Sign Language Recognition

Camilla Savarese, Giorgia Fontana, Arturo Ghinassi, Valentino Sacco, Luca Romani

December 26, 2021

1 Abstract

Gesture recognition has become increasingly important in recent times. Why? Communicating with people with hearing disabilities can be difficult, therefore it is necessary to develope solutions for cross communication; furthermore during the pandemic sign language has also been adopted for domestic violence awareness, and those brief moment need a model capable of understanding hand gestures on the fly.

We have tried to create a model with this goal, starting from the Sign Language Digits Dataset, which contains a big set of pictures of sign language digits on a uniform background. In total we have 2180 images representing the numbers 0 to 9, in the format of numpy array and also one-hot vectors for the labels.

The main reference we took for this project, in order to evaluate our performance, is the paper "Arabic Sign Language Recognition", in which by using a Naive Bayes Classifier has been obtained an accuracy of about 0.65.

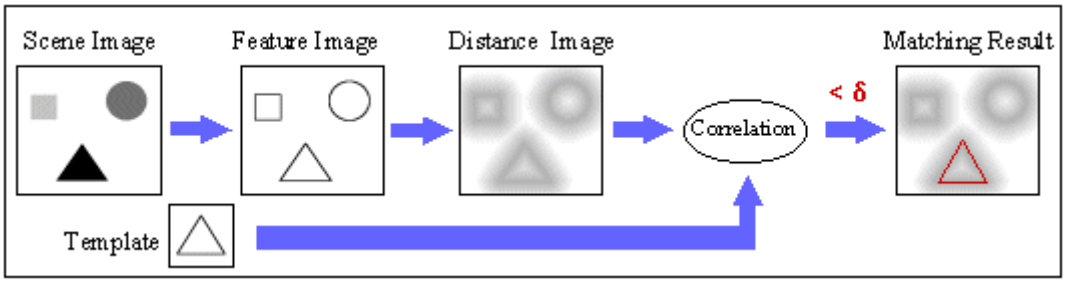
2 Method

We started by using a Naive Bayes classifier, implementing a probabilistic algorithm (based on Bayes' theorem) which calculates the probability of each label for a given object by observing its characteristics. Then, the classifier chooses the label with the highest probability. In practice we are looking for:

$$\arg\max_Y P(Y|X_1, ..., X_n) = P(X_1, ..., X_n|Y)P(Y)$$

Here we have neglected the denominator, being constant.
$P(Y|X_1, ..., X_n)$ represents the likelihood of a label given the features and $P(Y)$ is its prior.
(To be more precise we need to refer to a family of models, based on the assumptions on the distribution of the features, which will obviously lead to different shapes of the likelihood).
What makes this algorithm so efficient is the "naïve" conditional independence assumptions: all features X_i are mutually independent given the label. This, although not always true, works very well in practice, allowing to drastically reduce the parameters of the model.
Since the assumptions are naive, this approach has its limits and after facing them, we decided to see if we could enhance its performances by reducing the amount of features to deal with by preprocessing the dataset and performing Principal Component Analysis to what the classifier had to deal with.

We then moved on to a different approach based on the Chamfer distance:
The algorithms of this type rely on shape matching, using distance transforms and the shape of the target object is captured by a binary template. The scene image is pre-processed by edge detection and the so-called distance transform; this results in a distance image, where pixels contain the distances to the closest data pixels in the feature image.
Matching consists of translating and positioning the template at various locations of the distance image; the matching measure is determined by the pixel values of the distance image which lie under the data pixels of the transformed template. The lower these values are, the better the match between image and template at this location. If, for example, the average distance value lies below a certain threshold, the target object is considered detected.



The Chamfer distance we used as a similarity measure is defined as:

$$D(T, I) = \frac{1}{|T|} \sum_{t \in T} d_I(t)$$

where T is the set of points on template, I the set of points in the image and $d_I(t)$ is the minimum distance between point t and some point in I.

Finally we used Support Vector Machine (SVM). Basically, this algorithm finds a hyperplane that creates a boundary between data types. Learning takes place by capturing features of interests from the data, considered as a point in n-dimensional space (where n is the number of features, pixels in our case) with the value of each feature being the value of a particular coordinate. But what exactly is the best hyperplane? the one that maximizes the margins between all k classes. In other words: the hyperplane whose distance from the closest element of any other class is the biggest.

