Recognizing and transalting words from language sign

Alessio Arcudi
alessio.arcudi@studenti.unipd.it

Alessandro Mazzoni
alessandro.mazzoni.2@studenti.unipd.it

Alessandro Padella
alessandro.padella@studenti.unipd.it

Abstract

The problem of detecting words in language sign is a very hard problem for deaf people. For a goal like this one, Computer Vision could be a very interesting and challenging solutions; in fact, nowadays there are many machine learning algorithms that are able to gain information given a sequence of images.
In our case, we work with videos from Youtube where people mime words, and with machine learning tools (in particular with convolutional and recurrent neural network) we try to build an algorithm with the capability to recognize the word mimed given a video where someone mimes it.
For this purpose we start with the MS-ASL *dataset from Microsoft, available online.*

1. Introduction

The problem we chose to face is to train a model on videos in which there are deaf people who mime the american sign language labelled with the corresponding word, trying to come up with a classifier using videos from a dataset provided by Microsoft.
This choice has been guided by the fact that we consider the topic interesting and open to many applications in the real life(such as apps able to immediately translate videos in words).
Another reason behind this choice is the fact that we have never worked with a completely raw dataframe of videos in which may be present all types of noise and we felt it challenging. This is a skill that a data scientist must acquire for everyday applications, because it is very common to face not preprocessed and noisy datasets, and the ability of a data scientist must be to clear the data and to get information for them. That's the reason why we spent much time to exploit a satisfying and complete pre-process and filtering.
In the first part we downloaded the videos using `pytube3` and cut them keeping just the part in which there's the mime (it has been done using `moviepy` package); then we uploaded all the file to google drive because we needed to use Google Colaboratory's RAM and Disk and we used the `google.drive` packages to operate and load files contained in the drive. A first attempt of pre-processing has been done using keras-video and its frame generators function, but just the simple frame generator works properly (`OpticalFlow` and `SlidingFrameGenerator` are still nearing completion), so we moved to `opencv` as the course suggests.
In the second part we presented the methods that in literature are widely used for this kind of application, in particular highlighing the advantages of inserting a recurrent layer that can take care of the timing between a sequence of frames.
Then we presented the experiment we made: we passed the three dataset we got from the initial filtering in two networks with different recurrent layers, getting much better results with Long-Short Term Memory layer.
The results we obtained are consistent with the literature about this dataset ([1]), that aren't anyway so exciting both because of the quality of the dataset and because of the models we could develop (the authors in [1] achieved much better results building more powerful neural networks for which we don't have sufficient knowledge).

2. Related work

We have to say that there aren't many works on word detection from videos of people who mime them in the language sign.
The most important one, from where we took inspiration to build our work, is [1] by Hamid Reza Vaezi Joze and Oscar Koller, two researchers who work for Microsoft USA and Microsoft Germany respectively.
In this work, these researchers used 3 different approaches to build a machine with the capability to recognize a word in sign language:

- **2D-CNN**: the convolutional neural network approach is maybe the most common approach when dealing with Computer Vision problems. The very first way

was to build a classic CNN on the frames without a specific order or timing, and that gave very bad results; so, the further step they made was to insert in the neural network a LSTM layer that could give the sense of flowing informations through time: in this way they achieved much better results;

- **Body Key-Points**: with sophisticated techniques, the researches wer able to extract from the frames 137 body key-points, given the fact taht very recent studies perfected the hand key points relevation techniques. Then they built a hierarchical co-occurrence network (HCN) given in input those 137 body key-points (which include hand and face key-points);

- **3D-CNN**: following these approach the researchers optimized and slightly modified some 3 dimensional CNNs (3D-CNN and I3D CNN) yet trained on other datasets (on ImageNet and on Kinetics).

For our work we decided to take inspiration from the first approach in [1], that is the one that combined 2 dimensional CNN and recurrent neural network.
We had to do this, because the other approaches required many tools we didn't know and much more performing computers on which train the models, even if this approach didn't give very exciting results in [1] (they achieved 13.33% on the 100-words classification problem).

