MASTER IN COMPUTER SCIENCE PROFESSIONAL FOCUS IN DATA SCIENCE

Design and Testing of an Architecture for Deep Learning Experiments Applied to Sign Language Recognition

Fink, Jérôme

Award date:
2019

Awarding institution:
University of Namur

Link to publication

Université de Namur
Faculté d'informatique
Année académique 2018–2019

Design and Testing of an Architecture for Deep Learning Experiments Applied to Sign Language Recognition

Jérôme Fink

Maître de stage : Laurence Meurant

Promoteur : ________________________ (Signature pour approbation du dépôt - REE art. 40)
Anthony Cleve

Co-promoteur : Benoît Frénay

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques.

Abstract

These years, a field of study derived from artificial intelligence makes a lot of noise. It is the deep learning. These methods revolutionizes various domains due to its robustness when treating unstructured data such as images, sounds or video. Applying those new methods to sign language recognition could be valuable for the integration of deaf or hard of hearing people in our societies. To do that, the laboratories of french Belgian sign language (LSFB-lab) shared its expertise and provided us an important corpus of sign language.

This work aims to identify all the methods useful for solving this task and test them. The output of this research is a software architecture facilitating the setup of experimentation and the reuse of their components in order to create more easily other experimentation. This architecture may be used in the future to build and compare the various methods identified.

Keywords : sign-language - deep learning - video recognition

Résumé

Depuis quelques années, une discipline liée à l'intelligence artificielle fait beaucoup parler d'elle : Il s'agit du deep learning. Cette méthode a révolutionné de nombreux domaines grâce à sa robustesse quant au traitement de données non structurées tel que les images, le son ou la vidéo. Appliquer ces nouvelles méthodes à la reconnaissance de la langue des signes pourrait s'avérer utiles pour l'intégration des personnes sourdes ou malentendantes dans notre société. Pour ce faire, nous bénéficions de l'expertise du laboratoire de la langue des signes de Belgique francophone (LSFB-lab) qui a constitué un important corpus au cours des dernières années.

L'objectif de ce travail est de recenser les méthodes pouvant aider à accomplir cette tâche et de les tester. Cette recherche a mené à la conception d'une architecture logicielle permettant de plus facilement mettre en place des expérimentations et réutiliser les divers composants de celle-ci afin d'en créer de nouvelles plus facilement. Cette méthodologie pourra être appliquée lors de la suite de ce travail afin de tester et comparer diverses approches.

Mots-clés : Langage des signes - deep learning - reconnaissance vidéo

Acknowledgments

First and foremost I would like to express my special thanks to my professor Anthony Cleve who gave me the opportunity to work on this wonderful subject and who dedicated of his time in order to make this work a success.

I also thank professor Benoît Frénay for his availability, his helpful ideas and the time he was keen to give in order to make this work more precise.

Thanks to Laurence Meurant and the staff of the LSFB-lab for their precious insights.

Special thanks to Antoine Clarinval who had the kindness to read and correct this work.

Thanks the all the staff of the computer science faculty who welcomed me in their institution and made my internship an outstanding experience.

And last but certainly not least, thanks to my parents Luc and Viviane Fink. They supported me during 25 years and they offered me the luxury of education in the field and university of my choice.

Contents

Chapter 1

Introduction

For people with impaired hearing and speech, sign language is the primary means of communication [36]. However, the sign language is not a common skill in our society. Thus, interpreters are often needed during medical appointments, legal procedures or education. The development of a software able to translate sign language to text could empower the disabled people and increase their autonomy.

A translation algorithm is also necessary in order to create a dictionary allowing to search a word based on a video of the associated sign. The conception of such a dictionary could be critical for deaf born children for learning to read and write.

Since 2013, the laboratory of French Belgian Sign Language (LSFB-lab) based at the University of Namur decided to record a large amount of conversations in order to create a corpus useful to analyze the language and its usage. They recorded 150 hours of videos. 15 hours of them have been annotated: the annotation process is really slow and the amounts of people skilled enough to perform this task is very low. The LSFB-lab hopes that the creation of a translation software could help them to annotate their corpus.

This work is the continuation of the master thesis elaborated by Jeremy Lebutte and Anne Smal [29] in 2017. In their work they investigated the feasibility of a piece of software able to translate sign language automatically. They developed a solution and tested it on the LSFB corpus. By analyzing all the errors made by their software they were able to highlight some issues in the video captured and to propose some improvements for the capture of future videos in order to ease the creation of a recognition algorithm.

The current work goes further by investigating the new methods developed in the past few years for video recognition due to the popularization of machine learning techniques. The main contribution is the development of a framework allowing us to setup and evaluate a method with few code modification in order to quickly perform experiments with various different methods. This work is divided into four parts :

The first part presents the advantages of deep learning in our case and describes all the methods developed in the field of video recognition.

The second focuses on the contributions made. The requirements of the tools needed by the LSFB-lab are presented and the infrastructure needed in order to support the creation of such a software is discussed.

The third chapter shows a concrete implementation of the proposed framework and the first results thus obtained.

Finally, the last chapter provides insights about the future work that needs to be done in order to build a reliable algorithm for sign language recognition.

1.1 Machine Learning Background

When a developer wants to design an algorithm his first task is to identify the issues the algorithm should solve and then acquire the necessary knowledge to complete the task. Problems could be as simple as sorting a list of numbers or computing the distance between two points. It could also be more complex such as recommending products to customers based on their previous purchases or making sure that all the systems of a rocket are ready for launch.

Even if those tasks are different, the creation process of algorithms solving them is always similar. The engineer has to identify all the steps and conditions required in order to complete the task and transcribe them into source code. In order to make an algorithm able to distinguish apples from oranges or bananas, a developer could have written the following code :

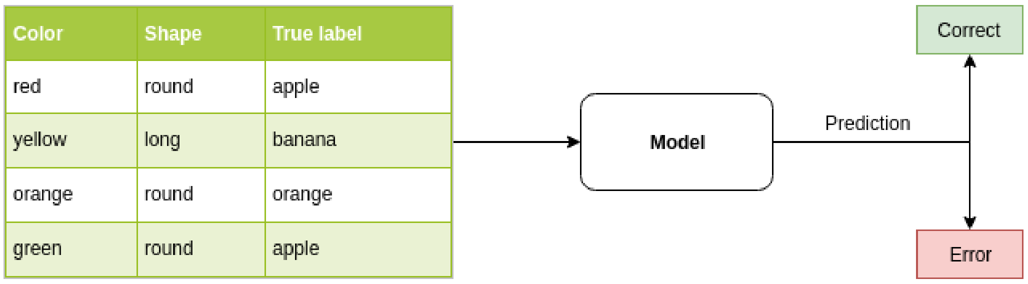
```
If color is green:
    result = "apple"

If color is orange:
    result = "orange"

If color is yellow:
    result = "yellow"
```

This simple algorithm may be enough to complete the task. But, sometimes, apples can be red, if we want to handle this case we need to add another condition in our code. Another annoying case is a green unripe banana. Currently, the algorithm will classify it as an apple. To handle this case a condition looking at the shape of the fruit must be added, complicating drastically our algorithm.

For very complex tasks, identifying and writing all the possibilities is so difficult that it became more costly to design and maintain the algorithms than doing its process by hand.

Machine learning approaches are different. As indicated by the name the purpose of those methods is to let the computer learn the rules and conditions required to complete a task. To do that a lot of examples must be provided to the machine. Showing the examples to the machine is called the *training* of the algorithm. The resulting representation of the problem constructed by the machine is called a *model*. During this phase the algorithm will try to predict the name of the fruit based on the information given. By comparing the predicted output and the true label the machine learning algorithm can adjust its model in order to get a behavior leading to less missclassifications.

Fig 1.1 shows an example for our fruit detection algorithm. There are plenty of different



Color	Shape	True label
red	round	apple
yellow	long	banana
orange	round	orange
green	round	apple

Figure 1.1: Training of a machine learning model for fruits recognition based on some criterion

algorithms available and selecting the best algorithm in order to train a correct model is often the key to success.

Once the model is trained, it will be used in production in a real system where it will be confronted to data that were not present in the training set. It should be able to handle them as efficiently as the training data. This robustness to new data is called the *model generalization*.

The model could be in three different state :

- **Underfitting** : The representation of the model is too simple and the model will not generalize to new data. This state happen in the early step of the training when the model have not seen enough data or if the data he saw was not representative of the whole set. It could also mean that the chosen machine learning algorithm is *too simple* for the manipulated data.

- **Good fit** : The model performs well and can generalize to new data.

- **Overfitting** : The representation of the model is too complex, this state happen when the model trained too much on the same data. It can perfectly classify data from the training set but its performance drops when confronted to other data. This could also be the chosen machine learning algorithm. The number of parameters to optimize could be too high making the model *too complex* for the data.

A graphical representation of these states is shown in fig 1.2. In order to detect and prevent these phenomena the data available for creating a machine learning algorithm are divided into two set. The training set, which will be used during the training of the model, and the test set, which will be used in order to evaluate the model on data it never encountered during its training. Comparing the two could help us identify the moment when the model reaches a perfect fit. Over or under fitting cases should be examined in order to determine the cause of this behavior.

The quality of a machine learning model is highly dependent of the quantity and diversity of data used for the training. These data should be representative of what the model will handle during its time in production and the quality of the data and their annotations is the corner stone of a good model.

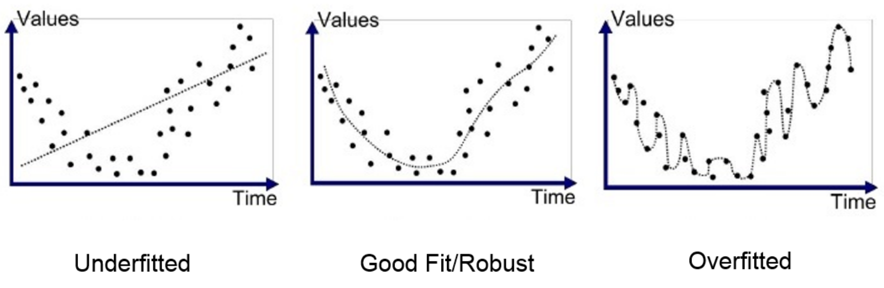


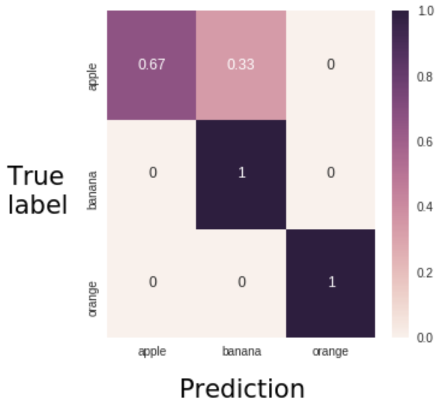
Figure 1.2: Graphical representation of the over and under fitting state and the good fit reproduced from https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76

1.2 Results Visualisation

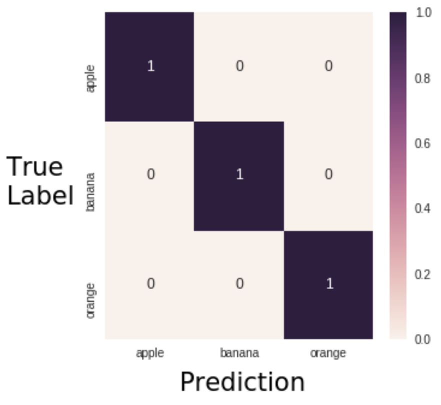
This section aims to explain the visualisation used in this work to evaluate the machine learning models developed. There are two of them. The first one is the confusion matrix, it is a classical way to show the prediction of a model. The other visualisation is a plot that shows how the data are distributed in a 2D space. Those visualisations are useful to diagnostic a machine learning model.

1.2.1 Confusion Matrix

A confusion matrix is a visualisation of the test set comparing the true label of the data to the prediction made by the model. Looking at a confusion matrix make it easier to figure out which classes are often miss-classified. Fig 1.3 show examples of confusion matrix for our machine learning model classifying bananas, oranges, and apples.



(a) InceptionV3



(b) VGG16

Figure 1.3: The image on the left is an imperfect confusion matrix. The image on the right is a perfect confusion matrix

A perfect confusion matrix means that the model always predicts the correct label. It is easily recognizable because only the diagonal of the matrix contains values. This is the case for the right image of the fig 1.3. Miss-classified data are represented by the value outside of the diagonal. In the left image of the fig 1.3, an apple was classified as a banana. This information is valuable when designing machine learning models.

1.2.2 Plotting the Data

The various parameters of a dataset could be displayed on a 2D scatter plot. This visualisation could show us if the data are already separated or if some of them are mixed. Mixed data requires a pre-processing or more powerful model to be classify correctly.

Fig 1.4 shows a scatter plot of the pedagogical dataset *iris*. This dataset contains information about three different variety of iris flower. Each variety is associated with a colour. The blue one is distinguishable from the other two. The green and orange one are mixed at some points, making them difficult to predict.

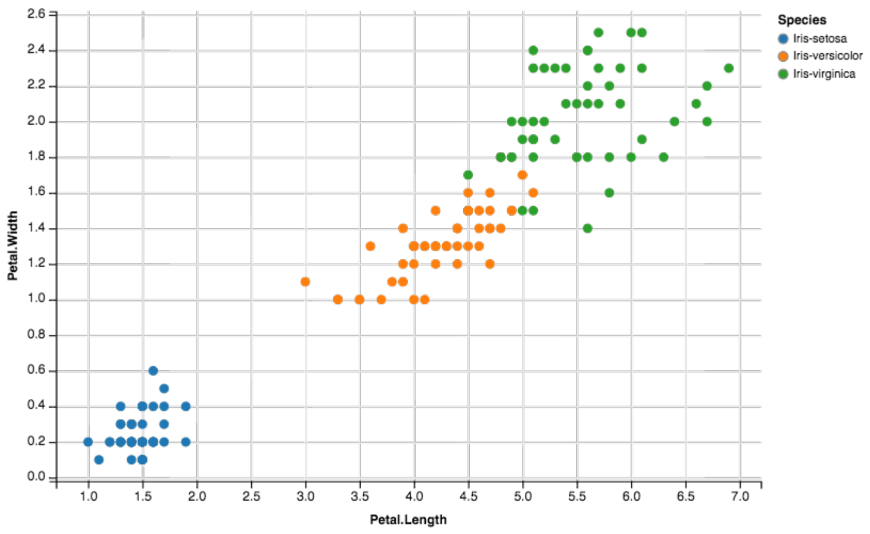


Figure 1.4: 2D plot of the iris dataset. The green and orange points are mixed. This make them difficult to classify. Illustration reproduced from `https://www.datacamp.com/community/tutorials/machine-learning-in-r`

This kind of visualisation helps to determine which classes are more likely to be confused before the training of a machine learning model. In order to plot a dataset with more than two information on a 2D scatter plot, there is a popular dimension reduction called t-sne [33]. The algorithm reduces the dimension of a dataset by making sure that the distances of two points in high dimension are preserved in the 2D representation. Making it a good choice for visualizing datasets.

Chapter 2

State of the Art : Video Recognition

The recent advances in image recognition allowed the automation of a lot of tasks that we thought impossible to solve by a machine. Deep learning algorithms prove to be very good at processing unstructured data, such as text, images and sound. Surprisingly, few works are applying those new deep learning methods to the sign language recognition.

This chapter provides more insight about what is deep learning and why it could be useful to try those methods in our case. An inventory of the methods developed for video recognition will be made and a comparison of those methods will help us to chose which way to go when experimenting those approaches on our sign language dataset.

