Customer Confusion Detection Using LSTM

Xuecong Tan
Simon Fraser University
Burnaby, BC V5A1S6
`xuecongt@sfu.ca`

Rong Li
Simon Fraser University
Burnaby, BC V5A1S6
`rong_li_2@sfu.ca`

Zhicheng Xu
Simon Fraser University
Burnaby, BC V5A1S6
`maxx@sfu.ca`

Abstract

The shopping experience is one of the most important factors that determines whether customers will regularly come to a store. In some situations, customers may confuse the description with other aspects of a certain product. If the in-store staff cannot help confused customers in a timely manner, they might lose a potential sale. Hence, we need a model to identify confused customers according to their body language and head movement. Our task included the following three steps: (a) obtain body position and movement through camera footage, (b) annotate and classify the data from part a, and (c) train our model by feeding it the annotated data. We utilized different recurrent neural network (RNN) models such as Long Short-Term Memory (LSTM) and Bidirectional LSTM to identify the customers who needed help.

1 Dataset

The annotation process tools and raw dataset were given to us by our project supervisor [6]. We decided to examine the annotation tools implemented by the previous group and developed our version to fit our scenario and development environment. We were extracting frames from 100 shopping scenes, each with three different shooting angles. For consistency, we chose to investigate one camera angle for each scene.

1.1 Extract frames

To extract data from videos that contain footage of customer behaviour, we first used FFmpeg to convert the videos into frames. FFmpeg helped us to extract six frames per second for each video we have. Then, we used OpenPose to process each frame that was extracted from the video to retrieve customer 2d body coordinates. During the time we tested OpenPose, it turned out that the CUDA version we had is not compatible with the OpenPose version we built by utilizing CMake. Hence, we downgraded the CUDA version to enable the GPU power for computing the customer 2d body coordinated using OpenPose and the GPU model we used is RTX3080. Since the RTX3080 only had 10GB of GDDR6x memory, the GPU was not able to compute two parallel OpenPose processes at once (one OpenPose process takes around 9GB of GPU memory space). It took us over a week of uninterrupted computing time to finish processing 36776 frames.

1.2 Standard of labels

The three different labels were “uncertain,” “confused,” and “not confused.” Because there were some frames in certain videos that did not contain any customers, we chose to discard those frames by giving them a label called “no tested object.” The frame with the label “no test object” was automatically dropped during the combine features stage. Before we label frames from a certain video, we watched through the video first to obtain a general idea of what was in it to make fewer mistakes during the actual annotation stage.

1.3 Combine features

The last step of the annotation process involved concatenating label customer 2D body coordinates and head orientation data into one JSON file as features for each frame. The head orientation data were given by our project supervisor, representing customer location coordinates and head orientations that were collected by a headwear device. After the concatenate process was completed, we had a total of 60 features per data frame—54 features were customer 2D body coordinates and 6 features were from customers’ head devices.

2 Data preprocessing

Humans infer whether a person is confused by a number of visual cues. Because each data frame does not have information about the time, which could illustrate the series of actions, we decided to modify our data frames to have a time factor. The 60 features for each data frame should also be normalized before passing into the neural network to avoid features with big values dominating other features and shortening the computation time with faster convergence.

2.1 Sliding window

To apply our RNN models, we decided to use sliding windows to preprocess our data as input to give our data a time factor and make them represent many series of movements. Each window contained a series of data frames that reflected certain movements. The window slid a certain amount of data frames until it reached the last data frame from a video. For example, if there were 980 frames from a video, and we slid the window frame-by-frame with the window size set to 10, after the sliding window preprocessing stage, we had 980 segments, each containing 10 frames of data.