3. Dataset

3.1. Data downloading

The dataset we deal with is the MS-ASL dataset, composed by three `.json` files (one for the training set, one for the validation set and the last one for the test set) containing lists of american words and, for each word, a related Youtube link to a video in which a person mimes that word in the sign language. We got those files downloading them from [2].
Once we got those files, next step was to download all the videos from Youtube and then extracting all the frames from them. For this purpose we used the function `Youtube` from the Python package `pytube3`. Then, we created a directory for each word and there we stored all the extracted frames from the related videos using two different approaches: the Python package `opencv` and the package `keras-video-generators`. We decided to deal only with a ten class classification, in the sense that we only kept data (videos and frames) related to the 10 words in the dataset that had the highest number of related videos, because dealing with more classes would have brought computational issues. These words are: book, eat, fish, help, milk, nice, pencil, red, sad and white.
In total, we have 234 videos splitted in this way: 180 videos for the training set, 30 videos for the validation set and 24 videos for the test set.

3.2. Pre-processing with keras

The first approach was the one with the functions of the package `keras-video-generators`. In this way we were able to exploit 3 ways to generate frames:

1. `VideoFrameGenerator` that takes a certain amount of frames from the given video;

2. `SlidingFrameGenerator` that takes frames with decay from the given video;

3. `OpticalFlowGenerator` that gives optical flow sequence from given frames.

Anyway, the results were not so satisfying because these functions are still in the experimental phase and are not optimized and ready to be used (in fact we tried to use these frames to train some networks but the frames didn't perform well at all).
So we moved to the Python package `opencv`, widely used for this kind of pre-processing.

3.3. Pre-processing with opencv

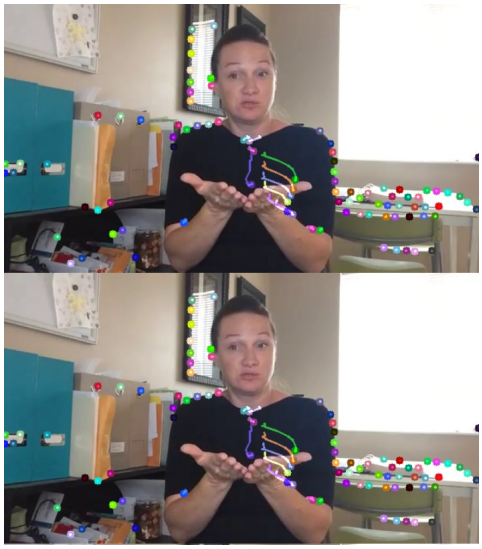
For the frame extraction from the downloaded videos we studied two different ways of pre-processing the frames when using the package `opencv`, the first one performing the Farneback algorithm and the second one performing the Lucas Kanade algorithm together with the Shi-Tomasi corner detector.

3.3.1 Farneback algorithm

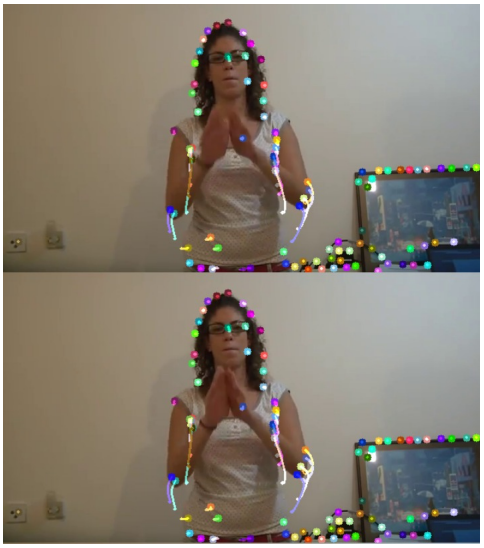
We extracted 22 equally distributed normal frames for each video and we saved them. Then we filtered the frames following the Farneback algorithm (that is implemented in `opencv` by the function `calcOpticalFlowFarneback`).
With this approach, the frames are ideally put in a row and then the algorithm keeps only track of the movement between one frame and the next one, highlightig the parts in the frames that clearly change in this flow.
We also built a third set of frames, overlapping the normal ones with the ones coming from the Farneback algorithm. Here an example of normal, Farneback and overlapped application for the same frame in the same video:

