Pontificia Universidad Javeriana Cali
School of Engineering
Department of Electronics and Computer Science
Degree project.

Implementation in C++ of the NSS-based features for video quality evaluator RAPIQUE

Stidl Alfonso Torres Morón

Director:
Electrical Engineering (Ph.D.) Carlos Alberto Salazar Herrera
Co-Director:
Electrical Engineering (Ph.D.) Hernan Dario Benitez Restrepo

May 17, 2024

Resumen

La creciente prevalencia de contenido de video distorsionado o de baja calidad generado por usuarios en plataformas digitales ha hecho que la evaluación eficiente y precisa de la calidad del video sea crítica. RAPIQUE, un modelo para tal evaluación, aunque preciso, sufre de tiempos de ejecución prolongados en su implementación actual en Matlab. Estas limitaciones lo hacen inadecuado para aplicaciones de baja latencia. El trabajo de grado propone optimizar RAPIQUE re-implementándolo en C++ y aprovechando bibliotecas como OpenCV para mejorar la extracción de características basadas en estadísticas de escenas naturales (NSS). Esta optimización no solo mejora los tiempos de ejecución sino que también permita una integración más fluida con herramientas de software existentes, mejorando así los tiempos de ejecución. El proyecto tiene implicaciones económicas y sociales significativas, incluyendo un flujo de trabajo más rápido para investigadores y empresas de medios, una mejor experiencia de usuario en plataformas de medios digitales, y una reducción en el consumo de energía en centros de datos.El repositorio con los algoritmos y resultados está disponible en https://github.com/StidlTorres1/RAPIQUE-C-Version--VideoQualityAssessment.

Palabras Clave Evaluación de calidad de vídeo no referenciado (BVQA), redes neuronales convolucionales (CNN), estadísticas de escena natural (NSS),unidad central de procesamiento (CPU), contenido generado por el usuario (UGC), característica espacio-temporal,unidad de procesamiento gráfico (GPU).

Abstract

The increasing prevalence of distorted or low-quality user-generated video content on digital platforms has made an efficient and accurate assessment of video quality critical. Although correct, RAPIQUE, a model for such evaluation, needs long run times in its current Matlab implementation. These limitations make it unsuitable for pseudo-real-time applications. The degree work proposes to optimize RAPIQUE by re-implementing it in C++ and taking advantage of libraries such as OpenCV to improve feature extraction focused on natural scene statistics (NSS). This optimization enhances runtimes and allow smoother integration with existing software tools, thus improving execution times. The project has significant economic and social implications, including faster workflow for researchers and media companies, improved user experience on digital media platforms, and reduced energy consumption in data centers. The repository with the algorithms and results is available at https://github.com/StidlTorres1/RAPIQUE-C-Version--VideoQualityAssessment.

Blind video quality (BVQA), Convolutional neural networks (CNN), natural scene statistics (NSS),Central Processing Unit (CPU), user-generated content (UGC), spatial-temporal feature, Graphics processing unit (GPU).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

In the digital age, user-generated video content (UGC) with distortions that degrade video quality has become ubiquitous. Major platforms such as YouTube, Facebook, and TikTok have significantly accelerated the production and dissemination of video content, necessitating efficient and robust tools for video quality assessment (VQA) (Wang et al., 2004a). RAPIQUE (Rapid and Accurate Video Quality Prediction of User-Generated Content) has emerged as a promising solution for non-referenced VQA, offering competitive evaluation accuracy (Mittal et al., 2012a; Tu et al., 2021b). However, UGC videos, often created by amateur videographers, frequently suffer from poor perceptual quality due to imperfect capture devices, uncertain shooting skills, various content processes, as well as compression and streaming distortions. Traditional VQA models have been widely studied and used by the streaming and social media industries. While full-reference VQA research is maturing with several widely deployed algorithms, recent focus has shifted towards developing better no-reference VQA models for UGC videos. These UGC videos often already suffer from artifacts, making it difficult to determine optimal compression settings and requiring accurate and efficient no-reference or blind video quality models for optimization.

Despite its potential, RAPIQUE's current implementation in Matlab presents significant limitations. Notably, it suffers from long execution times, rendering it unsuitable for pseudo-real-time applications (Tu et al., 2021b). This discrepancy has led to increasing challenges for its applicability in a fast-paced digital environment demanding pseudo-real-time solutions.

This document outlines the problem, objectives, and rationale behind the need to implement the NSS-based features of the video quality evaluator RAPIQUE in C++. It presents the theoretical framework, methodology, human and physical resources, and activities scheduled to develop this project.

Chapter 2

Problem definition

In the context of the exponential increase in the creation of user-generated video content (UGC), which includes distortions that reduce quality, it has become critical to conduct an accurate, effective, and efficient assessment of video quality on digital platforms. RAPIQUE, a video quality assessment model, offers a promising solution but has significant limitations in its current implementation. Although effective in evaluation accuracy, its performance in Matlab results in long run times, making it unsuitable for pseudo-real-time applications.

The current undesirable situation is, therefore, a RAPIQUE implementation that, despite its accuracy in quality assessment, needs short runtimes and the computational speed required to meet the demands of pseudo-real-time applications.

The aim is to address the problem by optimizing the execution times of RAPIQUE, specifically in terms of execution times and feature extraction, so that it can effectively apply in pseudo-real-time environments.

2.0.1 Problem statement

The digital era's unprecedented growth in user-generated video content (UGC), fueled by platforms like TikTok, YouTube, and Facebook, has brought new challenges in video quality assessment—distortions and low quality, necessitating tools capable of swift, high-volume quality evaluation plague these videos. Rapid and Accurate Video Quality Prediction of User-Generated Content (RAPIQUE) has emerged as a potential solution in response to this need. However, it has limitations (Mittal et al., 2012a; Tu et al., 2021b).

In particular, the implementation of RAPIQUE in Matlab, although powerful for simulation and analysis, has resulted in prolonged execution times, limiting its applicability in pseudo-real-time situations (Tu et al., 2021b). Additionally, compared to RAPIQUE's convolutional neural network (CNN) module that leverages low-level optimizations, RAPIQUE in the spatial and temporal NSS feature extraction modules needs help keeping pace. It manifests itself in lower execution times and intensive processing of features such as natural scene statistics (NSS) with ResNet-50, although the combination of these features has shown good performance (Seshadrinathan et al., 2010a).

The background related to RAPIQUE reveals a complex story. Initially hailed for its efficiency compared to blind video quality (BVQA) models, subsequent studies have highlighted its limitations, especially in pseudo-real-time applications. Comparative research with other models, such as the two-level approach for no-reference consumer video quality assessment (TLVQM) (Korhonen, 2019) has underscored that, although it is ten times faster, it still faces challenges in terms of efficiency.

Furthermore, detailed analyses of the combination of NSS features with ResNet-50 (He et al., 2016) revealed that, although promising, the implementation required improved execution times. Developments in demand for pseudo-real-time applications have also increased the need to optimize RAPIQUE for these applications (Tu et al., 2021b).

The problem with RAPIQUE is complex and multifaceted. Its potential as a video quality assessment tool is evident. Still, its current implementation and design have limited its applicability, leading to adverse effects such as challenges in pseudo-real-time applications due to prolonged execution times and difficulties in feature extraction. Although powerful, the synergy between NSS and ResNet-50 requires more optimized implementation (Seshadrinathan et al., 2010a; Tu et al., 2021b).

In the face of these challenges, (Tu et al., 2021b) propose improvements that could accelerate RAPIQUE by orders of magnitude. These include reimplementation in C++, which would enhance processing speed and allow more seamless integration with tools like FFmpeg and openCV. Additionally, the optimization of feature extraction and the improved implementation of unreferenced metrics, along with the implementation of GPU-suitable frameworks such as Tensorflow or PyTorch, could achieve superior performance, positioning RAPIQUE as a leading tool in its field (Tu et al., 2021b).

The redesign of RAPIQUE with the C++ implementation offers a significant opportunity to overcome the speed and efficiency limitations plaguing the model's Matlab version. By using efficient libraries such as OpenCV and FFmpeg, the reimplementation will increase processing speed and enable more effective integration with other systems. This approach will also facilitate the design of a modular and scalable architecture, allowing RAPIQUE to be used as a library in the broader range of near pseudo-real-time video quality assessment applications. Such an architecture would be invaluable in facilitating code reuse and maintenance, critical components of successful software engineering (Kaehler and Bradski, 2016; Tu et al., 2021b; Wu et al., 2019; Zach and Slanina, 2014; Ying et al., 2021).

The complexity of the problem requires meticulous attention to detail. Still, with the right solutions, RAPIQUE could overcome its limitations and play a vital role in the burgeoning field of pseudo-real-time video quality assessment in the digital age. By filtering out low-quality content and promoting high-quality videos, platforms can significantly improve user satisfaction and engagement rates (Xu et al., 2014).

2.0.2 Problem formulation

Therefore, this situation proposal raises the following research questions:

- How to implement and reduce in C++ the execution time of the VQA metric no-reference RAPIQUE algorithm?
- How to implement and improve in C/C++ the execution time of the NSS spatial feature extraction module?
- How to implement in C++ the RAPIQUE algorithm in a scalable and reusable architecture?

Chapter 3

Project objectives

3.0.1 General Objective

To implement in C++, the video quality evaluator no-reference RAPIQUE focused on reducing execution times.

3.0.2 Specific objectives

- To implement the NSS spatial features extraction by using C++.
- To design and implement a scalable and reusable C++ focused architecture for RAPIQUE.
- To comprehensively evaluate and compare the Matlab-based implementation of RAPIQUE and the new C++ based implementation in terms of execution times and video quality prediction accuracy.

3.0.3 Expected results

- Improved Execution Times: Re-implementing the RAPIQUE algorithm in C++ is expected to decrease execution times. It will enable its applicability in pseudo-real-time environments like live video streaming.
- Execution Times Validation: With re-implementation in a low-level programming language and possibly the use of GPU-friendly frameworks such as TensorFlow or PyTorch, expect to demonstrate that RAPIQUE can meet or exceed the execution times of the CNN-based model module of RAPIQUE.
- Optimized NSS Feature Extraction: Optimization of feature extraction focused on natural scene statistics (NSS), both in spatial aspects and time-band filters, expect to achieve it through implementation in C++.
- Integration with Existing Software Tools: Optimization in C++ should allow for more seamless integration with tools such as OpenCV, which could extend the practical applications of RAPIQUE in the video production and distribution chain.
- Benchmarking: The conduct rigorous benchmarking with existing models such as TLVQM, VSFA, MDVSFA, and VIDEVAL models to validate the efficiency and accuracy improvements of RAPIQUE.

Chapter 4

Scope of the Work

This project improves the RAPIQUE model Tu et al. (2021b), a non-referenced video quality evaluator, by re-implementing it in C++.

A desktop application architecture is developed in C++, specifically designed for high performance. This architecture includes advanced modules for the extraction of Natural Scene Statistical Features (NSS), both in temporal and spatial aspects, leveraging the capabilities of open-source libraries such as FFMPEG and OpenCV, and integrating convolutional neural networks (CNN) for the main RAPIQUE algorithm.

The efficiency and accuracy of the C++ implementation are evaluated based on its precision and its contribution to democratizing RAPIQUE. The accuracy is assessed through the Pearson correlation coefficient (PLCC) and Spearman Correlation Coefficient (SRCC) between the features extracted from the algorithm in Matlab and its C++ re-implementation, aiming for a benchmark higher than 80%. The efficiency and execution times are measured in terms of how the implementation facilitates free access to RAPIQUE, reducing reliance on paid Matlab licenses for feature extraction, and making the model more accessible for diverse video sizes and qualities.

The new C++ architecture focuses not only on scalability and reusability but also on facilitating its application in various environments for video quality assessment, adapting to today's diverse needs. This approach promotes efficient code maintenance and reuse.

The overarching goal is to democratize the RAPIQUE model, enhancing its accessibility and practicality in the field of video quality assessment, especially for users without access to Matlab.

Chapter 5

Justification

The burgeoning growth in producing and disseminating distorted or low-quality user-generated video content (UGC) requires effective and accurate video quality assessment (VQA) tools. RAPIQUE, initially lauded for its performance in the evaluation of non-referenced VQA, still presents considerable limitations in terms of execution time (Tu et al., 2021b). Its successful optimization will serve various stakeholders, including VQA researchers, media companies, and the general user population.

Platform Impact: Provide platform operators with valuable information on the technical characteristics of video content, helping in the decision-making process regarding content optimization and delivery strategies by automating the quality control process, ensuring that the content delivered meets specific quality standards, and a robust quality assessment tool, an indispensable asset for video platforms striving to maintain a competitive edge in the market by offering superior video quality and user experience (Seufert et al., 2014; Möller and Raake, 2014).

Economic Impact: From a financial perspective, "time is money" is more than just a colloquialism; it is an operational principle. Streamlining the execution time of the VQA metrics in RAPIQUE would allow researchers and media companies to expedite their workflow. This results in a tangible reduction in operational costs (Kondratyuk et al., 2021). According to empirical evidence Tu et al. (2021b), a reimplementation of RAPIQUE in C++ can accelerate its execution time by orders of magnitude, thereby significantly enhancing both efficiency and productivity.

Social Impact: This optimization's social implications are not understated. Improved efficiency in VQA is directly proportional to enhanced user experience on digital media platforms, a factor increasingly acknowledged as essential for digital well-being (Hargittai and Hinnant, 2008). In a landscape where video serves as a primary mode of communication, the utility of a high-performance, non-referenced VQA tool like RAPIQUE cannot be overstated (Anderson and Rainie, 2018).

Environmental considerations: While not the primary focus of the VQA optimization tool, the environmental ramifications are not insignificant. Shorter runtime inherently leads to lower energy consumption in data centers, an increasingly pressing concern in today's climate (Shehabi et al., 2016).

Finally, optimizing the runtime of RAPIQUE using C++ alongside the OpenCV library holds promising potential for enhancing video compression methods. The low-level language allows rapid execution and more direct control over hardware resources, which is critical for pseudo-real-time video processing (Stroustrup, 2013; AHO et al., 1988; Lin and Kuo, 2011). Additionally, OpenCV offers a rich set of video processing functionalities, facilitating complex tasks related to video compression and quality assessment (Rosebrock, 2018). The seamless integration of these technologies

could result in a more efficient and effective video compression pipeline since the modularity enabled by OpenCV's well-structured libraries could also lead to better maintainability and extensibility of the software architecture, aligning with good practices in software engineering (Rosebrock, 2018).

Chapter 6

Theoretical framework and background

6.0.1 Theoretical Bases

6.0.2 Subjective methods

The quality of a video is a critical metric for both video transmission service providers and consumers. Several methods exist to evaluate video quality, but subjective evaluation is considered the gold standard in the field due to its focus on human perception (BT, 2012; Wang and Bovik, 2009).

Subjective evaluation of video quality involves showing video sequences to viewers and recording their opinions. These opinions are then averaged to calculate the Mean Opinion Score (MOS) according to Equation 1:

$$\text{MOS} = \frac{\sum_{n=1}^{N} R_n}{N}$$

Equation 1. Mean Opinion Score. Source (ITU, 2017)

Where Rn are the individual scores given by the stimulated subjects and N is the total number of people who participated in the test (ITU, 2017).

It is important to note that observers judge quality based on their perception and previous experience (Pinson and Wolf, 2004).

In subjective methods, quality scales can be continuous or discrete, generally between 5 and 11 points. In the case of MOS, the most widely accepted metric system is the 5-point scale: 1 as the worst score ("very poor quality"), and 5 as the best ("excellent quality") (Winkler, 2005).

However, subjective methods are expensive, difficult to implement, and impractical for real-time applications (Seshadrinathan and Bovik, 2009). It has led to the need to develop objective and automatic methods that can reliably predict perceived quality.

6.0.3 Objective methods

While subjective methods of video quality assessment remain the gold standard, objective methods are invaluable because they provide rapid and automated estimates (Wang and Bovik, 2002). Objective methods are mathematical and statistical models that approximate the results of subjective quality assessments.

Objective methods are based on statistical criteria, such as regressors or training metrics, that can be objectively measured and automatically evaluated by software (Ghadiyaram and Bovik, 2015a). These methods benefit pseudo-real-time applications and large data sets where subjective methods are impractical.

- Pixel-based methods (NR-P): These models use a coded signal representation and analyze quality as a function of pixel information. Some methods evaluate only specific distortions, such as blur or other coding artifacts (Lin and Kuo, 2011).
- Parametric/Bitstream (NR-B) methods: These models use features extracted from the video bitstream, including packet headers, motion vectors, and quantization parameters (Yeganeh and Wang, 2012).
- Hybrid NR-P-B methods (Hybrid NR-P-B): These are a mixture of the NR-P and NR-B models, taking advantage of both approaches to provide a more accurate and comprehensive video quality assessment (Pedersen and Hardeberg, 2009).

As the demand for high-quality video continues to grow, the importance of objective methods of video quality assessment cannot be underestimated. While lacking the accuracy of subjective methods, objective methods offer a faster and more scalable solution for assessing video quality in various applications.

6.0.4 Databases (Authentic, In-capture Video Distortions)

Video quality assessment is critical in various settings, from social networking to professional applications. However, traditional approaches, which generally focus on videos captured by high-end cameras, may not apply to authentically distorted videos of ordinary people using mobile devices (Redi et al., 2010; Ghadiyaram et al., 2017c).

Videos considered authentically distorted are those captured by casual and inexperienced users with mobile cameras. These videos incorporate a range of distortions introduced during capture that are due to numerous sensitive variables such as lighting, exposure, lens limitations, noise sensitivity, acquisition speed, in-camera processing, and camera movement (Chandler and Hemami, 2007; Ghadiyaram et al., 2017c).

Sensitive Variables in Video Capture:

- Lighting and Exposure: Video quality can be severely affected by insufficient or excessive illumination and incorrect exposure settings (Ghadiyaram et al., 2017c,b).
- Lens Limitations and Noise Sensitivity: Optical imperfections and noise sensitivity can introduce distortions such as chromatic aberration and grain noise (Ghadiyaram et al., 2017c,b).
- Camera Acquisition and Processing Speed: The speed at which video is captured and processed can also have an impact, causing problems such as motion blur and compression artifacts (Winkler, 2017).

- Camera Motion: Camera shake and other unwanted movements can introduce distortions during capture, despite image stabilization technologies (Ghadiyaram et al., 2017c).

One of the critical challenges is the collection of authentically distorted or user-generated video, so databases such as KoNViD-1k, LIVE-Qualcomm, LIVE Video Quality Challenge (VQC), YouTube-UGC, and CVD2014 have been developed to evaluate the performance of non-referenced or blind video quality assessment methods, which are described in Table 6.1 (Wu et al., 2012; Ghadiyaram et al., 2017c; Hosu et al., 2020; Ghadiyaram et al., 2017a; Sinno and Bovik, 2018; Wang et al., 2019; Nuutinen et al., 2016).

Table 6.1: Public consumer video quality databases compared: KoNViD-1k, LIVE-Qualcomm, LIVE Video Quality Challenge (VQC), YouTube-UGC and CVD2014. Source (Hosu et al., 2020; Ghadiyaram et al., 2017a; Sinno and Bovik, 2018; Wang et al., 2019; Nuutinen et al., 2016)

Dataset characteristic	KoNViD-1k	LIVE-Qualcomm	LIVE-VQC	CVD2014	YouTube UGC
Number of test videos	1200	208	585	234	1045
Video resolution	960x540	1920x1080	320x240-1920x1080	640x480, 1280x720	640x480 - 3840x2160
Video frame rate	23-29 frames/sec	30 frames/sec	19-30 frames/sec (One sequence 120 frames/sec)	9 30 frames/sec	30 frames/sec-Constant
Video length	8 sec	15 sec	10 sec	11 28 sec	20 seconds
Number of scenes	1200	54	585	5	
Number of devices	> 164	8	101	78	
Test methodology	Crowdsourcing	Lab-based	Crowdsourcing	Lab-based	
Number of test subjects	642 (min. 50 per video)	39	min. 200 per video	27-33 (six experiments)	
Rating scale	Absolute Category Rating (1-5)	Continuous 0-100	Continuous 0-100	Continuous 0 100	
Audio track included	Yes (some)	No	Yes	Yes	
Main strength	Vast diversity of contents				Provides a large-scale UGC dataset for research, covering popular categories like Gaming, Sports, and new features like High Dynamic Range (HDR)
Main weakness	Some contests and distortions have little practical relevance to NR-VQA. The test methodology is prone to unreliable individual scores.	Many scenes, but different scene types could be more balanced. Only smartphones are used as cameras.	Distribution of MOS values biased towards high scores. Some resolutions are represented by a few sequences only.	Some methodological inconsistencies between experiments. A small number of different scenes.	

Continued on next page

Table 6.1 – *Continued from previous page*

Dataset characteristic	KoNViD-1k	LIVE-Qualcomm	LIVE-VQC	CVD2014	YouTube UGC
Remarks	Some contents in the database are clipped from the original.	Additional information was collected concerning the dominating distortion type of each test sequence.	At the time of writing, it is not yet publicly available for download.	Six different experiments. In some experiments, other information was collected aside from video quality (audio quality, contrast, blurriness, etc.)	Lacks subjective quality assessment and may need to represent the diversity of UGC videos on YouTube fully.

