Dynamic Gesture Recognition
Project Report

Daniel Benesch
beneschd@student.ethz.ch

Christian Bohn
cbohn@student.ethz.ch

ABSTRACT

In this project, we demonstrate a deep learning approach to classifying video data of human subjects performing a variety of hand gestures. Building on previous work we combined state-of-the-art methods, analyzed them, and experimented with our own ideas on each part of the deep learning pipeline. We particularly focused on constructing architectures that are able to leverage the given input modalities as much as possible, on preprocessing that supports those architectures, and on regularization in order to mitigate overfitting. Our core architecture consists of a Convolutional Neural Network (CNN) with temporal convolutions whose output is fed into a Recurrent Neural Network (RNN). In order to leverage the different strengths of our individual networks, we produce our final predictions by applying a voting procedure on the outputs of our best 9 models.

Keywords: Gesture Recognition, Deep Learning, ChaLearn

1 INTRODUCTION

For this project, we had the task of classifying video data of 50 to 150 frames per sample to 20 classes of hand gestures. To this end, we were provided with multiple input modalities describing the scene: For each frame of 80 by 80 pixels, we had access to an RGB image, the pixelwise depth data, a segmentation mask for removing the background, as well as the positions and orientations of 20 joints in the subject's skeleton. Our training, validation, and test sets consisted of video samples from the Montalbano dataset that was introduced in the ChaLearn LAP 2014 Challenge. Its videos depict a vocabulary of 20 Italian sign gesture categories. The evaluation metric was classification accuracy.

2 ARCHITECTURE

We implemented three main models as outlined in the following sections.

2.1 Recurrent Temporal CNN Model (RTCNN)

A visualization of this architecture can be seen in Figure 1.
As input to our architecture, first the RGB and depth modalities of the video data get concatenated together along their last dimension resulting in a tensor with four channels: The R, G, and B color channels, as well as the depth data for each pixel. This tensor is then fed into the initial layer of our CNN. In each layer, we first compute a 3×3 spatial convolution followed by a temporal convolution over the previous, current and subsequent frame for each pixel. This corresponds to the (2+1)D convolution approach of [3]. After both convolutions, we perform batch normalization. At the end of each convolutional layer, the frames get downsampled using either maximum or average pooling with a pool size of 2×2. This reduces both spatial dimensions of the data tensor by $\frac{1}{2}$. Overall, our CNN contains 5 such layers, each doubling the number of channels in the tensor: The 1st layer produces an output with 16 channels, the 2nd with 32, the 3rd with 64, the 4th with 128, and the 5th with 256. The output of the last convolutional layer then is passed through one dense layer to produce a vector representation of 256 entries for each frame in a video. These vectors form the inputs to the RNN whose bidirectional LSTM cells and another dense layer produce 20 logits for each frame, corresponding to the 20 gestures we seek to classify. We left the provided RNN model largely unchanged as it already provided strong performance. We experimented with a multi-layer RNN and a larger intermediate representation size, but were not able to achieve better results with that. We ended up exchanging the RNN cells for bidirectional LSTM cells.
To create a final label for a video, we average the logits of the individual frames in an element-wise manner and apply the Softmax function to the vector of averaged logits, producing a vector of valid probabilities. The predicted label then is the *argmax* of the resulting vector.

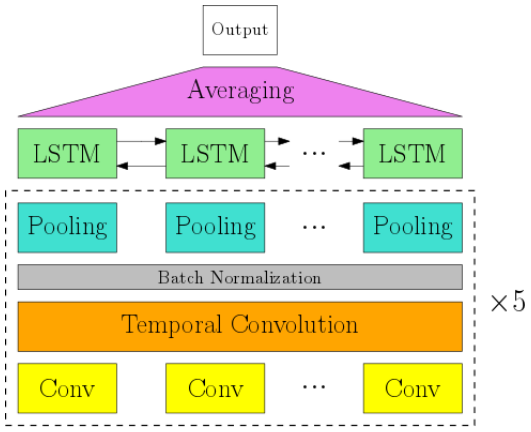


Figure 1: RTCNN Model Architecture

Residual Version: Additionally, we also implemented a residual version [1] of the above net using residual connections from the input to the 3rd layer and the 3rd to the 5th layer of the CNN. That did not yield any significant improvement, however.

