

**MACHINE-LEARNING BASED QOE PREDICTION  
FOR DASH VIDEO STREAMING**

**TAN JUN YUAN**


**A project report submitted in partial fulfilment of the  
requirements for the award of Bachelor of Engineering  
(Honours) Electrical and Electronic Engineering**

**Lee Kong Chian Faculty of Engineering and Science  
Universiti Tunku Abdul Rahman**

**Jan 2021**

**DECLARATION**

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature :   
\_\_\_\_\_

Name : TAN JUN YUAN  
\_\_\_\_\_

ID No. : 1602288  
\_\_\_\_\_

Date : 10/4/2021  
\_\_\_\_\_

**APPROVAL FOR SUBMISSION**

I certify that this project report entitled “**MACHINE-LEARNING BASED QOE PREDICTION FOR DASH VIDEO STREAMING**” was prepared by **TAN JUN YUAN** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Honours) Electrical and Electronic Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature :   
\_\_\_\_\_

Supervisor : Tham Mau Luen  
\_\_\_\_\_

Date : 4/5/2021  
\_\_\_\_\_

Signature : \_\_\_\_\_

Co-Supervisor : \_\_\_\_\_

Date : \_\_\_\_\_

The copyright of this report belongs to the author under the terms of the copyright Act 1987 as qualified by Intellectual Property Policy of Universiti Tunku Abdul Rahman. Due acknowledgement shall always be made of the use of any material contained in, or derived from, this report.

© 2021, Tan Jun Yuan. All right reserved.

## ACKNOWLEDGEMENTS

I would like to thank everyone who had contributed to the successful completion of this project. First, I wish to express my sincere gratitude to my final year project supervisor, Dr. Tham Mau-Luen, for his previous guidance, advice and insightful comments throughout my research and thesis writing. His experience and insights have enabled me to complete this project successfully.

In addition, I would also like to thank my parents and friends, who had provided me support and encouragement throughout the project. Finally, I also want to express my thankfulness to UTAR for giving me the opportunity and accept me into the graduate program.

## ABSTRACT

Quality of experience (QoE) is an essential metric for video service platforms such as Youtube and Netflix to monitor the service perceived by their end-users. Driven by the popularity of MPEG-Dynamic Adaptive HTTP Streaming (DASH) format among service providers, a plethora of QoE prediction models have been proposed for MPEG-DASH video streaming. However, conventional models are established based on machine learning techniques, which are unable to extract high-level features from low-level raw inputs via a hierarchical learning process. The capabilities of deep learning have paved the new way for more powerful QoE prediction models. The aim of this project is to propose a deep-learning-based QoE prediction method. The starting point of the project is a state-of-the-art framework called DeepQoE, which encompasses three phases: feature pre-processing, representation learning and QoE predicting phase. The framework is further improved by integrating ensemble learning in the prediction phase. Extensive experiments are conducted to evaluate the performance of the proposed QoE prediction model as compared to conventional algorithms. By using a publicly available LIVE-NFLX-II dataset, the newly trained model outperforms not only conventional methods but also the DeepQoE by 0.226% and 0.06% in terms of Spearman Rank Order Correlation Coefficient (SROCC) and Pearson Linear Correlation Coefficient (LCC), respectively.



2.4	Machine Learning	10
2.4.1	Decision Tree	11
2.4.2	Random Forest	12
2.4.3	Gradient Boosting	13
2.4.4	K-Nearest Neighbour (k-NN)	13
2.4.5	Limitations on Machine Learning algorithms on QoE prediction	14
2.5	Deep Learning	14
2.5.1	Restricted Boltzmann machine (RBM)	15
2.5.2	Convolutional Neural Network (CNN)	16
2.5.3	Summary of Deep Learning	16
2.6	Deep Learning Approaches on Video QoE Prediction	17
2.6.1	CNN-QoE for Continuous QoE Prediction	17
2.6.2	DeepQoE	18
<b>3</b>	<b>METHODOLOGY AND WORK PLAN</b>	<b>20</b>
3.1	Overview of Project Work Plan	20
3.2	Environment Setup	21
3.2.1	Linux	21
3.2.2	FFmpeg	21
3.2.3	Caffe	21
3.2.4	Pytorch	21
3.3	Deep Learning Model Framework	23
3.4	Pre-trained Models	25
3.5	Dataset Evaluation – LIVE-NFLX-II	26
3.6	Dataset Pre-processing	27
3.6.1	Feature selection	27
3.6.2	Feature Extraction (C3D & GloVe)	28
3.6.3	Extracting into CSV file	29
3.7	Training for Deep Learning Model	29
3.8	10-Fold Cross-Validation	30
3.9	Evaluation Metrics of QoE Prediction Model	31
3.10	Project Planning and Resource Allocation	31



3.11	Anticipated Problems and Solutions	32
<b>4</b>	<b>RESULTS AND DISCUSSION</b>	<b>33</b>
4.1	Introduction	33
4.2	Hyper-parameter tuning	33
4.3	Evaluation of Representation from Deep Learning Model	36
4.4	Enhancement to the Deep Learning Model with Ensemble Learning	38
4.5	Summary of Results	40
<b>5</b>	<b>CONCLUSIONS AND RECOMMENDATIONS</b>	<b>41</b>
5.1	Conclusions	41
5.2	Recommendations for Future Work	42
	<b>REFERENCES</b>	<b>43</b>
	<b>APPENDICES</b>	<b>49</b>

**LIST OF TABLES**

Table 2.1: 5-point Absolute Category Rating (ACR) scale	5
Table 3.1: Machine Learning Algorithms in Scikit-Learn and Respective Paths	22
Table 3.2: Comparison of public QoE databases for HTTP-based adaptive video streaming (Bampis <i>et al.</i> , 2018)	26
Table 3.3: Features of the LIVE-NFLX-II dataset	27
Table 4.1: Results of Hold-out Test on Different Algorithm	39
Table 4.2: Results of Deep Learning Model Before Ensembling vs after Ensembling	39

## LIST OF FIGURES

Figure 2.1: The SSIM measurement system (Kotevskiss and Mitrevski, 2010)	8
Figure 2.2: Example of decision tree model (Safavian and Landgrebe, 1991)	11
Figure 2.3: Architecture of the RF algorithm (Aung and Hla, 2009)	12
Figure 2.4: Architecture of Restricted Boltzmann Machine (RBM) (Mocanu <i>et al.</i> , 2015)	15
Figure 2.5: Proposed CNN-QoE architecture and residual block used (Duc <i>et al.</i> , 2020)	17
Figure 2.6: Architecture of the DeepQoE framework (Zhang <i>et al.</i> , 2020)	18
Figure 3.1: Flowchart for Project Workflow	20
Figure 3.2: Proposed QoE Prediction Model Framework	23
Figure 3.3: The Architecture of C3D Model labelled with number of output units (Tran <i>et al.</i> , 2015)	25
Figure 3.4: Example Illustration of Feature Extraction of C3D	28
Figure 3.5: Illustration of 10-Fold Cross-Validation (Berrar, 2018)	30
Figure 3.6: Gantt Chart of Task List Project Planning	32
Figure 4.1: Graph of RMSE against Learning Rate	34
Figure 4.2: Graph of RMSE against Training Ratio	34
Figure 4.3: Graph of SROCC against Training Ratio	35
Figure 4.4: Graph of LCC against Training Ratio	35
Figure 4.5: Graph of RMSE using pre-processed data vs representation	37
Figure 4.6: Graph of SROCC using pre-processed data vs representation	37
Figure 4.7: Graph of LCC using pre-processed data vs representation	38

**LIST OF SYMBOLS / ABBREVIATIONS**

ABR	Adaptive Bitrate
ACR	Absolute Category Rating
AI	Artificial Intelligence
ANN	Artificial Neural Network
ASAC	Adaptive streaming of Audiovisual Content
CDN	Content Delivery Network
CNN	Convolutional Neural Network
DASH	Dynamic Adaptive Streaming via HTTP
DL	Deep Learning
DNN	Deep Neural Network
JND	Just Noticeable Difference
k-NN	k-Nearest Neighbor
ML	Machine Learning
MOS	Mean Opinion Score
MSE	Mean Squared Error
LCC	Pearson Linear Correlation Coefficient
PSNR	Peak Signal-to-Noise Ratio
QA	Quality Assessment
QoE	Quality of Experience
RBM	Restricted Boltzmann Machine
ReLU	Rectified Linear Unit
RF	Random Forest
RMSE	Root Mean Squared Error
RRED	Reduced Reference Entropy Differencing
SeLU	Scaled Exponential Linear Unit
SROCC	Spearman Rank Order Correlation Coefficient
SRRED	Spatial Reduced Reference Entropy Differencing
ST-RRED	Spatio-temporal Reduced Reference Entropy Differencing
TCN	Temporal Convolutional Network
TRRED	Temporal Reduced Reference Entropy Differencing
VNI	Visual Networking Index

**LIST OF APPENDICES**

Appendix A: LIVE-NFLX-II Dataset and DeepQoE Download Source	49
Appendix B: Shell Script ‘feature_extraction.sh’ (Extract features from frames for LIVE-NFLX-II)	49
Appendix C: Python Code 1 (PKL-to-CSV converter for LIVE-NFLX-II dataset)	50
Appendix D: Python Code 2 (Deep Learning Training Code)	53
Appendix E: Python Code 3 (Evaluation Code for section 4.4)	55

## CHAPTER 1

### INTRODUCTION

#### 1.1 General Introduction

In the past few years, there is a rapid increase in the usage of mobile devices and multimedia applications. Mobile data traffic is also increasing significantly, with mobile video being one of the main contributors. Based on the Cisco Visual Networking Index (VNI), mobile video traffic contributed 59 per cent of the global mobile data traffic in the year 2017. It is also predicted that the global mobile data traffic will increase up to 7-fold from the year 2017 to 2022 while having 79 per cent of the traffic is in video.

To combat the bandwidth fluctuations, MPEG-Dynamic Adaptive HTTP Streaming (DASH) is now widely adopted in modern major video streaming platforms such as Youtube and Netflix. In MPEG-DASH format, the videos will be separated into different durations of temporal segments, where each segment is encoded at different qualities, resulting in different sizes (Vasilev et al., 2018). This enables DASH clients to have flexible optimization strategies, where they can scale up or scale down the video quality by selecting the most optimum segment for different end-users.

All video streaming services need to find a balance between the operating cost and the quality of service perceived by their end-users. To cope with the increasing video traffic, they need to plan their resource allocation and bandwidth to ensure that the experience of end-users is not affected greatly while being cost-effective. Starting from this premise, it is clear that video quality of experience (QoE) opens the possibilities for content providers to optimize their streaming service strategies.

QoE can be measured as the level of satisfaction or displeasure with the quality of service provided to the end-users. In recent years, QoE has become an important metric for companies to provide insight on their resource allocation and bandwidth provision. The factors that affect the QoE are called influence factors which can be further categorized into system, context and human influence factors. The resolution, bitrate, and demographic of end-users are some of the examples of influence factors.

## 1.2 Importance of the Study

The video streaming market is a big trend that is still growing rapidly in terms of usage and volume. Hence, predicting QoE is an important topic for companies to ensure the overall experience of end-users. Specifically, the focus is on how to improve QoE prediction accuracy so that the outcome can be further utilized by video streaming companies to monitor and increase their QoE provided.

With popular video streaming services like Youtube and Netflix supporting the MPEG-DASH video format, other smaller video streaming companies also need to improve their QoE so that the company can remain competitive with other bigger companies. With an effective video QoE prediction model, companies can gain more insight and have more control over their overall streaming quality and bandwidth.

## 1.3 Problem Statement

Following the booming growth of video streaming traffic, video streaming services such as Youtube and Netflix need to process, store and deliver huge amounts of video data every day. These video streaming providers must improve the QoE of their end-users to remain competitive in the market. Hence, video QoE prediction is essential for video streaming services to determine the quality of the video provided by their services.

Although there are conventional ways of predicting QoE, such as using the reduced reference entropy difference (RRED) and Peak Signal-to-Noise Ratio (PSNR) algorithms, they still have some limitations. The conventional schemes can only fit well only when an ideal network condition is assumed, meaning that video content must be streamed consistently to clients without interruption. Thus, conventional methods are not able to predict QoE with consistent accuracy when rebuffering occurs. This problem becomes more relevant in a MPEG-DASH format, as it is streamed through HTTP, and network condition may differ for different end-users.

Recent years have witnessed the application of machine learning (ML) in boosting QoE prediction accuracy. However, conventional machine learning algorithms are unable to extract high-level features from low-level raw inputs

automatically. Hence, there is a need for a better QoE prediction model that has no dependency on handcrafted feature engineering.

#### **1.4 Aim and Objectives**

The aim and objectives of the project include:

- To study state-of-the-art deep-learning-based QoE prediction model for MPEG-DASH video
- To propose an enhancement to the deep-learning-based QoE prediction model
- To evaluate the performance of the proposed method with respect to the conventional methods

#### **1.5 Scope and Limitation of the Study**

This project focuses on the implementation of a deep learning model to predict the QoE of MPEG-DASH video.

Some limitations are considered in the project. There are PC hardware limitations in terms of QoE model training. Besides, the inference speed of QoE prediction and real-time QoE prediction are excluded from the scope of this project due to limited computational resources.

#### **1.6 Contribution of the Study**

There are many publications and research on QoE prediction models. This project aims to apply deep learning for QoE prediction model to address the disadvantages of conventional QoE prediction models. To overcome the requirements of large training datasets for deep neural networks, transfer learning on pre-trained models was utilized. Finally, the proposed framework introduces ensemble learning to seek further improvement, and the results are compared with the existing QoE prediction algorithms in terms of the collinearity improvement between the predicted QoE and the real QoE.



## **1.7 Outline of Report**

Chapter 1 provides an overview of the importance and problems of conventional QoE prediction in DASH videos and includes the aim and objectives of the project.

The literature review in Chapter 2 highlights the conventional QoE prediction methods and introduces related works on deep learning for QoE prediction. The methodology in Chapter 3 explains the framework proposed for the deep learning model, setup of the training environment and training process.

The results and discussions are provided in Chapter 4. Chapter 5 concludes the results of the deep learning model trained for QoE prediction and provides suggestions for future methodology improvement and future research directions.

## CHAPTER 2

### LITERATURE REVIEW

#### 2.1 Introduction to Video Quality of Experience

To access video QoE, there are two types of video QoE assessment which includes subjective analysis and objective quality modelling. For subjective analysis, it is done by directly giving participants to watch and rate a score for each video clip viewed, whereas objective quality modelling involves the identification of the QoE parameters by using the results from the subjective analysis.

According to Seufert (2019), most conventional QoE assessments use the 5-point Absolute Category Rating (ACR) scale as shown in Table 2.1, which is a type of ordinal rating scale.