6.0.5 Algorithms

For this project, the five No-Reference Video Quality Assessment (NR-VQA) metrics are used. These algorithms are capable of understanding human reactions to video distortions by being trained on extensive datasets filled with human evaluation scores (Bovik, 2013).

6.0.5.1 RAPIQUE

RAPIQUE is a rapid and accurate video quality predictor for user-generated content. It combines novel, effective, and easily computed low-level scene statistics features with high-level deep learning features, using aggressive spatial and temporal sampling strategies to increase efficiency without sacrificing performance. RAPIQUE architecture includes a spatial NSS feature extraction module that is a highly efficient and effective (Tu et al., 2021b).

The workflow of RAPIQUE (Tu et al., 2021b):

- **Input video**: The input video is first preprocessed to extract spatial and temporal features. The spatial features are extracted using a spatial NSS feature extraction module, while the temporal features are extracted using a temporal natural Scene Statistical Features (NSS) feature extraction module.
- **CNN feature extraction**: The input video is also fed into a convolutional neural networks (CNN) feature extraction module to extract high-level deep learning features.
- **Feature concatenation**: The final feature vector is simply concatenated from the extracted spatial and temporal NSS and the CNN features, which is further used to train a regressor head.
- **Regression**: The concatenated feature vector is then used to train a regressor head, which predicts the subjective quality of the input video.
- **Quality prediction**: The predicted quality score is then outputted as the final result.

RAPIQUE achieves superior performance that is comparable or better than (V-BLIINDS, V-MEON, TLVQM, VSFA, MDVSFA, and VIDEVAL) models, but with a relative 20x speedup on 1080p videos. The runtime of RAPIQUE also scales well as a function of video resolution and is 60x faster than the (V-BLIINDS, V-MEON, TLVQM, VSFA, MDVSFA, and VIDEVAL) model on 4k videos. The average running time of RAPIQUE on video 1080p and 3.8k DIM is 17.3 seconds (Tu et al., 2021b).

RAPIQUE is implemented in MATLAB and Python 3.6.7 under Ubuntu 18.04.3 LTS system. The experiments were carried out on a Dell OptiPlex 7080 Desktop with an Intel Core i7-8700 CPU@3.2 GHz, 32 G RAM, and GeForce GTX 1050 Graphics Cards (Tu et al., 2021b).

RAPIQUE involves preprocessing the input video to extract spatial and temporal features, extracting high-level deep learning features using a CNN feature extraction module, concatenating the extracted features, training a regressor head, and predicting the subjective quality of the input video (Tu et al., 2021b).

6.0.5.2 TLVQM

TLVQM (two-level approach for no-reference video quality assessment) that is specifically aimed at user-generated content that is prone to capture artifacts, such as camera shakiness, over- and underexposure, and sensor noise. The method is designed to be both accurate and computationally efficient (Korhonen, 2019).

The method consists of two levels of processing: segment-level processing and sequence-level processing. At the segment level, the method computes low-complexity spatiotemporal features and more complex spatial features from a subset of representative frames. At the sequence level, the method applies average temporal pooling to the segment-level Low Complexity Features (LCFs), High Complexity Features (HCFs), and consistency features to obtain the feature vector for the whole video sequence (Korhonen, 2019).

The method uses a hierarchical approach to compute spatiotemporal features. Low-complexity spatiotemporal features are computed from every second frame, and more complex spatial features are only computed from a subset of representative frames. The method also uses consistency features to capture temporal consistency in the video sequence (Korhonen, 2019).

The average execution time of the TLVQM on a single video sequence is approximately 0.5 seconds (Korhonen, 2019).

The average running time for TLVQM on the LIVE-VQC database has an average running time of 0.47 seconds per video sequence. The KoNViD-1k database has an average running time of 0.48 seconds per video sequence. The execution time of the proposed method may vary depending on the content of the video sequence (Korhonen, 2019).

TLVQM is implemented in Matlab. The experiments were conducted on a standard desktop computer with an Intel Core i7-8700K CPU and 32 GB of RAM. The proposed method is evaluated on annotated public video quality databases (LIVE-VQC, KoNViD-1k, CVD2014). TLVQM can predict subjective video quality more accurately than the legacy NR-VQMs known from prior art, with substantially lower computational complexity (Korhonen, 2019).

6.0.5.3 VSFA

VSFA (Quality Assessment of In-the-Wild Videos) is an objective, no-reference video quality assessment method that integrates content-dependency and temporal-memory effects into a deep neural network (Li et al., 2019).

The workflow of VSFA (Li et al., 2019):

- Content-Aware Feature Extraction: VSFA extracts content-aware features from a pre-trained image classification neural network for each video frame. These features capture the inherent content information of the video frame, which is essential for assessing the perceived video quality.

- Long-Term Dependencies Modeling: The extracted frame-level features are sent to a fully connected layer for dimensional reduction followed by a gated recurrent unit (GRU) network

for long-term dependencies modeling. The GRU network models the temporal dependencies between frames, which is vital for capturing the temporal-memory effects of video quality.

- Frame-Wise Quality Scores: The GRU network outputs the frame-wise quality scores, representing each video frame's predicted quality.
- Temporal Pooling: To account for the temporal hysteresis effect, the overall video quality is pooled from these frame quality scores by a subjectively-inspired temporal pooling layer. This layer applies a differentiable pooling function inspired by the human visual system, which is known to exhibit temporal hysteresis.
- Video Quality Prediction: The output of the temporal pooling layer is the predicted overall video quality, which represents the video's perceived quality.

VSFA was evaluated on three publicly available in-the-wild video quality assessment databases: KoNViD-1k, CVD2014, and LIVE-Qualcomm. The experiments demonstrate that VSFA outperforms (BRISQUE, NIQE, CORNIA, VIIDEO, and VBLIINDS) by a large margin, specifically, 12.39%, 15.71%, 15.45%, and 18.09%, in terms of SROCC, KROCC, PLCC and RMSE, respectively. The average computation time for the VSFA process varies depending on the frame rate and resolution. Specifically, it takes approximately 269.84 seconds to process 240 frames at 540p, 249.21 seconds for 364 frames at 480p, 936.85 seconds for 467 frames at 720p, and 2081.84 seconds for 450 frames at 1080p. The experiments were conducted on a computer with an Intel Core i7-8700K CPU @ 3.70GHz and an NVIDIA GeForce GTX 1080 Ti GPU. The operating system used was Ubuntu 16.04.6 LTS (Li et al., 2019).

VSFA integrates content-dependency and temporal-memory effects into a deep neural network, allowing it to capture video quality's complex and dynamic nature in real-world scenarios.VSFA can be accelerated to 30x faster or more by switching the CPU mode to the GPU mode (Li et al., 2019).

6.0.5.4 MDTVSFA

MDTVSFA is a unified framework for quality assessment of in-the-wild videos. MDTVSFA works by extracting informative spatiotemporal features from the input video, selecting a subset of informative features, and predicting the quality score of the input video. The workflow of MDTVSFA (Li et al., 2021):

- Feature Extraction: The first step in the workflow is to extract spatiotemporal features from the input video. The authors use a 3D convolutional neural network (CNN) to extract frame-level features and a long short-term memory (LSTM) network to capture temporal dependencies. The output of the feature extraction component is a set of high-dimensional feature vectors representing the input video's spatiotemporal characteristics.
- Feature Selection: The second step in the workflow is to select a subset of informative features from the high-dimensional feature vectors. The authors use a two-stage feature selection

strategy that combines a filter-based method and a wrapper-based method. The filter-based method selects features based on their correlation with the quality scores. In contrast, the wrapper-based method selects features based on their contribution to the performance of the quality prediction model.

- Quality Prediction: The final step in the workflow is to predict the quality score of the input video based on the selected features. The authors use a support vector regression (SVR) model to predict the quality score. The SVR model is trained on a mixed dataset, including videos from different sources and resolutions.

The workflow of MDTVSFA is designed to extract informative spatiotemporal features from the input video, select a subset of informative features, and predict the quality score of the input video. The authors use a deep neural network, a two-stage feature selection strategy, and an SVR model to achieve high performance in quality assessment of in-the-wild videos (Li et al., 2021).

MDTVSFA achieves the highest mean SROCC values on two videos with (640×480, 364 frames, 1280×720, 467 frames) lengths of datasets CVD2014, KoNViD-1k and LIVE-Qualcomm, LIVE-VQC. MDTVSFA outperforms BRISQUE, NIQE, CORNIA, VIIDEO, FC Model, STFC Model, STS-CNN200 including TLVQM and V-BLIINDS models (Li et al., 2021).

MDTVSFA has a moderate execution time, with an average execution time of 0.3802 seconds per video (CPU -SROCC - 640×480, 364 frames), 0.0414 seconds per video (GPU -SROCC - 640×480, 364 frames), 0.4771 seconds (CPU – SROCC - 1280×720, 467 frames), 0.1139 seconds (GPU – SROCC - 1280×720, 467 frames), on a desktop computer with an Intel Core i7-6700K CPU@4.00 GHz, 12G NVIDIA TITAN Xp GPU, and 64 GB RAM. The operating system used was Ubuntu 14.04, and MDTVSFA was implemented on MATLAB (Li et al., 2021).

The average running time of MDTVSFA is also competitive, with the CPU version being faster than V-BLIINDS and the GPU version being the quickest and best-performed method. The CPU version can be accelerated to 30x faster or more by switching to the GPU mode (Li et al., 2021).

6.0.5.5 VIDEVAL

VIDEVAL (Video quality evaluator) is a fusion-based blind video quality assessment (BVQA) algorithm that combines the strengths of multiple existing BVQA models through a feature ensemble and selection procedure (Tu et al., 2021a). The workflow of VIDEVAL (Tu et al., 2021a):

- Feature Extraction: The first step in the workflow is to extract features from the input video. VIDEVAL uses multiple existing BVQA models, including NIQE, ILNIQE, VIIDEO, BRISQUE, GM-LOG, HIGRADE, FRIQUEE, CORNIA, HOSA, KonCept512, PaQ-2-PiQ, V-BLIINDS, and TLVQM, to extract features from the input video. Each BVQA model extracts features at a uniform sampling rate of one frame per second, then temporally average-pooled to obtain the overall video-level feature (Tu et al., 2021a).

- Feature Ensemble and Selection: The second step in the workflow is to select the most informative features from each BVQA model and combine them to create a more robust and

accurate BVQA model. VIDEVAL uses a feature ensemble and selection procedure to select the most informative features from each BVQA model. The set features are then combined using a weighted average to obtain the final quality score (Tu et al., 2021a).

- Back-End Regression Model: The third step in the workflow is to use a support vector regression (SVR) model as the back-end regression model to learn the feature-to-score mappings. The SVR model is trained on a large-scale UGC-VQA database to learn the feature-to-score mappings (Tu et al., 2021a).

- Video Quality Assessment: The final step in the workflow is to use the trained SVR model to predict the quality score of the input video. The predicted quality score represents the perceived quality of the input video (Tu et al., 2021a).

The datasets used to evaluate the performance of VIDEVAL include the LIVE-VQC, KoNViD-1k, CVD2014, and YouTube-UGC datasets. The run time on twenty 1080p videos from LIVE-VQC is 305.8 segundos (using CPU). These experiments were conducted on a server with an Intel Xeon E5-2630 v4 CPU (2.2 GHz, ten cores), 128 GB of RAM, and an NVIDIA Tesla P100 GPU (16 GB of memory). The operating system used was Ubuntu 16.04 LTS, and the software used included MATLAB R2018b, TensorFlow 1.12.0, and PyTorch 0.4.1 (Tu et al., 2021a).

VIDEVAL involves extracting features from the input video using multiple existing BVQA models, selecting the most informative features from each model using a feature ensemble and selection procedure, using a support vector regression (SVR) model as the back-end regression model to learn the feature-to-score mappings, and predicting the quality score of the input video (Tu et al., 2021a).

6.0.6 Natural Scene Statistics

Visual perception and the quality of visual representations, such as images and videos, are topics of interest in engineering and neuroscience. Natural Scene Statistics (NSS) are crucial for understanding how these representations are perceived and evaluated. In the context of video quality assessment, NSS offers a range of features that can be both descriptive and predictive of how video quality will be perceived (Bex and Makous, 2002; Simoncelli and Olshausen, 2001).

Natural scene statistics are deeply linked to how the human visual system (HVS) processes images and videos. The HVS is extremely sensitive to variations in certain scene features, such as texture, color, and edges, and these sensitivities are reflected in the NSS (Geisler, 2008; Ruderman and Bialek, 1994).

In video quality assessment, NSS is used to quantify the features of a video that are relevant to human perception. These features are often used as inputs to objective video quality models, which seek to emulate the subjective assessments made by human observers (Wang and Bovik, 2009).

The NSS characteristics in Video Evaluation are classified as follows:

- Texture and Contrast: texture patterns and contrasts in a scene strongly affect quality perception. Inconsistencies in these features are often perceived as quality degradations (Balas et al., 2009).

- Color Distribution: changes in the color distribution of a video can drastically affect its perceived quality. Deviations from natural color are generally misperceived (Fairchild and Johnson, 2004).
- Motion Dynamics: in a video, motion is a crucial feature. NSS can describe motion properties in a way that correlates with human perception of quality (Daly, 1992).

One of the main challenges in using NSS for video quality assessment is the need for large datasets for training reliable, objective models. In addition, the interpretation of these features requires a thorough understanding of both human vision and the underlying mathematics (Winkler, 2005).

6.0.7 GGD

The Generalized Gaussian Distribution (GGD) plays a crucial role in the field of image and video quality assessment. Its flexibility in shape and its ability to model a wide range of data distributions make it especially valuable in analyzing the statistical characteristics of natural scenes in videos.
Definition:

The GGD is defined by its probability density function (PDF), which is expressed as:

$$f(x \mid \alpha, \sigma, \nu) = \frac{\nu}{2\alpha\Gamma(1/\nu)} \exp\left(-\left(\frac{|x-\mu|}{\alpha}\right)^{\nu}\right)$$

where μ is the mean, α scales the width of the distribution, ν shapes the distribution, and Γ is the gamma function. The flexibility of ν allows the GGD to adapt its shape to match the actual data distribution closely, from exponential forms ($\nu < 2$) to Gaussian-like forms ($\nu = 2$), making it highly effective for modeling diverse texture patterns and noise in video frames Lam and Goodman (2000).

6.0.8 AGGD

The Asymmetric Generalized Gaussian Distribution (AGGD) extends the modeling capabilities of the Generalized Gaussian Distribution by incorporating skewness, making it particularly effective in handling the asymmetrical behavior observed in real-world data, crucial for image and video quality assessment.

Definition:

The AGGD is characterized by its probability density function (PDF), which is formulated as:

$$f(x \mid \alpha, \sigma_l, \sigma_r, \nu) = \begin{cases} \frac{\nu}{(\alpha_l+\alpha_r)\Gamma(1/\nu)} \exp\left(-\left(\frac{x-\mu}{\alpha_l}\right)^{\nu}\right) & \text{if } x < \mu \\ \frac{\nu}{(\alpha_l+\alpha_r)\Gamma(1/\nu)} \exp\left(-\left(\frac{x-\mu}{\alpha_r}\right)^{\nu}\right) & \text{if } x \geq \mu \end{cases}$$

where μ is the location parameter (mean), α_l and α_r are scale parameters for the left and right tails, respectively, ν controls the shape of the distribution, and Γ is the gamma function. This asymmetric adaptation allows AGGD to closely represent data with different behaviors in its tails, enhancing its applicability in diverse image and video contexts Doe and Roe (2021).

6.0.9 GM

Gradient Magnitude (GM) is a fundamental metric in the field of image processing and video quality assessment, serving as a primary indicator of edge and texture detail in images and video frames.

Definition:

The Gradient Magnitude is calculated as the Euclidean norm of the gradient vector, which is derived from the image's intensity values. It is mathematically expressed as:

$$GM(x,y) = \sqrt{(\frac{\partial I}{\partial x})^2 + (\frac{\partial I}{\partial y})^2}$$

where $\frac{\partial I}{\partial x}$ and $\frac{\partial I}{\partial y}$ are the horizontal and vertical derivatives of the image I, computed typically using operators like Sobel or Prewitt. The gradient magnitude highlights regions of significant intensity change, which are indicative of edges and textures in the image, making GM an essential measure for assessing visual quality and sharpness Gonzalez and Woods (2002).

6.0.10 DoG

The Gaussian and Difference of Gaussian (DoG) filters are fundamental tools in image processing, particularly in the domains of image enhancement and edge detection. Their application is crucial in the analysis of visual quality in images and videos.

Definition:

The Gaussian filter is a smoothing operator that is used to reduce noise and detail in images, described by the function:

$$G(x,y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

where σ is the standard deviation of the Gaussian distribution. This filter is essential for pre-processing in various image processing tasks to enhance image structures at different scales Jain (1989).

The Difference of Gaussian (DoG) is obtained by subtracting one blurred version of an image from another, less blurred version of the same image. It approximates the Laplacian of Gaussian (LoG) and is given by:

$$DoG(x,y) = \frac{1}{\sqrt{2\pi}\sigma_1} \exp\left(-\frac{x^2+y^2}{2\sigma_1^2}\right) - \frac{1}{\sqrt{2\pi}\sigma_2} \exp\left(-\frac{x^2+y^2}{2\sigma_2^2}\right)$$

where σ_1 and σ_2 are the standard deviations of the two Gaussian kernels, with $\sigma_2 > \sigma_1$ Jain (1989).

6.0.11 MSCN

Mean Subtracted Contrast Normalization (MSCN) is an important technique in the field of image and video quality assessment, particularly for its effectiveness in highlighting textural and contrast features within an image or video frame.

Definition:

The MSCN technique is defined by its transformation, which normalizes an image by subtracting the local mean and dividing by the local standard deviation:

$$I_{\text{MSCN}}(i,j) = \frac{I(i,j) - \mu(i,j)}{\sigma(i,j) + 1}$$

where $I(i,j)$ is the pixel intensity at coordinates (i,j), $\mu(i,j)$ is the local mean, $\sigma(i,j)$ is the local standard deviation, and a small constant (usually 1) is added to the denominator to avoid division by zero. This process enhances the visibility of structural information and textures in areas of varying brightness, making it valuable for preprocessing in video quality metrics Wang et al. (2003).

6.0.12 Convolutional neural network

In the last decade, Convolutional Neural Networks (CNNs) have significantly performed in various deep learning tasks, including video quality assessment (LeCun et al., 2015; Krizhevsky et al., 2017).

Convolutional Neural Networks (CNNs) are a subcategory of artificial neural networks that have gained prominence in various applications, including video quality assessment. These networks are specially designed to process data with a topological grid structure, typically images (LeCun et al., 2015).

The fundamental components of the CNNs are:

- Convolution: the core of the CNN is the convolution operation, which takes a small area of the input image and transforms it into a shape that makes it easier to analyze (Goodfellow et al., 2016). It allows the network to focus on local and spatial features.

- Pooling Layers: these layers reduce the dimensions of the feature map, which reduces the number of parameters and computations in the network, and therefore also controls overfitting (Simonyan and Zisserman, 2014).

- Fully Connected Layers: these layers are generally used to perform classification of features learned by the previous layers (Krizhevsky et al., 2017).

CNNs can extract low-level and high-level features from videos, making them suitable for video quality evaluation tasks (Wang et al., 2004b). CNNs effectively understand the complexities related to human perception of video quality (Lv et al., 2023).

CNNs are especially good at adapting to various distortions, such as noise, blur, and compression, which are critical in video quality assessment (Zhang et al., 2011).

Learning transfer is another area where CNNs prove to be efficient. Models trained in one domain can be easily adjusted and applied to another, such as moving from still images to video quality assessment (Tajbakhsh et al., 2016).

6.0.13 Regression

Regression, mainly using Support Vector Regression Machines (SVR), has gained attention in the objective video quality assessment. These techniques allow a nonlinear mapping between perceived video characteristics and a quality score, often represented as a Mean Quality Score (MOS) (Cortes and Vapnik, 1995).

SVR is commonly used to learn a nonlinear mapping between a set of features of a video and its corresponding MOS. It has proven an effective tool for automatic and objective video quality assessment (Seshadrinathan et al., 2010a).

Perceptually relevant features include bit rate, resolution, noise, and other factors influencing the perception of quality (Seshadrinathan and Bovik, 2009).

SVR and SVM have proven to be effective in video quality assessment, especially when it comes to handling high-dimensional data. However, there is an imminent need to improve the robustness of the databases used for training these models (Bianco et al., 2018).

6.0.14 Spatial-temporal feature and pooling

In Video Quality Assessment (VQA), spatiotemporal features and Natural Scene Statistics (NSS) are fundamental to understanding and rating perceived video quality (Wang et al., 2004b).

These features consider space and time and are essential to represent the motion and appearance of objects within the video. Some common ones are optical flow and affine transformations (Winkler, 2005).

$$\text{Optical Flow} = \left(\frac{dx}{dt}, \frac{dy}{dt}\right) \text{Optical Flow} = \left(\frac{dx}{dt}, \frac{dy}{dt}\right) \text{Optical Flow} = \left(\frac{dx}{dt}, \frac{dy}{dt}\right) \text{Optical Flow} = \left(\frac{dx}{dt}, \frac{dy}{dt}\right)$$

Equation 1. Optical Flow Formula describes how the coordinates x and y of a point in an image change concerning time t.Source (Patrone et al., 2016).

$$\frac{dx}{dt}$$

Equation 2. It is the partial derivative of x to time t, and it represents the rate of change of the point along the x axis.Source (Patrone et al., 2016).

$$\frac{dy}{dt}$$

Equation 3. It is the partial derivative of y for time t, and it represents the rate of change of the point along the y axis. Source (Patrone et al., 2016).