To determine the size of such intervals, it was ideal to have videos that each illustrated one action, which was clean and easier to achieve an accurate decision of the confusion level of customers, however, the provided dataset made it difficult to annotate each video separately. The provided datasets only contained 145 sets of shopping scenes, and each shopping scene could have a variety of moments of “being confused” or “being not confused”. It was impossible to label a whole video, unlike the majority of other action detection projects where videos contain single actions [1]. Therefore, we used the previously mentioned sliding window approach instead. We believe that the actions performed by customers within that fixed-sized timeframe should be consistent. Some timeframes may get chopped exactly at the moment of transition between actions, but these cases are few and barely influence the training process.

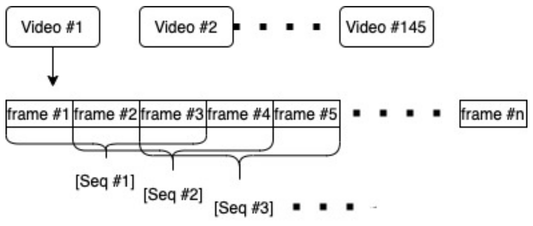

Figure 1: Sliding window approach.

2.2 Data normalization

Because the feature that represents the customer's 2D coordinates had a value ranging from 0 to over 1,000, and the feature value of hat orientation was between -1 and 2, and they were two different features, we decided to separately normalize the customer 2D coordinates and hat orientation. Both customer 2D coordinates and hat orientation were normalized into the range between -2 and 2 using MinMaxScaler from the sklearn library.

2.3 Feature selection

We have 60 features in total while 54 for OpenPose coordinates and 6 for hat orientation and translation. We got a large number of features and too many features usually lead to a powerful model and learn to fit the training set very well then cause overfitting. There is a greater chance of redundancy in features and of features that are not at all related to prediction. The main approach we did for feature selection was RFECV (Recursive Feature Elimination with Cross-Validation). As its name suggested, REFCV will select features by recursively considering smaller and smaller sets of features. The general process of REFCV is the least important predictor(s) are removed, then the model is re-built, and importance scores are computed again. However, this method is expensive to run, and putting all features into REFCV is not reasonable and time-consuming. We first find all the feature pairs that are highly correlated (with correlation coefficient > 0.8) and keep only one of them. This action reduced the number of features from 60 to 37. Secondly, we run RFECV with only 37 features and we were using Random Forest in REFCV since we can get the importance of the attributes from tree models. Our REFCV will remove one feature at each iteration and the optimal number of features is shown in the right graph of Figure 2 and we can see that around 34 will be the optimal number of features. Lastly, we dropped the three least import features shown in the left graph of Figure 2. After we did the feature selection, our model gained a similar validation accuracy which is around 83%, and alleviated the issue of over-fitting as shown in Figure 4.

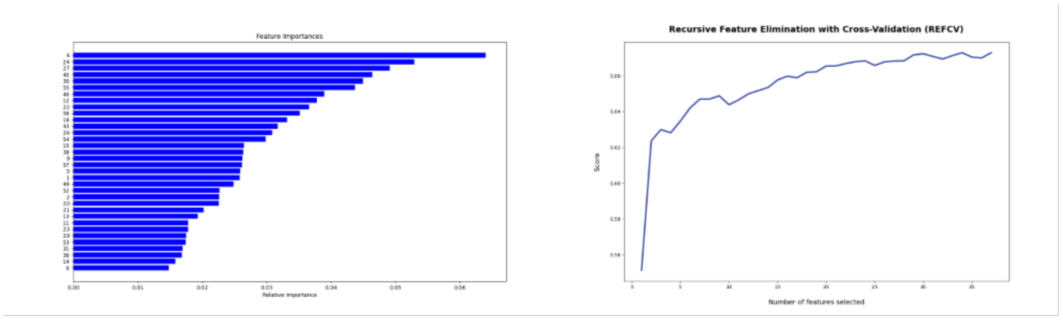

Figure 2: Feature selection

3 Models

We have implemented several different basic classification models and recurrent neural network models. Initially, we train basic models such as SVM, Vanilla neural network and frame by frame rather than using a sliding window. However, we have noticed the accuracy suffered from the lack of interconnections between actions. Therefore, RNN models were implemented with sequences of frames instead.

