RNN with and without CNN

This project deals with the approach of recurrent neural networks (RNN) by implementing six models (three RNN models and three RNN + CNN models) designed to classify 12 human actions (at night). names_class : ['Drink', 'Picking', 'Push', 'Run', 'Throwing objects', 'boxing', 'lifting weights', 'receiving the phone', 'stand', 'walking on stairs', 'walking with flashlight', 'waving']

The dataset is 'https://www.kaggle.com/api/v1/datasets/download/lakavathakshay/noctact-har', used in this project, contains 6613 .mp4 videos of up to 15 seconds.

The project contains:

- Download dataset (download_data.ipynb)
- Data reprocessing(video_to_dataset.ipynb)
- Model created with SimpleRNN and description of SimpleRNN networks
- Model created with LSMT and description of LSMT networks
- Model created with GRU and description of GRU networks
- Model created with SimpleRNN + CNN
- Model created with LSMT + CNN
- Model created with GRU + CNN
- Comparison of the results obtained

Data preprocessing

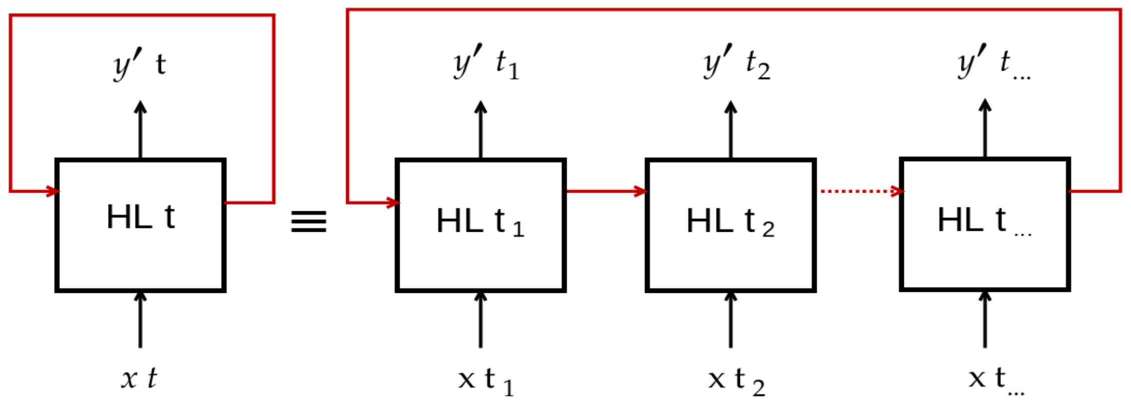
Data preparation for processing is performed in the video_to_dataset.ipynb file using the video_convert class. This class has the following features:

- creating the database, i.e., retrieving information from the directory structure containing the films (indexing characteristics and indexing labels)
- changing image size in two steps
  - define the desired image ratio between width and height by adding values of 0, as the case may be, left and right, or top and bottom
  - changing the resolution of images to the desired form
- extracting the desired number of frames so that the captured frames are distributed evenly over the length of the film
- generating the training dataset
- generation of the data set for validation
- generating a small test data set (normally, the test set should be much larger, but in this case, it is used to test functionality)
- method of saving data sets on physical media

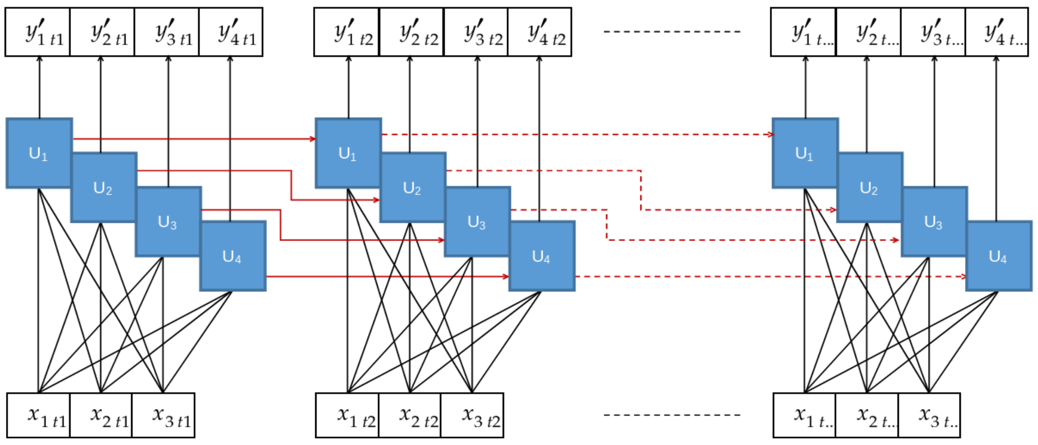
Recurrent neural networks (RNN)

Recurrent neural networks (RNNs) are neural networks used especially for time series or models whose predictions are based on data sets that have the characteristic of a chain of successive phenomena, such as sequences of film, sound, text, etc.