3.3.2 Lucas Kanade algorithm and Shi-Tomasi corner detector

Another possible approach is the one that follows the Lucas Kanade algorithm.
According to this algorithm, in each frame of a sequence there are some corner points (highlighted and recognized with the Shi-Tomasi corner detector) and the algorithm can take into account the flow of these points along the sequence, keeping track of their movements during the time.
We tried with some of the available videos, and we saw that the first frame in the video is very important for the detection of the corner points; this thing, together with the heterogeneity of our videos, could bring many problems. In fact there are videos in which the person shows clearly his hands from the first frame, and this technique can understand the hands as corner points and keeps memory of their movements, while otherwise in many other videos the hands appear later and the coner detection doesn't perform well.
For this reason we decided to discard this way of preprocessing.
Just to be more clear, here a frame where this algorithm was able to detect the hand movement (the lines highlited take care of the flow of the corner points):



And here an example of an unsuccessful detection, in which the hand are not recognized as corner points and so the algorithm doesn't keep track of their movement:



3.4. Scrapping and initialization

Once we got all these different kinds of sets, the next step was to prepare these frames transalting them into tensors.
But before doing that, we had to check the measures of the frames: the images come from different videos with different resolutions, so we had to be sure that we were dealing with same-size frames.
We saw that the littlest frame has a pixel measure that is 240x320, so we scrapped all the others keeping just a 240x320 image.
After that, we were able to translate them into tensors.

We also performed a one-hot encoding among the ten classes of the label.

4. Model

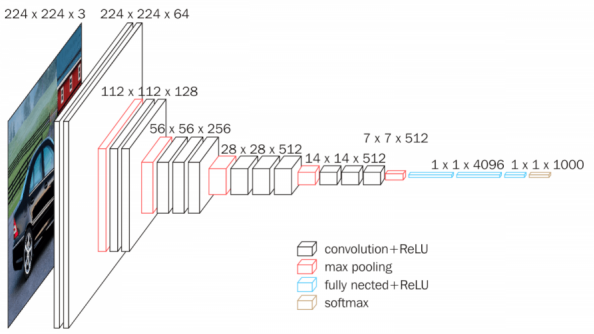
For the aim of this job we decided to implement some deep CNN. This is a very common approach when dealing with computer vision problems beacuse these methods can very well deal with grid structures, such as images in our case.
The method we decided to implement is the VGG network, that in literature are mostly studied in two different variants: VGG8 and VGG16. Let's see these neural networks more in details.

4.1. VGG network

VGG network takes his name from the Visual Geometry Group at Oxford University, where computer scientists started thinking about building CNN by using blocks of layers instead than single layers.
In particular, the structure of a block consists o a sequence of these layers:

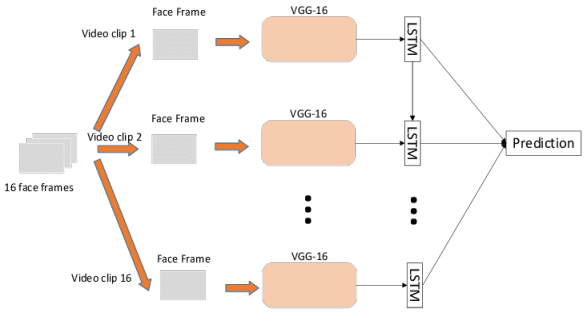
1. a convolutional layer with small kernel with padding;
2. a dense layer with a nonlinear activation function (usually the ReLU function);,
3. a pooling layer (usually the max pooling layer).

as thought in the first paper that theorized VGG networks ([3]).
The fact of using small kernels for the convolutional layer (in [3] they proposed to use 3x3 kernels) allows to use less parameters (avoiding in this way some kind of overfitting) and to build deeper neural networks, and this usually leads to reach better performances and results.
Moreover, after a sequence of these blocks, there is a sequence of fully connected layers in which the last layer has a softmax activation function to finalize the classification.
We talk about VGG8 and VGG16 thinking that the numbers 8 and 16 refer to the number of blocks in the network.
In literature ([4]) usually authors also insert a dropin layer in the convolutional block with the aim to favor the convergence of the network when it's untrainable, and this can happen when dealing with very deep neural networks (in fact in the VGG8 network there is no dropin layer because it's less deep).
Here you can find a scheme representing the structure of the VGG16 network (the numbers are related to the hyperparameters chosen for the experiment presented in [3]):