Table 2.1: 5-point Absolute Category Rating (ACR) scale

Score	Category
5	Excellent
4	Good
3	Fair
2	Poor
1	Bad

Seufert also stated that the 5-point ACR scale is not consistent and have less meaning between the differences and ratios of each categorical value. Thus, for video QoE prediction, a more accurate metric for test scores can be presented by QoE distributions (Seufert, 2019) by realizing the ACR-scale QoE results as a form of multinomial distribution. QoE distributions allow meaningful evaluation of the QoE scores in a concise way.

## 2.2 DASH Video Streaming

For MPEG-Dynamic Adaptive HTTP Streaming (DASH), it enables video service providers to split the video content into different temporal segments and then offer it in several quality profiles. Each quality profile will then be encoded at different sets of video encoding rate, resolution and codec attributes (Sideris et al., 2015). The video quality is then chosen based on the network rate and the end user's specifications, such as terminal screen resolution and processing power.

Streaming via HTTP can provide some benefits. First, by using HTTP, the video data can be delivered more efficiently as it is delivered in large segments. Second, the video streaming providers do not need to maintain a session state in their servers, saving additional cost for server resources (Sodagar, 2011).

In the MPEG-DASH standard, it does not define any specific adaptive bitrate streaming (ABR) algorithm. Therefore, there are different types of ABR algorithms proposed for seamless MPEG-DASH streaming, such as Adaptive Streaming of Audiovisual Content (ASAC) and Buffer Occupancy based Lyapunov Algorithm (BOLA).

Different adaptive bitrate streaming algorithms can have different effects on the QoE of end-users. Zhao et al. (2017) stated that the optimization of the client-side ABR algorithm and definition of QoE parameters are two important factors to provide a high QoE for end-users streaming via DASH. However, the focus of this project will be on defining the QoE parameters and predicting the QoE for DASH video streaming.

## 2.3 Conventional QoE Prediction Algorithms

To identify the video QoE parameters, there are several conventional methods that can compute the QoE, such as the peak signal-to-noise ratio (PSNR) (Huynh-Thu and Ghanbari, 2012), structural similarity index metric (SSIM) (Kotovski and Mitrevski, 2010) and the spatio-temporal reduced reference entropy differencing (ST-RRED) (Soundararajan and Bovik, 2013).

### 2.3.1 Peak Signal-to-Noise Ratio (PSNR)

Peak Signal-to-Noise Ratio (PSNR) is a QoE prediction algorithm used widely in image and video processing for its fast computation and relatively simpler to understand.

For video sequences, the Mean Squared Error (MSE) is calculated between each pair of the corresponding reference and processed video frames. The equation for the MSE calculation is:

$$MSE(i) = \frac{1}{WH} \sum_{x=0}^{W-1} \sum_{y=0}^{H-1} [Y_r(x, y, i) - Y_p(x, y, i)] \quad (2.1)$$

where

$W$  = frame width (pixels),  $H$  = frame height (pixels)

$Y_r$  = luminance values of the reference video frame

$Y_p$  = luminance values of the processed video frame

Then, the PSNR for each pair of reference and processed video frames is:

$$PSNR(i) = 10 \log_{10} \frac{I^2}{MSE(i)} \quad (2.2)$$

where

$i$  = maximum pixel luminance value

Then, the PSNR value can be expressed as:

$$PSNR = \frac{1}{N} \sum_{i=1}^N PSNR(i) \quad (2.3)$$

According to Huynh-Thu and Ghanbari (2012), the PSNR value can be used as an assessment of video quality across different test cases with fixed content and fixed codec without the presence of freezing.

However, PSNR is unable to assess the relationship between spatial and temporal qualities. Results had shown that PSNR is unable to provide consistent accuracy of quality prediction across different frame rates (Huynh-Thu and

Ghanbari, 2012). Thus, PSNR is an unfeasible quality metric to identify the QoE of different video contents across different video formats or datasets.

### 2.3.2 Structural similarity index metric (SSIM)

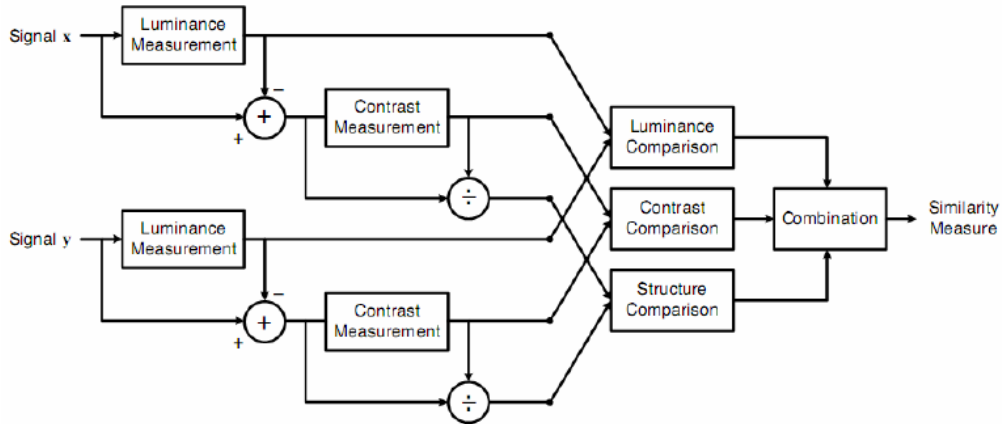


Figure 2.1: The SSIM measurement system (Kotevskiss and Mitrevski, 2010)

Structural similarity index metric (SSIM) is a newer quality compared to PSNR, and the SSIM index extracts the effect of luminosity to evaluate the structural data from the image. Figure 2.1 shows the measurement system of the SSIM index.

The measurements of the SSIM index depends on three parameters which include luminance comparison, structure comparison and contrast comparison. The SSIM index is the combination of the three components, as shown in Equation 2.4:

$$S(x, y) = f(l(x, y), c(x, y), s(x, y)) \quad (2.4)$$

According to Kotevski and Mitrevski (2010), the SSIM metric performs better than the PSNR algorithm and has a comparable performance compared to the Human Visual System. However, SSIM has limitations such as the SSIM values can become largely inverted if there is a bigger change in contrast, colour and brightness.

### 2.3.3 Spatio-temporal reduced reference entropy differencing (STRRED)

For image and video quality assessment (QA) algorithms, they can be categorized as full-reference and no-reference algorithms. While full-reference algorithms such as SSIM or PSNR had seen significant progress, the progress on no-reference QA algorithms is still slow. The reduced reference QA is a no-reference algorithm that involves only sending or supplying some of the data of the reference image that is useful.

The spatial reduced reference entropy differencing (SRRED) index can be computed by using spatial multiscale multi-orientation decompositions of each frame in both distorted and reference videos, while the temporal reduced reference entropy differencing (TRRED) index can be obtained by computing the frame differences between each frame.

As the SRRED and TRRED indices can operate individually, hence the spatial and temporal frequency responses cannot affect each other. Thus, the product form of SRRED and TRRED can be used and form the STRRED index.

$$STRRED_k = SRRED_k TRRED_k \quad (2.5)$$

While the STRRED method shows a better improvement than PSNR and SSIM in some cases (Soundararajan and Bovik, 2013), there are still limitations such as the overall runtime and efficiency of the algorithm.

### 2.3.4 Multiple Linear Regression Analysis

Regression is used to measure the correlativity between the dependent variable (output) and the independent variables (input). By using regression, the numerical relationship between the output and input variables can be defined with mathematical equations.

Multiple linear regression can determine the functional relation between an output variable and a group of input variables. The multiple linear regression model is given by:

$$y = a_0 + a_1x_1 + a_2x_2 + \dots + a_px_p + e \quad (2.6)$$

where  $y$  is the dependent variable,  $x_i$  are different independent variables,  $a_i$  is the weightage of each independent variables (from 1 to  $p$ ),  $a_0$  is the bias or intercept, and  $e$  is the error term.

However, there are some limitations for multiple linear regression analysis. For instance, there may be a possibility of falsely concluding a correlation is a factor. Although linear regression is easier to use, linear regression is less flexible as it is only limited to linear cases, thus cannot fit into some types of non-linear curves.

### **2.3.5 Limitations on conventional QoE prediction methods**

From all the conventional QoE prediction methods mentioned, it is seen that the conventional methods generally have limitations such as they are not flexible enough to make their prediction reliably with different datasets. Hence, machine learning algorithms were introduced to QoE prediction models to achieve an improvement on the accuracy and reliability.

## **2.4 Machine Learning**

Machine learning is a branch or subset of artificial intelligence (AI), and it is used in predictive data analytics (Kelleher, Namee and D'Arcy, 2015). Machine learning can be defined as a process to extract patterns from a huge amount of data from a dataset.

The machine learning model is needed to be trained priorly to predict a specific output. Each data in the dataset contains a label  $y$  and several features  $x$ , and the dataset will be separated into a training and test set. During the training process, the ML model is fed with the label and features to identify the weightage of each feature. There are different types of ML algorithms which include reinforcement learning, supervised learning and unsupervised learning (Ayodele, 2010).

The objective of unsupervised machine learning is to explore a similar group of trends within the data (Guerra et al., 2011). There is no information on the features in the dataset, letting the ML model to identify hidden patterns within the unlabelled dataset.

Supervised learning can be defined as training a data sample from the dataset with the correct features labelled (Sathya and Abraham, 2013). The task

of supervised learning is to train the ML model based on the dataset of  $N$  instances (Guerra et al., 2011), then the predicted outcome of the model will be compared with the actual outcome to determine its accuracy of the machine learning model.

The feasibility of ML algorithms on QoE prediction such as decision tree, random forest and k-nearest neighbours (k-NN) will be discussed in the next section.

### 2.4.1 Decision Tree

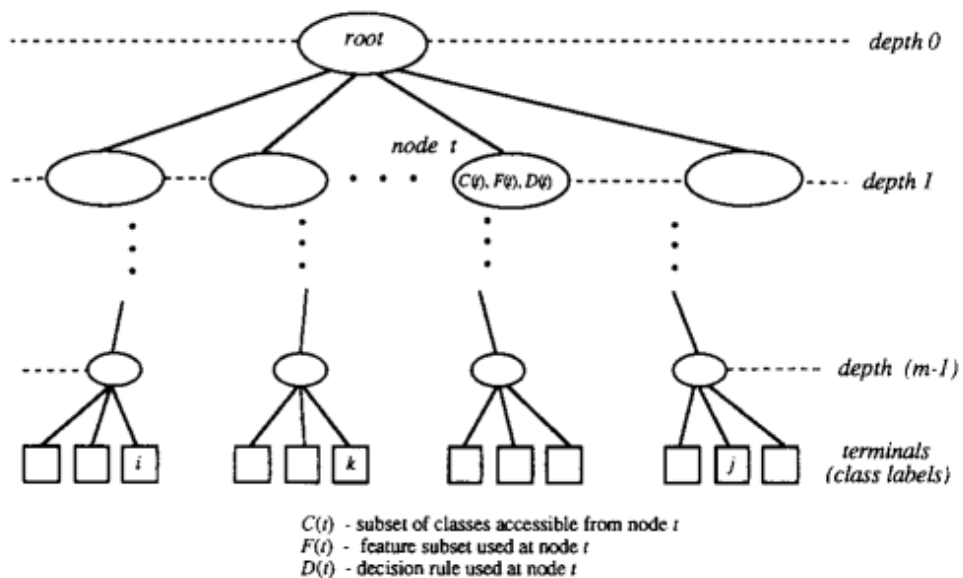


Figure 2.2: Example of decision tree model (Safavian and Landgrebe, 1991)

Decision tree is a type of ML algorithm that uses the divide-and-conquer approach to classification and regression problems (Myles et al., 2004). The decision tree algorithm is a simple to interpret yet fast and effective option for machine learning (Casas et al., 2017). An example of a decision tree model is shown in Figure 2.2.

Casas et al. (2017) had addressed the problem of the reliance on passive traffic QoE prediction for cellular networks relying on passive traffic measurements and QoE crowd-sourced feedback. The model chosen by Casas et al. was based on the decision tree model, as it is stated that the decision tree



model is attractive for large-scale monitoring while having a great performance and a decent prediction speed.

However, the decision tree can be non-robust if the dataset itself is not balanced. For instance, if the mean opinion score (MOS) classes from the dataset are highly imbalanced when training, the model itself may be biased after training. Hence, more work may be needed in order to counterbalance if there are over-represented classes in the dataset.

## 2.4.2 Random Forest

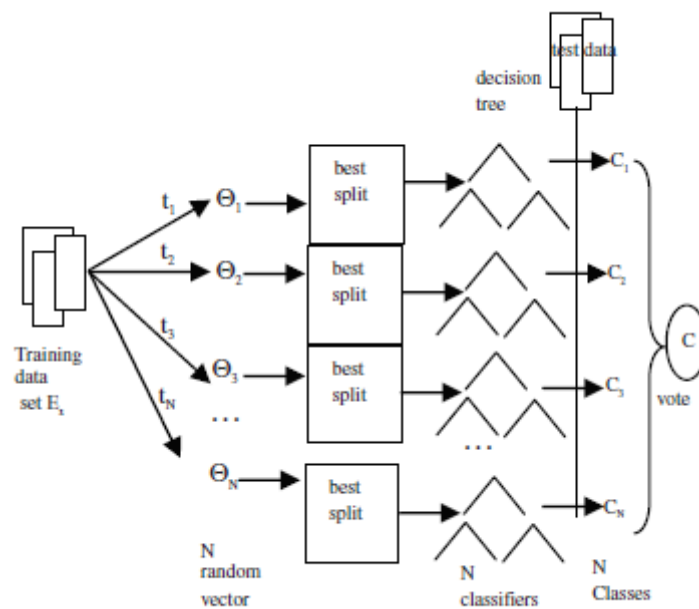


Figure 2.3: Architecture of the RF algorithm (Aung and Hla, 2009)

Random Forest (RF) is a type of ensemble classifier (Mushtaq, Augustin and Mellouk, 2012), meaning that it uses multiple learning algorithms to get a better prediction accuracy. In RF algorithm, it uses several Decision Tree models, which means that each tree in the model will make a prediction, then the class with the most votes will be chosen by the RF model. The architecture of the RF algorithm can be seen in Figure 2.3.

Compared with the decision tree classifier, the experiment carried out by Aung and Hla (2009) showed that the RF-based classification has higher accuracy and the RF method is reliable. However, in some cases, the RF algorithm has been shown to perform poorly in instances of class imbalance

(Segal, 2004). Although additional class weighting parameters can overcome the problem, they will complicate the evaluation process of the model, making the method not feasible.

