A Network a Day Keeps The Boredom Away - Application of Human Action Recognition on the Breakfast Dataset

Li Fengzi A0206468Y
Xiuping Hua A0206514N
Shi Haohui A0206537E
Jonathan Simon Wagner A0152784X

April 14, 2020

1 Introduction

The recognition of human actions is a crucial aspect of computer vision. Current researchers are faced with a lot of ambiguity though since the human body can move in various ways and the same action can also be performed in multiple different variations. Overall, the field of human activities can be grouped in three categories

Gestures which is a body action trying to convey a message. Examples are the worldwide recognized sign of the thumbs up as well as other commonly known messages conveyed using hands, face or other body parts.

Action representing physical body movements of a single or multiple persons.

Interaction where multiple people - but at least two - perform an action that is directed at each other such as shaking hands or having a conversation.

This project is focused on the second category, namely the human action recognition. For this, the Breakfast Dataset was leveraged for training and testing a classifier, labeling sequences of human actions related to breakfast preparation with the correct description of the current task. Individual videos have an overall action label and can itself consist of multiple sub-actions. The number of sub-actions per video varies.

This paper displays the project flow, starting with initial data exploration, explaining some of the model architectures chosen to make accurate predictions, showing several metrics how the best performing model was selected as well as concluding with some action items that could have been done to further increase the accuracy.

2 Data Exploration

The first step of Data Analysis projects is to get an overview of the data that is available, it's shapes and forms in order to fit the models appropriately to it. In this project, we were faced with 1460 training videos showing humans performing breakfast preparation activities. Each video had an overall class describing the main action happening in the video as well as sub-actions with more granular task descriptions. The total number of sub-actions was 47 (excluding the SIL frames where no explicit action is happening). Each video is of different length and can consist of different numbers of sub-actions as well as every sub-action can consist of different amount of frames.

Figure 1 is displaying the varying length of training videos with regards to overall frames, ignoring the different amounts of sub-actions inside the training video. Most training videos fall between 0 and 4000 frames but there are some outliers up to over 10000 frames per video which makes the distribution very skewed. During the modeling process, these varying lengths have to be taken into account when truncating or padding videos to include them in one batch for processing.

Figure 1: Distribution of Training Video length (in frames)

