Machine Perception Report

Danill Emstev
ETH Zürich
demtsev@ethz.ch

Emanuele Palumbo
ETH Zürich
palumboe@ethz.ch

Luis Berenguer Todo-Bom
ETH Zürich
beluis@ethz.ch

ABSTRACT

Recognizing individual gestures from video clips is a challenging task that has been the interest of research for computer vision. Borrowing ideas from other works, we used a deep learning architecture based on coupling 3D-CNN with RNN to create a pipeline to analyze videos and accurately recognize gestures. The general idea stems from the work of Molchanov et al. [1], and in addition we tried to substitute the plain 3D-CNN module with the (2+1)D-CNN module proposed in [3]. The CNN module of the architecture was pre-trained on the Sports-1M Dataset [2]. We also worked on replacing the RNN module with Attention [4]. We obtained a 0.7047 accuracy on the test set.

1 INTRODUCTION

The goal of this project was to use deep-learning driven models to classify individual hand-gestures from a variety of data sources. Even though we were given multiple data sources (rgb, depth, segmentation masks and skeleton), we decided to only use rgb data, given that we observed that most works only use this information([2], [1]) on related tasks. In particular, we largely took inspiration from the work of Molchanov et. al [1], that tackle the task by using a recurrent 3D-CNN network. We implemented a very similar model and in addition also explored other kinds of approaches, such as attention-based models.

2 MODEL ARCHITECTURE

2.1 Recurrent 3D-CNN

Inspired from Molchanov et. al [1], to tackle the task, we implemented a recurrent 3D-CNN model. Given that we are dealing with videos, which are basically a sequence of frames, it intuitively makes sense to use a combination of CNN and RNN to tackle the task. Indeed, one would like to exploit both the huge power of CNNs to extract meaningful features from image data, and the ability of RNNs to capture temporal dependencies when dealing with sequence data. On top of this, the usage of 3D convolutions instead of 2D convolutions should help the CNN component to model short-term temporal dependencies in addition to the spatial ones. Loosely speaking, the model consists of a 3D Convolutional Neural Network, with a recurrent layer stacked on top, that propagates information through the time-steps. In the following section, we give a formal description of the model architecture in its general form. We then made some design choices such as the type of recurrent cell for the recurrence, or the particular form of 3D-convolution that is used; and these will be further explained later.

Each sample (or video) X is a datapoint

$$X \in \mathbb{R}^{l \times h \times w \times c}$$

where l, h, w, c denote the number of frames in the video and the height, width and number of channels for each frame, respectively. The 3D-CNN, to be able to extract spatio-temporal features, needs to take as input a sequence of frames. It can hence be formalized as a parametrized mapping

$$F_\theta : \mathbb{R}^{s \times h \times w \times c} \to \mathbb{R}^a$$

where a is the representation size of the CNN, and the s parameter controls the number of frames that are input to the network, and hence the horizon of temporal dependencies we would like our CNN to capture. Note that, controlling for the other design choices, an increase in s will lead to an increase in number of parameters. Hence, given a video

$$X \in \mathbb{R}^{l \times h \times w \times c},$$

for each time-step $t \in \{1, ..., l - s + 1\}$ we give a window of s frames Y_t, starting at frame t, as input to the CNN, and obtain feature vector

$$z_t = F_\theta(Y_t) \quad \in \quad \mathbb{R}^a$$

Note that an additional dense layer is appended to the output of the CNN, in order to be able to easily control the representation size a. As the recurrent layer we use a single LSTM cell G_η that takes as input z_t and outputs h_t as

$$h_t = G_\eta(z_t, h_{t-1}, c_{t-1})$$

where c_{t-1} is the cell state that propagates over time. Finally, the class-conditional probabilities p_t are obtained via

$$p_t = \text{softmax}(W_0 h_t + b_0)$$

with weights W_0 and bias b_0.

As for the loss, we kept the standard already present in the provided code, using cross-entropy loss with the option of either averaging the loss over the time-steps, or considering only the last one. We opted for the option "average-loss".