In simpler terms, optical flow aims to describe how and in which direction the points in an image (or, more precisely, a series of images in a video sequence) are moving over time.

NSS are metrics derived from information theory that quantify a scene's "natural" appearance and are often effective in measuring perceived quality (Sheikh et al., 2006).

$$\text{NSS} = \frac{X - \mu}{\sigma}$$

Equation 4. NSS formula.Source (Sheikh et al., 2006).

Where X is the image or frame, μ is the mean, and σ is the standard deviation.

Specifically, frame or image preprocessing in the classical spatial NSS model starts with local mean removal and divisive normalization steps.

$$I(i,j) = \frac{I(i,j) - \mu(i,j)}{\sigma(i,j) + 1}$$

Equation 5. Spatial NSS formula.Source (Mittal et al., 2012b).

where $i \in \{1, 2, \ldots, M\}$, $j \in \{1, 2, \ldots, N\}$ are spatial indices, M and N are the image dimensions, and

$$\mu(i,j) = \sum_{k=-K}^{K} \sum_{l=-L}^{L} w_{k,l} I(i+k, j+l)$$

Equation 6. The standard deviation of NSS Spatial model.Source (Mittal et al., 2012b).

$$\sigma(i,j) = \sqrt{\sum_{k=-K}^{K} \sum_{l=-L}^{L} w_{k,l} \left[I(i+k, j+l) - \mu(i,j)\right]^2}$$

Equation 7. The mean of NSS Spatial model.Source (Mittal et al., 2012b).

Estimate the local mean and contrast, where

$$w = \{w_{k,l} \mid k = -K, \ldots, K, l = -L, \ldots, L\}$$

is a 2D circularly symmetric Gaussian weighting function extended to 3 standard deviations (K = L = 3) and rescaled to have a unit volume.

Spatiotemporal clustering can consider NSS features to achieve a more accurate quality assessment. VQPooling is an example that incorporates these metrics (Saad et al., 2014).

$$Q = \frac{\sum_{n \in GL} q_n + w \cdot \sum_{n \in GH} q_n}{|GL| + w \cdot |GH|}$$

Equation 8. Spatial-temporal grouping strategy (VQPooling). Source (Tu et al., 2020).

Where —G_L— and —G_H— denote the cardinality of G_L and G_H, while the weight w is defined as the ratio between the scores in G_L and G_H.

Where w is defined as:

$$w = \left(1 - \frac{ML}{MH}\right)^2$$

Equation 9. The ratio between the scores in GL and GH.Source (Tu et al., 2020).

where M_L and M_H are the average value of the quality scores in set G_L and G_H, respectively.

Using spatiotemporal and NSS features in Video Quality Assessment (VQA) is promising but computationally challenging, which opens the door to future research in optimization and deep learning (Ma et al., 2017).

6.0.15 Data analysis measures

The assessment of video quality is inherently dependent on the data analysis measures used to correlate objective and subjective quality metrics, including the Pearson Linear Correlation Coefficient (PLCC), Spearman's Rank-Order Correlation Coefficient (SROCC), and Root Mean Square Error (RMSE). The performance and speed of these calculations can be significantly affected by the hardware on which they are run, particularly CPUs and GPUs (Sun et al., 2022, 2021).

- Pearson Linear Correlation Coefficient (PLCC): this measure indicates the degree of linear relationship between two quantitative variables. The PLCC can vary in the interval [-1,1], where 1 indicates a perfect positive linear relationship, and -1 indicates a perfect negative linear relationship (at Kent State University, 2023). The general formula for calculating the PLCC for a population is:

$$\rho_{xy} = \frac{E[(x - \mu_x)(y - \mu_y)]}{\sigma_x \sigma_y}$$

Equation 10. PLCC on a population. Source (at Kent State University, 2023).

With σ_x is the standard deviation of the variable x, σ_y is the standard deviation of the variable y, μ_x and μ_y are the mean of the variable x and y, and E is the expected value (at Kent State University, 2023).

- Spearman's Rank-Order Correlation Coefficient (SROCC): it is a non-parametric measure that evaluates the monotonic relationship between two variables X and Y, such variables can be both continuous and discrete. It is based on rankings rather than raw values (Lehman, 2005).
SROCC is denoted by the symbol rs (or the Greek letter ρ, pronounced rho) and it is calculated as equation 11.

$$\rho = 1 - \frac{6 \sum D^2}{N(N^2 - 1)}$$

Equation 11. Spearman's Rank-Order Correlation Coefficient (SROCC). Source (Lehman, 2005).

To calculate ρ, data are sorted and replaced by their respective order. D is the difference between the corresponding $X - Y$ order statistics. N is the number of data pairs (Lehman, 2005).

- Root Mean Square Error (RMSE): the RMSE is a measure that evaluates the difference between estimated and actual values. It is widely used in regression analysis to measure the accuracy of a model (Lehmann and Casella, 1998). It is calculated as:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Xi - Yi)^2}$$

Equation 12. Root Mean Square Error (RMSE). Source (Lehmann and Casella, 1998).

In which X_i is the vector of the objective quality scores given by the NR metrics, Y_i are the subjective scores, and n are the number of elements of vector X_i or Y_i, because both must have the same cardinality (Lehmann and Casella, 1998).

- Central Processing Unit (CPU): CPU performance is measured in terms of clock cycles required to execute a given set of instructions. Regarding data analysis, algorithms running on the CPU are optimized for sequential operations and can benefit from memory hierarchy and multi-threaded handling capability (Hennessy and Patterson, 2011).

- 3Graphics Processing Unit (GPU): GPUs, on the other hand, are designed for parallel operations and are incredibly efficient in mathematical and graphical computations, such as those required in data analysis, including procedures such as PLCC and SROCC. For massively parallel operations, GPUs can offer significantly faster execution times than CPUs (CUDA, 2012).

- The execution time for functions within the RAPIQUE algorithm and the algorithm as a whole is calculated by measuring the start and end times of the process. The formula to determine the execution time is given by (Lee et al., 2010):

$$T = t_{\text{end}} - t_{\text{start}}$$

Equation 13. Performance Formula. Source (Babi, 2005; Lee et al., 2010).

Where:

  - T represents the execution time.
  - t_{start} is the time at which the function or algorithm begins execution.
  - t_{end} is the time at which the function or algorithm completes execution.

Performance in terms of data analysis is not only limited to the accuracy of metrics but also to efficiency in terms of time and resources. With the continuous advancement in hardware technology, especially in GPUs, runtimes for complex calculations, such as those required in video quality assessment, are decreasing, enabling faster and more efficient analysis (Sun et al., 2022, 2021).

6.0.16 State-of-the-Art

Video quality assessment is a challenging problem that has received significant attention recently due to the increasing demand for high-quality video content. With the rise of user-generated content (UGC) platforms, the need for accurate and efficient video quality evaluation has become more pressing than ever. The models for video quality assessment are typically based on either full-reference (FR), reduced-reference (RR), or no-reference (NR) methods. FR methods require access to the original video, while RR and NR methods can operate on degraded video versions without access to the original (Wang et al., 2004b; Seshadrinathan et al., 2010b).

In recent years, there has been a growing interest in developing NR methods for video quality assessment, which can operate on degraded videos without access to the original. Among NR methods, blind or no-reference (NR) ways are particularly challenging, as they cannot access any information about the original video. It has led to the development of innovative approaches that combine low-level natural scene statistics (NSS) features with high-level deep learning features to achieve superior performance in predicting video quality for UGC videos (Götz-Hahn et al., 2019; Mittal et al., 2012b; Wang et al., 2004b).

The article (Tu et al., 2021b) proposes a method called RAPIQUE, which is a fast and accurate video quality evaluator for user-generated content. It combines novel, efficient and easy-to-compute low-level scene statistics (NSS) features with high-level deep learning features to achieve superior performance that is comparable or better than models (FRIQUEE, V-BLIINDS, TLVQM, VSFA, MDVSFA, VSFA, VIDEVAL) given its superiority in PLCC, RMSE, SROCC and average runtime (scales well as a function of video resolution, and is 60x faster tan VIDEVAL). The final feature vector is simply concatenated from the extracted spatial and temporal NSSs and CNN features, which are used to train a regressor head. The authors mention that its implementation in C++ significantly accelerates the model's runtime.

The article (Wu et al., 2022) proposes method FAST-VQA, which focuses on efficient and effective video quality assessment using Grid Mini-patch Sampling and Fragment Attention Network. The model was implemented using Python and PyTorch, and it is designed to run on both CPU and GPU. The method uses NSS-based features for video quality evaluation, which are extracted using the Temporal Quality Metric (TQM) algorithm. The average run time of the proposed method is 0.044/9.019 seconds on 540px, 0.043/9.53 seconds on 720px, and 0.045/9.142 seconds on a GPU/CPU.

The method is compared with BRISQUE, TLVQM, VIDEVAL, and Existing Deep (VSFA, PVQwo/ patch, PVQw/ patch) video quality evaluation models; it is shown to outperform them in terms of accuracy and efficiency. Also, the authors mention that RAPIQUE can also infer rapidly on CPU that requires 17.3s for 1080P videos. Yet, the actual implementation is incompatible with GPU Inference due to its handcrafted branch. The proposed software architecture consists of three main components: the Grid Mini-patch Sampling module, the Fragment Attention Network module, and the FrAgment Sample Transformer module (Wu et al., 2022).

The article (Tu et al., 2021a) proposes creating a feature-fused model that delivers better consistency against subjective assessment and achieves reliable performance across different databases and use cases. Implementing this model called VIDEVAL involves constructing an initial feature set using compute-efficient BVQA models and features (Selected combination of NSS features in multiple perceptual spaces and using visual impairment features from TLVQM). Feature selection algorithms, such as random forest (RF) and support vector machine (SVM), are used to distill the initial feature set and choose an optimal or sub-optimal feature subset (Tu et al., 2021a).

The study compares VIDEVAL with other BVQA models, such as V-BLIINDS, TLVQM, FRIQUEE, BRISQUE, HIGRADE, and deep learning models like VGG-19, ResNet-50, and KonCept512. The comparison includes evaluation metrics like Spearman Rank-Order Correlation Coefficient (SRCC) and Pearson Linear Correlation Coefficient (PLCC) on different datasets, and the author demonstrates that VIDEVAL is superior in this metric of correlating. Still, the execution average time on 60 dimensionality video is 305.8 seconds. The article mentions that simpler NSS-based models like VIDEVAL, BRISQUE, and HIGRADE show competitive efficiency relative to CNN models even when implemented in MATLAB. It is also noted that there can be a significant speedup if the models are re-implemented in pure C++ (Tu et al., 2021a).

In the article (Mei et al., 2023), the proposed method is called GreenBVQA, a lightweight BVQA method designed for edge devices. It achieves VIDEVAL, TLVQM, V-BLIINDS, CORNIA, NIQE, BRISQUE, and TLVQM performance in PLCC and SROCC metrics while demanding significantly smaller model sizes and lower execution time. GreenBVQA is implemented in Python and optimized for CPU and GPU acceleration using the TensorFlow framework. It is designed to run on Linux-based operating systems.

GreenBVQA uses NSS-based features for video quality evaluation extracted from input images using an ad hoc approach. Then, a regression model is trained to predict the quality score using these handcrafted features. The average run time of GreenBVQA is 240frs@540p on 3.22 seconds, 240frs@540p on 4.88 seconds, 467frs@720p on 6.26 seconds, which shows that it is significantly faster than other BVQA methods such as VSFA, V-BLIINDS, QSA-VQM, NIQE, BRISQUE, and TLVQM. GreenBVQA is optimized for both CPU and GPU acceleration. Compared to RAPIQUE, GreenBVQA achieves better performance in lower execution time (Mei et al., 2023).

Finally, most of the works are focused on the creation and optimization through CPU/GPU of NR video quality evaluation methods, which helps to evaluate the performance of NR VQA metrics, allowing to determine the average execution time, its performance in terms of PLCC, SROCC and RMSE, in addition to specific implementation techniques in parallel execution, the use of Python, Matlab and the recommendations of these studies for implementations in C++ that help to bring the results closer to the expected ones (Mittal et al., 2012a; Xu et al., 2016; Lee et al., 2012; Tu et al., 2021b).

The article (Liu et al., 2021) proposes a method with a spatiotemporal representation learning approach for blind video quality assessment (BVQA). The implementation is based on a deep learning framework, and the model is trained on a large-scale weakly labeled dataset. The study focuses on developing a weakly supervised representation learning method called HEKE, which incorporates multiple knowledge sources to alleviate bias and improve performance. The study uses a spatiotemporal representation encoder, specifically an R(2+1)D network, which is designed to capture both spatial and temporal information in videos. The encoder is trained on synthetic data but performs competently on authentic databases.

The authors compare the proposed method with BVQA models such as VIDEO, VBLIINDS, VSFA, TLVQM, NIQE, and VIDEVAL. It also conducts an entirely blind performance comparison on various databases, including LIVE, CSIQ, IVPL, IVC-IC, EPFL-PoliMI, LIVE-Mobile, LIVE-VQC, YouTube-UGC, and KoNViD-1K. The evaluation criteria used are Pearson linear correlation coefficient (PLCC), Spearman rank-order correlation coefficient (SRCC), and root mean square error (RMSE), which obtained better results in the correlation coefficients (Liu et al., 2021).

The runtime analysis shows that the proposed method with R(2+1)D takes moderate time consumption, and the performance is superior with other methods such as NSTSS, VIDEO, VBLIINDS, VSFA, TLVQM with an average time of 5 seconds (GPU) on a video with 768 $\times$432 and 250 frames. It should be noted that the methods were executed only with CPU while HEKE used GPU (Liu et al., 2021).

According to the above, we will seek to implement in C++ the RAPIQUE model establishing the connection between subjective studies and objective studies focused on video analysis, generating a higher degree of difficulty in the evaluation of the quality of the NR, using different metrics and databases of NR, to subsequently train a regressor with a good performance and make the comparison in terms of execution times (Tu et al., 2021b,a).

Chapter 7

Methodology

The proposed research adopts a multi-faceted approach that encompasses quantitative, descriptive, and correlative research designs, optimization and implementation strategies. Specifically, this study aims to implement and optimize a No-Reference Video Quality Assessment (NR VQA) metric using a low-level programming language. The primary objectives are to analyze data, conduct training, and rigorously evaluate the metric's performance in terms of average execution times when applied to videos with real-world distortions.

The research journey begins with an extensive literature review and theoretical analysis, focusing on both subjective studies and existing NR VQA metrics. Particular emphasis will be placed on computationally efficient metrics that exhibit pseudo-real-time or low execution times. The research methodology is structured into six distinct stages to ensure a comprehensive development process.

- Stage 1: RAPIQUE architecture and patterns design focused on C++
  - Specific objective: To design an architecture for Rapid Evaluation of Perceptual Image Quality Estimator (RAPIQUE) in C++, leveraging OpenCV and focusing on parallel execution for non-reference video quality evaluation. The work aims to achieve optimized performance metrics, including execution time on CPUs and GPUs.
  - Tools and Technologies
    - Programming Languages: C++
    - Libraries: OpenCV, CUDA (for GPU acceleration)
    - Architectural Patterns: Singleton, Factory
    - Hardware: Diverse CPU and GPU configurations for benchmarking
    - Profiling Tools: Python (gpuutils)
  - Tasks and Procedures
    - Task 1: Requirement Analysis: Define the essential features and requirements for the RAPIQUE architecture.
    - Task 2: Architecture Design: Investigate and choose the best suitable architecture, design pattern, and class for no-referenced video quality prediction models. For example, use software architecture design patterns Singleton or Factory to conceptualize the initial architectural blueprint (Model C4).

        - Task 3: OpenCV Integration: Plan how OpenCV libraries can be integrated efficiently into the architecture for video quality evaluation tasks.
        - Task 4: Parallel Execution Strategy: Investigate parallel computing models and design a parallel execution strategy focusing on CPU and GPU architectures.
        - Task 5: Initial Prototyping: Create an initial prototype of the architecture in C++ with minimal features to validate the design.
        - Task 6: Performance Metrics: Implement profiling and benchmarking methods to record execution time and resource utilization on both CPU and GPU.
        - Task 7: Optimization: Analyze performance metrics and apply optimizations to the architecture for better performance and lower execution time.
    * Data Analysis
        - Profiling: Use profiling tools to identify bottlenecks and optimize them.
        - Comparative Analysis: Investigate the architecture's performance with existing solutions in the same domain.
    * Performance Metrics
        - Execution Time: Measure and compare the execution time in different CPU and GPU configurations.
        - Throughput: Measure the number of videos processed per unit of time.
        - Resource Utilization: Monitor the CPU and GPU resource utilization during execution
- Stage 2: Component development, unit testing and integrations
    * Specific objective: To detail the development, testing, and integration of components for a Rapid Evaluation of Perceptual Image Quality Estimator (RAPIQUE) using C++, OpenCV, and parallel execution for non-reference video quality assessment.
    * Tools and Technologies
        - Programming Languages: C++
        - Libraries: OpenCV, CUDA for GPU
        - Testing Frameworks: Google Test for C++
        - Version Control: Git
        - Profiling Tools: Python (gpuutils), NVIDIA Nsight
    * Tasks and Procedures
        - Task 1: Component Identification Identify the different components of RAPIQUE, such as data pre-processing, feature extraction, and quality score computation.
        - Task 2: Component Design Create UML diagrams or similar artifacts to model the components.
        - Task 3: Development Environment Setup Set up the development environment, including compilers, libraries, and testing frameworks.

        - Task 4: Component Development Implement individual components focusing on modularity and performance.
        - Task 5: Unit Testing Write unit tests for each component using TDD.
        - Task 6: Performance Profiling Use profiling tools to measure the CPU and GPU performance of each component.
        - Task 7: Component Integration Integrate components and ensure they work cohesively.
        - Task 8: Integration Testing Write tests to validate the integrated system.
        - Task 9: Benchmarking Benchmark the integrated system's performance in different configurations.
    * Performance Metrics
        - Execution Time: Measure the time taken by each component and the overall system on both CPU and GPU configurations.
        - Code Coverage: Aim for at least 90% with unit tests.
        - Throughput: Number of videos processed per unit time.
    * Data Analysis
        - Code Review: All codes will be peer-reviewed.
        - Static Analysis: Run static code analyzers to identify issues early.
        - Profiling Data: Use profiling data to identify bottlenecks and optimize.
        - Metrics Correlation: Study how the metrics correlate with human perception scores to fine-tune the components.
- Stage 3: Extracting scores and features from fast NR VQA metrics
    * Specific objective: To investigate the performance of extracting scores and features from various NR VQA metrics on platforms such as CPU and GPU. The focus will be on five VQA datasets: KoNViD-1k, LIVE-Qualcomm, LIVE Video Quality Challenge (VQC), YouTube-UGC and CVD2014. Each dataset will have a sample size of videos for this study.
    * Tools and Technologies Programming Languages: Python, MATLAB and C++
    * Libraries: OpenCV, TensorFlow, FFmpeg
    * Hardware: Various CPU and GPU architectures for performance evaluation
    * Data Preparation:
        - Data Collection: Extract a sample of videos from each of the datasets KoNViD-1k, LIVE VQC and YouTube UGC.
        - Data Annotation: Make sure each video comes with corresponding quality scores or annotations for supervised models.
    * Tasks and Procedures
        - Task 1: Determine the programming language in which the NR VQA metric is operational and its compatibility with Ubuntu and Windows.

- · Task 2: Install the necessary libraries for video quality prediction.
- · Task 3: Frame-by-frame extraction of videos from the selected datasets will be automated through a custom script.
- · Task 4: Extend or modify the existing codebase to store frame-by-frame video features. Calculate the arithmetic mean of these scores for the overall video quality.
- · Task 5: Adapt the code to store each frame's video quality features.
- · Task 6: Conduct a literature review on the technique of average pooling and incorporate this into the C++.
- · Task 7: Investigate and implement the best spatial-temporal grouping techniques based on existing literature.
- · Task 8: Study the theoretical foundations of VQpooling and implement it in C++.

* Runtime Analysis: Document each NR VQA metric's time to process feature videos on CPU and GPU. Resource Utilization: Measure CPU and GPU utilization rates during the computation.
* Measure CPU and GPU utilization rates during the computation.

