The Hong Kong Polytechnic University
香港理工大學

Department of Electronic and Information Engineering

Final Year Project Final Report

(2019/20)

Machine Learning-based Video Quality Assessment

Student Name:	Ho Naomi Joyce
Student ID:	
Programme Code:	42470
Supervisor(s):	Dr Yui-Lam Chan
Submission Date:	May 12th, 2020

The Hong Kong Polytechnic University
Department of Electronic and Information Engineering

EIE4430 / EIE4433 Honours Project

1. Student Name: ___**Ho Naomi Joyce**____________ (Student No.:)

2. Programme Code: **42470**

3. Project Title: **Machine Learning-based Video Quality Assessment**

4. Supervisor Name: **Dr Yui-Lam Chan**

5. Project summary (State clearly the project objectives and results achieved by yourself.)
[Please list in point form where appropriate]

- Project Objectives and Results
  - Understood the problems raised in the project
  - Cooperated and communicated effectively with my supervisor and tutor regularly for the interest of project management
  - Applied the professional engineering knowledge and skills learned onto the project to create solutions for different constraints
  - Learned to employ new tools and equipment to gather information for the project
  - Worked individually by self-discipline to ensure the smooth completion of the project

DECLARATION OF ORIGINALITY

Except where reference is made in the text of this report, I declare that this report contains no material published elsewhere or extracted in whole or in part from any works or assignments presented by me or any other parties for another subject. In addition, it has not been submitted for the award of any other degree or diploma in any other tertiary institution.

No other person’s work has been used without due acknowledgement in the main text of the Report.

I fully understand that any discrepancy from the above statements will constitute a case of plagiarism and be subject to the associated.

____

Signature

Contents

1. Introduction

In the past, when the internet had not been well developed yet, television was the sole source for broadcasting videos. Nonetheless, times have changed. Communication technology becomes more and more advanced. Today's audience does not only rely on the traditional medium to enjoy videos. Instead, there is an increasing tendency of viewing them on online streaming platforms such as Netflix and YouTube. The rising demand for video streaming services leads to the competitive market of those websites. For this reason, service providers have come to an awareness of how viewers perceive videos, which is also known as Quality of Experience (QoE). In order to control and optimise the standard of their services, Video Quality Assessments (VQA) are conducted to study the influences of video distortion toward QoE.

The means of VQA can be primarily categorised in two: subjective and objective VQA. Subjective VQA is defined as a test which "involves human participants providing scores of perceived visual quality after watching test sequences in a controlled environment" [1]. It is the most reliable way to evaluate QoE as human beings are those who receive videos at the end [2]. The brief process of executing this assessment has been described by Arun Kumar and Chandramathi [3]. A group of specialists were requested to watch video clips. The judges determined the quality of the films according to their perception by scoring each of them a Mean Opinion Score (MOS).

The implementation of subjective VQA, however, is "general complex, expensive, and time consuming" [4]. The process of computing MOS requires human resources and a significant amount of time to train the audience and determine the quality of the videos. Hence, it is not practical for real-time video processing and applications.

Due to this rationale, objective VQA is conducted instead to resolve the disadvantage of subjective VQA.

In this project, a machine learning-based no-reference objective VQA (NR-VQA) will be conducted to assess the quality of videos. Shahid et al. [5] explain that NR approach "does not require access to the original image/video" to judge the features and artifacts in the distorted videos. It is a pragmatic way to be implemented when resources are limited, thus is favourable in running a real-time objective VQA. To conduct that, a machine learning-based NR-VQA model will be established to measure MOS of each video in the aspect of human perception. Its structure follows that of a long short-term memory (LSTM) of recurrent neural network (RNN). Temporal features of the videos are going to be collected and applied into the model, together with the given spatial and temporal features [6].

Upon the completion of this project, the outcomes of the trained NR-VQA model will be compared with the MOS collected from a subjective VQA. Principal component analysis (PCA) will be applied in the model to improve the performance. The capability of the resourceless NR-VQA will also be juxtaposed with the achievements of existing full-reference VQA (FR-VQA), where the original videos are available in testing.

2. Project Objectives

1. Construct a machine learning LSTM model for NR-VQA to reflect the QoE perceived by human when viewing videos

2. Extract and evaluate the temporal features of videos to detect the temporal artifacts in them

3. Apply the extracted temporal and the provided spatial and temporal features to train the model and simulate the execution of a subjective video quality assessment to predict the MOS of each video

4. Test the model and compare the MOS obtained from it to those from the subjective VQA to observe the accuracy

5. Perform PCA feature selection to enhance the model performance

6. Compare the performance of the proposed NR-VQA with those of FR-VQA

3. Required Hardware and Software

Hardware

Name	Requirements
PC	MS Window 10, with a Minimum of 256GB Storage and 12GB RAM

Software

Name	Function(s)
Python	Programming Language
Anaconda Navigator	Distribution of Python Programming, Package and Environment Manager

4. Methodology

4.1. Database

The sources which will be used in this project are from the database MCL-V prepared by Lin et al. [7] for their original study of subjective VQA. They will be input to the model for testing and training. The database provides 12 uncompressed high definition (HD) source videos (Fig. 1). Each had been processed and distorted into two distortion types: H.264/AVC compression and the compression followed by scaling [7]. For each distortion type, the videos were distorted with four distortion levels. Generally, there are a total of 12 * 2 * 4 = 96 distorted videos in the database as the primary sources to undergo the VQA.

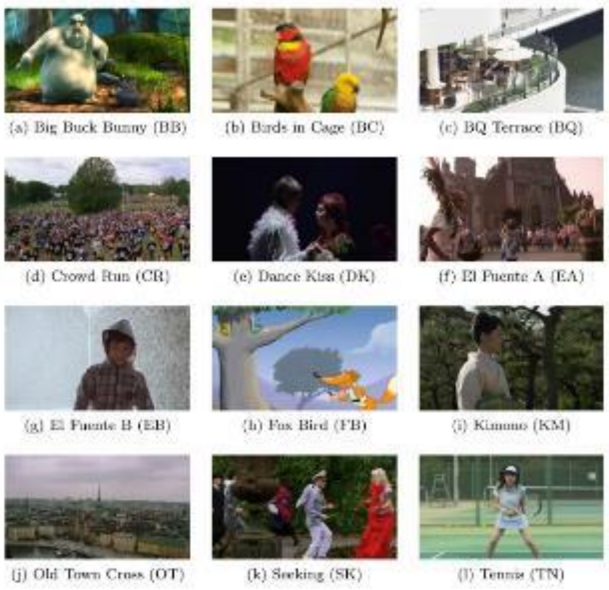
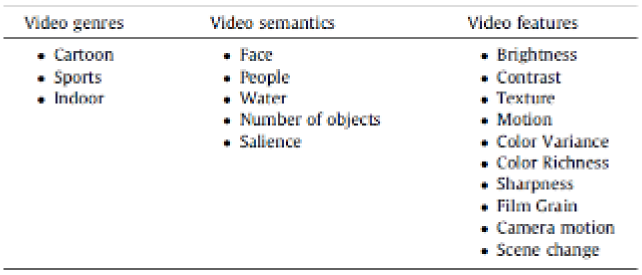


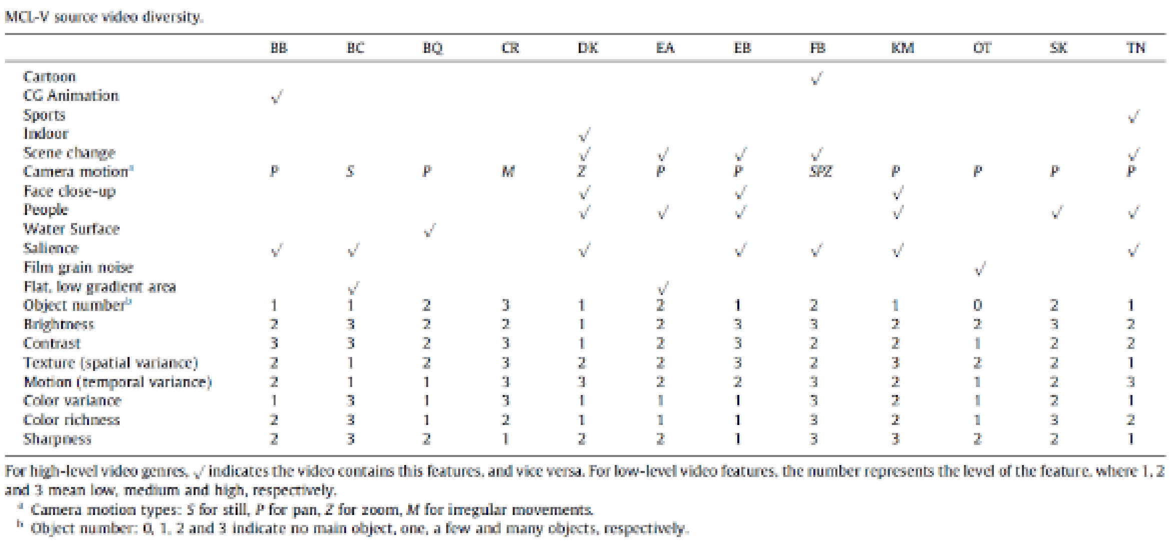
Fig. 1: The 12 source videos [7]

Moreover, to ensure the diversity of the sources in the database, the source videos were classified into three groups according to their characteristics: high-level video genres, mid-level video semantics and low-level video features [7]. This helps to study the influence of different traits in the videos towards MOS. The description of the groups and the categorisation of each video are shown in Fig. 2 and Fig. 3 respectively.



Video genres	Video semantics	Video features
• Cartoon	• Face	• Brightness
• Sports	• People	• Contrast
• Indoor	• Water	• Texture
	• Number of objects	• Motion
	• Salience	• Color Variance
		• Color Richness
		• Sharpness
		• Film Grain
		• Camera motion
		• Scene change

Fig. 2: Description of the groups regarding the characteristics of the videos [7]



MCL-V source video diversity.

	BB	BC	BQ	CR	DK	EA	EB	FB	KM	OT	SK	TN
Cartoon								√				
CG Animation	√											
Sports												√
Indoor					√							
Scene change					√	√	√	√				√
Camera motion[a]	*P*	*S*	*P*	*M*	*Z*	*P*	*P*	*SPZ*	*P*	*P*	*P*	*P*
Face close-up					√		√		√			
People					√	√	√		√		√	√
Water Surface			√									
Salience	√	√			√		√	√	√			√
Film grain noise										√		
Flat, low gradient area		√				√						
Object number[b]	1	1	2	3	1	2	1	2	1	0	2	1
Brightness	2	3	2	2	1	2	3	3	2	2	3	2
Contrast	3	3	2	3	1	2	3	2	2	1	2	2
Texture (spatial variance)	2	1	2	3	2	2	3	2	3	2	2	1
Motion (temporal variance)	2	1	1	3	3	2	2	3	2	1	2	3
Color variance	1	3	1	3	1	1	1	3	2	1	2	1
Color richness	2	3	1	2	1	1	1	3	2	1	3	2
Sharpness	2	3	2	1	2	2	1	3	3	2	2	1

For high-level video genres, √ indicates the video contains this features, and vice versa. For low-level video features, the number represents the level of the feature, where 1, 2 and 3 mean low, medium and high, respectively.
[a] Camera motion types: *S* for still, *P* for pan, *Z* for zoom, *M* for irregular movements.
[b] Object number: 0, 1, 2 and 3 indicate no main object, one, a few and many objects, respectively.

Fig. 3: Categorisation of each video in each group [7]

In the experiment done by Lin et al. [7], the MOS regarding all distorted videos received from the subjective VQA were given. They will be compared to the MOS obtained from the model for testing and validation.

4.2. Temporal Artifacts

This project will extract and compute the temporal artifacts of each frame. The work of Shahid et al. [5] suggests the typical steps of acting the assessment as in Fig. 4. Suppose the frame freeze is the target temporal impairment to be detected. To begin with, the interframe difference of the pixel intensity of video frames is calculated. Then, by applying specific techniques, the location and possibility of frame freeze or drops will be surveyed. A threshold is set to assist the detection of the occurrence of possible impairments. Finally, an overall value will be computed to reflect the impacts of the temporal artifacts in the video.

In this project, the procedures to obtain the temporal information of a frame is similar to the steps shown in Fig. 4. Instead, the targets are jerkiness, flickering and mosquito noise. These three temporal artifacts are going to be collected from the videos to undergo the VQA.

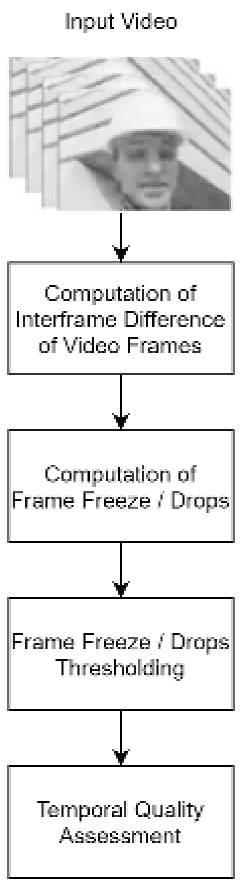

Input Video
Computation of Interframe Difference of Video Frames
Computation of Frame Freeze / Drops
Frame Freeze / Drops Thresholding
Temporal Quality Assessment

Fig. 4: Basic steps of conducting a temporal quality assessment [5]

4.2.1. <u>Jerkiness</u>

Jerkiness affects the quality of a video by showing uneven or unsteady motions to the audience [8]. It is caused by the temporal down-sampling of the video [5]. Borer [9] explains that due to the limited bandwidth of the channel for transmitting the information of a video, the frame rate of the media is reduced to save on the bandwidth. As such, the transmission will be delayed because of network congestion or video data loss. Some frames will be dropped, repeated or have their display time extended, leading to a sudden change in the frame rate [10]. Eventually, viewers will experience jerkiness while watching the video.

To quantify jerkiness, the motion intensity and the display time of each frame in a video have to be calculated.

Equation (1) suggested by Borer [9] is employed to measure the motion intensity.

$$m_{i+1}(v) = \sqrt{\sum_x (f_{i+1}(x) - f_i(x))^2}, \tag{1}$$

As mentioned, spectators become aware of jerkiness owing to the unsmooth movement of the objects in the video as a result of the change in the frame rate. Therefore, the motion intensity $m_{i+1}(v)$ should be calculated by computing the root mean square of the difference of luminance (Y-component) of consecutive frames. $f_i(x)$ is the value of luminance of frame i at position x. It can be extracted from the video YUV file in a matrix of the dimension of the video. The luminance values of the next frame $i+1$ will be subtracted by those of the previous frame i at the same position. The discrepancies will then be squared and summed up. Finally, the square root of the summation is the motion intensity of frame i.

The overall jerkiness of a video frame can be estimated based on the model in (2) proposed by Iacovoni et al. [10].

$$J_n = K \cdot MA_n^l \cdot (t_n - t_{n-1}) \tag{2}$$

In the study of Iacovoni et al. [10], jerkiness J_n is the product of the motion intensity MA^l_n and the gap between the transcoding time of the current frame n and the last frame $n-1$, which is t_n –

t_{n-1}. MA^l_n is equivalent to the motion intensity $m_{i+1}(v)$ in (1). It is noted that K is a constant to scale the product for a convenient graph plotting, and the transcoding time is the time for converting a media file from one format to another [11]. However, this not applicable in this project for the videos to be used are already processed by Lin et al. [7]. In contrast, the display time of each frame will be adopted. Jerkiness becomes noticeable when the frame rate alters suddenly, so do the motion of the objects in the video. Consequently, the motion intensity drops to 0 and the objects move slower when the display time of the frame is lengthened. Likewise, the motion intensity jumps up, and the objects twitch when the display time resumes to the original duration. This can be observed in Fig. 5, in which Borer [9] visualises how jerkiness appears while there is a dramatic variation in the motion intensity $m_{i+1}(v)$ over the display time Δt_i of frame i. Thereby, $t_n - t_{n-1}$ will be replaced by the display time of frame n.

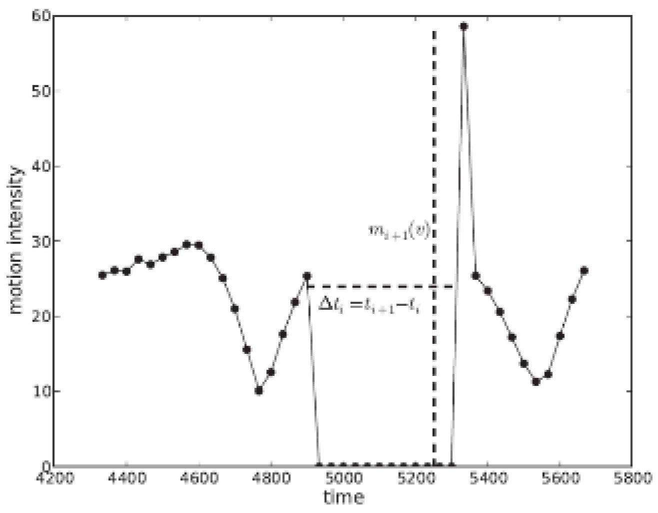


Fig. 5: Change in the motion intensity over the display time of the frame [9]

In this project, the value of jerkiness J_n of each frame in (2) will be the input of the LSTM model, where K is the scaling constant and MA^l_n is the motion intensity. t_n is the timestamp time of frame n, so $t_n - t_{n-1}$ is the display time of frame n. Since MA^l_n is equivalent to $m_{i+1}(v)$, so it is computed by (1).