### **2.4.3 Gradient Boosting**

Gradient boosting (Friedman, 2011) is a type of machine learning boosting for regression and classification problems. It creates a prediction model by ensembling many weak prediction models. Boosting is a technique for primarily reducing bias that can convert weak learners such as decision trees, into a strong learner. Unlike RF which build each tree independently, gradient boosting builds the model step by step by building one tree at a time to improve the shortcoming of existing weak learners.

In 2012, Yu et al. proposed an end-to-end and no-reference QoE prediction for real-time video streaming in 3G networks. The model proposed manage to predict MOS value more accurately compare to G.1070 model by ITU-T. However, the parameter of the gradient boosting model is needed to be tuned carefully to prevent over-fitting. Moreover, gradient boosting may not be a good choice if there is a lot of noise in the dataset, as it can result in overfitting.

### **2.4.4 K-Nearest Neighbour (k-NN)**

The k-NN algorithm is an instance-based ML method (Mushtaq, Augustin and Mellouk, 2012). The key idea for the k-NN algorithm is to classify the new test sample by calculating the distance to the nearest  $k$  values, then assigning it to the majority class of its nearest neighbours.

The Nearest Neighbour rule can achieve high performance consistently without any inferred assumptions (Islam et al., 2008) compared with other supervised learning methods. According to Cicco, Mascolo and Palmisano (2019), the k-NN algorithm had achieved higher accuracy than the ‘global’ algorithm when predicting the QoE of Content Delivery Networks (CDN), and its performance is comparable with the oracle benchmark model from the experiment.

However, there are limitations for the k-NN algorithm. For example, the k-NN algorithm is sensitive to localized data, where local anomalies can affect its outcome significantly (V. Murudkar and D. Gitlin, 2019). Besides, the k-NN

algorithm is not suitable for all cases, specifically for cases with very large datasets (Kordos, Blachnik and Strzempa, 2010), and overfitting problem occurs easily for the algorithm, so it needs to involve techniques such as bootstrapping or cross-validation.

#### **2.4.5 Limitations on Machine Learning algorithms on QoE prediction**

Overall, for conventional ML algorithms such as Decision Tree and RF, it has a limitation such that they rely on hand-crafted features in the dataset. In many cases, different feature extraction methods are needed to find suitable features for a specific task (Rachmadi *et al.*, 2017). Yet, with different extraction techniques, the features may still not be generalized enough.

Along with the development of deep learning algorithms in the past few years, it has slowly been integrated into predictions models, replacing conventional ML algorithms as the method for training prediction models. Rachmadi *et al.* (2017) compared the performances between ML and DL algorithms and found out that DL algorithms had a better performance when compared to conventional ML methods.

### **2.5 Deep Learning**

Deep Learning (DL) is a subfield of ML, in which its concept originated from the artificial neural networks (ANN) approach (Zhang *et al.*, 2017), in which the term ‘deep’ in deep learning referring to the use of many hidden layers in the feedforward neural networks.

DL is a type of representation-learning method, as it can transform representations starting from a low-level raw input into high-level representation or feature. Without any human interference in feature engineering, DL algorithms can model high-order and evaluate feature interactions (Yue *et al.*, 2020) in the dataset. Besides, with deep learning, end-to-end training can be performed for an application because the deep neural network can offer rich representability (Hang, 2018).

In QoE prediction, deep learning models can provide automatic feature engineering across different datasets. Thus, by using DL techniques, the feature of the datasets can be utilized more efficiently as it is able to generalize better across different datasets.

For deep learning, the dependency of data is a major problem, as it needs a large amount of data to learn all the patterns or behaviours. DL prediction models tend to have a higher accuracy when the number of data increases. One of the possible reasons is because the expressive vector space of the DL model must be large to discover more patterns of the data (Tan et al., 2018). By utilizing a machine learning technique called transfer learning, a DL model trained on a particular task can be repurposed to predict a different task. Deep transfer learning can reduce the training data needed and lessen the training time for a particular task.

There are several types of DL algorithms or architecture available such as the Restricted Boltzmann machine (RBM) and Convolutional Neural Network (CNN), which will be discussed in the next subsections.

### 2.5.1 Restricted Boltzmann machine (RBM)

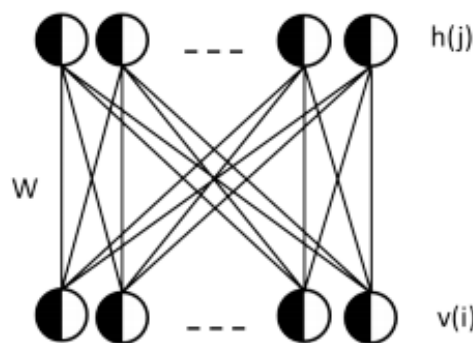


Figure 2.4: Architecture of Restricted Boltzmann Machine (RBM) (Mocanu et al., 2015)

RBM is a type of DL algorithm used to generate stochastic models of artificial neural network (ANN). RBM models consist of Boltzmann Machines, which are neural networks containing a set of bidirectional stochastic nodes. This ensures that the neural networks will be less vulnerable to the local minima, thus able to learn correspondingly to fit a generative model of data.

Due to RBMs' architecture and configuration of neural networks, RBMs and their variants possess great generalization abilities (Mocanu et al., 2015). In the RBM, data patterns will be represented by the activation of the nodes in the

‘visible’ layer while the ‘hidden’ layer of the RBM is used to detect features in the data (Hinton, 2012).

Figure 2.4 shows the visualized architecture of the RBM. In the RBM architecture, there is a full connection between units in the visible layer and the hidden layer. No connection will exist between the units in the same layer.

### **2.5.2 Convolutional Neural Network (CNN)**

Convolutional neural network (CNN) is a type of deep learning neural network, and it has shown great performance in processing structured two-dimensional data such as images and videos (Liu et al., 2017).

CNNs are regularized multi-layer neural networks that have convolution layers and sub-sampling layers which are connected alternatively in the middle part of the network. The function of the convolution layer is to extract features while the sub-sampling layer is to map the features and reducing data size of the signal.

To achieve scale invariance, by passing through an activation function (commonly ReLu or sigmoid function), the features can be mapped in the first sub-sampling layer. The process repeats for the total number of layers, and the feature map can be determined.

There are several advantages of using CNN. In CNN, convolution is used instead of using general matrix multiplication, hence decreasing the number of weights and reducing the overall complexity of the network. Besides, CNN needs lesser pre-processing compared, as manual feature engineering is not needed, and only the filters in CNN are needed to be trained. Parameter sharing is also utilized in CNN during the learning process, leading to a reduction in memory and efficiency improvement.

### **2.5.3 Summary of Deep Learning**

For QoE prediction, DL can provide benefits such as being able to generalize well across databases. Besides, CNN can provide great performance when processing images and videos, which is also suitable for the application for predicting video QoE.

## 2.6 Deep Learning Approaches on Video QoE Prediction

In order to investigate further the feasibility of deep learning on video QoE prediction, several approaches from different research papers were studied.

### 2.6.1 CNN-QoE for Continuous QoE Prediction

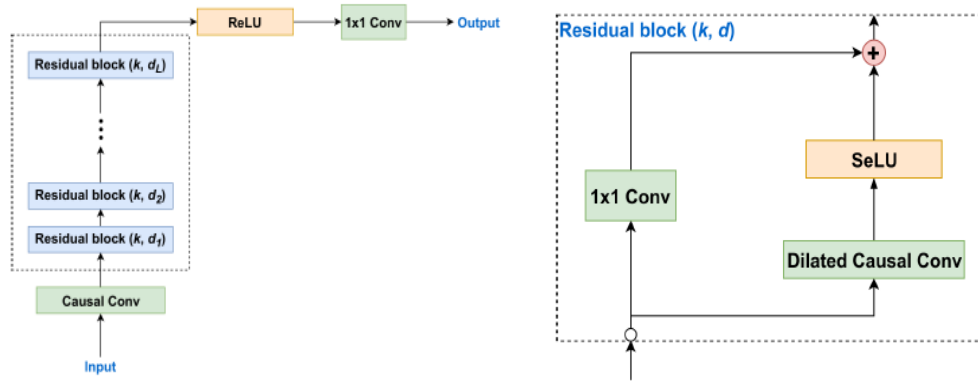


Figure 2.5: Proposed CNN-QoE architecture and residual block used (Duc et al., 2020)

In 2020, Duc et al. (2020) proposed an improved Temporal Convolutional Network (TCN) based model called CNN-QoE. TCN is a variation of CNN and is one of the alternative solutions for sequence modelling tasks such as extracting temporal dependencies in sequential data such as videos.

Figure 2.5 shows the CNN-QoE architecture proposed by Duc et al.. The CNN-QoE model basically utilizes techniques such as 1D convolutions, dilated causal convolution layer and a Scaled Exponential Linear Units (SeLU) activation function.

For the feature engineering part, Duc et al. had used STRRED to measure the visual quality of the video. Features such as playback status, the number of rebuffering events and the time since last rebuffering or bitrate switch were also extracted.

The CNN-QoE model can achieve a great performance in the accuracy of continuous QoE prediction as well as decrease the complexity of computations during training. Thus, this shows that CNN and its variations can achieve great performances for QoE prediction for video streaming such as DASH videos.

## 2.6.2 DeepQoE

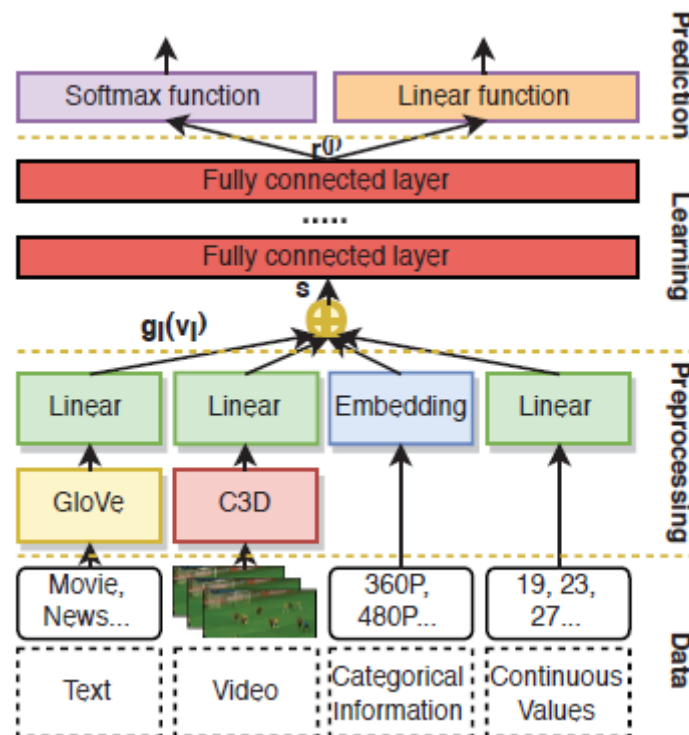


Figure 2.6: Architecture of the DeepQoE framework (Zhang et al., 2020)

According to Zhang et al. (2020), there are two disadvantages of current QoE prediction models, which involves the over-dependence on specific dataset feature extraction and the inflexibility for transfer learning. Hence, a framework called DeepQoE was proposed. Figure 2.6 shows the illustration of the architecture of the DeepQoE framework.

DeepQoE is a three-phase deep-learning-based framework that can provide an end-to-end pipeline for QoE prediction. DeepQoE uses convolutional neural networks (CNN) to perform feature extraction on different datasets, then uses a deep neural network (DNN) to form a representation for the DeepQoE model as input data. The learned representation can be used for both classification and regression tasks.

For the performance of the DeepQoE model on smaller datasets, it shows a comparable result in comparison with other models. It was explained that deep learning models perform better in larger datasets. The representation derived from DeepQoE proved to be more efficient as compared to all other non-machine-learning based algorithms. Specifically, for large datasets, the

DeepQoE framework performed better than other non-machine-learning-based models in terms of accuracy (90.94% vs 82.84%). DeepQoE is a state-of-the-art architecture and is referenced in research papers such as deep neural network comparison by Tao *et al.* in 2019 and the neural network architecture proposed by Hu, Liu and An in 2020.



## CHAPTER 3

### METHODOLOGY AND WORK PLAN

#### 3.1 Overview of Project Work Plan

Figure 3.1 shows the general flow of the project work plan. One publicly available QoE dataset will be evaluated and chosen for the project. Pre-trained models such as C3D are utilized for feature extraction via transfer learning, and several algorithms such as support vector machine (SVM) and Gradient boosting are used for evaluation. The details of the work plan are discussed in subsequent sub-chapters.

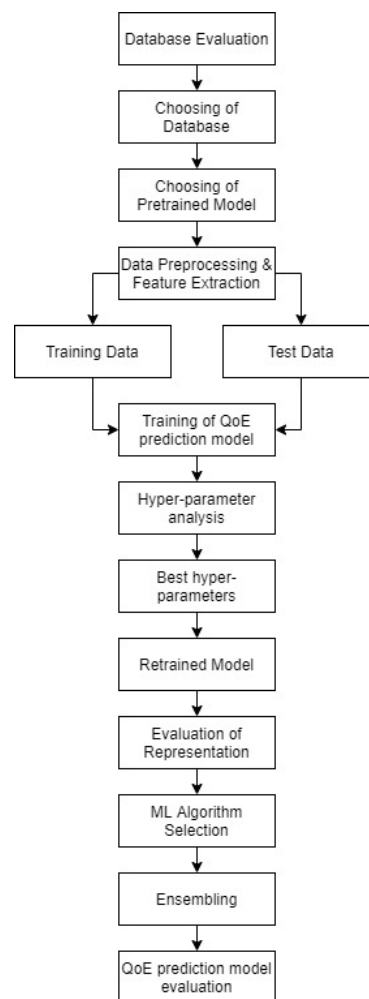


Figure 3.1: Flowchart for Project Workflow

## **3.2 Environment Setup**

### **3.2.1 Linux**

For the project, two virtual machines were set up via VMWare, and the operating systems installed include Ubuntu 20.04 and Ubuntu 16.04 LTS. Ubuntu 20.04 was utilized to perform training and evaluation tasks, while Ubuntu 16.04 was utilized to perform data pre-processing. Two virtual machines were set up due to different requirements, as Ubuntu 20.04 was unable to set up the C3D pre-trained model successfully. Thus, Ubuntu 16.04 virtual machine was used for feature extraction purposes.

### **3.2.2 FFmpeg**

FFmpeg is an open-source command-line-based program that consists of a wide range of libraries, mainly for processing multimedia files and streams. In the project, the FFmpeg library is used to convert the raw video footage in the database into frames so that it can act as an input for the C3D model to extract its features.