2.1 Multinoulli Naive Bayes

Our first classifier is the Multinulli Naive Bayes Classifier, in which by assuming we have discrete features, pixels $\in \{0, 1\}$, we transformed the images into black and white ones by comparing every pixels value to a certain threshold.
To set the better performing hyper parameter, we used the cross-validation with number of fields equal to 20 and, as can be seen in Figure 1, with a threshold of 0.59 the model obtained a score of 0.515.
Also, to get cleaner images, we have removed the corners, which are not useful for classification.

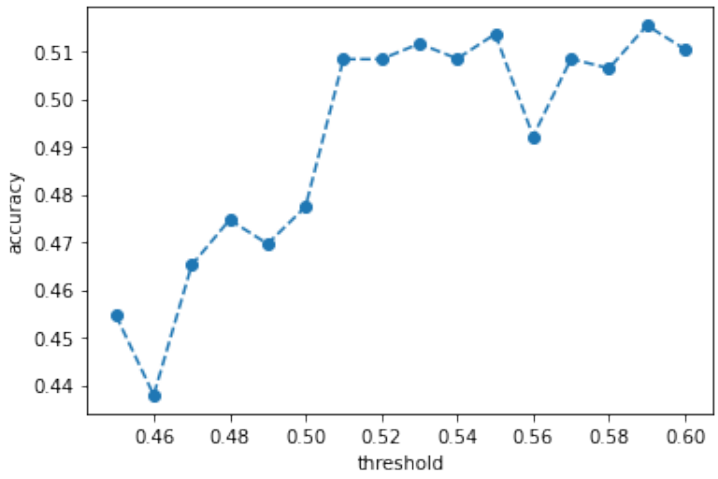


Figure 1: Accuracy wrt the threshold

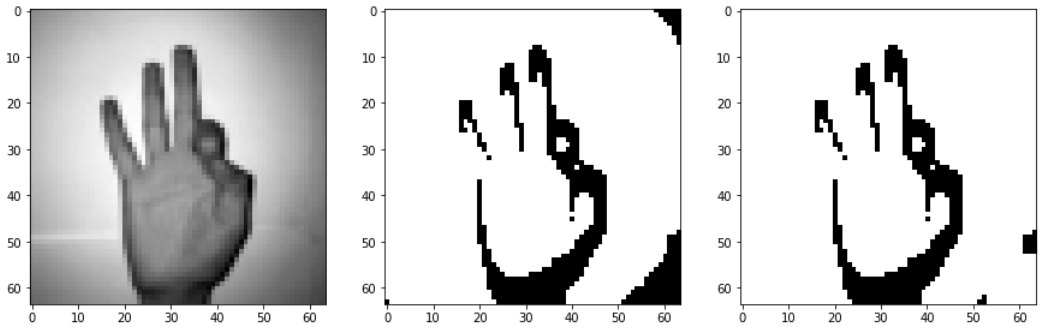


Figure 2: Dataset transformation

Testing the model in this way, the accuracy is about 0.52. It's not particularly high, and a plausible explanation is that as can be seen in Figure 2, this transformation does not fit our dataset very well: The pixels that will matter towards our likelihoods are the black ones, hence here is given much more importance to less relevant features such as the wrist, instead of focusing on the one having more variation such as fingers.

Concluding, we believe that discretizing the features is not the best choice in this case as it leads to an excessive loss of information.

2.2 Gaussian Naive Bayes

Following the Naive Bayes approach we proceeded with a Gaussian Naive Bayes, which assumes that the likelihoods of the features follow a continuous Gaussian distribution. This makes sense for our case as the dataset we're using is made of grey-scaled images of different nuances and illuminations.
Differently from the Multinulli approach, here we take into account the whole image, and training this model allowed us to reach an accuracy of 0.62.
The score is quite close to what we took as a reference, but as can be seen from Figure 3, there

is much confusion among some specific digits like number 2 and 4 which gets often recognized as number 6. Another sistematic error is predicting 6 while the true value is 4.