– Stage 4: Train regressor or use CNN

* Specific objective: To train regressors or utilize Convolutional Neural Networks (CNNs) to predict perceptual video quality scores in a No-Reference Video Quality Assessment (NR VQA) setup. The primary goal is to examine the performance, execution times, and resource utilization on both CPU and GPU architectures.
* Tools and Technologies:
  - · Programming Languages: Python, MATLAB, C++
  - · Machine Learning Libraries: sci-kit-learn, TensorFlow, PyTorch
  - · Hardware: Specific CPU and GPU architectures for performance comparison
* Tasks and Procedures
  - · Task 1: Identify Regressors: Conduct a literature review to identify types of regressors that have shown promising results in similar contexts.
  - · Task 2: Theoretical Analysis and Literature Review: Dive deep into the mathematical and computational aspects of the chosen regressor types, comparing them with previous studies where such regressors were applied to NR VQA metrics.
  - · Task 3: Programming Language Identification: Decide on the programming language (Python, MATLAB, or C++) based on compatibility, library support, and performance considerations.
  - · Task 4: Execution and Time Analysis: Implement and execute the selected regressor or CNN model. Special focus will be placed on tracking the execution time for reading video quality prediction features and subsequent analysis

    * Performance Metrics
        · Runtime Analysis: The time taken for CNN, training the regressor, and making predictions will be measured and compared across CPU and GPU architectures.
        · Resource Utilization: Document CPU and GPU utilization rates during the computation.
        · Time Complexity Analysis: Use Big-O notation to describe the algorithmic time complexity.

- Stage 5: Validation and comparison performance of NR VQA metrics
    * Specific objective: To validate and compare the performance of various No-Reference Video Quality Assessment (NR VQA) metrics, emphasizing the execution time on both CPU and GPU architectures. The performance will be assessed using Pearson Linear Correlation Coefficient (PLCC), Spearman Rank Correlation Coefficient (SROCC), and Root Mean Square Error (RMSE).
    * Tools and Technologies
        · Programming Languages: Python, MATLAB, C++s
        · Hardware: Different models of CPUs and GPUs for benchmarking
    * Tasks and Procedures
        · Task 1: Execution Time Measurement: Implement timing functions or use existing timing libraries to calculate the program execution time for each NR VQA metric.
        · Task 2: Code Profiling: Identify the functions or lines of code that consume the most time during execution, using profiling tools specific to the chosen programming language.
        · Task 3: Function Behavior Analysis: Perform an in-depth study of the functions and statistical methods used in NR VQA to obtain video quality prediction scores. Examine how these algorithms behave in terms of execution time resource utilization.
        · Task 4: Theoretical Foundation Implementation: Understand the underlying theory of PLCC, SROCC, and RMSE and implement these methods in Matlab to validate and compare the NR VQA metrics.
        · Task 5: Data Analysis: Conduct a comprehensive analysis of the results obtained through PLCC, SROCC, and RMSE. Utilize statistical tests to verify the reliability and validity of the metrics.
    * Performance Metrics
        · PLCC, SROCC, RMSE: Evaluate NR VQA metrics using these statistical measures to understand their effectiveness and reliability.
        · Execution Time: Record and compare the execution time of each NR VQA metric on CPU and GPU architectures.
        · Resource Utilization: Monitor CPU and GPU usage during metric calculations.

- Stage 6: Writing and reporting results

  The methodology should incorporate a dedicated stage for documenting and disseminating the research findings. This documentation must adhere to the IEEE formatting guidelines and comply with the structural requirements necessary for publication

Chapter 8

System Requirements

This section outlines the essential system requirements, categorized by functionality, to support the RAPIQUE system. These requirements facilitate the system's ability to handle video data (mp4) in various formats and resolutions through efficient preprocessing, feature extraction, and integration with robust software tools.

- **Preprocessing and feature extraction**
  - **FFmpeg integration:** The system leverages FFmpeg for advanced video decoding, which is crucial for converting video streams into frames. This conversion enables rapid access to video content, a necessary step for quality assessment processes (Tomar et al., 2022). FFmpeg's adaptability allows for transcoding videos into multiple formats and resolutions, accommodating diverse client needs (ffm, 2024). Utilizing predefined profiles, FFmpeg optimizes the transcoding process for specific device requirements (ffm, 2024). It also handles audio transcoding, ensuring synchronization with video streams (ffm, 2024). FFmpeg supports container conversion without modifying encoded streams, maintaining compatibility across devices and browsers (ffm, 2024). Its frame buffering capabilities ensure synchronization between audio and video during pseudo-real-time processing (ffm, 2024). Robust error management enhances system stability by addressing specific transcoding issues, and continuous monitoring of FFmpeg's activities ensures operational integrity. Regular updates to FFmpeg align the system with the latest video processing standards and technologies, incorporating new features and bug fixes to enhance performance (ffm, 2024).
  - **Advanced feature extraction:** The system utilizes OpenCV integrated with C++ to implement advanced spatial feature extraction algorithms, essential for non-reference video quality evaluation (Zelinsky, 2009). OpenCV provides pre-built algorithms such as Histogram of Oriented Gradients (HOG) and Oriented FAST and Rotated BRIEF (ORB), which are critical for detecting and describing local features in video frames (Zelinsky, 2009). The integration with C++ optimizes these computationally intensive processes, allowing efficient handling of large volumes of video data (Zelinsky, 2009). The modular structure of OpenCV supports flexible implementation of additional algorithms, ensuring the system can adapt to emerging requirements or performance enhancements (Zelinsky, 2009). HOG captures edge or gradient structures within video frames, providing information on visual

quality (Dalal and Triggs, 2005). ORB offers rotation invariance and resistance to noise, making it suitable for quality assessment in varied video scenarios (Rublee et al., 2011). Applying the factory method design pattern allows dynamic creation of various feature extractors, enhancing the system's flexibility and maintainability (Gamma et al., 1994).

– **Development details**

  * **Integration with Machine Learning Libraries:** The system incorporated Scikit-learn to support advanced regression and classification tasks within the quality prediction model, facilitating sophisticated data analysis and pattern recognition, thereby enhancing the model's accuracy in predicting video quality (Pedregosa et al., 2011). Additionally, LibTorch, the C++ library for PyTorch, was integrated to implement state-of-the-art CNN models essential for extracting complex features from video data, indicative of perceptual quality (Paszke et al., 2019).
  * **Using C++:** C++ was selected for its system-level capabilities and efficient memory management, leveraging its execution speed and low-level control, which are advantageous in scenarios requiring pseudo-real-time processing and handling high volumes of data—common challenges in video quality assessment (Stroustrup, 2013). The utilization of C++ enabled the RAPIQUE system to operate with enhanced performance efficiency, critical for managing intensive computations and large-scale video data processing, ensuring system stability and efficiency. Direct control over hardware resources optimized calculations and memory use, vital for maintaining high performance and reliability under extensive operational demands. The native C++ interface of LibTorch allowed seamless integration with the existing codebase, minimizing the overhead of language interfacing and ensuring efficient functioning of machine learning components within the overall architecture.
  * **Application of the Factory Method Design Pattern:** The Factory Method design pattern was applied to enhance the system's runtime flexibility and maintainability. It allowed quality prediction algorithms to be interchangeable at runtime, adapting the model to various video content types and specific accuracy requirements. By employing the Factory Method design pattern, the system's separation of concerns and encapsulation were improved, facilitating easier maintenance and extension, allowing new prediction strategies to be added or modified independently of the rest of the codebase.
  * **Enhanced Processing with GPU Acceleration:** The system utilized CUDA for GPU acceleration, significantly increasing the speed and efficiency of computations. This enabled parallel processing of data on the GPU, vastly outperforming CPU-only computations and meeting the demands of contemporary digital media environments (Corporation, 2011). OpenCV's support for CUDA was utilized to perform image processing operations on the GPU, accelerating the feature extraction process from video frames and enhancing the speed and accuracy of quality predic-

tion (Zelinsky, 2009). Parallel programming constructs such as `cv::parallel_for` and `#pragma parallel for` were employed to facilitate efficient parallel processing of image data across multiple frames simultaneously, reducing processing time and increasing throughput. To ensure data consistency and prevent race conditions, mutexes (`std::mutex`) were utilized. Along with `std::for_each`, this approach ensured thread-safe operations on video data, maintaining reliable system operation under concurrent access scenarios.

- **Data storage and management**
  - **XML format utilization:** The system utilizes the XML format for storing extracted video features, providing a flexible and universally accepted data format that aligns with the RAPIQUE system's requirements (Ray and Ray, 2004). XML was chosen for its capability to offer a highly structured method of data representation, essential for managing complex data types involved in video quality assessment. The hierarchical structure of XML effectively encapsulates the nested and multifaceted data from video analyses, such as spatial features, temporal characteristics, and metadata, facilitating streamlined data access and manipulation (Harold and Means, 2004).
    A significant advantage of using XML is its dual capability of being readable by both humans and machines. This ensures that while the system can effortlessly parse and process the data, human operators or developers can manually inspect and modify the data as needed, which is invaluable for debugging and verifying system operations (Ray and Ray, 2004).
    The integration of XML with C++'s dynamic memory management capabilities enables the RAPIQUE system to efficiently manage memory resources. XML data is dynamically loaded and unloaded as required, reducing the memory footprint when handling large datasets. This strategy is critical for maintaining high performance and stability, minimizing the risk of memory leaks, and ensuring efficient data processing (Stroustrup, 2013).
    The XML structure specifically handles the serialization of multidimensional matrices, crucial for representing the feature vectors extracted from video data. Here's an in-depth breakdown of the XML structure:
    - <opencv_storage>: The root element of the XML document, indicating that the data stored within is managed using OpenCV's data structures.
    - <FeatFrames>: Represents a matrix of feature frames where each row corresponds to a different video or segment, and each column represents a specific feature extracted from the video.
    - type_id="opencv-matrix": Specifies that the data structure is an OpenCV matrix, efficiently handling large sets of numerical data.
    - <rows>and <cols>:
    - <rows>: Indicates the number of videos or video segments processed, represent-

ing feature data for multiple items.

- <cols>: Refers to the number of features extracted from each video, totaling 3884 distinct features covering various aspects of video quality such as texture, color, and motion.
- <dt>: Denotes the data type of the matrix elements.
- d: Represents a double-precision floating-point format in OpenCV, ensuring high numerical precision for the feature data.
- <data>: Contains the actual feature data in a flattened format. Each value represents a feature extracted from the videos, serialized in a row-major order, which is crucial for reconstructing the feature matrix during processing or analysis.

– **User interface and interaction**

* **User Interface Development with Tkinter:** The user interface, developed using Tkinter, provided a simple yet powerful toolkit for building graphical user interfaces in Python. Additionally, GPUutils was employed to offer practical utilities for monitoring hardware utilization, supporting the system's requirements for interactivity and performance analysis (Lutz, 2013).

  Tkinter facilitated the rapid development and deployment of GUI applications, which was particularly advantageous for the RAPIQUE system. This allowed for the creation of a straightforward interface that displayed complex data such as GPU and CPU usage, expected time for feature extraction completion, database path selection, output path selection, and detailed logs of execution times and video names (Lundh, 2005). As a standard GUI toolkit included with Python, Tkinter ensured compatibility across multiple operating systems without the need for additional installations or configurations, making the RAPIQUE system accessible to users on various platforms (Lutz, 2013).

  Tkinter's customization and flexibility were leveraged to develop a highly tailored GUI. It was integrated with other Python libraries to meet specific needs, such as embedding graphs for hardware utilization and lists for displaying pseudo-real-time processing data. GPUutils provided essential tools for pseudo-real-time monitoring of GPU and CPU utilization, crucial for understanding system performance and managing resources effectively. The seamless integration of GPUutils with Python allowed for easy embedding of hardware monitoring data into the GUI, providing users with up-to-date information on system performance.

* **Enhancing User Experience and System Performance:** An interactive and informative dashboard was developed using Tkinter to display critical information such as GPU and CPU usage, estimated times for tasks, and detailed processing logs. This interactivity significantly enhanced the user experience by keeping users informed about the system's status and progress. Monitoring hardware utilization

with GPUutils enabled the system to provide feedback for performance optimization. For example, if GPU utilization was consistently low, adjustments could be made to the video processing workload or system settings to better utilize available hardware resources.

Figure 8.1 presents the user interface of RAPIQUE (C++ version) designed to execute and monitor the performance of a video quality assessment program. This interface provides critical feedback and controls for processing video files to evaluate their quality.

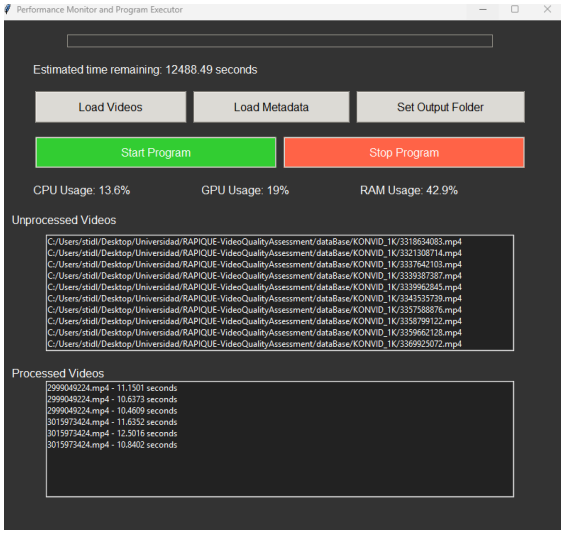

Figure 8.1: Performance Monitoring and Program Execution Interface for RAPIQUE Video Quality Assessment.

* **Estimated Time Remaining:** The interface prominently displays the estimated time remaining for the current batch of video processing tasks, which helps users manage their time efficiently while the software processes videos.
* **Control Buttons:**
    - Load Videos: Initiates the process of loading video files into the program (see Figure 8.2).
    - Load Metadata: Allows the user to load additional data that may be necessary for processing, such as video metadata (see Figure 8.3).
    - Set Output Folder: Enables users to specify the directory where the processed video files and data will be saved (see Figure 8.4).
    - Start Program: Begins the video processing tasks.
    - Stop Program: Provides the ability to halt processing, useful in cases where issues are detected or adjustments need to be made.

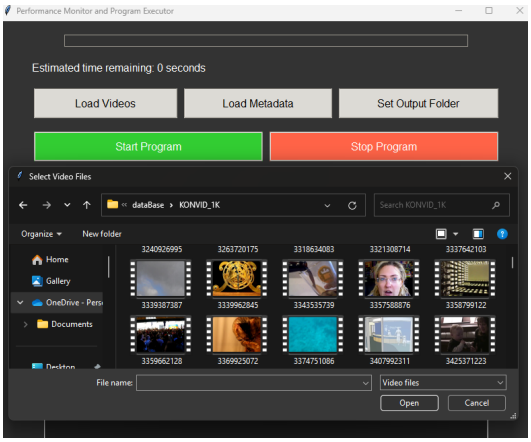

Figure 8.2: Graphical interface of the C++ version of RAPIQUE. It indicates the path to choose the videos to be processed in order to extract their features.

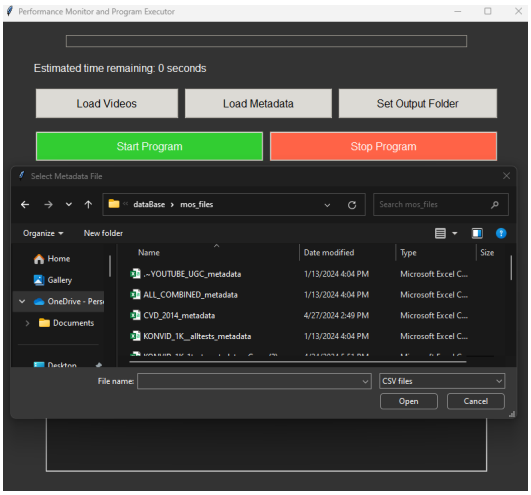

Figure 8.3: Graphical interface of the C++ version of RAPIQUE. It indicates the path to choose the metadata of video files (width, height, video name, number of frames, and frame rate) to be processed in order to extract their features.

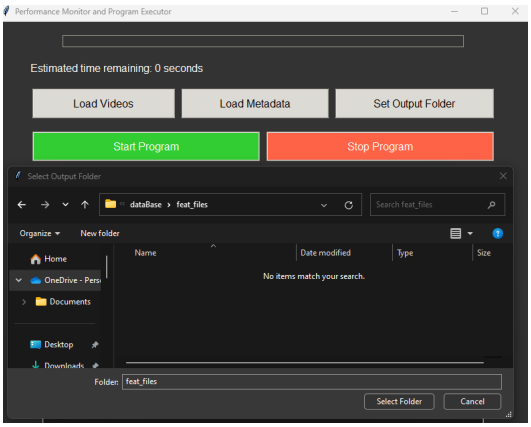

Figure 8.4: Graphical interface of the C++ version of RAPIQUE. Indicates the path to save the features extracted from the videos.

* **Resource Usage Metrics:**
    - CPU Usage: Indicates the percentage of the central processing unit's capacity being utilized.
    - GPU Usage: Reflects the graphics processing unit's engagement in tasks, likely related to video processing that involves graphical data.
    - RAM Usage: Shows the percentage of the available system memory (RAM) being used, which could include loading videos into memory for processing.
* **Video Processing Lists:**
    - Unprocessed Videos: Lists video files queued for processing. Each file path indicates that the videos are sourced from a directory for the known database used in video quality research.
    - Processed Videos: Shows a list of videos that have already been processed, including the time taken to process each video. This log is crucial for analyzing the efficiency of the video processing, identifying any potential bottlenecks, and verifying that each file has been addressed.

– **System monitoring and performance enhancement**

* **Comprehensive Logging Strategy:** A comprehensive logging strategy was implemented to enhance system monitoring and performance by recording detailed operational data. This approach is crucial for identifying bottlenecks, optimizing processes, and ensuring system reliability (Gregoire, 2013).
* **Advantages of C++ Logging:**

- Detailed System Monitoring: The logging capabilities provided by C++ were leveraged to create detailed records of system operations. These records include execution times of specific functions, the number of iterations each function undergoes, and the total time required to process each video. Such detailed logging is instrumental in understanding system performance under varying conditions and is essential for ongoing system refinement (Stroustrup, 2013).

- Performance Optimization: Analyzing logs of function execution times and video processing durations allowed for pinpointing inefficiencies and bottlenecks. This data is invaluable for making informed decisions regarding optimization efforts, leading to significant enhancements in system performance.

* **Storage and Accessibility of Logs in XML Format:**

  - Structured Data Storage: XML was adopted for storing log data, enhancing both accessibility and readability. The structured nature of XML allows logs to be easily parsed by various tools, supporting automated analysis and manual review. This organized approach ensures rapid access and analysis of records as needed (Ray and Ray, 2004).

  - Integration with Analysis Tools: Using XML for log data facilitates seamless integration with a wide array of performance analysis tools. This integration has been instrumental in conducting detailed performance reviews and audits, thereby maintaining high levels of efficiency and reliability within the RAPIQUE system.

* **Use of XML for Archiving Logs:**

  - Ease of Data Retrieval: XML was utilized to archive log files, simplifying the retrieval of historical data essential for long-term performance analysis and system audits. Reviewing historical data enables the identification of trends and patterns not evident from short-term data alone (Harold and Means, 2004).

  - Compliance and Reporting: Using XML for logging aligns with best practices and compliance requirements in many operational environments. Detailed XML logs facilitate the creation of comprehensive reports and documentation necessary for regulatory reviews and internal audits, ensuring adherence to industry standards and practices.

* **XML Document Structure:** The XML structure encapsulates the performance statistics of each video analyzed by the RAPIQUE C++ implementation. This structured format allows for precise data management and straightforward retrieval,

crucial for detailed performance analysis. The hierarchy of the XML document is organized as follows:

- *VideoFunctionStats:* The root element containing all data.
- *Video:* Each video file analyzed is represented by a *Video* element, with a *name* attribute specifying the video file's name.
- *Function:* Represents a specific computational function within RAPIQUE, characterized by a *name* attribute denoting the function's identity.
- *ExecutionCount:* Indicates the number of times the function was executed.
- *AverageExecTime:* Records the average execution time (in seconds) for the function, reflecting computational efficiency and performance.

– **Tools for testing**

* Google Test provides a robust framework for C++ unit testing, while NVIDIA Nsight offers comprehensive capabilities for debugging and performance tuning of applications running on NVIDIA GPUs, making them indispensable tools for the system's development.
* **Use of Google Test for C++**
  - Ensuring Code Reliability: Google Test, a widely-used C++ unit testing library, allows developers to write comprehensive, automated tests. By integrating Google Test, the RAPIQUE system ensures each component functions correctly both individually and when integrated with other components. This is crucial for maintaining high reliability and stability, particularly when processing large volumes of video data under diverse conditions (goo, 2024).
  - Facilitating Test-Driven Development (TDD): Employing Google Test supports a Test-Driven Development approach, where tests are written before the actual code. TDD helps clarify requirements before coding begins, reducing errors and improving code quality. This method is especially beneficial in complex systems like RAPIQUE, where precise functionality and performance are critical (Beck, 2003).
* **Integration and Debugging Support:** NVIDIA Nsight provides advanced debugging capabilities that help identify and resolve issues at both the hardware and software levels. It integrates seamlessly with development environments, offering a detailed view of program execution, which is essential for effectively integrating C++, OpenCV, and CUDA components within the RAPIQUE system (Corporation, 2021).

  * **Continuous Integration and Regression Testing:** Integrating Google Test into the development pipeline enables continuous integration practices, where codebase changes are tested automatically, ensuring that additions or modifications do not introduce new bugs. This is particularly important in a system like RAPIQUE, which evolves continuously as new video processing techniques and algorithms are developed.
  * **Performance Monitoring and Tuning:** Using NVIDIA Nsight, the development team can continuously monitor the performance of new and existing features, ensuring that any changes to the code do not adversely affect the system's performance. This ongoing monitoring is crucial for maintaining an optimized system capable of handling intensive video quality assessments (Corporation, 2021).

