Projet repport
Artificial Intelligence & Machine Learning (623.504, 22S)

Alexia HARIVEL
Baptiste CALLARD

August 7, 2022

Table des matières

I Introduction

In the course **Artificial Intelligence & Machine Learning (623.504, 22S)**, we have to carry out a project to apply the theoretical and practical knowledge we have acquired since the beginning of the semester. In this process, we will develop our project report in which we will present our motivations, our data as well as our models to solve our project.

II Motivations

We wanted a project related to computer vision because it was a field that interested us both. Moreover, we wanted a meaningful project. We chose to do it around the computer understanding of sign language. Our objective was to develop an algorithm capable of detecting in real time the most probable meaning of the sign made by the operator. This is a very interesting and important topic because it can allow people who do not know sign language to understand the message. In many cases, this could facilitate the inclusion of a mute person in different environments. The aim of our project is to be able to correctly classify 29 signs that correspond to the 26 letters of the alphabet as well as the signs that represents : SPACE (a space between two letters), DELETE (deletion of the last letter) and NOTHING (no sign just background). We based our study on two popular datasets. In this paper we have chosen to study the problem with two methods. The first method is the most conventional and used CNN. We have also implemented a method using SVM and preprocessing to extract the important features. As expected the results obtained with the CNN were better, we obtained more than 99% of well classified features. However, we were surprised by the performance of the method with the SVM which uses old methods. Finally, we have developed an interface to perform sign recognition in real time. The interface is attached to the project. As we wanted our model to be efficient and robust to run on real time data, we used two datasets. For the distribution, Alexia worked on the SVM part and Baptiste on the CNN part. The other parts were realized and thought together.

III Data

Thus, as introduced earlier, we were interested in the classification of sign languages. There are several sign languages depending on the country. We focused on American Sign Language (ASL) because there were more resources. We quickly came across a dataset on kaggle [1]. It consists of 87,000 labelled images of size 200x200 which are divided equally into 29 classes. Thus, all classes contain 3000 images. The 29 classes are the 26 letters of the alphabet and 3 classes for NOTHING, DELETE, SPACE. The only problem one could give to this dataset is its lack of diversity. Indeed, the images come from a video that has been cut into images (a video is available in this project [3]). So although the angle of the arm, the rotation and the zoom change over the course of the images, the background is almost always the same, the hand is always the same person. It is a very popular and massively used dataset as the statistics below show.

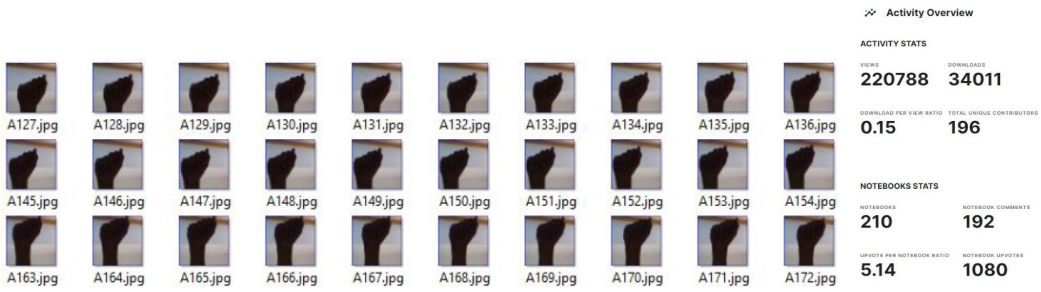


FIGURE 1 – Dataset 1 [1] : https ://www.kaggle.com/datasets/grassknoted/asl-alphabet ?resource=download

The second dataset [2] is complementary to the first as it has exactly the same classes and format, but it is more diverse and the images are more different. The backgrounds are different and it has different operators' hands. It is slightly less popular despite its quality. This is probably due to the small number of images compared to the first data set. It contains 100 times less images than the first one. That is to say 870 images of size 200x200 and labelled, which are fairly distributed in 29 classes. Thus, the two datasets are complementary and equally distributed but not balanced between them.

