IJFANS INTERNATIONAL JOURNAL OF FOOD AND NUTRITIONAL SCIENCES

ISSN PRINT 2319 1775 Online 2320 7876

Research paper **© 2012 IJFANS. All Rights Reserved**, **Volume 11, Sp.Iss 6 , 2022**

On Device efficient prediction model for detection of jank using DNN models

S. Sagar Imambi[1]

[1] Professor, Dept. of CSE, Koneru Lakshmaiah Education Foundation (KLEF), Vaddeswaram, Green fields, Guntur, Andhra Pradesh, India -522302

M. Krishna Vamsi[2], Shaik Riyaz Basha[3]

[2,3] Dept. of CSE, Koneru Lakshmaiah Education Foundation (KLEF), Vaddeswaram, Green fields, Guntur, Andhra Pradesh, India -522302

DOI : 10.48047/IJFANS/11/S6/035

Abstract

Android produces UI by producing and presenting a frame from a mobile app on the screen. If the app's UI rendering is slow, the system is forced to skip frames. When this occurs, the user notices a repeated flicker on their screen, which is known as Jank. Jank issues can manifest in various forms, causing disruptions such as unstable frame rates, heightened latency, AppDeadlineMissing, and BufferStuffing. BufferStuffing occurs when the app runs beyond its expected duration, leading to jank. To quantify this issue, we determine the total time taken by the app frame, commencing with the choreographer wake-up as the starting point and concluding with max(GPU, post time) as the endpoint. Post time represents when the frame was dispatched to SurfaceFlinger. Notably, due to the parallel operation of the GPU, the GPU can complete its task after the post time. This situation is more of a state than a true jank occurrence and typically arises when the app continually dispatches new frames to SurfaceFlinger before the previous frame has been presented. This continuous influx of frames leads to the stuffing of the internal Buffer Queue with frames yet to be presented, hence the term "Buffer Stuffing." These additional buffers in the queue are presented one after the other, resulting in increased latency. This can eventually reach a point where there are no more buffers available for the app to utilize, leading to a blocking wait during dequeuing. Importantly, even if the actual work performed by the app remains within the deadline, the stuffed nature of frames leads to their presentation at least one vsync late, introducing elevated input latency. While the visual appearance of frames may remain relatively smooth in this state, the late presentation is associated with increased input latency. In this work, the LSTM Model(Long Short Term Memory) was used for the detection of Jank. LSTMs provide us with a large range of parameters such as learning rates, and input and output biases. Hence, no need for fine adjustments. Experimental result shows that the LSTM Model was able to predict the frame drop with an efficiency of 98% thereby enhancing the overall user experience.

Keywords: Andriod, CNN, Frame drop, Jank, LSTM, RNN,

1. Introduction

Many factors influence the cause of jank during UI rendering which can be categorized as internal factors like large inflates, animations, layouts, etc., and external factors like CPU/GPU, Battery, Touch inputs(by user), etc. A sophisticated Learning-based model is required to predict the jank occurrence in real time for smooth performance, To avoid jank and sluggish responsiveness when an Application is drawing to the screen by predicting the next frame drop and taking corresponding actions. This work aims to develop a learning model to predict the next frame drop that can occur on a smartphone while the user is using the application. This model works as a Generic model for all the scenarios (scroll, app switching).

1.2 Type of janks

AppDeadlineMissed

A jank was caused by the application running longer than planned. To determine the length of the application, wakeup is used as the start time,
and maximum (GPU, streaming time) is used as the end time.

The frame's post time is when it was dispatched to SurfaceFlinger. Given that the GPU often works concurrently, there's a chance that it experienced unexpected delays in completing its tasks.

BufferStuffing:

This is more of a situation than a performance issue. It happens when an application continually sends new frames to SurfaceFlinger before the previous frame has a chance to be shown. The term "buffer stuffing" describes the situation where the core buffer queue becomes overwhelmed with unprocessed buffers that rarely get displayed.

SurfaceFlingerCpuDeadlineMissed

SurfaceFlinger is anticipated to be completed within the time frame specified. SurfaceFlingerCpuDeadlineMissed is the jank if the main thread operated for a prolonged period. The amount of CPU time devoted to SurfaceFlinger's main thread. If device composing was employed, the full composition period is included. This includes the duration required for generating the draw calls and transferring them by hand. the frame off to the GPU if GPU composition was employed.