In our case, the dataset use contains few data compared to the datasets used in other fields of study, that is why we are also going to investigate approaches developed in order to train an algorithm with few data. Those methods are referred to as *one-shot learning.*

We will also discuss all the evaluation metrics that could be use and why it is important to chose it wisely.

2.1 Deep Learning and Image Recognition

The creation of algorithms able to classify images from raw pixels is challenging. The classical approaches for this task was to extract information from the images with techniques such as SIFT [22] which detect keypoints in the image and their location, HoG [7] which use gradient difference in the images to characterize it, etc. Then use these information, called features, to predict the actual content of the images.

In order to compare the results of each methods, a challenge called Imagenet [8] was set up. Imagenet is a dataset made of 14 millions annotated images distributed in more than 22.000 classes. Fig 2.1 show the performance reached by the top-5 algorithms on the Imagenet challenge in the past few years.

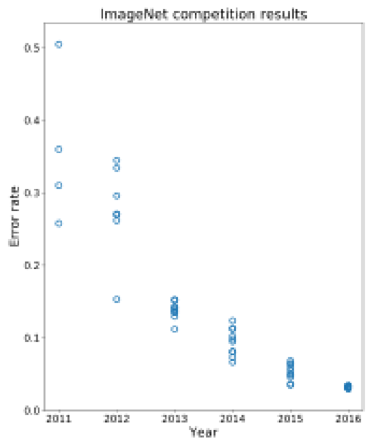


Figure 2.1: Performance for the image net challenge. Each dots represent one method and its score. Image reproduced from `https://en.wikipedia.org/wiki/ImageNet`

We can clearly see, in 2012, that one team beats all the others significantly. It was the first time that deep learning algorithms were successfully used during this competition. No classical methods was able to compete with these kind of algorithm and, since then, all the top team at the Imagenet challenge use deep learning algorithms.

Now the question is how does deep learning methods vary from classical methods and why are they so effective for image classification ?

Designing an algorithm using handcrafted features requires a lot of domain specific knowledge. The features to extract in order to classify a tumor scan as cancerous or benign are not the same used to classify images of cats or dogs. Thus, a lot of effort was put into feature extraction and, despite these efforts, it was easy to miss some important features or to extract irrelevant ones. Doctors may be able to tell if a tumor is cancerous but it is more difficult for them to explain which observation led to this conclusion and it is even more difficult to translate their reasoning into an algorithm.

To avoid these issues, there is a method called *representation learning*. Raw inputs are fed to the representation model and it discovers the best features needed to perform a given classification task. These representations are usually called *embedding*. Deep learning networks perform representation learning at various level [21]. The networks will learn which feature to extract from the raw data. The extracted features will be used to determine the class of the data.

The downside of this method is the computing power and the amount of data needed in order to determine the correct feature to extract. Also, even if we do not have to focus on the feature engineering anymore, a lot of work should be done in order to optimize the deep learning architecture for our problem.

2.1.1 Convolutional Neural Network

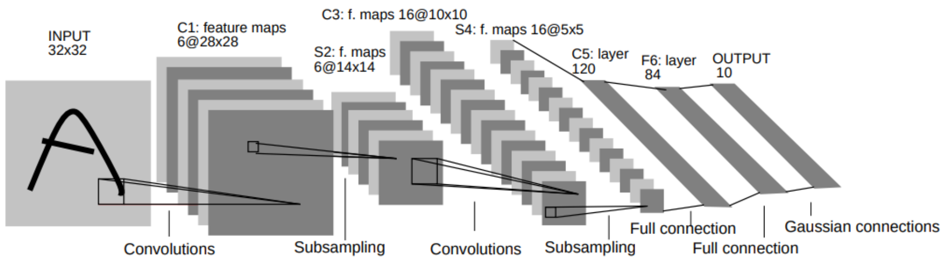


Figure 2.2: Architecture of LeNet proposed by Yann Lecun and copied from [20]

The purpose of a convolution layer is to extract patterns from the images. A sliding window called *kernel* is moved through the images and extract relevant information from it. The application of a kernel on the input pixels is called a *feature map*. The parameters of the kernel determine which feature it will extract. Those parameters are learned during the training phase of the network. The only choices left to the designers of the network are the number of kernels to apply and the size of the kernels. In LeNet, the first Convolution layer consists of 6 kernels having a size of 5*5 pixels. Thus, the output of the first layer of LeNet will give 6 features map having the same size than the input images and containing relevant information extracted by the kernels. During the training phase, each kernel will learn to extract different useful features. But the output is 6 times larger than the input. If we perform directly another convolution, the size of the network will dramatically grow. That is why each convolution layer is followed by a subsampling layer in the LeNet. The *subsampling layer* will resume the information by reducing the resolution of the input images. Doing this will affect the precise position of detected features in the images but the exact position of each feature are not relevant to take a decision [20]. The subsampling layer also use a kernel for reducing the resolution. In LeNet the first subsampling layer use a 2*2 kernel. It means that the value of each pixel of the input images will be averaged in one pixel of the output images.

By stacking multiple layers of convolution and subsampling, the LeNet can learn which features to extract. The network ends with classical fully connected layers to classify the image based on these features. This approach is still the most used in the field of image recognition. The AlexNet network had a similar architecture and modern image recognition model are still made of a succession of convolution layer and Subsample layer (called *Pooling layer*)[28].

2.2 Action Recognition and Sign Language Recognition with Deep Learning

The problems of image recognition were quickly overcome by deep learning methods once the hardware became powerful enough to handle complex convolutional neural networks. Video recognition seems to be a natural extension of image recognition, but the progress made in this field are much slower. Two factors could explain the difficulty gap between image and video recognition

- **Computational cost** : 2D images are much lighter than a video which is a succession of images (called *frames*). Processing video data take inevitably more time than processing an image. A neural network designed for handling 2D images has less parameters to optimize than a network that should handle multiple frames of a video to make a prediction. These two aspects combine made the video data much more cumbersome to manipulate.

- **Temporal aspect** : The order of the event is key information when it comes to interpreting a video. The features extracted by the network should encode the motion in addition to the 2D features characterizing each frame of the video. This constraint makes it more difficult to design a network for video than for images.

To evaluate and train video recognition networks, various datasets have been created. The most popular are :

- **UCF-101**[30] : It consists of 27 hours of videos. 13 000 clips are present in the dataset and 101 classes are represented. It is one of the most popular datasets for benchmarking model.

- **Kinetics**[15] : This dataset is made of 400 classes with 400 examples for each class. The total number of video clips is 160 000 and each clip could have a length of 10 seconds more or less. An extension of this dataset was proposed. It is called Kinetics-600 and contains 600 classes with 600 examples for each[4].

Various approaches have been tried on this kind of datasets their results are used to compare them. The existing architecture for video recognition could be separated into five categories [5] as depicted in Fig 2.3

2.2.1 LSTM-based Approach

To capture temporal information, some network architecture uses LSTM layers. LSTM stand for Long Short-Term Memory and was first introduced in 1997 [12] by Sepp Hochreiter. These layers are useful when processing longs sequence of data. They are used in the various models for speech recognition, writing recognition, etc. Therefore, it is not surprising that researchers are trying to include them in solution tackling the problem of action recognition. LSTM layers can retain information between two evaluations. Fig 2.4 show the flow of data in an LSTM network A is an LSTM cell, X_t the input value and h_t the output of the network. the A layer has an output looping on itself allowing it to conserve a state between two evaluations.

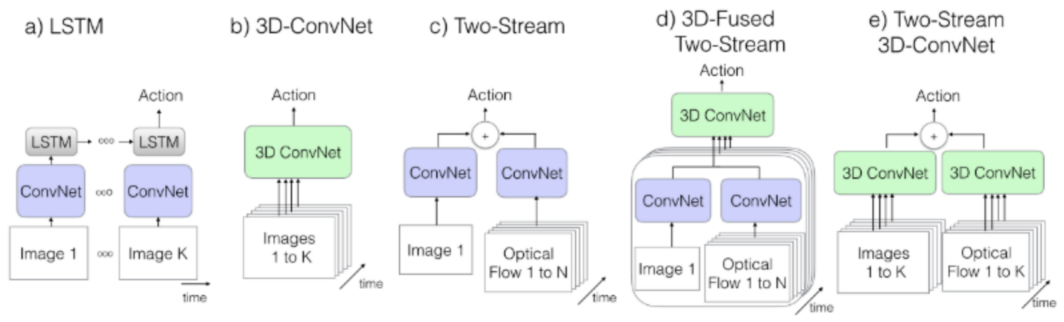


Figure 2.3: Classification proposed by J.Carreira and A.Zisserman reproduced from [5]

This loop allows the network to keep contextual data in a long sequence and, thus, use these contextual data to provide better predictions.

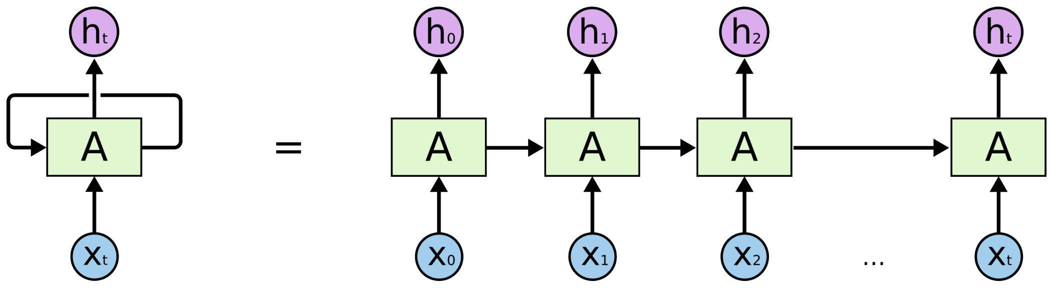


Figure 2.4: Recurrent Neural Network illustration reproduced from colah's blog `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

For action recognition, Donahue and Hendricks[10] proposed an approach using LSTM coupled with classical CNN methods. Each frame of the video passing through a classical CNN network to produce an embedding of the image. This embedding passes through two LSTM layers that will capture the sequence information and produce an output. This method allows us to reuse a trained CNN model from image recognition challenge with pre-trained weight.

The *Long-term Recurrent Convolutional Networks* proposed by Donahue [10] was tested on the UFC-101 dataset and achieved an accuracy of 68.2%.

2.2.2 Two Stream Approach

The idea behind these techniques is to divide the problem into two networks. Each network focuses on a particular aspect of the action identification. The Simonyan and Zisserman approach [27] uses one network to identify spatial features based on few full resolution images and another network inspect the temporal flow of the video based on features extracted during a pre-processing phase. The high-level architecture is shown in fig 2.5. This approach was designed to reduce the computational complexity of the network while conserving enough property of the temporal dimension to classify actions. As mentioned, the temporal stream of the network use features as an input. Those features are obtained by using various classical pre-processing methods extracting different metrics from the raw video. Recently, a team of researchers from Facebook released an architecture called SlowFast Networks [11]. One network called the slow pathway work on full resolution images selected in the video frames at a rate of 1 frame every 15 frames. The fast pathway will proceed more frames, typically 1 frame out of 2, at a lower resolution in order to focus on the temporal features. Only classical convolution layers are used. They will learn during the training which features to extract for classifying videos. The Facebook paper did not use the same metrics than the Simonyan paper for the evaluation of their model. Thus, it is hard to tell which methods are the most effective at the classification task.

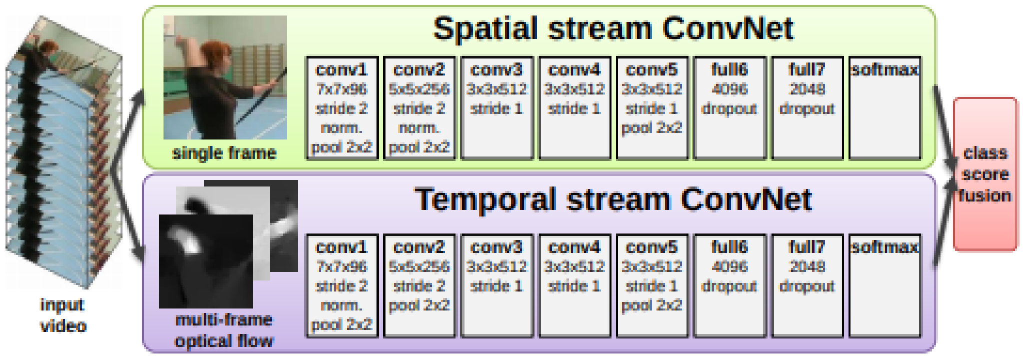


Figure 2.5: Two stream network architecture proposed by Simonyan and Zisserman at NIPS 2014 reproduced from [27]

The accuracy obtained by the *Two-Stream Convolutional Network* proposed by Simonyan [27] is 88% on UCF-101.

The *Slow-Fast Network* proposed by Feichtenhofer [11] was not evaluated on UFC-101. Its accuracy on the Kinetics-400 dataset is 79%.

2.2.3 3D Convnet

The success of deep learning in image recognition is linked to the development of convolutional layers. Thus, when it was time to focus on video recognition, a lot of researcher tries to find a way to design a 3D convolutional layers performing the convolution on the temporal flow too. As mentioned, the naive approach was not efficient and require a lot of computing power. Effort has been made for reducing the complexity of these layers and they were able to achieve competitive performances. In 2015, Du Tran [32] have developed *C3D*: A convolutional layer able to extract information from a series of 2D images and conserving the sequence information during the convolution.

The main advantages of the C3D layer is its versatility. Other techniques describe a whole architecture, this technique is only a layer that could be placed in all the other architectures. Therefore, it is possible to take the 427 layers of Resnet V2 and replace all the 2D convolution by C3D layers. Doing so will have a catastrophic impact on the performances of the Resnet network as C3D layers are inevitably more complex than classical convolution. No benchmark comparing the C3D and classical 2D convolution performance were found.

In a paper from 2017 [5], Carreira and its team made an important discovery. They prove that it was possible to reuse the weights of models trained on 2D images to the same model inflated with C3D. They took the Inception V1 model used for image recognition and replaced all the 2D convolution by C3D. Then, they were able to take the weight of the model trained on imagenet and reuse them in the C3D layer proposed by Tran [32]. This trick makes the inflated Inception V1 model easier to train as it use the *knowledge* acquired when training on imagenet.

It could be useful to add C3D layers in the architecture to improve the performance on video. In term of accuracy performance, a model made of C3D built by Du Tran achieve a 82% accuracy on the UFC-101 dataset [32]. On their side, Carreira achieved an accuracy of 95.6% on UCF-101 by *inflating* pre-trained imagenet weight into C3D layers [5].

2.2.4 Conclusion

Since 2012 the landscape of machine learning applied on complex unstructured data was disrupted by deep learning techniques. They have been very successful in various tasks such as image recognition. More and more works are done in this field in order to solve challenges always more complex.

Thus, a wild range of architecture and approach were developed each of these techniques being more suited for some kind of problem. However, it is often challenging to determine which method will be optimal for our case. A lot of iteration should be performed during the design of the solution before reaching the correct architecture. These iterations are not free. It requires man and computer power in order to design and train the algorithm. The computation power needed in the case of video analysis is not negligible. The difficulty to find correct video dataset and the power needed to get some good results are the two main difficulties in the field of video classification. Reviewing the existing methods enable us to identify the most successful approach. As shown in fig2.6 the Two-stream approach and the inflated C3D are the ones leading to the best results. This approach should be tried in priority when designing the architecture for our problem before considering the others.