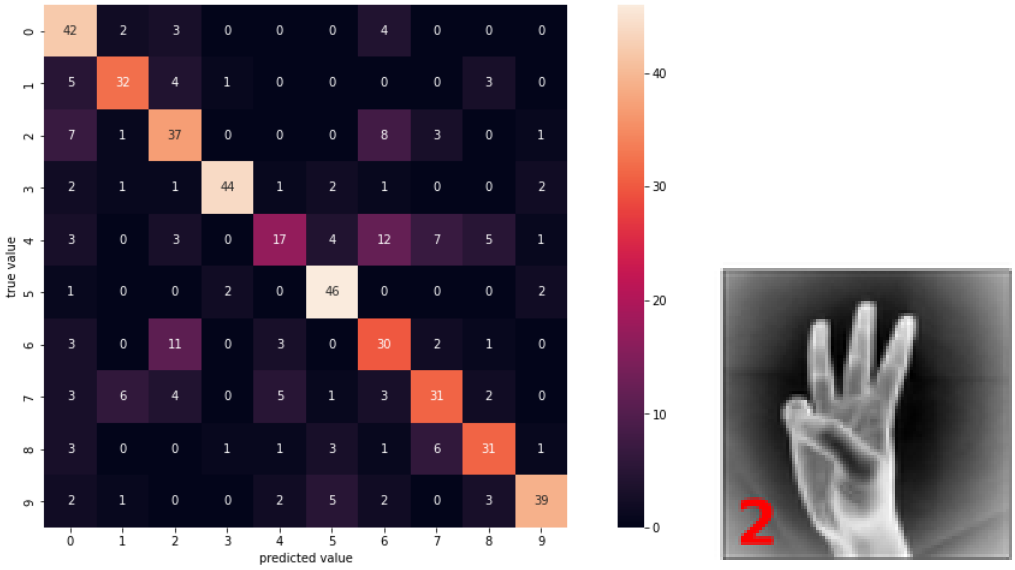


Figure 3: Confusion matrix and example of a 6 predicted as 2

2.2.1 Breaking Features

To better understand our problem we took a deeper look at the dataset. We have found that many images are inaccurate, rotated or have "anomalous" characteristics.

In Figure 4 we have other examples besides the one in Figure 3: in the first case the image is crooked, in the second case the glaze gives a greater relevance to those pixels. In presence of anomalies like these, our model doesn't work very well; to try to overcome this problem we decided to pre-process our dataset using Principal Component Analysis and then reapply the Gaussian Naive Bayes Model.

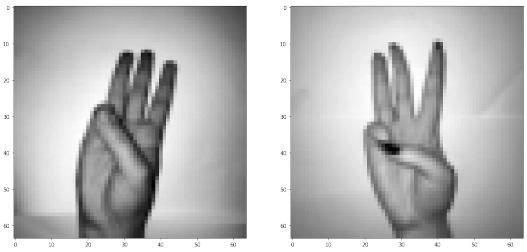


Figure 4: Confusing images

2.3 Naive Bayes with PCA

The PCA allows to obtain a dimensional reduction, keeping only the directions of greater variance. In this way we should be able to capture the most important pixels, that is, those that most discriminate the images.
Since PCA is very dependent on the scale of the data and for the most part our dataset seemed to have the hand placed often in the same position, we decided to subtract the average image from the dataset itself.

As you can see in Figure 6, preprocessing our dataset led to reducing the impact of less variant features such as the wrist and the hand contours and therefore to enhance the impact of those that have the most variance.

After this, we used again cross validation to choose the number of components from our PCA. As

