Applied Deep Learning

Project topic

The standard way humans interact with machines is based on the usage of devices like a keyboard, a mouse or a gamepad.

In order to enhance the way the human-to-machine interactions occur, new methods have been developed: a consolidated example is voice interaction by means of devices like Amazon Alexa.
Another type of interaction can be movement-based, captured by a standard RGB camera or by depth sensors like Kinect.

This project is a *Computer Vision* project covering the task *object classification*. More precisely, it focuses on the static gesture classification problem with the idea of creating a simple HMI - Human Machine Interface - that maps hand postures to user-defined commands.

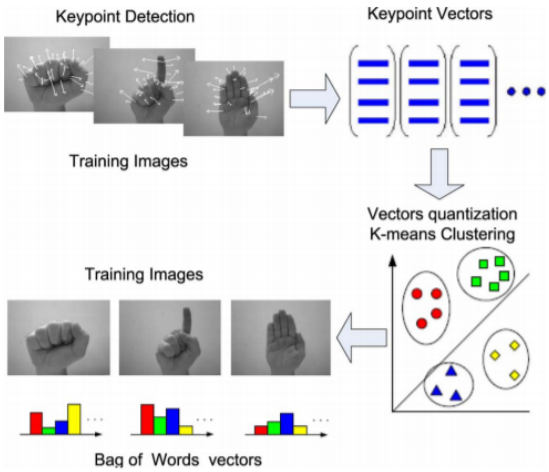
Project type

The project type is a mixture of mainly two categories: *bring your own data* and *beat the classic*, however, I have chosen the **Beat the classic** type.

The traditional approach with which I compared my method is *Bag of Visual Words* (also called Bag of Features) paired with *SVM*.
I was interested in comparing this approach with deep learning because I have already used BoF+SVM during my university studies.

Bag of Features extract features using one of the many available methods (e.g. SIFT, SURF, ORB, ...) to build a vector representation of an object. This vector

is then given as input to a classifier (like SVM or even a neural network) to detect the class of the object.



The main weakness of Bag of Features is that spatial information is completely lost.

The reference paper where this established method was applied is:

Real-Time Hand Gesture Detection and Recognition Using Bag-of-Features and Support Vector Machine Techniques

Published: 15 August 2011, just one year before AlexNet.

As BoF doesn't provide detection capabilities, they isolated hands exploiting skin-color information extracted from a face detected in the image using the Viola-Jones method.

The metric used to compare the deep-learning method with the BoF+SVM method is *accuracy*:

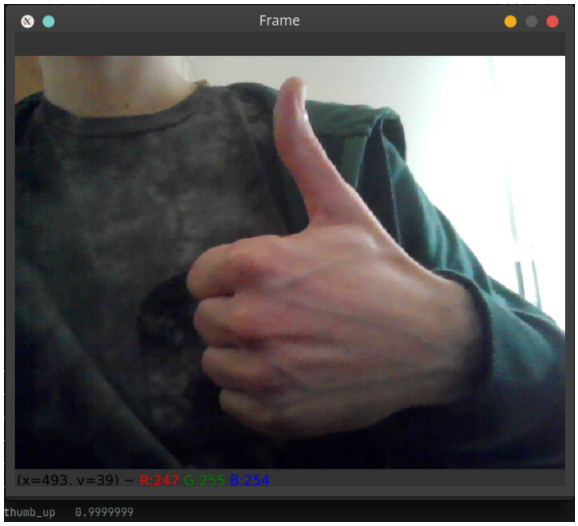
Posture Name	Number of Images	Correct	Incorrect	Recognition Accuracy	Recognition Time (Second/frame)
Fist	97	92	5	94.85%	0.017
Palm	102	100	2	98.04%	0.017
C	112	108	4	96.43%	0.017
V(Two)	95	91	4	95.79%	0.017
Five	134	127	7	94.78%	0.017
Index	119	116	3	97.48%	0.017

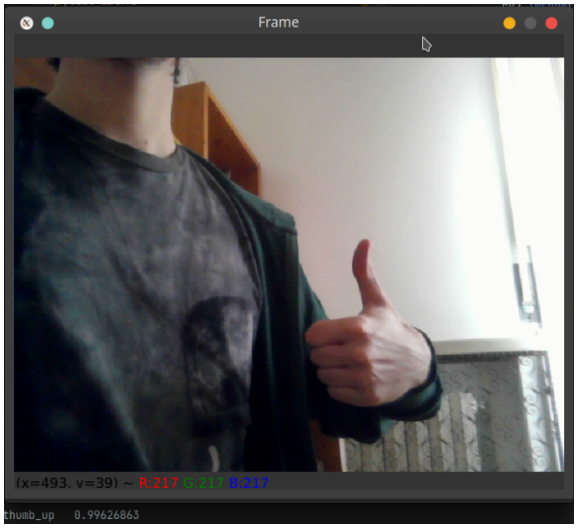
sults

The (average) accuracy of the BoF+SVM approach is 96.23%, that is my target value.
However, the resulting accuracy is 99.83%, which is a decent improvements if compared to the classic solution for this problem.

The reference approach claims to be scale invariant, rotation invariant and lighting change invariant. These properties derive from the choice of using SIFT features.

The deep learning model was trained with hands of different sizes and subject to different illuminations, therefore the approach appears to have the same properties:





(both the hand poses are correctly recognized with >0.99 accuracy)

In term of complexity (e.g. number components, preprocessing) the deep learning approach is much simpler and easier to setup, while obtaining better results.