### **3.2.3 Caffe**

Caffe is a deep learning framework and is written in C++ with a Python interface. It is an open-source library and it has advantages such as having an expressive architecture, extensible code and decent speed. In this project, Caffe is needed to be set up to use the pre-trained C3D model for feature extraction.

### **3.2.4 Pytorch**

Pytorch is a machine learning framework that is based on the Torch library. It can be used for applications such as natural language processing and image classification. Pytorch is used for the training process of the QoE prediction model in the project.

#### **3.2.4.1 Scikit-Learn**

Scikit-learn is a Python machine learning library. Scikit-learn is built on NumPy, SciPy, and also matplotlib libraries in Python. It features different types of

classification and regression algorithms, including gradient boosting, RF, linear regression and much more.

For the project, linear regression, random forest, gradient boosting and SVM are used for evaluation purposes and the paths of the machine learning models are shown in Table 3.1.

Each algorithm will have different parameters to set, and based on those different parameters; the algorithms will produce different results. Hence, when evaluating the feature representation aspect of the deep learning model, fixed parameters are used during the evaluation process.

Table 3.1: Machine Learning Algorithms in Scikit-Learn and Respective Paths

Classifiers	Paths
Linear Regression	<code>sklearn.linear_model.LinearRegression</code>
Random Forest	<code>sklearn.ensemble.RandomForestRegressor</code>
Gradient Boosting	<code>sklearn.ensemble.GradientBoostingRegressor</code>
Support Vector Machine	<code>sklearn.svm</code>

### 3.3 Deep Learning Model Framework

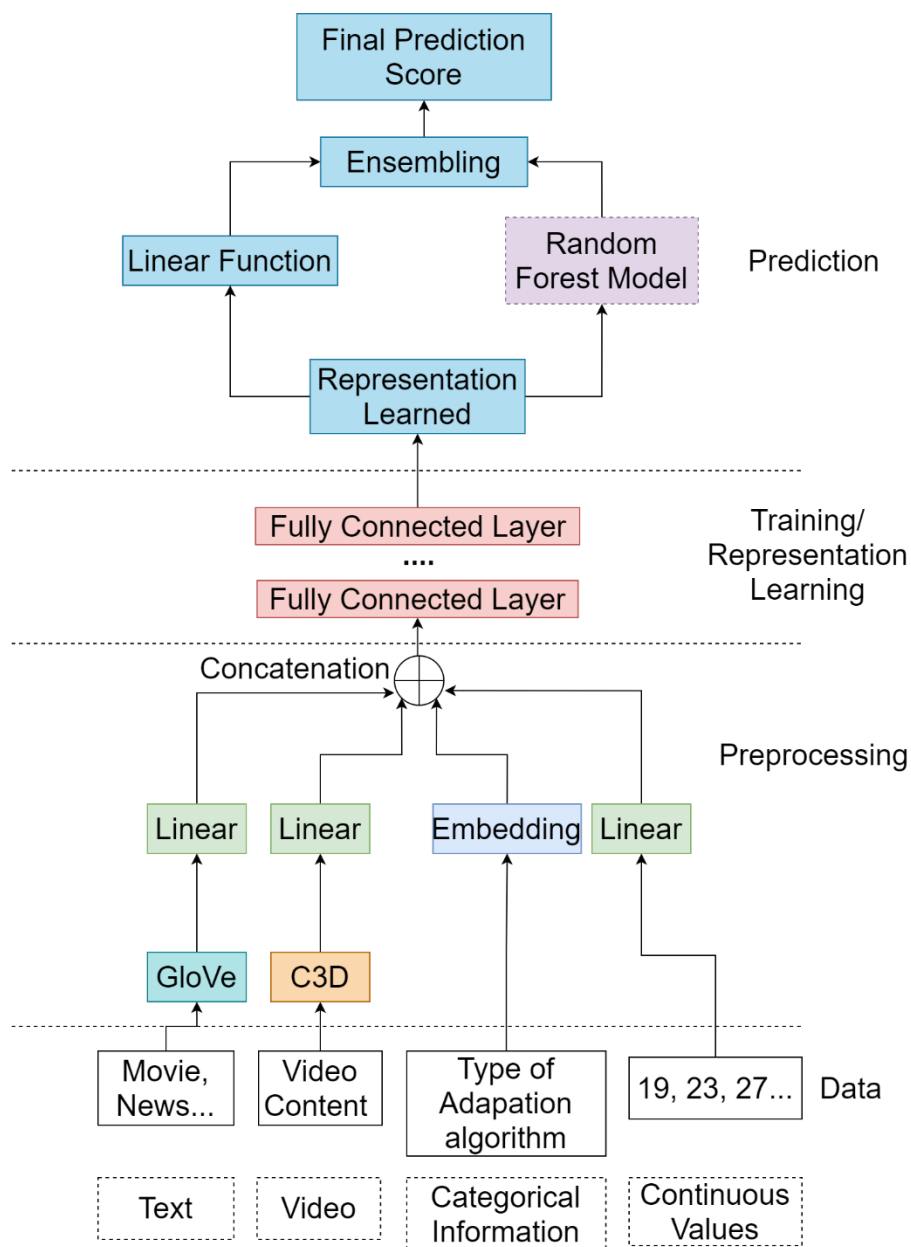


Figure 3.2: Proposed QoE Prediction Model Framework

Figure 3.2 shows the proposed deep learning model framework in this project. The deep learning model architecture is based on the DeepQoE model, with an additional step to perform ensemble learning. The framework integrates a random forest model to perform ensembling with a linear layer together. The purpose of performing ensembling is to further enhance the accuracy of the deep learning model by exploiting the advantages of different algorithms used. Random forest is great with high dimensional data, which is suitable for the

framework as it takes in the high-dimension representation from the deep learning model as input. Besides, the random forest model has the versatility to perform both classification and regression problems. With the ensembling of random forest model, the model will have a lower chance of overfitting and can achieve better performance.

The neural network consists of four phases, which includes data separation, data pre-processing, learning or training phase and prediction phase. For the data separation phase, the raw input features are categorized into four categories, which include video, text, categorical information and continuous values.

After the features are categorized, the data will be pre-processed differently according to the category. For video footage, the video will be extracted and transformed into a 512-dimensional vector by C3D. The text category such as the video genre will be processed by GloVe. Besides, the categorical information will be pre-processed by an embedding layer, while the continuous values will be pre-processed by a linear layer. The details of the pre-trained models will be discussed in the next subchapter.

Then, the feature vectors obtained from the data pre-processing phase will be concatenated into one feature vector. It will then be fed into the neural network for the learning phase after it is split into a training set and test set. The vector will be passed through a number of fully connected layers, and a representation will be learnt before fed into an output linear layer and the random forest model.

For regression tasks, the number of features for each individual trees are recommended to be  $N/3$  (Hastie, Tibshirani and Friedman, 2008), where  $N$  is the number of features. Thus, this recommended value will be chosen as a default parameter for random forest in this project.

Finally, ensembling is performed by taking the average of the prediction from the linear layer and the random forest model. The results of the framework can now be evaluated.

### 3.4 Pre-trained Models

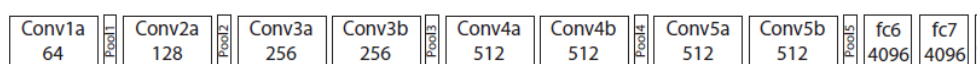


Figure 3.3: The Architecture of C3D Model labelled with number of output units (Tran et al., 2015)

The C3D model (Tran et al., 2015) is a modified version of Caffe to support 3D CNNs. The C3D model can be used to train, fine-tune or test 3D ConvNets efficiently. Figure 3.3 shows the architecture of the C3D and the respective output units for each section. There is a C3D pre-trained model available, which is trained on the Sports-1M dataset, and the tools for feature extraction was available open-source. C3D extracts features from 16 continuous frames with an 8-frame overlap between two consecutive clips; thus it can be utilized in this project to extract video information for model training. With transfer learning, the neural network can utilize the learned features from the C3D model instead of training from scratch. This brings a huge advantage as the C3D is trained on a huge dataset; thus it can reduce the dependency of training a video feature extraction model. Hence, the C3D pre-trained model is utilized to extract features from the frames of the videos in this project.

The GloVe model (Jeffrey, Socher and Manning, 2014) is pre-trained on Wikipedia and is an unsupervised learning algorithm to obtain a vector representation of words. For the framework proposed, the GloVe is used to extract the words into a 50-dimensional vector. The pre-trained model and code are available publicly; hence it is used to extract features from text-type data in this project.

### 3.5 Dataset Evaluation – LIVE-NFLX-II

The LIVE-NFLX-II database (Bampis *et al.*, 2018) contains 420 video streams that are derived from 15 different original videos of diverse content, which includes action, documentary, sports, animation and video games. The video sources were rendered under different lighting conditions. Besides, the videos in the database were derived using seven different network traces to simulate the real-life effects of network variability during the HTTP-based adaptive video streaming.

In the database, four different client-based ABR algorithms were also deployed to create a rich streaming QoE database. The spatial and temporal activities were also recorded. Moreover, a wide range of encoding bitrate and qualities was used when deriving the videos. The videos were given scores by the subject in a continuous manner, capturing the time-varying nature of QoE due to conditions such as rebuffering and scene cuts. Z-score normalization was applied to the MOS in the database. The normalized continuous scores per subject were averaged to compute a continuous MOS score for each frame.

A comparison was also made by Bampis *et. al.* (2018) to compare the differences between other HTTP-based adaptive video streaming databases. The comparison can be seen in Table 3.2.

Table 3.2: Comparison of public QoE databases for HTTP-based adaptive video streaming (Bampis *et al.*, 2018)

Description	[27]	[9]	[7]	[30]	[29]	[18]	[10]	[19]	[11]	[31]	LIVE-NFLX-II
client adaptation	X									X	X
continuous QoE		X				X		X	X		X
actual network traces	X										X
buffer model	X								X	X	X
public					X	X	X	X		X	X
> 400 test videos										X	X
> 60 subjects	X		X								X
rebuffering + quality	X	X		X			X		X	X	X
content-based encoding			X		X						X

### 3.6 Dataset Pre-processing

#### 3.6.1 Feature selection

Table 3.3: Features of the LIVE-NFLX-II dataset

Number of features: 36 features	
<input checked="" type="checkbox"/> c/h adaptation_algorithm ✓	<input type="checkbox"/> per_segment_encoding_width
<input type="checkbox"/> buffer_evolution_sec	<input type="checkbox"/> playback_duration_sec
<input checked="" type="checkbox"/> c/h content_name	<input type="checkbox"/> playout_bitrate ✓
<input checked="" type="checkbox"/> c/h content_name_acronym	<input type="checkbox"/> PSNR ✓
<input type="checkbox"/> content_spatial_information ✓	<input type="checkbox"/> rebuffer_duration_sec
<input type="checkbox"/> content_temporal_information ✓	<input type="checkbox"/> rebuffer_number
<input type="checkbox"/> continuous_zscored_mos ✓	<input checked="" type="checkbox"/> c/h reference_yuv_video
<input checked="" type="checkbox"/> c/h cropping_parameters	<input type="checkbox"/> retrospective_zscored_mos
<input checked="" type="checkbox"/> c/h distorted_mp4_video	<input type="checkbox"/> scene_cuts ✓
<input type="checkbox"/> frame_rate ✓	<input type="checkbox"/> scene_cuts_detected ✓
<input type="checkbox"/> height	<input type="checkbox"/> selected_streams
<input type="checkbox"/> is_rebuffered_bool ✓	<input type="checkbox"/> SSIM ✓
<input type="checkbox"/> MSSIM ✓	<input type="checkbox"/> STRRED ✓
<input type="checkbox"/> N_playback_frames	<input type="checkbox"/> throughput_trace_kbps
<input type="checkbox"/> N_rebuffer_frames	<input checked="" type="checkbox"/> c/h throughput_trace_name
<input type="checkbox"/> N_total_frames	<input type="checkbox"/> video_duration_sec
<input type="checkbox"/> per_segment_encoding_height	<input type="checkbox"/> VMAF ✓
<input type="checkbox"/> per_segment_encoding_QP	<input type="checkbox"/> width

There are a total of 36 features available in the data files of the LIVE-NFLX-II dataset. The metrics are provided in .pkl and .mat format. Hence, to simplify the training process, only 14 useful independent features are chosen and will be converted to a CSV file for easier processing, as shown in Table 3.3.

For this project, only continuous values and categorical information were chosen, such as playout bitrate, adaptation algorithm, spatial information and temporal information, frame rate, resolution of the video, rebuffering and the scene cuts detected for each frame.

There are two types of dependent data in the LIVE-NFLX-II dataset. This includes the ‘retrospective\_zscored\_mos’ and ‘continuous\_zscored\_mos’. For this project, since the C3D is extracting the video features every 16 frames, hence the ‘retrospective\_zscored\_mos’ is not used as the QoE prediction model will predict the z-score MOS continuously.

Besides, specific features such as ‘content\_name’, ‘content\_name\_acronym’, ‘reference\_yuv\_video’ and ‘distorted\_mp4\_video’ were not included as they are dataset-specific features such as video name or



the file name. This is because the aim of the project is to train a QoE prediction model that can generalize well; thus these data will not be used.

Features such as ‘buffer\_evolution\_sec’ were not used as this feature is affected by the per second playout bitrate; hence it is redundant. Redundant data such as ‘N\_rebuffer\_frames’, ‘N\_playback\_frames’, ‘N\_total\_frames’, ‘rebuffer\_duration\_sec’, ‘playback\_duration\_sec’, ‘video\_duration\_sec’ and ‘rebuffer\_number’ were also not used as there is a feature called ‘is\_rebuffered\_bool’, which provides a rebuffer status (0 or 1) for every frame for the video.

For the QoE prediction model training, video information such as ‘frame\_rate’, ‘adaptation\_algorithm’, ‘content\_spatial\_information’ and ‘content\_temporal\_information’ are used. Besides, the playout bitrate and the scene cuts in the video were used. Finally, continuous value features such as the VMAF, PSNR, SSIM, MSSIM and STRRED were also used as a feature.

