Video Quality Estimation with RAPIQUE-Python – A Tutorial

Anonymous submission

Abstract

Data quality is an important consideration to the development of machine learning models as poor-quality data can slow model training, yield inadequate performance, and introduce model bias. This is particularly challenging for the development of models utilizing user generated video as human review of the entire dataset to ensure necessary quality is likely prohibitive. An alternative approach is to automate the review process to identify low-quality videos. A stand-out method for blind video quality estimation of user generated video content is the Rapid and Accurate Video Quality Evaluator (RAPIQUE) which has demonstrated nearly State of The Art (SoTA) performance in quality estimation with SoTA performance in time to inference. However, adoption of the published version of RAPIQUE is limited in part by the fact that it is coded in MATLAB, which is not as popular in the machine learning community as Python. This paper describes our open-source Python implementation of RAPIQUE, including fine-tuning for video quality estimation on videos obtained from mobile devices, and explains its use, which will enable users to easily start estimating video quality.

Introduction

It is no question that video content is the king of the internet for User Generated Content (UGC). It is equally obvious that many of the large players in the video space utilize large machine learning models that operate on these videos for recommendation systems, cataloguing, and information retrieval systems. Providing unfiltered UGC data to deep learning models is unwise as it may lead to unintended biases in such models. While these biases have been well studied in other domains such as sequence-to-sequence models where large unfiltered swaths of data are provided to the model, this is not the case for user generated videos (Bender et al. 2021; Sheng et al. 2020; Shwartz, Rudinger, and Tafjord 2020).

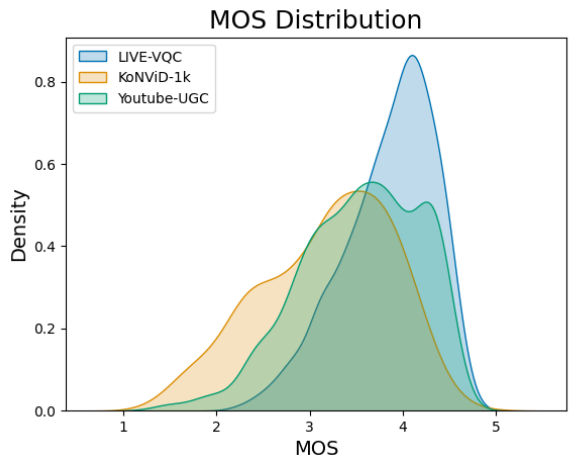
In the healthcare context, there is growing interest in collecting patient physical and behavioral health information in natural settings (Meister, Deiters, and Becker 2016). One avenue for such data collection is patient provided video recordings from which discrete digital biomarkers (DBM) may be obtained. However, poor video quality can significantly degrade DBM measurements. Therefore, automated video quality screening is an integral component necessary to large scale automated processing of videos in this context.

By filtering poor quality videos, we may mitigate these biases. Several automated video quality assessment (VQA) models have been proposed to estimate video quality in the last few years (Moorthy and Bovik 2011; Mittal, Moorthy, and Bovik 2012; Saad, Bovik, and Charrier 2014; Kundu et al. 2017). Several methods however rely on Natural Scene Statistics (NSS), which is a low-level method to extract distortions in images (Mittal, Moorthy, and Bovik 2012; Xue et al. 2014; Ghadiyaram and Bovik 2017; Tu et al. 2021a). A standout method from these is the Rapid and Accurate Video Quality Evaluator (RAPIQUE) for its balance of speed and performance (Tu et al. 2021b). To evaluate the performance of VQA models, there are three publicly available blind VQA datasets: KoNViD-1k (Hosu et al. 2017), LIVE-VQC (Sinno and Bovik 2018), and YouTube-UGC (Wang, Inguva, and Adsumilli 2019). Of the these datasets, RAPIQUE achieves rank 3 in two datasets and rank 5 in the third dataset. Where this method shines is its time to inference where it out competes all others (Tu et al. 2021b). For users that ingest large quantities of videos daily, time to inference is a key metric. RAPIQUE achieves this speed by focusing on a few low-level statistics that are quickly computed (Tu et al. 2021b).