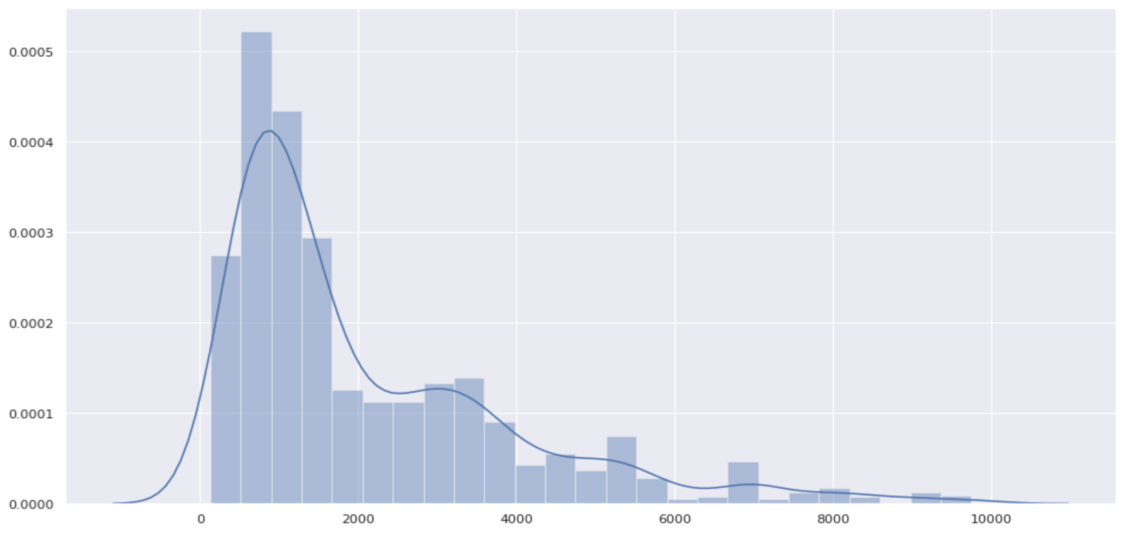
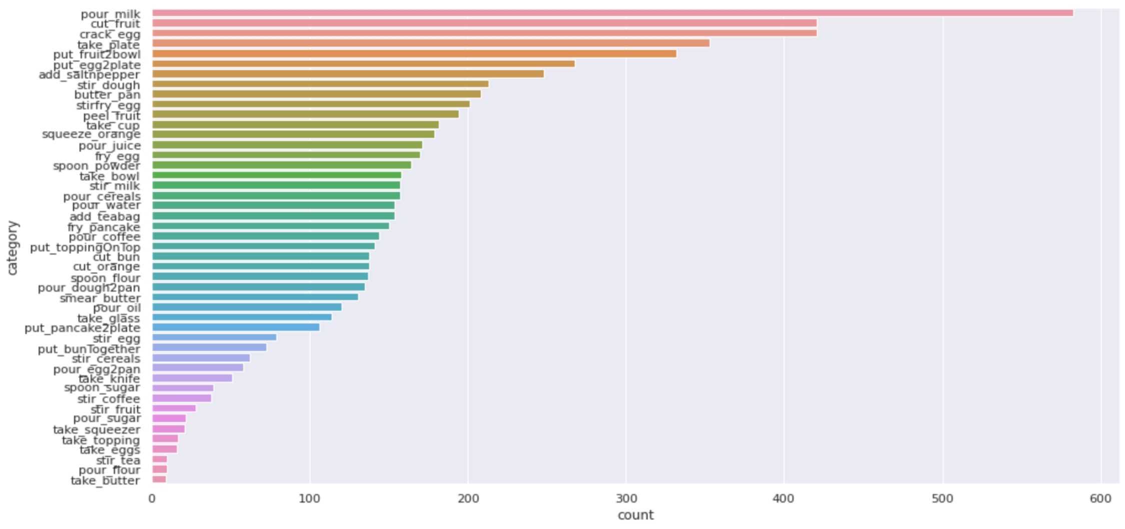


Figure 2 shows the total number of occurrences for each sub-action inside the provided training videos. It can be seen that some sub-actions have a much larger probability of occurring than others. This might lead to a class imbalance that has to be further checked during the modeling process.

Figure 2: Number of occurrences for each sub-action



3 Pre-Processing of Data

Since several differences in length between the different videos, but also between different sub-actions, were identified during Data Exploration, certain pre-processing steps had to be performed in order to be able to feed all training and test videos into the same architecture.

In total, three methods were used to account for this difference, namely *max-pooling*, *average-pooling* and *frame difference*. These methods are supposed to extract features for each segment in all videos and test their performance utilizing a baseline model. It was shown that max-pooling is performing the best.

To prepare the data for the modeling as well as for validation purposed, we split the dataset into training, validation and testing set with a 8:1:1 ratio. The maximum number of segments in one video is 23 and 16, for the training and validation dataset respectively. Since different length of videos could not be fed into neural network models, we decided to pad several 0 arrays with the shape (1,400) at the end of each video to make each video contain a total of 28 segments.

4 Considered Model Architectures

In the following chapters, the considered model architectures are further explained together with their respective accuracies. Each architecture has its own advantages and drawbacks.

4.1 Multi-layer LSTM Model

To start with, we used a multi-layer LSTM model as our baseline model. The model summary with each layer is shown in Table 1. Softmax is used in the output layer for the multi-classification problem and the number of nodes is 48 because we have total 48 classes including SIL. We choose Adam as our optimizer, the train batch size is 400 and the optimal number of epochs after early stopping based on the validation data was 103.