Recurrent neural networks (RNNs) have an architecture similar to artificial neural networks (ANNs), but unlike ANNs, RNNs use the same weights in each run sequence, while also passing on a feature emitted by the previous sequence, i.e., HL t1 combined with the characteristics from sequence HL t2



Taking the idea illustrated on the right side of the diagram above, the distribution of the data set ordered over a length of time (x_t) in the cells of the RNN network is illustrated in the diagram below. It should be noted that in the case of multiple RNN layers, y'_t (the prediction from the current layer) becomes the input feature of the upper layer, i.e., x_t.

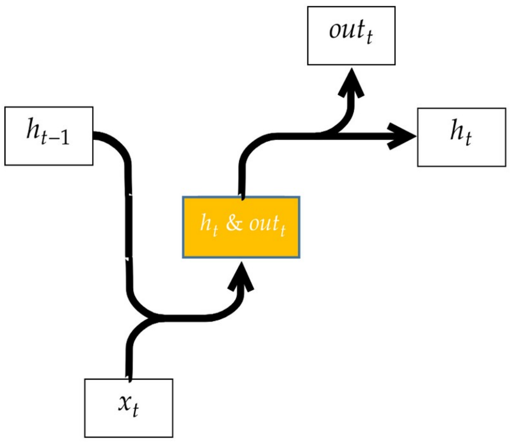


$y'_{1\,t1}$ $y'_{2\,t1}$ $y'_{3\,t1}$ $y'_{4\,t1}$
$y'_{1\,t2}$ $y'_{2\,t2}$ $y'_{3\,t2}$ $y'_{4\,t2}$
$y'_{1\,t...}$ $y'_{2\,t...}$ $y'_{3\,t...}$ $y'_{4\,t...}$
U₁ U₂ U₃ U₄
$x_{1\,t1}$ $x_{2\,t1}$ $x_{3\,t1}$
$x_{1\,t2}$ $x_{2\,t2}$ $x_{3\,t2}$
$x_{1\,t..}$ $x_{2\,t..}$ $x_{3\,t..}$

Simple RNN

SimpleRNN is the simplest form of the recurrent neural networks. Its functionality is achieved by combining the input features of sequence t1 with the features emitted by the previous sequence t $[h_{t-1},x_t]$ and activated with a hyperbolic tangent. The equation of the activation function is $out_t = tanh\left([h_{t-1},x_t]\right)$

Equations for implementing the SimpleRNN block
$h_t \,\&\, out_t = tanh\left(w \times [h_{t-1},x_t]\,.copy\right)$

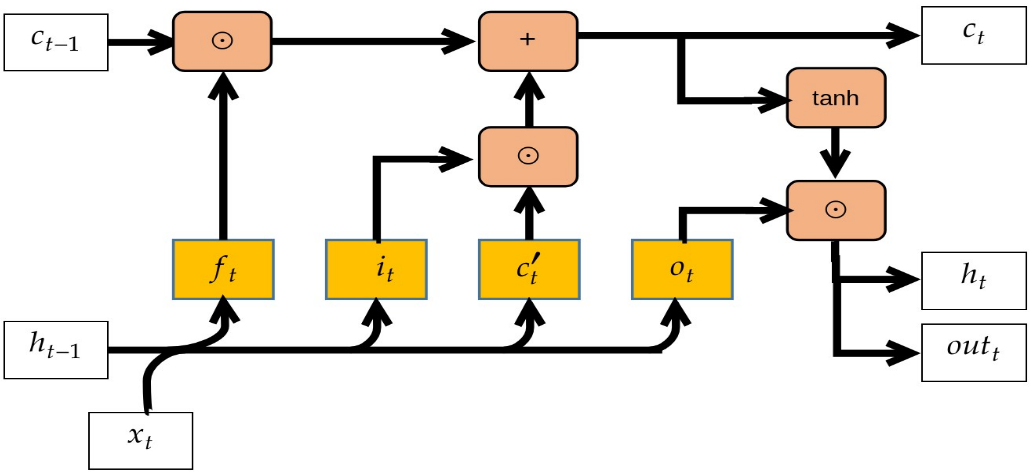


```
# Define sample input shape (e.g., 16 sequences, 5 time steps, 3 features)
batch_size = 16
time_steps = 5
features = 3
input_data = tf.random.normal((batch_size, time_steps, features), dtype = tf.float32)
print(input_data.shape)

# number of units per layer
units_size_per_layer  = 4

SimpleRNN_whole_sequence_output, SimpleRNN_final_memory_state =
SimpleRNN(units_size_per_layer, return_sequences=True, return_state=True)(input_data)

print(SimpleRNN_whole_sequence_output.shape)
print(SimpleRNN_final_memory_state.shape)
```

```
(16, 5, 3)
(16, 5, 4)
(16, 4)
```

Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) is an improved version of simple Recurrent Neural Networks. The main difference between SimpleRNN and LSTM is that, in addition to the hidden state taken from the previous sequence and concatenated with the input features corresponding to the time sequence, LSTM networks have a memory cell with extended information as a period (Cell State).