FIGURE 2 – Dataset 2 [2] : https ://www.kaggle.com/datasets/danrasband/asl-alphabet-test

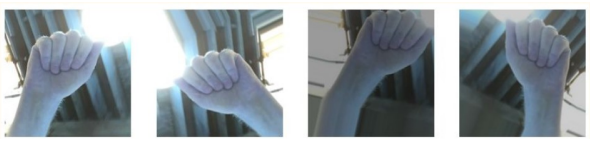
III.1 CNNs data utilisation and processing

Our first approach was to train the model only on the first dataset and to see if the predictions were good on the second. As shown in the following sections, the results were poor on the second dataset although they were excellent on the first. Thus, we then made the choice to integrate the second set into the first by augmenting the training data.

We first divided the two data sets into training and testing sets. For this, we used a stratify split with 0.1 (resp. 0.3) in the first (resp. second) dataset.

As for the pre-processing, we only normalized the images and then provided tensors of dimension (64,64,3,$n_{instances}$) to the CNN. We transformed the labels with the one hot encoding representation. We also performed data augmentation on the training data of the second dataset (not for testing data). For this we used the tensorflow function **ImageDataGenerator**. Thus, we only need to provide it with intervals and the model randomly generates a new image.

Here are some examples of generation for an image :



parameters	value	parameters	value
rotation range	±15	height shift range	±0.2
horizontal flip	True	brightness rangerange	[0.4,1.5]
width shift range	±0.2	fill mode	nearest

FIGURE 3 – Images augmentation

Set	Amount
training	78909 (without augmentation) / 90480 (with *20) / 114840 (with *60)
testing	8961

Finally, in order not to have all the augmented images to follow we shuffled the training and testing.

III.2 SVM pre-processing

Before building the support Vector Machine Model, we had to do some pre-processing.

— Since each image has color dimension, one of our pre-processing steps is to extract each of these to create separate blue, red and green training examples.

— The second pre-processing step is to create a Gaussian blur to see if it can help improve training accuracy. The Gaussian blur takes the average of a pixel's eight nearest neighbors and returns the average as its new pixel value.

— We try some image processing functionalities to extract the contour/edges of images. Edge detection can be tricky because there can be edges in the background as well and thus how tightly to identify the edges can affect the quality of the output image and the prediction. So we choose to not use it.

As the dimensions of our training data are still quite large after re-sizing the images, we try to reduce the dimensions using Principal Component Analysis in order to see if we can make the algorithms run more efficiently and if we can improve the accuracy for the classifiers. The goal was to retain 95 per cent of the variance. Originally we wanted to retain 99 per cent of the variance but discovered that this requires the number of components to go up to thousands, essentially defeating the purpose of PCA. That is why We choose to set the number of principal components to 110 so that we can reach 95 per cent explained variance.

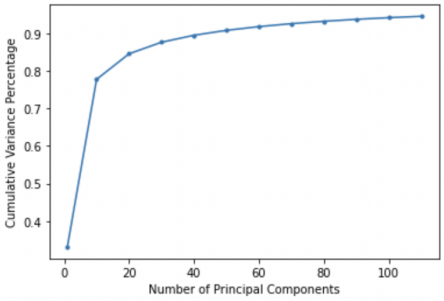


FIGURE 4 – Number of principal components vs cumulative explained variance

IV Theoretical part

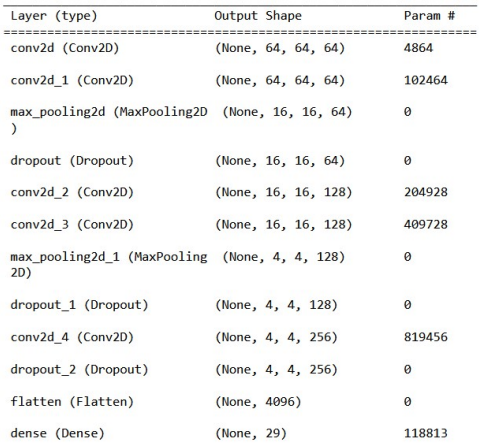
Our problem is in the field of computer vision which is a multi-class supervised image classification problem. Computer image analysis is a field that has been extensively studied. A major event in the research in this field took place with the appearance of neural networks. A second turning point was the creation of convoluted neural networks. Previously, the methods used were machine learning algorithms such as KNNs, SVMs, Random forest or PCA. The disadvantage was that these methods required a lot of data engineering time and domain knowledge. Indeed, the data scientist had to be able to extract the important features himself by applying filters and transforming the data. In contrast, the goal of CNNs is to use a model that can learn to extract features from a large amount of data, the only requirement is to have enough data. CNNs also have the advantage of taking into account the spatial dimension, which is not present with deep forward networks..