SurfaceFlingerGpuDeadlineMissed

SurfaceFlinger's main thread's CPU and GPU composition times combined took longer than anticipated. Here, the CPU time would still have been inside the allotted period, however, due to the GPU task not being finished promptly, the frame was postponed to the subsequent vsync cycle.

DisplayHAL

When SurfaceFlinger completes its task and promptly sends the frame to the HAL, the situation is known as a "DisplayHAL jank," However, the frame is not shown during the vsync event. It was displayed on the following vsync. It's possible that SurfaceFlinger did not provide enough time for the HAL's work, or it's possible that the work of the HAL was delayed.

This article also provides a thorough overview of the various techniques and strategies used in developing on-device efficient prediction models for jank detection, highlighting CNN with LSTM as a potential solution. It examines the benefits of CNNs in capturing intricate patterns in user interactions and system behaviors, with an emphasis on their ability to provide exact and timely jank predictions.

CNNs can easily incorporate query features and item features (due to the flexibility of the input layer of the network), which can help capture the specific interests of a user and improve the relevance of recommendations. The LSTM model was forecasted with the time-series log file data for a long lead period for the various applications of Android mobile phones.

The model architecture determines the complexity and expressivity of the model. By adding hidden layers and non-linear activation functions (for example, ReLU), the model can capture more complex relationships in the data.[4,5. Expanding the parameter count generally results in a more challenging and costlier model training process.

2. Survey of literature

2.1 Mobile Jank Prediction:

The present disclosure relates to methods and apparatus for frame processing. The apparatus can determine a current frame offset duration when a current frame rendering completion time is after the first VSYNC time. In some aspects, the current frame offset duration can be equal to a difference between the first VSYNC time and the current frame rendering completion time. The apparatus can also determine whether the sum of a previous frame GPU execution duration and the current frame offset duration is less than or equal to a first VSYNC period. In some aspects, the first VSYNC period can begin at the first VSYNC time and end at a second VSYNC time.[9] Additionally, the apparatus can execute a current frame based on the determination of whether the sum of the previous frame GPU execution duration and the current frame offset duration may be equal or low to the first VSYNC period.

2.2 LSTM Sequence Generation:

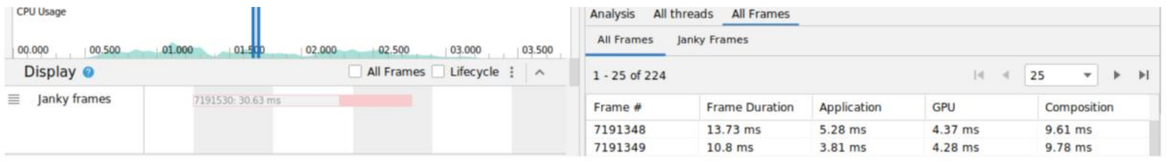
. Recurrent Neural Networks (RNNs) have proven to be effective in addressing tasks involving sequential predictions. This work aims to create a generative model for text. Even though, RNN has its limitations such as vanishing and exploding gradient descent problems, and inefficiency in keeping track of long-term dependencies. To overcome these drawbacks, Long Short Term Memory (LSTM) has been a path-breaking solution to deal with sequential data and text data in particular. This paper delineates the design and working of text generation using word-level LSTM-RNN.[1]

The LSTM neural network is an Encoder-Decoder built on a bidirectional multilayer architecture where the input sequence to the encoder is a list of user dialogue acts and the decoder output sequence is a list of system dialogue acts. All dialogue acts are defined at the intent level and are extracted from the Town Info corpus for tourist information provided by the FP7 Classic Project funded by the European Union. In their study, the LSTM configuration they proposed was pitted against a fully connected Hidden Markov Model (HMM) architecture. In this HMM model, user dialogue acts serve as states, and system dialogue acts serve as observations [11]. Following a series of diverse experiments, the findings collected from the analysis of the Town Info corpus unequivocally demonstrated that the LSTM-based system surpassed the HMM-based system in terms of performance.

3. Methodology

Data Collection and Data PreProcessing:

The dataset containing the time stamps of several frames was taken from our mentors and we used the pandas library to pre-process the data, There are a couple of problems with the raw data. Firstly, the Time and Amount columns exhibit significant variability, making them unsuitable for direct use. As a solution, omit the Time column, as its meaning is unclear, and apply a logarithmic transformation to the Amount column to narrow its range.