### 3.6.2 Feature Extraction (C3D & GloVe)

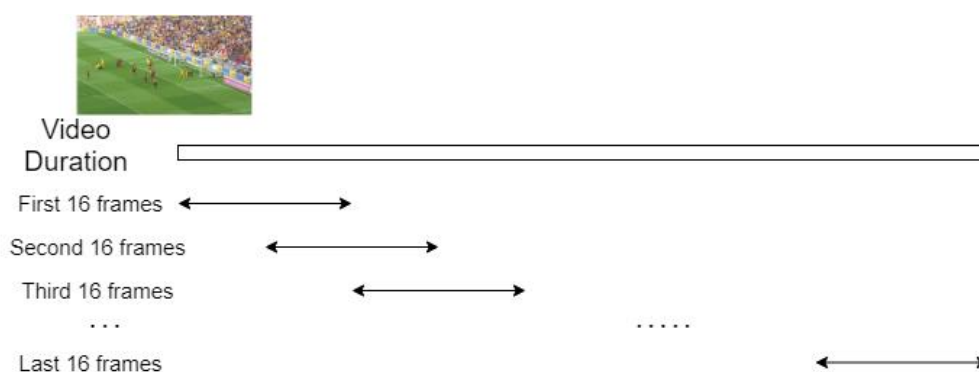


Figure 3.4: Example Illustration of Feature Extraction of C3D

Additionally, in the framework proposed, the raw footage of the video will be extracted by a pre-trained model called C3D. First, the mp4 raw video will be extracted into frames by using FFmpeg in the Linux command terminal. After all the frames of each video have been extracted, C3D will extract every 16 frames of a video into a 512-dimension vector. Eight frames are overlapped between each extraction so that the temporal feature of the video will not be lost. The illustration of the feature extraction for C3D is shown in Figure 3.4.

For this dataset, the GloVe pre-trained model is not utilized. This is because that there are no text features or data such as video genre in the LIVE-NFLX-II dataset. However, the GloVe feature extraction code is still available in the source code.

### **3.6.3 Extracting into CSV file**

The information for all 420 videos are provided in individual .pkl files and contains redundant information such as the video file name. Hence, to simplify the training process, all features chosen are extracted to a .csv file instead. A python code is utilized to organize all of the training data needed, which includes the 512-dimension feature extracted from the C3D pre-trained model.

As mentioned in the previous sub-chapter, the 512-dimension feature is extracted from every 16 frames in each video. Hence, when compiling into the CSV file, the average data for the 16 frames are obtained. Then, the z-scored continuous MOS is also averaged to serve as the true value for the dependent variable. All data for every 16 frames of the videos are saved in the CSV file.

## **3.7 Training for Deep Learning Model**

Before the training process begins, the dataset is split into a training set and a testing set, which will be a 90% and 10% ratio in this project. Then, the training data will be fed for training, while the test set will be used for model evaluation. This process is called the hold-out test.

During the deep learning model training, the features are mapped accordingly before being fed into three fully connected layers in the training phase. Firstly, the adaptation algorithm is mapped to a 10-dimension vector after an embedding layer. Features such as frame rate, 'rebuffer bool', scene cuts, playout bitrate are each transformed into 5-dimension vectors by a linear layer. Metrics such as MSSIM, VMAF, STRRED, SSIM and PSNR are encoded into five 10-dimension vectors. The C3D extracts raw video footage into a 512-dimension vector, which will be encoded into a 412-dimension vector.

All feature vectors are then concatenated into a single feature vector before being fed into the first fully connected layer of the neural network. There are three fully connected layers in the neural network for this project, whereas

the activation function for the output layer is a linear function. Dropout with a ratio of 0.5 is applied to these layers to prevent overfitting.

In the training stage, Adadelta is used as the optimizer and the number of training epoch is 200. After the dimensions and layers in the neural network model is defined, the training process is carried out via Pytorch. After the model is successfully trained, the results can be evaluated via Scikit-Learn.

### 3.8 10-Fold Cross-Validation

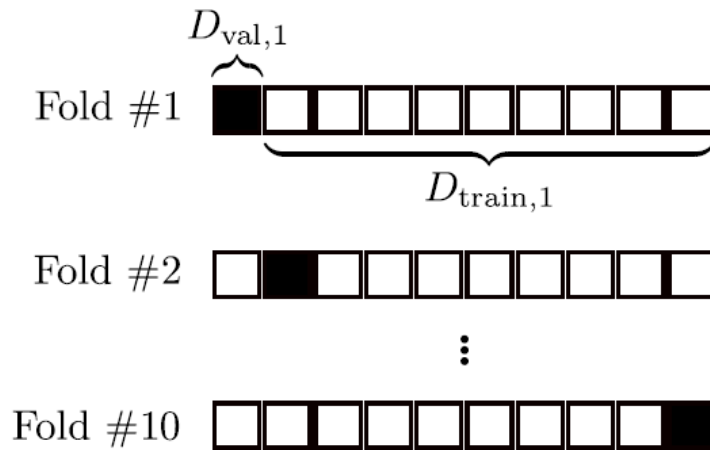


Figure 3.5: Illustration of 10-Fold Cross-Validation (Berrar, 2018)

In this project, 10-fold cross-validation is carried out during hyper-parameter analysis of the deep learning model. The illustration of 10-fold cross-validation is shown in Figure 3.5. 10-fold cross-validation is usually preferred over than hold-out test, as it provides the opportunity to train on multiple train-test splits instead of only one train-test. This will prevent a biased result, as the hold-out test score is dependent on the train-test split. However, 10-fold cross-validation is more computationally expensive than a hold-out test, especially when there is a huge amount of data.

### 3.9 Evaluation Metrics of QoE Prediction Model

To evaluate the QoE prediction model trained, there will be several metrics that can be considered, which includes the Root Mean Square Error (RMSE), the Pearson Linear Correlation Coefficient (LCC) and Spearman Rank Order Correlation Coefficient (SROCC).

RMSE is the standard deviation of the error of predictions. It measures how spread out the prediction errors are from the regression line. As the task performed by the deep learning model in this project is regression, hence RMSE is used instead of accuracy. RMSE can be defined as:

$$RMSE = \sqrt{\overline{(f - o)^2}} \quad (3.1)$$

where  $f$  is the expected values and  $o$  is the predicted value. The bar above the squared difference is the mean.

By measuring LCC, the degree of linearity can be ensured as it measures the linear correlation between two variables which will be subjective and predicted QoE. A higher value is preferred as it identifies the degree of simplicity of the trained ML model.

SROCC measures the monotonic relationship between 2 variables, which will be the predicted MOS and the real MOS in this project. A higher value of SROCC will be better in this case, as a positive value means that a variable value will increase monotonically when the other variable increases.

### 3.10 Project Planning and Resource Allocation

Figure 3.6 shows the Gantt chart planned for the project based on the project workflow mentioned in Section 3.1. Time and processing speed were taken into account for the planning and resource allocation for the overall timeline of the project.

No	Task Name	Jan. 2021		Feb. 2021				Mar. 2021				Apr. 2021					
		18/1	25/1	1/2	8/2	15/2	22/2	1/3	8/3	15/3	22/3	29/3	5/4	12/4	19/4		
1	Choosing of Database	█															
2	Choosing of Pre-trained models		█														
3	Environment Setup			█													
4	Data Preprocessing & Feature Extraction				█												
5	Training of QoE Prediction Model					█											
6	Hyper-parameter analysis								█								
7	Retraining of QoE Prediction Model								█								
8	Algorithm Selection											█					
9	Evaluation of QoE Prediction Model											█					
10	Poster Preparation											█					
11	Report Writing				█												

Figure 3.6: Gantt Chart of Task List Project Planning

### 3.11 Anticipated Problems and Solutions

There are several problems that are anticipated. One of the problems is that different datasets will have different metrics available, while the impact of the metric cannot be fully known. Thus, deep evaluation of datasets must be carried out before choosing for feature extracting or QoE model training. Besides, it is expected that the physical memory or processing power of the personal computer alone is not sufficient to run the training processes. However, for this project, the CUDA version of the code is avoided, and the CPU is being utilized for the training and evaluation process instead.

## CHAPTER 4

### RESULTS AND DISCUSSION

#### 4.1 Introduction

To evaluate the deep learning model framework proposed in this project, several tests were done. As the deep learning model is performing regression tasks, the SROCC and LCC values will be used as evaluation metrics. Besides, RMSE is also included during evaluation.

Measures such as hyper-parameter tuning, the feature extraction ability of the deep learning model and the overall predicting results of the deep learning model are evaluated. The detailed results from the three scenarios are discussed in the following sub-chapters.

#### 4.2 Hyper-parameter tuning

Hyper-parameter analysis was conducted to find the best parameter for the proposed deep learning QoE prediction model framework. Besides, it can show the overall effectiveness of the proposed framework. In this project, the analysis of the learning rate and the training ratio was conducted. 10-fold cross-validation was conducted during the analysis to get the best hyper-parameter.

Figure 4.1 shows the performance of the deep learning model on different learning rates. Learning rates of 0.001, 0.005, 0.01, 0.05, 0.1 and 1 were chosen to perform training. Among the learning rates, the lowest RMSE was achieved at 0.1 learning rate, having an RMSE of 0.3381. It is important to note that the framework performed rather consistent from 0.005 to 1 learning rate. Since 0.1 learning rate achieved the lowest RMSE, it is used as the default learning rate in the project.

The second hyper-parameter to be analyzed is the training ratio. In this part, 50%, 60%, 70%, 80% and 90% of the LIVE-NFLX-II dataset were used as training set while the same 10% will be used as the test set. The RMSE, SROCC and LCC values were recorded in Figure 4.2, Figure 4.3 and Figure 4.4.

From the results, it can be observed that the performance of the deep learning model improves steadily from a 0.41704 to 0.3496 RMSE when the

size of the training set increases. The deep learning model has the lowest RMSE and the highest SROCC and LCC values when the training ratio is 90%. This shows that the deep-learning-based method can scale effectively with more data.