3.1 Basic models

Many researchers have shown that Machine Learning algorithms such as SVM(Support Vector Machine), KNN(K-Nearest Neighbors algorithm) have a good performance on data classification. Niu and Suen[4] used a hybrid CNN–SVM classifier for recognizing handwritten digits with an accuracy of 99.81%. Our project is also a classification problem, therefore, we will start to train some basic models such as SVM and compare the result with the models such as LSTM. We applied three different basic models, SVM, KNN, and Random Forest with the input with all the 60 features.

Table 1: Basic models accuracy

Basic Model	Training Accuracy(%)	Validation Accuracy (%)	5-fold cv (%)
SVM(linear kernel)	62.7	60.8	60.0
KNN	88.7	70.0	52.0
Random Forest Classifier	80.4	74.2	67.0

We used the grid search to tune the hyper-parameters for SVM and KNN. For SVM, we tuned the parameters C and γ ranging in (0.001, 0.01, 0.1, 1, 10, 100) and C=10.0,γ=0.01 produced the most optimal result while the KNN model has the highest accuracy with parameter n_neighbors = 2. To evaluate the models, we perform a 5-fold cross-validation. The result is shown in Table 1. The result shows that all the basic models have around 60% to less than 70% accuracy. However, one thing to notice is both KNN and Random Forest are over-fitting and their scores in 5-fold cross-validation drop significantly. The result still needs improvement and these models seem to highly rely on numeric data such as hat orientation rather than guessing the possible relationship between video sequences. These basic models could produce a good result for data that is not in time series such as handwritten digits recognition.

3.2 Vanilla neural network

The basic neural network we used for baseline results contains three regular deeply connected neural network layers. The first two layers both use ReLU(Rectified Linear Unit) as the activation function and the first layer takes each data frame as input. ReLU allows the network to converge quickly which lets us identify our results conveniently. Both deeply connected layers have 127 neurons. The last layer is the output layer which has three neurons for three different classes(labels) with softmax as its activation function to produce probability distribution. Since our task is a classification problem with single-label categorization, we used sparse categorical cross-entropy as our loss function.

To train and validate our vanilla neural network model, we separate the dataset into training and validation datasets with a proportion of 8:2. According to the plot above, the vanilla neural network model can only achieve around 62 percent accuracy for both training and validation datasets with training loss close to 0.9. Therefore, we can conclude that the basic neural network model with no LSTM is not good enough to detect customers' confusion according to our frame data.

3.3 LSTM

The process of determining if a customer is confused should be treated as a subject in action detection. It makes intuitive sense to consider a sequence of actions instead of just one frame. We believe the previous frame-by-frame training on plain dense layers failed due to the overlook of the time factor. A sequence of actions within some interval should be a data point to feed into models. Therefore, a sliding window approach(section 2.1) was implemented.

To train sequences of frame information, RNN is a natural choice for data sequence training. Moreover, LSTM is selected as our base model to prevent the potential vanishing gradient problem as the size of our sliding window may get huge. LSTM can learn long-term temporal relationships using its memory cells. The memory cells can learn to forget previous memories and consider the current input. Then determining how much of the useful memory to be transferred to the next hidden state [2]. We have stacked two LSTM layers for greater model complexity as shown in Figure 3. The first layer of LSTM outputs a sequence of hidden state vectors to the second layer. The second LSTM layer also outputs a single hidden state vector from the last hidden state to a dense layer. The dimension of the output hidden state vector was set to be 127. The activation function for both LSTM layers was Tanh, which was examined with better performance than Relu. As result, our LSTM model outperformed the previous plain neural nets by 20 percent in terms of validation accuracy. It was a strong evidence that our approach of adding time factor into our model was successful.