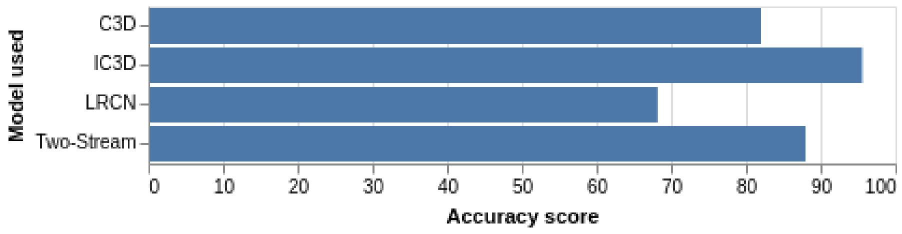


Figure 2.6: Comparison of the accuracy score on the UFC-101 dataset [30]. The model shown on this graph are: The C3D Network developped by Du Tran [32], the Inflated C3D model proposed by Carreira [5], The Long-term Recurrent Convolutional Network from [10] and the two-stream network of Simonyan [28].

2.3 One-Shot Learning Methods

The main limitation of deep-learning methods is the amount of data needed to train a useful network. Data gathering often requires the insight of an expert in the domain. This could lead to extra-cost when designing a solution. In the LSFB context, we have a lot of annotated signs instances but we also have a lot of different classes. Consequently, the number of examples per sign is not huge.

These issues are well known and several methods have been developed to help deep learning network to learn useful features to classify data based on few examples. In this chapter, we are going to focus on relevant one-shot learning methods for deep learning.

2.3.1 Siamese Network

This method was introduced in 1993 by Jane Bromley [3] in the context of handwritten signature recognition. The method developed can learn if two signatures are the same or different based on various information such as the velocity of the pen, raw pixels, etc. The training data is made of genuine signatures and forgeries made by relatives of the original signer or total strangers. They achieved to detect 80% of the forgery with their architecture. The high-level concept of the approach is illustrated by fig 2.7. There are two identical neural networks trained to analyze the signatures passed as an input. The neural networks have linked weight which means that the parameters are shared between the two models. This ensures that each model constructed by the two networks are strictly the same and will react the same way to a given input. The output of the model is an embedding of the signature in a different feature space. In this feature space, we want all the matching signatures to be close to each other. Each set of genuine signatures will create one cluster in the feature space, the forgeries will be outliers in the space. The siamese network learns the feature space during the training phase. The training consists to feed the siamese network with a pair of signatures. The loss function will force the network to maximize the distance between the embedding of two different signatures and minimize the spacing between two genuine signatures from the same person.

Siamese network approach artificially increases the size of the training set by constructing a pair of elements. The fact that the network does not predict a label but position the input element in a feature space allows us to extend more easily the number of classes that the network should be able to discriminate. Another happy side effect of the method is that we can compute the embedding of a signature and store it in a database. The embedding could later be used for comparison with a new signature.

Another paper used a siamese architecture with a deep-learning network[16] to solve a recognition challenge based on the Omniglot Dataset[18]. It contains 1623 characters from 50 alphabets with 20 handwritten examples for each character. The authors reached the human performance level with their approach. They also pointed out that an existing model could be extended to take into account new classes with minimal retraining.

Applied on a more complex task such as face recognition, siamese networks needs to be coupled with transfer learning methods to reuse an existing model to boost their performance.

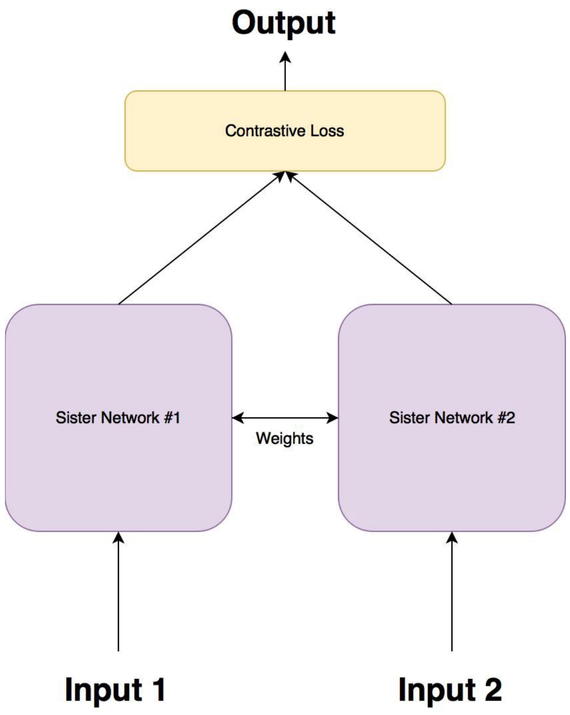


Figure 2.7: The idea behind siamese network reproduced from `https://hackernoon.com/one-shot-learning-with-siamese-networks-in-pytorch-8ddaab10340e`

2.3.2 Triplet Loss

Triplet loss is an extension of the siamese network approach. The finality is the same as the siamese network approach, a triplet loss architecture will build a feature space where it is easier to discriminate the different classes. In a triplet loss architecture, there are three identical networks with shared weight. The approach was first introduced in 2015 [26] to create a feature space for face recognition applications. The network is trained with triplets of input. In the case of face recognition, we will have 3 images. There is the *anchor* image, the *positive* image which represents the same face then the anchor image and finally, the *negative* image which represents a different face. During the training, the loss function will force the algorithm to minimize the distances between the anchor and the positive while maximizing the distances between the anchor and the negative. Thus, each iteration the neural network learn how to differentiate and match faces. The training process is illustrated by Fig 2.8

During the training, a metric is used to determine the performance of the model and to converge to the best solution. This metric is called the Loss function. In the case of triplet loss the loss function is the following[26] :

$$Loss = \sum_{i}^{N}[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha]$$

where f is a function representing the neural network. $f(x)$ gives the embedding computed by the neural network for the input x, x^a is the anchor image, x^p is the positive image and x^n

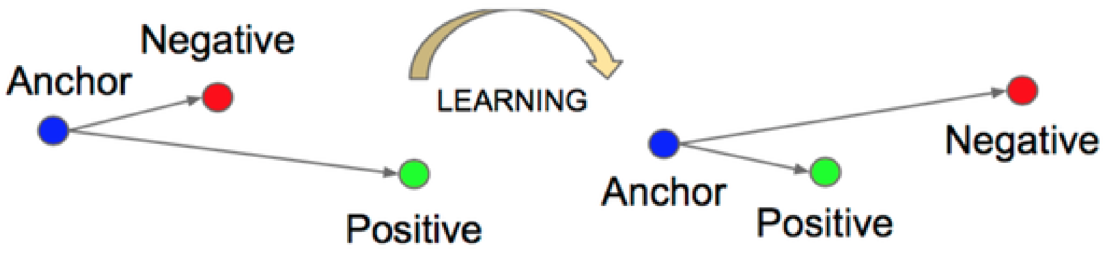


Figure 2.8: Triplet loss training reproduced from Schroff's paper[26]

is the negative image for each triplet i. The α is the margin allowed between the anchor and the positive example. The margin ensures that each class of data will be correctly separated in the embedding space.

To train our model, we need to create triplets of input. But, if the triplets created already respect the constraint forcing the positive example to be closer to the anchor than the negative one, the triplet will not be useful for training the model. If we create our triplets randomly, during the early stage of the training, a lot of them will provide a useful example of negatives instances too close to the anchor. But, as the training progress and the model is getting good at separating some class, more and more of our random triplets will not help to train the model.

A better algorithm for the triplet generation should be designed. This algorithm must use the most advanced version of our model to only keep the triplet that it failed to separate correctly. This will only keep the triplets helping the neural network to learn how to separate the different classes effectively. The triplet selection could be done :

- **After each epoch**: The computation of the triplets occurs each time the model has seen all the data provided in the training set. The model used to compute the triplets is not the most advanced.
- **After each batch**: After each update of the model, the triplets are computed again to benefit from the latest results.

Creating triplets after each batch is the most popular approach and helps the model to converge faster [26][2].

During the conception of their facenet network, Schroff and his team [26] selected 40 faces from the same person and construct pair of positive and anchor based on all the combination of those 40 faces. Then, for each of these (x^a, x^p) duet they select the hardest negative example define by :

$$argmin_{x_i^n} \|f(x_i^a) - f(x_i^n)\|_2^2$$

But, after some experimentation, Schroff concludes that selecting only the hardest negative could lead to premature convergence to local minima. Also the hardest negative is likely to be the result of an error of labelling in the dataset. Finding the $argmin_{x_i^n}$ also require to look at all the negative examples present in the dataset which is time and resource consuming.

To avoid all these issues, it is a better idea to chose the negative example according to this formula [2] :

$$[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha] > 0$$

The formula ensures that the chosen negative example is inside the α margin of the anchor example. Even if the negative is not closer to the anchor than the positive example, the separation is not yet correctly done and requires more training of the algorithm to, hopefully, occurs. Also, the formula allows us to create triplets without having to look at all the negatives examples of our dataset.

Once the triplet model is trained, the evaluation is just a binary problem where the model determines if a duet of input data represent the same or different signs. For each duet of test data, the embedding is computed and each element are compared them to define if they are the same or different. We have to choose a comparison metric and a threshold of distance to classify correctly the duet. Classical metrics found in the literature[26][2] are Euclidean distance or Cosinus Similarity.

Another way to evaluate the model without using is to predict the class of an embedding by associating the evaluated sign to the label of the closest neighbour. If the clusters are well defined, the closest neighbour should be another example of the same sign. By doing this, it is easier to identify which signs are miss labelled.

2.4 Evaluating Models

To compare the models produced, we need to agree on an evaluation metric that will be used for each of our experiments. These metric musts be chosen carefully depending on what is expected from the model and will univocally characterize the *performance* of our solutions.

There is 4 types of metrics enabling us to evaluate a model: The **accuracy**, the **precision**, the **recall** and the **F1 score**. To explain the differences between each of these, we will take look at a dummy example representing the predictions of a model on 100 instances from the test set. These tables are called *confusion matrix* 2.1. The diagonal of the matrix represent the correct predictions. The other cells are the ones containing the mistakes committed by the model.

		Predicted value	
		negative	positive
Actual value	negative	98	0
	positive	1	1

Table 2.1: Result of the evaluation of a dummy model on the test set inspired by [13]

To compute the **accuracy score** of this model it is straightforward. The formula of the accuracy score is [13] :

$$accuracy = \frac{\#Correctly\ classified\ instances}{\#Total\ instances}$$

in our case, the accuracy of the model is 99%. The accuracy is not the perfect metric for all the cases. If the negative and positive classes are not balanced, like in our dummy example, or if the positives label is more critical to detect than the negatives one. For instance, if the model is aimed to detect a deadly virus, you do not want the model to miss a positive example. The other metrics provide solutions for each of these problems. To clarify the following formulas, a name is given a to all the cells of the confusion matrix:

		Predicted value	
		negative	positive
Actual value	negative	True Negative	False Positive
	positive	False Negative	True Positive

Table 2.2: Confusion matrix nomenclature

First, we will talk about the **Precision score**. It is computed with the following formula[13] :

$$precision = \frac{\#True\ positive}{\#False\ positive + \#True\ positive}$$

This metric focuses only on the miss-classified negative. It is a perfect metric if the cost of a *false positive* is high. A classical example is spam detection where a false positive is an important email that was labeled as a spam. In our dummy example, the precision score is 100%

The **Recall score** is computed with this formula :

$$recall = \frac{\#True\ positive}{\#False\ negative + \#True\ positive}$$

Here we are calculating how many positive instances were detected by our model. This is typically the kind of metrics used if it is critical to identify a positive instance. This is the perfect metric for medical exams. In our dummy model, the Recall score is 50%.

The last traditional metric is the **F1 score**. It is a mix between the recall and the precision. On the website of *scikit learn* the formula is the following :

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

By applying this formula to the dummy example, we obtain an F1 score of 67%. The main advantage of the F1 score compared to the accuracy is its robustness when the classes are not balanced. As we see, in our dummy example we had a lot more negative than positive. The accuracy value gives us a 99% accuracy even if the model had only correctly classified one positive. The F1 score is more reliable in this case as it penalises more the poor performance of the model on the positive class. All these formulas can be extended for a multi-class problem.

Each video of a sign is associated with one and only one label. It is possible to use those classical metrics during future experiments run on them. The most useful metrics for the sign language case are the *accuracy* when experimenting on a balanced dataset or the *F1* score.

2.5 Conclusion

Action recognition in video is still considered to be a difficult challenge, mainly due to the computational complexity of the task. Many methods were developed and none of them are significantly better than the others, we will have to try them to see which ones are the best in our case.

Another difficulty is the small number of examples available in the LSFB corpus. Deep learning methods require a lot of data in order to train the model properly. Even if methods were developed to train models with few data they does not converge quickly making them time and resource consuming.

Chapter 3

Sign Language Datasets

Leveraging the state of the art algorithms for video processing requires a lot of data. The LSFB-lab was able to provide a dataset made of 15 hours of annotated video. To determine if their dataset is the most suited for the task, a list of all the open sign language dataset is made in this chapter.

The possibility to merge multiple sign language datasets into one is also investigated.

3.1 LSFB Dataset

Since 2015, the LSFB-lab of the University of Namur builds a corpus for the French Belgian sign Language (LSFB)[23].

The LSFB corpus is made of two kinds of data :

- *Single sign videos* : Videos containing only one sign perform from, and to, a neutral position. Those signs are performed by 3 different signers in front of a green screen. There is only one example for each sign of the dataset.
- *Conversational videos* : Videos of a conversation between three people. Two people tell a story in sign language and the third one moderates the conversation. Three cameras capture each of the speaker's moves.

The single sign video dataset is made of signs who are cleanly performed. Unfortunately, there is not enough example for a deep learning approach. The Fig 3.1 shows samples from the single sign dataset. The speakers are starting from a static position, performing the sign and get back to the static position.

We looked at the conversational video data to enrich the single sign video. There are 150 hours of video picturing 100 different speakers. 10 hours out of the 150 are annotated. The annotation contains the starting time of a sign in the video, the ending time and the label associated with the performed sign. This format allows us to easily retrieve signs from the conversational video to use them in our classification algorithm. The downside of this dataset is its quality. The signs are captured in a conversation, therefore, the starting position of the sign depends on the previous sign performed. Also, some speakers make different signs

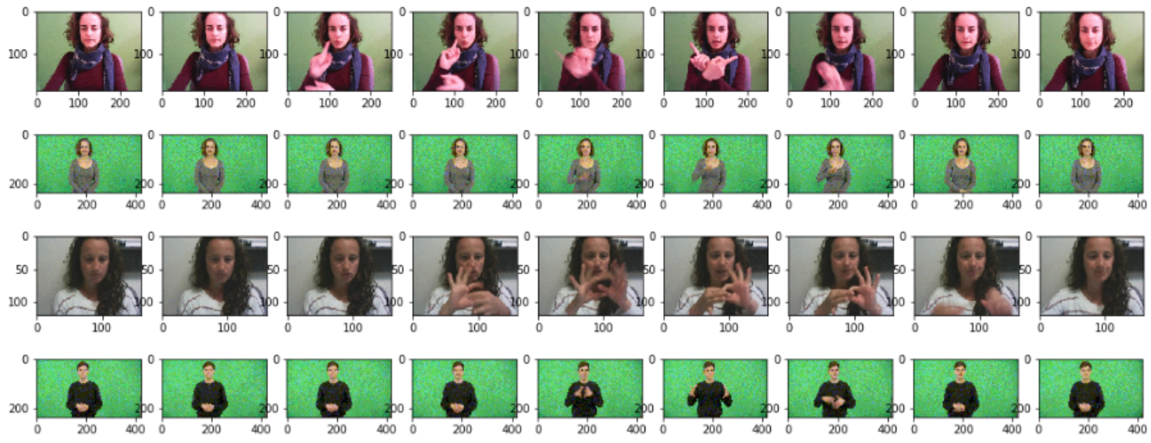


Figure 3.1: Sample of sign from the single sign video. Each row depict one sign being performed.

to express the same idea depending on their region and their age. Each rows of Fig 3.2 shows speaker performing the sign for *give*. Despite that, there is a lot of examples for the common signs and the variety of speakers makes us confident about our ability to identify signs using modern techniques. Also, the fact that signs are performed naturally way during a conversation will allow the construction of an algorithm that could be used naturally by sign language speakers and that will be robust to the little variation brought by the performer of the sign.