We have enough images to properly train deep learning or machine learning models. Naturally, we chose to implement a CNN, which is the most commonly used method. For comparison, we also wanted to try to implement an older method that uses the SVM algorithm coupled with filters to extract the data.

IV.1 CNN

For the training of the model, there were several parameters to optimise. We tested different architectures by varying the size of the kernel (conv and max polling), the number of layers,

number of neurons. An important part was the choice of the right layers and the right activation functions. For example, the Relu activation function made popular with AlexNet which allows to accelerate the convergence and easily the back propagation. Finally, it is also necessary to use softmax at the end, taking care to have the same number of neurons as classes. We also looked at whether dropout and batch normalisation would lead to better learning. We saw that it was not easy to find a model from scratch. So, while doing our research we found a structure that came up often that we chose to adopt. Then we had to choose the number of epochs and the batch size. We chose a batch size of 64. Each of the iterations taking between 1hour and 2hours and for time constraints we chose to limit ourselves to 7 epochs ($\sim$ 14h maximum). We finally chose a structure without batch normalisation but with dropouts. Thus our network has 1,660,253 parameters and has the following structure :



Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 64, 64, 64)	4864
conv2d_1 (Conv2D)	(None, 64, 64, 64)	102464
max_pooling2d (MaxPooling2D)	(None, 16, 16, 64)	0
dropout (Dropout)	(None, 16, 16, 64)	0
conv2d_2 (Conv2D)	(None, 16, 16, 128)	204928
conv2d_3 (Conv2D)	(None, 16, 16, 128)	409728
max_pooling2d_1 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_1 (Dropout)	(None, 4, 4, 128)	0
conv2d_4 (Conv2D)	(None, 4, 4, 256)	819456
dropout_2 (Dropout)	(None, 4, 4, 256)	0
flatten (Flatten)	(None, 4096)	0
dense (Dense)	(None, 29)	118813

FIGURE 5 – CNN

We used the Adam optimizer which is popular with these default settings. Our metric was accuracy and our loss function was categorical cross entropy which is widely used in classification. Once we had chosen and optimised all these parameters, we wanted to study the impact of data augmentation on our results. We wanted our model to be as robust as possible. Indeed, we wanted to use it to make predictions in real time. In total, we tested three models with different amounts of data augmentation on the training set of the second dataset. We did not augment the first set because it is larger than the second (100x). Moreover, the images are already very similar (figure 1).

Experimentation	Training	testing 1	testing 2
1	dataset1	dataset1	dataset2
2	dataset1+dataset2 (augmented x20)	dataset1+dataset2	Nan
3	dataset1+dataset2 (augmented x60)	dataset1+dataset2	Nan
4 (bonus in appendix)	dataset1+dataset2 (augmented x60)	dataset1+dataset2	dataset3

This last step allowed us to greatly increase the performance of our model. In the Notebook, a third set was tested. This one is completely different in the way the images are taken. On this 3rn dataset, the first two models had very poor results. On the other hand, the last model started to learn something with low accuracy but with interesting perspectives for transfer learning or data augmentation. We will put the results in the appendix but it remains a part of curiosity and evolution perspectives.

IV.2 SVM

Support vector machines work by placing hyperplanes through the data in N-dimensional space when a simple linear line can not separate the data.

First, we run the Red, Green, Blue and blurred data through a Support Vector Machine model along with transformations through PCA to determine which data set has the highest accuracy. As we can see on the figure 6, no dataset seems to have a better performance than another.

Support Vector Machine:

	Data Description	Accuracy
0	No_PCA	46.7%
1	PCA	48.7%
2	Red_PCA	47.9%
3	Blue_PCA	48.7%
4	Green_PCA	48.2%

FIGURE 6 – Comparison of the accuracy for different dataset

So we decide to perform a gridsearch to found the optimal parameters C and gamma.