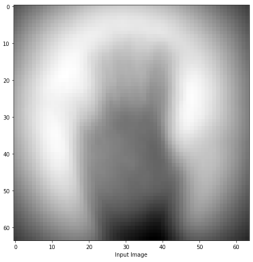


Figure 5: Average Image

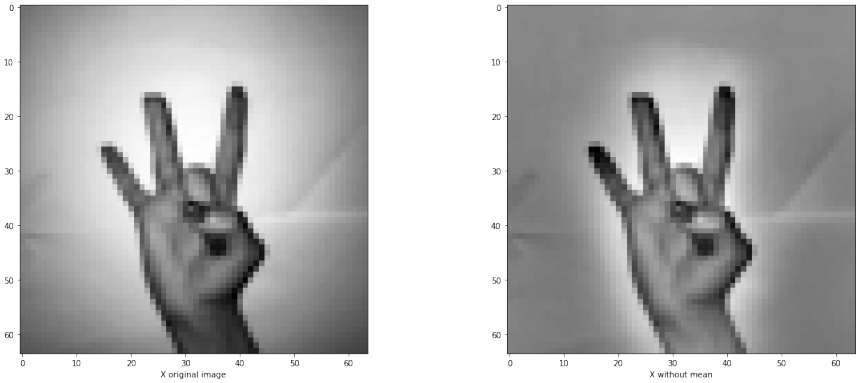


Figure 6: Pre-processed image

you can see from Figure 7 we obtained the best average score with 55 components : 0.73.
It is interesting to note that the trend is not monotonously increasing with respect to the number of components, this is because by taking too high a number of components, the non-relevant features are also included in the model, returning to the initial situation we were in. This can be intended as a sort of a trade-off between accuracy and information.

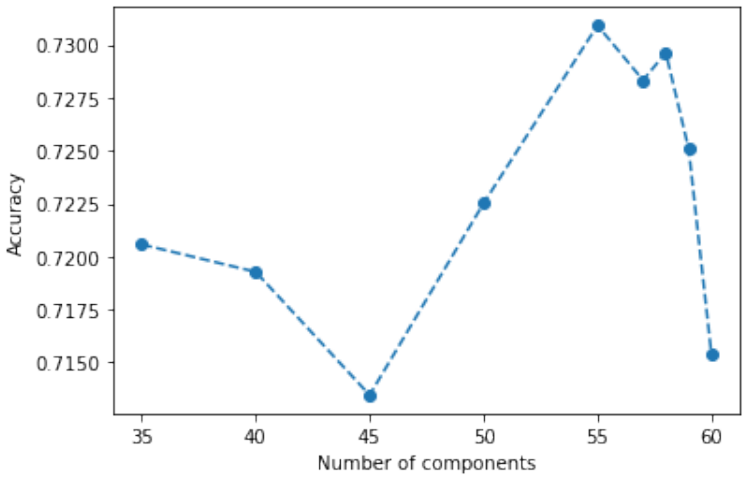


Figure 7: Accuracy wrt number of components

With the chosen model, we got an accuracy during testing of 0.763.

Effectively, applying the PCA actually improved the Naive Bayes' performance and reduced the confusion we had in the previous step. As noted from the matrix in Figure 8, there still is space for improvement, but we've obtained results we consider surprising and more than acceptable.

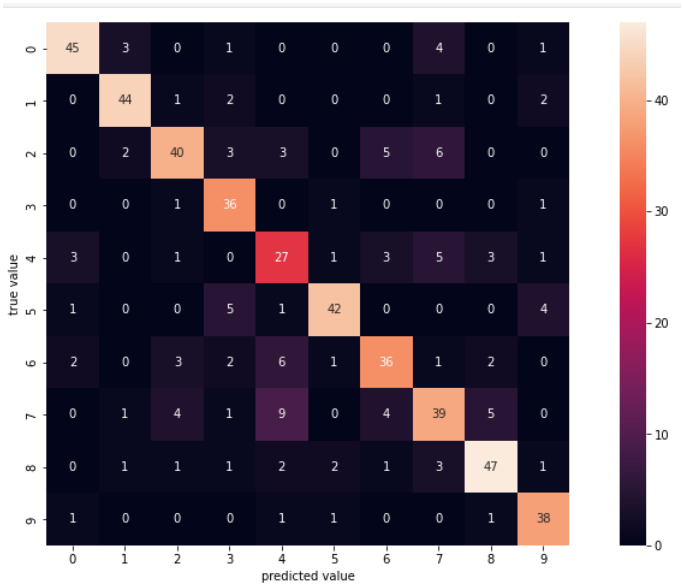


Figure 8: Confusion matrix PCA-NB

2.4 Edge Detection and Moulinoulli NB

Just as we tried to improve the Gaussian NB using PCA, we try to get better results by applying edge detection before implementing the NB multinoulli.
So the images that we pass to the model have been subjected to edge detection (we used FIND_EDGES from Pillow) in the sense that the edges are highlighted and everything else becomes negligible, because we consider them essential features for the classification.
In this way we get 0.68 as final accuracy: definitely a better result than the previous one! Actuallywe can see an improvement in the quality of the image compared to the first processing we had for the first approach with Multinomial NB.
From the confusion matrix it is clear that the model systematically predicts 6 instead of the true label 2, we have similar errors also for predicted 4 while true 8, 7 and predicted 2 while true 6.