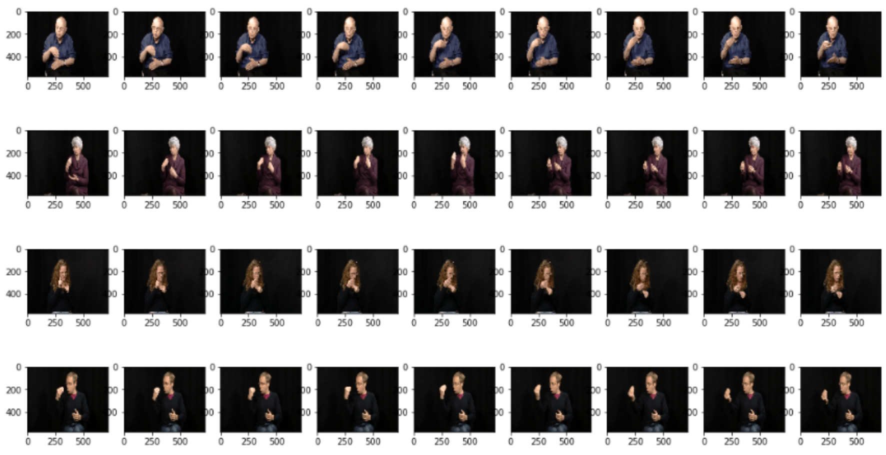


Figure 3.2: Sample from the conversational dataset. All the speaker perform the word *give*.

As explained before, the file format used to annotate conversational video is really easy to exploit and well documented. But, after the extraction, we also need to clean the data we are going to manipulate. During a conversation, there are three kinds of signs that can be encountered.

- **Classic sign** : Signs with a known meaning
- **Descriptive sign**: Signs or set of signs with no particular meaning use to mimic a

complex situation. They are used when there is no classic sign available to describe the object or the concept or when the signer wants to mimic a visual situation like walking in front of a car, etc.

- **Proper noun signs** : In order to express the name of a person, countries, and cities. In order to do that they reuse classic signs and associate them with entities. For instance, the city of Bastogne in Belgium is represented by the sign meaning "star" due to its famous monument *The Mardasson Memorial*. The understanding of this kind of sign relies highly upon the context of the conversation and the knowledge of the interlocutor.

In this work, we will only try to translate classic signs with fix and known translation. The descriptive signs will be discarded, the proper noun signs could be used to enrich the classic sign data. It is possible to use all the *"star"* signs used to say *Bastogne* to have more occurrences of this sign in the corpus.

After this processing we obtain the following statistics.

- 4660 signs with, at least, 1 occurrence
- 627 signs with more than 10 occurrences
- 52 different signers

3.2 Other Datasets

To choose the best dataset to train our model, All the available datasets should be considered. The finality of the LSFB project is to provide an application allowing hearing impaired to search the signification of a sign by using only the webcam on their laptop. Thus, we voluntarily discard datasets using additional capture methods such as colored gloves or hand sensors. The datasets captured with the Microsoft Kinect, a camera able to capture the depth of the filmed objects, are kept as we can use only the regular video from the Kinect and discard the depth video associated. The table 3.1 summarize the content of the biggest sign language datasets.

It is quite hard to obtain true numbers about the content of each dataset. Some of them lack statistics about there contents but, overall, the LSFB dataset is quite good compared to the others. The Chinese dataset has more sign with more than 10 examples but the speakers are less diversified. It could be interesting to use it to train a model or to test if transfer learning could work in the SLR field.

The reader could also wonder why we didn't mix the various dataset to create a single big dataset for the training. The reason is that sign languages are local construction and there is no such thing as an international sign language. Each country have its sign language and the same sign could have a different meaning depending on the country. If we mix the dataset, there is a high risk to have examples of the same sign associated with different labels. It could be possible to discard these kinds of signs to keep only the original sign of each Sign Language over the world but the cost of doing so will be greater than the gain.

Description	**Sample size**	**Different signs**	**Examples per sign**	**Number of signers**
DGS Kinect 40[24] German sign language	2800	40	70	14
American sign language lexicon[1]	3800	3000	at least 1	2
Auslan Signbank	7797	unknown	unknown	100
DEVISGN[34] Chinese Sign Language dataset	24 000	2000	12	8
Signum[17] German Sign Language	unknown	450	unknown	25

Table 3.1: Content of the biggest sign languages dataset

3.3 Conclusion

The sign language corpus provided by the LSFB-lab is one of the biggest datasets available currently. It is the perfect candidate for testing and comparing the various models produced by our experiments. Despite that, the amount of data available is low. This will make the creation of a robust model challenging.

Chapter 4

Solution Design

This chapter focuses on explaining the needs of the LSFB-lab. Identifying which software will use our *intelligent system* will allow us to determine which type of data will be provided as an input to our system.

There is a lot of different methods for video recognition. Choosing the best one for the LSFB-lab case will require to perform a lot of tests on their data. To perform them efficiently, an architecture enabling us to run experiments quickly and keep track of the results must be designed.

Finally, finding a platform where experiments can be run is one of the critical aspects of the future of this work. Video datasets take a lot of disk space and it requires a lot of time and computing power to apply a pre-processing to each video or to train an algorithm on them. It is hard to find good enough hardware to run such heavy processes. A section discusses the existing platforms providing such hardware with their advantages, and weaknesses.

4.1 Requirements Analysis

This section aims to identify and present the needs of the LSFB-lab. An algorithm able to translate sign language into text could be useful in the various tools of the laboratory and knowing which product will use it could give us insight about the type and shape of data the algorithm will receive once in production.

4.1.1 The Annotation Process

The LSFB lab gathered 150 hours of video involving 100 signers to study the French Sign Language of Belgium. Currently, only 10 hours of these videos have been annotated. This may seem like a small amount but knowing that, on average, one sign takes less than a second to be performed. 10 hours of video contains already a huge amount of signs and are tedious to annotate.

The videos are recorded in a studio owned by the LSFB-lab. There is two cameras, one for each interlocutor. They are placed to capture all the upper body of each speaker. They did not use 3D cameras able to capture the depth information and they did not equip the speakers with gloves for recording additional movement information.

To annotate the video, the LSFB-lab staff uses a software called ELAN. This software allows them to indicate the starting time and ending time of each sign, those isolated signs are associated with a label called a gloss. Each gloss is unique for a particular sign and represents roughly its french translation. All this information is stored in an XML file.

The annotation process requires two team members. The first one will annotate the raw video, the second one will check that all the signs were labeled correctly. To be more efficient, the LSFB-lab would be interested in a solution capable to replace the first team member. Annotating sign video requires the uncommon skills of being fluent in the two language[29]. There is few people able to translate the video of the LSFB-lab slowing the process of annotating the video gathered.

In summary, the high-level requirements should be :

- The algorithm must be able to determine the start and end of a sign,
- The algorithm must associate all the isolated sign with the correct gloss,
- The algorithm must use images from a classical camera,
- The speakers should not be expected to follow a dress code or wear particular gloves

The algorithm will not work alone on the annotation process. A team member of the LSFB-lab will still perform a check of the annotated video to correct the mistakes left by the algorithm. Thus, even if the algorithm is only able to annotate 60% of the signs correctly it will still be useful.

4.1.2 The Corpus Website

The LSFB corpus is not just used by the LSFB-lab to analyze the language. It is also a public corpus available on a website enabling anyone to search for a translation from French to sign language. This website is a unique tool helping people to learn more about the French Sign Language of Belgium but, on the other hand, the impaired users could not use the website intuitively. To find the French word associated with a sign, they have to search it by gloss but those gloss were arbitrarily chosen by the team of the LSFB-lab during the annotation phase. To query more naturally the website, the more natural option will be to let the user perform the sign in front of a webcam.

Every modern laptop is equipped with a webcam. Using this capture method is the best way to bring all the functionalities for as many people as possible.

In this case, we can only focus on the top-5 accuracy of the model. It means that, when evaluating the model, we will ask the algorithm to provide us five gloss who are the most likely to be associated with the video of the sign and if the correct gloss is in this list of five gloss we consider that the prediction was correct.

We can use the top-5 accuracy in this case as the five results could be displayed to the user of the dictionary and he can choose the correct one himself.

The system allowing this should respect those high-level requirements :

- The algorithm must associate the correct gloss with the performed sign,
- The algorithm must use raw video from a webcam,
- The user should not be expected to follow a dress code or wear particular gloves,
- The top-5 accuracy could be use in order to evaluate the model,

As we can see, these two cases share a lot of high-level requirements. This encourages us in the idea that the algorithm could be shared by the two use case. The priority of the LSFB-lab is to make this feature work on their website. The rest of this work will focus on the signed search but special attention will be given to the modularity of the solution to maximize the shared elements between the two systems. If the models created could provide good performance in the two tasks of the LSFB-lab it would be a big plus.

4.1.3 Requirements

We will mainly focus on the creation of the algorithm shared by the two use cases of the LSFB-lab. Namely the algorithm that will be able to associate a gloss to a clip showing a single sign performed.

We can extract a more exhaustive list of requirements for this module. The requirements can be divided into two categories :

- **The functional requirements** : Element describing the behavior of the system,
- **The non functional requirements**: Additional constraint put on the behavior

Functional Requirements

The input of the machine learning model will be a video clip showing only a single person in the video frame performing only one sign. The video sequence should start at the beginning of the sign and end when the sign is completed. The upper body of the person should be visible and centered in the video clip. The hand should never go offscreen.

The top-1 accuracy will be used when evaluating the model. Having a good top-1 accuracy is beneficial for the two use cases of the LSFB lab.

The machine learning model should easily be deployed in production and provide interface enabling other systems to query it.

Non functional requirements

The machine learning system should be able to classify a sign captured by a laptop webcam.

The system should be able to provide an answer within 3 seconds in order to keep the navigation on the website as fluid as possible.

Scope of This Work

It is utopian to think that it is possible to create a model able to annotate perfectly signs from raw video in just 3 months. Video recognition is still a challenge in the machine learning community and there is a lot of different methods developed in order to solve this problem. To find which methods is the best for this case, we will need to create and evaluate a lot of different models.

During this internship, the focus will be on the creation of an architecture facilitating the setup and evaluation of various experiments to reduce the time spent between each iteration and to maximize the reusability of the code developed for each experiment.

4.2 Project Architecture

The challenge of this architecture is to make it versatile enough to test various machine learning workflow and train various models. Each element from the experiment pipeline should be easily replaced by another implementation. Fig 4.1 show all the main steps we have to care about to select, train and test the best model for our case.

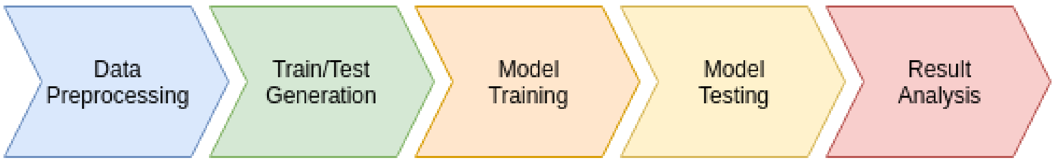


Figure 4.1: The dataflow pipeline used in order to run experiments in Machine Learning project.

- **Data pre-processing** : During this step, the raw data will be transformed to be fed in the model. This step could be very complex and involve a lot of inner steps as it could require to apply various normalization process, filtering the outliers, etc.
- **Train/Test generation**: Once the data are ready to be used, we have to separate them in a test and training set. The separation should be deterministic in order to have the same training and test set for all the experiments. Comparing two models that were not trained and evaluated on the same data could be misleading. As we are working with video the train and test set could not be load entirely in memory. We have to set up a mechanism enabling us to load only the video when needed to avoid to saturate the memory. If we are working with a siamese network or a triplet loss, the train and test set should be composed of duet or triplet correctly formed. The code of this step should handle all these behaviors.
- **Model Training** : The model will receive the training and test data generated and will begin its training phase. There is a lot of variability at this step, we need to be able to change the architecture of the model, to determine how many layers should be frozen in order to make some transfer learning and we also need to determine criterion that will indicate when the model ends his training phase. All these parameters should be easily modified. The model weight and architecture should be saved at the end of the training to continue it later or to load the model again for further evaluations.
- **Model Testing** : The model will be applied to the test data and various metrics will be extracted from the predictions. The test data prediction should be saved in a file in order to investigate them more easily later or to run an additional script on them without having to reuse the model to generate the raw predictions.
- **Result Analysis**: This step will generate various evaluation metrics or visualizations based on the prediction of the model on the test set.

Executing the whole pipeline could be time-consuming, especially in our case were we are working with a lot of videos. The data pre-processing phase alone is expected to last several hours. The pipeline designed should be able to skip some processes if the results have already been computed earlier. The fig 4.2 shows with more details how all the different steps from the classical machine learning pipeline are connected.

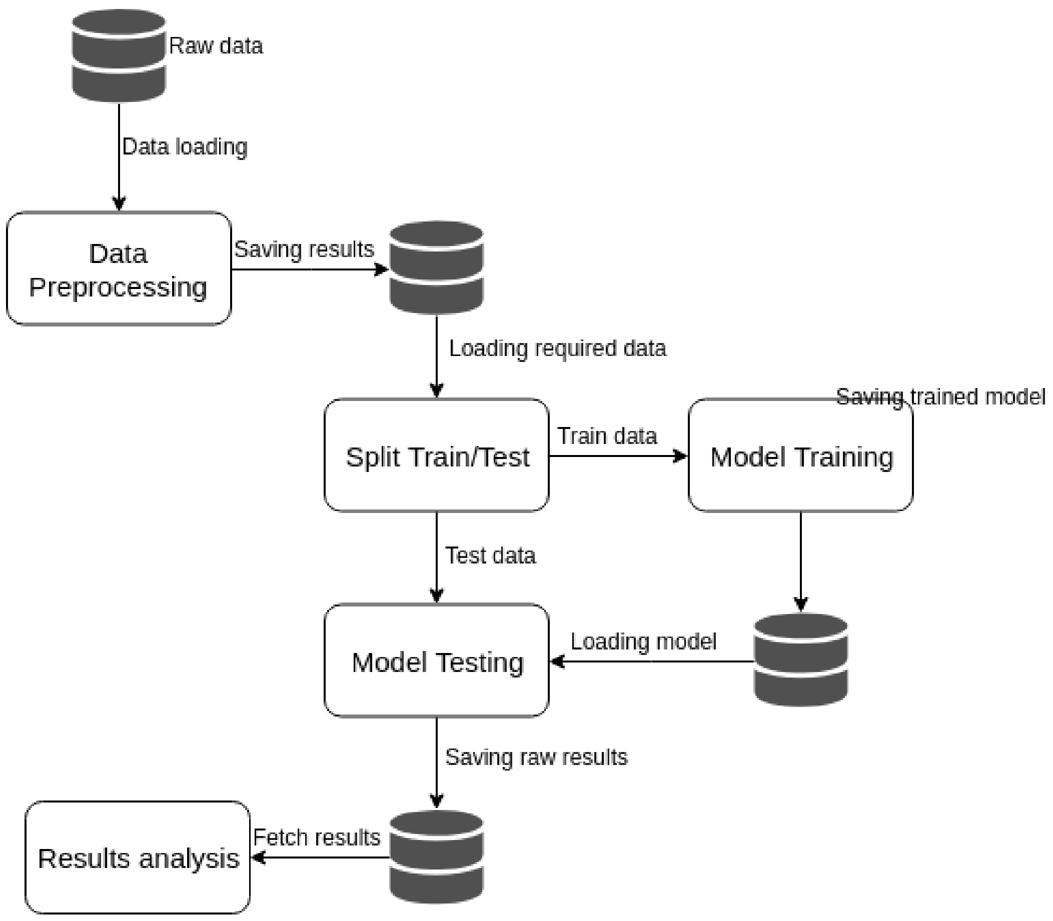


Figure 4.2: Diagram showing how all the different step of a classical Machine Learning pipeline are connected and the checkpoint system that should be provided by the framework.