Frame #	Frame Duration	Application	GPU	Composition
7191348	13.73 ms	5.28 ms	4.37 ms	9.61 ms
7191349	10.8 ms	3.81 ms	4.28 ms	9.78 ms

Fig1. junk frames in Android mobile

Building Model:

Built a stacked LSTM model for the detection of jank, Stacking LSTM to allow for greater model complexity. In the case of a simple feedforward net, we stack layers to create a hierarchical

feature representation of the input data to then use for some machine learning task. The same applies to stacked LSTMs. At every time step an LSTM, besides the recurrent input. If the input is already the result of an LSTM layer (or a feedforward layer) then the current LSTM can create a more complex feature representation of the current input.

Train and Test:

Split the dataset into train, validation, and test sets. The validation set is used during the model fitting to evaluate the loss and any metrics; however, the model does not fit with this data. The test set is completely unused during the training phase and is only used at the end to evaluate how well the model generalizes to new data. This is especially important with imbalanced datasets where overfitting is a significant concern due to the lack of training data.

Predict:

After testing the model with 20% of the data, using the same testing data to predict the frame drop. Compared the results between the LSTM model and the DNN program, from which 94% of the prediction was accurate.

To predict the frame drop we used two different methods DNN and LSTM.Used the Pandas Python library to download CSVs into a Pandas Data Frame. Pandas has many helpful utilities for loading and working with structured data.

The use of the LSTM (Long Short Term Memory) Model helps in detecting jank by predicting frame drops in real time. Jank refers to the repeated flickering on the screen caused by slow UI rendering. When the system is forced to skip frames due to slow rendering, it leads to problems like unstable frame rate and increased latency.

The LSTM model, a variant of recurrent neural networks (RNNs), is tailored for processing sequential data and capturing extended temporal relationships. In the context of jank detection, the LSTM Model is trained on a dataset containing time stamps of frames. It learns the patterns and characteristics of frames that are likely to be dropped or cause jank.

The LSTM model uses system features like CPU usage, memory usage, and frame rate to forecast frame drop occurrences. It considers the historical patterns in the input data sequence and adjusts its internal state by incorporating both the current input and prior states.[2,6]

These three parts of an LSTM cell are known as gates. The first part is called Forget gate, the second part is known as the Input gate and the last one is the Output gate.

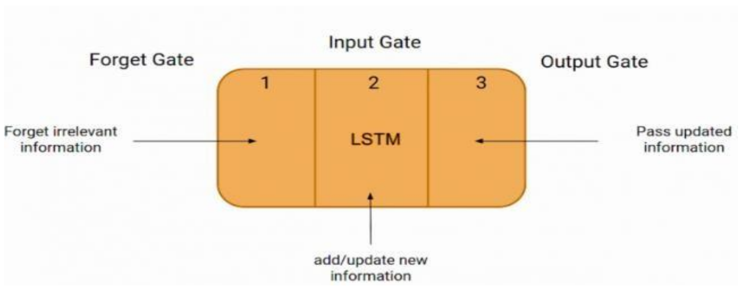


Fig 2 LSTM architecture

3. 2 Algorithm:

Step1:Input Data Representation

Let X(t) be the input data at time step t, representing the features extracted from the s system (e.g., CPU usage, memory usage, frame rate).

Let Y(t) be the output variable at time step t, representing the presence or absence of jank (1 for jank, 0 for non-jank).

Step 2: DNN Model:

Apply a DNN model to process the data input X(t) at each time step t

The DNN model can have multiple hidden layers with appropriate activation functions (e.g., sigmoid, ReLU) to capture complex relationships between the input features.

Step 3: LSTM Model:

Apply an LSTM model to capture temporal dependencies in the input data sequence.

The LSTM model takes the output of the DNN model at each time step t as input and updates its internal state based on the current input and the previous state. The LSTM model can incorporate multiple layers of LSTM units, each equipped with suitable activation functions such as sigmoid and tanh, enabling it to acquire and model long-term dependencies in data.

Step 4: Output Layer:

Apply a fully connected layer with a suitable activation function (e.g., sigmoid, softmax) to obtain the predicted probability of jank at each time step t.