The architecture of LSTM networks consists of three gates:

- Forget gate: determines what information is deleted from the cell memory
- Input gate: controls what information is added to the cell memory
- Output gate: controls what information comes out of the cell memory

Equations for implementing the LSTM block

$f_t = sigmoid\left(w_f \times [h_{t-1},x_t].copy + b_f\right)$ Forget gate

$i_t = sigmoid\left(w_i \times [h_{t-1},x_t].copy + b_i\right)$ Input gate

$c_t' = tanh\left(w_c \times [h_{t-1},x_t].copy + b_c\right)$

$o_t = sigmoid\left(w_o \times [h_{t-1},x_t].copy + b_o\right)$

$\boxed{c_t = f_t \odot c_{t-1} + i_t \odot c_t'}$ - Cell State

$\boxed{h_t \,\&\, out_t = tanh\left(c_t\right) \odot o_t}$ Output gate

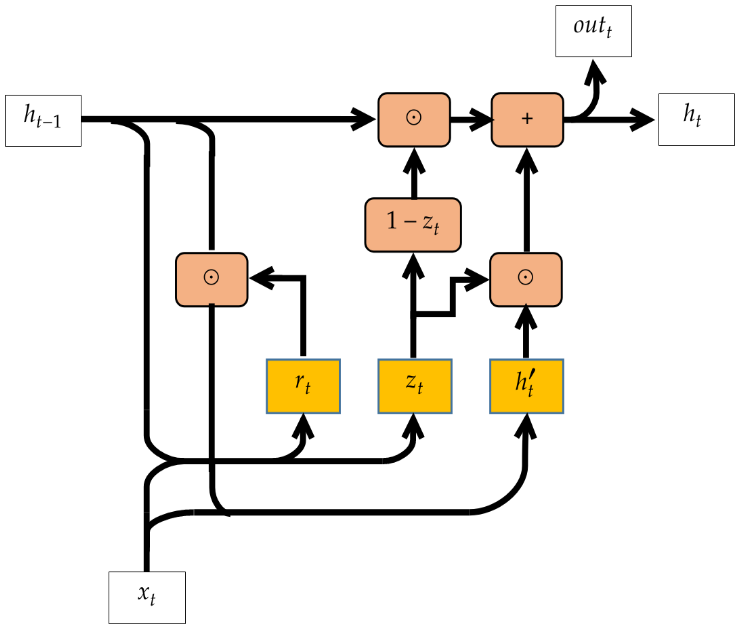
```
# Define sample input shape (e.g., 16 sequences, 5 time steps, 3 features)
batch_size = 16
time_steps = 5
features = 3
input_data = tf.random.normal((batch_size, time_steps, features), dtype = tf.float32)
print(input_data.shape)

# number of units per layer
units_size_per_layer  = 4

LSTM_whole_sequence_output, LSTM_final_memory_state, LSTM_final_carry_state =
LSTM(units_size_per_layer, return_sequences=True, return_state=True)(input_data)

print(LSTM_whole_sequence_output.shape)
print(LSTM_final_memory_state.shape)
print(LSTM_final_carry_state.shape)
```

```
(16, 5, 3)
(16, 5, 4)
(16, 4)
(16, 4)
```

Gated Recurrent Unit (GRU)

Gate recurrent units (GRUs) are a type of RNN whose principle is to use gate mechanisms to selectively update the hidden state at each time step, allowing them to retain important information and eliminate irrelevant details. GRU is a simplified version of LSTM architectures and consists of two main gates: the update gate and the reset gate.

- Update gate (zt): this gate decides how much information from the previous hidden state h t-1 should be retained for the next
- Reset Gate (r_t): This gate determines how much of the hidden state from the past h $_{t-1}$ should be forgotten.