4.2.2. <u>Flickering</u>

Flickering is the alteration of luminance between consecutive frames over time [8]. It usually happens in old black and white films where disruption of luminance from frame to frame arises when the frame rate is too low with coding artifacts [5], [12].

To illustrate that, Fig. 6 shows the flickering artifact in the frame sequence between frame 3843 to 3850 of an old film [13]. The frame rate of the video is modified from 29.97 fps to 12 fps for a clear recognition of the impairment.

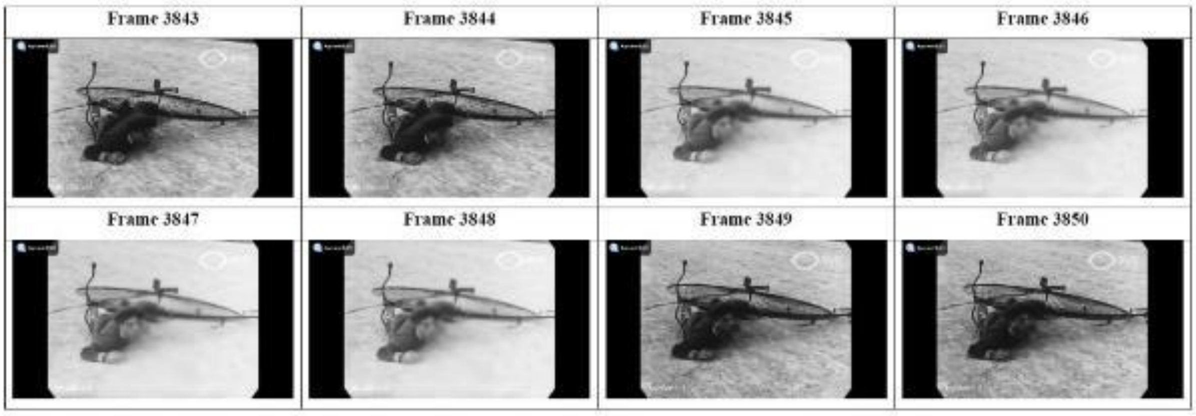


Fig. 6: An excerpt of the frame sequence of an old film

The damage caused by flickering is distinct because of the evident change in brightness. The brightness increases when the video is played from frame 3844 to frame 3845. The phenomenon continues until frame 3848. At frame 3849, the brightness dims.

Pandel [14] introduces a metric which evaluates flickering of each frame in a video by computing the difference of luminance of a macroblock between subsequent frames. The measurement is designed to appraise flickering in predictive coded video sequences as in H.264 compression.

The basic form of the model is a two-state model (Fig. 7) that reflects the update (*u*) and no update (*n*) state of a macroblock.

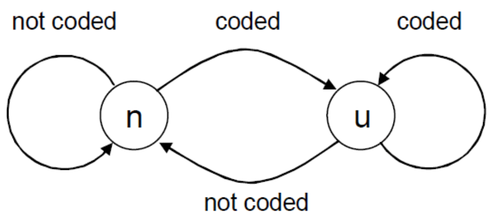

$$\begin{tikzcd} n \arrow[loop left, "\text{not coded}"] \arrow[r, bend left, "\text{coded}"] & u \arrow[l, bend left, "\text{not coded}"] \arrow[loop right, "\text{coded}"] \end{tikzcd}$$

Fig. 7: Two-state model reflecting the update (*u*) and no update (*n*) state of a macroblock [14]

The change of state *D* in (3) is the change in the mean square difference between the luminance $g_i(x)$ of the summation of the pixel *x* in a macroblock *a* between frame *i* and frame *i-1*. *n* is the number of pixels in a macroblock.

$$D = \frac{1}{n}\sum_{x}\{g_i(x,a) - g_{i-1}(x,a)\}^2 \tag{3}$$

Fig. 8 demonstrates the following four conditions that alter the state of the macroblock from one state to another according to the difference between the change of state *D* and a manually set threshold Θ.

1. If the current state is at *n* and $D < \Theta$, the state of the macroblock remains at *n*.
2. If the current state is at *n* and $D \geq \Theta$, the state of the macroblock goes to *u*.
3. If the current state is at *u* and $D = 0$, the state of the macroblock goes to *n*.
4. If the current state is at *u* and $D > \Theta$, the state of the macroblock remains at *u*.

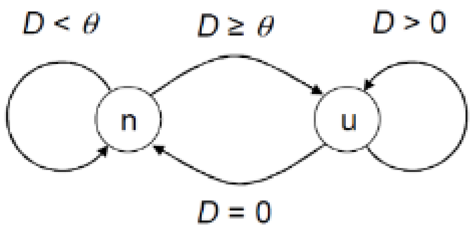


Fig. 8: Two-state model of changing the state of a macroblock with conditions [14]

Primarily, the flickering value $m_{fl,i}$ of macroblock *i* in (4) is computed the number of changes in the state of that macroblock $n_{c,i}$ over its maximum changes $n_{c,max}$ within a period. The duration can be a few hundred milliseconds to one second.

$$m_{fl,i} = \frac{n_{c,i}}{n_{c,\max}} \tag{4}$$

After that, $m_{fl,i}$ will be adjusted to $m_{flA,k}$ in (5) by computing the summation of the number of changes in the state of macroblock i together with those of its adjacent macroblocks. All the macroblocks will form a macroblock set named Ak. The sum of the values will then be divided by the product of its maximum changes $n_{c,max}$ and the total number of macroblocks *b* in set A_k.

$$m_{flA,k} = \frac{1}{b \cdot n_{c,\max}} \cdot \sum_{i \in A_k} n_{c,i} \tag{5}$$

The overall flickering value $m_{flA,max}$ of a frame in (6) is the flickering value of the macroblock set with the maximum number of changes in state. $m_{flA,max}$ of each frame will be used as the input of the LSTM model.

$$m_{flA,\max} = \max_{k}(m_{flA,k}) \tag{6}$$

4.1.1. Mosquito Noise

Mosquito noise is a temporal artifact which can be seen in smooth areas due to fluctuations of luminance around "high-contrast edges or moving objects", caused by "the coding differences for the same area of a scene in consecutive frames of a sequence" [15]. As the name implies, the impairment can be easily identified because it looks like mosquitos flying around the edges of an object (Fig. 9).

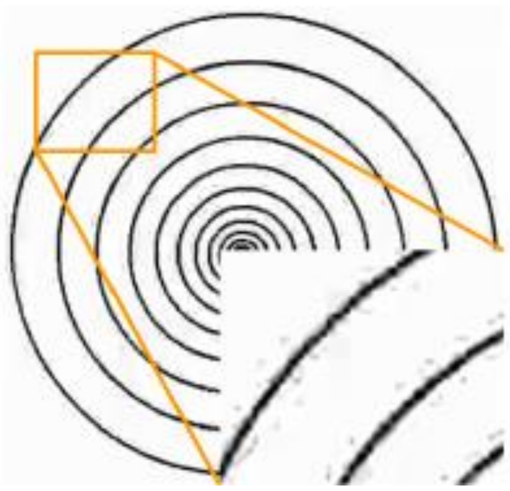


Fig. 9: Mosquito noise on spiral objects formed by H.264 compression [16]

Since mosquito noise happens around the edges of objects, this type of damage can be measured by detecting and calculating the difference of pixel intensity of edges between consecutive frames.

Punchihewa and Keerl [17] propose a way to quantify the mosquito noise of a video frame with a circular test pattern. Equation (7) measures the absolute pixel intensity difference $P(\alpha, i)_{diff}$ of the edges of the test object between consecutive frames t and t-1. α is the profile angle (from 0^{o} to 359^{o}) of the test pattern and i is the pixel position.

$$P(\alpha, i)_{diff} = |P(\alpha, i)_t - P(\alpha, i)_{t-1}| \tag{7}$$

The mosquito noise diagram $mnd(\Delta p, \alpha)$ in (8) of a video frame *t* is computed by summing up $P(\alpha, i)_{diff}$ of all pixels in a frame. *x* is the total number of pixels in a frame.

$$mnd(\Delta p, \alpha) = \sum_{i=0}^{x} P(\alpha, i)_{diff} \tag{8}$$

Nevertheless, this method only incorporates with the edges of the circular test pattern used by the researchers. Thus, it should be modified to fit into this project.

To achieve that, this project calculates pixel values of the edges of objects using a high-pass filter, particularly a Laplacian filter, to isolate the edge lines from the non-edge pixels. As the pixel intensity of edges differs from their neighbours with a high variation [18], edge pixels appear to be local maxima of a function if its first derivative is determined, as shown in the right of Fig. 10.

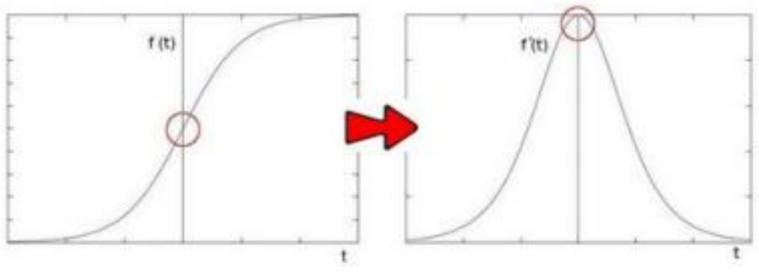


Fig. 10: A function *f(x)* (left) and its first derivative *f'(x)* (right) [18]

By obtaining the second derivative of the function, the intensity of the pixels at zero is neither increasing nor decreasing, so they are the edge pixels (Fig. 11).

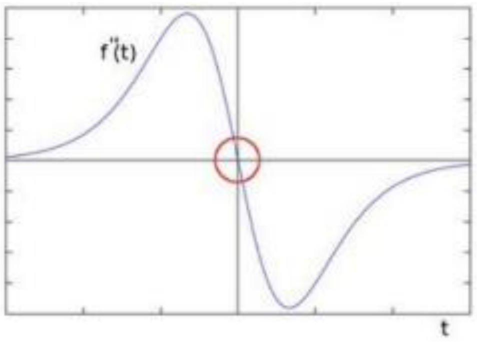


Fig. 11: The second derivative *f''(x)* of the function *f(x)* [18]

Hence, the Laplacian filter detects edges in a frame by deducing the second derivative of the image matrix in (9).

$$\text{Laplace}(f) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{9}$$

Take frame 20 of Kimono1_H264_1 (MOS: 0.7963) and Kimono1_H264_4 (MOS: 6.9259) as an example (Fig. 12) to zoom in and use the Laplacian filter to process the collar of the kimono (circled in red) (Fig. 13).

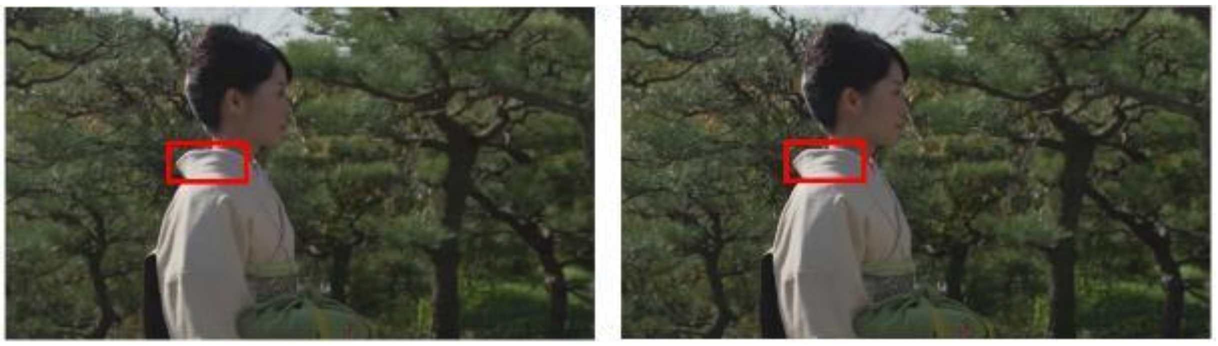


Fig. 12: Frame 20 of Kimono1_H264_1 (left) and Kimono1_H264_4 (right)

As observed, the frame of a video with the lower MOS (Kimono1_H264_1), suffers from more mosquito noise than that of the higher MOS (Kimono1_H264_4), with more “mosquitos” surrounding the edges of the collar. The edge pixels in Kimono1_H264_1 exhibit to be "eaten" by the noise. As a result, the video of the lower MOS has a lower edge pixel intensity. The total pixel intensities in Kimono1_H264_1 and Kimono1_H264_4 are 343355 and 461281 respectively. Note that only pixels with values higher than 10 are counted as edge pixels in order to speed up the computation time.

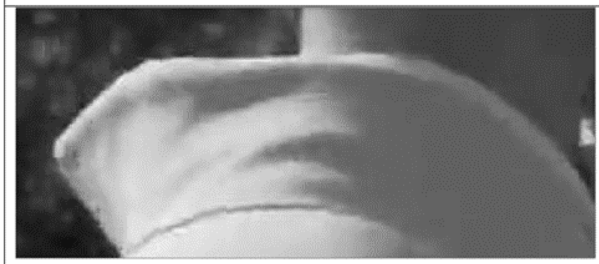
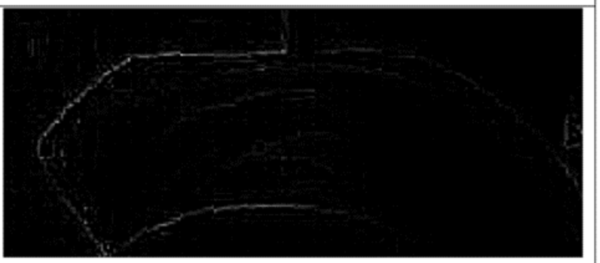
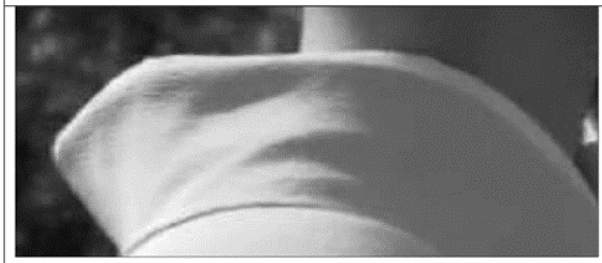
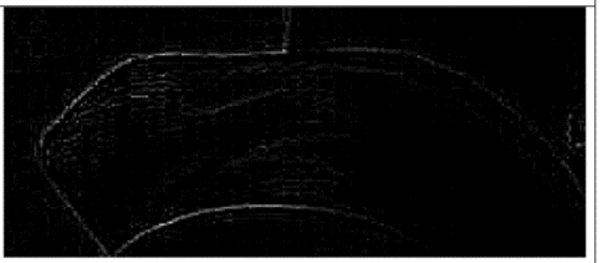
A Part of the Original Image (Greyscale)	The Part after Filtered by the Laplacian Filter
Frame 20, Kimono1_H264_1 (MOS: 0.7963)	
	
Frame 20, Kimono_H264_4 (MOS: 6.9259)	
	

Fig. 13: The original and filtered images of the collar of the kimono in frame 20 of Kimono1_H264_1 (upper) and Kimono1_H264_4 (lower)

By adopting the above idea, the pixel intensity $P(i,t)$ of an edge pixel *i* in frame *t* can be obtained by filtering the image matrix using the Laplacian filter as below:

$$P(i,t) = Laplace[m(i,t)] \tag{10}$$

The absolute pixel intensity difference $P(i,t)_{diff}$ of pixel *i* between consecutive frames *t* and *t-1* is as follows:

$$P(i,t)_{diff} = |P(i,t) - P(i,t-1)| \tag{11}$$

The mosquito noise diagram $mnd(\Delta p, \alpha)$ of video frame *i* is replaced by the number of existing pixels $exist(t)$ of that frame in (12). It is computed by adding $P(i,t)_{diff}$ of all *x* number of pixels in that frame.

$$exist(t) = \sum_{i=0}^{x} P(i,t)_{diff} \tag{12}$$

This project further expresses mosquito noise as a probability by subbing $exist(t)$ in a sigmoid function. A typical sigmoid function is computed by:

$$sigmoid(z) = \frac{1}{1+e^{-z}} \tag{13}$$

The graph of the formula is shown in Fig. 14.

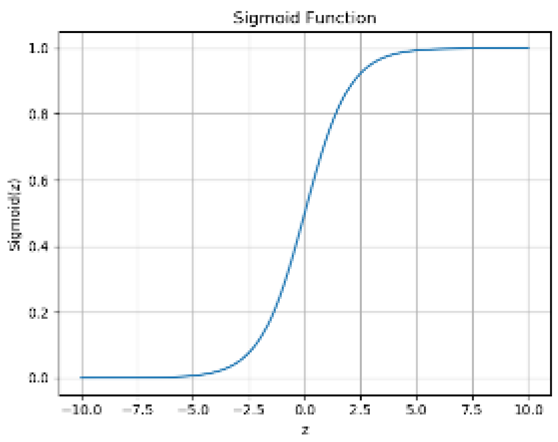


Fig. 14: A graph of a sigmoid function

As a frame of a video of a better MOS has a higher number of existing pixels, the difference of the pixel intensity between consecutive frames tends to be higher. According to the graph in Fig. 13, if subbing a higher $exist(t)$ as *z* into the sigmoid function, the outcome will be higher. To obtain the probability of a frame having mosquito noise, the sigmoid function should be subtracted from 1. Therefore, the final function to quantify the mosquito noise $MN(t)$ of frame *t* is as follows:

$$MN(t) = 1 - \frac{1}{1 + e^{-Kz}} \tag{14}$$

where *z* is $exist(t)$ in (12) and *K* is a coefficient to scale to $exist(t)$.