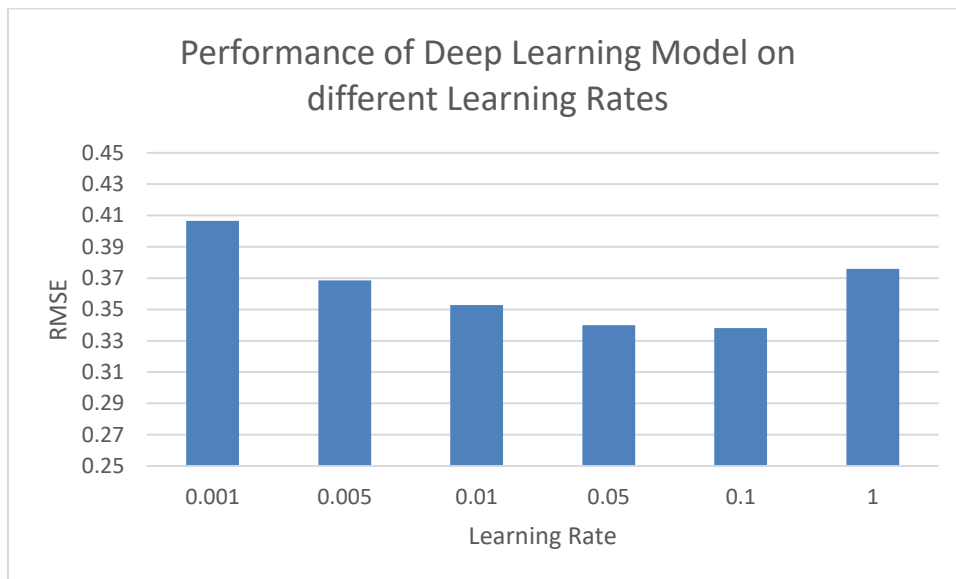


Figure 4.1: Graph of RMSE against Learning Rate

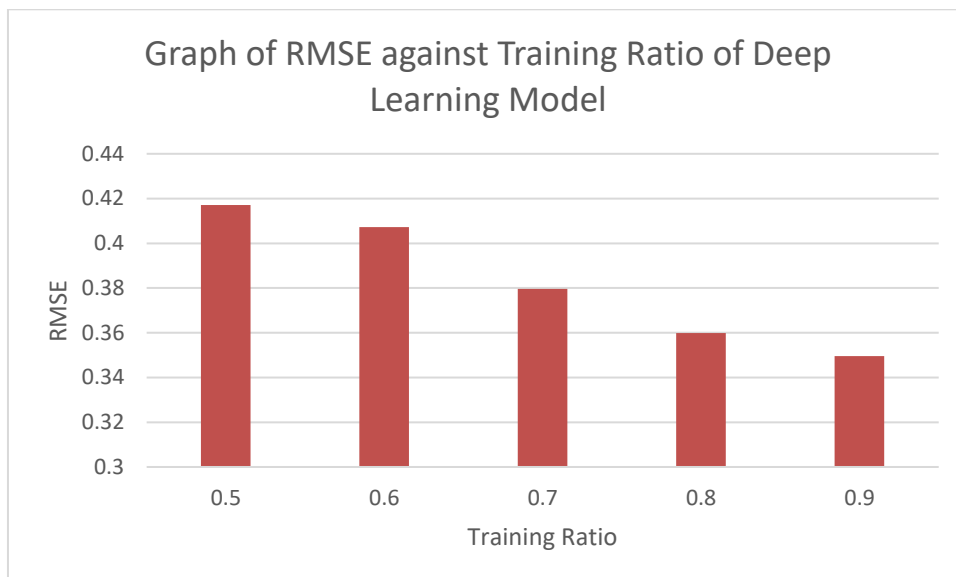


Figure 4.2: Graph of RMSE against Training Ratio

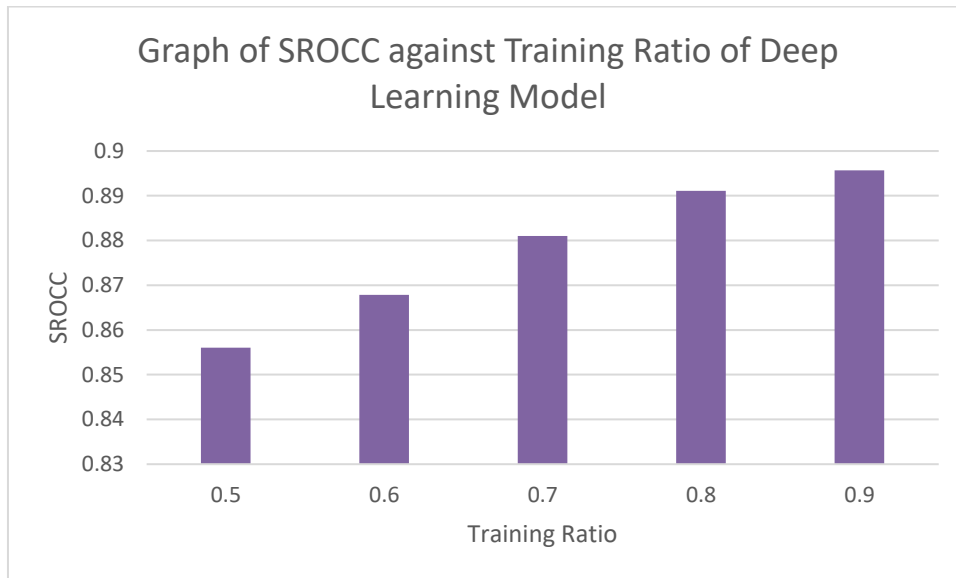


Figure 4.3: Graph of SROCC against Training Ratio

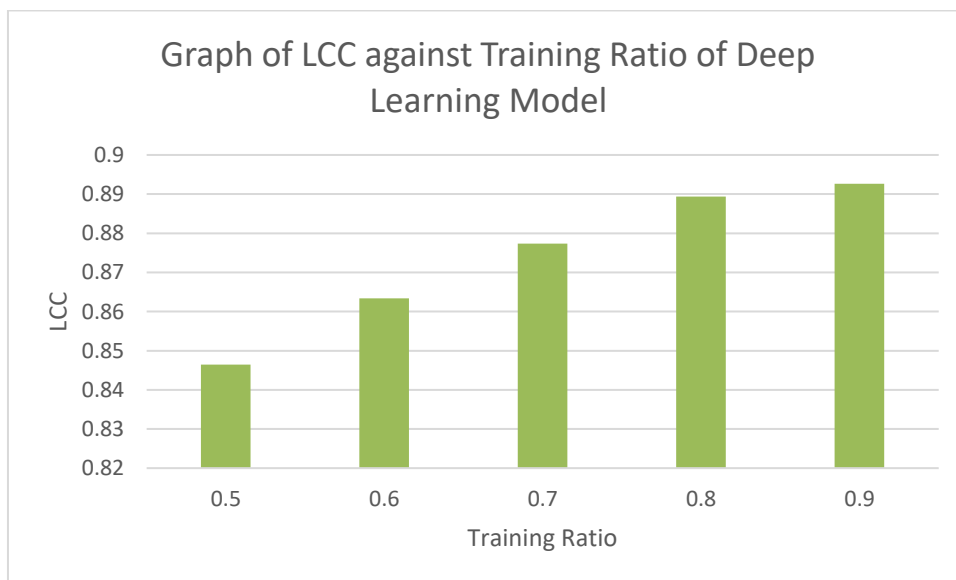


Figure 4.4: Graph of LCC against Training Ratio



### 4.3 Evaluation of Representation from Deep Learning Model

In this section, to show that the deep learning model framework can function well as a representation learning tool, the output from the last connected layer is extracted from the neural network. The 256-dimension vector is retrieved from the deep learning model and fed into other learning algorithms. Linear regression, gradient boosting, random forest and SVM were the algorithms chosen for this subsection.

To have a fair comparison, the RMSE, SROCC and LCC are being compared between using representation from the deep learning model and using the pre-processed features from the LIVE-NFLX-II database. The parameters for each algorithm are kept the same for a fair comparison.

For linear regression and SVM, default parameters are used when training with representation and with pre-processed data. For Gradient Boosting, the number of estimators is set to 500, the max depth is set to 4, the minimum samples split is five, and the learning rate is set to 0.01. For the Random Forest algorithm, the random state is set to be the same value, the number of estimators is set to be 100, and the max features are set to be  $N/3$ , where  $N$  is the total number of features. All source codes are available in the appendix.

After the hold-out test has been done, the RMSE, SROCC and LCC values are recorded, and the results are drawn in Figure 4.5, Figure 4.6 and Figure 4.7.

The results of all algorithms are then shown in separate graphs. As it can be seen, the RMSE of all algorithms dropped while SROCC and LCC improved. This shows that the representation generated by the deep learning model is useful.

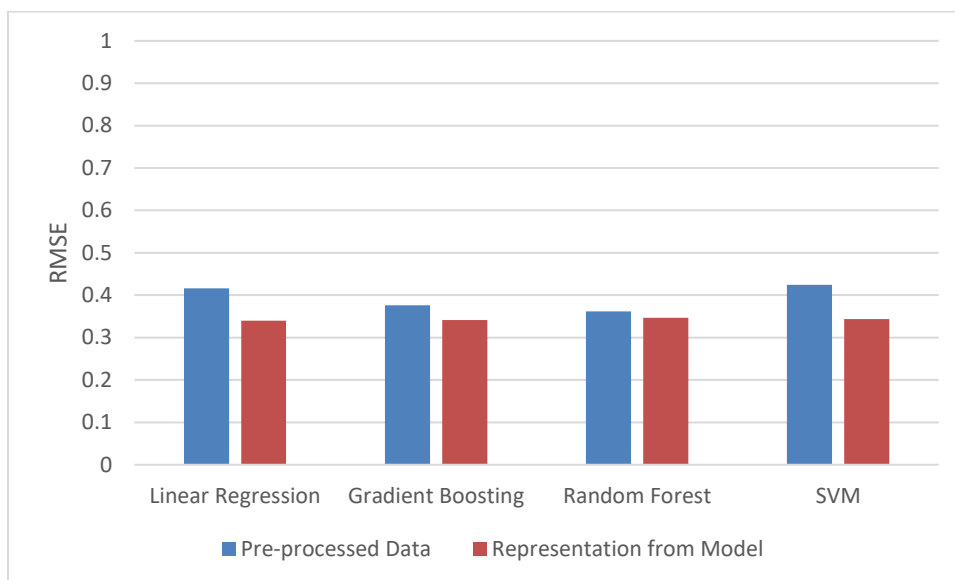


Figure 4.5: Graph of RMSE using pre-processed data vs representation

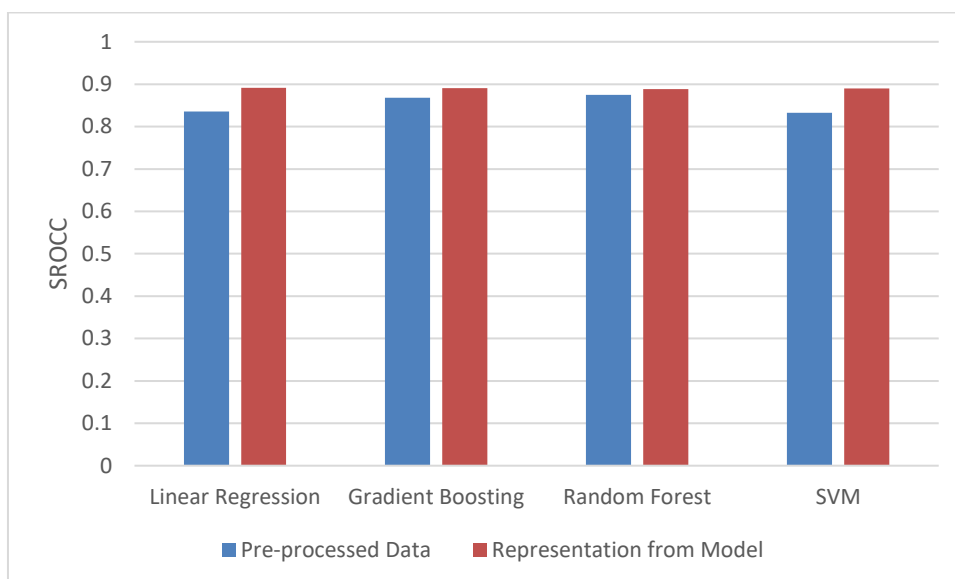


Figure 4.6: Graph of SROCC using pre-processed data vs representation

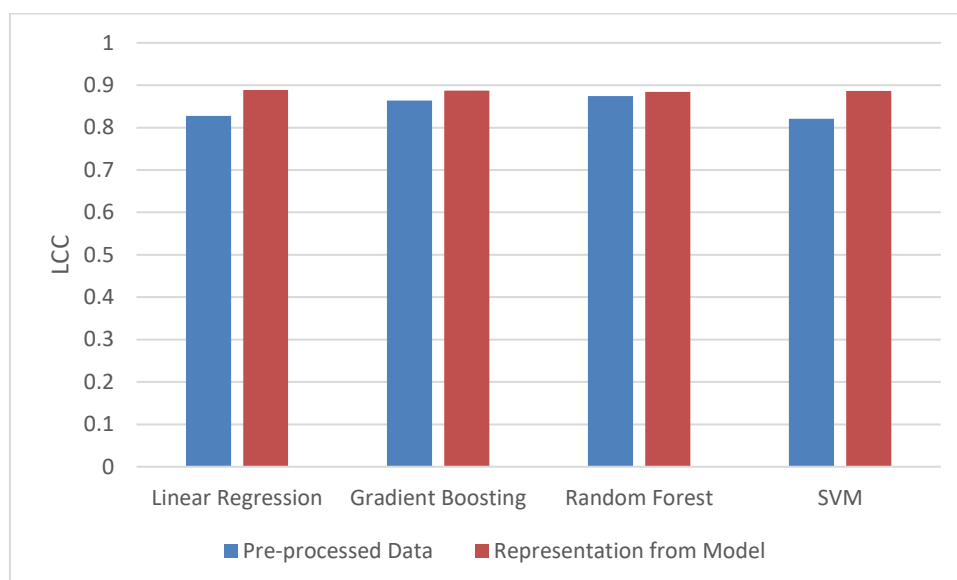


Figure 4.7: Graph of LCC using pre-processed data vs representation

#### 4.4 Enhancement to the Deep Learning Model with Ensemble Learning

In this section, the proposed framework in this project is being compared with the DeepQoE framework by Zhang et al.. For a fair comparison, the DeepQoE framework is reproduced based on the original source code from the author on the Github website: <https://github.com/cap-ntu/DeepQoE>. A slight modification is made on the neural network due to the usage of different datasets. However, the main structure and logic of the code are maintained the same.

The effectiveness of the deep learning model is evaluated. The deep learning framework proposed is compared with four different algorithms in a hold-out test. Due to computational limitation, a hold-out test is performed in this section. Hold-out test can evaluate the performance of different algorithms on unseen data. The performance of random forest, linear regression, gradient boosting and SVM are compared, and the results are shown in Table 4.1.

Table 4.1: Results of Hold-out Test on Different Algorithm

<b>Metrics</b>	<b>Deep Learning</b>	<b>Random Forest</b>	<b>Linear Regression</b>	<b>Gradient Boosting</b>	<b>SVM</b>
RMSE	0.33768	0.34300	0.41627	0.37605	0.42420
SROCC	0.89266	0.88557	0.83528	0.86781	0.83260
LCC	0.89065	0.88627	0.82768	0.86359	0.82065

From the results, it is observed that the performance with the proposed deep learning model achieved the best results among the other algorithms. The deep learning model has the lowest RMSE of 0.33768 and the highest SROCC and LCC values of 0.89266 and 0.89065. This can be attributed to the feature extraction ability of deep learning, as discussed in the previous subsection.

To further improve the results of the deep learning framework, an additional ensembling step is proposed in this project. A additional random forest is used to perform ensemble learning with the linear layer. The RMSE, SROCC and LCC were recorded for the deep learning model before ensembling and after ensembling. The results are then evaluated and compared in Table 4.2.

Table 4.2: Results of Deep Learning Model Before Ensembling vs after Ensembling

<b>Metrics</b>	<b>Deep Learning Model Before ensembling</b>	<b>After Ensembling</b>
RMSE	0.33768	0.33626
SROCC	0.89266	0.89468
LCC	0.89065	0.89116

From the results, it can be observed that the performance of the proposed deep learning framework had slightly better results after including random forest model for ensembling. It had improved by a slight 0.226% for SROCC and 0.06% for LCC. This shows the potential of the application of ensemble learning in the deep learning framework.

Hence, it can be observed that deep learning is effective in predicting QoE, achieving the best results among other shallow learning algorithms. Besides, the deep learning model can be further improved by utilizing ensemble learning in the predicting phase, exploiting advantages from different algorithms such as random forest.

#### **4.5 Summary of Results**

In this project, it can be observed that deep learning provides an advantage of being able to scale with a dataset and can utilize transfer learning to extract features using other pre-trained models. This can decrease the training time needed and also take advantage of the huge dataset from other pre-trained models.

Besides, the representation generated from the deep learning model is proved to be useful. When other shallow learning uses the representation to predict QoE, it can be observed that the values obtained improved compared to using pre-processed data from the dataset.

The deep learning model achieved the best results when compared to other shallow learning algorithms. Lastly, it can be observed that the implementation of ensemble learning is able to enhance the overall accuracy of the deep learning model for QoE prediction. SROCC, LCC and the RMSE values are improved after adding an ensembling process with the Random Forest model.

## CHAPTER 5

### CONCLUSIONS AND RECOMMENDATIONS

#### 5.1 Conclusions

This project has managed to identify several problems in the conventional machine learning QoE prediction model based on the literature review done. These problems include the ML conventional algorithms, which tend to rely on hand-crafted features and need different feature extraction methods. Thus, this project successfully addresses the issues by achieving the aim and objectives as follows.

This project has implemented a deep learning QoE prediction model for MPEG-DASH video via PyTorch and Scikit-learn. The deep learning prediction model was trained with the LIVE-NFLX-II dataset. The hyper-parameter of the model is then analyzed to obtain the default parameters for the training process.

Second, to show that deep learning can perform representation learning well, the performance of conventional methods are compared when trained with the model representation and when trained with pre-processed data. The representation from the deep learning model shows to be useful as the conventional methods improved when the representation is used for training.

Next, the performance of the deep learning QoE prediction model is evaluated. The model obtained better results than the conventional ML algorithms, with a lower RMSE and higher SROCC and LCC. The proposed framework is also evaluated, where ensemble learning is performed instead of using only a linear layer function as the output layer. This implementation managed to further increase the performance of the deep learning model.

In summary, the proposed deep learning QoE prediction model that integrates ensemble learning is able to predict continuous QoE and perform better than conventional machine learning algorithms.

## 5.2 Recommendations for Future Work

This final year project also suggests the following areas that are worth to be explored in the future:

It is suggested to develop the source codes using CUDA to enable GPU usage during training. GPU enables multiple computations simultaneously as it allows parallel computing; hence are more optimized for training deep learning models. With the implementation of GPU usage for training and inference, it will have a faster processing speed and is more suitable for applications such as real-time QoE prediction.

Besides, techniques to improve deep learning such as stacked generalization ensemble can be integrated to further investigate the corresponding prediction performance of the model.

Lastly, it is suggested that the deep learning QoE model can be applied in a real-time DASH video streaming system to further evaluate the usefulness of the deep learning model in real-time QoE prediction, helping maximize the QoE for the end-users.