Equations for implementing the GRU block

$z_t = sigmoid\left(w_z \times [h_{t-1}, x_t].copy\right)$ Update Gate

$r_t = sigmoid\left(w_t \times [h_{t-1}, x_t].copy\right)$ Reset Gate

$h'_t = tanh\left(w_h \times [r_t \odot h_{t-1}.copy, x_t.copy]\right)$

$h_t \ \& \ out_t = (1-z_t) \odot h_{t-1}.copy + z_r \odot h'_t$ - Ouput

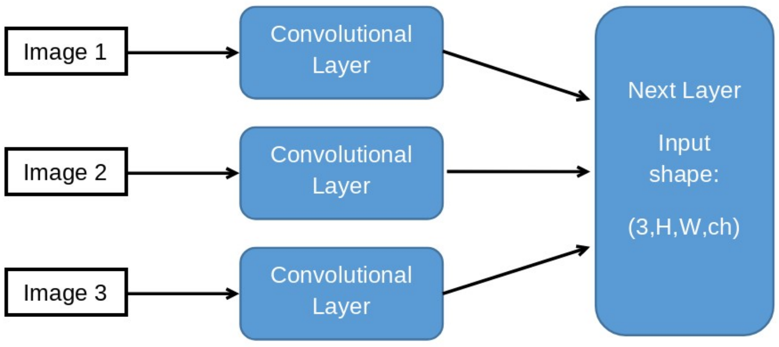
# Define sample input shape (e.g., 16 sequences, 5 time steps, 3 features)
batch_size = 16
time_steps = 5
features = 3
input_data = tf.random.normal((batch_size, time_steps, features), dtype = tf.float32)
print(input_data.shape)

```
# number of units per layer
units_size_per_layer  = 4

GRU_whole_sequence_output, GRU_final_memory_state = GRU(units_size_per_layer,
return_sequences=True, return_state=True, unroll=True)(input_data)

print(GRU_whole_sequence_output.shape)
print(GRU_final_memory_state.shape)
```

```
(16, 5, 3)
(16, 5, 4)
(16, 4)
```

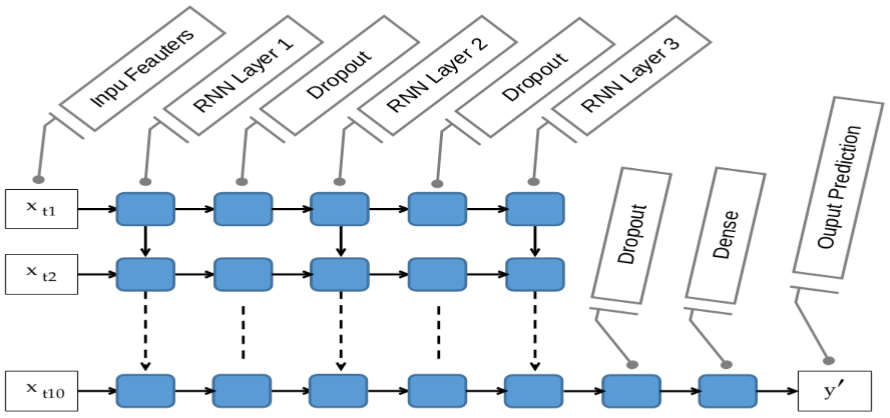
Time distributed layer

TimeDistributed is a method whereby a certain layer, method function, etc., can be executed successively with different input characteristics, returning a number of results equal to the number of inputs. This method is used, for example, in processing data series, video frames, audio sequences, etc., where each time step is treated independently with the same method. For example, the figure below shows the successive approach of a CNN layer using TimeDistributed.



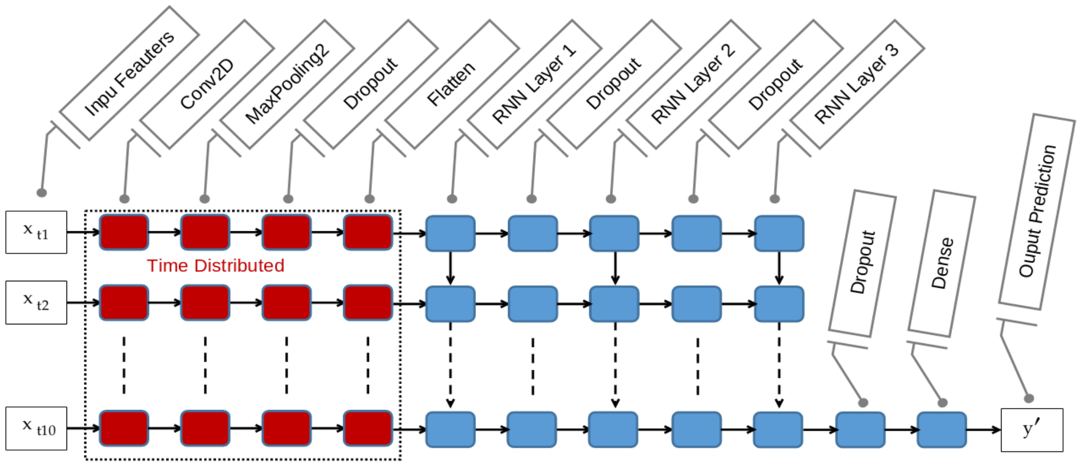
Description of models

The six models were designed to highlight the functionality of RNNs in the context of their use in models involving the analysis of actions in a video recording. To this end, three models were created that use only the three types of RNN, namely SimpleRNN, LSTM, and GRU. As can be seen in the diagram below, these models are composed of three RNN layers and a final Dense layer.



```
SimpleRNN_model - Trainable params: 29,523,564 (112.62 MB)
LSTM_model - Trainable params: 118,093,068 (450.49 MB)
GRU_model - Trainable params: 88,570,572 (337.87 MB)
```