$MN(t)$ in (14) represents the mosquito noise of each frame and it will be the input of the LSTM model.

4.2. Given Spatial and Temporal Artifacts

Metrics for measuring some spatial and temporal artifacts are given in a previous LSTM NR-VQA study [6]. For spatial impairments, they are Gaussian noise, blocking, blurriness and sharpness artifacts. But among all spatial inputs, blurriness is said to be very fluctuating, so it will be rejected in this experiment. Aside from that, the only temporal artifact provided is frame freeze, with its 5-class one-hot encodings of freeze percentages.

Overall, there are a total of three spatial and one temporal data and the associated one-hot classes given by the former research. They are going to be employed in the NR-VQA model together with the new temporal artifacts to predict MOS of each video.

4.3. Model Architecture for NR-VQA: Long Short-Term Memory (LSTM)

The architecture of the program for NR-VQA is LSTM. The fundamental of LSTM is a recurrent neural network (RNN). The major characteristic which differentiates RNN from other neural networks is the self-looping of the output of a cell [19]. But RNN faces the problem of vanishing and exploding when the weight associated with a hidden cell is smaller and greater than 1. To resolve that, LSTM is introduced.

RNN (Fig. 15) has only one gate with a hyperbolic tangent (tanh) activation function to handle inputs, whereas LSTM (Fig. 16) has three gates: a forget gate, an input gate and an output gate, and a cell state.

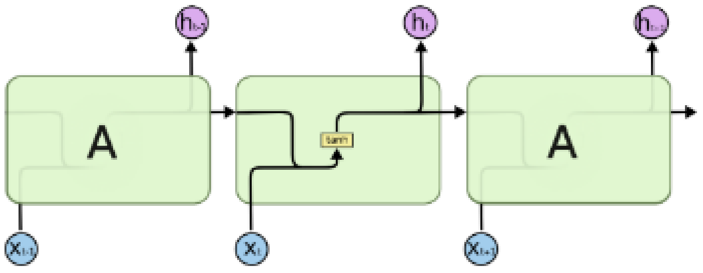


Fig. 15: The structure of an RNN [20]

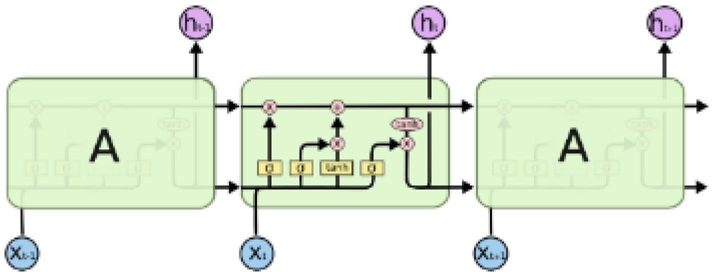
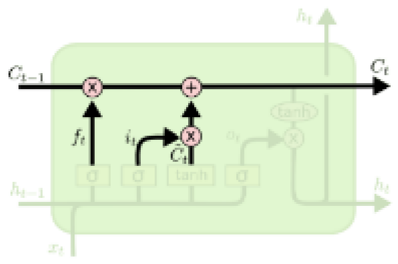


Fig. 16: The structure of LSTM [20]

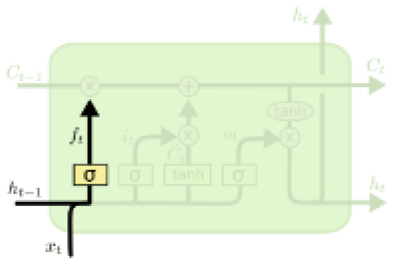
The cell state c_t is a vector containing the long-term memory that reserves part of the information passed by the last cell at *t-1* to the next cell at *t* (Fig. 17). It is formed by forgetting and increment of the previous cell. By updating the cell state over time, the network can handle "long-range dependencies" and instabilities due to exceptional patterns in inputs [19].



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Fig. 17: The cell state of LSTM [20]

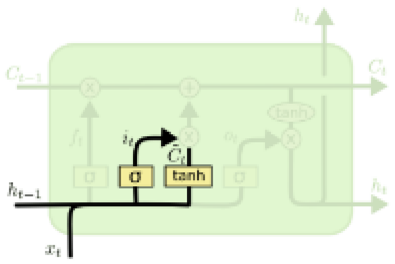
The forget gate is a sigmoid (σ) function (Fig. 18). It decides which elements in the cell state vector from the previous cell to be reset to 0 [19].



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Fig. 18: The forget gate of LSTM [20]

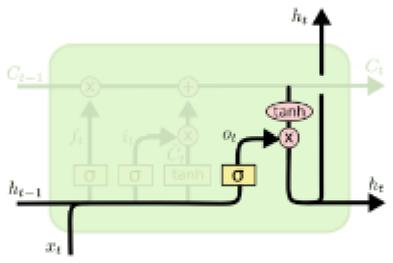
The input gate is combined with a sigmoid (σ) and a hyperbolic tangent (tanh) function (Fig. 19). It determines which element has to be incremented in the last cell state to form information in the new cell state [19].



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Fig. 19: The input gate of LSTM [20]

The output gate multiplies a sigmoid function (σ) with the hyperbolic tangent (tanh) function (Fig. 20). It controls if the functional form of the cell state leaks to the hidden state h_t to the next cell by assigning o_t as either 0 or 1 [19].



$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Fig. 20: The output gate of LSTM [20]

The nature of LSTM is related to time sequence, so they are suitable for analysing time-series like video and language processing, which the inputs are in a sequence and the outputs depend on timestamp [19]. Hence, LSTM is going to be adopted in this project.

4.4. Principal Component Analysis (PCA) Feature Selection

In the later stage of this project, feature selection, specifically using principal component analysis (PCA) is conducted to better the performance of the model.

PCA is a statistical technique to fit "a low dimensional affine subspace S of dimension d" D to a set of points $\{x_1, x_2, \ldots, x_N\}$ in a high-dimensional space R^D" [21] [22]. For instance, there a matrix of multivariate random variables $x \in R^D$ and an integer $d < D$ representing the number of principal components (PCs) of $x, y \in R^D$. The PCs are described by

$$y_i = \boldsymbol{u}_i^\top \boldsymbol{x} \in \mathbb{R}, \quad \boldsymbol{u}_i \in \mathbb{R}^D, \quad i = 1, 2, \ldots, d \tag{15}$$

(15) reveals that x is projected to an eigenvector u_i of its covariance matrix $\sum_x = \mathrm{E}[xx^\mathrm{T}]$. The maximisation of the variance of y_i is controlled by

$$\boldsymbol{u}_i^\top \boldsymbol{u}_i = 1 \quad \text{and} \quad \mathrm{Var}(y_1) \geq \mathrm{Var}(y_2) \geq \cdots \geq \mathrm{Var}(y_d) > 0 \tag{16}$$

Based on (16), for i is the order of a PC, the first PC of x has the maximum variance, the second PC has the second largest variance, and so on. The sum of the variance of all PCs equals to 1. Suppose the eigenvector of the first PC u_1^* is to be computed, it can be done by

$$\boldsymbol{u}_1^* = \underset{\boldsymbol{u}_1 \in \mathbb{R}^D}{\arg\max} \, \mathrm{Var}(\boldsymbol{u}_1^\top \boldsymbol{x}) \quad \text{s.t.} \quad \boldsymbol{u}_1^\top \boldsymbol{u}_1 = 1 \tag{17}$$

Practically, the d PCs of x can be computed at the same time, given that a random vector $y = [y_1, y_2, \ldots, y_d]^T \in R^d$, the matrix of eigenvectors $U = [u_1, u_2, \ldots, u_d] \in R^{D \times d}$ and $y = U^T x$. This is achieved by

$$\Sigma_{\boldsymbol{y}} \doteq \mathbb{E}[\boldsymbol{y}\boldsymbol{y}^\top] = U^\top \mathbb{E}[\boldsymbol{x}\boldsymbol{x}^\top] U = U^\top \Sigma_{\boldsymbol{x}} U \tag{18}$$

By applying PCA in this project, the input values of the datasets will be transformed and some of them will be eliminated depending on their variance. From the view of PCA, the PCs (or features in this case) with higher variance possess greater importance among all the data. Thus, they have a higher priority to be selected and preserved. As a result, only the chosen PCs will be analysed. Despite the reduction in the input dimension, the process does not deter the operation of the model.

One example of using PCA for input dimension reduction is a handwritten digit recognition by a neural network. Each data has 28 * 28 = 784 input pixels, which is the same as the dimension of a handwritten digit image (Fig. 21).

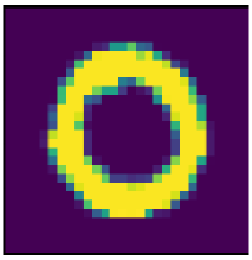


Fig. 21: An original handwritten 0 [22]

As mentioned, the pixels with a higher variance will be kept for training. To decide the number of training pixels, firstly, a PCA decomposition graph containing a curve of the cumulative variance of the pixels should be plotted as in Fig. 22.

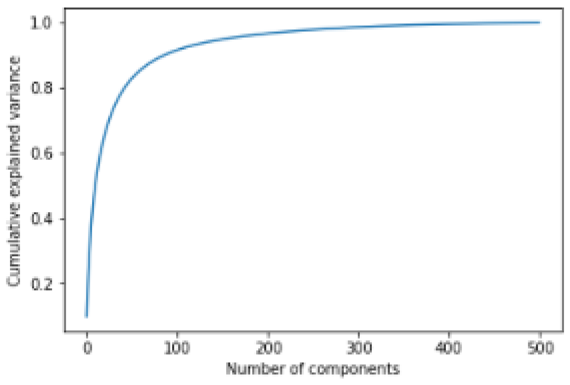


Fig. 22: PCA decomposition graph for handwritten digit recognition [22]

By observing the graph, when the number of PCs is 100, it is obvious that the change in the cumulative variance becomes slow. Hence, 100 PCs should be sufficient to train the network. After utilising PCA, the reconstructed input data will be illustrated as in Fig. 23.

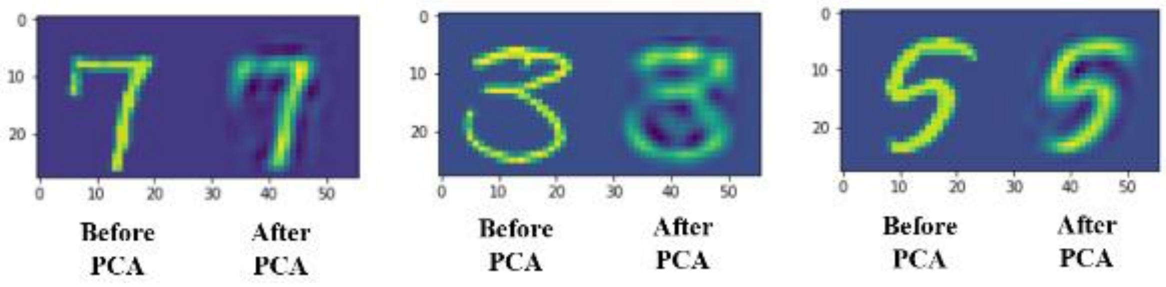


Fig. 23: The digit images before and after undergoing PCA [22]

The transformed images look blurrier due to the change in the pixel values and the reduction of inputs. Even so, they are recognisable because the distribution of the data does not change. Therefore, the neural network can still identify handwritten digits from the processed images.

The reason for having feature selection in this project is due to the curse of dimensionality. It implies that the dimension of inputs is too high for training a small dataset [23]. The performance of a model deteriorates as analysing high dimensional data require expensive processing power and resources [24]. Moreover, when the number of samples is smaller than their high dimensional inputs, traditional evaluation tools become incapable of explaining the data [24].

The issue of small dataset versus large input dimension is illustrated in Fig. 24. As the input dimension grows, the output data become more and more sparse, as shown from (a) to (c). The gaps between each data become larger. Given the high dimensional inputs, the model may predict decent outputs of the training data. However, when it works on the testing data, it has a higher chance to predict outcomes which hit on the spaces far away from the ideal data. The model will prone to overfit and perform poorly. Thus, it is necessary to perform feature selection by PCA for the NR-VQA in this project.

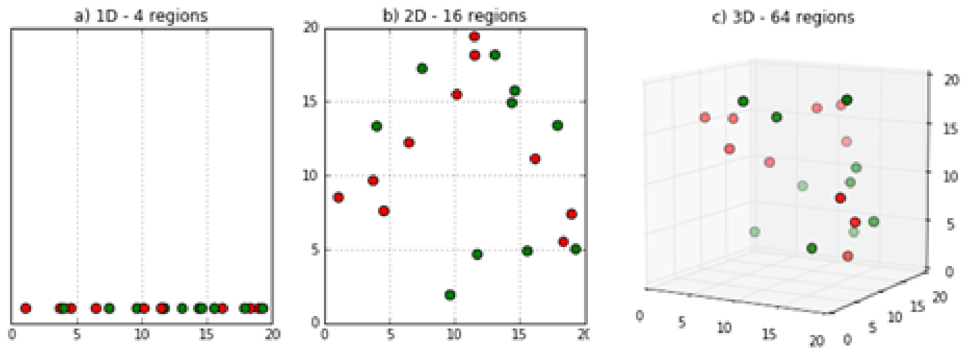


Fig. 24: Distribution of output data of 1D (a), 2D (b) and 3D (c) inputs [25]

4.5. Overall Model for NR-VQA

4.5.1. Without PCA Feature Selection

Fig. 25 pictures the LSTM Model for NR-VQA without feature selection.

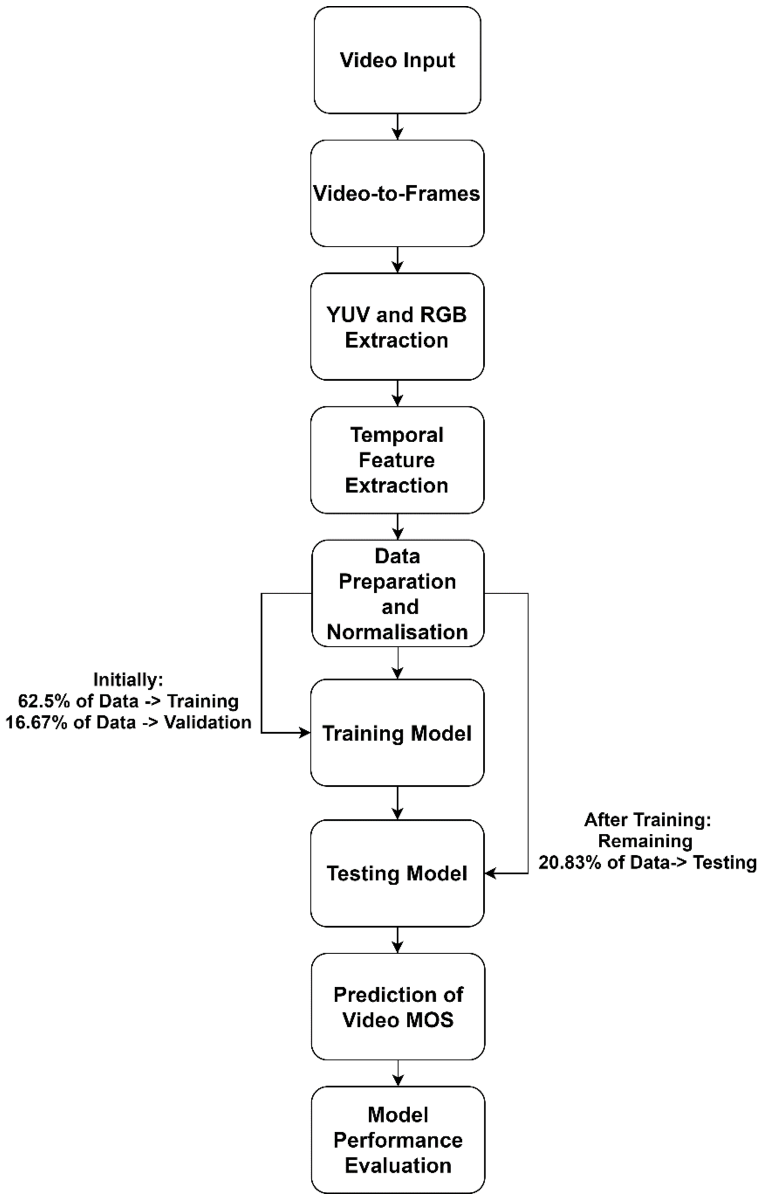


Fig. 25: NR-VQA model without PCA feature selection

The following are the steps of this proposed model:

1. Video Input: The YUV files of the videos are prepared in this stage as the inputs for feature extraction.

2. Video-to-Frames: The video files will be read to obtain video frames.

3. YUV and RGB Extraction: YUV and RGB matrices of each frame of the videos are extracted for further computations.

4. Temporal Feature Extraction: The three temporal features will be collected in this stage by analysing the video frames and the luminance matrices.

5. Data Preparation and Normalisation: The provided data from the previous NR-VQA study will be combined with the new temporal features. After that, the raw data will go through operations such as classification and transformation in order to optimise the performance of the training. The processed figures will be separated into three groups: 62.5 % (60 videos) will be the training set, 16.67% (16 videos) will be the validation set, and 20.83% (20 videos) will be the testing set.

6. Training Model: The training set will be fed to the LSTM model for the prediction of video MOS. To avoid overfitting and underfitting, the validation set will be used to unbiasedly judge the model fit when the model is learning from the training set and for hyperparameter tuning.