C, also known as regularization, will choose a smaller-margin hyperplane to make sure all the data is classified correctly for large values of C. For small values of C, it will look for a larger-margin hyperplane even if that means misclassifying points.

A low gamma means points far from the likely separation line are considered in the calculation while a high gamma means points close to the separation line are considered in the calculation. After performing the gridsearch on the these two parameters, we were able to increase our accuracy to 66.0%. We found the optimal values of C = 100 and gamma = 0.01. Our model found a C value somewhere in the middle to be optimal for our case. Our optimal gamma value was fairly low which could mean our data was sparse and far from the decision boundary.

V Implementation

We did this project on Python.

V.1 CNN

For this part we did not encounter any problem for the implementation except for the execution time (+10h per training). Indeed, the difficulty lies in the convergence of the methods. This second part required dozens of hours of calculations and tests. We used Keras, which allows us to use deep neural network algorithms such as those of tensorflow. It is a library for us who were beginners because it offers a fast, visual and simple experience. We ran the programs on a CPU. Also, we recorded the models. To make the execution reproducible we were careful to set the seed on numpy and tensorflow.

The tools that have been most useful in our development are data augmentation and the CNN. For this we used the **ImageDataGenerator** :

```python
list_X_train_2 = []
list_y_train_2 = []

n = 100 # number of image generated from 1 (the original is counted in n_new = n - 1)

for img, label in zip(X_train_2, y_train_2):
    list_X_train_2.append(img)
    list_y_train_2.append(label)
    img = expand_dims(img, 0)

    datagen = tf.keras.preprocessing.image.ImageDataGenerator(
        rotation_range=15,
        horizontal_flip=True,
        width_shift_range=0.2,
        height_shift_range=0.2,
        brightness_range=[0.4,1.5],
        fill_mode='nearest')

    aug_iter = datagen.flow(img, batch_size=1)


    # generate batch of images
    for i in range(n-1):
        image = next(aug_iter)[0].astype('uint8')
        list_X_train_2.append(image)
        list_y_train_2.append(label)
```

```python
model = Sequential()

# first layer of convolution
model.add(Conv2D(filters = 64, kernel_size = 5, padding = 'same', input_shape = (64, 64, 3)))
model.add(Activation('relu'))

# second layer of convolution
model.add(Conv2D(filters = 64, kernel_size = 5, padding = 'same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (4, 4)))
model.add(Dropout(0.5))

# third layer of convolution
model.add(Conv2D(filters = 128 , kernel_size = 5, padding = 'same'))
model.add(Activation('relu'))

# forth layer of convolution
model.add(Conv2D(filters = 128 , kernel_size = 5, padding = 'same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (4, 4)))
model.add(Dropout(0.5))

# fifth layer of convolution
model.add(Conv2D(filters = 256 , kernel_size = 5, padding = 'same'))
model.add(Activation('relu'))
model.add(Dropout(0.5))

# fully connected
model.add(Flatten())

# prediction
model.add(Dense(29, activation='softmax'))
```

FIGURE 7 – code data augmentation and CNN

V.2 SVM

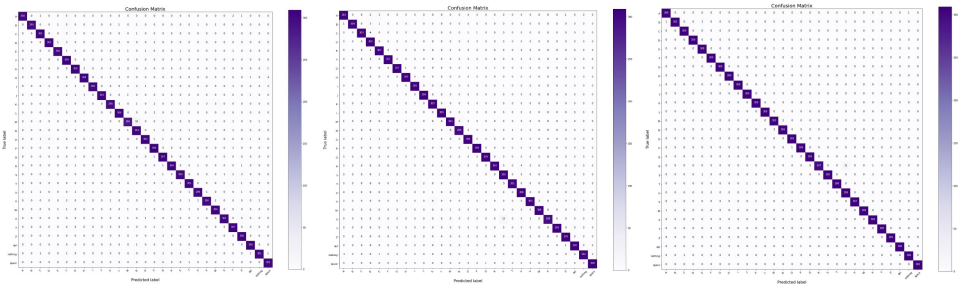
Performing the gaussian blur took several hours of computing, above all we repeat this task for each blue, red and green training examples from the first pre-processing. Also, computing the GridSearch was very long because of the number of different parameter combinations to test.