4.2. VGG-RNN network

There are further improvements ([5], [6]) of the VGG network approach that include a combination of deep CNN (in our case a VGG network) and recurrent neural network, such as long-short term memory layer (from now LSTM) or a gated recurrent unit (from now GRU).
Following this approach, the CNN performs a a feature extractor by fixing all of its parameters, and then it passes the results obtained to the LSTM layer which propagates the information through time. Lastly there are the fully connected layers that exploit the final classification.
Here you can find a scheme representing the structure of the VGG-LSTM network:



5. Experiment

We decided to implement two VGG8-RNN networks: the VGG16 would have been too expensive from a computational point of view because of his very high number of layers and consequently of parameters.
Here a summary of the structure of the VGG8-RNN model we built:

- **4 convolutional blocks**: each block is composed by a couple of convolutional layers with the same number of units (64 for the first couple, 128 for the second one, 256 for the third one and 512 for the last one) and a pooling layer (we chose to work with a max-pooling layer);
- **a RNN layer** that exploits the time connection between the frames in a sequence and gives a sense of continuity among them;

- **4 fully connected blocks**: each block is composed by a dense layer (whose numbers of units are mirror-shaped with respect to the one in the convolutional block: the first layer has 1024 units, the second one 512, the third one 256 and the last one 216) and a dropout layer to avoid overfitting;
- **a dense layer** for the final classification among the ten classes, using the softmax as activation function.

We decided to use in all the layers (both the convolutional and the dense ones) the ReLu as activation function.
We performed two different models: a VGG8-GRU network and a VGG8-LSTM network.
After some trials we decided to use the following hyperparameters:

- 200 epochs: we tried first with a lower number of epochs, but we saw that there was an important improvement of the accuracy after ≈ 120 epochs. So we decided to augment this hyperparameter, taking into account that a number of epoch too high would bring an overfitting;
- optimization method we used was Stochastic Gradient Descent with a learning rate of 0.001 after an unsuccessful trial with Adam optimizer;
- loss function we used was categorical crossentropy;
- measure we used was the catgorical accuracy.

5.1. Results

The results we got are the following for the comparison between VGG8-LSTM and VGG8-GRU with normal frames:

Model	Accuracy on test set
VGG8-GRU	16.67%
VGG8-LSTM	12.5%

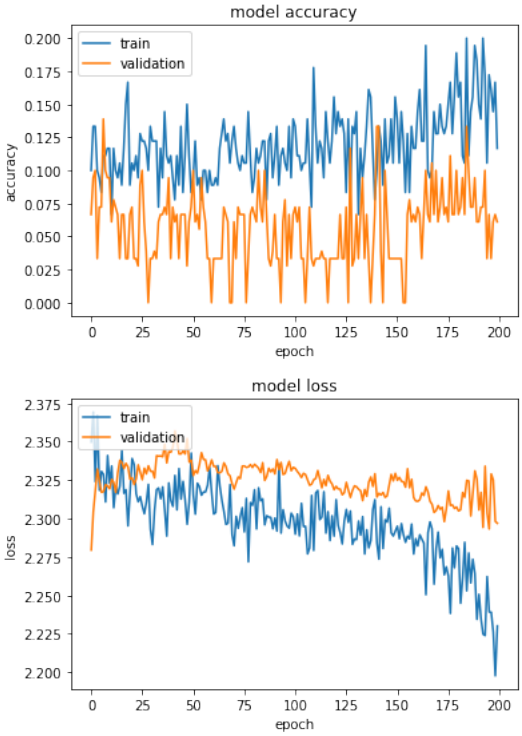
Otherwise, when making the same comparison over the Farneback frames we achieved these results:

Model	Accuracy on test set
VGG8-GRU	4.17%
VGG8-LSTM	12.5%

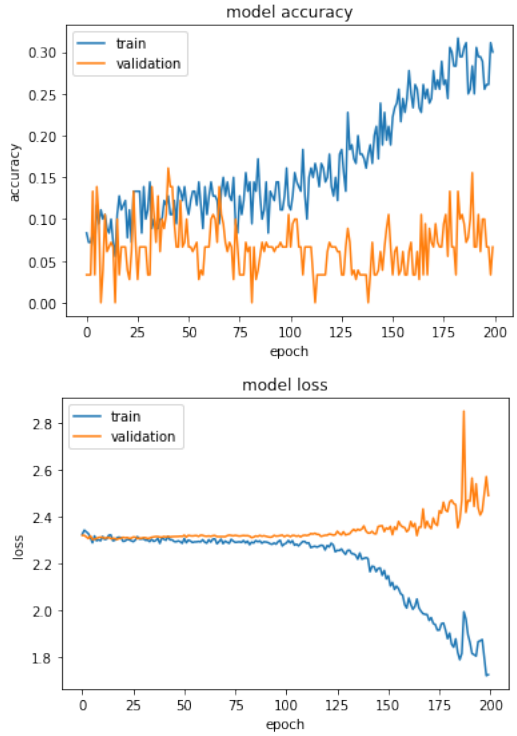
The last comparison was made over the frames where we overlapped the normal ones with the Farneback ones. Here the table containing the results:

Model	Accuracy on test set
VGG8-GRU	4.17%
VGG8-LSTM	16.67%

Here the graphs of the increasing of the accuracy and the loss decay among all the 200 epochs for the VGG8-GRU network with the overlapped frames:



And here the same graphs, but this time for the the VGG8-LSTM network:



The volatility of the graphs is big, and this suggests that are not very reliable to make conclusions.

5.2. Discussion of the results

If we make a comparison between the three ways of filtering the frames, we can clearly see that there isn't a huge difference between the three methods. We have slightly better perfomances when dealing with Farneback and with overlapped frames than when dealing with normal ones, and this is due to the fact that in these two type of frames there's a clear evidence of the movements that can help the neural network to discover the crucial points on which building the learning phase.
It's much more interesting to compare the performances of the VGG8-GRU and of the VGG8-LSTM networks, because the results show the huge difference between the two approaches. This is a very nice result, because in literature there's no evidence that one layer is always better than the other, so these percentages suggest us that for a problem like this (when we have to keep memory of long sequences of frame to detect one word) the LSTM approach is the best one.

6. Conclusion

If we watch the results that we achieved in an absolute way, we have to say that we didn't get good results at all because the percentages of accuracy that we reached are very low.
Anyway, when dealing with a machine learning problem it's always important to contextualize. About that, we have to say that we were forced (by our knowledge and the computational power of our computers) to choose the weakest model between the ones presented in [1], even if we know that the other models suggested would have made better performnces.
Another important issue we have to underline is the nature of our dataset. Also Hamid Reza Vaezi Joze and Oscar Koller agree on the fact (they wrote it in [1]) that the dataset is by one hand free available but, on the other hand, is full of problems. There are in fact many videos with colorful and moving backgrounds (such as some videos for the word "sea", that have some images of waves moving behind), and also many other videos where the person who mimes isn't in front of the camera.
Because of this heterogeneity of the videos it's not a surprise that the accuracy achieved is very low; rather we can say that the results we got are in line with the one achieved by Hamid Reza Vaezi Joze and Oscar Koller in [1] for the same model.

6.1. Further improvements

A very strong improvement to this kind of problem could be achieved if there was the possibility to work with homogeneous videos especially shot for that purpose (for example with the same dark background, the same size, the same length ecc.). Also the size of the dataset should be increased.
Finally, for dealing with this kind of projects it would be better to be in possession of stronger machines that could allow the building of deeper and consequently more powerful neural networks.

References

[1] Hamid Reza Vaezi Joze, Oscar Koller *MS-ASL: A Large-Scale Data Set and Benchmark for Understanding American Sign Language*, Microsoft, 2019.

[2] MS-ASL American Sign Language Dataset download page, *https://www.microsoft.com/en-us/download/details.aspx?id=100121*

[3] Karen Simonyan, Andrew Zisserman, *Very Deep Convolutional Networks for Hig-Scale Image Recognition*, Visual Geometry Group, Department of Engineering Science, University of Oxford, 2015.

[4] Leslie N. Smith, Emily M. Hand, Timothy Doster *Gradual DropIn of Layers to Train Very Deep Neural Networks*, Naval Research Laboratory with University of Maryland, 2015.

[5] Pooya Khorrami, Tom Le Paine, Kevin Brady, Charlie Dagli, Thomas S. Huang, *How Deep Neural Networks Can Improve Emotion Recognition on Video Data*, 2016.

[6] Xi Ouyang, Shigenori Kawaai, Ester Goh, Shengmei Shen, Wan Ding, Huaiping Ming, Dong-Yan Hang *Audio-visual emotion recognition using deep transfer learning and multiple temporal models*, 2017.