7. Testing Model: After the training, the testing set will be put into the model with the lowest validation loss.

8. Prediction of Video MOS: The outcomes from the last stage are the predicted MOS of the testing set. They will be employed to assess the achievement of the trained model.

9. Model Performance Evaluation: This will be done when the training results of different hidden nodes are gathered. The performance will be analysed by mean squared error (MSE) and Pearson correlation coefficient (PCC) between the ideal and the predicted MOS of the testing set.

4.5.2. With PCA Feature Selection

Fig. 26 is the LSTM Model for NR-VQA with feature selection.

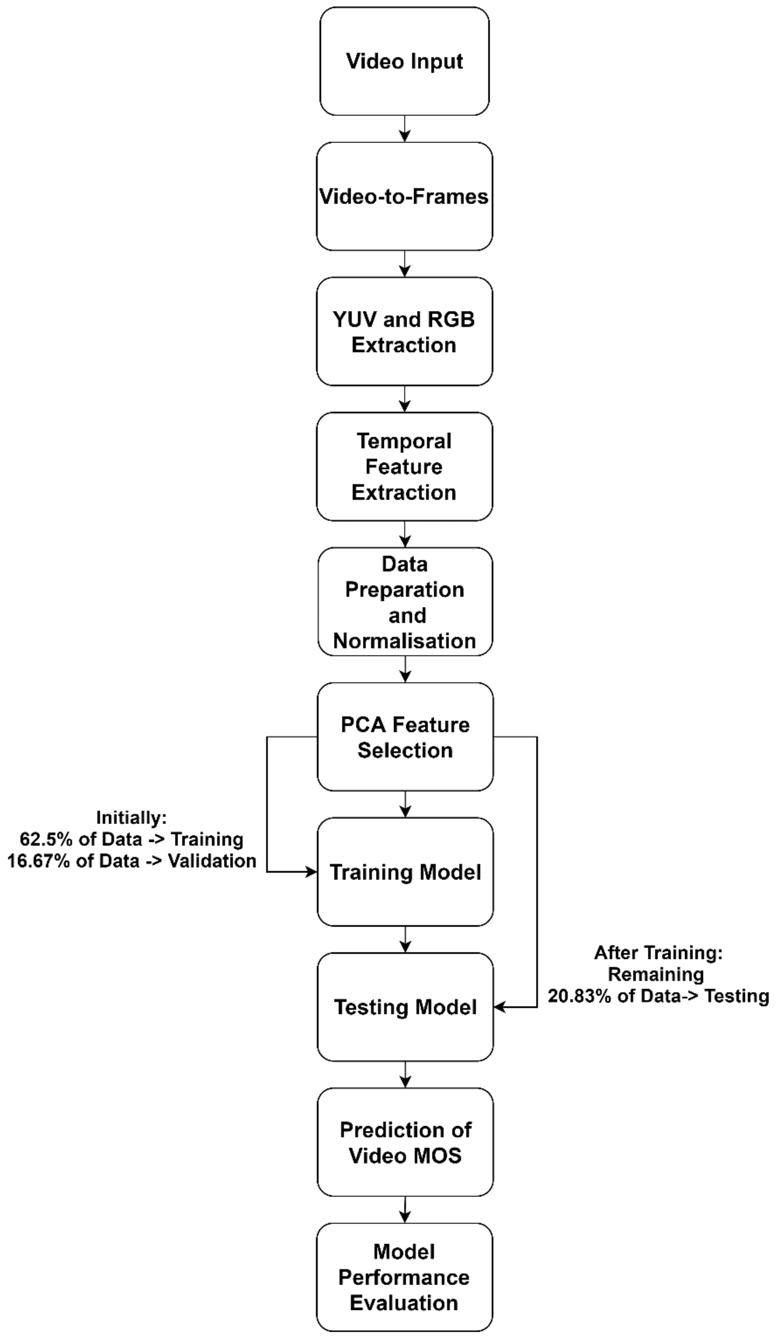


Fig. 26: NR-VQA model with PCA feature selection

As mentioned in the Methodology, feature selection will be made in the later stage of this project. PCA Feature Selection is thus added after Data Preparation and Normalisation to select a certain number of features to undergo training for better model performance. Other steps remain the same as in the model without feature selection.

5. Project Implementation

5.1. Project Development Environment

This project uses Python as the programming language to write the program. Various APIs and libraries are provided for the environment of this language to work on machine learning and multimedia processing tasks.

The principle library for machine learning is Keras. It provides various functions for building a machine learning model.

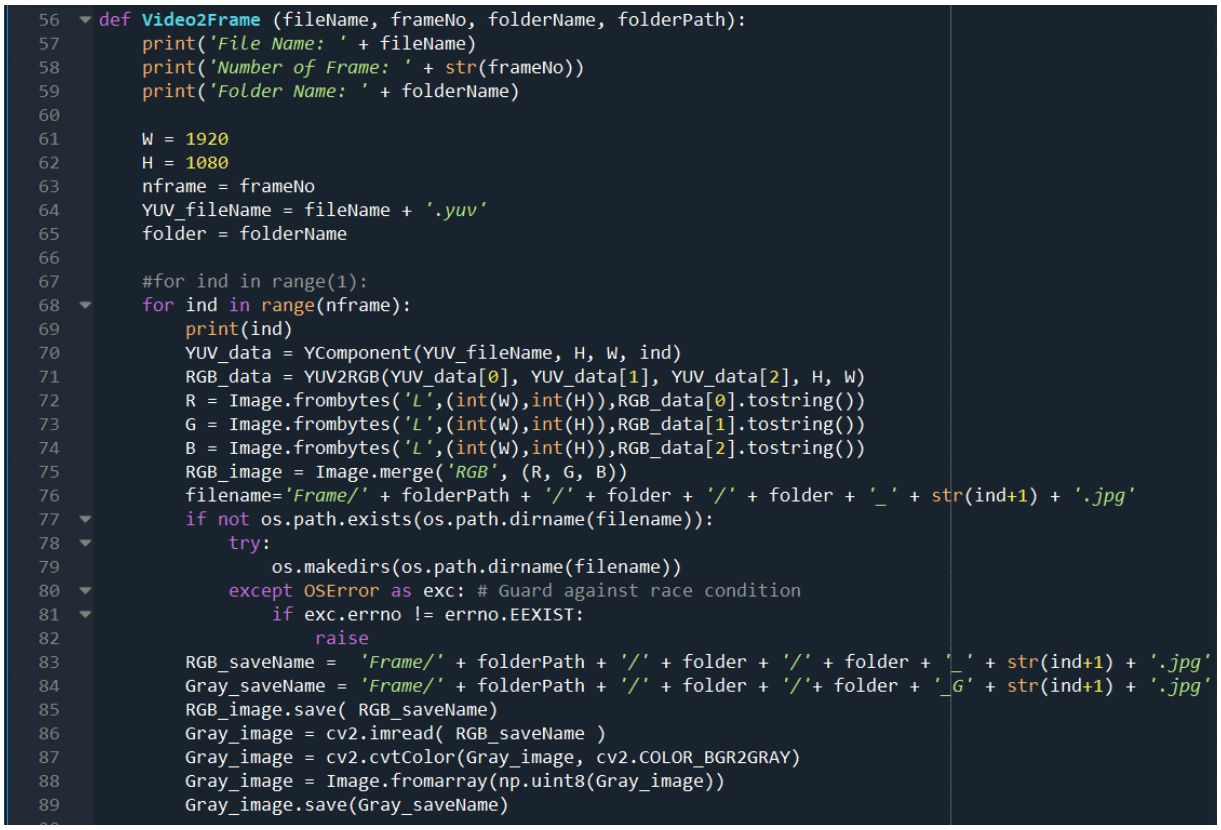
The table below are other essential libraries for developing the program and their functions in the project:

Name of Library	**Function(s) in the Project**
OpenCV	Mines Information from Videos
Matplotlib	Provides Plotting Tools
Numpy	Constructs and Manipulates Arrays and Matrices
scikit-learn	Provides Algorithms for Data Processing and PCA Feature Selection
SciPy	Offers Formulas for Scientific Computing
Pandas	Manipulates and Analyses Data from Excel Files using DataFrame object
Openpyxl	Manipulates Excel Files

Table 1: Essential libraries used in this project and their functions

5.2. Slicing Videos into Frames

The videos can be sliced into frames to facilitate this experiment, especially for the extraction of mosquito noise. To complete this task, the function Video2Frame (Fig. 27) provided by the former study [6] generates both colour and greyscale pictures of the frames from the videos.



```
56  def Video2Frame (fileName, frameNo, folderName, folderPath):
57      print('File Name: ' + fileName)
58      print('Number of Frame: ' + str(frameNo))
59      print('Folder Name: ' + folderName)
60  
61      W = 1920
62      H = 1080
63      nframe = frameNo
64      YUV_fileName = fileName + '.yuv'
65      folder = folderName
66  
67      #for ind in range(1):
68      for ind in range(nframe):
69          print(ind)
70          YUV_data = YComponent(YUV_fileName, H, W, ind)
71          RGB_data = YUV2RGB(YUV_data[0], YUV_data[1], YUV_data[2], H, W)
72          R = Image.frombytes('L',(int(W),int(H)),RGB_data[0].tostring())
73          G = Image.frombytes('L',(int(W),int(H)),RGB_data[1].tostring())
74          B = Image.frombytes('L',(int(W),int(H)),RGB_data[2].tostring())
75          RGB_image = Image.merge('RGB', (R, G, B))
76          filename='Frame/' + folderPath + '/' + folder + '/' + folder + '_' + str(ind+1) + '.jpg'
77          if not os.path.exists(os.path.dirname(filename)):
78              try:
79                  os.makedirs(os.path.dirname(filename))
80              except OSError as exc: # Guard against race condition
81                  if exc.errno != errno.EEXIST:
82                      raise
83          RGB_saveName =  'Frame/' + folderPath + '/' + folder + '/' + folder + '_' + str(ind+1) + '.jpg'
84          Gray_saveName = 'Frame/' + folderPath + '/' + folder + '/'+ folder + '_G' + str(ind+1) + '.jpg'
85          RGB_image.save( RGB_saveName)
86          Gray_image = cv2.imread( RGB_saveName )
87          Gray_image = cv2.cvtColor(Gray_image, cv2.COLOR_BGR2GRAY)
88          Gray_image = Image.fromarray(np.uint8(Gray_image))
89          Gray_image.save(Gray_saveName)
```

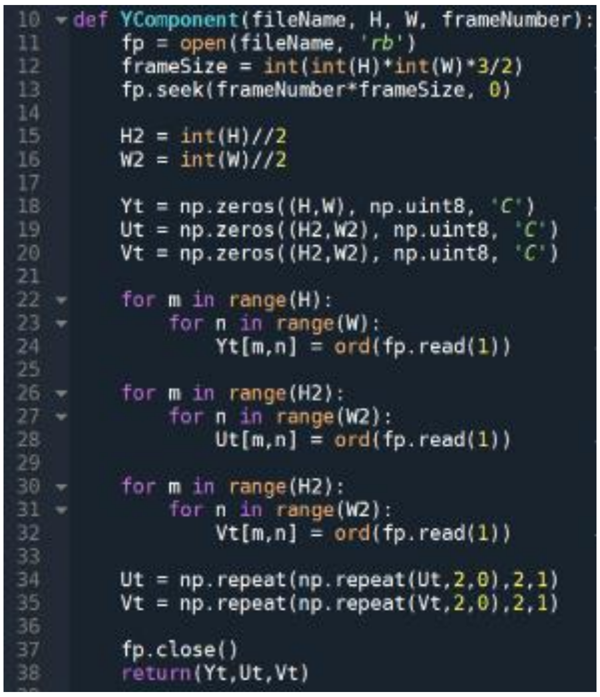
Fig. 27: Video2Frame function [6]

The process of slicing videos into frame involves the extraction of YUV and RGB matrices of each frame of the videos. YUV matrices are collected by YComponent, while RGB matrices are computed by importing the YUV arrays to YUV2RGB in row 71. The information of each frame is read byte by byte from the RGB matrices from rows 72 to 75. Eventually, the function outputs colour and greyscale JPEG photos of the video frames.

5.3. Extraction of YUV and RGB Matrices

YUV and RGB matrices are collected by the functions given from the previous research [6]. YUV encodes videos and images using the luminance (Y) and chrominance (U and V) components in the files, whereas RGB uses the red, green and blue colours to do so.

YUV arrays can be acquired by the function YComponent (Fig. 28) below.

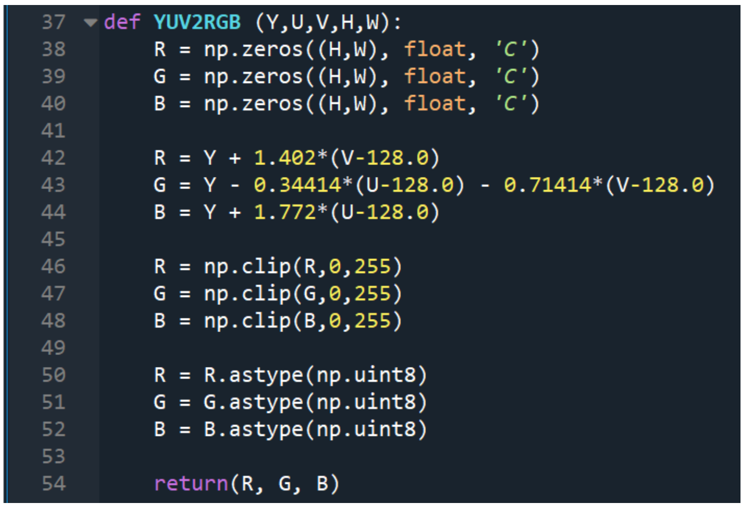


```python
def YComponent(fileName, H, W, frameNumber):
    fp = open(fileName, 'rb')
    frameSize = int(int(H)*int(W)*3/2)
    fp.seek(frameNumber*frameSize, 0)

    H2 = int(H)//2
    W2 = int(W)//2

    Yt = np.zeros((H,W), np.uint8, 'C')
    Ut = np.zeros((H2,W2), np.uint8, 'C')
    Vt = np.zeros((H2,W2), np.uint8, 'C')

    for m in range(H):
        for n in range(W):
            Yt[m,n] = ord(fp.read(1))

    for m in range(H2):
        for n in range(W2):
            Ut[m,n] = ord(fp.read(1))

    for m in range(H2):
        for n in range(W2):
            Vt[m,n] = ord(fp.read(1))

    Ut = np.repeat(np.repeat(Ut,2,0),2,1)
    Vt = np.repeat(np.repeat(Vt,2,0),2,1)

    fp.close()
    return(Yt,Ut,Vt)
```

Fig. 28: YComponent function [6]

The function reads the YUV file of a video. *Yt* is the matrix of luminance of the dimension of the height *H* (1080) and the width *W* (1920) of the video. *Ut* and *Vt* are the matrices of the colour components of half of the dimension. The program returns all *Yt*, *Ut* and *Vt* into a 3-D matrix. All three components are useful in getting RGB matrices and the JPEG photos of the video frames. In the extractions of jerkiness and flickering, only *Yt* is used for their computations.

RGB arrays can be obtained by the function YUV2RGB (Fig. 29).



```python
def YUV2RGB (Y,U,V,H,W):
    R = np.zeros((H,W), float, 'C')
    G = np.zeros((H,W), float, 'C')
    B = np.zeros((H,W), float, 'C')

    R = Y + 1.402*(V-128.0)
    G = Y - 0.34414*(U-128.0) - 0.71414*(V-128.0)
    B = Y + 1.772*(U-128.0)

    R = np.clip(R,0,255)
    G = np.clip(G,0,255)
    B = np.clip(B,0,255)

    R = R.astype(np.uint8)
    G = G.astype(np.uint8)
    B = B.astype(np.uint8)

    return(R, G, B)
```

Fig. 29: YUV2RGB function [6]

When a YUV matrix is obtained from YComponent, the components can be separated to be put into YUV2RGB to get the RGB matrix. R, G and B are computed by Y, U and V from rows 42 to 44 according to (19), (20) and (21) [6]. The RGB matrices are going to be used when extracting frames from the videos.

$$R = Y + 1.4075 \times (V - 128) \tag{19}$$

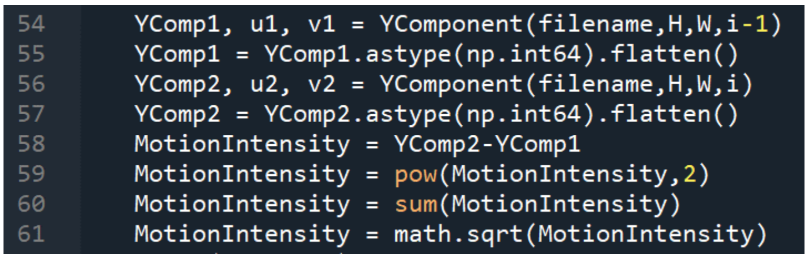
$$\mathrm{G} = \mathrm{Y} - 0.3455 \times (\mathrm{U} - 128) - 0.7169 \times (\mathrm{V} - 128) \tag{20}$$

$$B = Y + 1.779 \times (U - 128) \tag{21}$$

5.4. Extraction of Temporal Features

5.4.1. Jerkiness

Before determining the jerkiness impairment of each frame, the motion intensity should be found out first. The following code (Fig. 30) does the task:



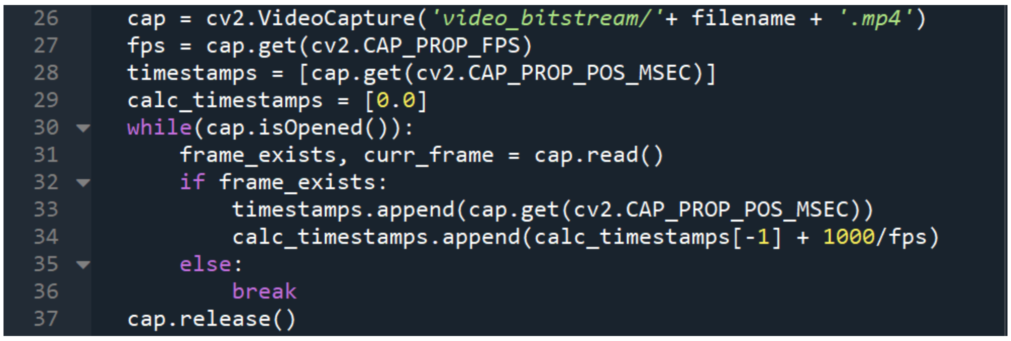
```
54        YComp1, u1, v1 = YComponent(filename,H,W,i-1)
55        YComp1 = YComp1.astype(np.int64).flatten()
56        YComp2, u2, v2 = YComponent(filename,H,W,i)
57        YComp2 = YComp2.astype(np.int64).flatten()
58        MotionIntensity = YComp2-YComp1
59        MotionIntensity = pow(MotionIntensity,2)
60        MotionIntensity = sum(MotionIntensity)
61        MotionIntensity = math.sqrt(MotionIntensity)
```

Fig. 30: Code for motion intensity calculation

where *i* is the frame number, *H* is the height (1080) and *W* is the width (1920) of the frame.