Step 5: Loss Function:

Define a suitable loss function (e.g., binary cross-entropy) to assess the difference between the predicted probability and the actual jank label Y(t) at each time step t. The dissimilarity between the anticipated output and the realized output is quantified by a loss function, such as mean squared error or cross-entropy.

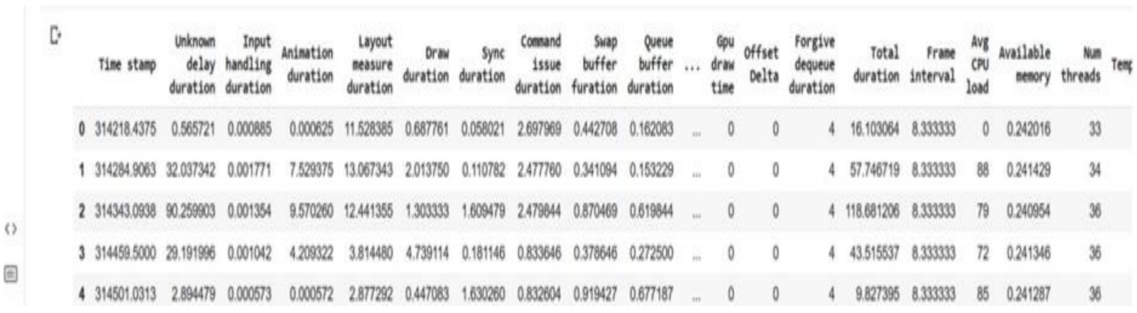
Step 6: Backward Propagation:

Compute the gradients of the cost function concerning the model parameters (including both DNN and LSTM parameters) using backpropagation through time. Update the model parameters using an optimization algorithm (e.g., gradient descent, Adam) to minimize the cost function.

4. Experimental Results and Discussion

We used the dataset containing information about all the processes of GPU and the data such as unknown delay duration, Input handling duration, Animation Duration, Layout measure duration, and so on. To find the best model to predict the jank frame, hyperparameters of both DNN and CNN are modified and the corresponding performances are tabulated.

4.1 Data set



```
               Unknown    Input              Layout                          Command     Swap    Queue        Gpu         Forgive                                  Avg                     
    Time stamp    delay handling Animation   measure     Draw     Sync       issue   buffer   buffer  ...  draw  Offset  dequeue      Total    Frame   CPU  Available      Num  Tem
              duration duration  duration  duration duration duration    duration furation duration       time   Delta duration   duration interval  load     memory  threads
0 314218.4375  0.565721 0.000885  0.000625 11.528385 0.687761 0.058021   2.697969 0.442708 0.162083  ...     0       0        4  16.103064 8.333333     0   0.242016       33
1 314284.9063 32.037342 0.001771  7.529375 13.067343 2.013750 0.110782   2.477760 0.341094 0.153229  ...     0       0        4  57.746719 8.333333    88   0.241429       34
2 314343.0938 90.259903 0.001354  9.570260 12.441355 1.303333 1.609479   2.479844 0.870469 0.619844  ...     0       0        4 118.681206 8.333333    79   0.240954       36
3 314459.5000 29.191996 0.001042  4.209322  3.814480 4.739114 0.181146   0.833646 0.378646 0.272500  ...     0       0        4  43.515537 8.333333    72   0.241346       36
4 314501.0313  2.894479 0.000573  0.000572  2.877292 0.447083 1.630260   0.832604 0.919427 0.677187  ...     0       0        4   9.827395 8.333333    85   0.241287       36
```

Fig 3: Sample data

The total size of the data set DS_MB is 2000 instances gathered from the devices. Another data set DS_AD from the Android devices contains 4200 instances. The data is divided into train and test data sets after preprocessing and normalization. 60 time stamps are used for the LSTM model.

Table 1:Configuration of data sets

	Train	test
DS_MB	1400	600
DS_AD	2940	1260

4.2 Configurations of Convolutional Neural Networks :

The CNN model is built using LENET as the base model. The learning rate is initialized as 0.01. The number of epochs is set to 150 for the CNN model. Stochastic Gradient Descent (SGD) is employed to optimize the model's loss function, while 20% of the training data is designated for validation purposes. Table 2 represents the results produced by the system with CNN and LENET as a classifier.