3.4 Multi-input LSTM

More approaches have been done in order to further improve our LSTM performance. As mentioned previously in the "Combine Features" section. We have two major sets of features where one being

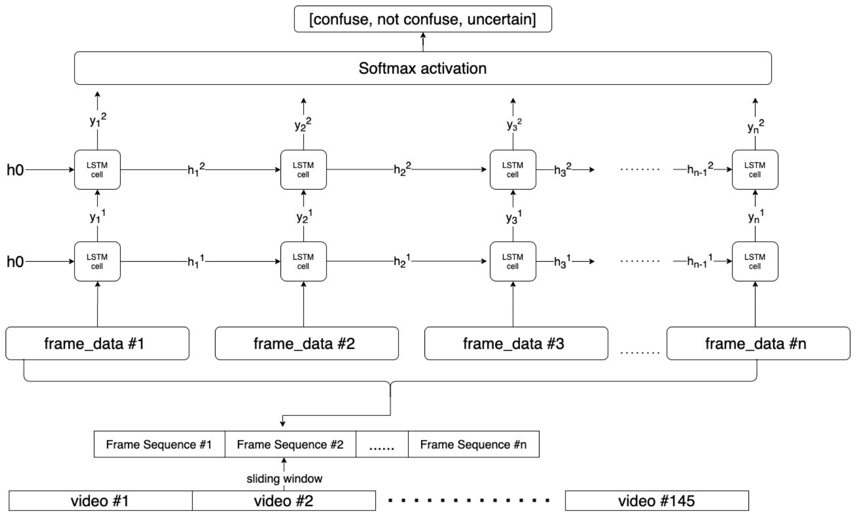

Figure 3: Stacked LSTM model.

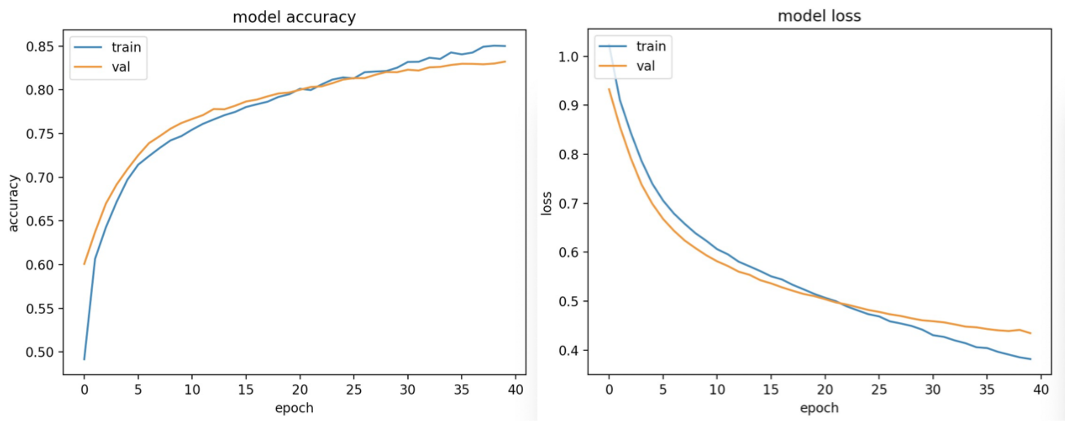

Figure 4: Result of stacked LSTM model.

OpenPose coordinates and the other one being hat orientation coordinates. There was a relatively large difference between these two sets of features in terms of size. Each frame has 54 OpenPose coordinate data points, while hat orientation only has 6 features per frame. When these two sets are combined to feed into the model, hat orientation features were overlooked due to its size being only 1/10 of OpenPose coordinates. Therefore, we have implemented a double LSTM model approach to compensate for the size disadvantage of hat orientation features [3]. The graphical representation of this model refers to Figure 4.

To that aim, we have built a combined model where 54 OpenPose coordinates features were fed into a two-layered LSTM's hidden cells, while the 6 hat orientation features were fed into another separated LSTM's hidden cells. A dense layer was added to both LSTM models to uniform the output vector size to 64. Finally, these two separated LSTM output two uniform-sized vectors to a shared activation layer. As a result, this double LSTM approach achieved slightly better validation accuracy(around 5%) than the previous single LSTM with a mixture of every feature. We believe such improvement was due to the compensation towards small-sized hat orientation features.