2.2 Combined Model Integrating Skeleton Data

This model combines a CNN followed by an RNN for RGB and depth data with an RNN model for the skeleton joint positions:
For RGB and depth data, we use the CNN-RNN model, as described above. Additionally, we feed the 3D joint positions and orientations from the skeleton data directly to the RNN model.
We then concatenate the outputs of these two models before feeding

them into a dense layer which computes the logits for each frame of the video.

2.3 Voting model

For our final prediction, we combined several of our various models. We let each model $m \in \{1, 2, \ldots, n\}$ predict the probabilities of a video belonging to label $i \in \{1, 2, \ldots, 20\}$. In particular, we chose 9 models that provided good performance. These models differ from each other in their choice of architecture (simple RTCNN, Combined Model, or Residual Version), their dropout rates, in the extent of augmentation, as well as their loss functions. Each model selects the label with the maximal probability as its prediction. Finally, a voting procedure on the predicted labels of the n models is performed in order to arrive at a final prediction (the most often predicted label gets selected).

3 PREPROCESSING

3.1 Centering the Videos

When inspecting the provided videos, we noticed that the subjects are not always in the center of the frame and that the distance to the camera is not uniform across all videos, either. We conjectured that this probably negatively affected the performance of our model. Hence, with the help of the skeleton data we cropped out a square around the subjects' head and hip positions from each image and rescaled it. Preprocessing the video frames in this way yields videos that are perfectly centered around the subject and sized appropriately depending on their height and distance to the camera. Using this preprocessing function resulted in a significant increase in accuracy on the public test set. A similar preprocessing function is used in [2].

3.2 Augmentation

We also experimented with augmenting our input data: We changed brightness, contrast, hue, and saturation using random parameters, and, again following [2] also rotated, sheared, and translated the images randomly.
We applied augmentation in an online fashion. The benefit of this is that in each epoch the training data of the model is new and potentially unseen to the model. For advanced augmentation, we also experimented with the *imgaug* library in an offline fashion.
Despite the fact that augmentation worked well in [2], in our case, the effect of using augmentation was negligible, so some of the predictions used in our final bagged submission include augmentation while others do not.

4 EXPERIMENTS

Over the course of the project, we tried to improve our performance using a wide variety of approaches: In part, those are standard deep learning techniques, others are our own original ideas that went beyond. In order not to exceed the page limit of this report too much, we cover the standard approaches quickly and spend more time on the aspects that we consider particularly special.

4.1 Different Loss Functions

The default loss function provided to us in the code skeleton was cross entropy:

$$CE(D) = \sum_{i=1}^{n} CE(D, i) = -\sum_{i=1}^{n}\sum_{c=1}^{K} y_{i,c} log(\hat{y}_{i,c}) \tag{1}$$

where D stands for a dataset, n is the number of samples, K denotes the number of classes, $y_{i,c}$ is 1 iff $y_i = c$ and 0 otherwise, and $\hat{y}_{i,c}$ is the model's prediction that sample i belongs to class c. Let furthermore, $L(i)$ denote the true label of sample i. In our scenario, where each sample can have exactly one true label (1) reduces to

$$CE(D) = -\sum_{i=1}^{n} \underbrace{y_{i,L(i)}}_{1} log(\hat{y}_{i,L(i)}) = -\sum_{i=1}^{n} log(\hat{y}_{i,L(i)}).$$

Observe, that thus the per-sample loss depends only on the model's predicted probability of sample i belonging to its true class $L(i)$. This is often wasteful when the evaluation metric is accuracy. Usually the final prediction gets obtained from the model's predicted probabilities by selecting the class with the highest predicted probability. Let $w(i) = \underset{c \in \{1,2,\ldots,K\}}{argmax} \{\hat{y}_{i,c} | y_{i,c} = 0\}$, i.e. the wrong label for sample i with the highest prediction score. Now consider 2 scenarios:

(1) $\hat{y}_{i,L(i)} = 0.48$, $\hat{y}_{i,w(i)} = 0.45$
(2) $\hat{y}_{i,L(i)} = 0.3$, $\hat{y}_{i,w(i)} = 0.1$