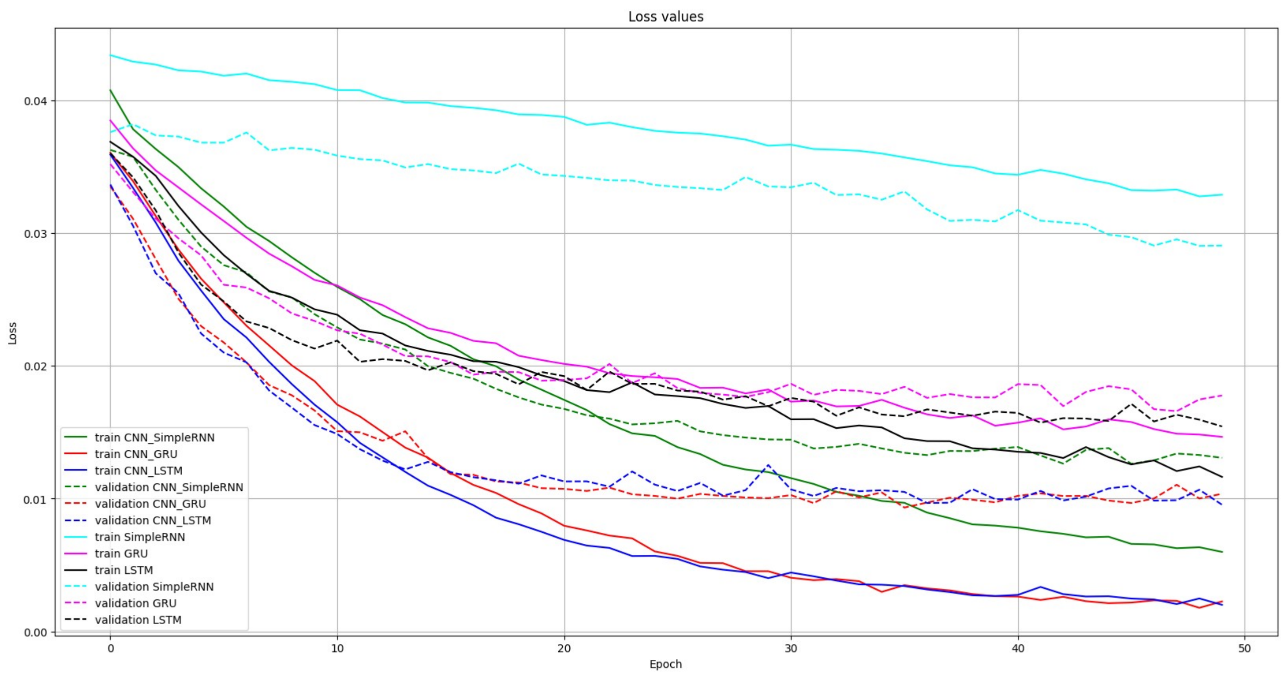
Due to the large input characteristics, i.e., 10 time steps containing 240x320x3 frames, the parameter matrices are very large, as can be seen above. One solution for improving the efficiency of models containing RNN networks for predicting datasets composed of video recordings is to use CNN networks in the composition of the models. Thus, three other models were created by extending the models mentioned above using the TimeDistributed method, in which a CNN layer was integrated. The diagram below shows the approach of the three new models.



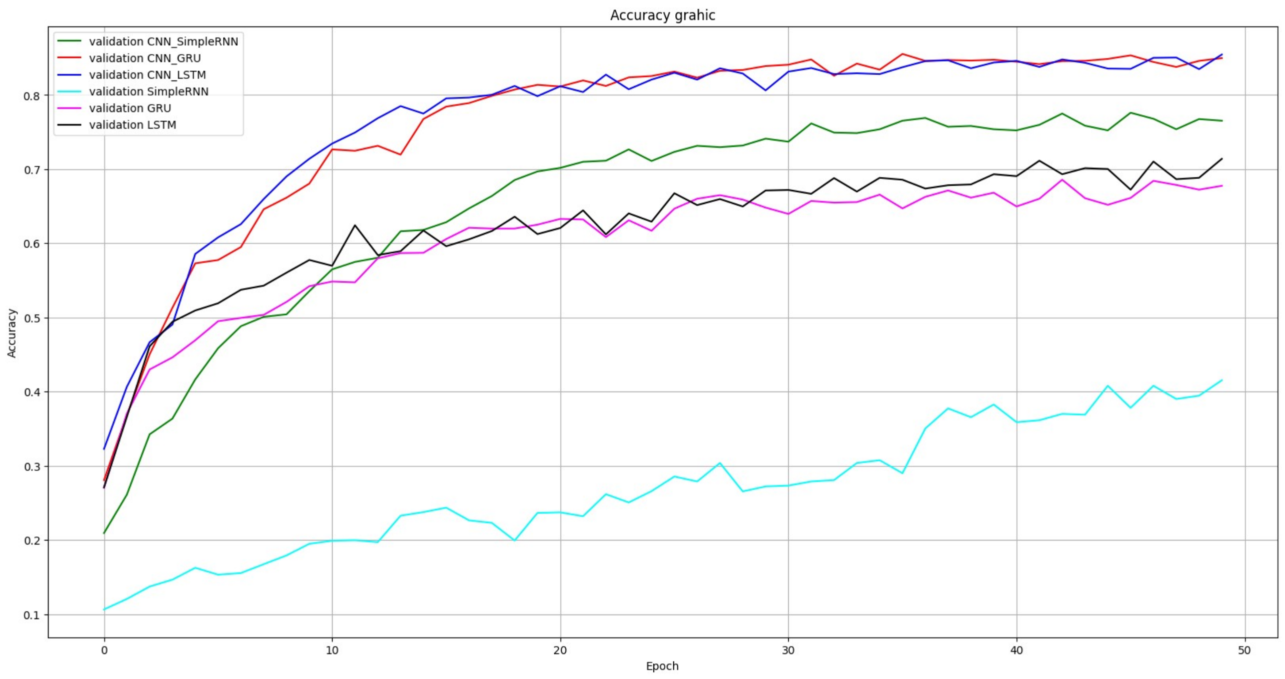
```
CNN_SimpleRNN_model - Trainable params: 3,089,004 (11.78 MB)
CNN_LSTM_model - Trainable params: 11,835,660 (45.15 MB)
CNN_GRU_model - Trainable params: 8,920,780 (34.03 MB)
```