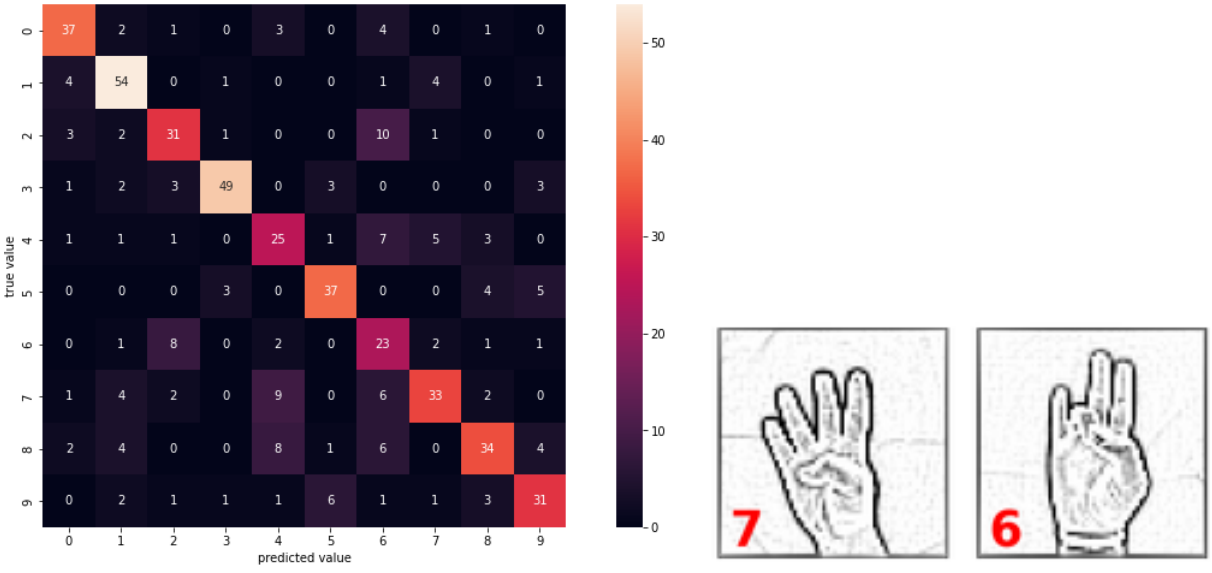


Figure 9: Confusion matrix and examples of errors

2.5 Edge detection, Chamfer distance

We have decided to consider also the Chamfer distance.
We tried many different approaches, because we couldn't get the desired results.
Firs of all, we selected $k = 5$ test images for each of the ten classes (randomly), so that each of these images was considered as a target from which to calculate the distance. For each "query image" that we pass to our model, we calculated the average Chamfer distance for each class, take the lower one and then comparing the labels.
Since the result was not good, in addition to the mean we added the absolute minimum, the median and a truncated average; moreover we have also tried different values of k (1,5,10,20 for each class) on a test-set of about 100 random images. In none of these cases the accuracy exceed 0.25.
As another attempt we tried to calculate the average image of each class and use this as a distance target, also applying a Gaussian filter to smooth the images,but also in this case the maximum accuracy is 0.39.

2.6 Support Vector Machine

Finally, let's move on to SVM.
For what regards the hyper-parameters we have C that is the cost of missclassification: a large C gives you low bias and high variance (low bias because you penalize the cost of missclasification a lot),a small C gives you higher bias and lower variance. Gamma is the parameter of a Gaussian Kernel (to handle non-linear classification): A small gamma gives you a pointed bump in the higher dimensions, a large gamma gives you a softer, broader bump. So a small gamma will give you low bias and high variance while a large gamma will give you higher bias and low variance.
For the selection of the hyperparameters we did the cross-validation with different levels of gamma and C and the best option seemed to be $\gamma =$ 'scale' ($\frac{1}{n_features * X.var}$) and $C = 100$.