VI Evaluation

To evaluate our results, we used confusion matrices that allow us to quickly assess our results. We also used precision, recall and f1-score. Thus, we could compare our different CNNs with each other as well as with the SVM approach.

VI.1 Comparison of CNNs

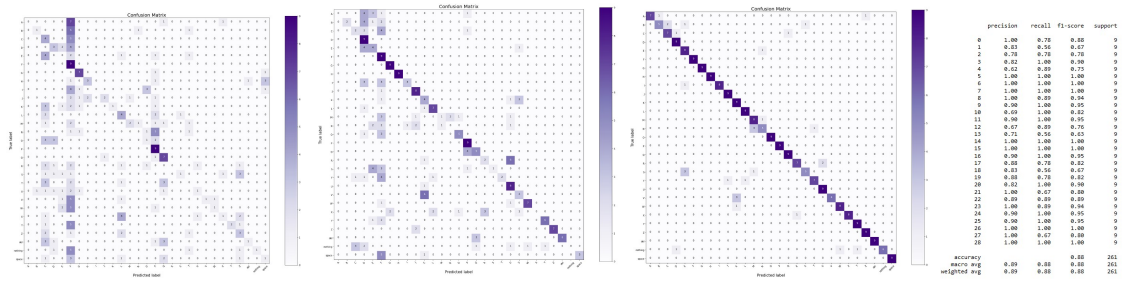
Thus, all our three experiments (see table above) show that the CNN is well dimensioned and powerful enough to solve our problem. Looking at the confusion matrices we can see that the models are equivalent even if they get better and better the more augmented images they use. Models 1, 2 and 3 obtain respectively almost 0.98%, 0.99% and 99.6% of accuracy. We will study the results further in the following section. We never evaluate our model on training images.



(a) experimentaion 1 without data augmentation (b) experimentaion 2 with x20 data augmentation (c) experimentaion 3 with x60 data augmentation

FIGURE 8 – comparison of confusion matrices

On the other hand, if we compare the confusion matrices only on the test set of the second dataset where the images are really different from the training data, we note that the performances are also clearly better for experiments 2 and 3. In fact, this shows that probability the first model was overfitting the first dataset and 2nd and 3rd can generalize with second dataset. Thus, our choice is naturally made for the third model which obtains 88% of success against 25% for the first model and 50%.



(a) experimentaion 1 without data augmentation (b) experimentaion 2 with x20 data augmentation (c) experimentaion 3 with x60 data augmentation (d) experimentaion 3

FIGURE 9 – comparaison des matrices de confusion et classification report pour le modèle 3

The third model makes some mistakes by confusing S with E, N with M and V with K. These signs are close even for human perception. We also tested with x100 for data augmentation but the model did not converge.

FIGURE 10 – similar images

VI.2 Comparison between CNN and SVM

To compare the CNN and the SVM quantitatively and qualitatively, we look at the following statistical values :

	precision	recall	f1-score	support
0	0.99	1.00	1.00	309
1	1.00	0.99	0.99	309
2	1.00	1.00	1.00	309
3	1.00	1.00	1.00	309
4	0.99	1.00	1.00	309
5	1.00	1.00	1.00	309
6	1.00	1.00	1.00	309
7	1.00	1.00	1.00	309
8	1.00	1.00	1.00	309
9	1.00	1.00	1.00	309
10	0.99	1.00	1.00	309
11	1.00	1.00	1.00	309
12	0.98	0.99	0.99	309
13	0.99	0.99	0.99	309
14	1.00	1.00	1.00	309
15	1.00	1.00	1.00	309
16	1.00	1.00	1.00	309
17	1.00	0.96	0.98	309
18	0.99	0.99	0.99	309
19	1.00	1.00	1.00	309
20	0.96	1.00	0.98	309
21	1.00	0.99	1.00	309
22	1.00	1.00	1.00	309
23	0.99	1.00	0.99	309
24	1.00	1.00	1.00	309
25	1.00	1.00	1.00	309
26	1.00	1.00	1.00	309
27	1.00	1.00	1.00	309
28	1.00	1.00	1.00	309
accuracy			1.00	8961
macro avg	1.00	1.00	1.00	8961
weighted avg	1.00	1.00	1.00	8961