<u>It should be noted that the descriptions of the Input, MaxPooling2...., and other methods have been omitted !</u>

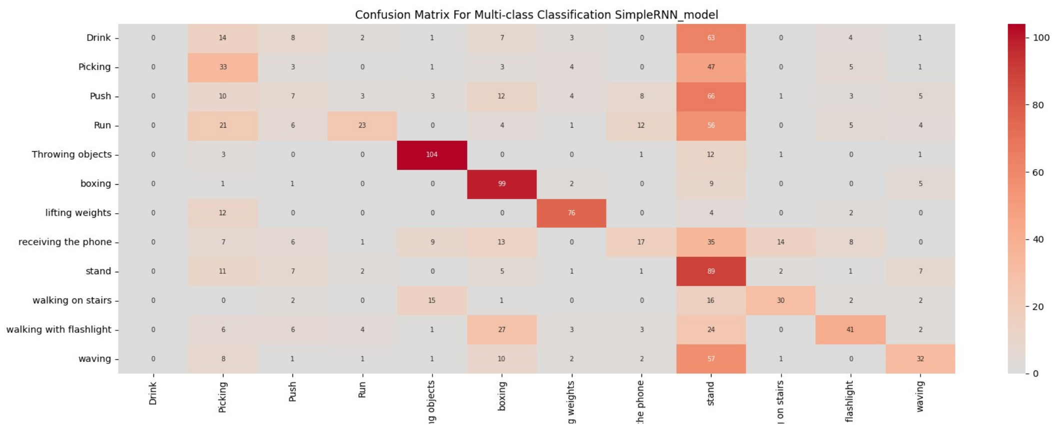
Comparative diagram of the loss function for the 6 models created