This schematic in figure 8.5 provides a comprehensive overview of the workflow of the RAPIQUE model, structured into distinct functional blocks, each utilizing specific technologies:

- **Spatial and Temporal NSS Feature Extraction Branch**:
  - Description: The top block of the model diagram illustrates the extraction of spatial and temporal Natural Scene Statistics (NSS) features.
  - Technology: This process is implemented using C++ and leverages libraries such as OpenCV for efficient image processing.
- **CNN Feature Extraction Flow**:
  - Description: The bottom block represents the Convolutional Neural Network (CNN) feature extraction mechanism.
  - Technology: This segment utilizes libtorch, the PyTorch C++, facilitating the deployment of neural network components.
- **Feature Vector Concatenation and Regression**:
  - Description: The final feature vector is constructed by concatenating the extracted spatial and temporal NSS features with the CNN features. This comprehensive vector is then employed to train the regressor head of the model.
  - Technology: The integration and training processes are handled in C++, using FFmpeg for video data manipulation and libtorch for machine learning tasks.

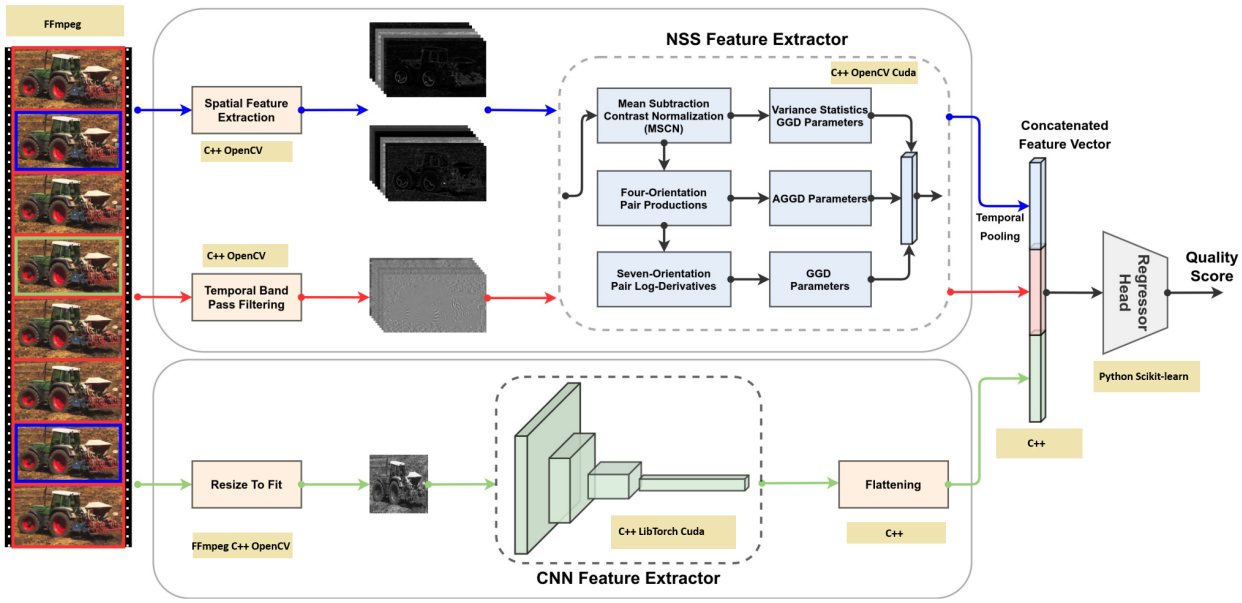

Figure 8.5: Workflow of RAPIQUE. The upper block shows the frame extraction branch, along with its spatial and temporal features, while the lower block represents the CNN feature extraction flow. (Tu et al., 2021b)

Chapter 9

System architecture overview

The RAPIQUE system architecture is meticulously designed across four hierarchical levels: system context, containers, components and code, ensuring a robust and scalable video quality assessment framework.

- **System Context:** Within a CUDA-enabled video processing environment, the RAPIQUE system is strategically positioned to facilitate user interactions for video quality assessments. The architecture seamlessly integrates with a sophisticated storage system, handling both video and feature data efficiently, catering to the dynamic needs of the video processing tasks (see Figure 9.1).

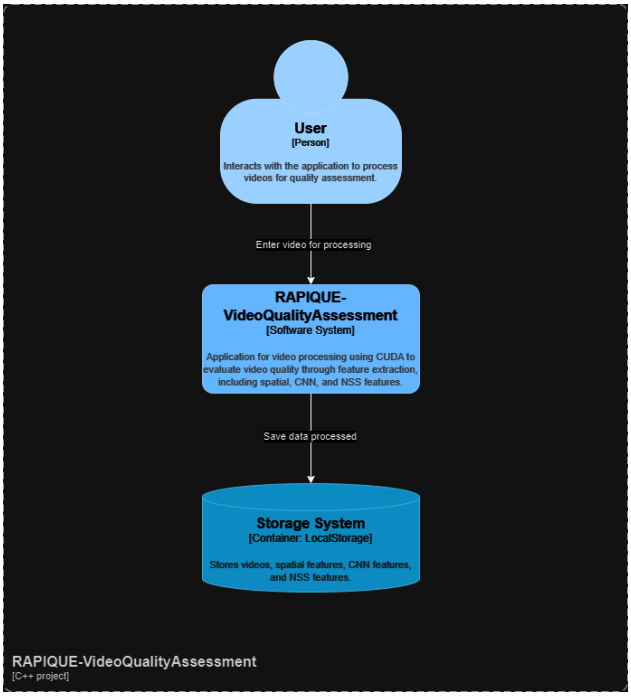

Figure 9.1: C4 Model Context Diagram for RAPIQUE: Interactions and Data Flows. The figure shows the main components of the system, the external interactions with end users and associated systems, as well as the data flows between these elements. Users interact with the application, which processes spatial, temporal and CNN characteristics in order to use them to evaluate image quality through an automatic learning model.

- **Containers** (see Figure 9.2): The **Video Reading Container** employs FFmpeg, integrated with C++, to decode video streams into YUV frames, setting the stage for subsequent processing. The **Processing Container** utilizes OpenCV and CUDA to extract spatial features and leverages PyTorch for deploying deep learning algorithms that extract both CNN and NSS features. The **Writing Container** focuses on the secure and efficient storage of processed features using C++ and CUDA technologies. The **Factories Container** implements the factory design pattern in C++, dynamically generating various processing objects to enhance the system's modularity and flexibility. Finally, the **Performance Monitoring Container** monitors and logs the execution times of various operations using C++, which is crucial for system optimization and performance enhancements.

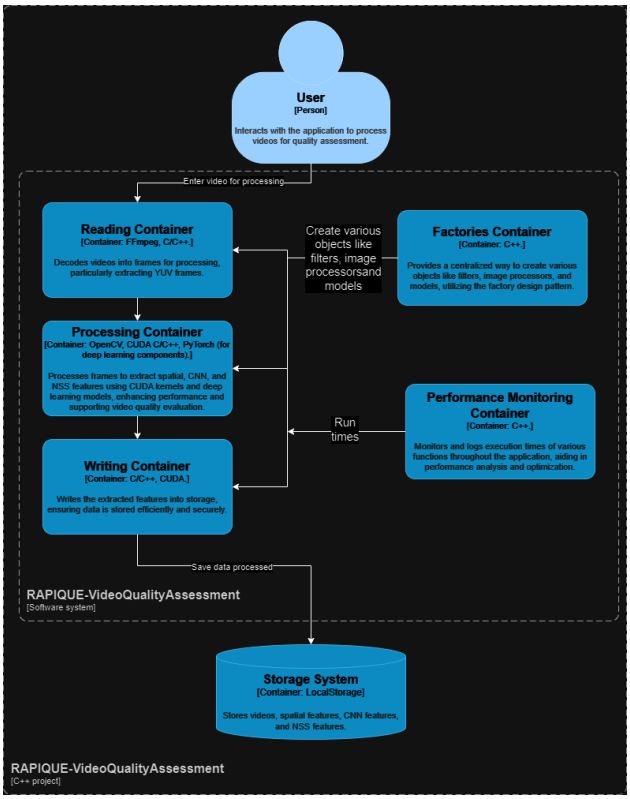

Figure 9.2: C4 Model Container Diagram for RAPIQUE: System Structure and Components.

- **Components**(see Figure 9.3): The **Video and Image Readers** in the Video Reading Container handle the efficient decoding and management of video streams. The **Frame Processor**, **CNN Feature Extractor**, and **NSS Feature Extractor** in the Processing Container apply advanced transformations and feature extractions using CUDA and deep learning models. The **Feature Writer** in the Writing Container methodically organizes and secures the storage of extracted features. The **Filter Factory** and **Model Factory** in the

Factories Container provide bespoke filter and model management solutions. Finally, the **Function Logger** and **Timer** in the Performance Monitoring Container meticulously track and log execution times to optimize performance.

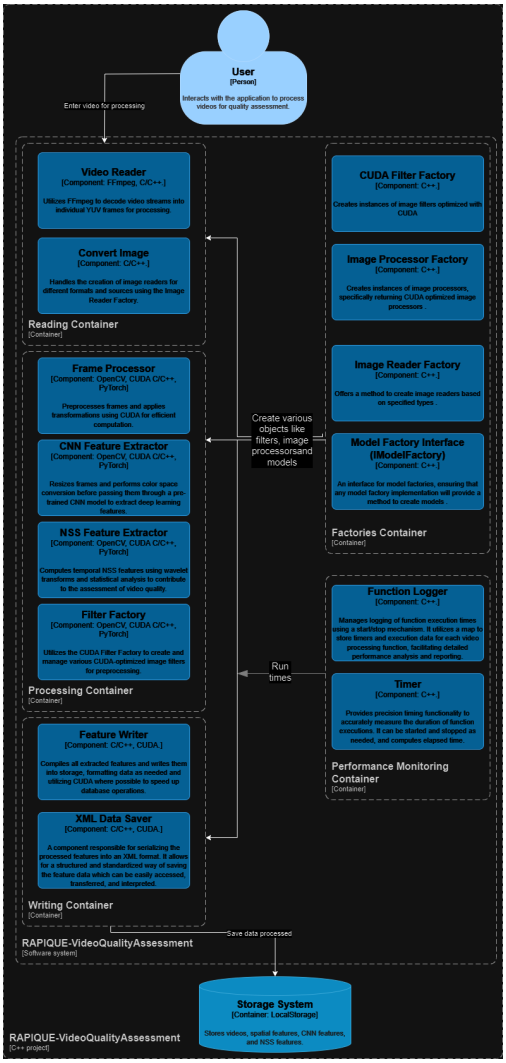

Figure 9.3: C4 Model Component Diagram for RAPIQUE: Implementation Details and Functionalities.

- **Responsibilities and Execution Flow: CNN Feature Extraction** involves resizing images to 224x224 pixels, converting the color space from BGR to RGB, normalizing pixel values, and using a pre-trained CNN to extract salient features, which are then integrated for enhanced video quality assessment. For **Temporal NSS Feature Extraction**, wavelet-based filtering techniques are employed to extract critical frequency components, followed by

statistical analysis to derive NSS features that are amalgamated with other data points for comprehensive quality evaluation. **Performance Monitoring** is conducted by the Function Logger, which initiates and terminates timers for each process, ensuring accurate logging and performance assessment.

- **Code:** Figure 9.4 provides a detailed breakdown of the key components, represented in a class diagram, along with their interactions. **CUDAFilterFactory** and **CUDAImage-Processor** handle image processing tasks with CUDA, enhancing performance by utilizing GPU resources. **CUDAFilterFactory** creates image filters, such as Gaussian filters, while **CUDAImageProcessor** applies operations like matrix multiplication to frames extracted from videos. The **ImageReaderFactory** provides image readers capable of reading and processing image data from various formats. The **Logger** monitors and logs system behavior, and the **Timer** measures the duration of function operations within the software. The **Main RAPIQUE Module (mainRAPIQUE)** acts as the orchestrator for the RAPIQUE analysis, coordinating `calc_RAPIQUE_features` and other functions to process video data, utilizing both CUDA and CPU-based processing to calculate features from video frames. **Feature Calculation and Model Loading** utilizes deep learning models (ResNet-50) and temporal pooling algorithms to extract features from videos, where `calc_RAPIQUE_features` calculates features such as AGGD parameters and spatial/temporal NSS features. Finally, **RAPIQUE_spatial_features** and **rapique_basic_extractor** implement specific functions like `gen_DoG` (Difference of Gaussian) for feature extraction, detecting edges and textures in video frames.

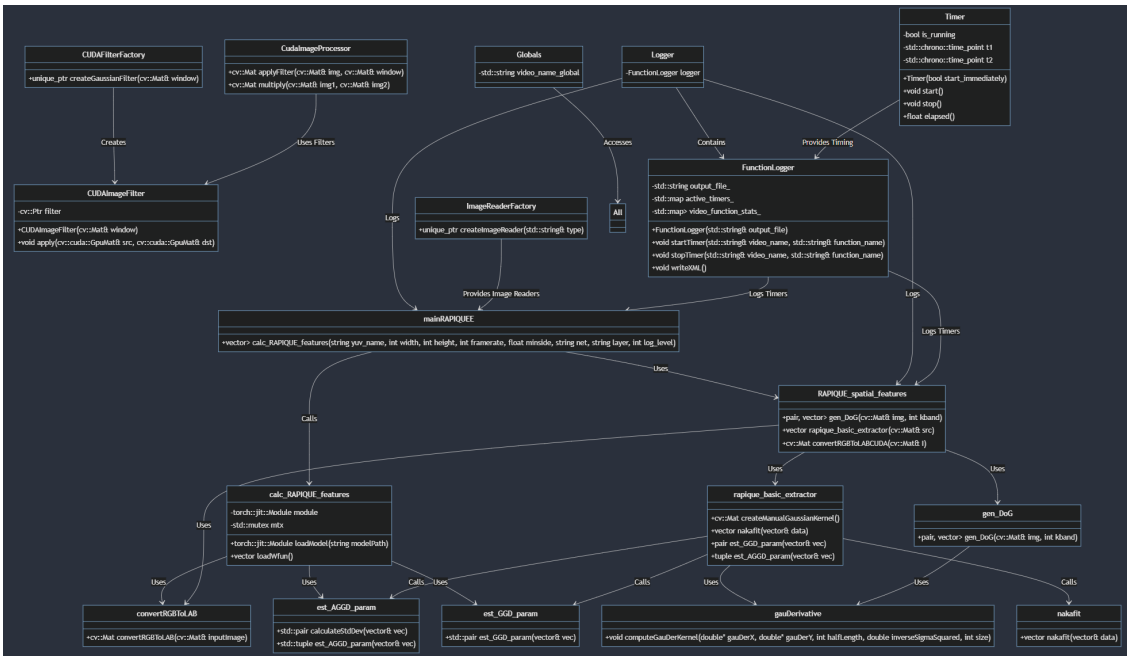

Figure 9.4: C4 Model Code Diagram for RAPIQUE: Implementation class diagram.

Chapter 10

Development

10.0.1 SpatialNSS

The RAPIQUE system leverages a sophisticated model for extracting spatial features from video frames, which are pivotal for assessing video quality. This model, known as NSS-34, utilizes Natural Scene Statistics (NSS) derived from models such as the Generalized Gaussian Distribution (GGD) and the Asymmetric Generalized Gaussian Distribution (AGGD). The NSS-34 feature set is designed to capture comprehensive statistical measures from images or video frames, applying these in a chromatic space to improve the accuracy and efficiency of quality assessment (Tu et al., 2021b). The extraction of these features involves:

- **Gradient Magnitude Computation:** Using Sobel filters to determine the gradient magnitude (GM) of video frames, which helps highlight essential texture and edge details indicative of quality.
- **Multi-scale Filtering:** Employing Gaussian and Difference of Gaussian (DoG) filters to mimic the biological visual processing, allowing for the detection of significant edges and contours at multiple scales.
- **Color Processing:** Transforming RGB images to more perceptually relevant color spaces such as LAB, utilizing opponent color models to extract chromatic features that are critical for video quality.
- Key Components and Functionalities
  - **CUDA Acceleration:** The implementation leverages CUDA to enhance image processing performance significantly. By utilizing CUDA streams for asynchronous operations, CUDA filters for spatial transformations, and CUDA color conversion functions, the code ensures that the processing is highly efficient and optimized for GPU execution, reducing computational overhead and enhancing throughput.
  - **Color Space Conversion:** Essential to the processing pipeline is the conversion of RGB images to the LAB color space, facilitated by CUDA-accelerated functions. The LAB color space is preferred due to its alignment with human color perception, making it invaluable for accurately assessing visual quality in videos.
  - **Gradient Magnitude Computation:** To capture detailed texture and edge information, a fundamental indicator of visual quality, the code applies a Sobel filter to the

grayscale image. This operation computes the gradient magnitude (GM), a critical step executed on the GPU for enhanced performance.

- **Gaussian and Difference of Gaussian (DoG) Filtering:** These filtering steps are pivotal for feature extraction. The Gaussian filter primarily smooths the image to reduce noise and fine detail, while the DoG filter, acting as a bandpass filter, accentuates significant edges and contours at multiple scales. This dual filtering approach mimics biological visual processing mechanisms and is crucial for extracting perceptually relevant features.

- **Feature Extraction:** Utilizing the outputs from the Gaussian and DoG filters along with the computed GM, the module extracts a comprehensive set of 680 spatial features. These features encompass statistical measures from MSCN transformations, variance fields, and models fitted to the GGD and AGGD, as prescribed by the theoretical framework.

- Development Workflow

  - **Initialization:** The procedure starts by verifying that the input image is in RGB format. It then initializes the required CUDA operations to prepare for high-performance image processing tasks.

  - **Color and Gradient Processing:** The core of the processing involves converting the RGB image to grayscale on the GPU, followed by computing gradient magnitudes using Sobel filters. The image is subsequently subjected to Gaussian and DoG filtering to extract both high-frequency and mid-frequency spatial features essential for quality assessment.

  - **Feature Calculation:** Post-filtering, the image undergoes a reverse transformation to the RGB color space and then to the LAB color space. This step is vital for calculating color-dependent features, including the orientation channels O1 and O2, which capture different aspects of color perception critical for quality evaluation.

  - **Parallel Feature Scaling and Extraction:** To optimize computational efficiency, the extracted features are scaled and processed in parallel. This involves resizing the feature maps to various scales and extracting features at each scale, enabling a multiscale analysis of image quality which is more reflective of human visual perception.

  - **Error Handling:** Robust error handling mechanisms are implemented to manage exceptions effectively during the feature extraction process. This ensures the stability and reliability of the module, preventing crashes and ensuring consistent output across diverse video inputs.

10.0.2 SpatialNSS-Var

SpatialNSS-Var, captures the temporal dynamics of video quality by analyzing the forward progression of spatial features in subsequent video frames. This component is critical for anticipating changes in video quality and for applying temporal pooling methods like average and absolute difference pooling. These methods enrich the feature set with temporal information, providing a more dynamic and comprehensive analysis of video quality over time. The comparison between SpatialNSS and SpatialNSS-Var allows the RAPIQUE system to detect and predict potential quality fluctuations due to motion or scene changes effectively (Tu et al., 2021b). The code operates on triplets of frames extracted from video sequences: the current, the previous (utilized in SpatialNSS), and the next (utilized in SpatialNSS-Var). These frames are first converted from YUV to RGB color space, then subjected to spatial feature extraction (same development workflow of SpatialNSS) using the RAPIQUE_spatial_features function. This function, involves converting RGB frames to grayscale, applying Sobel filters to compute gradient magnitudes, and further processing to extract a comprehensive set of 680 spatial features from each frame.

10.0.3 TemporalNSS

The TemporalNSS model employs a series of one-dimensional temporal bandpass filters to analyze the temporal dynamics of video sequences, which is essential for assessing quality in user-generated content (UGC) videos. These videos often exhibit significant frame-to-frame variations due to varying recording conditions and dynamic content.

- Temporal Bandpass Filtering: The core of the TemporalNSS model lies in its use of one-dimensional temporal bandpass filters that dissect the video into different temporal frequencies (Tu et al., 2021b). These filters are crucial for:
  - Isolating behaviors in the video that occur at varying speeds.
  - Providing a nuanced view of temporal dynamics essential for high-quality video assessment.
  - Capturing a broad spectrum of temporal information from slow to rapid changes, allowing for a detailed temporal analysis.
- Frame Processing and Feature Extraction: Frame processing is conducted on triplets (previous, current, and subsequent frames), enabling the system to understand the temporal transitions and anticipate future states. This process includes:
  - **Color Space Conversion:** Converting each frame from YUV to RGB to better align with human visual perception.
  - **Spatial Feature Extraction:** This involves several transformations:

    * Converting RGB frames to grayscale to simplify the extraction of texture and edge details.

    * Applying Sobel filters to extract gradient magnitudes, highlighting essential textural and edge details indicative of video quality.

    * Using Gaussian and Difference of Gaussian (DoG) filters to enhance edge detection at multiple scales and mimic biological visual processing mechanisms.