2.1.1 (2+1)D Convolution. Tran et al. [3] propose a variant of the usual 3D-convolution layer, called (2+1)D-convolution. It consists in separating the spatial and the temporal convolution operation in two different steps. A usual 3D convolution operation (here for simplicity we talk about a single channel) operates with a filter of size $t_f \times h_f \times w_f$, that are the temporal dimension, the height and the width respectively. Instead we decompose this operation in two convolution operations: the first one with filter $1 \times h_f \times w_f$, and the second one with filter $t_f \times 1 \times 1$. Note that this way we introduce an additional non-linearity, which Tran et al. [3] claim can be useful via increasing the modelling capacity. In addition to this they find evidence that the optimization is easier. The model is shown in Figure 1

2.2 Attention Approach

Given the sequential nature of given data, frames of videos, and the fact that not all frames contribute equally to the classification of the video, we decided to implement the attention mechanism.

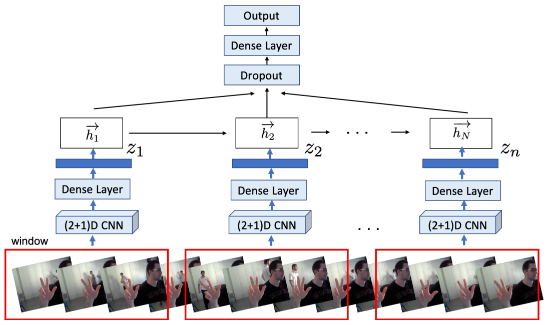

Figure 1: Best Model. The sequence of frames in the video is processed by sliding window and (2+1)D CNN with dense layers to extract a sequence of z_1, z_N features. Then this sequence is given to bidirectional RNN. Dropout with dense layer are applied in the end

.

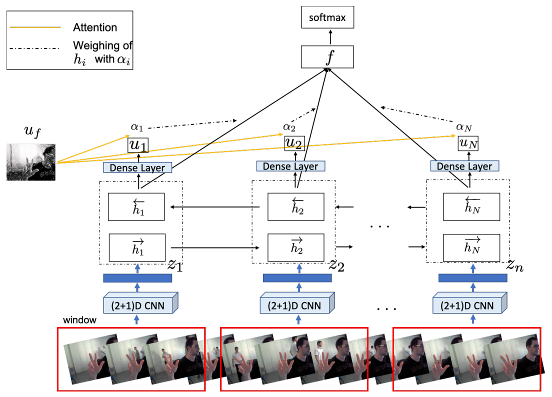

Figure 2: Attention model. The sequence of frames in the video is processed by sliding window and (2+1)D CNN to extract a sequence of z_1, z_N features. Then this sequence is given to bidirectional LSTM. Output states h_1, ..., h_N are given to dense layers to produce features u_1, .., u_N. The frame context feature u_f is learned to produce attention coefficients α_1, α_N The linear combination of α_i and h_i is calculated to receive f. The dense layer with softmax are applied to get the label.

.

We put an analogy between frames and words from NLP and came up with the following methods:

2.2.1 *Hierarchical Attention Network for video classification.*

Inspired by [5] we have adapted their architecture for the sentence level: We first feed frames through (2+1)D CNN layers to get hidden representation vectors z_1..., z_N. Each of these encodes a window of frames, which we will call *clip* in the following. The bidirectional GRU is used to capture information from both time directions of clips and we get clips annotations h_i. The clip annotation h_i is given to a one-layer MLP to get u_i. We measure the importance of the clip as the similarity of u_i with a video context vector u_f and get a normalized importance weight α_i through a softmax function. The video context vector u_f is randomly initialized and jointly learned during the training process. The video context vector u_f can be seen as a high level representation of a fixed query "what is the informative clip" over the clips similarly to those used in document structure with words and sentences. The model is shown in Figure 2

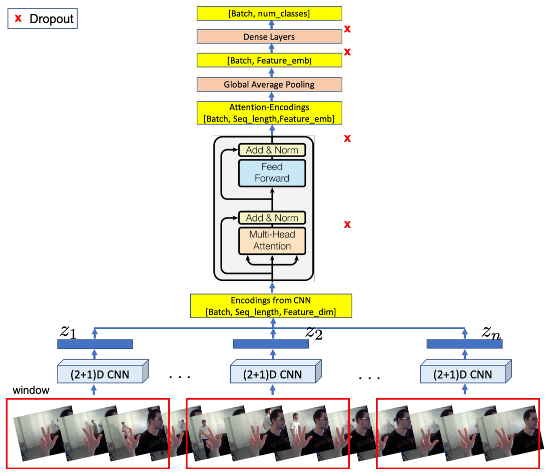

Figure 3: Transformer Model. The sequence of frames in the video is processed by sliding window and (2+1)D CNN to extract a sequence of z_1, z_N features. These features are given as an input to the Multi-Head attention mechanism with residuals and normalisation layers. The updated attention embeddings are processed with average pooling and dense layers in the end. Dropout is applied after Multi-Head attention, attention layer, pooling and dense layers.