3.5 Bidirectional LSTM

When we were annotating data, we found out that watching a few frames ahead of the current frame will be easier to annotate the data. For example, a frame should be annotated as 'confused' if its previous and future few frames are marked as 'confused' too. Because a customer usually has a

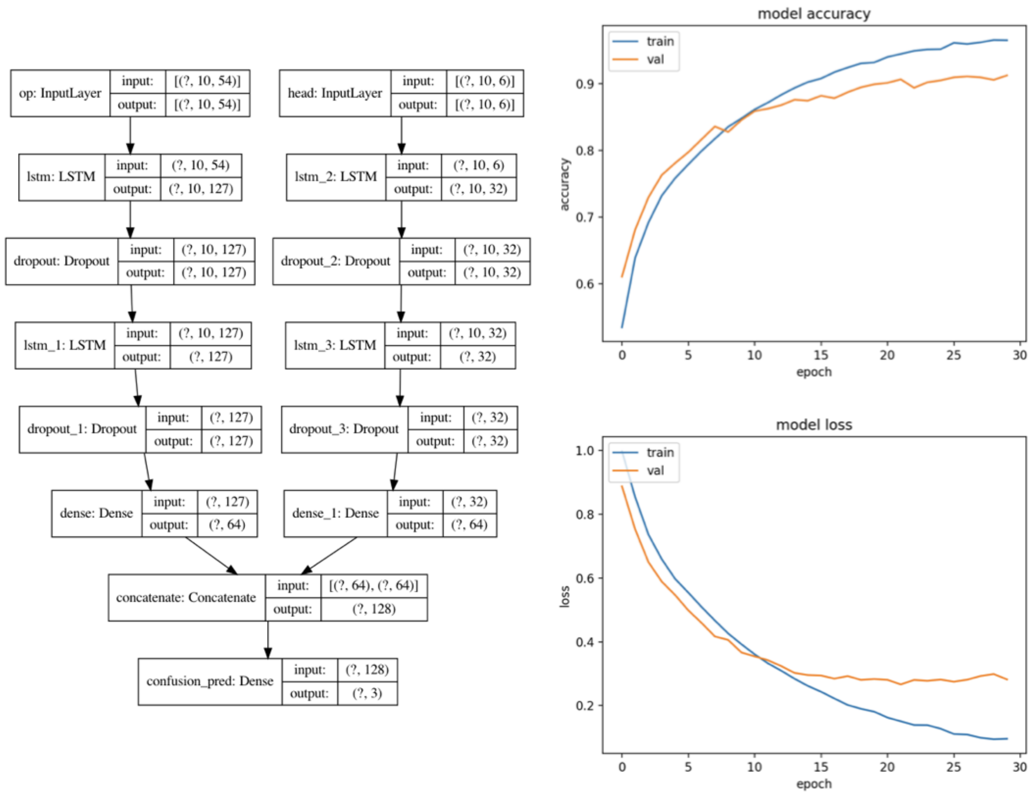

Figure 5: Multi-input LSTM model & results.