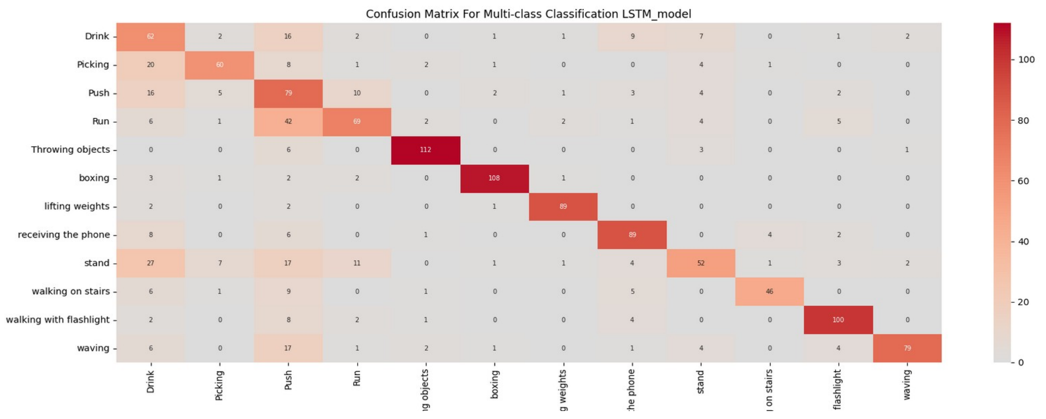


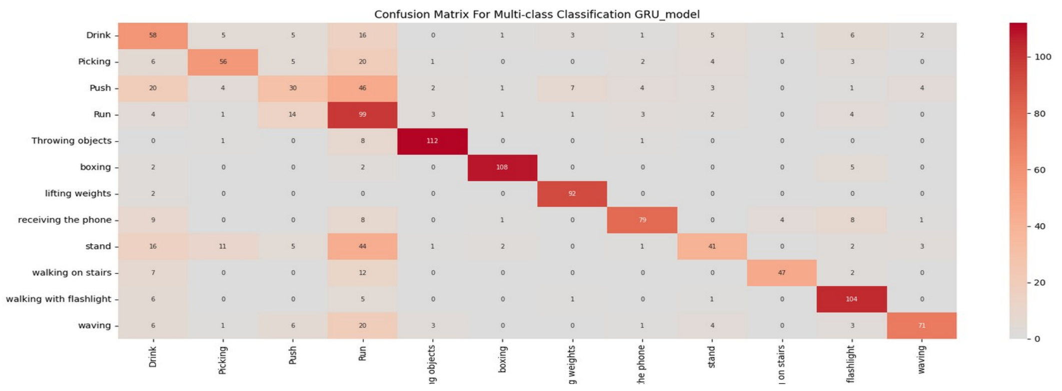
Comparative diagram of efficiency (prediction accuracy) for the six models created



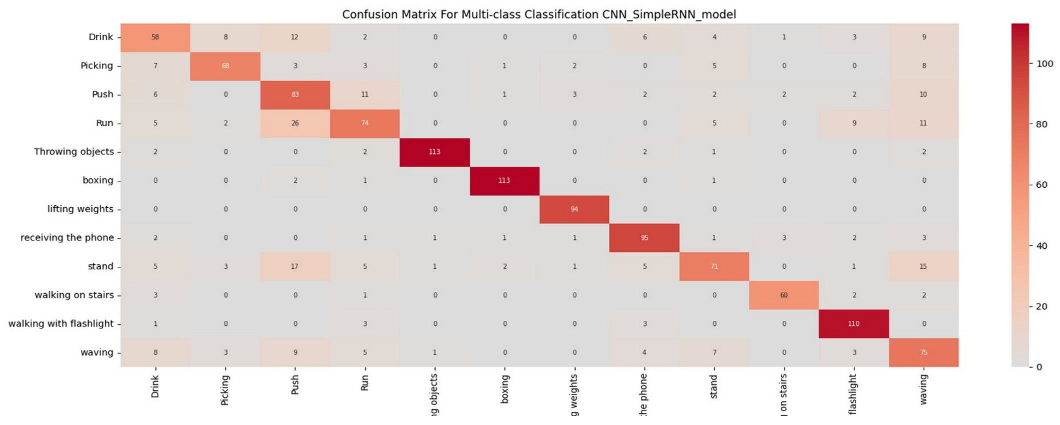
Confusion matrices for models built with RNN



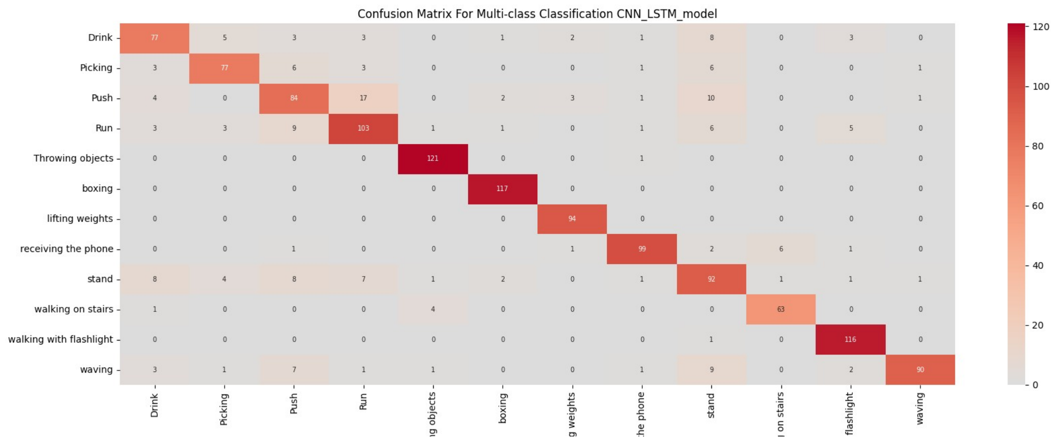




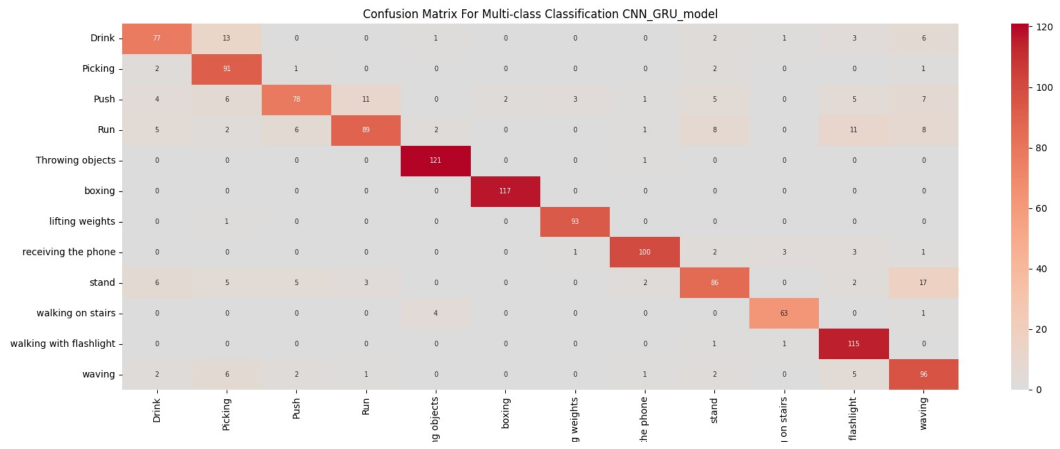
Confusion matrices for models built with RNN and CNN