Table 1: Multi-layer LSTM Model Summary

Input Shape: (1460, 28, 400)		
Hidden Layer (type)	**Output Shape**	**Param #**
LSTM	(None, 28, 2048)	20062208
Dense	(None, 28, 1024)	2098176
Dense	(None, 28, 256)	262400
Dense	(None, 28, 128)	32896
Dropout 0.1	(None, 28, 128)	0
Dense	(None, 28, 48)	6192
Total params: 22,461,872		

4.2 CNN-LSTM model

The second considered model architecture is a CNN-LSTM model and involves using Convolutional Neural Network layers for feature extraction on input data combined with LSTMs. The model summary is in Table 2 and the optimal number of epochs is 91 after considering early stopping.

Table 2: CNN-LSTM Model Summary

Input Shape: (1460, 28, 400)		
Hidden Layer (type)	**Output Shape**	**Param #**
Conv1D	(None, 28, 32)	3739648
Maxpooling1D	(None, 28, 32)	0
Dropout 0.2	(None, 28, 32)	0
LSTM	(None, 28, 2048)	17047552
Dense	(None, 28, 1024)	2098176
Dense	(None, 28, 256)	262400
Dense	(None, 28, 128)	32896
Dropout 0.1	(None, 28, 128)	0
Dense	(None, 28, 48)	6192
Total params: 19,485,648		

4.3 Bi-Directional-GRU Model

The third model is a Bi-Directional-GRU model which contains two Bi-Directional-GRU layers and one Time Distributed Dense layer. Compared with Multi-layer LSTM, this model achieves a much better test accuracy since it has the power to capture more temporal dynamics of video frames. The detailed model summary is in Table 3. The optimal number of epochs is 81.

Table 3: Bi-Directional GRU Model Summary

Input Shape: (1460, 28, 400)		
Hidden Layer (type)	**Output Shape**	**Param #**
Bidirectional GRU	(None, 28, 2048)	8755200
Bidirectional GRU	(None, 28, 2048)	18880512
Time Distributed (Dense)	(None, 28, 48)	98352
Total params: 27,734,064		

4.4 Bi-Directional-LSTM model

The Bi-Directional-LSTM model which contains two Bi-Directional-LSTM layers and one Time Distributed Dense layer was also applied and achieves similar results as the Bi-Directional GRU model introduced in the previous chapter. The model summary is in Table 4 and the number of epochs is 88.

Table 4: Bi-Directional LSTM Model Summary

Input Shape: (1460, 28, 400)		
Hidden Layer (type)	**Output Shape**	**Param #**
Bidirectional LSTM	(None, 28, 1024)	3739648
Bidirectional LSTM	(None, 28, 1024)	6295552
Time Distributed (Dense)	(None, 28, 48)	49200
Total params: 10,084,400		

4.5 Hyper-parameter Tuning and Model Regularization

The following regularization methods were used in all models explained in the previous chapters to avoid overfitting. In model architecture building aspect, we add dropout layers to Multi-layer LSTM model and CNN-LSTM model, set the dropout rate to 0.08 in the Bi-Directional GRU model and Bi-Directional LSTM model and search for the best number of hidden nodes through hyper-parameter tuning. During the training process, early stopping and learning rate adjusting were applied as well according to the development in validation loss over the epochs.

5 Model Evaluation

The test accuracy of each model is shown in Table 5.

Table 5: Test Accuracy of Models

Model	**Multi-layer LSTM Model**	**CNN-LSTM Model**	**Bi-Directional GRU Model**	**Bi-Directional LSTM Model**
Test Accuracy	0.6614	0.6529	0.8976	0.9018

As explained in the chapter about Data Exploration, there was a significant class imbalance when looking at the individual sub-actions. Usually, a classification report together with a confusion matrix would be used to identify any prediction imbalances that can be targeted. Since the test accuracy for the models reported above is quite close to perfect accuracy, only some sub-actions were looked at, that exhibited very low frequency in the training data, namely take_butter, stir_fruit and take_eggs. The classification reported for these classes is displayed in Table 6.