## REFERENCES

- Aung, W.T. and Hla, K.H.M.S., 2009. Random Forest Classifier for Multi-category Classification of Web Pages. *2009 IEEE Asia-Pacific Services Computing Conference (APSCC)*, pp.372–376.
- Ayodele, T.O., 2010. Types of Machine Learning Algorithms. *New Advances in Machine Learning*, pp.19–49.
- Bampis, C.G., Li, Z., Katsavounidis, I., Huang, T.Y., Ekanadham, C. and Bovik, A.C., 2018. Towards perceptually optimized end-to-end adaptive video streaming. *arXiv*, pp.1–16.
- Berrar, D., 2018. Cross-validation. *Encyclopedia of Bioinformatics and Computational Biology: ABC of Bioinformatics*, 1–3(January 2018), pp.542–545.
- Casas, P., D’Alconzo, A., Wamser, F., Seufert, M., Gardlo, B., Schwind, A., Tran-Gia, P. and Schatz, R., 2017. Predicting QoE in cellular networks using machine learning and in-smartphone measurements. *2017 9th International Conference on Quality of Multimedia Experience, QoMEX 2017*, (May).
- Cicco, L. De, Mascolo, S. and Palmisano, V., 2019. QoE-driven resource allocation for massive video distribution. *Ad Hoc Networks*, [online] 89, pp.170–176. Available at: <<https://doi.org/10.1016/j.adhoc.2019.02.008>>.
- Duc, T.N., Minh, C.T., Xuan, T.P. and Kamioka, E., 2020. Convolutional Neural Networks for Continuous QoE Prediction in Video Streaming Services. *IEEE Access*, 8, pp.116268–116278.
- Friedman, J.H., 2011. Greedy Function Approximation: A Gradient Boosting Machine. In: *Annals of statistics*. pp.1189–1232.



Guerra, L., McGarry, L.M., Robles, V., Bielza, C., Larrañaga, P. and Yuste, R., 2011. Comparison between supervised and unsupervised classifications of neuronal cell types: A case study. *Developmental Neurobiology*, 71(1), pp.71–82.

Hang, L., 2018. Deep Learning for natural language processing: advantages and challenges. *National Science Review*, 5(1), pp.22–24.

Hastie, T., Tibshirani, R. and Friedman, J., 2008. The Elements of Statistical Learning. In: *The Elements of Statistical Learning*, 2nd editio. Springer.p.592.

Hinton, G.E., 2012. A practical guide to training restricted boltzmann machines. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7700 LECTU, pp.599–619.

Hu, H., Liu, Z. and An, J., 2020. Mining Mobile Intelligence for Wireless Systems: A Deep Neural Network Approach. *IEEE Computational Intelligence Magazine*, 15(1), pp.24–31.

Huynh-Thu, Q. and Ghanbari, M., 2012. The accuracy of PSNR in predicting video quality for different video scenes and frame rates. *Telecommunication Systems*, 49(1), pp.35–48.

Islam, M.J., Wu, Q.M.J., Ahmadi, M. and Sid-Ahmed, M.A., 2008. Investigating the Performance of Naive- Bayes Classifiers and K- Nearest Neighbor Classifiers. *2007 International Conference on Convergence Information Technology (ICCIT 2007)*, pp.1541–1546.

Jeffrey, P., Socher, R. and Manning, C.D., 2014. GloVe: Global Vectors for Word Representation. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. pp.1532–1543.

Kelleher, J.D., Namee, B. Mac and D'Arcy, A., 2015. *Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, And Case Studies*. MIT Press, 2015.

Kordos, M., Blachnik, M. and Strzempa, D., 2010. Do we need whatever more than k-NN? *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6113 LNAI(PART 1), pp.414–421.

Kotevski, Z. and Mitrevski, P., 2010. Experimental Comparison of PSNR and SSIM Metrics for Video Quality Estimation. *ICT Innovations 2009*, pp.357–366.

Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y. and Alsaadi, F.E., 2017. A survey of deep neural network architectures and their applications. *Neurocomputing*, [online] 234(October 2016), pp.11–26. Available at: <<http://dx.doi.org/10.1016/j.neucom.2016.12.038>>.

Mocanu, D.C., Exarchakos, G., Ammar, H.B. and Liotta, A., 2015. Reduced reference image quality assessment via Boltzmann Machines. *Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management, IM 2015*, (2015), pp.1278–1281.

V. Murudkar, C. and D. Gitlin, R., 2019. Machine Learning for Qoe Prediction and Anomaly Detection in Self-Organizing Mobile Networking Systems. *International Journal of Wireless & Mobile Networks*, 11(2), pp.01–12.

Mushtaq, M.S., Augustin, B. and Mellouk, A., 2012. Empirical study based on machine learning approach to assess the QoS/QoE correlation. *2012 17th European Conference on Network and Optical Communications, NOC 2012, 7th Conference on Optical Cabling and Infrastructure, OC and I 2012*, (April 2012).

Myles, A.J., Feudale, R.N., Liu, Y., Woody, N.A. and Brown, S.D., 2004. An introduction to decision tree modeling. *Journal of Chemometrics*, 18(6), pp.275–285.

Rachmadi, M.F., Del C. Valdés-Hernández, M., Agan, M.L.F. and Komura, T., 2017. Deep learning vs. conventional machine learning: Pilot study of WMH segmentation in brain MRI with absence or mild vascular pathology. *Journal of Imaging*, 3(4), pp.1–19.

Safavian, S.R. and Landgrebe, D., 1991. A Survey of Decision Tree Classifier Methodology. *IEEE Transactions on Systems, Man and Cybernetics*, 21(3), pp.660–674.

Sathya, R. and Abraham, A., 2013. Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification. *International Journal of Advanced Research in Artificial Intelligence*, 2(2), pp.34–38.

Segal, M.R., 2004. *Machine learning benchmarks and random forest regression*. Center for Bioinformatics & Molecular Biostatistics, University of California, San Francisco.

Seufert, M., 2019. Fundamental advantages of considering quality of experience distributions over mean opinion scores. *2019 11th International Conference on Quality of Multimedia Experience, QoMEX 2019*, (2).

Sideris, A., Markakis, E., Zotos, N., Pallis, E. and Skianis, C., 2015. MPEG-DASH users' QoE: The segment duration effect. *2015 7th International Workshop on Quality of Multimedia Experience, QoMEX 2015*, pp.1–6.

Sodagar, I., 2011. The MPEG-dash standard for multimedia streaming over the internet. *IEEE Multimedia*, 18(4), pp.62–67.

Soundararajan, R. and Bovik, A.C., 2013. Video quality assessment by reduced reference spatio-temporal entropic differencing. *IEEE Transactions on Circuits and Systems for Video Technology*, 23(4), pp.684–694.

Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C. and Liu, C., 2018. A survey on deep transfer learning. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11141 LNCS, pp.270–279.

Tao, X., Duan, Y., Xu, M., Meng, Z. and Lu, J., 2019. Learning QoE of Mobile Video Transmission with Deep Neural Network: A Data-Driven Approach. *IEEE Journal on Selected Areas in Communications*, 37(6), pp.1337–1348.

Tran, D., Bourdev, L., Fergus, R., Torresani, L. and Paluri, M., 2015. Learning Spatiotemporal Features with 3D Convolutional Networks. *ICCV*.

Vasilev, V., Leguay, J., Paris, S., Maggi, L. and Debbah, M., 2018. Predicting QoE Factors with Machine Learning. *IEEE International Conference on Communications*, 2018-May, pp.0–5.

Yu, X., Chen, H., Zhao, W. and Xie, L., 2012. No-reference QoE prediction model for video streaming service in 3G networks. *2012 International Conference on Wireless Communications, Networking and Mobile Computing, WiCOM 2012*, pp.0–3.

Yue, T., Wang, H., Cheng, S. and Shao, J., 2020. Deep learning based QoE evaluation for internet video. *Neurocomputing*, 386, pp.179–190.

Zhang, H., Dong, L., Gao, G., Hu, H., Wen, Y. and Guan, K., 2020. DeepQoE: A Multimodal Learning Framework for Video Quality of Experience (QoE) Prediction. *IEEE Transactions on Multimedia*, 9210(c), pp.1–1.

Zhang, L., Tan, J., Han, D. and Zhu, H., 2017. From machine learning to deep learning: progress in machine intelligence for rational drug discovery. *Drug Discovery Today*, [online] 22(11), pp.1680–1685. Available at: <<https://doi.org/10.1016/j.drudis.2017.08.010>>.

Zhao, S., Li, Z., Medhi, D., Lai, P. and Liu, S., 2017. Study of user QoE improvement for dynamic adaptive streaming over HTTP (MPEG-DASH). *2017 International Conference on Computing, Networking and Communications, ICNC 2017*, pp.566–570.

## APPENDICES

### Appendix A: LIVE-NFLX-II Dataset and DeepQoE Download Source

- LIVE-NFLX-II  
<https://utexas.app.box.com/v/LIVE-NFLX-Plus/>
- DeepQoE Github Source Code  
<https://github.com/cap-ntu/DeepQoE>

### Appendix B: Shell Script ‘feature\_extraction.sh’ (Extract features from frames for LIVE-NFLX-II)

---

```
if [ ! -f c3d_resnet18_sports1m_r2_iter_2800000.caffemodel ];then
  wget https://www.dropbox.com/s/qqfrg6h44d4jb46/c3d_resnet18_sports1m_r2_iter_2800000.caffemodel
fi

GLOG_logtostderr=1 ../../build/tools/extract_image_features.bin c3d_resnet18_ucf101_feature_extraction.prototxt
c3d_resnet18_sports1m_r2_iter_2800000.caffemodel -i 10 1863 ucf101_video_frame.prefix prob pool5 fc8
```

## Appendix C: Python Code 1 (PKL-to-CSV converter for LIVE-NFLX-II dataset)

```

import glob
import os
import csv
import pickle
import numpy as np

def read_binary_blob (file_name):

    fid = open(file_name, 'r')

    #s contains size of the blob e.g. num x chanel x length x height x width
    s = np.fromfile(fid, np.int32, 5)

    m = s[0] * s[1] * s[2] * s[3] * s[4]

    # data is the blob binary data in single precistion (e.g float in C++)
    data = np.fromfile(fid, np.float32, m)
    fid.close()

    return (s, data)

#Go into the DATA directory
currentpath = os.getcwd()
os.chdir(currentpath + '/Pkl_Files')

#get all file names with .pkl format
all_pkl_file_list = glob.glob("*.pkl")

#Path to get pool_5 paths
pool5_path = "/home/alleria/Desktop/C3D-master/C3D-master/C3D-v1.1/data/videos/"

#back to current path to prepare writing a csv file
os.chdir(currentpath)

with open('data_extracted_continuous.csv', mode='w') as csv_file:
    fieldnames = ['Algorithm', 'content_name', 'frame_number', 'frame_rate', 'is_rebuffered_bool', 'scene_cut_count',
'scene_cut_detected_count', 'playout_bitrate', 'MSSIM', 'VMAF', 'STRRED', 'SSIM', 'PSNR', 'continuous_zscored_mos', 'C3D']

    writer = csv.DictWriter(csv_file, fieldnames=fieldnames)

    writer.writeheader()

    for eachfile in all_pkl_file_list:
        frame_number = 1

        with open(eachfile, 'rb') as f:
            data = pickle.load(f)

            #Get essential data
            video_name = data['distorted_np4_video'][:-4]
            total_frame = int(data['N_total_frames'])
            MSSIM_list = []
            VMAF_list = []
            STRRED_list = []
            PSNR_list = []
            SSIM_list = []

            metric_count = 0
            for test in data['is_rebuffered_bool']:
                if test == 0:
                    MSSIM_list.append(data['MSSIM'][metric_count])
                    VMAF_list.append(data['VMAF'][metric_count])
                    STRRED_list.append(data['STRRED'][metric_count])
                    PSNR_list.append(data['PSNR'][metric_count])
                    SSIM_list.append(data['SSIM'][metric_count])
                    metric_count += 1
                else:
                    MSSIM_list.append('0')
                    VMAF_list.append('0')
                    STRRED_list.append('0')
                    PSNR_list.append('0')
                    SSIM_list.append('0')

            last_playout_bitrate = data['playout_bitrate'][0]
            segment_count = 0
            for test in data['playout_bitrate']:
                if test == 0:
                    encoding_height_list.append('0')
                    encoding_width_list.append('0')
                    encoding_QP_list.append('0')
                    last_playout_bitrate = test
                elif test == last_playout_bitrate:
                    encoding_height_list.append(data['per_segment_encoding_height'][segment_count])
                    encoding_width_list.append(data['per_segment_encoding_width'][segment_count])
                    encoding_QP_list.append(data['per_segment_encoding_QP'][segment_count])
                    last_playout_bitrate = test

```

## Appendix C (continued)

```

else:
    segment_count += 1
    encoding_height_list.append(data['per_segment_encoding_height'][segment_count])
    encoding_width_list.append(data['per_segment_encoding_width'][segment_count])
    encoding_QP_list.append(data['per_segment_encoding_QP'][segment_count])
    last_playout_bitrate = test

#print(len(MSSIM_list),total_frame)

loop_count = (total_frame-8)//8 # (X-16)/8 - 1 because 16 frames, but 8 frames overlapping.

for i in range(0,loop_count):
    is_rebuffered_bool = 0
    scene_cut_count = 0
    scene_cut_detected_count = 0
    playout_bitrate = 0
    MSSIM = 0
    VMAF = 0
    STRRED = 0
    PSNR = 0
    SSIM = 0
    continuous_zscored_mos = 0

    for j in range(frame_number-1,frame_number+15):

        for test in data['scene_cuts']:
            if int(test) == int(j):
                scene_cut_count += 1

        for test in data['scene_cuts_detected']:
            if int(test) == int(j):
                scene_cut_detected_count += 1

        is_rebuffered_bool += int(data['is_rebuffered_bool'][j])
        playout_bitrate += float(data['playout_bitrate'][j])
        MSSIM += float(MSSIM_list[j])
        VMAF += float(VMAF_list[j])
        STRRED += float(STRRED_list[j])
        PSNR += float(PSNR_list[j])
        SSIM += float(SSIM_list[j])
        continuous_zscored_mos += float(data['continuous_zscored_mos'][j])

    playout_bitrate = playout_bitrate/16
    MSSIM = MSSIM/16
    VMAF = VMAF/16
    STRRED = STRRED/16
    PSNR = PSNR/16
    SSIM = SSIM/16
    continuous_zscored_mos = continuous_zscored_mos/16

    #frame_rate_rounded = round(data['frame_rate'])
    #print(data['frame_rate'], frame_rate_rounded)

    pool5_location = pool5_path + video_name + '/' + "{:06d}".format(frame_number) + ".pool5"
    _c3d_vector = read_binary_blob(pool5_location)

    writer.writerow({'Algorithm': data['adaptation_algorithm'],
'content_name': data['content_name'],
'frame_number': frame_number,
'frame_rate': data['frame_rate'],
'is_rebuffered_bool': is_rebuffered_bool,
'scene_cut_count': scene_cut_count,
'scene_cut_detected_count': scene_cut_detected_count,
'playout_bitrate': playout_bitrate,
'MSSIM': MSSIM,
'VMAF': VMAF,
'STRRED': STRRED,
'PSNR': PSNR,
'SSIM': SSIM,
'continuous_zscored_mos': continuous_zscored_mos,
'C3D': c3d_vector,
})

    frame_number += 8