In the first setting, the model has some difficulty deciding which of the 2 classes is the correct one. In the second setting, the decision of the model is clearer. Although the prediction for the true class is higher in the first scenario, we as data scientists would clearly prefer the second scenario. This thought experiment shows that purely focusing on a high $\hat{y}_{i,L(i)}$ is not what we aim for. Instead, we would also like the difference $\hat{y}_{i,L(i)} - \hat{y}_{i,w(i)}$ to be as large as possible. Thus, we should encode this in our loss function by also penalizing high predictions for wrong class labels. We achieved this by using the *logloss*:

$$Logloss(D, i) = -\sum_{c=1}^{K} y_{i,c} log(y_i, c) + (1 - y_{i,c}) log(1 - y_{i,c})$$

or the *hinge loss* which is known to be a large-margin classifier (what we aim for).
Although theoretically sound, using these loss functions did not increase our performance significantly.

4.2 Adapting Sequence Mask

As stated in chapter 2 we use bidirectional LSTMs. Here, we can make use of information based on previous and future frames. However, the first and last n frames might not get enough information from each of the 2 directions to classify the frame correctly. Thus, we adapted the sequence mask such that the first and last n logits get masked out. This approach was inspired by [2].

4.3 Regularization

We implemented almost all regularization techniques we learned about in the lecture:

- Normalization
- Batch Normalization
- Dropout
- Early Stopping
- Data Augmentation (offline and online)
- Regularization terms for weights
- Gradient clipping

We invested a lot of time in data augmentation. In the interest of remaining within the page limit we won't list them all here. We want to point out though, that we also precomputed the skeleton images and trained our models with this input data. However, the performance did not increase over the model used with RGBD modalities.

Implementing all of these took a lot of time. To our surprise and frustration, almost none of the techniques had a positive effect. Only early stopping and batch normalization increased our generalization performance.

4.4 Miscellaneous

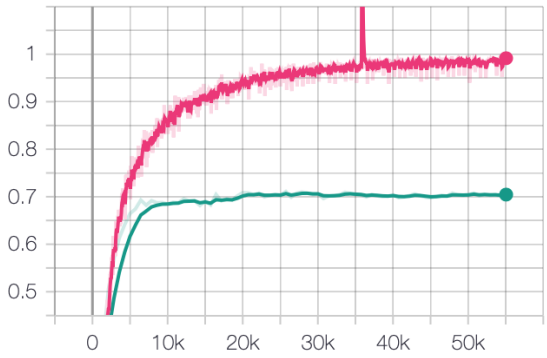
Due to their structure, dilated convolutions [4] [5] allow a big receptive field while keeping the number of parameters small. Surprisingly, their implementation proved challenging, since the height and with of the final resulting representation had to be computed manually.
We also tried to use different validation splits since the initial validation split seemed to be very odd (the performance on the validation set and the test set differed vastly and using the additional validation data for training decreased test performance). However, this also gave the same results.
Additionally, we experimented with GRU cells instead of vanilla LSTMs and tried various activation functions. Leaky ReLU gave us the best performance. Furthermore, we also tuned basic parameters like the batch size and the learning rate.

5 RESULTS

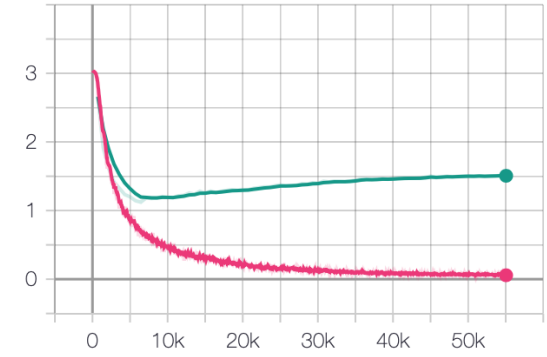
A comparison of the results for different models can be found in Table 1. It indicates whether a model uses temporal convolutions, batch normalization, whether the video data gets centered, and if it performs voting using the outputs of multiple other models. For each model we report the accuracy on the validation data (where available) and on the public test data. We chose to only show properties that achieved a significant performance increase over their predecessors. Since e.g. our architectures all performed quite similar to each other and dropout did not make a vast difference, we did not include the concrete values in the table.