(a) CNN

Dev accuracy: 73.0
Classification Report

	precision	recall	f1-score	support
1	0.66	0.61	0.63	100
2	0.62	0.75	0.68	100
3	0.77	0.79	0.78	100
4	0.72	0.71	0.71	100
5	0.40	0.72	0.52	100
6	0.76	0.74	0.75	100
7	0.84	0.81	0.82	100
8	0.75	0.88	0.81	100
9	0.84	0.76	0.80	100
10	0.88	0.74	0.80	100
11	0.80	0.76	0.78	100
12	0.89	0.75	0.82	100
13	0.66	0.70	0.68	100
14	0.82	0.69	0.75	100
15	0.75	0.72	0.73	100
16	0.81	0.79	0.80	100
17	0.89	0.81	0.85	100
18	0.65	0.60	0.63	100
19	0.72	0.65	0.68	100
20	0.70	0.79	0.74	100
21	0.61	0.62	0.61	100
22	0.59	0.51	0.55	100
23	0.57	0.69	0.62	100
24	0.73	0.70	0.71	100
25	0.71	0.70	0.70	100
26	0.79	0.71	0.75	100
27	0.90	0.84	0.87	100
28	0.92	0.97	0.94	100
29	0.85	0.66	0.74	100
accuracy			0.73	2900
macro avg	0.74	0.73	0.73	2900
weighted avg	0.74	0.73	0.73	2900

(b) SVM

FIGURE 11 – classification report

It is clear that the CNN model has better results than the SVM model. With the CNN, the precision, recall, F1-score for each class is 0.99 or 1.0 whereas the average of the precision, recall and F1-score is around 0.73. So we decided to use CNN for our interface to perform sign recognition in real time with webcam.

VI.3 Comparison with the literature

We find that our approach is good because we have implemented two different methods. A machine learning method and a deep learning method. Thus, we could observe the power of CNNs in image analysis. We had limited resources for memory and time. Thus, we were able to have live results with a webcam of computer resolution, which makes our project a success. As we explained at the beginning of this report, the field of computer is really documented, and the sign language recognition issue is the focus of several research papers. Many stages are involved in vision-based sign language recognition. Its can be categorised into five stages : image acquisition, image pre-processing, segmentation, feature extraction, and classification. Many people use CNNs. A different method to the one we have used and one that comes up often is to use the spatial coordinates of the fingers in space. This is for example possible with openCv models that have already been trained to have the position of fingers.

VII Interface

In order to test our model in real conditions, we implemented a very simple interface. We used the Mediapipe library to locate the hand(s) in the camera field and frame it with a red rectangle. The prediction of the sign made by the operator is done only from the photo of the hand (in the red rectangle). So whatever is in the rest of the camera field will not disturb the result. The sign found and its threshold are shown in real time at the top left of the video screen.
This allowed us to see that our model works relatively well in real life conditions, as shown in the pictures below.