The result of each step is saved in a database. The saved data could be seen as *checkpoints* for our experiments. The data from the checkpoints could be reused in different experiments or when reproducing a previous one saving us time when running multiple experiments relying on the same pre-processed data.

The data flow depicted in fig 4.2 is the following: The raw data are stored in a database accessible by the pipeline. The data are loaded into the pre-processing module to be cleaned or to extract some information from them. The pre-processing results are saved and could be reused for other experiments saving us precious time. Those pre-processed data are loaded and split into a test set and a training set. The training set is sent to a machine learning algorithm to train a model. The model trained is, then, saved. The test data are sent into the testing module. This module loads the model to test (usually, the one who was trained at the previous step). The raw predictions of the model are saved and could be load by a results analysis module who compute evaluation metrics based on the raw predictions.

The principal output of the pipeline is the model saved by the training module. If the performances of the model are good enough, it can be selected and put in various tool requiring its predictions. The model saved could be copied and reused everywhere.

4.2.1 Strategy Pattern

The machine learning pipeline proposed is made many steps and the behavior of each step should be replaced easily in order to run a different experiment. We need to decouple the behavior of each step.

We can take advantage of the *Strategy* design pattern. This pattern is useful when dealing with classes having multiple choice of algorithms to perform the same behavior.

To explain the concept behind the Strategy pattern we will take the example of a thermometer display. The software receives a raw value from a sensor and must display it in a human-readable format. There are different units possible, therefore, there are different algorithms to transform the raw data into one of the units and display it. All this could be done in the thermometer class like this :

```
class thermometer is
    private chosen_unit

    method display() is
        if chosen_unit == "fahrenheit" then
            self.display_fahrenheit
        else
            self.display_celsius

    method display_fahrenheit() is
        something()

    method display_celsius() is
        something_else()
```

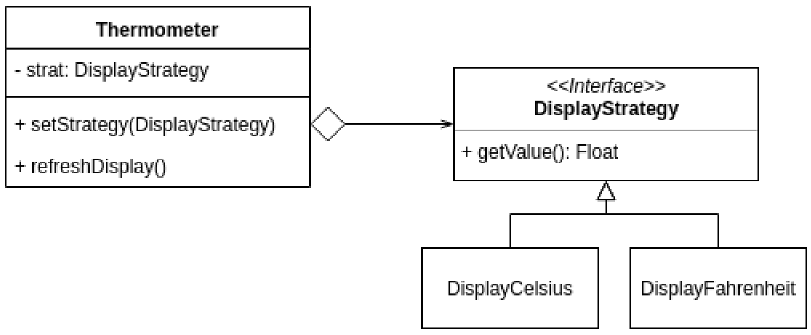


Figure 4.3: UML representation of the strategy pattern with the thermometer example

That will work but as the number of algorithms used to compute the temperature increase, the *display* method will become messier. Also, depending on the complexity of the algorithm the length of the class could significantly increase. Too long classes are often considered to be a bad smell in software engineering. To avoid that the *Strategy* pattern propose to decouple the algorithms and the class using those algorithms. The algorithms will be encapsulated into a *Strategy* class. The user of the thermometer class could select one of those strategies in order to display the temperature in the appropriate unit. Fig 4.3 show the UML hierarchy of the various class involved in this method. It is the role of the user of the class to provide the strategy to use.

In our case, we will have various types of strategy. One for each step of the pipeline. The configuration will provide information about the strategy to use and its parameters. Setting this structure will allow us to simply change the pipeline between experiments.

4.2.2 Builder Pattern

The *strategy* pattern is useful to address the issue of variability inside each step of the pipeline. With it we will be able to change the algorithm used in each step effortlessly.

But our framework allows the user to run just the task he wants and not all of them. The concrete pipeline executed could change from one experiment to another. We need to find a pattern that enables us to modify the pipeline to execute.

The *builder* design pattern could offer us an easy way to address this issue. This pattern enables us to build a complex object step by step. A classical example to illustrate the behavior of this pattern is a house configurator software. Houses are made of various rooms, could have a garden, a pool, etc. Usually, this variability is handled by the constructor of the class but, when the number of parameters is too high, the constructor becomes unreadable for a developer as shown in this pseudo code example

```
class House is
    private number_rooms
    private pool
    private garden
    private sauna
    private number_walls
    private number_windows

    method constructor(nb_rooms, nb_walls, nb_windows, pool,
                        garden, sauna) is
        // Generate the correct house instance



house = new House(4, 4, 6, false, true, false)
```

The call to the constructor is quite long and it is hard to guess the meaning of each parameter without looking to the class declaration. This also lead to the existence of a lot of useless parameters. House having a sauna are quite uncommon but we still have to specify it during the construction of the object

The builder class will handle all the construction of the object and provide a more friendly way to design the object needed by our use case. A simple builder architecture is shown in fig

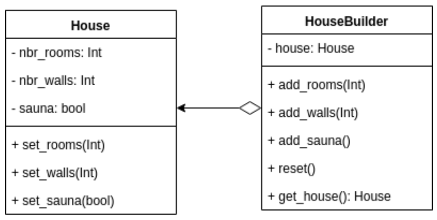


Figure 4.4: Simple builder class handling the construction of a new house instances

The user could now use the *HouseBuilder* class in order to generate a house instance like this :

```
builder = HouseBuilder()

builder.add_rooms(4)
builder.add_walls(4)
house = builder.get_house()
```

The code above is more readable than the version using a constructor for the house. This pattern will be useful to create a custom pipeline containing only the steps needed for our experiments. The different builders could be designed in order to handle the creation of the pipeline for various tasks.

For instance, we could have the *PipelineBuilder* who will configure a pipeline that could be run on a local computer but we could also design a *CloudPipelineBuilder* able to start all the needed resources in the cloud.

4.2.3 Experiment Configuration

One of the key requirements of the framework is to be able to set up quickly an experiment or to quickly reproduce a previous experiment.

To do that, the user will have to describe his experiment in a single file. The framework will follow the description in order to set up and run the correct experiment pipeline.

This file will, therefore, contain all the information needed to fully characterize the pipeline. For each step, the following information should be provided :

- **The input data**: Where the current step should read the data needed to perform its task,
- **The process to execute**: Give the name of a function or a class to use to perform its task,
- **Where to persist the output** : Describe in which format and where to store the result of the process for the current step.

The user could specify that information for all the steps he wants to run during its experiment. He also has to specify some global information such as the *name* of the experiment in order to identify it later more easily.

The actual content and the language used for this configuration file will be highly dependent of the implementation of the conceptual framework and the format required should be well documented. In some advanced use cases, it could be useful to investigate the creation of a *Domain Specific Language* for the configuration of the pipeline but such methods are beyond the scope of this work.

In our implementation, we will reuse a standard format for configuration files such as XML or JSON.

4.2.4 Full architecture

All the useful concepts identified are merged in a single UML diagram 4.5 representing the proposed architecture for our conceptual framework.

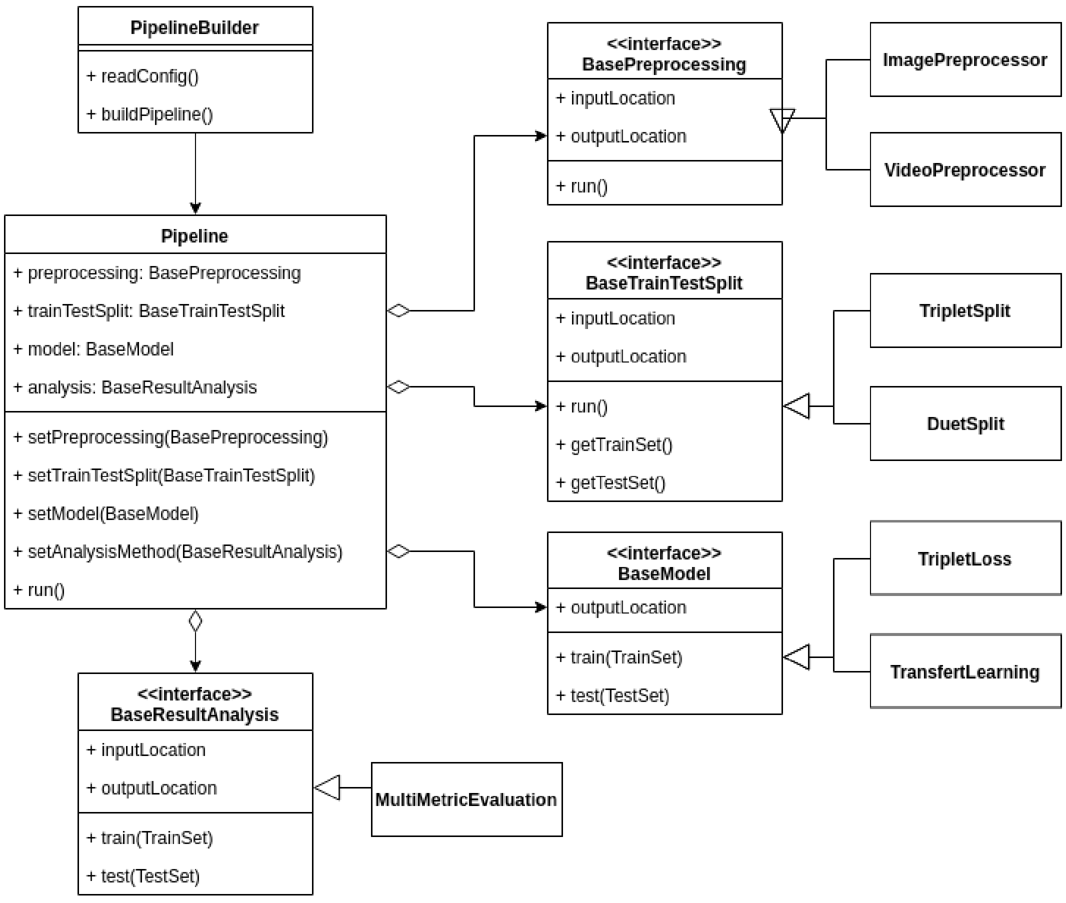


Figure 4.5: Representation of the architecture for the proposed framework

The main elements of this architecture are :

- **PipelineBuilder** : This class is the conductor of the whole system. Its purpose is to read the configuration file provided by the user and to load all the needed classes required for building the correct pipeline. Once the pipeline is constructed its role is over and the pipeline is ready to be executed. In a classical builder pattern, the building steps and parameters are hardcoded in the builder class itself. In our case, these are externalized in the configuration file making the whole process more flexible but, if the configuration file syntax becomes too verbose or is not well documented, it could be more difficult to read and to write.

- **Pipeline** : This class will take care of the execution of the whole process. It should be able to execute the different steps in the correct order and to skip the unnecessary steps if needed.
- Base interfaces : Each step in the machine learning pipeline is represented by an interface providing all methods needed by the pipeline in order to run the experiment. Two steps have been merged in a single class: *model training* and *model testing*. This choice is motivated by the fact that the model uses in the training phase is the same than the one uses during the test phase.

Implementing this architecture require to answer some questions highly dependent of the technologies used by the user and its team. However, once the core of the framework is implemented, it is possible for a team to easily share and create new steps for their experiments. They just have to implement the right interface to make it compatible with the experiment pipelines.

4.3 Infrastructure

To train and test various versions of our machine learning model, we need a server infrastructure powerful and available enough to run quick experiments. One of the first tasks was to find and set-up such an infrastructure.

Training huge deep learning model requires specific infrastructure. Fortunately, the popularity of deep learning encouraged the various providers to adapt their offer to support these kinds of techniques. The price is one of the key factors when choosing a platform, the other one is the quality and power of the GPU. For my task the GPU needs to :

- **Be compatible with the python library Tensorflow 8.0.** This constraint us to use NVidia GPU with a compute capability of 3.5 of higher,
- **Having at least 12 Go of cache.** To load data in the GPU, the memory cache should be consequent when working with video or images,
- **The GPU should always be available for us.** We cannot afford to wait for a GPU to be available. The training is a big bottleneck for this work if we add another one by waiting for the availability of the platform the advancement will be too slow

GPUs are not the only requirement for our infrastructure. Raw video data took a lot of hard drive space. Besides the raw data, we will have to store pre-processed data that will also increase the disk usage. The system should have at least 1.5 To of hard drive.

Having those criteria in mind, there is plenty of option available we will compare them to choose the best for our case.

4.3.1 Online Infrastructure

Recently, leading companies in the IT world developed cloud platform. These platform promise to reduce the IT cost of businesses by providing easy to set up servers in a distant data center. The purpose of this section is not to discuss if they kept their commitments but if their offer for deep learning is good enough for my work.

The Walloon universities of Belgium also set up a cluster of computers to run resource consuming experiments. Each university has a server that is open to every researcher. This cluster of servers is called CECI.

Cloud platforms and the CECI cluster provide us IT support for managing the servers. We do not need extended system administrator knowledge to use them.

We will compare each of these platforms to identify their strengths and weaknesses and their prices.

CECI Cluster

The inter-university cluster is the first platform we investigate because it is free to use for every researcher in the university. The cluster is made of 7 servers hosted by each Walloon university. Only 3 of these 7 servers provide GPU. The GPU provided by the clusters are too old for the Tensorflow version we aimed to use. The cluster is currently not an option for this work.

However, the UMons university will soon upgrade his server with brand new GPUs. Could we migrate the project on their server once it is done?

After further investigation, we conclude that CECI cluster is not an option due to its lack of disk space and the number of jobs who run on the cluster.

Once a job is submitted it enters a queue and have to wait that all the preceding job finishes before being run. A job could be queued for 2 days before being run. This delay between submission and launch is too high to conduct quick experiments in order to chose the most suitable model. The cluster is not made for machine learning project.

Amazon Web Service

This is currently the leader in cloud platforms. AWS is well known and provides a lot of services to help companies to deploy their app, store their data, manage the identity of their employee, etc.