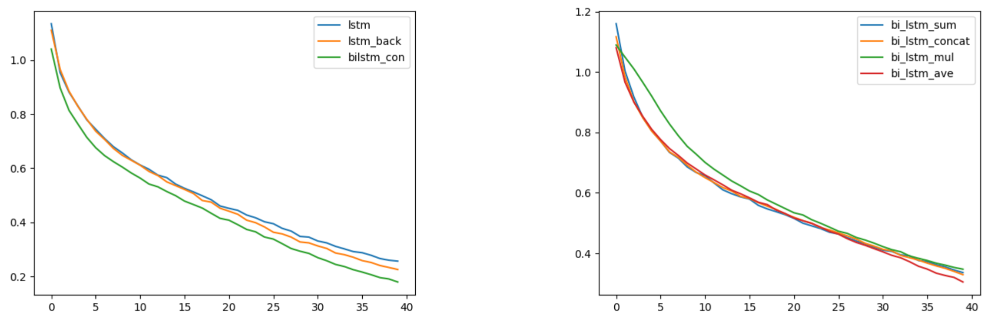

Figure 6: Loss values plot for 40 epochs

continuous sequence of frames of ‘confused’ and it is rare to see a customer confused for a while then suddenly not confused, and then confused again. However, a traditional LSTM predicts the current output only based on previous inputs and this model will not produce the optimal result in our case. Intuitively, it is hard to make a good annotation without watching a few frames ahead, and keeping both past and future information in our mind helps us to do the annotation quicker and correctly.

Context relationship is important in our case since a customer could have a series of confused, not confused, uncertain throughout a video. Unlike Geest and Tinne[5], they applied a two-stream LSTM for online action detection and each action video is independent of each other while our video sequences are not. To compare the performance between traditional LSTMs and bidirectional LSTM. We did an experiment by plotting the loss values of 40 epochs for three different models, forward

LSTM, backward LSTM, and bidirectional LSTM. The result is shown in the left graph of Figure 6. Both loss values of forwarding and backward LSTM are close to each other while bidirectional has lower loss values for all the epochs. Therefore, bidirectional LSTM may result in better performance. The workflow of bidirectional LSTM is shown in Figure 6. The input will be video sequences with 36776 in total. Then we have two LSTM layers, forward and backward LSTM. Forward will take the input as regular order while backward will consume the video sequences as a reversed copy. In a word, we are running inputs in two ways, one from past to future and one from future to past. After these two LSTM take the normalized inputs and produce the outcomes, it has 4 different ways of merging these two outcomes from different LSTM layers. Then we do a similar thing as did in comparing different LSTM models and the result is shown in the right graph of Figure 6. We can see that each mode has similar performance and I choose the average mode which will take the average of outcomes of forwarding and backward LSTM. Then we take this average hidden sequence into another LSTM decoder and finally a fully connected layer with a sigmoid activation function which produces output values between 0 and 1. The result is shown in Figure 8 and the accuracy is better than traditional LSTM.