In line 54 and 56, the luminance components *YComp1* of frame *i-1* and *YComp2* of frame *i* are obtained as matrices respectively from YComponent in Fig. 28. The values in both matrices will be set from unsigned 8-bit to signed 64-bit integers in line 55 and 57 for handling negative values after the subtraction in line 58. The commands from line 58 to 61 follow the calculation in (1).

The timestamp for each frame to display can be retrieved by running the commands in Fig. 31.

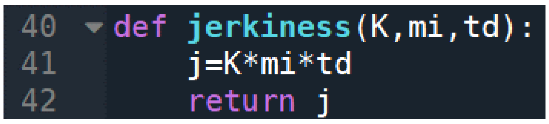


```
cap = cv2.VideoCapture('video_bitstream/'+ filename + '.mp4')
fps = cap.get(cv2.CAP_PROP_FPS)
timestamps = [cap.get(cv2.CAP_PROP_POS_MSEC)]
calc_timestamps = [0.0]
while(cap.isOpened()):
    frame_exists, curr_frame = cap.read()
    if frame_exists:
        timestamps.append(cap.get(cv2.CAP_PROP_POS_MSEC))
        calc_timestamps.append(calc_timestamps[-1] + 1000/fps)
    else:
        break
cap.release()
```

Fig. 31: Code for recording the timestamp of each frame in a video

The mp4 file of a video will be input into the VideoCapture of OpenCV. While the file is being read, the current position in millisecond and the frame rate will be captured and put into the lists *timestamps* and *calc_timestamps* respectively. The timestamps of the video are the difference between *timestamps* and *calc_timestamps*.

After acquiring the motion intensities and display times, the function jerkiness (Fig. 32) quantifies the jerkiness of a frame. The value is also scaled by $K = 0.01$.



```
def jerkiness(K,mi,td):
    j=K*mi*td
    return j
```

Fig. 32: jerkiness function

5.4.2. Flickering

Before figuring flickering, all video frames need to be divided into macroblocks. The dimension of a frame is 1920 * 1080, so it will be divided into 16 * 8 macroblocks (Fig. 33).

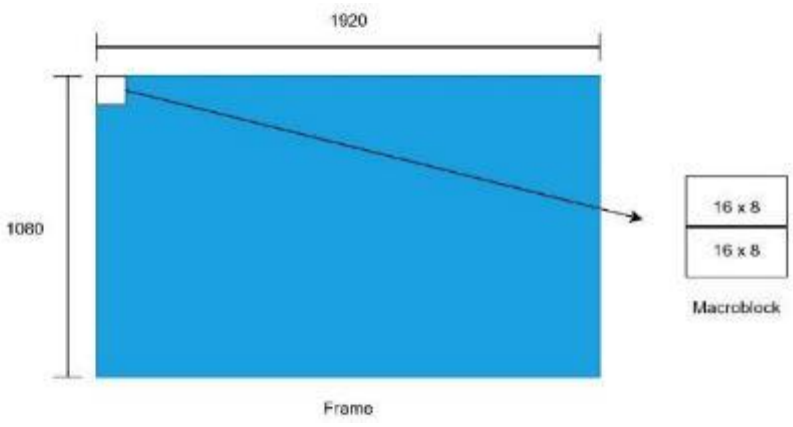
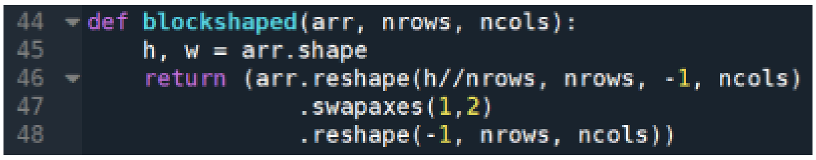


Fig. 33: Dividing a frame into macroblocks

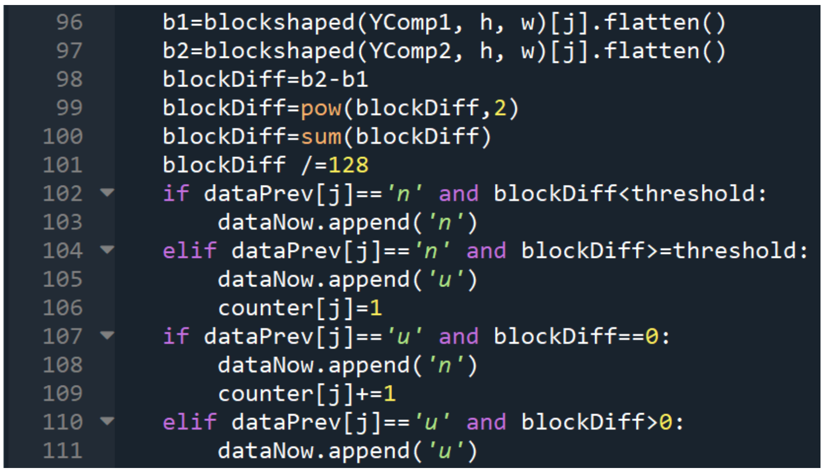
The following function blockshaped (Fig. 34) divides a frame into macroblock, where *arr* is the frame matrix, *nrows* is the height (1080) and *ncols* is the width (1920).



```
def blockshaped(arr, nrows, ncols):
    h, w = arr.shape
    return (arr.reshape(h//nrows, nrows, -1, ncols)
                .swapaxes(1,2)
                .reshape(-1, nrows, ncols))
```

Fig. 34: blockshaped function

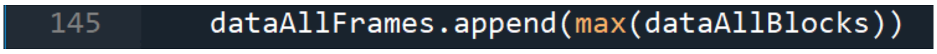
The mean square difference of luminance between the current macroblock *b2* and the previous macroblock *b1* is obtained by running the codes below (Fig. 35). Note that the number of pixels in a macroblock 128 represents *n* in (3). A manually set *threshold* 300 is used in the if-cases to calculate the states of a macroblock.



```
b1=blockshaped(YComp1, h, w)[j].flatten()
b2=blockshaped(YComp2, h, w)[j].flatten()
blockDiff=b2-b1
blockDiff=pow(blockDiff,2)
blockDiff=sum(blockDiff)
blockDiff /=128
if dataPrev[j]=='n' and blockDiff<threshold:
    dataNow.append('n')
elif dataPrev[j]=='n' and blockDiff>=threshold:
    dataNow.append('u')
    counter[j]=1
if dataPrev[j]=='u' and blockDiff==0:
    dataNow.append('n')
    counter[j]+=1
elif dataPrev[j]=='u' and blockDiff>0:
    dataNow.append('u')
```

Fig. 35: Code for finding the mean square difference of luminance between macroblocks

After all the states of a macroblock are computed and adjusted using the states of the neighbouring macroblock, the value of the macroblock with the maximum number of changes in the state is the flickering value of the frame (Fig. 36).

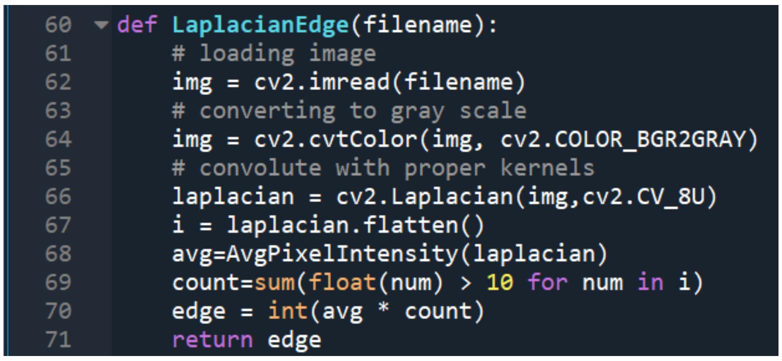


```
dataAllFrames.append(max(dataAllBlocks))
```

Fig. 36: Code for finding the flickering of a frame

5.4.3. <u>Mosquito Noise</u>

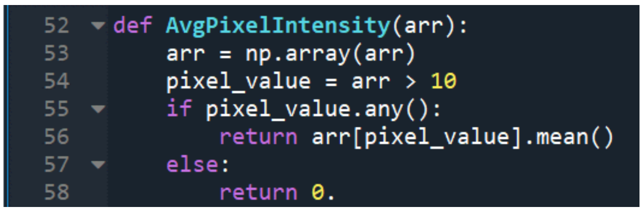
The function LaplacianEdge in Fig. 37 calculates the pixel intensity of the edge pixels in a frame.



```
60  def LaplacianEdge(filename):
61      # loading image
62      img = cv2.imread(filename)
63      # converting to gray scale
64      img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
65      # convolute with proper kernels
66      laplacian = cv2.Laplacian(img,cv2.CV_8U)
67      i = laplacian.flatten()
68      avg=AvgPixelIntensity(laplacian)
69      count=sum(float(num) > 10 for num in i)
70      edge = int(avg * count)
71      return edge
```

Fig. 37: LaplacianEdge function

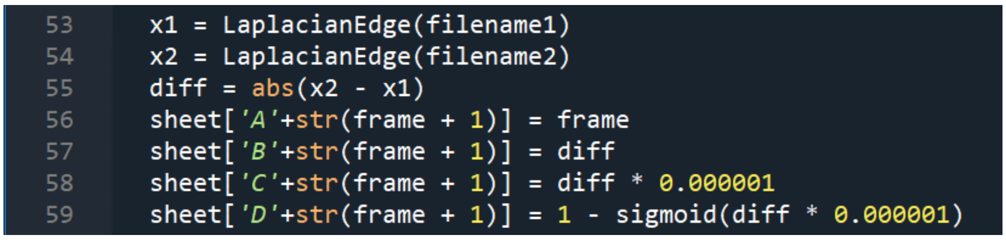
The function Laplacian of OpenCV transforms the input frame *filename* by filtering it using a Laplacian filter as shown in line 66. To find the pixel intensity of the edge pixels, the average pixel intensity *avg* of each pixel is computed by another function AvgPixelIntensity (Fig. 38). As stated in the Methodology, in order to optimise the computation efficiency of the program, only pixels with values higher than 10 are counted as edge pixels. Hence, *count* is the number of pixels with intensities higher than 10. They are considered as the edge pixels. The pixel intensity *edge* of the edge pixels in a frame is the product of *avg* and *count*.



```
def AvgPixelIntensity(arr):
    arr = np.array(arr)
    pixel_value = arr > 10
    if pixel_value.any():
        return arr[pixel_value].mean()
    else:
        return 0.
```

Fig. 38: AvgPixelIntensity function

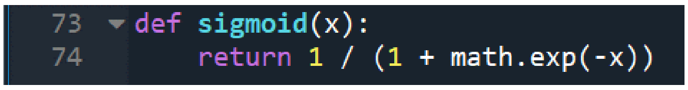
The code to find the mosquito noise of a frame is shown in Fig. 39.



```
    x1 = LaplacianEdge(filename1)
    x2 = LaplacianEdge(filename2)
    diff = abs(x2 - x1)
    sheet['A'+str(frame + 1)] = frame
    sheet['B'+str(frame + 1)] = diff
    sheet['C'+str(frame + 1)] = diff * 0.000001
    sheet['D'+str(frame + 1)] = 1 - sigmoid(diff * 0.000001)
```

Fig. 39: Code for finding the mosquito noise of a frame

Line 55 calculates the absolute difference of the pixel intensity of the edge pixels between consecutive frame *x1* and *x2*. To scale it, it is multiplied by 0.000001 in line 58 before it is converted into a sigmoid probability in line 59. The sigmoid function is written as follows:



```python
def sigmoid(x):
    return 1 / (1 + math.exp(-x))
```

Fig. 40: Code for sigmoid function

5.5. Construction of the LSTM Model

In the Methodology, it is discussed that the program structure to perform the NR-VQA is an LSTM. It is a many-to-one model with three hidden layers with variable hidden nodes in each LSTM cell (Fig. 41). The number of cells in each layer is 180, which equals the maximum number of frames in a video. The number of inputs is also fixed at 180, whereas the only outcome is the MOS corresponding to the video.

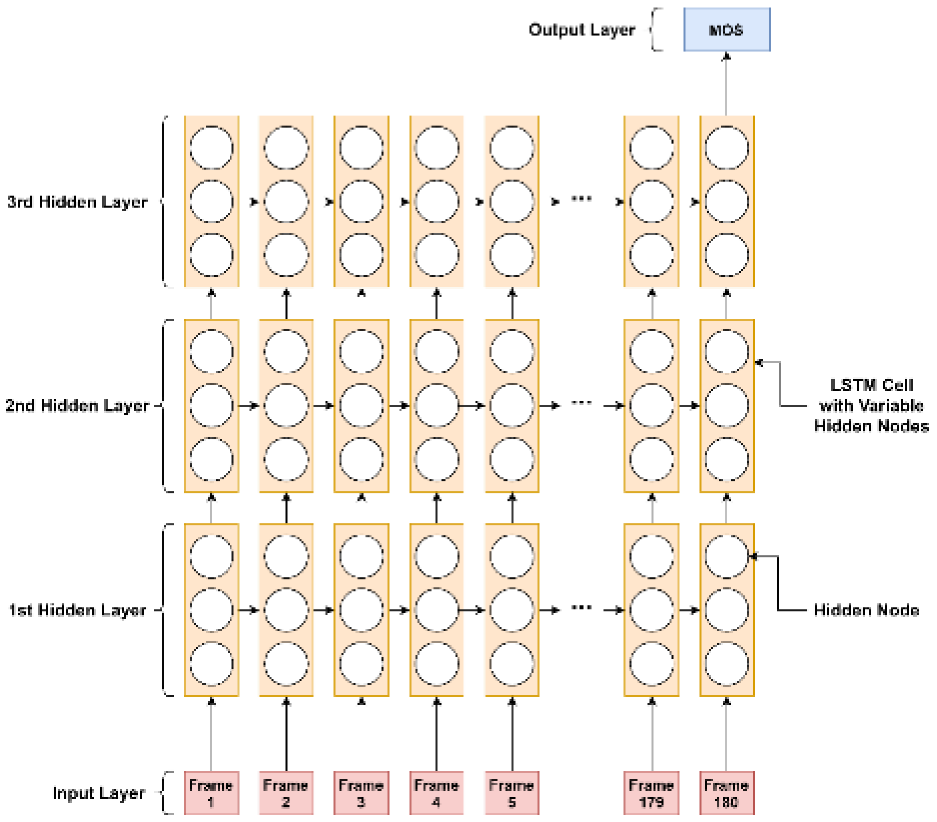
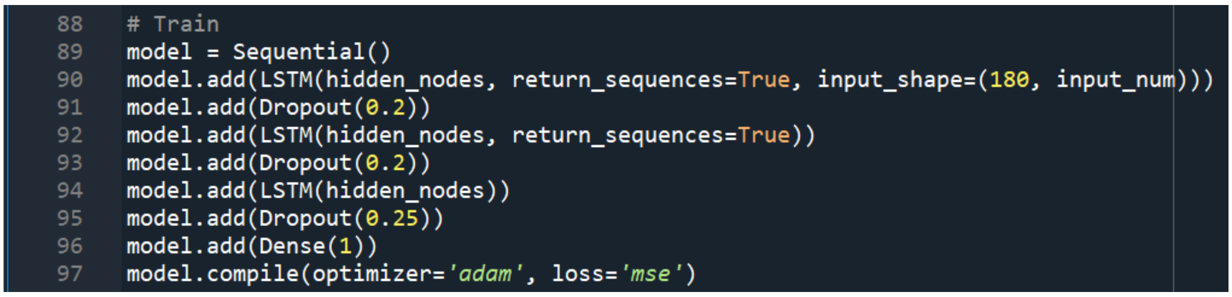


Fig. 41: The LSTM structure for the NR-VQA

The model can be created as a *Sequential* object with some *LSTM* layers added into it (Fig. 42). In practice, *Dropout* layers of 0.2 and 0.25 are also added in between the *LSTM* cells to avoid overfitting. The final output is generated at the *Dense* layer.