	Drink	Picking	Push	Run	ıg objects	boxing	g weights	he phone	stand	ı on stairs	flashlight	waving
Drink	58	8	12	2	0	0	0	6	4	1	3	9
Picking	7	68	3	3	0	1	2	0	5	0	0	8
Push	6	0	83	11	0	1	3	2	2	2	2	10
Run	5	2	26	74	0	0	0	0	5	0	9	11
Throwing objects	2	0	0	2	113	0	0	2	1	0	0	2
boxing	0	0	2	1	0	113	0	0	1	0	0	0
lifting weights	0	0	0	0	0	0	94	0	0	0	0	0
receiving the phone	2	0	0	1	1	1	1	95	1	3	2	3
stand	5	3	17	5	1	2	1	5	71	0	1	15
walking on stairs	3	0	0	1	0	0	0	0	0	60	2	2
walking with flashlight	1	0	0	3	0	0	0	3	0	0	110	0
waving	8	3	9	5	1	0	0	4	7	0	3	75



	Drink	Picking	Push	Run	ıg objects	boxing	g weights	he phone	stand	ı on stairs	flashlight	waving
Drink	77	5	3	3	0	1	2	1	8	0	3	0
Picking	3	77	6	3	0	0	0	1	6	0	0	1
Push	4	0	84	17	0	2	3	1	10	0	0	1
Run	3	3	9	103	1	1	0	1	6	0	5	0
Throwing objects	0	0	0	0	121	0	0	1	0	0	0	0
boxing	0	0	0	0	0	117	0	0	0	0	0	0
lifting weights	0	0	0	0	0	0	94	0	0	0	0	0
receiving the phone	0	0	1	0	0	0	1	99	2	6	1	0
stand	8	4	8	7	1	2	0	1	92	1	1	1
walking on stairs	1	0	0	0	4	0	0	0	0	63	0	0
walking with flashlight	0	0	0	0	0	0	0	0	1	0	116	0
waving	3	1	7	1	1	0	0	1	9	0	2	90



	Drink	Picking	Push	Run	ıg objects	boxing	g weights	he phone	stand	ı on stairs	flashlight	waving
Drink	77	13	0	0	1	0	0	0	2	1	3	6
Picking	2	91	1	0	0	0	0	0	2	0	0	1
Push	4	6	78	11	0	2	3	1	5	0	5	7
Run	5	2	6	89	2	0	0	1	8	0	11	8
Throwing objects	0	0	0	0	121	0	0	1	0	0	0	0
boxing	0	0	0	0	0	117	0	0	0	0	0	0
lifting weights	0	1	0	0	0	0	93	0	0	0	0	0
receiving the phone	0	0	0	0	0	0	1	100	2	3	3	1
stand	6	5	5	3	0	0	0	2	86	0	2	17
walking on stairs	0	0	0	0	4	0	0	0	0	63	0	1
walking with flashlight	0	0	0	0	0	0	0	0	1	1	115	0
waving	2	6	2	1	0	0	0	1	2	0	5	96

Bibliographer

https://www.datacamp.com/tutorial/tutorial-for-recurrent-neural-network

https://www.exxactcorp.com/blog/Deep-Learning/recurrent-neural-networks-rnn-deep-learning-for-sequential-data

https://medium.com/analytics-vidhya/what-is-rnn-a157d903a88

https://medium.com/analytics-vidhya/lstms-explained-a-complete-technically-accurate-conceptual-guide-with-keras-2a650327e8f2

https://www.geeksforgeeks.org/deep-learning/deep-learning-introduction-to-long-short-term-memory/

https://www.geeksforgeeks.org/machine-learning/gated-recurrent-unit-networks/

https://d2l.ai/chapter_recurrent-modern/gru.html

https://www.detailedpedia.com/wiki-Gated_recurrent_unit

https://medium.com/smileinnovation/how-to-work-with-time-distributed-data-in-a-neural-network-b8b39aa4ce00

https://levelup.gitconnected.com/hands-on-practice-with-time-distributed-layers-using-tensorflow-c776a5d78e7e

https://colah.github.io/posts/2015-08-Understanding-LSTMs/