The AWS ecosystem is well documented but the diversity of service could make the platform cumbersome to use. For Machine Learning, three services could be useful :

- **EC2** : Elastic Computing Cloud is a service enabling us to rent server in the cloud. Amazon provides different type of instances with different kind of RAM, CPU, and GPU. The instance could be started at any time and take 2-3 minutes to be set up. The instance is billed per hour of usage. The price depends on the usage and demand. During the configuration of the instance, Amazon proposes an OS made for Deep-Learning with the latest tensorflow installed, all the GPU drivers setup and pre-installed python libraries.
- **S3** : Simple Storage Service is a product enabling permanent storage of data. Amazon takes care of the replication and availability of the, so-called, S3 bucket and promise a 99.99% durability of the file stored. The pricing depends on the amount of data stored and the number of call to the S3 API for retrieving or pushing data.
- **Sagemaker** : Sagemaker is a high-level service enabling to train, evaluate and deploy machine learning models on their cloud platform. Sagemaker could be seen as an abstraction of the EC2 and S3 services, it will allocate S3 space and run EC2 instance to store the raw data and execute the models. The code should respect a given architecture to be executed. Fig 4.6 show the workflow to follow when constructing a sagemaker app. This service also provides a Jupyter-lab environment for experiments.

Quickly, we decided to not use the Sagemaker service. Even if the tool looks well designed and guarantees that our code will be well structured, using the SageMaker way of doing machine learning will trap us on the platform. We do not want to rely heavily on a cloud provider as the code is likely to be moved to another platform or to a local server. But it could be a good idea to take some inspiration from the code architecture of sagemaker to design our Machine Learning pipeline.

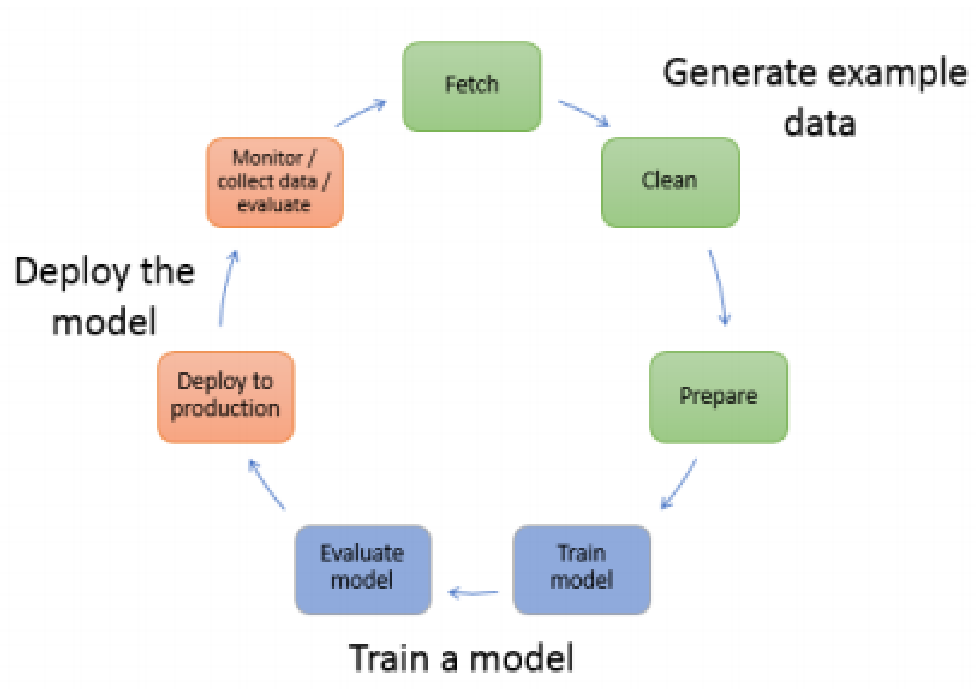


Figure 4.6: The SageMaker workflow imposed by amazon. Reproduced from `http://docs.aws.amazon.com/sagemaker/latest/dg/sagemaker-dg.pdf`

If we use AWS it will be on an EC2 instance. The GPU instance of EC2 provides an NVidia Tesla K80 which is compatible with the latest version of tensorflow and provides 12Go of GPU memory. The price of such an instance is around 1$ per hour.

The pricing of the S3 storage is a bit more obscure as they bill the number of API call without specifying if one call fetching several files cost the same price than a call fetching only one file. The price is set to 0.023$ per Go stored and 0.02$ per API call. With wise use of the space and data transfer, the cost could stay reasonably low.

Google Cloud

Google cloud is one of the main competitors of AWS. Their offer is comparable. The platform is well documented. It proposes less diverse services than AWS the platform is then much more readable and it is easier to spot the services useful for our case. Developers with really specific needs will not find the perfect service for their case and will prefer AWS. For our case, google cloud propose these services :

- **Compute Engine** : This service is comparable to Amazon EC2. It allows us to start a custom server. The OS installed on them does not provide any by default support for tensorflow and python. We will need to install them once the instance is started. Google allows us to add NVidia Tesla K80 to the instance.
- **Cloud Storage** : This is the equivalent to Amazon S3.
- **ML Engine** : This service provides a server pre-configured to run machine learning

model and a command line too to push the model on the server. The training process is described in a configuration file that will be read by the server in order to perform the training. A dashboard allows us to monitor the progression of the model. Once again the code needs to be formatted in a particular way if we want to use this service at full potential. It is also possible to run jupyter notebooks with that service.

- **TPU unit** : Tensor Processing Unit is custom GPU developed by google. They are optimized for tensor computation which is used in for deep learning algorithms. Their speed is 15 to 30 times faster than classical GPU[14].

Google Cloud Platform is very close to the AWS offers. Their compute engines are a bit cheaper. A GPU instance cost, on average, 0.70$ per hour. And the storage is a little bit more expensive from 0.026$ to 0.036$ per Go.

In my opinion, Google services need a bit of work after being launched to accomplish their task. You need to install python and all the needed libraries. On the Amazon platform, a lot of software are pre-installed allowing us to directly use their services *out of the box*.

The TPUs are still in bêta and the pricing is secret. To access them you need to negotiate with their sales. If one day they become available directly from the Compute Engine configurator at a reasonable price it could be interesting to try them.

Floydhub

Floydhub is branded as the Github of Machine Learning. Their platform provides a git server, jupyter notebooks and their system support all the main python deep learning library. To use their GPUs we have to subscribe to their platform for 9$ per month. This gives access to 7 days of NVidia Tesla K80 per month. It is also possible to rent GPUs for 1.20$ per hour. The big limitation of the platform is the storage. It is not possible to store more than 100Go of data.

Google Colab

Google Colab is a free jupyter notebooks environment running on GPU. Data could be fetched from a google drive account. This platform is made for researchers and runs on the unused GPU of the Google cloud platform. If there is no GPU available the model train on a standard CPU. The availability of this platform is very low and the training could be cut at any moment.

4.3.2 offline GPU

Another option is to buy some GPU for the faculty of computer science and to use them freely. But will this solution be cheaper than renting GPU in the cloud? As we have seen, the most popular GPU for deep learning is the NVidia Tesla K80. But this product is not produced anymore in favor of a new generation architecture. Also, GPUs from the *Tesla* family are expensive GPU designed for data centers.

For private usage, there is another GPU developed by NVidia: the *RTX 2080 Ti*. The RTX architecture is optimized for matrix calculation making them a perfect choice for Machine Learning development. Fig 4.7 shows a benchmark of various NVidia GPU on various deep learning architecture types. The 2080 Ti is one of the top GPU available. The price of this card is around 1000$.

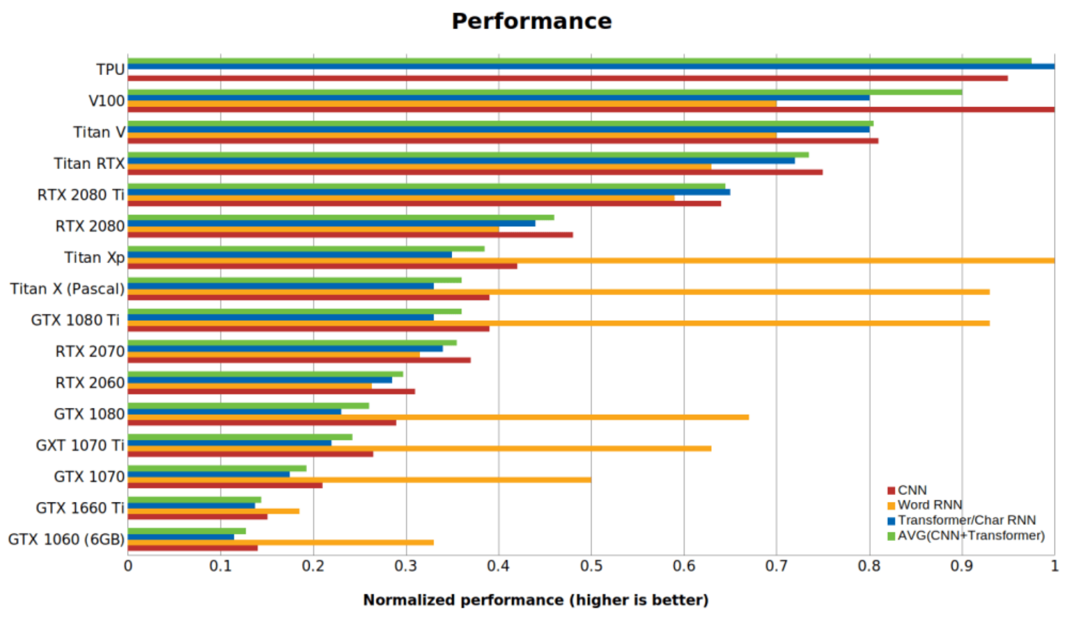


Figure 4.7: Benchmark of various NVidia GPU performed by Tim Dettmers[9]

Adding GPU to a Server

The GPUs could be added to a server owned by the faculty to make them accessible by every researcher. This solution assumed that there is a server compatible with the GPU acquired. If not, it is still possible to construct a compatible server for 4000$. The cost of such a server is equivalent to 4000 hours of GPU on AWS, 5700 hours on Google Cloud or 3500 hours on Floydhub.

Once the server is acquired, it needs to be installed and managed. An employee will be assigned to those tasks increasing the cost of the infrastructure. Finally, a system must be set up to avoid two people to use the GPU at the same time as it could badly affect the overall performance of a training task.

Building a GPU server is a good idea in the long run if the usage is intensive.

External GPU

Another solution is to build GPUs and an external bay. The GPU can be plug to a USB C port of the researcher computer and he can use it as a normal GPU. The main flaw of this method is the performance drop due to the slower data transfer rate of the USB C. The performance drop is between 15% and 40% depending on the external bay used. The price of this configuration is between 1300$ and 1800$. This solution is much more affordable than a server and does not require someone to manage the infrastructure. Each user configures their environment and can use the GPU to accelerate their computation when needed assuming that they have a USB C connector on their computer.

4.3.3 Retained solution

The internship during only three months, we decided to begin with cloud solution to quickly train and test or firsts models. our choice stopped on AWS because we had already some experience with that platform enabling us to start rapidly the project. A budget request was made to buy some external GPU.

Once we have the External GPU, I suggest to keep them to test the complete pipeline and to estimate the quality of the model and to launch big compute instances on a cloud platform in order to train the final model and to evaluate it.

4.4 Conclusion

The system required by the LSFB-lab is complex and will be made of multiple subsystems. Future works should focus on one of the most complex subsystem consisting of a machine learning algorithm able to associate a sign to its French translation. Experiments will be run in order to design the best algorithm for the task.

The quantity of experiments required to design a robust model is time-consuming and comparing or managing the various models created during these experiments could be cumbersome. A conceptual framework assessing these issues was created. This framework highlights the important step of a machine learning pipeline and suggest a code architecture facilitating the modification of the behavior of each experiment's step and encouraging to save the resulting output of each step in order to be able to reuse some intermediate results for later experiments. It could also be the backbone for a team project allowing each team to enrich a library of module compatible with the experiment pipeline used by all.

Due to the increasing popularity of deep learning, finding a platform proposing GPU to train our model is not as challenging as it was. But this kind of equipment is still expensive and it is difficult to identify if it is much worse to rent GPU on one of the various cloud platforms available or if it is better to buy our GPUs for our private use.

Chapter 5

Implementation

This chapter presents more our implementation of the conceptual framework discussed previously. This implementation includes the creation of the core of the experiment system and the development of modules that could be reuse in order to create a pipeline with real behavior.

We are also going to talk about the firsts experiments run on the implemented framework. Before running experiments an exploration of the LSFB-lab data was made in order to know the manipulated data a little bit better and to see what is its content. Then we move to the actual experiments for building a sign recognition system.

The various experiment and their results will be presented.

5.1 Data Exploration

When starting a data science project, the first action to take is to acquire knowledge about the data we are going to manipulate. In our case, the data consist of 975 Go of raw video along with ELAN annotation files. The first action was to extract each labeled sign from the raw video using the annotation file provided.

Since a lot of tools for data science are written in python, It was decided to use this language for this project. The tokenization of the video was done by using an existing python library facilitating the manipulation of the ELAN format.

Each extracted sign was saved as a *GIF* in a folder named as the gloss associated with this sign. The id of the speaker performing the sign is put in the filename to keep this information. Nevertheless, the information of the hand performing the sign is lost. It is intentional as we want to build a model agnostic of the hand used for the sign. However, if we find out that this is not possible, it is possible to modify the tokenization in order to keep more meta-information about the performed sign.

Once all the signs are isolated, it is easier to explore them to acquiring some knowledge about the manipulated data. Some basic metrics were extracted from the raw sign files. Those metrics are stored in a *JSON* file to be reused during the statistical analysis. This file contains, for each set of signs the following information :

- **Number of examples** : The number of video clip showing the sign performed,
- **Number of signers** : The number of unique signer performing the sign,
- **Max frames** : The number of frames of the longest sign in the sample,
- **Min frames** : The number of frames of the shortest sign in the sample,
- **Average frames** : The average number of frames required to perform the sign,
- **Signers** : Set of IDs for each signers present in the clip,
- **Name** : The gloss associated with the performed sign

Due to the quantity and the nature of the data, generating such a file took 6 hours. If simple processing like that already takes so long, we could conclude that it will be too time-consuming to process all the data at runtime when executing the pipeline. We will need to save all the pre-processing in a database to reuse them between experiments.

By observing which signs are the most common, we were able to identify signs who are irrelevant for the training. The identified signs were the following :

- **PALM-UP** : This label is associated with a shrug. This sign is highly dependant of the context and is like punctuation in sign language. This could be interpreted as a question mark, an exclamation point or invite the interlocutor to answer to the previous sentence. Therefore, this sign is present everywhere making it the most common in the dataset. Its presence could bias our model. As it is not a critical sign for the search in the dictionary, we decided to discard it from the training data.
- **INDECIPHERABLE** : It is another common label in the dataset. This label is only there to identify all the signs for which the meaning could not be established. Therefore, we can have to discard this class before training the model.

The duration of a sign is also a piece of important information that could be explored, the observations were :

- **Average duration** : Between 55 and 5 frames
- **Max duration** : Between 700 and 6 frames
- **Min duration** : Between 20 and 1 frames

As expected, all the signs cannot be performed in the same amount of time. The video clips are encoded at a rate of 24 images per second, this means that a sign is, on average, performed in less than 2 seconds. Some of them taking only a few milliseconds.

The observation made on the max duration and the min duration of a set of signs is highly valuable when cleaning our dataset. We can identify some outliers in the data. The longest sign in the dataset takes nearly 30 seconds, Fig 5.1 shows clearly that the sign labeled *LBUOY(7):DEUX* contains outliers when compared to the other signs duration. there is also a significant amount of sign clip taking 12 seconds to be performed which is surprising when we know that other clips of the same signs take only milliseconds. After discussing with the annotator, it seems that these long during signs are instances of *holded sign.* When speakers are interrupted during their explanations or wonder about the next sign to perform before continuing their story, they tend to keep their hands in the final configuration of the last sign performed. This explains why these signs could last a significant amount of time.