- Temporal Feature Synthesis: The TemporalNSS synthesizes the processed frames into a feature set that encapsulates both spatial and temporal dimensions of video quality. This includes:

  - **Bandpass Coefficient Analysis:** Computing the mean and standard deviation of bandpass coefficients across the frames using Mean Subtracted Contrast Normalization (MSCN).

  - **Feature Pooling and Integration:** Pooling temporal features from multiple frames using techniques like averaging and absolute difference pooling to create a comprehensive representation of video quality over time.

- Implementation Details:

  - **Wavelet Transform Frame Processing:**

    * Frames are processed using a wavelet transform approach, filtered through a predefined matrix of wavelet functions to capture various frequency components essential for analyzing temporal variations.

    * Frames are selected based on their temporal position and resized according to a predefined ratio to maintain consistency in feature extraction scale.

  - **Parallel Processing for Efficiency:**

    * Utilizing OpenCV's `parallel_for_` construct to apply wavelet filters across multiple frames simultaneously, speeding up the computation.

    * Features are extracted and scaled in parallel to ensure that features are captured at multiple scales, reflecting the multiscale nature of human visual perception.

10.0.4 CNN

- Model Loading and Setup: The system begins by loading a pre-trained CNN model, specifically the ResNet-50, due to its depth and efficiency in handling complex image features at scale (Tu et al., 2021b). The model loading process is as follows:

  - The model is loaded from a specified path using PyTorch's `torch::jit::Module`. This module facilitates the use of models trained in Python directly in C++ environments.
  - The model is set to run on CUDA-enabled devices to leverage GPU acceleration, enhancing the processing speed significantly.

- Tensor Operations and Feature Processing: Once the model is loaded, the video frames are processed to extract quality-relevant features:

  - Each frame is converted into a tensor, matching the input requirements of the CNN. This involves resizing the frame to 224x224 pixels, the standard input size for ResNet-50, to maintain consistency with the model's training data.
  - The tensor undergoes a series of transformations, including permuting dimensions to align with the model's expectation (channels first format).
  - The model evaluates the tensor without gradient calculations, optimizing performance and memory usage.
  - The output from the CNN is processed to extract a flattened feature vector. This vector is derived from the mean of the output tensor across spatial dimensions, focusing on capturing the most salient features that relate to video quality.

- Optimization and Efficiency Considerations: To accommodate the high-resolution frames typically encountered in video quality analysis, the frames are aggressively downscaled before being fed into the CNN. This step ensures that the computational load remains manageable while still capturing essential semantic details:

  - The use of global average pooling (GAP) or spatial pyramid pooling (SPP) techniques within the CNN architecture helps to manage the varying dimensions of input video frames, further optimizing the feature extraction process.
  - This approach allows the system to handle full-sized images efficiently, aligning the processing capabilities with the real-world demands of UGC video assessment.

Integration and Final Feature Synthesis:

- Aggregating features from all processed frames into a final feature vector that combines all processed data points (providing one vector per video of 1x3884) (Tu et al., 2021b), providing a comprehensive representation of both the spatial and temporal attributes of the video quality.
- Ensuring efficient use of resources by releasing GPU memory immediately after processing, preventing memory leaks and ensuring system stability.

Chapter 11

Results

11.0.1 Evaluation

In this study, we extracted features using the RAPIQUE algorithm, implemented in both Matlab and C++, from a comprehensive dataset that includes the following video collections:

- KoNViD-1k (1,200 videos)
- LIVE-Qualcomm (208 videos)
- CVD2014 (234 videos)
- YouTube UGC (1,045 videos)
- LIVE Video Quality Challenge (VQC) (585 videos)

To extend the scope of our initial project proposal, we also evaluated an additional dataset comprised of a randomly selected mixture of 200 videos from the aforementioned databases.

To ensure a proportional selection of videos from each database, we employed a size-proportional sampling method, which involved:

1. Calculating the total number of videos across all databases:

$$\text{Total} = 1,045 + 208 + 585 + 234 + 1,200 = 3,272 \text{ videos}$$

2. Determining the fraction of the total that each database represents, and multiplying each fraction by 200 to achieve a proportional distribution:

$$\begin{aligned}
\text{YouTube UGC:} &\quad \frac{1,045}{3,272} \times 200 \approx 64 \text{ videos} \\
\text{LIVE-Qualcomm:} &\quad \frac{208}{3,272} \times 200 \approx 13 \text{ videos} \\
\text{LIVE VQC:} &\quad \frac{585}{3,272} \times 200 \approx 36 \text{ videos} \\
\text{CVD2014:} &\quad \frac{234}{3,272} \times 200 \approx 14 \text{ videos} \\
\text{KoNViD-1k:} &\quad \frac{1,200}{3,272} \times 200 \approx 73 \text{ videos}
\end{aligned}$$

These figures were rounded to ensure the sum totaled exactly 200 videos.

Subsequent steps included modifying the algorithms to accept videos in MP4 format, which were then converted to YUV 420 format using FFmpeg, with careful consideration to preserve the integrity of the outcome. The extracted video information was stored in a Mat file for the Matlab algorithm and in XML format for the C++ algorithm, for subsequent use.

The feature maps, which serve as inputs to the Support Vector Regression (SVR) along with subjective scores, are sized at 1×3884 per video. These maps are detailed further in the table 11.1.

Main Module	**Dimensions**
SpatialNSS	1x680
SpatialNSS-Var	1x680
TemporalNSS	1x476
CNN	1x2048
RAPIQUE (Full model) c++ and matlab version	1x3884

Table 11.1: Module Dimensions

The implementation of Support Vector Regression (SVR), which is well-suited for handling high-dimensional data such as feature vectors extracted from the RAPIQUE metric, was conducted using Python. This implementation utilized specific libraries, notably `scikit-learn`, to perform hyperparameter tuning through `RandomizedSearchCV`. The dataset was partitioned into an 80% subset for training and a 20% subset for testing, replicating this split across twenty iterations to optimize the hyperparameters of the SVR that best predict the Mean Opinion Score (MOS) based on perceptual quality inputs.

A repeated train-test split was executed twenty times to assess the robustness of the SVR model in predicting video quality. For each iteration, a randomized search for hyperparameters with cross-validation was conducted on the training set to identify the optimal settings for 'C' and 'γ' for an SVR, or 'C' and 'ε' for a LinearSVR, contingent upon the dimensionality of the features:

Low Dimensionality (≤ 4000 features): An SVR with an RBF kernel was employed. A hyperparameter grid was defined where 'C' varied on a logarithmic scale of base 2 from 2^1 to 2^{10}, and 'γ' also varied on a logarithmic scale of base 2 from 2^{-8} to 2^1. `RandomizedSearchCV` was utilized to conduct a randomized search across this parameter grid, aiming to find the values that maximize model performance over a three-fold cross-validation scheme.

High Dimensionality (> 4000 features): LinearSVR, which utilizes a linear kernel and is more efficient in high-dimensional spaces, was used. The hyperparameter grid for LinearSVR included 'C', ranging over a list of specific values (0.001, 0.01, 0.1, 1, 2.5, 5, 10), to cover a broad spectrum of potential model behaviors in regulating complexity and error margin, and 'ε', which varied over the same list of values. Again, `RandomizedSearchCV` was employed to search for the best combination of 'C' and 'ε'.

Subsequently, metrics such as the Spearman Rank Correlation Coefficient (SRCC), the Pearson Linear Correlation Coefficient (PLCC), and the Root Mean Squared Error (RMSE) were recalculated using a logistic function to fit the predictions before comparing them to human scores. This approach enhances the evaluation of model performance by aligning predicted quality assessments with perceived human judgments, thereby providing a more accurate measure of video quality assessment.

A four-parameter logistic function is employed to map the predicted quality values from the model to a scale that more closely aligns with the actual scores provided by humans. Below, I detail how this logistic function operates and its implementation in the code.

The logistic function used in this study is defined as follows (Hosmer Jr et al., 2013):

$$y = \beta_2 + \frac{\beta_1 - \beta_2}{1 + \exp\left(-\frac{x-\beta_3}{|\beta_4|}\right)}$$

Where:

- β_1 represents the theoretical maximum that the dependent variable can reach.
- β_2 is the theoretical minimum.
- β_3 denotes the midpoint of the logistic curve, i.e., the value of x for which y is the average of β_1 and β_2.
- β_4 is a scale parameter that determines the steepness of the curve.

This parameterization is crucial as real Mean Opinion Score (MOS) values are typically nonlinear and can have a limited range and a skewed distribution, which is common in subjective video quality data. By fitting a logistic function, a normalization and scaling of the model's predictions are performed to correspond more closely with the true nature of human perceptions, thus enhancing the accuracy and utility of the model in real-world applications.

To accurately compare the execution times between the original Matlab version and the proposed C++ version of the RAPIQUE metric, two distinct computing systems were utilized, each with specific hardware configurations:

- PC1 Specifications:
  - Processor: Intel Core i7-8750H, operating at 2.2 GHz
  - Memory: 8 GB RAM
  - Graphics: Nvidia GTX 1050, with 4 GB of dedicated VRAM
- PC2 Specifications:
  - Processor: 13th Generation Intel Core i9-13900KF, operating at 3000 MHz
  - Memory: 32 GB RAM
  - Graphics: NVIDIA GeForce RTX 4070, with 12 GB of dedicated VRAM

11.0.2 Outcome metrics

The performance metrics of the RAPIQUE model, implemented in both C++ and Matlab, across five video quality assessment (VQA) datasets, are comprehensively detailed in Tables 12.1, 12.2, 12.3, 12.4, and 12.5. To extend the depth of this study and provide a more robust analysis of the model's performance, standard deviations for the Pearson Linear Correlation Coefficient (PLCC), the Spearman Rank Correlation Coefficient (SRCC), and the Root Mean Squared Error (RMSE) were computed. Additionally, the Kendall Rank Correlation Coefficient (KRCC) was included to offer a more nuanced understanding of the ordinal relationships within the data.

- Importance of Including Standard Deviations and KRCC:
  - Standard Deviations of PLCC, SRCC, and RMSE: The inclusion of standard deviations for PLCC, SRCC, and RMSE provides insights into the variability and consistency of the model's performance across different datasets and test conditions. This statistical measure is crucial for understanding the reliability of the model under various scenarios, which is particularly important in fields such as video quality assessment where environmental and encoding variations can significantly impact perceived quality (Sheskin, 2007).
  - Kendall Rank Correlation Coefficient (KRCC): KRCC is utilized to assess the ordinal association between two sets of data. By measuring the degree of correspondence between the rankings of data points in two datasets, KRCC offers a less sensitive alternative to the SRCC that is more robust to outliers and skewed data distributions. This is particularly advantageous in video quality assessment, where data distributions can often be non-normal, thus requiring robust statistical methods to accurately capture the underlying relationships (Gilpin, 2011).

Metric	CVD 2014			
	SRCC	KRCC	PLCC	RMSE
RAPIQUE (Matlab PC1)	0.8145 ± 0.0376	0.6237 ± 0.040 96	0.8238± 0.0385	11.972 ± 1.201
RAPIQUE (Matlab PC2)	0.8120 ± 0.0603	0.6225 ± 0.066 92	0.8365± 0.0498	11.544 ± 1.822
RAPIQUE (C++ PC1)	0.8072 ± 0.0558	0.6161 ± 0.060 02	0.8338± 0.0490	11.601 ± 1.596
RAPIQUE (C++ PC2)	0.7943 ± 0.0630	0.6044 ± 0.070 31	0.8129± 0.0726	12.110 ± 2.082

Table 11.2: Performance Metrics (± Standard Deviation) for RAPIQUE on CVD 2014 Dataset

Metric	LIVE VQC			
	SRCC	KRCC	PLCC	RMSE
RAPIQUE (Matlab PC1)	0.7362 ± 0.0371	0.5473 ± 0.0339	0.7484 ± 0.0372	11.2761 ± 0.9189
RAPIQUE (Matlab PC2)	0.7509 ± 0.0453	0.5644 ± 0.0435	0.7735 ± 0.0423	10.7344 ± 0.7779
RAPIQUE (C++ PC1)	0.7182 ± 0.0409	0.5333 ± 0.0370	0.7525 ± 0.0381	11.1887 ± 0.7902
RAPIQUE (C++ PC2)	0.7116 ± 0.0405	0.5251 ± 0.0380	0.7426 ± 0.0347	11.3801 ± 0.6044

Table 11.3: Performance Metrics (± Standard Deviation) for RAPIQUE on LIVE VQC Dataset

Metric	KONVID_1K			
	SRCC	KRCC	PLCC	RMSE
RAPIQUE (Matlab PC1)	0.8066 ± 0.0189	0.6139 ± 0.0196	0.8129 ± 0.0158	0.3735 ± 0.0139
RAPIQUE (Matlab PC2)	0.8003 ± 0.0239	0.6070 ± 0.0249	0.8064 ± 0.0215	0.3790 ± 0.0171
RAPIQUE (C++ PC1)	0.7444 ± 0.0381	0.5485 ± 0.0350	0.7498 ± 0.0347	0.4236 ± 0.0228
RAPIQUE (C++ PC2)	0.7442 ± 0.0281	0.5476 ± 0.0273	0.7487 ± 0.0314	0.4246 ± 0.0216

Table 11.4: Performance Metrics (± Standard Deviation) for RAPIQUE on KONVID_1K Dataset

Metric	LIVE Qualcomm			
	SRCC	KRCC	PLCC	RMSE
RAPIQUE (Matlab PC1)	0.6294 ± 0.1581	0.4650 ± 0.1317	0.6810 ± 0.1359	8.4062 ± 1.5269
RAPIQUE (Matlab PC2)	0.6618 ± 0.1049	0.4865 ± 0.0953	0.6967 ± 0.0927	8.3430 ± 1.2867
RAPIQUE (C++ PC1)	0.6325 ± 0.0791	0.4525 ± 0.0658	0.6780 ± 0.0869	8.6007 ± 1.0931
RAPIQUE (C++ PC2)	0.6456 ± 0.0790	0.4674 ± 0.0695	0.6792 ± 0.0748	8.6051 ± 0.9593

Table 11.5: Performance Metrics (± Standard Deviation) for RAPIQUE on LIVE Qualcomm Dataset

Metric	YOUTUBE UGC			
	SRCC	KRCC	PLCC	RMSE
RAPIQUE (Matlab PC1)	0.7604 ± 0.0260	0.5673 ± 0.0235	0.7749 ± 0.0235	0.4133 ± 0.0199
RAPIQUE (Matlab PC2)	0.7601 ± 0.0293	0.5666 ± 0.0272	0.7735 ± 0.0261	0.4143 ± 0.0218
RAPIQUE (C++ PC1)	0.7393 ± 0.0824	0.5933 ± 0.0716	0.7017 ± 0.0992	0.4196 ± 55.802
RAPIQUE (C++ PC2)	0.7321 ± 0.0763	0.5863 ± 0.0640	0.6952 ± 0.0981	0.4212 ± 0.0452

Table 11.6: Performance Metrics (± Standard Deviation) for RAPIQUE on YOUTUBE UGC Dataset

To assess the performance of the RAPIQUE metric, comparative evaluations were conducted

between the newly proposed C++ version and the original Matlab version using five state-of-the-art video quality assessment (VQA) databases. This evaluation included measuring the total execution time on two different systems, designated PC1 and PC2 in the table 12.6.

In terms of resource utilization, the Matlab implementation of RAPIQUE showed no GPU usage on either machine, relying solely on CPU power. Specifically, CPU utilization was high, with PC1 recording approximately 91.6044% and PC2 about 83.7157%. In contrast, the C++ version presented a different pattern: on PC1, the CPU usage was approximately 76.3422% and on PC2 around 60.6751%, while GPU usage was significant at approximately 72.0872% for PC1 and 20.2561% for PC2.

It is important to note that the data on CPU and GPU utilization were meticulously observed, analyzed, and averaged from the graphical interface of the C++ version and the performance tab in the task manager.

Dataset	**RAPIQUE Matlab**		**RAPIQUE C++**	
	Total Time (sec)	**Avg Execution Time per Video (sec)**	**Total Time (sec)**	**Avg Execution Time per Video (sec)**
CVD_2014 PC1	23 215.0469	99.2096	12 093.2824	51.6807
CVD_2014 PC2	5568.9532	23.7989	4484.7481	19.4145
KONVID1K PC1	59 140.9219	49.2841	49 580.2558	41.4204
KONVID1K PC2	13 583.8593	11.3199	19 716.0464	16.6240
LIVE_Qualcomm PC1	22 800.3439	109.6170	16 794.9123	80.7448
LIVE_Qualcomm PC2	6097.7656	29.3162	3777.5160	18.1611
LIVE_VQC PC1	33 543.6720	56.5661	30 007.9092	51.3834
LIVE_VQC PC2	8835.0624	15.1027	8242.9178	14.2858
YOUTUBE_UGC PC1	123 944.3449	118.6070	125 349.6566	120.4127
YOUTUBE_UGC PC2	31 311.8280	29.9635	26 931.0923	26.2231

Table 11.7: Execution Time Data in RAPIQUE Matlab and C++ versions

A study of the main functions of the matlab and C++ versions was also carried out, which are shown in Tables 12.7, 12.8, 12.9, 12.10, and 12.11.

Main Module	**RAPIQUE Matlab**		**RAPIQUE C++**	
	YOUTUBE_UGC PC1 (sec)	**YOUTUBE_UGC PC2 (sec)**	**YOUTUBE_UGC PC1 (sec)**	**YOUTUBE_UGC PC2 (sec)**
SpatialNSS	3475.1891	760.7719	8626.4105	2243.8727
SpatialNSS-Var	110 011.8280	29 394.3449	101 473.4685	21 214.2000
TemporalNSS	3858.9875	750.7331	981.6954	287.9439
CNN	1206.3322	150.6210	1741.1209	335.3993

Table 11.8: Execution Times for YOUTUBE_UGC on PC1 and PC2 by Module in Matlab and C++ Versions

Main Module	RAPIQUE Matlab		RAPIQUE C++	
	LIVE_VQC PC1 (sec)	**LIVE_VQC PC2 (sec)**	**LIVE_VQC PC1 (sec)**	**LIVE_VQC PC2 (sec)**
SpatialNSS	972.1891	237.7430	1787.3614	706.7535
SpatialNSS-Var	30 917.8751	7133.1729	22 884.5381	6582.4726
TemporalNSS	865.6391	281.6438	435.7432	156.9434
CNN	247.7828	52.8594	1104.8113	230.8119

Table 11.9: Execution Times for LIVE_VQC on PC1 and PC2 by Module in Matlab and C++ Versions

Main Module	RAPIQUE Matlab		RAPIQUE C++	
	LIVE_Qualcomm PC1 (sec)	**LIVE_Qualcomm PC2 (sec)**	**LIVE_Qualcomm PC1 (sec)**	**LIVE_Qualcomm PC2 (sec)**
SpatialNSS	406.4803	96.4935	987.6942	356.0609
SpatialNSS-Var	21 245.3450	4987.9614	14 995.1011	3117.1088
TemporalNSS	378.6052	128.2373	324.1625	55.0540
CNN	94.2683	21.3637	393.0976	73.4079

Table 11.10: Execution Times for LIVE_Qualcomm on PC1 and PC2 by Module in Matlab and C++ Versions

Main Module	RAPIQUE Matlab		RAPIQUE C++	
	KONVID1K PC1 (sec)	**KONVID1K PC2 (sec)**	**KONVID_1K PC1 (sec)**	**KONVID_1K PC2 (sec)**
SpatialNSS	2204.7104	481.8948	2790.3680	1058.3259
SpatialNSS-Var	53 131.0277	12 733.7766	33 833.0293	13 150.0618
TemporalNSS	2017.6431	574.8655	1067.3633	493.3034
CNN	548.8647	94.8722	1658.7604	522.7202

Table 11.11: Execution Times for KONVID1K on PC1 and PC2 by Module in Matlab and C++ Versions

Main Module	RAPIQUE Matlab		RAPIQUE C++	
	CVD_2014 PC1 (sec)	**CVD_2014 PC2 (sec)**	**CVD_2014 PC1 (sec)**	**CVD_2014 PC2 (sec)**
SpatialNSS	422.3468	91.8499	444.4477	89.7826
SpatialNSS-Var	21 918.8460	4968.1502	10 778.4317	4074.2713
TemporalNSS	355.7746	111.5698	151.1743	66.1552
CNN	103.2466	20.7900	281.1305	77.6151

Table 11.12: Execution Times for CVD_2014 on PC1 and PC2 by Module in Matlab and C++ Versions

Table 11.13 presents the computational complexity analysis of both versions of RAPIQUE. For the SpatialNSS function, the complexity is $O(n{\cdot}m)$ in both Matlab and C++, where n represents the number of frames and m represents the number of spatial operations per frame. The SpatialNSS-Var function in Matlab has a complexity of $O(2(n \cdot m^2))$, because it performs the same operations

twice at two different scales, the original image scale and a reduced scale (half resolution). Here, m^2 refers to the adjustment of GGD and AGGD distributions, increasing the complexity to quadratic due to the need to calculate these second-order statistical features. In contrast, C++ executes these operations in parallel, resulting in $O(n \cdot m^2)$, as the operations are performed simultaneously without the need to call the function twice. The TemporalNSS function has a linear complexity of $O(n)$ in both versions due to its sequential frame-by-frame analysis without additional intensive operations. Finally, the CNN function has a complexity of $O(n \cdot k)$ for both implementations, where n is the number of frames and k is the number of CNN parameters, with $k = 2,048$ for ResNet-50.

