of a training set containing 3000 images for each category (ASL letter) and a testing set containing one image for each category. For this project, only a few categories/letters will be used, namely: L, A, S, G, K, V, B, and R. The letters A and S, as well as K and V will be combined as to increase the generalization capacity of the model. For instance, the letters A and S are both fists that only differ by the placement of the thumb. Combining the two letters will lead the model to have a more general "idea" of what the sign is. For these four letters, only 1500 images will be used, such that each category has a total of 3000 images. The labels will be strings, specifically the "command" column in the following table:

Command	**ASL letter**	**Sign**
Line	L	"L" shape
Ellipse	A, S	Fist
Rectangle	G	Point right (or left with left hand)
Triangle	K, V	Two fingers up (peace sign)
Move	B	Palm
Delete	R	Index and middle fingers crossed

The preprocessing itself consists of loading each image (with OpenCV), converting the color from BGR to RGB and applying the mediapipe hands solution. The conversion of color was implemented in mediapipe's hands example code and did increase the detection capacity and accuracy. The conversion from BGR to RGB or from RGB to BGR yields the same results (B and R are switched), therefore, even if the input images from the dataset were in RGB format, using the BGR to RGB conversion works as expected. The Google Colab file used to preprocess the data is available on this project's GitHub repository.

3. Machine learning model

As proposed in deliverable 1, the ML model for the hands recognition is a random forest classifier. This is the model that was used also for hand sign recognition in a project during MAIS Hacks 2021, LiveSigns. It did seem to yield good results, so there was no need to try other models.

a. Framework and tools

The model that was used is, specifically, `sklearn.ensemble.RandomForestClassifier`. The parameters were left to their default values since the accuracy scores were of 1.0. As is discussed

in sections 3.b and 4, adding extra datapoints in the testing set might be necessary in order to get a better idea of the model's performance and to adjust hyperparameters accordingly.

b. Validation methods

To validate the model, the test set provided by the database was used. However, since this test set consists of only one datapoint per category, it might be necessary to either take datapoints from the training set and put them in the testing set instead (and retrain the model with the new datasets), or to add datapoints of our own to the testing set. The model was tested also with live video feed on our own computers, and it seemed to classify the signs quite well. The sign that seems to be the least well recognized is the sign for 'delete', which is often "confused" for the 'triangle' sign. The model also seems to detect hand much better when the hand is closer to the camera, i.e., takes about ¾ of the video feed's height. This could be an issue as the goal is to be able to draw, which cannot be done if the hand takes this much space on the video feed.

c. Challenges faced

At first, two letters were used for the 'rectangle' sign, the letters G and H (also with the goal to generalize in mind). However, these two signs are actually quite different, as the middle finger completely changes position between the two signs. After further reflection, we decided to only use the sign for G. Another challenge was that, at first, when testing the model with a live video feed, it behaved very poorly. Upon further testing, we discovered that it worked very well with the left hand, but very poorly with the right hand. The issue was resolved by flipping the image horizontally when a right hand was detected. More precisely, when the mediapipe hands solution detects a right hand, the x-coordinate data points are replaced by the expression $1 - x$, since the coordinates are all comprised in the interval $[0, 1]$.

The model was saved using the `pickle` library, which saves python objects as .pkl files. This seemed to work correctly, however there might be a better solution for this specific case.

4. Preliminary results

The training and testing scores were each of 1.0. For the training set, this might indicate overfitting. For the testing set, it is probably a consequence of the fact that the testing set is very small, as well as the fact that the images in the testing set look very much alike the images in the training set. It is therefore very possible that the model is overtrained. From the live video feed testing, it seems still that the model behaves quite well, however this testing is not very quantifiable. As mentioned in 3.b, it might be worthwhile to add more datapoints in the testing set in order to get a more accurate representation of the model's performance. This could also allow for hyperparameters to be tuned, to improve the model's performance. The evaluation metrics presented in deliverable 1 were the accuracy of the sign detection as well as the accuracy of the drawing. The former has been discussed; the latter, however, has not yet been tested. The next step will be to code the drawing part of the project.

5. Next steps

The model behaves generally quite well and could be used as-is. However, it might affect the usability of the end program if the sign detection is not accurate enough. Fine tuning the model would therefore be beneficent. Perhaps this could be done through the tuning of hyperparameters, or through

different methods of data preprocessing. These include altering the colors in another way, or other image-related preprocessing processes; however, cropping is not possible in this situation since the hand must be able to move around the frame, and color alteration is already a part of the process.

The next steps also include the drawing part of the project. That is, taking as input the position datapoints of one hand landmark over a period of time, and fitting a shape to the datapoints by using a regression algorithm. The sign detection would determine which shape and therefore equation is sought (ellipse equation, line equation, etc.). If this is not possible, another solution would be to determine the shape's equation or size and format using only the strict minimum of points. For instance, for a circle, the user would select three points and a circle passing through these three points would be drawn (the three points could be set by holding the according sign in the same position for a predetermined amount of time). Finally, the equation would be used to draw the shape on a canvas, using a python library such as pygame.

Once this part works, the two parts will have to be brought together into the final product. Then, the program will be made into a webapp.