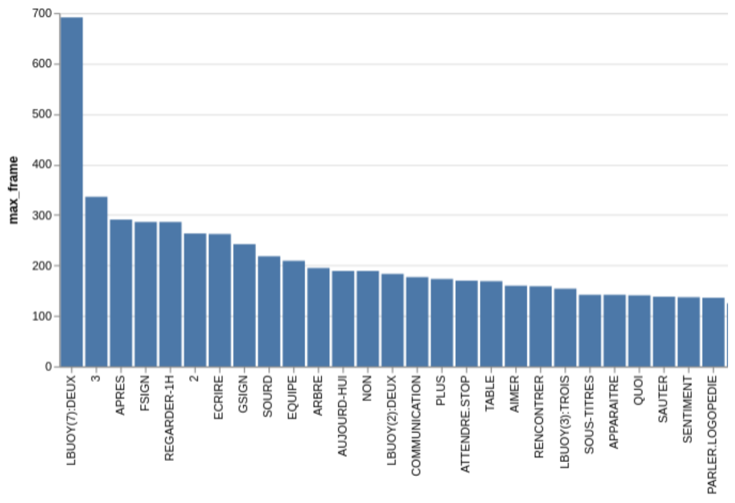


Figure 5.1: The signs taking the most time to be performed

Holded signs could be an issue as the actual action is made at the very beginning of the video clip and the majority of the clip only contains a static sign position. During the pre-processing phase, special treatment should be done for *normalizing* these signs.

Finally, the last important metric is the number of different signers represented in the corpus. We already discuss the importance of the variability of the speakers when it comes to applying deep learning algorithms to the dataset. We only analyze the 627 signs having more than 10 examples. In this subsample, less than 50% of the signs are performed by more than 10 different speakers and more than 85% of the signs are performed by more than 5 speakers. At this point, it is hard to say if this variability is enough but we are not in a catastrophic situation where a majority of signs are performed by one or two speakers.

Even if those statistics are simple, they allow us to acquire knowledge about our dataset and to highlight some issues about it. The observations will help us to avoid some pitfalls during the pre-processing stage.

5.2 Framework Implementation and Module Development

Earlier, this work introduced a conceptual framework aimed to facilitate the setup and replication of experiments for Machine Learning workflow. It is time to implement it. The technology chosen was Python due to the number of libraries for machine learning and data manipulation available for this language.

In this work, it was chosen to use a popular library for deep learning called *Keras*[6]. Our implementation will be highly dependent of the object and workflow defined by Keras thus, a module developed with other frameworks (Tensorflow, Pytorch) will have lower chances to be compatibles. This is not an issue in teams using always the same technology. Teams using multiple technologies should be careful if they want to see their developed module reuse in various projects.

The following sections will present the different modules created in order to run the first experiments on the LSFB corpus. However, this section will not cover all the details of my implementation and how to reuse the developed module. If you are interested in more precise documentation explaining how to build or reuse module in the python architecture is available on this github repository : https://github.com/Jefidev/LSFB-experiment

5.2.1 Data Pre-processing

To exploit the raw sign video provided by the LSFB-lab and to test the first iteration of the pipeline, a simple data pre-processing module was designed. This module takes as an input a folder containing all the raw *GIF* video sorted by sign and the output of the process is a new folder containing only one image for each sign performed in our corpus. The image is taken in the middle of the raw *GIF*.

The main idea behind this operation is to design, at first, a simpler version of the problem allowing us to test all the developed modules and to have a baseline result enabling us to situate the performance of our future model. Taking the image in the middle of the video was not a random guess. Members of the LSFB-lab tells us that seeing a static image of a sign being performed could be enough to determine the meaning of the sign in some cases. By choosing the middle frame, we make sure that the sign is actually performed and not in a transition phase with the previous or next sign.

Running this pre-processing phase took 6 hours. The results of this step were stored in a database. This will allow us to run multiple experiments on these data without having to re-run this phase each time.

5.2.2 Train Test Generation

Loading and splitting the pre-processed data into two different sets requires particular attention. The whole dataset does not fit in memory, we should then load the images only when needed during training and test. Fortunately, Keras already provides a solution to this issue: *Sequence*.

Keras models could receive a Sequence in parameter for the training and the testing. Objects inheriting from a sequence must implement a callback function that will load data in memory progressively during the training. Keras optimize the training of the model with the data loading by running them in two different threads making the whole process as efficient as possible.

In our implementation, the Train Test Generation step is expected to return two sequences. One for the train set and one for the test set.

Currently, one module was developed for splitting the data and two Sequence objects were created for loading the sign language images. The split module filters the input data by dropping the irrelevant signs identified during the data exploration phase and split the remaining data into two sets used in the train sequence and test sequence object.

The two sequences created are the following :

- **ImageSequence** : It is a simple object that load a batch of images when required by the model. The loaded image is in RGB and are normalized in order to ease the model training.
- **TripletSequence** : This sequence generates a batch of triplet to train a model using the triplet loss methods explained in the stat of the art part of this work. The triplet are generated following the *semi-hard* formula.

5.2.3 Model

The module created is a triplet loss model using transfer learning. It was the hardest module to create and to test as it needed several experiments before working properly and each experiment took a large amount of time before revealing if the modifications made caused some issues or improvement. This is also one of the modules with the most variability with a lot of parameters and meta parameters. Some of these meta parameters could be set in the configuration file. We can change the value for the input size of the image and the size of the embedding space.

The model also uses transfer learning approach to reuse the weight of an *InceptionV3* model trained on imagenet. Fig 5.2 show the constructed model. The InceptionV3 layers are frozen, it means that their weight will not be updated during the training phase. The one learned on imagenet will be kept. D1 is a *Dense* layer having an output size of 2048, D2 is another *Dense* layer with an output size of 1048 and finally D3 is the last *Dense* layer having the output size specified in the configuration file and an linear activation function (a.k.a identity function). Usually, the activation function forces the neural layer to output values between 0 and 1. The linear function is not limited to a range of value enabling the last layer to properly separate each point of the embedding space.

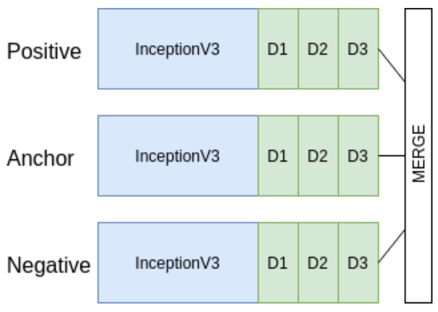


Figure 5.2: Architecture of the triplet loss module implemented during this work

The network receives a triplet of images as input. Each image is evaluated by the network, the embedding produced are merged and sent to a loss function that will penalize the network if the negative image is closer to the anchor than the positive image using the formula cited by Schroff in his Facenet paper[26].

5.2.4 Analysis

The only thing produced by our model is an embedding point for each image in the test set. The module developed will analyze the quality of this embedding will produce two outputs.

First it will generate a **confusion matrix**. The predicted label of a test image will have the value of the true label of its closest neighbor in the embedding. This process was time-consuming when working with a large amount of test data. This step could be optimized in order to find more quickly the predicted label of an image. The metric used to compute the distance between the different point is the *euclidean distance*. The confusion matrix has the advantage to be easy to read and provides interesting insight about the performance of the model for each class. If there is too many different class the confusion matrix could become unreadable that's why we compute the *f1 score* and the *accuracy* on the results used to build the matrix.

Another output is a 2D visualization of the embedding. To project the points produced by the deep learning model from a 128 dimensions space to a 2-dimensional space, we used a reduction dimension methods called *t*-SNE [33]. compared to other dimensional reduction methods, *t*-SNE tries to conserve the structure present in the high dimensional space when building the 2D representation. This means that if two points are close to each other in the embedding space they will be put close to each other in the 2D representation constructed by t-SNE which is interesting in our case. Other algorithms are not made to conserve those properties.

Visualizing all the class present in our dataset on a 2D graph is not possible. There will be too much point and there are not enough different colors to represent each of the classes. That's why we only keep the 10 most common class when building this visualization. A lot of information us, therefore, lost but the visualization could be read by a human and we can quickly see if the triplet loss model makes a good job at separating the different classes.

5.3 Conducted Experiments

All the modules described in the previous section were developed during the various experiments made for designing a baseline model for sign language recognition. The purpose of these early steps was to test and improve the structure of the project and to have a first reference model that could be used to compare the future ones

5.3.1 Whole Corpus Experiment

During the first iteration of experiments, all the relevant signs from the corpus were taken. Signs with less than 10 examples were also removed from the data. The dataset contains 627 different signs.

The chosen approach is a triplet loss for generating an embedding where each cluster of distinct signs could be easily identified. This approach was proven successful on small datasets [35]

The pipeline instantiated by this experiment is shown in fig 5.3

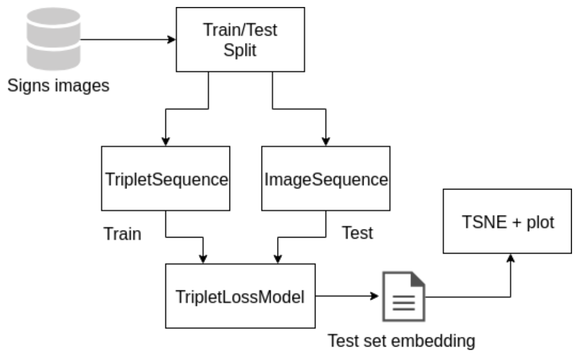


Figure 5.3: Instantiation of the experiment pipeline for the full data experimentation

The input data are fetched from a database where were stored the result of the pre-processing process isolating a single image from the middle of each sign video. From there, the pipeline runs as follow :

- **Train Test Split** : Will read the data from the database and filter them to keep only the one having more than 10 examples. Then, it split those data. 80% of them go in the train sequence and the remaining to the test sequence, respectively a triplet sequence and an image sequence.
- **The model** : The model uses the concepts of triplet loss and transfer learning. *Inception V3*[31] was use in the architecture. It is currently the visual recognition model with the best result on ImageNet provided with its weights by Keras. This part of the process was revised a lot of times in order to find the most effective architecture in this case.

  The model was trained with the data generated by the triplet loss sequence. The training phase was considered completed if the model does not improve since 3 iterations over the training dataset.

  When testing the model, it only computes the embedding of all the images in the test set and saves each embedding with the true label allowing us to inspect them later.
- ***t*-SNE plot and Confusion matrix** : To visualize the embedding created by our model, the test data are loaded and the dimension of the latent space is reduced to 2. This enables us to plot each instance on a scatter plot graph facilitating the inspection of the embedding.

  A confusion matrix was also plot but the high number of different class make it unreadable.

This experiment was the first using the framework build and suffer from all the bugs at each level of the architecture. It is during this experiment that the architecture of the framework improved. Also, when implementing each module, a few pitfalls came up.

The most critical one was the tendency of the triplet loss to *collapse*. This means that the model tended to map all the images to the same point in the latent space. Using *semi-hard triplet* instead of *hard triplet* to train our model should have made disappear this issue. Unfortunately, it persists and after some experiments, it turns out that the number of dimension of the latent space was probably too high and that seems to lead to a collapsing. The latent space was set to 1024 dimensions. The models created for *face recognition* by Schroff [26] have a latent space of 128 dimensions. Reducing the latent space seems to have resolved, or at least drastically mitigated the issue.

Even with this issue solved, the resulting models were not able to create a useful embedding. By watching at some example from the dataset we can measure the difficulty of the task. Fig 5.4 shows one of the many conflicting signs present in the dataset.

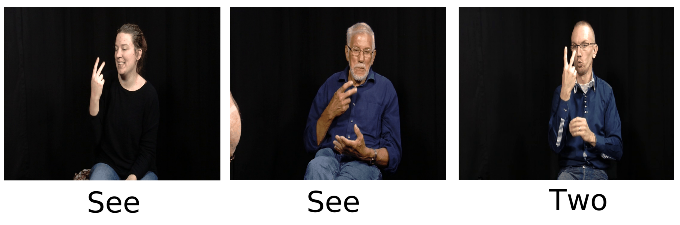


Figure 5.4: Hard examples from the dataset.

Even a human is not able to distinguish the different signs. The information enabling to discriminate those signs is held by the movement.

Working on the dataset does not give us interesting results. However, during this process, a lot of module for the pipeline was created and optimized for handling the large amount of data provided by the LSFB-lab.

5.3.2 Most Represented Signs

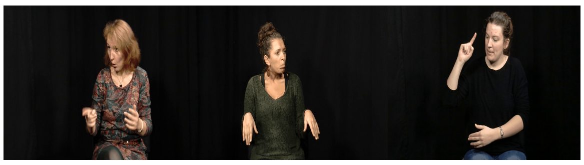
Analysing the results on the whole dataset is not easy, there are too many signs and too much confusion between them. In order to better understand the behaviour of the model, it was decided to train it on a subset of the LSFB-lab dataset. Naturally, the chosen signs were the one with the most examples in the corpus. We decided to take all the signs with more than 200 examples. Once again, the irrelevant signs were dropped. This reduced dataset is made of 9 different signs an example of each sign is shown in fig 5.5.

As you can see, the sign for *FALLOIR* is blurry due to the movement. It is not uncommon to see the most severe blur on extracted images. This could affect the ability of the model to classify the blurred signs.

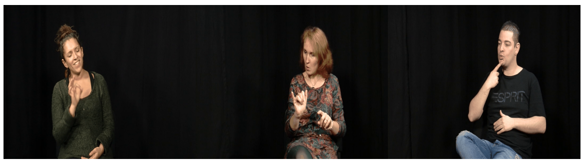
Once again the results were not compelling as shown in fig 5.6. The clustering produced shows that the various signs are not correctly separated into distinct clusters indicating that the triplet loss is not able to find a solution based on the data provided. The confusion matrix confirms this observation. The number of correct prediction is close to zero. The pre-processing phase should be changed to get better results on these data.

Our basic model is not able to predict correctly the label associated with a sign by looking at only one image. The movement information seems mandatory to differentiate these signs.

(a) AUSSI (b) AVOIR (c) C-EST



(d) FALLOIR (e) GSIGN (f) MAIS



(g) OUI (h) QUOI (i) SOURD

Figure 5.5: The most represented signs of the dataset

5.3.3 LSFB MNIST

The last experiment made was to test the model upon a specific category of signs: The numbers.

In contrary to the other signs, numbers are always executed in the same way whatever the signers. It consists of showing the number we want to express with the fingers of the hand.