We first tested SVM on standard images and obtained an accuracy during the testing of 0.86, a value we had never achieved before, we noticed only some troubles predicting 7's instead of 2.

We then pre-processed the dataset with the PCA, again using cross-validation (cv=20) to choose the number of components: the best case is 57 components with mean score 0.884.It should be noted that even taking a smaller number of components the model was efficient, with values not below 0.85.
Testing the model with our choice, the final accuracy is 0.895, and we can say that almost 90% of accuracy means a great model but we managed to improve it.

To do it, we applied SVM after applying a Gaussian filter for noise reduction. We used cross validation to choose between different σ values for the filter: (the larger σ, the greater the smoothness):

- sigma = 0.3 cross: 0.853
- sigma = 0.7 cross: 0.860
- sigma = 1.3 cross: 0.873
- sigma = 2 cross: 0.893
- sigma = 2.5 cross: 0.889

We then implemented the model with $\sigma = 2$ and finally reach a final accuracy of 0.907! Looking at the confusion matrix in Figure 10, the slight difference with the PCA has been made by classifying better the 7's with the 2's, while there are still some imperfections between 7 and 4.

2.7 Conclusion

We realized that there are many different approaches and algorithms to tackle an image recognition problem, and that you have to choose the best one based on your data.
In our case SVM is the one that obtains the best results ever, but also the moultinoulli NB after edge detection and the Gaussian NB with PCA seem promising, and one could think of continuing to analyze in more detail the errors they make for improve these models.

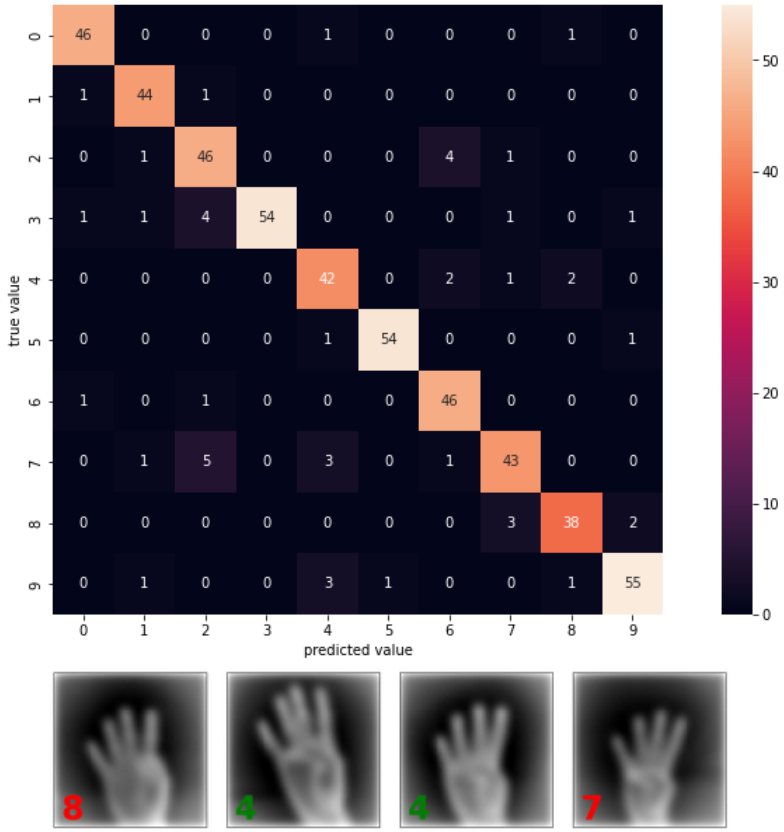


Figure 10: Confusion Matrix SVM

2.8 References

- https://medium.com/@dataturks/understanding-svms-for-image-classification-cf4f01232700
- https://www.analyticsvidhya.com/blog/2021/06/build-an-image-classifier-with-svm/
- https://vision.cs.utexas.edu/378h-fall2015/slides/lecture4.pdf
- Slides and laboratories from the lessons
- https://github.com/krrish94/chamferdist