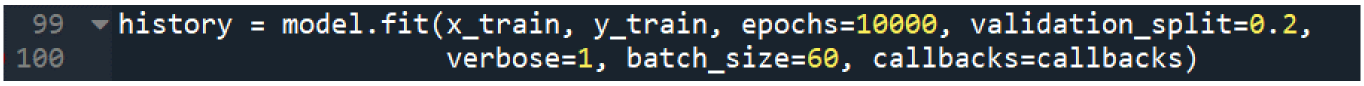
```python
# Train
model = Sequential()
model.add(LSTM(hidden_nodes, return_sequences=True, input_shape=(180, input_num)))
model.add(Dropout(0.2))
model.add(LSTM(hidden_nodes, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(hidden_nodes))
model.add(Dropout(0.25))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

Fig. 42: Code for the LSTM structure for the NR-VQA

5.6. Model Training and Testing

5.6.1. Model Training

The following commands are written to configure the model training:



```python
history = model.fit(x_train, y_train, epochs=10000, validation_split=0.2,
                    verbose=1, batch_size=60, callbacks=callbacks)
```

Fig. 43: Code for the model training

In every model training, the number of epochs is set at 10000 to make sure there are enough trained models to be collected. The batch size is set at 60.

Two callbacks are established for monitoring the model fit and for logging. In Fig. 44, the position pointed by the arrow is the lowest point of the validation loss. At this point, the model is the most well-trained, so the weights used in that attempt will be stored into an HDF file. After this point, the validation loss tends to be increasing, indicating that the models of the later epochs should overfit.

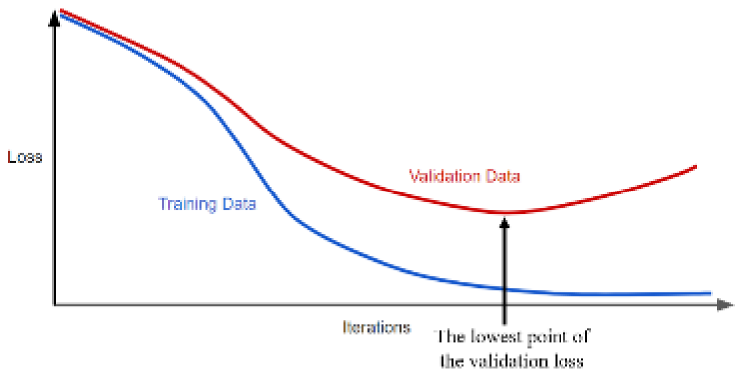
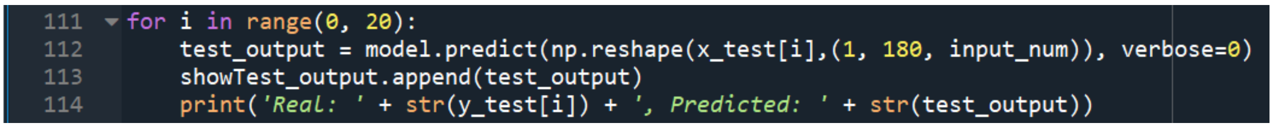


Fig. 44: Loss graph

In addition to the well-trained model, the numbers of hidden nodes of the LSTM cells are altered during each training to see how the model performance change. They include 5, 10, 20, 40, 60, 80 and 100 nodes.

5.6.2. <u>Model Testing</u>

As for testing, the following code runs after the model training to predict the MOS of the testing set. As shown below, 20 videos are arranged as the testing set and their feature inputs are put into the code in line 112 for conducting the prediction.



```
111  for i in range(0, 20):
112      test_output = model.predict(np.reshape(x_test[i],(1, 180, input_num)), verbose=0)
113      showTest_output.append(test_output)
114      print('Real: ' + str(y_test[i]) + ', Predicted: ' + str(test_output))
```

Fig. 45: Code for the model testing

After that, the performance of each model is assessed by the testing data. The feature inputs of the testing set are fed to the model loaded with the saved weights. A comparison between the predicted MOS and the ideal MOS will be found by calculating their mean squared error (MSE) and Pearson correlation coefficient (PCC).

In machine learning, it is common to adopt MSE to reckon the discrepancy between the actual and predicted outcomes. MSE equals to the sum of the squared error divided by the amount of data (22) [26].

$$\mathrm{MSE} = \frac{1}{n} \cdot \sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2 \tag{22}$$

As for PCC, it is a statistical metric to measure the degree of linear correlation r_{XY} between two variables, for instance, X and Y. It is defined by the following equation [26]:

$$r_{XY} = \frac{\mathrm{cov}(X, Y)}{\sigma_X \cdot \sigma_Y} \tag{23}$$

where $cov(X, Y)$ is the covariance (24) between X and Y, σ_X and σ_Y are the standard deviations (the square root of (25)) of X and Y respectively [26].

$$\mathrm{cov}(X, Y) = \frac{1}{n-1} \cdot \sum_{i=1}^{n} (x_i - \bar{x}) \cdot (y_i - \bar{y}) \tag{24}$$

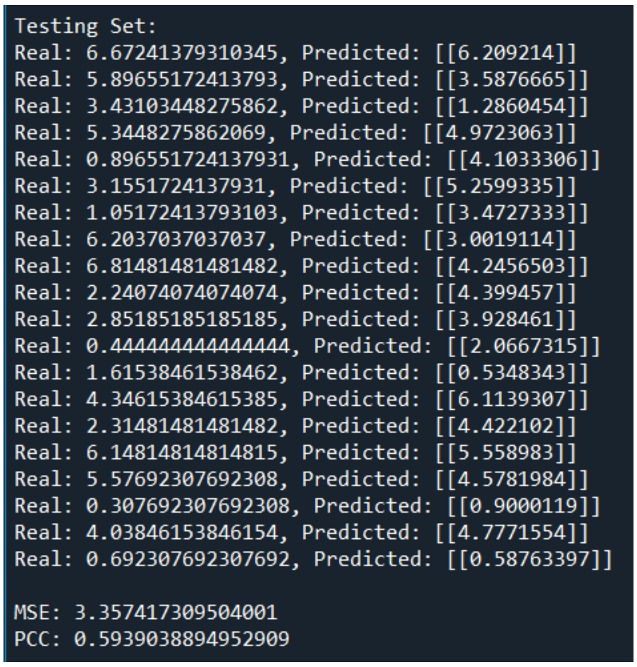
$$\sigma_X^2 = \frac{1}{n} \cdot \sum_{i=1}^{n} (x_i - \bar{x})^2, \quad \sigma_Y^2 = \frac{1}{n} \cdot \sum_{i=1}^{n} (y_i - \bar{y})^2 \tag{25}$$

The degree of linear correlation r_{XY} can be classified into the nine groups below:

Degree Range	**Description of the relationship between *X* and *Y***
$r_{XY} = 1$	Perfectly Positive
$0.8 \leq r_{XY} < 1$	Strongly Positive
$0.3 \leq r_{XY} < 0.8$	Moderately Positive
$0 \leq r_{XY} < 0.3$	Mildly Positive
$r_{XY} = 0$	No Linear Correlation
$-0.3 \leq r_{XY} < 0$	Mildly Negative
$-0.8 \leq r_{XY} < -0.3$	Moderately Negative
$-1 \leq r_{XY} < -0.8$	Strongly Negative
$r_{XY} = -1$	Perfectly Negative

Table 2: Description of PCC degree [26]

In the end, the following will be displayed on the console to show the real and predicted MOS, MSE and PCC of the 20 testing videos (Fig. 46).



```
Testing Set:
Real: 6.67241379310345, Predicted: [[6.209214]]
Real: 5.89655172413793, Predicted: [[3.5876665]]
Real: 3.43103448275862, Predicted: [[1.2860454]]
Real: 5.3448275862069, Predicted: [[4.9723063]]
Real: 0.896551724137931, Predicted: [[4.1033306]]
Real: 3.1551724137931, Predicted: [[5.2599335]]
Real: 1.05172413793103, Predicted: [[3.4727333]]
Real: 6.2037037037037, Predicted: [[3.0019114]]
Real: 6.81481481481482, Predicted: [[4.2456503]]
Real: 2.24074074074074, Predicted: [[4.399457]]
Real: 2.85185185185185, Predicted: [[3.928461]]
Real: 0.444444444444444, Predicted: [[2.0667315]]
Real: 1.61538461538462, Predicted: [[0.5348343]]
Real: 4.34615384615385, Predicted: [[6.1139307]]
Real: 2.31481481481482, Predicted: [[4.422102]]
Real: 6.14814814814815, Predicted: [[5.558983]]
Real: 5.57692307692308, Predicted: [[4.5781984]]
Real: 0.307692307692308, Predicted: [[0.9000119]]
Real: 4.03846153846154, Predicted: [[4.7771554]]
Real: 0.692307692307692, Predicted: [[0.58763397]]