Table 2:Accuracy of the CNN model

model	**Optimizer**	**accuracy**
CNN	ADAGRADE	84.5
CNN	SGA	85.8.
LENET	ADAGRADE	85.2
LENET	SGA	87.4

Table 3:Confusion matrix for the CNN model

	Dropped	Not Dropped
Dropped	1088	260
Not Dropped	52	600

4.2 Configurations LSTM Neural Networks :

The RNN and LSTM models are defined using 30 and 60 time stamps. The learning rate is initialized as 0.01. The number of epochs is set to 150 for the CNN model The SGD is used

to optimize the loss function of the model 20% of training data is considered as the validation data.
Table 4 represent the results produced by the system with RNN and LSTM as classifier by varying time stamps

Table 4:Accuracy of the CNN model

model	**timestamps**	**accuracy**
LSTM	60	94.7
LSTM	30	92.6
RNN	60	88.5
RNN	30	86.4

When comparing the experimental findings, it becomes evident that the LSTM Model 19 exhibits exceptional efficiency in predicting frame drops, achieving an accuracy rate of 94.7%. It is represented in table 5. This prediction capability enhances the overall user experience by allowing for proactive measures to be taken to avoid jank and ensure smooth performance during UI rendering on Android devices.

Table 5: comparison of the models

	Accuracy	Precision	Recall
CNN	85.8	85.8	82.4
RNN	88.5	86.5	85.8
LSTM	94.7	94.1	93.8

5. Conclusions

This research work focuses on the development of on-device prediction models for efficient jank detection, leveraging Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks. The study explores the application of these deep learning techniques in identifying and mitigating jank issues, with an emphasis on optimizing real-time performance and resource efficiency. The research aims to enhance the overall user experience by addressing jank-related problems in mobile applications. The experimental results indicate that the LSTM Model excels in accurately predicting frame drops, achieving an impressive accuracy rate of 94.7%

This prediction capability enhances the overall user experience by allowing for proactive measures to be taken to avoid jank and ensure smooth performance during UI rendering on Android devices.

References

1. Hochreiter, S. and Schmid Huber, J. (1996). Bridging a long time lags by weight guessing and "Long Short-Term Memory". In Silva, F. L., Principe, J. C., and Almeida,

L. B., editors, Spa Tio temporal models in biological and artificial systems, pages 65-72. IOS Press, Amsterdam, Netherlands. Serie: Frontiers in Artificial Intelligence and Applications, Volume 37.

2. Irwan Bell et al(2021) Revisiting ResNets: Improved training and scaling strategies. Advances in Neural Information Processing Systems (NeurIPS), vol .34,
3. L. Ekonomou, “Greek long-term energy consumption prediction using artificial neural networks,” Energy, vol. 35, no. 2, pp. 512-517, 2010.
4. Lang, K., Waibel, A., and Hinton, G. E. (1990). A time-delay neural network architecture for isolated word recognition. Neural Networks, 3:23-43. Miller, C. B. and Giles, C. L. (1993). Experimental comparison of the effect of order in recurrent neural networks. International Journal of Pattern Recognition and Artificial Intelligence.
5. Levada, Alexandre LM, et al. ”Novel approaches for face recognition: template-matching using dynamic time warping and LSTM Neural Network Supervised Classification.” 2008 15th International Conference on Systems, Signals and Image Processing. IEEE, 2008.
6. Manekar, A., & Pradeepini, G. (2021). Optimizing cost and maximizing profit for multi-cloud-based big data computing by deadline-aware optimize resource allocation. In Recent Studies on Computational Intelligence: Doctoral Symposium on Computational Intelligence (DoSCI 2020) (pp. 29-38). Springer Singapore.
7. Ota, Kaoru, et al.(2017) "Deep learning for mobile multimedia: A survey." ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM) 13.3s pp: 1-22.
8. S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural Comput., vol. 9, no. 8, pp. 1735–1780, 1997.
9. Selouani, S. A., & Yacoub, M. S. (2018,). Long short-term memory neural networks for artificial dialogue generation. In 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), Vol. 1, pp. 761-768
10. V. Polepally, et al(2021) "A Deep Learning Approach for Prediction of Stock Price Based on Neural Network Models: LSTM and GRU," *2021 12th International Conference on Computing Communication and Networking Technologies (ICT)*, Kharagpur, India, pp. 1-4