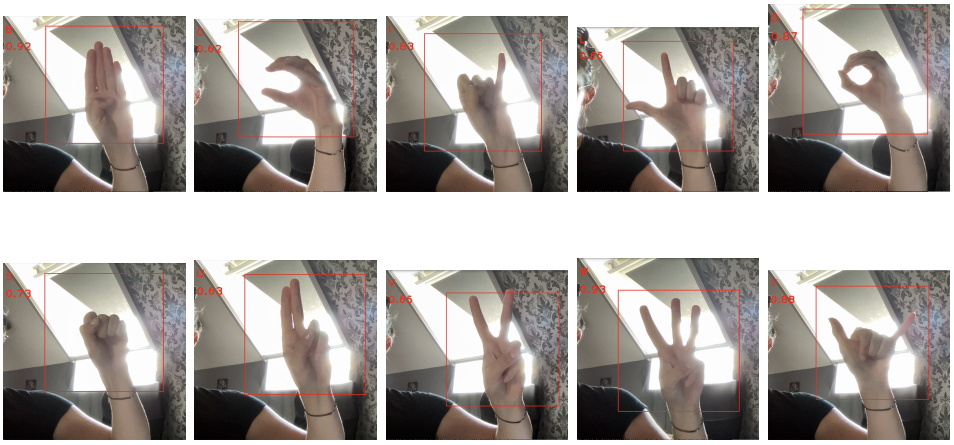


FIGURE 13 – Screen of the interface for different signs

VIII Conclusion

For SVM, we could see that we were limited by the capacity of the model because we were working in a very high dimensional space and this justified using deep learning. For the CNN, we had very good performances and we are satisfied with our results. As an improvement we think it would have been interesting to integrate more data in the training set. We were limited by the capacity of our computers and also because there are two other sets that contain ASL data but they do not contain the signs for del, space and nothing [4][5] (we already have some results with [4] in the appendix). It's easy to find images for nothing, for the other classes one could imagine doing data augmentation or over sampling to avoid having unbalanced classes. By using a google colab pro account we could easily manage this large amount of data and achieve much better performance. We have already implemented all the methods to achieve this.

IX References

[1] first dataset - kaggle : https ://www.kaggle.com/datasets/grassknoted/asl-alphabet ?re-source=download

[2] second dataset - kaggle : https ://www.kaggle.com/datasets/danrasband/asl-alphabet-test

[3] notebook - kaggle : https ://www.kaggle.com/code/danrasband/classifying-images-of-the-asl-alphabet-using-keras

[4] fourth dataset - kaggle : https ://www.kaggle.com/datasets/mrgeislinger/asl-rgb-depth-fingerspelling-spelling-it-out

[5] third dataset - Massey University : https ://www.massey.ac.nz/ albarcza/gesture_dataset2012.html

[6] notebook - kaggle : https ://www.kaggle.com/code/zeyadkhalid/sign-language-recognition-97-val-accuracy

[7] notebook - kaggle : https ://www.kaggle.com/code/gargimaheshwari/asl-recognition-with-deep-learning

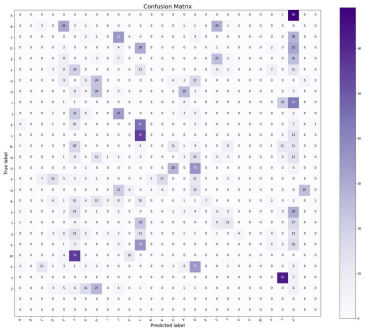
[8]notebook - kaggle : https ://www.kaggle.com/code/namanmanchanda/asl-detection-99-accuracy

[9]notebbok - kaggle : https ://www.kaggle.com/code/danrasband/w207-final-project

X ANNEX

X.1 ANNEX 1

Result with the best model and the third dataset.



	precision	recall	f1-score	support
0	0.00	0.00	0.00	70
1	0.00	0.00	0.00	70
2	0.00	0.00	0.00	70
3	0.00	0.00	0.00	70
4	0.09	0.11	0.10	70
5	0.10	0.29	0.14	70
6	0.18	0.13	0.15	70
7	0.22	0.34	0.27	70
8	0.00	0.00	0.00	70
9	0.27	0.47	0.34	70
10	0.00	0.00	0.00	70
11	0.24	0.79	0.36	70
12	0.00	0.00	0.00	70
13	0.00	0.00	0.00	70
14	0.61	0.40	0.48	70
15	0.00	0.00	0.00	70
16	0.09	0.19	0.12	70
17	1.00	0.10	0.18	70
18	0.07	0.09	0.08	70
19	0.31	0.17	0.22	65
20	1.00	0.06	0.11	70
21	0.00	0.00	0.00	70
22	0.00	0.00	0.00	70
23	0.00	0.00	0.00	70
24	0.58	0.90	0.71	70
25	0.02	0.09	0.03	70
26	0.00	0.00	0.00	0
27	0.00	0.00	0.00	0
accuracy			0.16	1815
macro avg	0.17	0.15	0.12	1815
weighted avg	0.18	0.16	0.13	1815

FIGURE 14 – confusion matrix and data report

We can see that the results are not as good but they are similar to those we had with the first experimentation. We also have results that are better than the random 16% against 3.5% if it was random. So it's encouraging and by doing data augmentation and then adding this set we could make the model more robust.