2.2.2 *Transformer.*

We replaced RNN with transformer architecture with multihead attention as in [4]. The model is shown in Figure 3 There are most important model parts:

Embeddings By applying 3DCNN (2+1)DCNN layer we obtain hidden representations (embeddings) of clips. These embeddings are inputs to the Multi-Head attention module.

Attention Layer This layer consists of Multi-Head attention part and fully connected layer with residual connections and layer normalizations.

Query/Key/Value For each clip, we create a Query vector Q, a Key vector K, and a Value vector V. These vectors are created by multiplying the embedding by three matrices that we trained during the training process.

Scores for self attention We need to score each clip of the input video against other clips. The score determines how much focus to place on other parts of the input clips as we encode a clip at a certain position. We take softmax of multiplication Query and Key divided by the square root of d_k (the square root of the dimension of the key vectors) multiplied by the value.

Residuals/Normalisation Each sub-layer (self-attention, ffnn) has a residual connection around it, and is followed by a layer-normalization step

Regularisation We use dropout. We apply it after Multi-Head attention and feed-forward layer, in the end after global pooling and after dense layers. We used ratio 0.1

3 DATA & TRAINING

In order to train our models, we utilized two main datasets. The first dataset is the Sports-1M, which consists of 1 million YouTube videos of a variety of sports. This dataset was used to pre-train our (2+1)D CNN model and is explained further in section 3.1. The second dataset is ChaLearn, which contains videos of hand gestures and their respective labels. This is our main dataset on which we wish to evaluate our model, and it was used to train the whole model end-to-end. We explore this dataset in section 3.2.

3.1 Sports 1M - CNN Pre-training

Sports1M is a dataset of 1 million YouTube sports videos we labels for each sport. In Molchanov et. al [1], they pre-trained their 3DCNN module on this dataset, since the whole network is very large and training end-to-end with only the ChaLearn dataset would not yield good results.

Due to the shear size of this dataset, we were unable to fully download it and store it in order to train the model. We devised a workaround solution that is based on the idea of streaming the data. When the first batch of data is requested, it is downloaded directly from YouTube at runtime. Once it is available, it is handed to the training pipeline and a new process is started that will download the next batch in parallel to the training, so that when the next batch is requested, it has already been downloaded and is already available.

We pre-trained the CNN module for only 1 epoch on the entire Sports1M dataset. This model was then saved and loaded for whole model training.

3.2 ChaLearn - Whole Model Training

Our whole model was trained for 20 epochs on the ChaLearn dataset. We performed multiple runs and observed that training for more than 20 epochs would result in over-fitting. We utilized regularization to prevent over-fitting as well as batch normalization. Once we had fully tested and experimented on the validation set, fixed all the parameters and trained on the entire dataset (train + validation).

4 RESULTS

Our best performing model obtained an accuracy of **0.7047** in the test set. Even though this particular task is hard, our results are quite underwhelming. In figure 3 we present the plots from Tensorboard for the same model with slightly different configurations.

The different configurations tested involved including batch normalization, early stopping, adding a few more CNN layers or a few more RNN layers, as well as reducing layers. We obtained very similar results in each run. We did experiment with not using the pre-trained CNN to understand if that was having an effect, but without the pre-training it would take longer to reach the same point. We also experimented with Leaky ReLu and bidirectional RNNs.