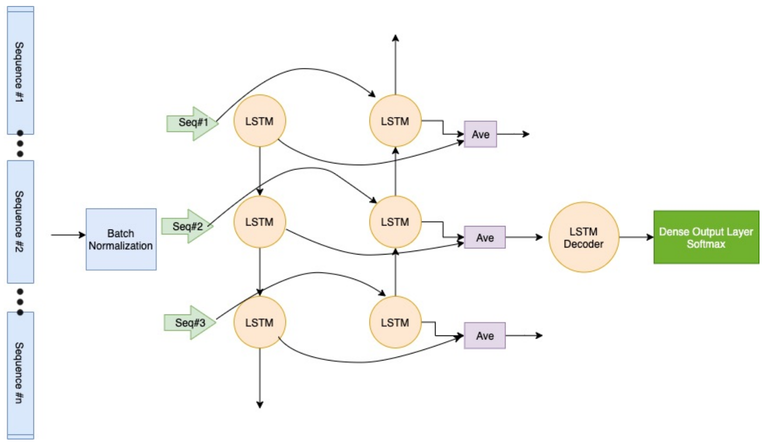

Figure 7: Bidirectional LSTM

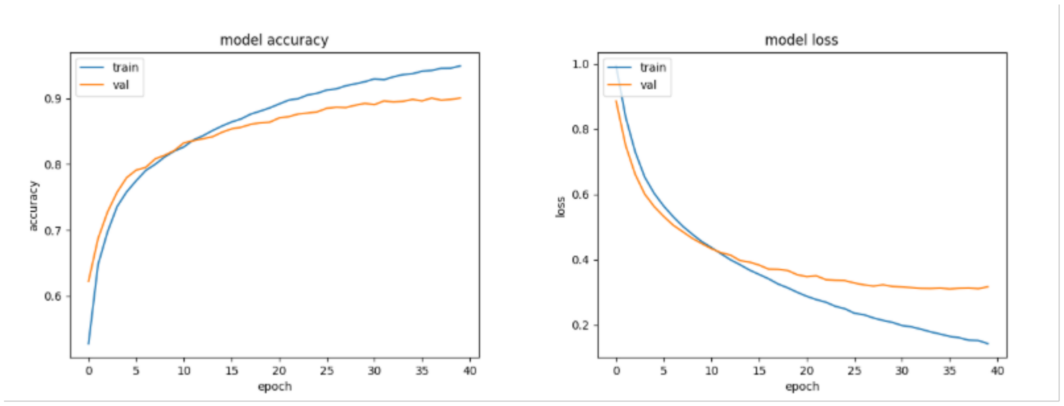

Figure 8: Result of Bidirectional LSTM

3.6 Results

Firstly, we have tried some basic models such as SVM and Vanilla neural network. These models have around 65% accuracy and basic models have severe over-fitting. Since our dataset is time-series data, we created the sliding windows to utilize this time factor. We normalized the video sequences and treated them as input for following LSTM models. The result is shown in Table 2. The 2-layered LSTM has better accuracy than the basic model and is not overfitting too much after we did the feature selection. Then we noticed that openpose has 54 features while head orientation has only 6 in 2-layered LSTM. We invertigated Multi-input LSTM which placed openpose coordinates and head orientation into different LSTM. By doing this, we have weighted openpose coordinates and head orientation equally. The accuracy increased by around 8% compared to 2-layered LSTM. Lastly, we noticed that LSTM will only predict results based on past information and we think keeping both past and future information would produce a better result in our case. The accuracy of bidirectional LSTM is around 90% which is similar to Multi-input LSTM.

Table 2: Models results

Models	Train-Accuracy(%)	Train-Loss(%)	Val-Accuracy(%)	Val-Loss(%)
Vanilla FCNN	61.5	0.872	62.3	0.851
2-layered LSTM	85.0	0.382	83.2	0.434
Multi-input LSTM	96.5	0.109	90.8	0.330
Bidirectional LSTM	94.9	0.142	90.0	0.316

References

[1] L. Ballan, M. Bertini, A. D. Bimbo & G. Serra. (2009). Action Categorization in Soccer Videos Using String Kernels. *Seventh International Workshop on Content-Based Multimedia Indexing*, Chania, 2009, pp. 13-18, doi: 10.1109/CBMI.2009.10.

[2] Gers, F. A., Schraudolph, N. N., & Schmidhuber, J. (2002). Learning precise timing with LSTM recurrent networks. *Journal of machine learning research*, 3(Aug), 115-143.

[3] Li, H., Shen, Y., & Zhu, Y. (2018, November). Stock price prediction using attention-based multi-input LSTM. In *Asian Conference on Machine Learning* (pp. 454-469).

[4] Niu, Xiao-Xiao, & Suen, Ching Y. (2012). A novel hybrid CNN–SVM classifier for recognizing handwritten digits. Pattern Recognition, 45(4), 1318–1325. https://doi.org/10.1016/j.patcog.2011.09.021

[5] Gammulle, Harshala, Denman, Simon, Sridharan, Sridha, & Fookes, Clinton. (2017). Two Stream LSTM: A Deep Fusion Framework for Human Action Recognition. 2017 IEEE Winter Conference on Applications of Computer Vision (WACV), 177–186. https://doi.org/10.1109/WACV.2017.27

[6] Leo Audibert and Barer Simon.“confusion_detection.”GitHub,github.com/leomorpho/confusion_detection.

Contribution

Zhicheng Xu: Feature selection, Basic models, Bidirectional LSTM
Xuecong Tan: Data Annotation, Sliding window, Vanilla NN, LSTM
Rong Li: Vanilla NN, LSTM, Multi-input LSTM