MSE: 3.357417309504001
PCC: 0.5939038894952909
```

Fig. 46: Sample results of the testing set

Among the two metrics, MSE is susceptible to outliers in the dataset [26]. Thus, PCC is used to the overall performance of a trained model, while MSE is kept as a reference to further analyse the results. The model with the highest PCC is considered as the best.

The criteria for separating the videos into the training, the validation and the testing sets are described in the next section.

5.7. <u>Data Preparation and Normalisation</u>

After combining the given and newly extracted artifacts into a data file, the videos can be arranged into three datasets. Around 80% (76 videos, including the validation set) and 20% (20 videos) of the videos are separated into the training and testing sets. Additionally, around 20% (16 videos) of the training set will be the validation set. Fig. 47 exemplifies the data separation of this project.

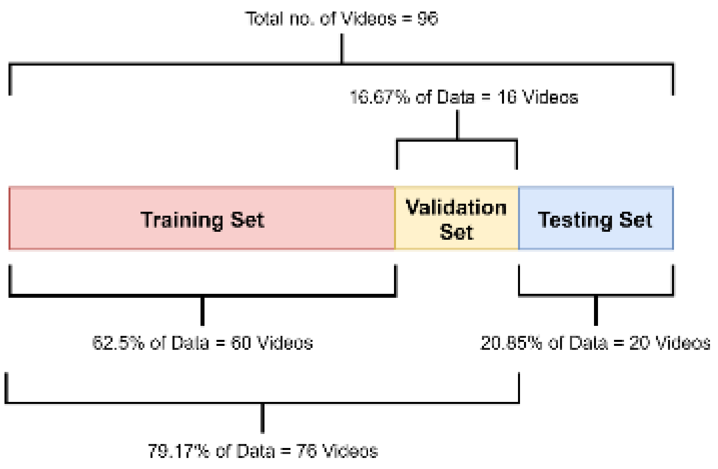

Total no. of Videos = 96
16.67% of Data = 16 Videos
Training Set
Validation Set
Testing Set
62.5% of Data = 60 Videos
20.85% of Data = 20 Videos
79.17% of Data = 76 Videos

Fig. 47: Data separation

To ensure the trained model can make fair and unbiased predictions, the videos are classified according to their MOS before they are categorised into the three datasets.

5.7.1. <u>Classification of the Videos</u>

The videos are initially classified into four classes according to their MOS and the range of their score as follows:

Class	Class Description	Score Range
4	Excellent	$6 \leq MOS \leq 8$
3	Good	$4 \leq MOS < 6$
2	Fair	$2 \leq MOS < 4$
1	Poor	$0 \leq MOS < 2$

Table 3: Categorisation of the videos

Each source video has eight compressed and distorted versions. Therefore, on average, 6 to 7 videos are selected from each of the 12 source videos to form the training and validation sets. They are categorised by their MOS into one of the classes. The rest of the data are the testing data. There is at least one video from each of the 12 source videos is selected to test the model.

By doing the above operation, the training (including the validation set) and testing datasets are created as follows:

Class	No. of Training Videos (training set + validation set)	No. of Testing Videos
4	15	4
3	21	5
2	17	5
1	23	6
Total Number	76	20

Table 4: The number of training (including the validation set) and testing videos in each class

The following table shows the number of training (including the validation set) and testing data according to the source video:

Source Video	No. of Training Videos (training set + validation set)	No. of Testing Videos
Big Buck Bunny	6	2
Birds in Cage	6	2
BQ Terrace	6	2
Crowd Run	7	1
Dance Kiss	7	1
El Fluente A	6	2

El Fluente B	6	2
Fox Bird	6	2
Kimono	6	2
Old Town Cross	7	1
Seeking	6	2
Tennis	7	1
Total Number	76	20

Table 5: The number of training (including the validation set) and testing data according to the source video

5.7.2. Data Standardisation and Scaling

The performance of the model depends on the inputs of the training data. Thus, to ensure the quality of the raw attributes do not affect the unbiasedness of the training, data normalisation should be executed before training.

In this project, the raw data of all videos are normalised by standardisation and scaling. This is done to avoid "biasing toward extreme values" [27] and features with a large range dominating over others [28].

The function StandardScaler of scikit-learn is applied to the raw data to transform them into a set of more normally distributed data as in Fig. 48.

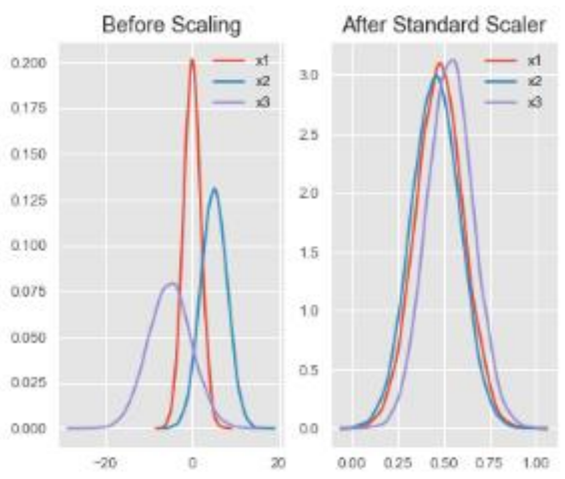


Fig. 48: Transformation of data by StandardScaler [29]

The original data on the left are scaled to their standard scores (z-scores) on the right by the following formula:

$$z = \frac{x - \mu}{s} \tag{26}$$

where μ is the mean and s is the standard deviation of the data [30].

Besides, the standardised data are scaled to a range between 0 to 1 by MinMaxScaler of scikit-learn. Fig. 49 shows an example of the effect of that.

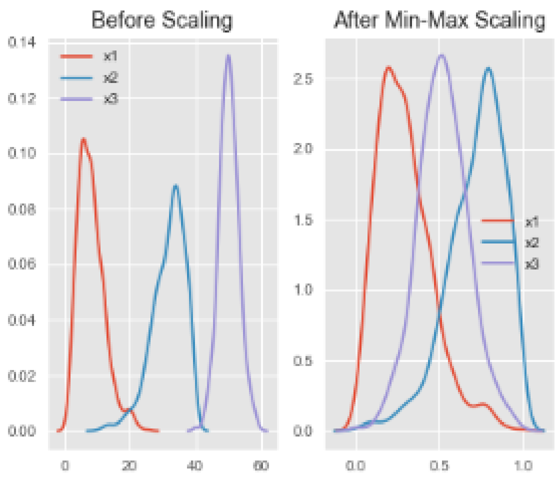


Fig. 49: Transformation of data by MinMaxScaler [29]

The formula of this conversion is represented as follows:

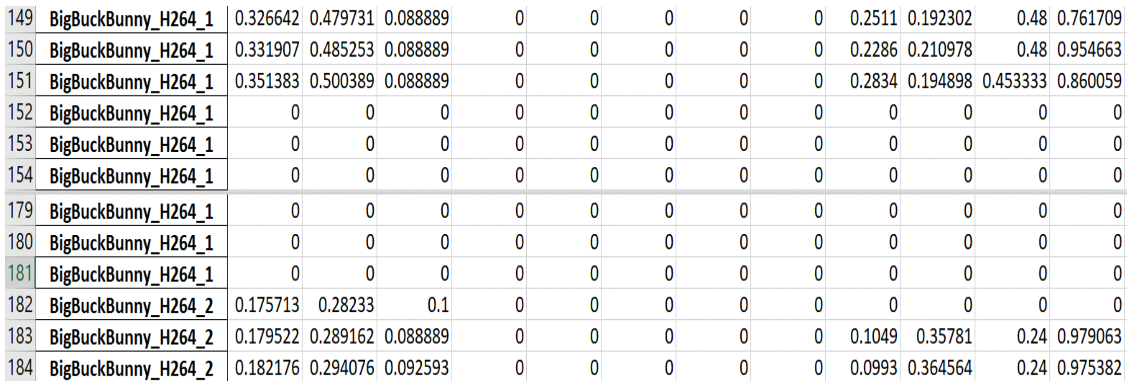
$$m = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{27}$$

where x the original value, x_{max} is the maximum value and x_{min} is the minimum value of the range [31]. As the data range will be within 0 and 1, x_{max} is 1 and x_{min} is 0.

5.7.3. Zero Padding of Data

The input sequence for both training and testing will be fixed at 180. As the frame number of the videos are not the same, the inputs of some videos will be padded with zeros after the last frame. The rows of the data matrices will then be extended to 180.

Take the data of BigBuckBunny_H264_1 with 150 frames as an instance in Fig. 50. After the data in row 151 (frame 150), zeros are padded until row 181. While the input sequence is being fed to the model, the input values after frame 150 are zeros (Fig. 51).



149	BigBuckBunny_H264_1	0.326642	0.479731	0.088889	0	0	0	0	0	0.2511	0.192302	0.48	0.761709
150	BigBuckBunny_H264_1	0.331907	0.485253	0.088889	0	0	0	0	0	0.2286	0.210978	0.48	0.954663
151	BigBuckBunny_H264_1	0.351383	0.500389	0.088889	0	0	0	0	0	0.2834	0.194898	0.453333	0.860059
152	BigBuckBunny_H264_1	0	0	0	0	0	0	0	0	0	0	0	0
153	BigBuckBunny_H264_1	0	0	0	0	0	0	0	0	0	0	0	0
154	BigBuckBunny_H264_1	0	0	0	0	0	0	0	0	0	0	0	0
179	BigBuckBunny_H264_1	0	0	0	0	0	0	0	0	0	0	0	0
180	BigBuckBunny_H264_1	0	0	0	0	0	0	0	0	0	0	0	0
181	BigBuckBunny_H264_1	0	0	0	0	0	0	0	0	0	0	0	0
182	BigBuckBunny_H264_2	0.175713	0.28233	0.1	0	0	0	0	0	0	0	0	0
183	BigBuckBunny_H264_2	0.179522	0.289162	0.088889	0	0	0	0	0	0.1049	0.35781	0.24	0.979063
184	BigBuckBunny_H264_2	0.182176	0.294076	0.092593	0	0	0	0	0	0.0993	0.364564	0.24	0.975382

Fig. 50: Zero padding of the input data of a video

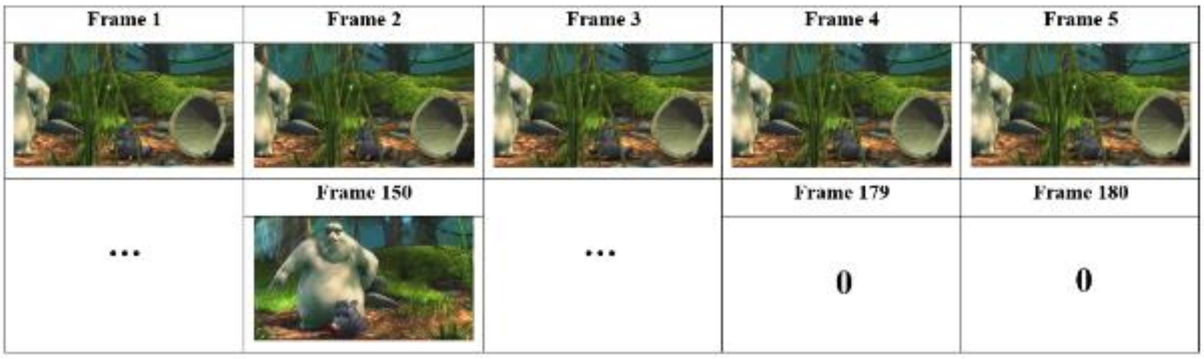


Fig. 51: Input sequence of a video padded with zeros

5.8. <u>Configuration of PCA</u>

The number of features for training can be determined by plotting PCA decomposition graph (Fig. 52).

No. of PCs	**0**	**1**	**2**	**3**	**4**	**5**	**6**
Cumulative Explained Variance	0	0.481358	0.752338	0.847028	0.901388	0.936556	0.954687
No. of PCs	**7**	**8**	**9**	**10**	**11**	**12**	
Cumulative Explained Variance	0.971933	0.983884	0.994436	~1	~1	1	

Table 6: Cumulative explained variance of different number of principal components

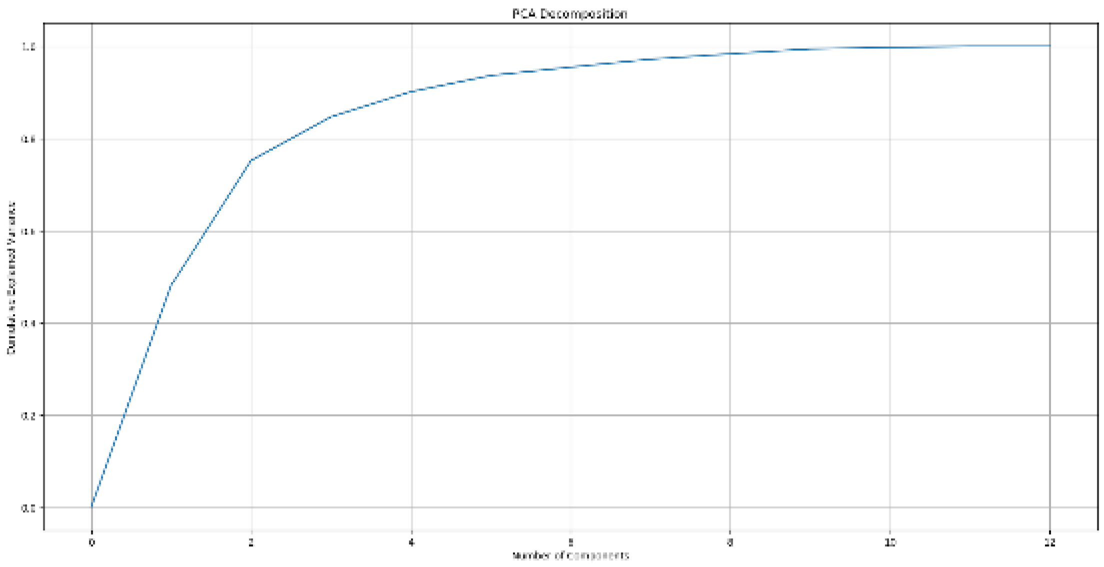
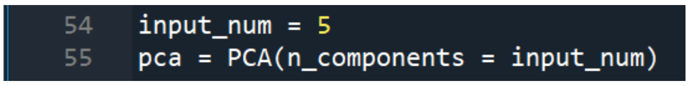


Fig. 52: PCA decomposition graph for MOS prediction

The features of higher variance have a higher priority to be chosen and saved as PCs by PCA. In this project, the number of PCs chosen will have the cumulative explained variance above 0.9 but low than 1 (Table 6). In this case, 4 to 9 features of higher explained variance are selected as PCs for training.

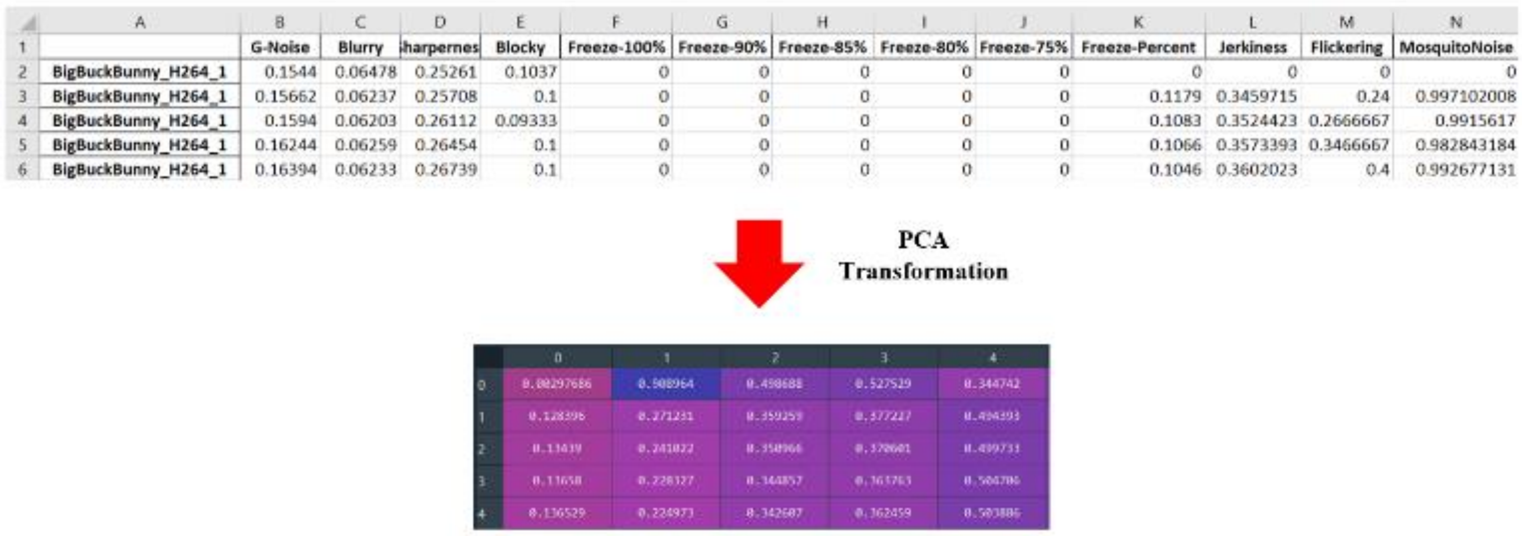
To configure PCA, the threshold *n_components* of the scikit-learn PCA function must be set in the program. An example in Fig. 53 has *n_components* set at 5, meaning that there are 5 PCs.