```



## Appendix C (continued)

```

if not total_frame % 8 == 0:
    frame_number = total_frame - 15
    is_rebuffered_bool = 0
    scene_cut_count = 0
    scene_cut_detected_count = 0
    playout_bitrate = 0
    MSSIM = 0
    VMAF = 0
    STRRED = 0
    PSNR = 0
    SSIM = 0
    continuous_zscored_mos = 0

for j in range(frame_number-1, frame_number+15):

    for test in data['scene_cuts']:
        if int(test) == int(j):
            scene_cut_count += 1

    for test in data['scene_cuts_detected']:
        if int(test) == int(j):
            scene_cut_detected_count += 1

    is_rebuffered_bool += int(data['is_rebuffered_bool'][j])
    playout_bitrate += float(data['playout_bitrate'][j])
    MSSIM += float(MSSIM_list[j])
    VMAF += float(VMAF_list[j])
    STRRED += float(STRRED_list[j])
    PSNR += float(PSNR_list[j])
    SSIM += float(SSIM_list[j])
    continuous_zscored_mos += float(data['continuous_zscored_mos'][j])

    playout_bitrate = playout_bitrate/16
    MSSIM = MSSIM/16
    VMAF = VMAF/16
    STRRED = STRRED/16
    PSNR = PSNR/16
    SSIM = SSIM/16
    continuous_zscored_mos = continuous_zscored_mos/16

    pool5_location = pool5_path + video_name + '/' + "{:06d}".format(frame_number) + ".pool5"
    _c3d_vector = read_binary_blob(pool5_location)

    writer.writerow({'Algorithm': data['adaptation_algorithm'],
                    'content_name': data['content_name'],
                    'frame_number': frame_number,
                    'frame_rate': data['frame_rate'],
                    'is_rebuffered_bool': is_rebuffered_bool,
                    'scene_cut_count': scene_cut_count,
                    'scene_cut_detected_count': scene_cut_detected_count,
                    'playout_bitrate': playout_bitrate,
                    'MSSIM': MSSIM,
                    'VMAF': VMAF,
                    'STRRED': STRRED,
                    'PSNR': PSNR,
                    'SSIM': SSIM,
                    'continuous_zscored_mos': continuous_zscored_mos,
                    'C3D': c3d_vector,
                    })

```

## Appendix D: Python Code 2 (Deep Learning Training Code)

```

from __future__ import print_function

import os
import sys
import numpy as np
import pandas as pd
import pickle
import logging
import torch.optim as optim
import torch.utils.data as data_utils
from scipy.stats import spearmanr as sr
from scipy.stats import pearsonr as pr
from DeepQoE.nets import *
from DeepQoE.config import cfg, parse_arguments
from scripts.generate_pretrained_models import generate_streaming_data_continuous, show_results
from DeepQoE.data_loader import QoENFLX2DatasetContinuous

def main(args):
    #STEP 1: DEFINE MODEL (SEMI-DONE) AND OPTIMIZER
    model = TestingNNContinuous()
    if args.use_gpu and torch.cuda.is_available():
        torch.cuda.set_device(args.gpu_id)
        model.cuda()
    #print(model)

    optimizer = optim.Adadelta(model.parameters(), lr=0.1, rho=0.95, eps=1e-08, weight_decay=1e-6)

    #STEP 2: PRODUCE OUT ALL THE VECTORS NEEDED, [?,?,?,?]
    x_train, y_train, x_test, y_test, feature_labels, X_full, y_full = generate_streaming_data_continuous()
    #print(y_test)

    #STEP 3: CONVERT FROM THE NUMPY VECTOR TO TENSOR, THEN PASS TO DATALOADER
    train_data = QoENFLX2DatasetContinuous(x_train, y_train)
    train_loader = data_utils.DataLoader(train_data, batch_size=64, shuffle=False)

    test_data = QoENFLX2DatasetContinuous(x_test, y_test)
    test_loader = data_utils.DataLoader(test_data, batch_size=4031, shuffle=False)

    #STEP 4: TRAIN AND TEST
    for epoch in range(args.epochs):
        train(train_loader, model, optimizer, epoch)

    output = test(test_loader, model)
    print(output.shape)

    #STEP 5: CALCULATE LOSS
    predict = output.cpu().data.numpy()
    #print(predict.shape)

    currentpath = os.getcwd()
    torch.save(model, 'model.pt')

def train(train_loader, model, optimizer, epoch):
    model.train()
    for batch_idx, sample_batched in enumerate(train_loader):
        pid = os.getpid()
        #print(batch_idx)

        x_1 = torch.autograd.Variable(sample_batched['algorithm']).cuda()
        x_2 = torch.autograd.Variable(sample_batched['frame_number'])
        x_3 = torch.autograd.Variable(sample_batched['frame_rate'])
        x_4 = torch.autograd.Variable(sample_batched['is_rebuffer_bool'])
        x_5 = torch.autograd.Variable(sample_batched['scene_cuts_count'])
        x_6 = torch.autograd.Variable(sample_batched['scene_cuts_detected_count'])
        x_7 = torch.autograd.Variable(sample_batched['playout_bitrate'])
        x_8 = torch.autograd.Variable(sample_batched['MSSIM'])
        x_9 = torch.autograd.Variable(sample_batched['VMAF'])
        x_10 = torch.autograd.Variable(sample_batched['STRRED'])
        x_11 = torch.autograd.Variable(sample_batched['SSIM'])
        x_12 = torch.autograd.Variable(sample_batched['PSNR'])
        x_13 = torch.autograd.Variable(sample_batched['C3D'])
        target = torch.autograd.Variable(sample_batched['zmos'])
        x_13 = x_13.reshape((x_13.shape[0], 512))

        optimizer.zero_grad()
        prediction, _ = model(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_10, x_11, x_12, x_13)
        loss = F.mse_loss(prediction, target.float())

        loss.backward()
        optimizer.step()

        if batch_idx % 566 == 0:
            print('{}\tTrain Epoch: {} \tLoss: {:.6f}'.format(
                pid, epoch, loss.item()))

```

## Appendix D (continued)

```

def test(test_loader, model):
    model.eval()
    test_loss = 0
    for sample_batched in test_loader:
        x_1 = torch.autograd.Variable(sample_batched['algorithm']).cuda()
        x_2 = torch.autograd.Variable(sample_batched['frame_number'])
        x_3 = torch.autograd.Variable(sample_batched['frame_rate'])
        x_4 = torch.autograd.Variable(sample_batched['is_rebuffer_bool'])
        x_5 = torch.autograd.Variable(sample_batched['scene_cuts_count'])
        x_6 = torch.autograd.Variable(sample_batched['scene_cuts_detected_count'])
        x_7 = torch.autograd.Variable(sample_batched['playout_bitrate'])
        x_8 = torch.autograd.Variable(sample_batched['MSSIM'])
        x_9 = torch.autograd.Variable(sample_batched['VMAF'])
        x_10 = torch.autograd.Variable(sample_batched['STRRED'])
        x_11 = torch.autograd.Variable(sample_batched['SSIM'])
        x_12 = torch.autograd.Variable(sample_batched['PSNR'])
        x_13 = torch.autograd.Variable(sample_batched['C3D'])
        target = torch.autograd.Variable(sample_batched['zmos'])
        x_13 = x_13.reshape((x_13.shape[0], 512))

        output, _ = model(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_10, x_11, x_12, x_13)
        test_loss += F.mse_loss(output, target.float(), size_average=False).item()
        # print (output)
        # print(target.float())

    test_loss /= len(test_loader.dataset)
    print('\nTest set: Average loss: {:.4f}'.format(test_loss))
    return output

if __name__ == '__main__':
    main(parse_arguments(sys.argv[1:]))

```

---

## Appendix E: Python Code 3 (Evaluation Code for section 4.4)

```

from DeepQoE.nets import *
import torch.optim as optim
import os
import sys
import numpy as np
import pandas as pd
import pickle
import logging
import torch.optim as optim
import torch.utils.data as data_utils
from scipy.stats import spearmanr as sr
from scipy.stats import pearsonr as pr
from DeepQoE.data_loader import QoENFLX2DatasetContinuous
import torch.utils.data as data_utils
from scripts.generate_pretrained_models import generate_streaming_data_continuous, show_results, show_results_noRF
from torch.autograd import Variable
from sklearn import datasets, ensemble
from sklearn.inspection import permutation_importance
from sklearn import metrics
from sklearn.neural_network import MLPRegressor
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.ensemble import RandomForestRegressor
from sklearn import svm

def test(test_loader, model):
    model.eval()
    extracted = []
    test_loss = 0
    for sample_batched in test_loader:
        x_1 = torch.autograd.Variable(sample_batched['algorithm']).cuda()
        x_2 = torch.autograd.Variable(sample_batched['frame_number'])
        x_3 = torch.autograd.Variable(sample_batched['frame_rate'])
        x_4 = torch.autograd.Variable(sample_batched['ts_rebuffer_bool'])
        x_5 = torch.autograd.Variable(sample_batched['scene_cuts_count'])
        x_6 = torch.autograd.Variable(sample_batched['scene_cuts_detected_count'])
        x_7 = torch.autograd.Variable(sample_batched['playout_bitrate'])
        x_8 = torch.autograd.Variable(sample_batched['MSSIM'])
        x_9 = torch.autograd.Variable(sample_batched['VMAF'])
        x_10 = torch.autograd.Variable(sample_batched['STRRED'])
        x_11 = torch.autograd.Variable(sample_batched['SSIM'])
        x_12 = torch.autograd.Variable(sample_batched['PSNR'])
        x_13 = torch.autograd.Variable(sample_batched['C3D'])
        target = torch.autograd.Variable(sample_batched['zmos'])
        x_13 = x_13.reshape((x_13.shape[0], 512))

        output, _ = model(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_10, x_11, x_12, x_13)
        _ = Variable(_, requires_grad=True)
        _ = _.detach().numpy()
        extracted.append(_)
        test_loss += F.mse_loss(output, target.float(), size_average=False).item()
        # print(output)
        # print(target.float())

    test_loss /= len(test_loader.dataset)
    print('\nTest set: Average loss: {:.4f}'.format(test_loss))
    return output, extracted

PATH = 'model.pt'

model = torch.load(PATH)
extracted_features = []
model.eval()

#print(model)

optimizer = optim.Adam(model.parameters(), lr=1.0, rho=0.95, eps=1e-08, weight_decay=1e-6)

#STEP 2: PRODUCE OUT ALL THE VECTORS NEEDED, [?, ?, ?, ?, ?]
x_train, y_train, x_test, y_test, feature_labels, X_full, y_full = generate_streaming_data_continuous()
#print(y_test)

#STEP 3: CONVERT FROM THE NUMPY VECTOR TO TENSOR, THEN PASS TO DATALOADER
train_data = QoENFLX2DatasetContinuous(x_train, y_train)
train_loader = data_utils.DataLoader(train_data, batch_size=36274, shuffle=False)
# for x in train_loader:
#     print(x)

test_data = QoENFLX2DatasetContinuous(x_test, y_test)
test_loader = data_utils.DataLoader(test_data, batch_size=4031, shuffle=False)

#STEP 4: TRAIN AND TEST
# for epoch in range(args.epochs):
#     train(train_loader, model, optimizer, epoch)

output, extracted_features = test(train_loader, model)
extracted_features = extracted_features[0]

```

## Appendix E (continued)

```

output, test_extracted_features = test(test_loader, model)
test_extracted_features = test_extracted_features[0]
#print(test_extracted_features.shape)

y_train_RF = y_train.reshape((36274,))
y_test_RF = y_test.reshape((4031,))

regressor = MLPRegressor(hidden_layer_sizes=())
regressor.fit(extracted_features, y_train_RF)
y_pred_MLP = regressor.predict(test_extracted_features)
y_pred_MLP = y_pred_MLP.reshape((4031,1))

RFRegressor = RandomForestRegressor(n_estimators=100, random_state=0, max_features=175)
RFRegressor.fit(x_train, y_train_RF)
y_pred_RF = RFRegressor.predict(x_test)
y_pred_RF = y_pred_RF.reshape((4031,1))

linear_representation = LinearRegression()
linear_representation.fit(x_train, y_train_RF)
y_pred_LinReg = linear_representation.predict(x_test)
y_pred_LinReg = y_pred_LinReg.reshape((4031,1))

print('\nFOR RF:')
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred_RF))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred_RF))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred_RF)))
print("SROCC using DEEPQOE + RF: " + str(sr(y_test, y_pred_RF.reshape(-1, 1))[0]))
print("LCC using DEEPQOE + RF: " + str(pr(y_test.flatten(), y_pred_RF.flatten())[0]))

print('\nFOR LinReg:')
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred_LinReg))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred_LinReg))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred_LinReg)))
print("SROCC using DEEPQOE + RF: " + str(sr(y_test, y_pred_LinReg.reshape(-1, 1))[0]))
print("LCC using DEEPQOE + RF: " + str(pr(y_test.flatten(), y_pred_LinReg.flatten())[0]))

print('\n\nFOR Deep Neural MLP:')
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred_MLP))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred_MLP))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred_MLP)))
print("SROCC using MLP: " + str(sr(y_test, y_pred_MLP.reshape(-1, 1))[0]))
print("LCC using MLP: " + str(pr(y_test.flatten(), y_pred_MLP.flatten())[0]))

params = {'n_estimators': 500,
          'max_depth': 4,
          'min_samples_split': 5,
          'learning_rate': 0.01,
          'loss': 'ls'}
regressor2 = ensemble.GradientBoostingRegressor(**params)
regressor2.fit(x_train, y_train_RF)
y_pred2 = regressor2.predict(x_test)
y_pred2 = y_pred2.reshape((4031,1))
print('\nFOR Gradient Boosting:')
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred2))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred2))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred2)))
print("SROCC using DEEPQOE + RF: " + str(sr(y_test, y_pred2.reshape(-1, 1))[0]))
print("LCC using DEEPQOE + RF: " + str(pr(y_test.flatten(), y_pred2.flatten())[0]))

regressor2 = svm.SVR()
regressor2.fit(x_train, y_train_RF)
y_pred2 = regressor2.predict(x_test)
y_pred2 = y_pred2.reshape((4031,1))
print('\nFOR SVM:')
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred2))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred2))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred2)))
print("SROCC using DEEPQOE + RF: " + str(sr(y_test, y_pred2.reshape(-1, 1))[0]))
print("LCC using DEEPQOE + RF: " + str(pr(y_test.flatten(), y_pred2.flatten())[0]))

```