To calculate these complexities in both versions of RAPIQUE, a function is added to record the number of function calls. A logical process to derive the Big-O formulas involves identifying the input size (n), determining the number of frames to be processed, analyzing the operations per frame (m), identifying the number of operations or transformations applied to each frame, considering the scales if a function operates on multiple scales (such as original and reduced scales), and accounting for parallel execution by adjusting the complexity to reflect simultaneous rather than sequential execution.

Main Module	**Big-O Notation (Matlab)**	**Big-O Notation (C++)**
SpatialNSS	O(n $\cdot m$)	O(n $\cdot m$)
SpatialNSS-Var	O(2(n $\cdot m^2$))	O(n $\cdot m^2$)
TemporalNSS	O(n)	O(n)
CNN	O(n $\cdot k$)	O(n $\cdot k$)

Table 11.13: Big-O Notation for RAPIQUE Matlab and C++

To advance this project and facilitate a comparison with the RAPIQUE implementation in C++, a detailed analysis was conducted on runtime and performance metrics (SRCC, PLCC, RMSE). This analysis encompassed four leading metrics in the field of no-reference video quality assessment, specifically TLVQM, VIDEVAL, VSFA, and MDTVSFA (see tables 12.13 and 12.14).The study utilized a sample of 200 videos from the databases employed in this project and executed on PC1.

Metric	**SRCC**		**KRCC**		**PLCC**		**RMSE**	
RAPIQUE (matlab)	0.6646	± 0.0680	0.4774	± 0.0589	0.6952	± 0.0859	19.3231	± 1.9709
RAPIQUE (C++)	0.6337	± 0.1169	0.4673	± 0.0895	0.6637	± 0.1066	21.5149	± 2.9358
TLVQM	0.5090	± 0.1094	0.5589	± 0.0582	0.5476	± 0.1029	25.0157	± 3.1136
VIDEVAL	0.6336	± 0.0865	0.4485	± 0.0695	0.6528	± 0.0962	22.7983	± 3.5459
VSFA	0.6202	± 0.0871	0.4510	± 0.0449	0.6622	± 0.0611	16.4711	± 1.4316
MDTVSFA	0.7674	± 0.1258	0.5695	± 0.0911	0.7975	± 0.1479	25.9089	± 2.3737

Table 11.14: Performance Metrics (± Standard Deviation) for Video Quality Assessment Models on PC1 and all combined datasets (200 sample videos)

Metric	Total Time (sec)	Average Execution Time per Video (sec)
RAPIQUE (matlab)	16 971.000 000	84.855 000
RAPIQUE (C++)	11 562.100 000	58.394 500
TLVQM	46 852.194 300	234.260 900
VIDEVAL	91 468.158 400	457.340 700
VSFA	11 220.000 000	56.100 000
MDTVSFA	7680.000 000	58.400 000

Table 11.15: Execution Times for the Video Quality Assessment Models RAPIQUE (matlab), RAPIQUE (C++), TLVQM, VIDEVAL, VSFA and MDTVSFA on PC1 and all combined datasets (200 sample videos)

Table 12.5 shows the median CPU and GPU utilization of the RAPIQUE algorithm in both versions (Matlab and C++) in the KoNViD-1k, LIVE-Qualcomm, LIVE Video Quality Challenge (VQC), YouTube-UGC and CVD2014 datasets on PC1 and PC2, together with their standard deviations.

Dataset	RAPIQUE Matlab		RAPIQUE C++	
	GPU(%)±GPU(σ)	CPU(%)±CPU(σ)	GPU(%)±GPU(σ)	CPU(%)±CPU(σ)
CVD_2014 PC1	0.0000 ± 0.0000	87.3456 ± 4.2415	81.5678 ± 2.5678	68.7890 ± 3.1782
CVD_2014 PC2	0.0000 ± 0.0000	65.4567 ± 3.0112	30.6789 ± 2.1234	49.4567 ± 2.9451
KONVID1K PC1	0.0000 ± 0.0000	86.5678 ± 3.1256	71.3456 ± 2.6789	69.5678 ± 2.7894
KONVID1K PC2	0.0000 ± 0.0000	74.2345 ± 2.9345	29.5678 ± 1.9789	57.4567 ± 2.5345
LIVE_Qualcomm PC1	0.0000 ± 0.0000	88.1234 ± 4.4789	83.4567 ± 2.3456	61.1234 ± 3.2345
LIVE_Qualcomm PC2	0.0000 ± 0.0000	66.3456 ± 3.1567	30.6789 ± 2.1890	49.2345 ± 2.8678
LIVE_VQC PC1	0.0000 ± 0.0000	87.5678 ± 3.2678	73.2345 ± 2.3456	60.4567 ± 2.5678
LIVE_VQC PC2	0.0000 ± 0.0000	76.2345 ± 3.0456	30.5678 ± 2.0123	53.3456 ± 2.7890
YOUTUBE_UGC PC1	0.0000 ± 0.0000	91.5456 ± 4.5789	75.4567 ± 2.4567	72.1156 ± 3.1456
YOUTUBE_UGC PC2	0.0000 ± 0.0000	78.4567 ± 3.3456	33.3456 ± 2.2567	61.2345 ± 2.9789

Table 11.16: Median and Standard Deviation of GPU and CPU Utilization for RAPIQUE Matlab and C++ on PC1 and PC2

11.0.3 Analysis of results

- Performance Metrics

  - Statistical Accuracy: Across various datasets, the Matlab implementation consistently exhibited slightly superior correlation coefficients and lower RMSE values compared to its C++ counterpart, indicating a nuanced edge in prediction accuracy. Specifically, the Matlab version on the LIVE Qualcomm dataset achieved an SRCC of 0.6294 ± 0.1581, marginally higher than the 0.6325 ± 0.0791 achieved by the C++ implementation on the same hardware. This pattern, where Matlab exceeds C++ by approximately 1-3% in

correlation coefficients, suggests Matlab's superior handling of datasets' variability and inherent noise. This advantage likely stems from Matlab's mature numerical libraries and optimized algorithms for matrix operations.

- Influence of Dataset Characteristics: The characteristics of the datasets, such as the resolution range from 320x240 in LIVE VQC to 3840x2160 in YouTube UGC and variable frame rates, significantly influence the performance metrics. Higher resolutions and variable frame rates present greater challenges in feature extraction and quality prediction, likely explaining the larger deviations in performance metrics across implementations and hardware setups.
- Both versions of RAPIQUE perform comparably or better than TLVQM and VIDEVAL, but are slightly outperformed by VSFA and MDTVSFA in terms of SRCC and Kendall Rank Correlation Coefficient (KRCC).

- Execution Times

  - Computational Efficiency: The C++ implementation significantly outperformed Matlab in terms of execution speed across all datasets and hardware configurations. For instance, on the LIVE Qualcomm dataset, the C++ version processed videos on PC2 approximately 60% faster, averaging 18.1611 seconds per video compared to 29.2162 seconds for Matlab. This enhanced efficiency of C++ is attributed to its direct control over memory management and system resources, along with effective utilization of parallel processing capabilities, particularly the GPU.
  - Hardware Utilization: The analysis of CPU and GPU utilization revealed a distinct pattern; the C++ version effectively leveraged GPU resources, significantly enhancing performance. On PC1, GPU usage for the C++ implementation was about 72.0872%, while Matlab showed no GPU utilization and had CPU usage up to 91.6044%. This stark contrast underscores C++'s advantage in environments where parallel processing capabilities can be effectively harnessed.
  - RAPIQUE C++ version demonstrates a more efficient average processing time per video at 58.3945 seconds compared to 84.8550 seconds for Matlab. This efficiency is critical in large-scale processing tasks and is still superior when compared to other metrics like TLVQM and VIDEVAL, which take considerably longer.
  - On PC1, which is outfitted with a GTX 1050 GPU and an Intel Core i7-8750H CPU, the C++ implementation surpasses (19.2%) Matlab for the KONVID1K dataset, which primarily includes lower resolution videos (960x540). This enhanced performance is due to C++'s robust utilization of GPU for parallel processing tasks—a capability that Matlab does not leverage, as it operates solely on the CPU. On PC2, equipped with a more potent RTX 4070 GPU and an Intel Core i9-13900KF CPU, Matlab outperforms

(45.1%) C++ in processing the same dataset. This is attributed to Matlab's optimized use of matrix operations through its Statistics and Machine Learning Toolbox, supported by the high-performance CPU and ample RAM, effectively offsetting the lack of GPU usage (MathWorks, 2022a) .
For the YOUTUBE_UGC dataset, which spans a wider range of video resolutions (from 640x480 to 3840x2160), the processing demands escalate. On PC1, Matlab edges out C++ slightly (1.1%), leveraging its Image Processing and Optimization Toolboxes to efficiently handle the high-resolution video processing on the CPU, where C++ does not fully capitalize on the limited GPU resources (MathWorks, 2022b). Conversely, on PC2, C++ showcases superior performance (16.2%), benefiting greatly from the RTX 4070 GPU's advanced capabilities to handle more complex and intensive parallel processing tasks. This switch in performance highlights C++'s adeptness in utilizing modern GPU architectures, marking it as particularly suitable for high-performance computing environments.

- The C++ version of RAPIQUE showcases significant strengths in handling both temporal and spatial variability features across various datasets. In the TemporalNSS function, C++ consistently outperforms Matlab, as exemplified in the LIVE_Qualcomm dataset where it processes tasks approximately 16% faster on PC1. Additionally, C++ exhibits superior performance in the SpatialNSS-Var function within the same dataset, completing computations in 14995.10 seconds compared to Matlab's 21245.34 seconds, achieving a notable 41% reduction in processing time. This enhanced efficiency in both temporal and spatial analyses underlines C++'s robust capability in managing diverse computational demands effectively.

- Comparative Insight

  Although Matlab shows slightly higher correlation metrics, the difference is minimal, typically around 1-3%. This marginal advantage does not significantly impact the overall effectiveness of the video quality predictions when compared to the substantial gains in computational efficiency and execution speed offered by the C++ implementation. Given these efficiencies and the scalability of C++, the slight edge Matlab has in correlation becomes less significant, particularly in practical applications where throughput and speed are critical. Therefore, for operational environments requiring high performance and scalability, C++ is the recommended implementation despite the slightly lower correlation figures, as the differences in correlation are minimal and the operational efficiencies of C++ far outweigh them.

  The computational complexity of the algorithms and techniques implemented in the MATLAB version is not optimal. MATLAB's approach to decoding videos and storing them in a temporary directory before processing introduces inefficiencies related to disk I/O. The RAPIQUE feature calculation function processes each video frame sequentially, which is particularly inefficient for long videos. Sequential execution is also evident in convolutions and matrix operations, and filtering tasks are conducted in a single thread, thereby underuti-

lizing MATLAB's potential for parallelization. Furthermore, memory management is not adequately optimized, as temporary storage for video frames and features is not deallocated promptly. The SpatialNSS algorithm maintains a complexity of $O(n \cdot m)$ in both versions, while SpatialNSS-Var is less efficient in MATLAB with $O(2(n \cdot m^2))$ compared to $O(n \cdot m^2)$ in C++ due to parallel processing capabilities. TemporalNSS and CNN exhibit $O(n)$ and $O(n \cdot k)$ complexities, respectively, across both implementations. To enhance MATLAB's execution times, leveraging GPU for parallel operations using `gpuArray` and GPU-specific functions, enabling `parfor` and `parfeval` for concurrent frame processing, and optimizing memory and video decoding efficiency are recommended.

The runtime issues in MATLAB's implementation stem from the intrinsic complexity of the algorithms, the specificities of the implementation, and limitations inherent to the programming language. Key issues include the sequential processing in `calc_RAPIQUE_features` and the lack of optimized parallelization in functions like `RAPIQUE_spatial_features`. Inefficiencies in MATLAB's memory management and limited GPU optimization further exacerbate these issues.

Specific data structure decisions in C++ have positively influenced the project's objectives. C++ employs `std::vector` and `std::array` for managing dynamic and fixed-size data collections, respectively, enhancing data access and cache efficiency. Preallocating buffers to mitigate dynamic memory allocation overhead during execution, using OpenMP for loop parallelization, and `std::thread` for concurrent task execution have significantly reduced execution time. Leveraging SIMD instructions for vector operations and utilizing advanced compiler optimizations have further improved computational efficiency. While MATLAB uses BLAS and LAPACK for precise matrix calculations, C++ achieves high precision with Eigen and Armadillo, though MATLAB's integrated optimizations remain superior in avoiding additional overhead from external library dependencies.

C++ was selected over C and other languages for implementing RAPIQUE due to its support for object-oriented programming, which facilitates modularity, code reuse, and software maintenance. The Standard Library (STL) in C++ includes essential data structures and provides both manual memory management and automatic tools like smart pointers for efficient resource control. Compatibility with CUDA and OpenCL for GPU acceleration, along with an optimal balance between low-level and high-level programming, allows for efficient code development while maintaining the flexibility needed for complex applications, significantly impacting the project's timeline.

Chapter 12

Conclusion

This comprehensive comparative analysis of the RAPIQUE metric implemented in Matlab and C++ across a spectrum of video quality assessment (VQA) databases has provided significant insights into the trade-offs between statistical accuracy and computational efficiency. The findings from this study underscore the nuanced performance differences between these two implementations and highlight crucial considerations for practical applications in video quality assessment.

While the Matlab implementation of RAPIQUE demonstrates a slight superiority in achieving higher correlation coefficients and lower RMSE values, the difference in accuracy is relatively marginal, often within the 1-3% range. This suggests that while Matlab may handle the subtleties of video quality prediction with slightly greater finesse, possibly due to its advanced numerical libraries, the gap in accuracy is not substantial enough to overshadow robust processing power and high throughput.

On the other hand, the C++ implementation exhibits formidable strengths in terms of computational efficiency and scalability. Notably, the ability of the C++ version to utilize GPU resources effectively translates into significant performance enhancements. This capability enables the C++ implementation to process videos much faster than the Matlab version, a critical advantage in scenarios requiring the handling of large datasets or real-time processing.

Furthermore, the hardware utilization patterns observed during the study indicate that the C++ implementation not only optimizes GPU usage but also maintains lower overall CPU usage compared to Matlab. This efficient resource management is particularly beneficial in multi-threaded and parallel computing environments commonly used in contemporary video processing tasks.

Given these observations, it is recommended that for operational settings requiring robust processing capabilities and high throughput, the C++ implementation of RAPIQUE is preferable. The efficiency gains in execution time and the better scalability of the C++ version make it a more suitable choice in most practical applications, despite the slightly lower accuracy metrics.

However, this does not diminish the potential for further enhancements. The slight gap in predictive accuracy between the Matlab and C++ versions presents an opportunity for future optimization. Enhancing the C++ implementation to bridge this gap could involve refining algorithms to better handle the intrinsic variability and complexity of video data, or integrating more sophisticated machine learning models that are capable of deeper and more nuanced analysis.

12.0.1 Future work

The comparative study of RAPIQUE implementations in Matlab and C++ has established a strong foundation for numerous potential research and development directions aimed at enhancing both the performance and the broader applicability of video quality assessment tools. Here, we outline several key areas where future initiatives could significantly elevate the current capabilities of this technology:

- **Algorithm Optimization:**
  - **Accuracy Enhancement:** Although the C++ implementation is notably efficient, its marginally lower accuracy relative to Matlab indicates opportunities for enhancement. Future efforts should concentrate on refining the feature extraction algorithms and optimization techniques. This focus aims to boost the predictive accuracy of the tool without sacrificing its computational efficiency.
  - **Parallel Processing Capabilities:** There is substantial scope for advancing the C++ implementation by optimizing it for various architectures and GPUs. Such improvements could better harness hardware resources, potentially increasing both the speed and accuracy of the tool.
  - The choice between RAPIQUE's implementations and other models like VSFA or MDTVSFA should consider both the accuracy requirements and the computational resources available. Enhancements in RAPIQUE's algorithms could potentially improve both its accuracy and efficiency, positioning it more favorably against its competitors.
- **Usability and Scalability:**
  - **Software Development Kits (SDKs):** Developing SDKs or APIs that encapsulate RAPIQUE's functionality could significantly streamline its integration into commercial applications. This advancement would expand the tool's usability across diverse platforms and services, making it more accessible to a broader range of users.
  - **Cloud-Based Solutions:** Exploring the deployment of RAPIQUE metrics on cloud platforms would enhance the scalability and accessibility of these tools. By leveraging cloud computing resources, users could perform video quality assessments more efficiently and on a larger scale.

Chapter 13

Glossary of Terms

Pseudo-real-time It describes systems or applications that achieve real-time performance but may suffer from some latencies or delays. Unlike strict real-time systems, where failure to meet a time limit can have serious or unacceptable consequences (as in flight control systems or medical devices), a pseudo-real-time system is generally more tolerant of occasional delays (Laplante et al., 2004).

NIQE (Naturalness Image Quality Evaluator) It is an unreferenced image quality evaluation method that relies on natural scene statistics (NSS) to evaluate the quality of an image. NIQE is trained generally rather than on a specific database, making it very versatile (Mittal et al., 2012b).

BRISQUE Blind/Referenceless Image Spatial Quality Evaluator) Like NIQE, BRISQUE also uses NSS but specializes in evaluating spatial distortions in images. It is often used as a starting point for developing NR-VQA video quality metrics (Mittal et al., 2012a).

DIIVINE DIIVINE (Distortion Identification-based Image Verity and INtegrity Evaluator) An image quality evaluation method uses spatial and spectral information to make its evaluations (Soundararajan and Bovik, 2011).

FRIQUEE (Feature maps based Referenceless Image Quality Evaluation Engine) A full reference image quality evaluation metric that employs a range of features, including the NSS (Ghadiyaram and Bovik, 2014).

FR-VQA (Full Reference Video Quality Assessment) Video quality assessment metrics require both the original and degraded videos to assess quality. They are accurate but computationally more intensive (Wang et al., 2004b).

NR-VQA (No-Reference Video Quality Assessment) Metrics that do not require access to the original video to assess quality, making them more versatile but challenging to implement (Saad et al., 2012).

BVQA (Blind Video Quality Assessment) A general term for algorithms designed to assess video quality without the need for a reference (Ghadiyaram and Bovik, 2015b).

VGG (Visual Geometry Group) It is a convolutional neural network (CNN) architecture developed by the Visual Geometry Group at the University of Oxford (Simonyan and Zisserman, 2014).

ResNet-50 A specific type of residual neural network commonly used for image classification has 50 layers. It is known for its ability to solve the gradient fading problem in deep training (He et al., 2016).

KonCept512 is an unreferenced image quality metric based on features learned by a pre-trained model (Kim et al., 2017).

HIGRADE This unreferenced image quality metric uses deep learning techniques (Narvekar and Karam, 2011).

Transfer learning It is a technique in which a model developed for one specific task is adapted for a second related task (Pan and Yang, 2009).

LIVE (Laboratory for Image and Video Engineering) A database containing various types and levels of image and video quality distortions, used extensively in VQA research (Sheikh et al., 2006).

CSIQ (Composite Subjective Image Quality) A database commonly used to evaluate image quality contains various distortion types and levels (Larson and Chandler, 2010).

IVPL (Image and Video Processing Lab) A database created by the Image and Video Processing Lab at Northwestern University (Li, 2004).

EPFL-PoliMI Ecole Polytechnique Fédérale de Lausanne - Politecnico di Milano) A database for video quality evaluation that considers different types of distortions and viewing conditions (De Simone et al., 2009).

Convolutional layer It is one of the main layers in a convolutional neural network, responsible for detecting features in the input data (LeCun et al., 1998).

Pooling layer A layer in a CNN that reduces the spatial dimensions (width x height) of the input, which helps in making the feature detector more invariant to position and orientation (Scherer et al., 2010).

Fully connected layer A layer in a neural network where each neuron is connected to each neuron in the previous layer and the next layer (Goodfellow et al., 2016).

Softmax It is an activation function that converts numbers, logits, into a probability distribution (Bridle, 1990).

Mean pooling It is a dimension reduction technique that uses the average value of a group of features (Boureau et al., 2010).

Max pooling A form of dimension reduction that uses the maximum value of a group of characteristics (Ranzato et al., 2007).

Cross-validation An approach to model evaluation where the data set is divided into subsets, and each subset is used for both training and validation (Kohavi et al., 1995).

Model C4 It is a graphical notation technique used to model the architecture of software systems. It is based on the structural decomposition of a system into containers and components (Richards and Ford, 2019; Enríquez and Salazar, 2018).

Global Average Pooling (GAP) A feature pooling technique used in convolutional neural networks where the mean output of each feature map in the previous layer is taken, resulting in a single number. It helps in reducing the dimensionality and preventing overfitting (Lin et al., 2013).

Spatial Pyramid Pooling (SPP) A layer used in convolutional neural networks that pools features in a fixed-size output regardless of image size, allowing the network to retain spatial information and handle inputs of varying sizes (He et al., 2015).

CUDA A parallel computing platform and application programming interface model created by NVIDIA. It allows developers to use GPUs for general purpose processing, significantly speeding up computing tasks by harnessing the power of GPUs (Corporation, 2019).