```
input_num = 5
pca = PCA(n_components = input_num)
```

Fig. 53: Code for configuring PCA

It is noted that one may not be able to figure out which features have higher variance by just observing the PCA translated and reduced inputs as their values are all converted and sorted according to the variance of each feature in a descending manner (Fig. 54). But as explained in the Methodology, this does not affect their distribution.


PCA
Transformation

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1		G-Noise	Blurry	Sharpernes	Blocky	Freeze-100%	Freeze-90%	Freeze-85%	Freeze-80%	Freeze-75%	Freeze-Percent	Jerkiness	Flickering	MosquitoNoise
2	BigBuckBunny_H264_1	0.1544	0.06478	0.25261	0.1037	0	0	0	0	0	0	0	0	0
3	BigBuckBunny_H264_1	0.15662	0.06237	0.25708	0.1	0	0	0	0	0	0.1179	0.3459715	0.24	0.997102008
4	BigBuckBunny_H264_1	0.1594	0.06203	0.26112	0.09333	0	0	0	0	0	0.1083	0.3524423	0.2666667	0.9915617
5	BigBuckBunny_H264_1	0.16244	0.06259	0.26454	0.1	0	0	0	0	0	0.1066	0.3573393	0.3466667	0.982843184
6	BigBuckBunny_H264_1	0.16394	0.06233	0.26739	0.1	0	0	0	0	0	0.1046	0.3602023	0.4	0.992677131

	0	1	2	3	4
0	0.00297686	0.908964	0.498688	0.527529	0.344742
1	0.128396	0.271231	0.359259	0.377227	0.494393
2	0.13439	0.241022	0.350966	0.370601	0.499731
3	0.11658	0.228127	0.344857	0.363763	0.504706
4	0.136529	0.224973	0.342607	0.362459	0.503886

Fig. 54: Comparison of the first five rows between the original and PCA transformed data (5 PCs)

After the transformation, the data are scaled to the range between 0 to 1 using MinMaxScaler so that their values will be in the same range.

6. Results

6.1. Initial Training

In the initial training, all 12 features (Gaussian noise, blocking, sharpness, frame freeze and its 5-class one-hot encodings freeze percentages, jerkiness, flickering and mosquito noise are the inputs. The results of the training of 5 to 100 hidden nodes and the lowest validation loss are summarised in the following table:

No. of Nodes	Epoch	Training Set Loss	Validation Loss	Training + Validation Set Loss	Training + Validation Set PCC	Testing Set Loss	Testing Set PCC
5	1673	2.741584	3.935477	3.01287911	0.62777812	5.5985641	0.2664208
10	2854	3.983636	3.0047352	4.65105598	0.32471066	5.2310608	0.1994951
20	7674	2.42131	2.3657331	1.88778679	0.78832487	4.2779945	0.3950053
40	7117	1.771276	2.5645134	1.6274234	0.82284663	4.3302755	0.4422262
60	7500	1.649109	3.0519562	1.99414588	0.77433808	3.3574173	0.5939039
80	682	1.892916	1.8686432	1.71021931	0.80897045	5.5367048	0.3285236
100	561	3.655327	1.6261252	3.65291193	0.55552143	5.6799842	0.2987146

Table 7: Results of the model with 12 features

From the table, the best model is of 60 hidden nodes at 7500th epoch, with a testing set loss (MSE) and PCC of 3.3574 and 0.5939 respectively. The real and predicted data are considered to have a moderately positive linear correlation regarding Table 2.

6.2. Training with PCA Feature Selection

After applying PCA, the results of the trained models with 4 to 9 PCs are as follows:

4 PCs							
No. of Nodes	**Epoch**	**Training Set Loss**	**Validation Loss**	**Training + Validation Set Loss**	**Training + Validation Set PCC**	**Testing Set Loss**	**Testing Set PCC**
5	9715	4.166239	3.400902	3.5051326	0.54012429	4.224024	0.373458
10	5151	1.997271	2.6514206	2.1060105	0.76253268	6.965065	0.168808
20	4093	0.569398	1.8382915	0.574867	0.94134256	3.372454	0.597752
40	1282	3.151057	2.1675835	2.2519619	0.74964276	5.024254	0.360167
60	5359	2.701427	1.6260335	2.2649175	0.74455057	4.548734	0.456537
80	6800	0.122605	2.7611413	0.6300382	0.934298	3.813198	0.61044
100	6504	2.097037	2.7458451	3.0175515	0.63756217	3.509179	0.560113

5 PCs							
No. of Nodes	**Epoch**	**Training Set Loss**	**Validation Loss**	**Training + Validation Set Loss**	**Training + Validation Set PCC**	**Testing Set Loss**	**Testing Set PCC**
5	8199	4.269392	3.6822095	3.9192433	0.50499811	5.532576	0.076423
10	752	3.95692	3.8674209	3.6964938	0.50589508	5.363934	0.117387
20	8050	3.890567	2.6506689	5.6893431	0.50028902	9.252919	0.247466
40	3578	0.704483	1.2033491	0.7134453	0.9261184	2.046262	0.763572
60	9769	0.49468	2.680995	1.5922485	0.84280097	8.522534	0.060594
80	8610	4.664323	2.2359941	3.7547756	0.58546598	5.186428	0.438695
100	6400	0.717816	1.5645299	0.9286591	0.90593607	4.656727	0.43097
6 PCs							
No. of Nodes	**Epoch**	**Training Set Loss**	**Validation Loss**	**Training + Validation Set Loss**	**Training + Validation Set PCC**	**Testing Set Loss**	**Testing Set PCC**
5	9799	1.354513	1.7366378	1.387613	0.85078307	4.120991	0.541044
10	3601	7.529644	1.6703783	4.3571918	0.48907231	5.75778	0.224716
20	4857	0.391896	0.696534	0.2981275	0.97027783	1.612383	0.847862
40	9087	1.401437	3.026	1.1616026	0.89466503	3.731401	0.693768
60	5455	2.658505	1.2178605	2.6193994	0.7109677	5.274808	0.423935
80	1589	2.383781	1.7513819	2.5090244	0.70846896	5.072689	0.29285
100	2270	2.132776	1.2502849	1.4706029	0.84009059	2.756523	0.663903
7 PCs							
No. of Nodes	**Epoch**	**Training Set Loss**	**Validation Loss**	**Training + Validation Set Loss**	**Training + Validation Set PCC**	**Testing Set Loss**	**Testing Set PCC**
5	3000	3.201858	2.3248992	2.7458639	0.67986687	3.794456	0.47756
10	2483	1.597168	0.6613392	1.0067629	0.8974883	2.218175	0.751818
20	6488	2.366874	2.1098533	2.2809416	0.77206936	6.251809	0.218656
40	1912	2.371829	2.5519032	2.0232282	0.77095464	6.145922	0.052162
60	384	2.970089	2.4446263	2.8280541	0.67119703	3.736024	0.490482
80	2581	0.156316	1.1035175	0.2975875	0.97121369	1.969978	0.776061
100	9874	2.809896	1.6157116	1.9494219	0.78192491	5.458905	0.215694
8 PCs							
No. of Nodes	**Epoch**	**Training Set Loss**	**Validation Loss**	**Training + Validation Set Loss**	**Training + Validation Set PCC**	**Testing Set Loss**	**Testing Set PCC**
5	8126	2.548157	2.7155046	1.9646371	0.78689766	5.929026	0.204466
10	8488	0.521883	1.1048646	0.7205358	0.9422601	2.79894	0.692736
20	2629	2.527728	2.431181	2.3482486	0.72478277	4.530408	0.416876

40	8278	1.309358	1.3469522	1.6208131	0.84491412	2.842563	0.726029
60	5761	3.137047	1.6146007	3.1345901	0.66216885	7.789882	0.033569
80	8277	1.644552	2.297365	2.9980576	0.71212357	4.253603	0.577042
100	7923	1.728506	1.646989	1.4746883	0.83908663	3.425941	0.542703
9 PCs							
No. of Nodes	**Epoch**	**Training Set Loss**	**Validation Loss**	**Training + Validation Set Loss**	**Training + Validation Set PCC**	**Testing Set Loss**	**Testing Set PCC**
5	7055	2.592423	1.8937253	2.1005888	0.7654202	4.787138	0.355208
10	6099	1.667865	1.1802887	1.1599396	0.88012754	2.069208	0.766801
20	2588	2.739038	2.0138617	2.5525715	0.71127697	3.816189	0.479186
40	7535	0.400198	0.9468867	0.3054905	0.96869351	2.205659	0.776019
60	2422	3.437068	2.1397066	3.1106936	0.62947866	3.515645	0.564504
80	1499	5.180573	2.133523	4.652744	0.35318753	6.152397	-0.02676
100	772	2.809245	1.5665543	2.473373	0.70724718	3.680933	0.497005

Table 8: Results of the models with PCA feature selection

From Table 8, the best model is of 6 PCs with 20 hidden nodes at 4857th epoch. The testing set loss (MSE) and PCC is 1.6124 and 0.8479, respectively. The real and predicted data are said to have a strong positive linear correlation regarding Table 2.

7. Evaluation

In the initial training, the model of 60 hidden nodes has a PCC of 0.5939, expressing it produces moderately positive correlated data. Nevertheless, the PCC of other models are below than 0.5, and their outcomes are either moderately or mildly positively correlated. In general, the model performs mediocrely with 12 features regardless of the number of hidden nodes.

The curse of dimensionality causes undistinguished results. As explained in the Methodology, the input dimension is too high for the small dataset. In this project, a video input includes three spatial (Gaussian noise, blocking and sharpness) and nine temporal (frame freeze and its 5-class one-hot encoding, jerkiness, flickering and mosquito noise) data. The dimension of the input is 12 columns * 180 rows (some videos with less than 180 frames will have zeros padded in their input) = 2160. As for the outputs, there are only 96 MOS referenced to the 96 videos. Consequently, if all 12 features are used in training, the trained models will be subject to overfitting due to the curse of dimensionality, leading to a low PCC and a relatively high MSE in the testing set.

After carrying out the PCA feature selection, the input dimension is reduced. The model performance is then significantly enhanced, with the PCC improves from 0.5939 to 0.8479. As seen in the loss graphs of both models, the model with 12 features (Fig. 55) overfits at an earlier epoch than that with 6 PCs (Fig. 56).

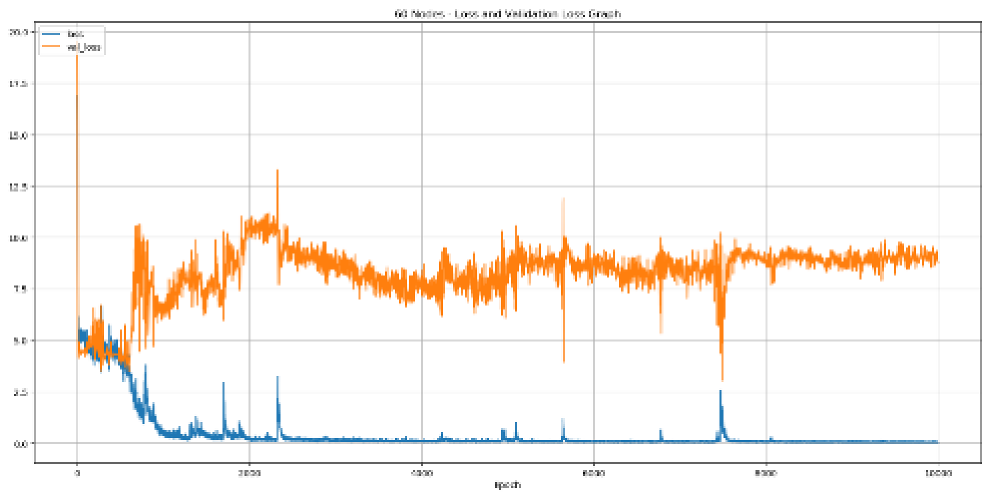


Fig. 55: Loss Graph of the model of 60 hidden nodes with 12 features

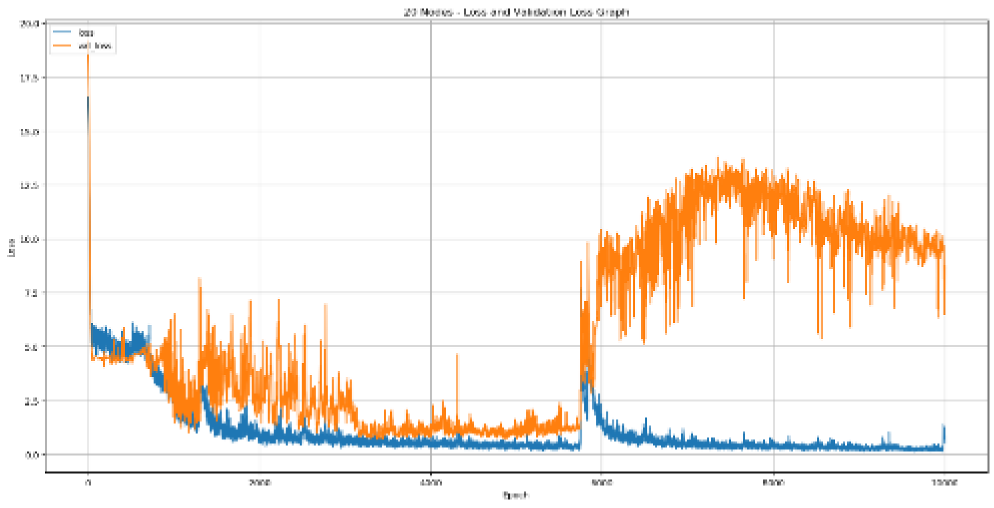


Fig. 56: Loss Graph of the model of 20 hidden nodes with 6 PCs

On the report of Lin et al. [7], some FR-VQA algorithms have been used to test on the collected MOS of the database MCL-V. Their performances are compared with those of the two proposed models in this project (Table 9).

Model / Metric Name	**PCC**
Proposed LSTM NR-VQA with PCA Feature Selection	**0.8479**
FSIM [7], [32]	0.7500
VADM [7], [33]	0.7420
GSM [7], [34]	0.7090
S-MAD [7], [35]	0.6810
VIF [7], [36]	0.6600
GMSD [7], [37]	0.6500
SSIM [7], [38]	0.6500
ST-MAD [7], [35]	0.6340
MSSIM [7], [39]	0.6210
T-MAD [7], [35]	0.6000
Proposed LSTM NR-VQA without PCA Feature Selection	**0.5939**
PSNR [7]	0.4720

Table 9: Comparison of the model performances between the proposed and FR-VQA models

The proposed LSTM NR-VQA with PCA feature selection in this project outperforms the metrics for FR-VQA presented in Table 9. The high PCC of the proposed model evidences its effectiveness in assessing video quality objectively as to control and optimise QoE.

8. Conclusion

This project suggests an LSTM NR-VQA model for evaluating video quality. The database has 96 distorted and compressed videos. Three types of temporal impairments, namely jerkiness, flickering and mosquito noise, are extracted from each video frames. A previous LSTM NR-VQA experiment also provides three spatial features: Gaussian noise, blocking and sharpness, and six temporal features: frame freeze and its five one-hot encodings freeze percentage. The new collected temporal artifacts and the given data are combined to undergo analysis conducting by the proposed model. The final predicted outcomes are MOS of the testing set.

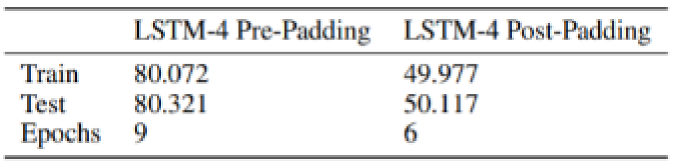
The initial prediction with 12 features has a PCC of 0.5939 achieved by the model of 60 hidden nodes. The result is not satisfactory due to the curse of dimensionality, which the dimension of the original inputs is higher than the number of sample data. After reducing the number of features from 12 to 6 using PCA, the PCC notably raises to 0.8479. As a result, PCA feature selection can enhance the model performance by dropping some unimportant features.

By comparing the suggested LSTM NR-VQA model with PCA feature selection with some FR-VQA algorithms, the model in this project performs more decently in MOS prediction. This confirms that the newly extracted temporal features are some of the keys to determine video quality. On the other hand, it shows that the model is competent to estimate video quality by examining the artifacts and predict MOS. Overall, the proposed model is a valid substitution of subjective VQA and FR-VQA as it can effectively simulate the human perception towards video quality.

9. Further Developments

Though the project has achieved a satisfactory result, some improvements can be made to heighten the model capability.

One of the approaches is to pre-pad zeros into the input matrix. Suggested by D. M. Reddy and N. V. S. Reddy [40], it is said that how the zeros are padded into inputs, obviously affects the efficiency of LSTM. Say an experiment is conducted to classify the attitude expressed in tweets. Two datasets are created. One of them has an input matrix with pre-padded zeros, while another has a post-padded one. By investigating the data using LSTM to categorise the tweets, the results are as follows:



	LSTM-4 Pre-Padding	LSTM-4 Post-Padding
Train	80.072	49.977
Test	80.321	50.117
Epochs	9	6

Table 10: Accuracies of the tweet classification using zero pre-padded and post-padded inputs [40]

As revealed in Table 10, the model with zero pre-padded inputs generates fairer results by giving an accuracy of around 80% of the testing set. The one with zero post-padded inputs, however, has a subpar achievement with merely 50% of testing accuracy. Therefore, in order to boost the model capability, the input matrix should be pre-padded instead of post-padded.

References

[1] F. M. Moss, C.-T. Yeh, F. Zhang, R. Baddeley and D. R. Bull, “Support for reduced presentation durations in subjective video quality assessment,” *Signal Processing: Image Communications,* vol. 48, pp. 38-49, Aug. 2016.

[2] Z. Wang, H. R. Sheikh and A. C. Bovik, “Objective Video Quality Assessment,” in *The Handbook of Video Databases: Design and Applications*, B. Furht and O. Marqure, Eds., Boca Raton, CRC Press, 2003, pp. 1041-1078.

[3] P. M. Arun Kumar and S. Chandramathi, “Video Quality Assessment Methods: A Bird' s-Eye View,” *International Journal of Computer and Information Engineering*, vol. 8, no. 5, pp. 772-778, Apr. 2014.

[4] B. Ortiz-Jaramillo, J. Niño-Castañeda, L. Platiša and W. Philips, "Content-aware objective video quality assessment," *Journal of Electronic Imaging*, vol. 25, no. 1, pp. 1-16, Feb. 2016.

[5] M. Shahid, A. Rossholm, B. Lövström and H.-J. Zepernick, "No-reference image and video quality assessment: a classification and review of recent approaches," *EURASIP Journal on Image and Video Processing*, vol. 2014, no. 1, pp. 1-32, Aug. 2014.

[6] N. W. Kwong, "Video Quality Assessment based on Machine Learning," B.Eng thesis, Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, Hong Kong, 2019.

[7] J. Y. Lin, R. Song, C.-H. Wu, T. Liu, H. Wang and C.-C. J. Kuo, "MCL-V: A streaming video quality assessment database," *Journal of Visual Communication and Image Representation*, vol. 30, no. 1, pp. 1-9, Jul. 2015.

[8] J. Urban, "Compression Artifacts: Why Video Looks “Bad”," *AVTechnology*, Nov. 16, 2017. [Online]. Available: https://www.avnetwork.com/avtechnology/compression-artifacts-why-video-looks-bad. [Accessed Dec. 28, 2019].

[9] S. Borer, "A Model of Jerkiness for Temporal Impairments in Video Transmission," presented at 2010 Second International Workshop on Quality of Multimedia Experience (QoMEX). [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5516155 [Accessed Dec. 28, 2019].

[10] G. Iacovoni, S. Morsa, and R. Felice, "Quality-Temporal Transcoder Driven by the Jerkiness," presented at 2005 IEEE International Conference on Multimedia and Expo. [Online]. Available: https://ieeexplore.ieee.org/document/1521705 [Accessed Dec. 28, 2019].

[11] C. McIntosh, Ed., *Cambridge Advanced Learner's Dictionary*, Cambridge: Cambridge University Press, 2013.

[12] C. R. Anil, and D. B. R. Krishna, "H.264 Video Coding Artifacts Measurement and Reduction of Flickering," M.S. thesis, Department of Electrical Engineering, Blekinge Institute of Technology, Karlskrona, 2014. [Online]. Available: https://www.semanticscholar.org/paper/H.264-VIDEO-CODING-ARTIFACTS-%3A-MEASUREMENT-AND-OF-Chodisetti-Dasari/792aa99f54a1143b3f50ca65860b089b6b89a7a8 [Accessed Dec. 28, 2019].

[13] guy jones, *1818 to 1890s Bicycle Models (from 1915 documentary)*, 4 May 2017. [Video file]. Available: https://www.youtube.com/watch?v=WI6lgUlEUFU. [Accessed Dec. 28, 2019].

[14] J. Pandel, "Measuring of Flickering Artifacts in Predictive Coded Video Sequences," presented at 2008 Ninth International Workshop on Image Analysis for Multimedia Interactive Services. [Online]. Available: https://ieeexplore.ieee.org/document/4556926 [Accessed Dec. 28, 2019].

[15] S. Winkler, *Digital Video Quality: Vision Models and Metric*s, Chichester: John Wiley & Sons Ltd, 2005.

[16] C. Mantel, P. Ladret and T. Kunlin, "A temporal mosquito noise corrector," presented at 2009 International Workshop on Quality of Multimedia Experience. [Online]. Available: https://ieeexplore.ieee.org/document/5246943 [Accessed May 1, 2020].

[17] A. Punchihewa and A. Keerl, "Design of a Synthetic Test Sequence and a Metric for Evaluation of Mosquito Noise Due to Video Codecs," presented at 2011 Sixth IEEE International Symposium on Electronic Design, Test and Application. [Online]. Available: https://ieeexplore.ieee.org/document/5729537 [Accessed May 1, 2020].

[18] OpenCV, "Laplace Operator," ca. 2014. [Photograph]. Available: https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/laplace_operator/laplace_operator.html. [Accessed Apr. 28, 2020].

[19] C. C. Aggarwal, *Neural Networks and Deep Learning : A Textbook*, Cham: Springer, 2018.

[20] C. Olah, "Understanding LSTM Networks," 27 Aug. 2015. [Photograph]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed Dec. 28 2019].

[21] R. Vidal, Y. Ma and S. S. Sastry, *Generalized Principal Component Analysis*, New York: Springer, 2016.

[22] P. Milewski, "PCA decomposition and Keras neural network," *Kaggle*, ca. 2017. [Online]. Available: https://www.kaggle.com/pmmilewski/pca-decomposition-and-keras-neural-network. [Accessed May 1, 2020].

[23] S. Guillaume and F. Ros, Eds., *Sampling Techniques for Supervised or Unsupervised Tasks*, Cham, 2020.

[24] H. Lu, K. N. Plataniotis and A. N. Venetsanopoulos, *Multilinear Subspace Learning: Dimensionality Reduction of Multidimensional Data*, Boca Raton: CRC Press, 2014.

[25] DeepAI, "Curse of Dimensionality," *DeepAI*, n.d. [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/curse-of-dimensionality. [Accessed May 1, 2020].

[26] V. A. Profillidis and G. N. Botzoris, *Modeling of Transport demand: Analyzing, Calculating, and Forecasting Transport Demand*, Cambridge: Elsevier, 2019.

[27] D. Nettleton, *Commercial Data Mining: Processing, Analysis and Modeling for Predictive Analytics Projects*, Amsterdam: Morgan Kaufmann, 2014.

[28] S. García, J. Luengo and F. Herrera, *Data Preprocessing in Data Mining*, Cham: Springer, 2015.

[29] I. Dzindo, "Feature Scaling in Python," May 23, 2018. [Photograph]. Available: https://medium.com/@ian.dzindo01/feature-scaling-in-python-a59cc72147c1. [Accessed May 1, 2020].

[30] scikit-learn, "StandardScaler," *scikit-learn*, ca. 2019. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html. [Accessed May 1, 2020].

[31] scikit-learn, "MinMaxScaler," *scikit-learn*, ca. 2019. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html. [Accessed May 1, 2020].

[32] L. Zhang, L. Zhang, X. Mou and D. Zhang, "FSIM: A Feature Similarity Index for Image Quality Assessment," *IEEE Transactions on Image Processing*, vol. 20, no. 8, pp. 2378-2386, 2011.

[33] S. Li, L. Ma and K. N. Ngan, "Full-Reference Video Quality Assessment by Decoupling Detail Losses and Additive Impairments," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 7, pp. 1100-1112, 2012.

[34] A. Liu, W. Lin and M. Narwaria, "Image Quality Assessment Based on Gradient Similarity," *IEEE Transactions on Image Processing*, vol. 21, no. 4, pp. 1500-1512, 2012.

[35] P. V. Vu , C. T. Vu and D. M. Chandler, "A spatiotemporal most-apparent-distortion model for video quality assessment," presented at 2011 18th IEEE International Conference on Image Processing. [Online]. Available: https://ieeexplore.ieee.org/document/6116171 [Accessed May 1, 2020].

[36] H. R. Sheikh and A. C. Bovik, "Image information and visual quality," *IEEE Transactions on Image Processing*, vol. 15, no. 2, pp. 430-444, 2006.

[37] W. Xue, L. Zhang, X. Mou and A. C. Bovik, "Gradient Magnitude Similarity Deviation: A Highly Efficient Perceptual Image Quality Index," *IEEE Transactions on Image Processing*, vol. 23, no. 2, pp. 684-695, 2014.

[38] Z. Wang , A. C. Bovik , H. R. Sheikh and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600-612, 2004.

[39] Z. Wang , E. P. Simoncelli and A. C. Bovik, "Multiscale structural similarity for image quality assessment," presented at The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers. [Online]. Available: https://ieeexplore.ieee.org/document/1292216 [Accessed May 1, 2020].

[40] D. M. Reddy and N. V. S. Reddy, *Effects of Padding LSTMs and CNNs*, 2019. [Online]. Available: https://arxiv.org/abs/1903.07288. [Accessed May 1, 2020].