Running on my CPU (i7 8570h) the processing time for a single frame is ~80ms, while the classic approach requires about 60ms per frame. However, using the appropriate hardware (i.e. a GPU), the processing time can be reduced to about 20ms.

Project idea & Approach

The project is based on the *hand posture classification* task (also called static gesture recognition) with the idea of creating a simple HMI - Human to Machine Interface - that maps hand postures to user commands.

In order to run the project on non-specialized hardware and with decent FPS, the solution uses a CNN classifier based on MobileNetV2, which uses *depth-wise separable convolution* to reduce the complexity cost and model size of the network.

The classifier was trained performing *transfer learning*, starting from a model trained on the Imagenet dataset.
The classifier head was replaced with a dense layer using the *selu* activation function followed by an *AlphaDropout* layer.

Application

The final application maps the hand postures found by the network to machine commands.
Static gestures are recognized w.r.t. a threshold, the repetition of a gesture with enough confidence is mapped to a user-defined action.

The commands can be defined by changing the file *commands.json*: the commands are passed to the underlying system in order to be executed.
The default commands make use of xdotool, which can simulate keyboard commands and therefore make use of system shortcuts.

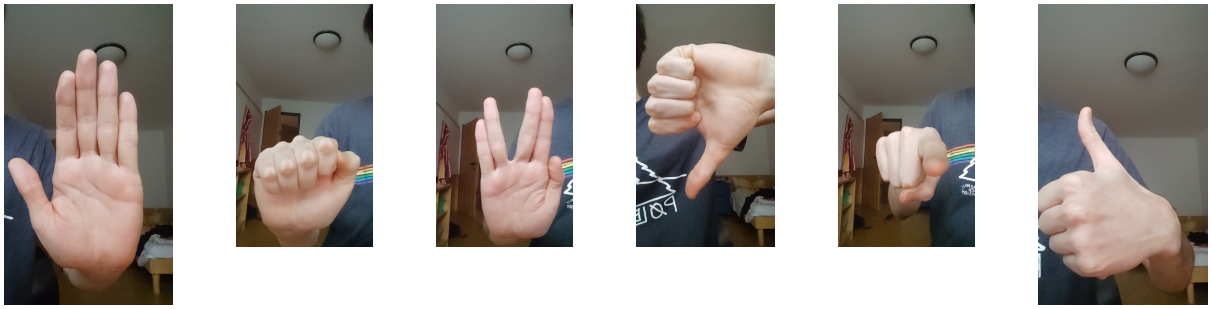
The processing of a single frame is quite fast: it takes ~80ms on a i7 8750h CPU.

Dataset

The model was trained on a dataset I collected. It contains 21k images representing 6 classes: fist, palm, pointer, spok, thumb_down, thumb_up.

To speed up the process, I created a python script that takes as input a video and extracts the frames to add them to the dataset.

The images were collected using two different cameras: my notebook's camera (which is quite noisy) and my smartphone camera.



Retrospective

The initial solution considered using the *Retinanet* object detector in order to extract the hand(s) from the input image and pass them to the classification network.
The object detector was later removed from the architecture due to different problems faced during the development. The main problem that I had was the lack of dedicated hardware (i.e. GPUs) which forced me to use online services like Google Colab and Floyd. Training on Google Colab was infeasible due to the restricted time sessions: to efficiently make use of the GPU the training data had to be loaded into the RAM, but my dataset size was very high and my connection speed limited so I had not enough time for training.
I then tried using Floyd but I faced other problems: the Retinanet implementation that I was using depended on a Keras version that wasn't available in Floyd, so I tried setting up a suitable environment to use the Floyd GPUs. This process was extremely time consuming and didn't lead to the expected result.
After a while, I decided to try changing the implementation I was using, but I was facing the same problems.
As I was spending a lot of time on problems that weren't really Deep Learning-related, I decided to remove the object detector and collect my own dataset of hand poses to train a single CNN classifier.

If I had to do the same project again, I would include an object detector in the architecture, but I would start from a more stable and known implementation or framework. Nevertheless, an object detector would have made the application much more heavy and I wouldn't have been able to run it on a laptop without a GPU. Therefore I should probably buy an external GPU.

Take-aways

- Data augmentation must be controlled: during training, I was making use of "moderate" data augmentation techniques like skewing, rotations, color changes, etc. Besides the moderated augmentation parameters, data augmentation was blocking the network from increasing its accuracy, which was randomly going up and down.
  After reducing the data augmentation impact, the model was able to smoothly learn and reach a good accuracy.
- The usage of dropout made the inference more "stable": the initial model was using a simple dense layer using the ReLu activation function. The validation accuracy was quite high (>99 %) but during real-world inference, I was experiencing some unexpected confident predictions (e.g. without any hand on the image, a class was detected with very high confidence).
  After changing the architecture by using the *SELU* activation function followed by an *Alpha Dropout* layer, I reduced the occurrence of the previously described problem and I got a small accuracy boost.
- The learning rate and the batch size can greatly affect the training: a too-high learning rate was stopping the model from exceeding 98% accuracy.

Time breakdown

- Dataset collection: 8h
- Network design & build: 25h
- Network training & tuning: 20h
- Presentation application: 4h
- Final report and presentation work: 10h

Comparing these numbers with the initial estimate, I spent more time on the dataset collection due to a design change. I also spent a lot of time trying to use the object detector and setting up a working environment.
I really underestimated the time required to setup a working environment. I also underestimated the time required to effectively use deep learning frameworks: despite I theoretically knew what I wanted to do, adapting the

code of the Retinanet to my use-case and solve different versions problem wasn't trivial (I didn't have previous experience with Tensorflow and Keras).