5.1 Remarks about overfitting

In all our training runs we noticed that we reach the optimal validation accuracy after the validation loss has assumed its minimum. To see this, take a look at figures 2a and 2b. The optimal loss gets obtained roughly around step 6000. However, at this point, the accuracy lies at $\approx 68\%$, while the optimal accuracy was $\approx 71\%$. We observed this behavior consistently. Thus, we decided to use early stopping with regards to validation accuracy and not to validation loss. As elaborated in chapter 4.1, this behavior (accuracy rises while cross entropy loss rises) is not odd and can be attributed to a suboptimality of the cross-entropy loss function for accuracy.



(a) Training and validation accuracy



(b) Training and validation loss

Figure 2: Metrics of a typical training run

All our models exhibit a very large discrepancy between validation- and test accuracy on the public test set. As can be seen in table 1, for some models, this discrepancy is as high as 16%.
Moreover, using the training and validation splits for training our model consistently yielded worse performance than training the model just on the training set. This was surprising to us. If several models consistently perform better by not using $\approx$ 2000 out of $\approx$ 7000 training samples, then this is suspicious. We tried to exchange parts of the validation set with the training set but that gave us roughly the same outcomes. We often experienced the case that intuitively good ideas lead to no improvement on the validation set, but to a significant performance gain on the test set.

Thus, the situation is quite dissatisfactory. We had to choose our best model to a large extent based on the public test score (of course we chose no model that has a particularly bad validation performance) because the validation set seems to be differently distributed. This goes against our intuition as data scientists. Thus, obviously, our project is at some risk of overfitting although we implemented almost all regularization concepts presented in the lecture (see section 4.3) and tried to build models with . However, we did not have much choice other than using regularization and going for the models that perform well on the public test set because the validation scores were not too useful.

Model	TempConvs	Batchnorm	Centering & Resampling	Voting	Val Accuracy	Test Accuracy
Basemodel	✗	✗	✗	✗	n/a	0.494
#2	✓	✓	✗	✗	0.689	0.804
#35	✓	✓	✓	✗	0.722	0.880
Votingmodel	✓	✓	✓	✓	n/a	**0.928**

Table 1: Comparison of the results for different models

Appendices

A FURTHER IDEAS TO IMPROVE THE PERFORMANCE

One option is to combine the different model architectures in a different way than we did with the voting. For our best performing approach, we use the idea described in section 2.3. This approach has the slight weakness, that in case of close decisions (e.g. the most likely label has probability 0.3, second-most likely label has probability 0.29) just the most likely label gets chosen. Therefore, after selecting the most likely label, the granularity about whether it was a close or a very clear decision is lost. A more promising approach would be the following:

(1) Let each model produce predicted probabilities for each label.
(2) $\forall i \in \{1, 2, \ldots, 20\}$: Average the predicted probability for label i across the n models.
(3) Select the label with the highest probability.

This strategy would treat close decisions more appropriately. Since we had this idea quite late in the project development process, the time would not have sufficed to retrain the individual models in order to produce their prediction probabilities.

We also argue that another big performance leap might be possible by experimenting even more with the validation set: For our architectures, the models that were trained with training and validation data performed consistently worse than the ones trained just on training data. It seems wasteful not to use those 2000 samples for training the final prediction. Thus, we think that by finding configurations that can use the validation set in a positive way, one could achieve higher scores.

REFERENCES

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
[2] Lionel Pigou, Aäron Van Den Oord, Sander Dieleman, Mieke Van Herreweghe, and Joni Dambre. 2018. Beyond temporal pooling: Recurrence and temporal convolutions for gesture recognition in video. *International Journal of Computer Vision* 126, 2-4 (2018), 430–439.
[3] Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. 2018. A closer look at spatiotemporal convolutions for action recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 6450–6459.
[4] Fisher Yu and Vladlen Koltun. 2015. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122* (2015).
[5] Fisher Yu, Vladlen Koltun, and Thomas Funkhouser. 2017. Dilated residual networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 472–480.