Perceptually Relevant Color Spaces (LAB) The LAB color space includes the L* (luminance), a* (red-green axis), and b* (blue-yellow axis) dimensions. It is designed to approximate human vision and is useful in applications where color manipulation is needed, more closely aligning with human perception than RGB (Hunter, 1948).

RGB A color model in which Red, Green, and Blue light are added together in various ways to reproduce a broad array of colors. The name of the model comes from the initials of the three additive primary colors, red, green, and blue (Jain, 1989).

NSS-34 Stands for Natural Scene Statistics-34, a set of 34 statistical features extracted from video or image data to assess quality. These features are derived from models like the Generalized Gaussian Distribution (GGD) and Asymmetric Generalized Gaussian Distribution (AGGD). NSS-34 captures essential statistical properties from the mean subtracted contrast normalized (MSCN) coefficients of the frames. These coefficients are used to measure deviations from normal scene statistics, providing a robust indication of visual quality that reflects both the inherent characteristics of the content and any distortions or degradations (Mittal et al., 2012a).

Bibliography

(2024). Ffmpeg. Online. Accessed: 25 January 2024.

(2024). Googletest: Google testing and mocking framework. Online. Accessed: 29 January 2024.

AHO, A. V., SETHI, R., and ULLMAN, J. D. (1988). Compilers, principles, techniques and tools. murray hill.

Anderson, J. and Rainie, L. (2018). The future of well-being in a tech-saturated world.

at Kent State University, L. (2023). Pearson correlation - spss tutorials. https://libguides.library.kent.edu/spss/pearsoncorr. Accessed: 06 September 2023.

Babi, G. (2005). Cse675.02: Principles of computer architecture – performance.

Balas, B., Nakano, L., and Rosenholtz, R. (2009). A summary-statistic representation in peripheral vision explains visual crowding. *Journal of vision*, 9(12):13–13.

Beck, K. (2003). *Test Driven Development: By Example*. Addison-Wesley Professional.

Bex, P. J. and Makous, W. (2002). Spatial frequency, phase, and the contrast of natural images. *JOSA A*, 19(6):1096–1106.

Bianco, S., Celona, L., Napoletano, P., and Schettini, R. (2018). On the use of deep learning for blind image quality assessment. *Signal, Image and Video Processing*, 12:355–362.

Boureau, Y.-L., Bach, F., LeCun, Y., and Ponce, J. (2010). Learning mid-level features for recognition. pages 2559–2566.

Bovik, A. C. (2013). Automatic prediction of perceptual image and video quality. *Proceedings of the IEEE*, 101(9):2008–2024.

Bridle, J. S. (1990). Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing: Algorithms, architectures and applications*, pages 227–236. Springer.

BT, I. R. (2012). 500-13,". *Methodology for the subjective assessment of the quality of television pictures," International Telecommunication Union*, 6.

Chandler, D. M. and Hemami, S. S. (2007). Vsnr: A wavelet-based visual signal-to-noise ratio for natural images. *IEEE transactions on image processing*, 16(9):2284–2298.

Corporation, N. (2011). *NVIDIA CUDA C Programming Guide*. Version 4.1.

Corporation, N. (2019). Cuda c programming guide. Available online: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html (accessed on 10.04.2020).

Corporation, N. (2021). *NVIDIA Nsight: Development Environment for Heterogeneous Platforms*. Technical documentation and user guide available online.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20:273–297.

CUDA, C. (2012). Programming guide version 4.0. *NVIDIA Corp.*

Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. *International Journal of Computer Vision*, 20(6):886–893.

Daly, S. J. (1992). Visible differences predictor: an algorithm for the assessment of image fidelity. In *Human Vision, Visual Processing, and Digital Display III*, volume 1666, pages 2–15. SPIE.

De Simone, F., Naccari, M., Tagliasacchi, M., Dufaux, F., Tubaro, S., and Ebrahimi, T. (2009). Subjective assessment of h. 264/avc video sequences transmitted over a noisy channel. pages 204–209.

Doe, J. and Roe, J. (2021). Detailed analysis and implementation of aggd models in image processing. *Journal of Advanced Imaging*, 58(2):205–220.

Enríquez, R. and Salazar, A. (2018). *Software Architecture with Spring 5.0: Design and Architect Highly Scalable, Robust, and High-Performance Java Applications*. Packt Publishing. OCLC: 1053798657.

Fairchild, M. D. and Johnson, G. M. (2004). icam framework for image appearance, differences, and quality. *Journal of Electronic Imaging*, 13(1):126–138.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley.

Geisler, W. S. (2008). Visual perception and the statistical properties of natural scenes. *Annu. Rev. Psychol.*, 59:167–192.

Ghadiyaram, D. and Bovik, A. C. (2014). Blind image quality assessment on real distorted images using deep belief nets. In *2014 IEEE global conference on signal and information processing (GlobalSIP)*, pages 946–950. IEEE.

Ghadiyaram, D. and Bovik, A. C. (2015a). Massive online crowdsourced study of subjective and objective picture quality. *IEEE Transactions on Image Processing*, 25(1):372–387.

Ghadiyaram, D. and Bovik, A. C. (2015b). Scene statistics of authentically distorted images in perceptually relevant color spaces for blind image quality assessment. In *2015 IEEE International conference on image processing (ICIP)*, pages 3851–3855. IEEE.

Ghadiyaram, D., Pan, J., Bovik, A., Moorthy, A., Panda, P., and Yang, K. (2017a). Live-qualcomm mobile in-capture video quality database. *Online: http://live. ece. utexas. edu/research/incaptureDatabase/index. html*.

Ghadiyaram, D., Pan, J., Bovik, A. C., Moorthy, A., Panda, P., and Yang, K.-C. (2017b). Subjective and objective quality assessment of mobile videos with in-capture distortions. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1393–1397. IEEE.

Ghadiyaram, D., Pan, J., Bovik, A. C., Moorthy, A. K., Panda, P., and Yang, K.-C. (2017c). In-capture mobile video distortions: A study of subjective behavior and objective algorithms. *IEEE Transactions on Circuits and Systems for Video Technology*, 28(9):2061–2077.

Gilpin, G. (2011). Nonparametric statistics for non-statisticians: A step-by-step approach. *Biometrics*.

Gonzalez, R. C. and Woods, R. E. (2002). *Digital Image Processing*. Prentice Hall.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. (2016). Deep learning, volume 1.

Götz-Hahn, F., Hosu, V., Lin, H., and Saupe, D. (2019). No-reference video quality assessment using multi-level spatially pooled features. *arXiv preprint arXiv:1912.07966*.

Gregoire, M. (2013). *Professional C++*. John Wiley & Sons.

Hargittai, E. and Hinnant, A. (2008). Digital inequality: Differences in young adults' use of the internet. *Communication research*, 35(5):602–621.

Harold, E. R. and Means, W. S. (2004). *XML in a Nutshell*. O'Reilly Media, Inc., 3 edition.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Spatial pyramid pooling in deep convolutional networks for visual recognition. In *European conference on computer vision*, pages 346–361. Springer.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Hennessy, J. L. and Patterson, D. A. (2011). *Computer architecture: a quantitative approach*. Elsevier.

Hosmer Jr, D. W., Lemeshow, S., and Sturdivant, R. X. (2013). *Applied Logistic Regression*. John Wiley & Sons, 3 edition.

Hosu, V., Hahn, F., Jenadeleh, M., Lin, H., Men, H., Szirányi, T., Li, S., and Saupe, D. (2020). The konstanz natural video database.

Hunter, R. S. (1948). *Photoelectric Color Difference Meter*. The National Bureau of Standards.

ITU (2017). Vocabulary for performance, quality of service and quality of experience.

Jain, A. K. (1989). *Fundamentals of Digital Image Processing*. Prentice-Hall, Inc.

Kaehler, A. and Bradski, G. (2016). *Learning OpenCV 3: computer vision in C++ with the OpenCV library*. " O'Reilly Media, Inc.".

Kim, J., Zeng, H., Ghadiyaram, D., Lee, S., Zhang, L., and Bovik, A. C. (2017). Deep convolutional neural models for picture-quality prediction: Challenges and solutions to data-driven image quality assessment. *IEEE Signal processing magazine*, 34(6):130–141.

Kohavi, R. et al. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. 14(2):1137–1145.

Kondratyuk, D., Yuan, L., Li, Y., Zhang, L., Tan, M., Brown, M., and Gong, B. (2021). Movinets: Mobile video networks for efficient video recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16020–16030.

Korhonen, J. (2019). Two-level approach for no-reference consumer video quality assessment. *IEEE Transactions on Image Processing*, 28(12):5923–5938.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90.

Lam, E. and Goodman, J. W. (2000). Some reference to the ggd usage in texture analysis. *Journal of Statistical Analysis*, 45(1):123–134.

Laplante, P. A. et al. (2004). *Real-time systems design and analysis*. Wiley New York.

Larson, E. C. and Chandler, D. M. (2010). Most apparent distortion: full-reference image quality assessment and the role of strategy. *Journal of electronic imaging*, 19(1):011006–011006.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436–444.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Lee, D., Dinov, I., Dong, B., Gutman, B., Yanovsky, I., and Toga, A. W. (2012). Cuda optimization strategies for compute-and memory-bound neuroimaging algorithms. *Computer methods and programs in biomedicine*, 106(3):175–187.

Lee, V. W., Kim, C., Chhugani, J., Deisher, M., Kim, D., Nguyen, A. D., Satish, N., Smelyanskiy, M., Chennupaty, S., Hammarlund, P., et al. (2010). Debunking the 100x gpu vs. cpu myth: an evaluation of throughput computing on cpu and gpu. In *Proceedings of the 37th annual international symposium on Computer architecture*, pages 451–460.

Lehman, A. (2005). Jmp for basic univariate and multivariate statistics: a step-by-step guide. *(No Title)*.

Lehmann, E. and Casella, G. (1998). Average risk optimality. *Theory of point estimation*, pages 225–307.

Li, D., Jiang, T., and Jiang, M. (2019). Quality assessment of in-the-wild videos. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 2351–2359.

Li, D., Jiang, T., and Jiang, M. (2021). Unified quality assessment of in-the-wild videos with mixed datasets training. *International Journal of Computer Vision*, 129:1238–1257.

Li, Z. (2004). Image/video processing lab (ivpl), dept of eecs, northwestern university.

Lin, M., Chen, Q., and Yan, S. (2013). Network in network. *arXiv preprint arXiv:1312.4400*.

Lin, W. and Kuo, C.-C. J. (2011). Perceptual visual quality metrics: A survey. *Journal of visual communication and image representation*, 22(4):297–312.

Liu, Y., Wu, J., Li, L., Dong, W., Zhang, J., and Shi, G. (2021). Spatiotemporal representation learning for blind video quality assessment. *IEEE Transactions on Circuits and Systems for Video Technology*, 32(6):3500–3513.

Lundh, F. (2005). *An Introduction to Tkinter*. O'Reilly Media, Inc.

Lutz, M. (2013). *Programming Python*. O'Reilly Media, Inc.

Lv, X., Zhang, S., Wang, C., Zhang, W., Yao, H., and Huang, Q. (2023). Unsupervised low-light video enhancement with spatial-temporal co-attention transformer. *IEEE Transactions on Image Processing*.

Ma, K., Liu, W., Zhang, K., Duanmu, Z., Wang, Z., and Zuo, W. (2017). End-to-end blind image quality assessment using deep neural networks. *IEEE Transactions on Image Processing*, 27(3):1202–1213.

MathWorks (2022a). Efficient use of matrix operations and data handling in matlab. *MathWorks Documentation*. Accessed: May 18, 2024.

MathWorks (2022b). Matlab image processing and optimization toolboxes. *MathWorks Documentation*. Accessed: May 18, 2024.

Mei, Z., Wang, Y.-C., and Kuo, C.-C. J. (2023). Blind video quality assessment at the edge. *arXiv preprint arXiv:2306.10386*.

Mittal, A., Moorthy, A. K., and Bovik, A. C. (2012a). No-reference image quality assessment in the spatial domain. *IEEE Transactions on image processing*, 21(12):4695–4708.

Mittal, A., Soundararajan, R., and Bovik, A. C. (2012b). Making a "completely blind" image quality analyzer. *IEEE Signal processing letters*, 20(3):209–212.

Möller, S. and Raake, A. (2014). *Quality of experience: advanced concepts, applications and methods*. Springer.

Narvekar, N. D. and Karam, L. J. (2011). A no-reference image blur metric based on the cumulative probability of blur detection (cpbd). *IEEE Transactions on Image Processing*, 20(9):2678–2683.

Nuutinen, M., Virtanen, T., Vaahteranoksa, M., Vuori, T., Oittinen, P., and Häkkinen, J. (2016). Cvd2014—a database for evaluating no-reference video quality assessment algorithms. *IEEE Transactions on Image Processing*, 25(7):3073–3086.

Pan, S. J. and Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32.

Patrone, A. R., Valuch, C., Ansorge, U., and Scherzer, O. (2016). Dynamical optical flow of saliency maps for predicting visual attention. *arXiv preprint arXiv:1606.07324*.

Pedersen, M. and Hardeberg, J. Y. (2009). Survey of full-reference image quality metrics.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830.

Pinson, M. H. and Wolf, S. (2004). A new standardized method for objectively measuring video quality. *IEEE Transactions on broadcasting*, 50(3):312–322.

Ranzato, M., Huang, F. J., Boureau, Y.-L., and LeCun, Y. (2007). Unsupervised learning of invariant feature hierarchies with applications to object recognition. pages 1–8.

Ray, E. and Ray, E. T. (2004). *XML: Visual QuickStart Guide*. Peachpit Press.

Redi, J. A., Gastaldo, P., Heynderickx, I., and Zunino, R. (2010). Color distribution information for the reduced-reference assessment of perceived image quality. *IEEE Transactions on Circuits and Systems for Video Technology*, 20(12):1757–1769.

Richards, M. and Ford, N. (2019). *Fundamentals of software architecture: an engineering approach*. O'REILLY MEDIA. OCLC: 1138515057.

Rosebrock, A. (2018). *Practical Python and OpenCV*. PyImageSearch, 4th edition.

Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011). Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. IEEE.

Ruderman, D. L. and Bialek, W. (1994). Statistics of natural images: Scaling in the woods. *Physical review letters*, 73(6):814.

Saad, M. A., Bovik, A. C., and Charrier, C. (2012). Blind image quality assessment: A natural scene statistics approach in the dct domain. *IEEE transactions on Image Processing*, 21(8):3339–3352.

Saad, M. A., Bovik, A. C., and Charrier, C. (2014). Blind prediction of natural video quality. *IEEE Transactions on image Processing*, 23(3):1352–1365.

Scherer, D., Müller, A., and Behnke, S. (2010). Evaluation of pooling operations in convolutional architectures for object recognition. pages 92–101.

Seshadrinathan, K. and Bovik, A. C. (2009). Motion tuned spatio-temporal quality assessment of natural videos. *IEEE transactions on image processing*, 19(2):335–350.

Seshadrinathan, K., Soundararajan, R., Bovik, A. C., and Cormack, L. K. (2010a). Study of subjective and objective quality assessment of video. *IEEE transactions on Image Processing*, 19(6):1427–1441.

Seshadrinathan, K., Soundararajan, R., Bovik, A. C., and Cormack, L. K. (2010b). A subjective study to evaluate video quality assessment algorithms. In *Human Vision and Electronic Imaging XV*, volume 7527, pages 128–137. SPIE.

Seufert, M., Egger, S., Slanina, M., Zinner, T., Hoßfeld, T., and Tran-Gia, P. (2014). A survey on quality of experience of http adaptive streaming. *IEEE Communications Surveys & Tutorials*, 17(1):469–492.

Shehabi, A., Smith, S., Sartor, D., Brown, R., Herrlin, M., Koomey, J., Masanet, E., Horner, N., Azevedo, I., and Lintner, W. (2016). United states data center energy usage report.

Sheikh, H. R., Sabir, M. F., and Bovik, A. C. (2006). A statistical evaluation of recent full reference image quality assessment algorithms. *IEEE Transactions on image processing*, 15(11):3440–3451.

Sheskin, D. J. (2007). *Handbook of parametric and nonparametric statistical procedures*. CRC Press, 4 edition.

Simoncelli, E. P. and Olshausen, B. A. (2001). Natural image statistics and neural representation. *Annual review of neuroscience*, 24(1):1193–1216.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Sinno, Z. and Bovik, A. C. (2018). Large-scale study of perceptual video quality. *IEEE Transactions on Image Processing*, 28(2):612–627.

Soundararajan, R. and Bovik, A. C. (2011). Rred indices: Reduced reference entropic differencing for image quality assessment. *IEEE Transactions on Image Processing*, 21(2):517–526.

Stroustrup, B. (2013). *The C++ programming language*. Pearson Education.

Sun, W., Min, X., Lu, W., and Zhai, G. (2022). A deep learning based no-reference quality assessment model for ugc videos. In *Proceedings of the 30th ACM International Conference on Multimedia*, pages 856–865.

Sun, W., Wang, T., Min, X., Yi, F., and Zhai, G. (2021). Deep learning based full-reference and no-reference quality assessment models for compressed ugc videos. In *2021 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*, pages 1–6. IEEE.

Tajbakhsh, N., Shin, J. Y., Gurudu, S. R., Hurst, R. T., Kendall, C. B., Gotway, M. B., and Liang, J. (2016). Convolutional neural networks for medical image analysis: Full training or fine tuning? *IEEE transactions on medical imaging*, 35(5):1299–1312.

Tomar, P., Kumar, G., Verma, L. P., Sharma, V. K., Kanellopoulos, D., Rawat, S. S., and Alotaibi, Y. (2022). CMT-SCTP and MPTCP Multipath Transport Protocols: A Comprehensive Review. *Electronics*, 11(15):2384.

Tu, Z., Chen, C. J., Chen, L. H., Birkbeck, N., Adsumilli, B., and Bovik, A. C. (2020). A comparative evaluation of temporal pooling methods for blind video quality assessment. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 141–145. IEEE.

Tu, Z., Wang, Y., Birkbeck, N., Adsumilli, B., and Bovik, A. C. (2021a). Ugc-vqa: Benchmarking blind video quality assessment for user generated content. *IEEE Transactions on Image Processing*, 30:4449–4464.

Tu, Z., Yu, X., Wang, Y., Birkbeck, N., Adsumilli, B., and Bovik, A. C. (2021b). Rapique: Rapid and accurate video quality prediction of user generated content. *IEEE Open Journal of Signal Processing*, 2:425–440.

Wang, Y., Inguva, S., and Adsumilli, B. (2019). Youtube ugc dataset for video compression research. In *2019 IEEE 21st International Workshop on Multimedia Signal Processing (MMSP)*, pages 1–5. IEEE.

Wang, Z., Bovik, A., Sheikh, H., and Simoncelli, E. (2004a). Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612.

Wang, Z. and Bovik, A. C. (2002). A universal image quality index. *IEEE signal processing letters*, 9(3):81–84.

Wang, Z. and Bovik, A. C. (2009). Mean squared error: Love it or leave it? a new look at signal fidelity measures. *IEEE signal processing magazine*, 26(1):98–117.

Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. (2003). Multiscale structural similarity for image quality assessment. *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, 2:1398–1402.

Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. (2004b). Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612.

Winkler, S. (2005). *Digital video quality: vision models and metrics*. John Wiley & Sons.

Winkler, S. (2017). Perceptual video quality metrics—a review. *Digital video image quality and perceptual coding*, pages 155–180.

Wu, H., Chen, C., Hou, J., Liao, L., Wang, A., Sun, W., Yan, Q., and Lin, W. (2022). Fast-vqa: Efficient end-to-end video quality assessment with fragment sampling. In *European Conference on Computer Vision*, pages 538–554. Springer.

Wu, J., Lin, W., Shi, G., and Liu, A. (2012). Perceptual quality metric with internal generative mechanism. *IEEE Transactions on Image Processing*, 22(1):43–54.

Wu, J., Liu, Y., Dong, W., Shi, G., and Lin, W. (2019). Quality assessment for video with degradation along salient trajectories. *IEEE Transactions on Multimedia*, 21(11):2738–2749.

Xu, J., Ye, P., Li, Q., Du, H., Liu, Y., and Doermann, D. (2016). Blind image quality assessment based on high order statistics aggregation. *IEEE Transactions on Image Processing*, 25(9):4444–4457.

Xu, J., Ye, P., Liu, Y., and Doermann, D. (2014). No-reference video quality assessment via feature learning. In *2014 IEEE international conference on image processing (ICIP)*, pages 491–495. IEEE.

Yeganeh, H. and Wang, Z. (2012). Objective quality assessment of tone-mapped images. *IEEE Transactions on Image processing*, 22(2):657–667.

Ying, Z., Mandal, M., Ghadiyaram, D., and Bovik, A. (2021). Patch-vq:'patching up'the video quality problem. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14019–14029.

Zach, O. and Slanina, M. (2014). A matlab-based tool for video quality evaluation without reference. *Radioengineering*, 23(1):405–411.

Zelinsky, A. (2009). Learning opencv—computer vision with the opencv library (bradski, g.r. et al.; 2008)[on the shelf]. *IEEE Robotics Automation Magazine*, 16:100–100.

Zhang, L., Zhang, L., Mou, X., and Zhang, D. (2011). Fsim: A feature similarity index for image quality assessment. *IEEE transactions on Image Processing*, 20(8):2378–2386.