Table 6: Classification Report Extract

Sub-Action	Precision	Recall	F1-Score	Support
take_butter	1	1	1	1
stir_fruit	0.5	0.67	0.57	3
take_eggs	1	0.33	0.5	3

It can be seen that for take_butter, the classification is perfect with a precision and recall of 1.0. However, the support and hence total number of occurrences in the groundTruth is only 1, meaning there was only one occurrence that needed to be classified correctly in order to get a perfect accuracy.

For the other two sub-actions in Table 6, the recall is quite low compared to the overall model accuracy. This would indicate an either lower number of true positives or higher number of false negatives. Since the precision for take_eggs is at 1.0, it can be inferred that the reason for a low recall score is the high number of false negatives. The model therefore is not able to reliably predict this class and "misses" occurrences of these actions in the test data. This seems logical looking back at the considerable class imbalance present in the training dataset.

There are several ways to overcome this issue which have not been applied for this project. The standard ways to deal with imbalanced datasets are undersampling, oversampling or the more modern solution of SMOTE (Synthetic Minority Oversampling Technique). These techniques could have been applied and any changes in accuracy in classification reports observed.

6 Final Architecture and Kaggle Submission

After submitting the results to the Kaggle competition, we observed considerable overfitting within the Multi-layer LSTM model and CNN-LSTM, hence we did not go ahead with their refinement and tuning. The two Bi-directional models were able to achieve around 0.76 on their own.

Therefore, we only trained these two Bi-directional models in the later phase and use the whole dataset instead of splitting it to get more train samples and increase accuracy further.

As a last step, we combine the results of two models to build an ensemble model. In this process we choose the best number of epochs through observing the plot of training loss to avoid overfitting.

In the end, the best result comes from the ensemble model and the final accuracy on Kaggle is reported to be 0.82710.

Appendix

Using MLP + KMeans to automatically detect Segment Information

In video action classification modeling, one of the most tedious task is to temporally detect and segment actions in untrimmed videos. Before training models, it is required to define and label sub-action boundaries to create annotations, which is very time consuming.

The objective of this section is to propose a method to automatically detect sub-action boundaries and use the embedding features for action classification. Kukleva et al.[1] proposed to use a continuous temporal embedding of frame-wise features to benefit from the sequential nature of activities. The main idea is to build a 3-layers MLP where two hidden layers with dimensionality 2D and D and the last output layer form the frame time stamp

$$t_{i,j} = i/N_j \tag{1}$$

where: i = Position of a frame in the whole video
N_j = Total number of frames in a certain video

From here, continuous latent features together with relative time dependencies representation of the frames can be captured. The last second layer continuous temporal embedding features are clustered using KMeans, from which segment information can be automatically detected by identifying the cluster group and the mean over time stamps of all frames belonging to each cluster. Therefore, each video segment information can be predicted by using the above training MLP and KMeans model before feeding into RNN models for action classification. In our project, we tried two schemes:

1. Only using KMeans to cluster each video frame and obtain the segment information which are then processed by using max-pooling and subsequently fed into our best model Bi-directional models. **Segment information for training dataset is not used**, however in order to make sure the final number of segment labels for test dataset is 1284, we use the test segment information. The best accuracy for the bonus part comes from the ensemble model combing results of two Bi-directional models and is recorded as 0.42289 on Kaggle.

2. Due to time constraint, we apply the above mentioned MLP for an activity class 'coffee' and obtain the continuous temporal embedding features. Future work could continue on training KMeans using the temporal embedding features for an activity class.

Individuals' Contribution

Li Fengzi Working on Pre-Processing of the data as well as finding the suitable model architecture.

Xiuping Hua Researching existing ways of human action recognition and determining the best suitable model architecture.

Shi Haohui Implementing researched model architectures, tuning the models until perfection and generating scores on Kaggle.

Jonathan Simon Wagner Working on data exploration, finding the suitable model architecture and putting together the report.

[1]Kukleva, Anna and Kuehne, Hilde and Sener, Fadime and Gall, Jurgen, "Unsupervised learning of action classes with continuous temporal embedding", I IEEE Conference on Computer Vision and Pattern Recognition (CVPR'19). 2019