The signs used to train the model goes from 1 to 5. We limited to these numbers for two main reasons :

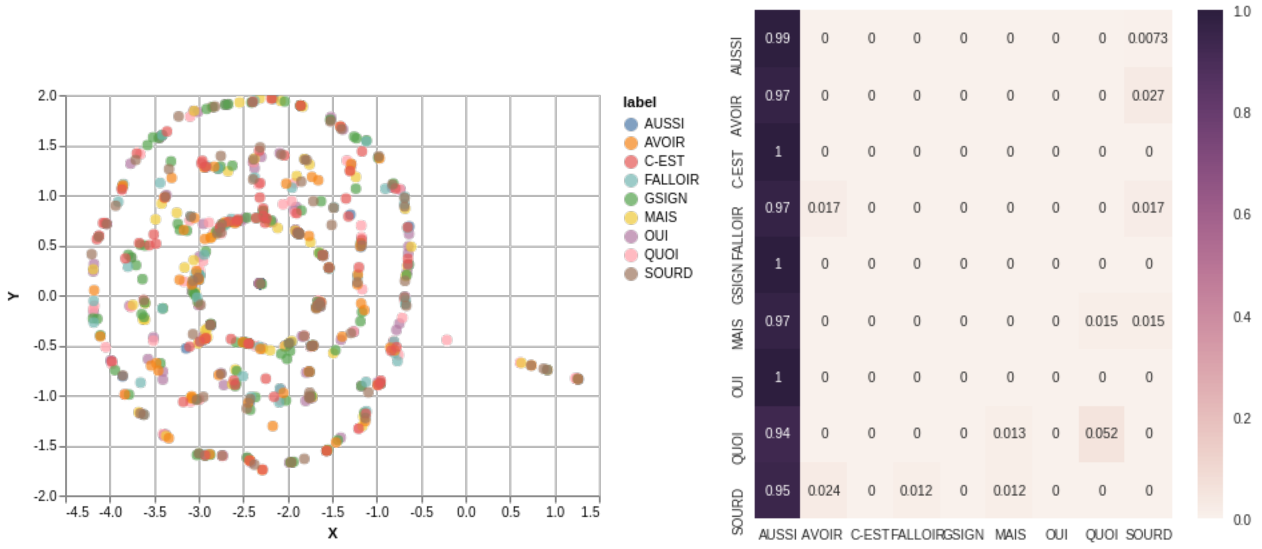
- To limit the examples to signs performed with only one hand,
- There are too few examples for signs 6,7,8 and 9 are

By looking at the fig5.7 we can see that, even if all the signs could easily be distinguish by a human, the task is not trivial. There is some motion blur on the images, it could be hard to distinguished the hand when the sign is performed to close to the head, the camera angle could mask some fingers of the images. The limited number of example is also a challenge.

These signs are static, the movement performed by the hand during their execution is not essential for differentiate a 1 from a 5. We can expect to have better results with a single image on this subset than on the subsets used in the other experiments.

The main purpose of this test is to find out if a transfer learning model can identify the number of fingers raised on a hand.

It turns out that this experiment was much faster to run, this allows us to test more configuration than on the previous experiments and that lead to an interesting observation. Since



(a) *t*-SNE (b) Confusion Matrix

Figure 5.6: Confusion matrix and 2D plot of the embedding

the beginning of the process, the transfer learning model was created upon an *inceptionV3* model. It is the more powerful model for image recognition provided by Keras. It turns out that, in our case, it was not the best one for transfer learning. We tested to train a model based on a *VGG16* model. This model is older than *InceptionV3* and is less performant on challenges like imagenet. But the results given by *VGG16* are better as shown in fig 5.8. The confusion matrix constructed with the results of our model using inception V3 is discouraging. There is no correct classification and the model always predict the most represented label in the corpus. The model using VGG16 is more encouraging and seems to be able to recognize some signs from the test set. For instance: 25% of the number *4* are correctly labeled.

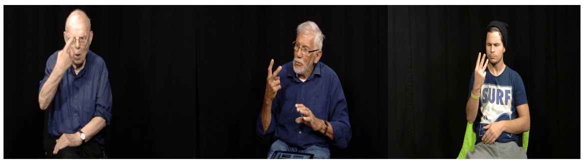
Unfortunately, this improvement is not enough to make a model able to classify the five different signs.

5.4 Conclusion

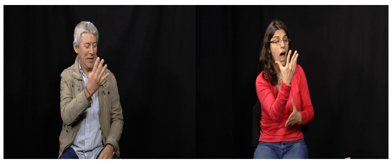
Working with a large amount of data is challenging. As it is impossible to review each piece of information by hand, it is critical to explore the data by doing simple but systematic statics analysis upon them. Those analyses able us to identify outliers and make us gain some insights about the manipulated data.

The framework designed revealed to be very convenient to use. The first experiment takes a lot of time to set up due to the number of modules written and tested but once they were ready it took only a couple of minutes to modify the configuration file allowing to run other experiments.

Unfortunately, the results of those experiments are not conclusive but the result analysis allows us to improve and debug the module used and to gain more knowledge about the case we have



(a) 1 (b) 2 (c) 3

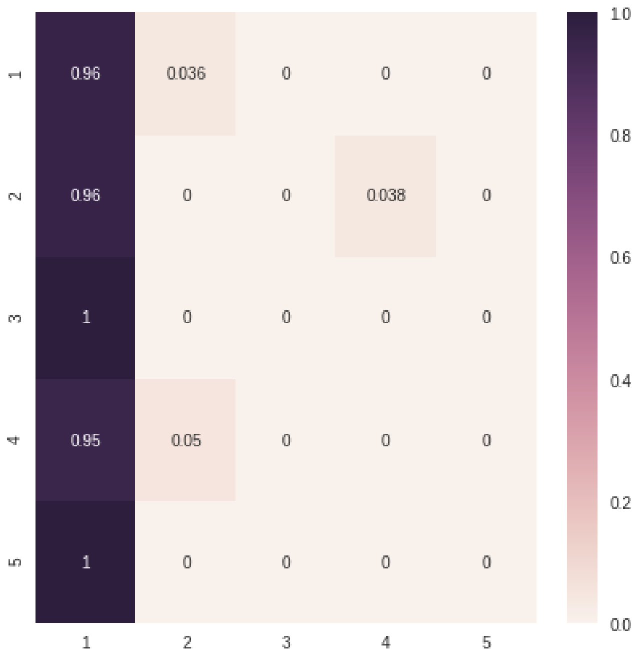


(d) 4 (e) 5

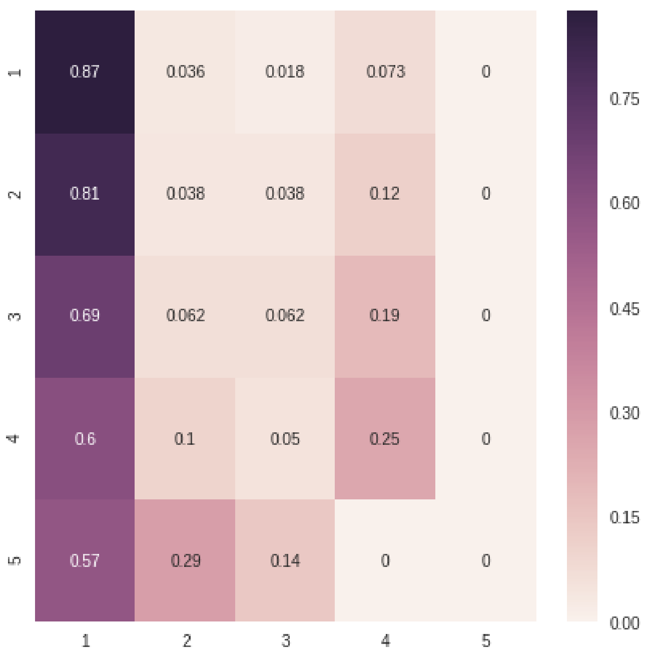
Figure 5.7: LSFB MNIST subset. The name MNIST is a reference to the famous handwritten digit corpus use by the Data Science community.

to solve. This knowledge will be useful when designing more powerful models.

By after, it became clear that using the whole dataset when designing the framework and the first module was not optimal. It would have been more effective to use a known dataset such as MNIST[19] for creating and testing all the modules in a more controlled way. Using an unknown dataset was, sometimes, miss leading. It was hard to figure out if a bad result comes from an error in the implementation of the triplet loss or from the dataset.



(a) InceptionV3



(b) VGG16

Figure 5.8: Confusion matrix computed on the test set of the MNIST LSFB data

Chapter 6

Conclusion and Future Work

In the past few years, video recognition methods have benefited from a renewed interest due to the success of deep learning methods. This lead to the creation of new methods for this task. These methods have shown their superiority over the classical ones.

These advances could be applied in our use case: The Belgian Sign Language recognition. But the process is not straightforward and require a lot of experimentation before giving results. The low number of examples force us to associate new advances in video recognition with one-shot learning methods to get better results.

This works is the first step leading to an algorithm able to translate sign language into text or speech. This chapter presents the work already done and the next steps needed for reaching our goal.

6.1 Conclusion

The current work could be divided into two parts: first, there was the design and implementation of an architecture for deep learning experiments. Then, experiments using the proposed architecture were run. Those steps were not sequential. The issue identified during the experiments set up helps to improve the architecture.

6.1.1 Architecture

The conceptual framework designed was implemented in python to profit from its ecosystem. The conception of the project architecture represents the majority of the time spent on this work. Having a good code organisation and making sure that all the steps could be reproducible will help for the next steps of the LSFB project.

Currently, the project contains several modules ready to be used in an experiment. They were tested and debugged during the first experimentations conducted. The full documentation of the python implementation and the existing modules is available on the GitHub repository of the project (`https://github.com/Jefidev/LSFB-experiment/`).

Working with our implementation is an investment. When designing a new experiment, it takes more time to write the code for each step of the process. The developer have to think about all the parameters required by the new module, he also has to think about the modularity of the piece of software he is building to make sure that it is reusable for future experiments. But this cost comes with its lot of benefits. The first one is the presence of a configuration file. To change the behaviour of the experiments, the developer just has to change parameters in this file instead of having to search the correct class to modify. Being able to reuse the existing module also save a lot of time.

The first experiment designed following the architecture took about two weeks. During this time all the modules were created and tested. The failures in the conceptual architecture were spot and corrected. Once it was done, it took only about two hours to set up a new experiment reusing a majority of existing modules. From this point of view, the implementation proposed is a success and could increase the number of experiments run in the same amount of time.

The conceptual architecture could be adapted for other use cases and languages making the approach flexible and technology agnostic. There is plenty of tools for machine learning and data science but, usually, they are too complex and only compatible with few technologies making them unsuited for medium-sized projects or companies using specific or homemade technologies. Our conceptual architecture is much more versatile but requires to be implemented by developers.

6.1.2 Experiments

These experiments enable us to test the python implementation developed. They provide us with more insight about the manipulated data and forces us to investigate the LSFB dataset in depth.

This teaches us that the chosen approach was not optimal, trying to create a first model for the whole dataset was utopian and analyzing the mistakes committed by the model on the 627 classes was hard. The first iteration on the whole corpus allowed to gain more knowledge about the data and how to manipulate them. But it was harder to figure out what was wrong with the model generated by this experiment.

The subset of the dataset picturing the signs for numbers was much more valuable as the results could be more easily analyze and that more experiments could be run on them in the same amount of time. It could be useful to design a model for this task before extending it for the whole dataset.

It is more useful, to begin with a simpler task and than move on to the more complex one than trying to solve directly the complex one. This was the main learning of the experiments conducted during this work.

6.2 Future Work

This master's thesis lay the groundwork of the LSFB project. But there is a lot more to do to get a model able to translate sign language into text. In the future, more module should be build to enrich the existing project and construct more complex experiments. The next step should be to take the time information into account. Until now, all the experiments were run on fix images. The information encoded in the time flow of the executed sign is lost when observing just one image. The model identified in the state of the art chapter should be implemented and tested for our case. Other approaches could be investigated (e.g hidden Markov model[25]).

The pre-processing applied to the raw data could also be improved. In its two-stream approach[28], Simonyan applies various methods of pre-processing methods on the video for extracting more information from them. It could be interesting to apply those process in our case. Once again, other methods exist an need to be investigated.

6.3 Final Words

The task of sign language recognition is far from trivial. As all the natural languages, sign language is complex and evolve in time. Signs could have various meanings or interpretations this variability is very confusing for a computer. Additionally, video recognition is a hard task that requires a lot of power.

A zoo of methods exists for tackling each aspect of the problem. The difficult and time-consuming part is to assemble and test these methods to construct a piece of software able to identify and translate signs robustly. This work aims to ease the process of trial and error and facilitate the job of the developer.

Bibliography

[1] Vassilis Athitsos, Carol Neidle, Stan Sclaroff, Joan Nash, Alexandra Stefan, Quan Yuan, and Ashwin Thangali. The american sign language lexicon video dataset. *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 0:1–8, 06 2008.

[2] Herve Bredin. TristouNet: Triplet loss for speaker turn embedding. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, March 2017.

[3] JANE BROMLEY, JAMES W. BENTZ, LÉON BOTTOU, ISABELLE GUYON, YANN LECUN, CLIFF MOORE, EDUARD SÄCKINGER, and ROOPAK SHAH. SIGNATURE VERIFICATION USING a “SIAMESE” TIME DELAY NEURAL NETWORK. *International Journal of Pattern Recognition and Artificial Intelligence*, 07(04):669–688, aug 1993.

[4] Joao Carreira, Eric Noland, Andras Banki-Horvath, Chloe Hillier, and Andrew Zisserman. A short note about kinetics-600, 2018.

[5] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jul 2017.

[6] François Chollet et al. Keras. `https://keras.io`, 2015.

[7] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. *Computer Society Conference on Computer Vision and Pattern Recognition*, 2005.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[9] Tim Dettmers. Which gpu(s) to get for deep learning: My experience and advice for using gpus in deep learning. Accessed: 2019-04-15.

[10] Jeff Donahue, Lisa Anne Hendricks, Marcus Rohrbach, Subhashini Venugopalan, Sergio Guadarrama, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):677–691, apr 2017.

[11] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. Slowfast networks for video recognition, 2018.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, nov 1997.

[13] Mohammad Hossin and Sulaiman M.N. A review on evaluation metrics for data classification evaluations. *International Journal of Data Mining Knowledge Management Process*, 5:01–11, 03 2015.

[14] Norman P. Jouppi, Al Borchers, Rick Boyle, and all. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture - ISCA 2017*. ACM Press, 2017.

[15] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, and Andrew Zisserman. The kinetics human action video dataset, 2017.

[16] Gregory R. Koch. Siamese neural networks for one-shot image recognition. 2015.

[17] Oscar Koller, Jens Forster, and Hermann Ney. Continuous sign language recognition: Towards large vocabulary statistical recognition systems handling multiple signers. *Computer Vision and Image Understanding*, 141:108–125, December 2015.

[18] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, dec 2015.

[19] Y. LECUN. The mnist database of handwritten digits. *http://yann.lecun.com/exdb/mnist/*, 1998.

[20] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[21] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, may 2015.

[22] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, nov 2004.

[23] Laurence Meurant. Corpus lsfb. first digital open access corpus of movies and annotations of french belgian sign language (lsfb)., 2015.

[24] Eng-Jon Ong, Helen Cooper, Nicolas Pugeault, and Richard Bowden. Sign language recognition using sequential pattern trees. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Providence, Rhode Island, USA, June 16 – 21 2012.

[25] L. R. Rabiner and B. H. Juang. An introduction to hidden markov models. *IEEE ASSp Magazine*, 1986.

[26] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2015.

[27] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and

K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 568–576. Curran Associates, Inc., 2014.

[28] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[29] Anne Smal and Jeremy Lebutte. Etude de la faisabilite d'un support automatise a l'annotation de videos en langue des signes : Cas du corpus lsfb. Master's thesis, UNamur, 6 2017.

[30] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild, 2012.

[31] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2016.

[32] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[33] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne, 2008.

[34] Hanjie Wang, Xiujuan Chai, Xiaopeng Hong, Guoying Zhao, and Xilin Chen. Isolated sign language recognition with grassmann covariance matrices. *ACM Transactions on Accessible Computing*, 8(4):1–21, may 2016.

[35] Meng Ye and Yuhong Guo. Deep triplet ranking networks for one-shot recognition, 2018.

[36] Lihong Zheng, Bin Liang, and Ailian Jiang". Recent advances of deep learning for sign language recognition. 2017.