In almost all of our runs, are models didn't train very much in the first 2 epochs, leading us to believe we were using a large learning

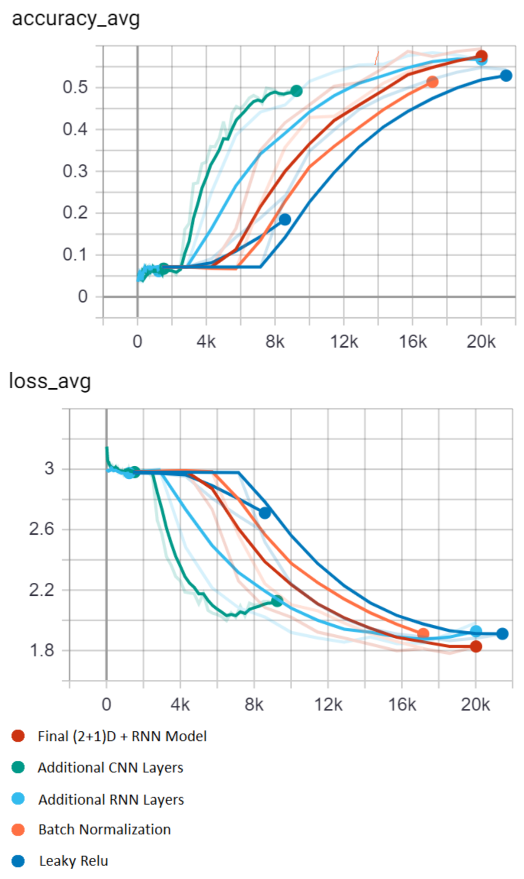

Figure 4: Various training runs with slightly different parameters

rate, as we employed exponential decay and it would only start learning after is had dropped significantly. Unfortunately, reducing the initial learning rate did not yield a significant difference.

Our final training run was using our best configuration and trained on the training data and on the validation data, which increased our accuracy significantly.

We should point out that our model was training on GPUs with 32GB of VRAM. Anything else the model would run out of memory. This suggests that we may have a bug in our implementation that is significantly effecting our results. Unfortunately we were unable to locate any mistakes in our code.

5 CONCLUSION

Despite having modest results, our simpler architecture benefited from not having too many issues with over-fitting, and with more time to understand the influence of training and model parameters, there might be room for improvement. We believe that the use of Attention would be an interesting direction to work on, and would possibly result in a better overall architecture, but unfortunately we were unable to obtain meaningful results with such models as of now.

6 APPENDIX

6.1 Attention model

Here we present a more comprehensive description of our algorithm with formulas we used in our implementation:

- We first feed frames (images) through 3dcnn layers to get hidden representation vectors $f_{t+1}..., f_{t+N}$.
- We then use a bidirectional GRU to capture information from both time directions of frames and we get frames annotations h_i

$$\overleftarrow{h_i} = \overleftarrow{GRU(z(i))}, i \in [1, N]$$
$$h_i = \overrightarrow{GRU(z(i))}, i \in [N, 1]$$

- We feed the frame annotation h_i through a one-layer MLP to get u_i

$$u_i = tanh(Wh_i + b)$$

- we measure the importance of the frame as the similarity of u_i with a video context vector u_f and get a normalized importance weight α_i through a softmax function. The video context vector u_f is randomly initialized and jointly learned during the training process.

$$\alpha_i = \frac{exp(u_i^T u_f)}{\sum_i exp(u_i^T u_f)}$$

- Apply softmax function to the output to get a final prediction

$$f = \sum_i \alpha_i h_i$$

6.2 Transformer model

We take softmax of multiplication Query and Key divide by the square root of d_k (the square root of the dimension of the key vectors) multiplied by the value.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V = z$$

For multiple heads H_0, ..., H_n we concatenate z_0, ..., z_n

6.3 Further Ideas for Attention Current Limitations

Our attention and transformer models work locally. On the small Sports1M dataset transformer gives 0.93 accuracy at the validation. Unfortunately, while running on a cluster accuracy is lower than our best model. We have experimented just with 2 heads given the lack of time. We also did embeddings of every frame using just 2D-CNN instead of sliding window and spatial-temporal (2+1)D-CNN. x

REFERENCES

[1] Pavlo Molchanov Xiaodong Yang Shalini Gupta and Kihwan Kim Stephen Tyree Jan Kautz. 2016. Online detection and classification of dynamic hand gestures with recurrent 3d convolutional neural networks. In *CVPR*.

[2] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. 2014. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 1725–1732.

[3] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, and M. Paluri. 2018. A Closer Look at Spatiotemporal Convolutions for Action Recognition. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 6450–6459.

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[5] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical Attention Networks for Document Classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, San Diego, California, 1480–1489. https://doi.org/10.18653/v1/N16-1174