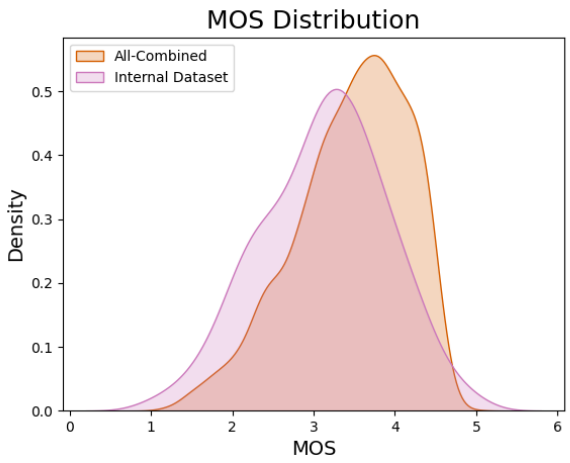
A significant limitation of RAPIQUE is that the official implementation is coded in MATLAB which makes it extremely difficult to integrate into production settings. The main contribution of this work is an implementation in Python, which is industry standard for machine learning applications, as well as validating the experimental results found in the original work. In addition to an implementation, we have created a Python package (PyPI) and an API for RAPIQUE that simplifies the use of RAPQIUE for machine learning practitioners of all levels. Lastly, we provide the ability to extract RAPIQUE features from multiple videos in parallel with RAPIQUE's application to production settings in mind.

RAPIQUE-Python API

Our RAPIQUE implementation, designated here as RAPIQUE-Python, operates primarily on directories of video files, processing each video down to a 3884 length feature vector. The RAPIQUE algorithm can be broken down into three main components, spatial feature extraction (1360 features), temporal feature extraction (476 features),



(a)



(b)

Figure 1: Subfigure (a) shows the label distribution (Mean Opinion Score) of the three Visual Quality Assessment (VQA) datasets. Subfigure (b) shows the label distribution of our internal dataset compared to the composite dataset All-Combined which is the union of the three VQA datasets.

and deep learning feature extraction (2048 features). The codebase is organized with the main driver function at the top level (`extract_rapique.py`) and the individual extractors, utilities and modeling code in their respective directories.

The main driver script, `extract_rapique.py`, is responsible for both creating the queue, which maintains the order in which each video is processed, and assigning videos to individual processing threads. By default, the number of threads is one, but our implementation allows this to scale up to an arbitrary number of processing threads.

Per the official RAPIQUE implementation, we sample 1 frame for each second of video. The sampled frame occurs halfway between each second of video (e.g. if the video is recorded at 30 frames per second, then the sampled frames will be 15, 30, 45, etc.). Each extractor utilizes the sampled frame differently. The spatial feature extractor uses the frames adjacent to the sampled frame (`cur_frame` ± 1) to compute the mean and difference between the two spatial feature vectors extracted from each frame. The temporal feature extractor extracts the next 7 frames (8 total) to apply a filter along the time axis. The deep learning feature extractor only operates on the current frame (Tu et al. 2021b).

Natural Scene Statistics

About half of RAPQIUE's total output are derived from Natural Scene Statistics (NSS) that can best be described as a simple a set of statistics computed around local decorrelated and gaussianized regions of the frame (Tu et al. 2021b). This is achieved by computing a coefficient map, $\hat{I}$, using the Means Subtraction Contrast Normalization (MSCN) method described in Equation 1 for an single channel of an image I (Mittal, Moorthy, and Bovik 2012; Kundu et al. 2017; Ghadiyaram and Bovik 2017; Zhang and Chandler 2013).

Feature Index	Description	Computation Procedure
1-2	Shape and Variance	Fit GGD to MSCN Coefficients
3-4	Mean, square reciprocal of CoV	Compute statistics on the variance from the MSCN map
5-8	Shape, mean, left variance, right variance	Fit AGGD to Horizontal pairwise products
9-12	Shape, mean, left variance, right variance	Fit AGGD to Vertical pairwise products
13-16	Shape, mean, left variance, right variance	Fit AGGD to Main-Diagonal (MD) pairwise products
17-20	Shape, mean, left variance, right variance	Fit AGGD to Secondary-Diagonal (S2) pairwise products
21-22	Shape and Variance	Fit GGD to D_1 to pairwise log-derivative
23-24	Shape and Variance	Fit GGD to D_2 to pairwise log-derivative
25-26	Shape and Variance	Fit GGD to D_3 to pairwise log-derivative
27-28	Shape and Variance	Fit GGD to D_4 to pairwise log-derivative
29-30	Shape and Variance	Fit GGD to D_5 to pairwise log-derivative
31-32	Shape and Variance	Fit GGD to D_6 to pairwise log-derivative
33-34	Shape and Variance	Fit GGD to D_7 to pairwise log-derivative

Table 1: Summary of NSS-34 features.

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

where i and j indicate the rows and columns of the image, respectively, C is a constant (typically $C = 1$) to handle divide by zero errors, and $\mu(i,j)$ and $\sigma(i,j)$ represent the weighted mean and standard deviation, respectively, of the neighborhood centered at (i,j) shown in Equations 2 and 3.

$$\mu(i,j) = \sum_{k=-K}^{K} \sum_{l=-L}^{L} w_{k,l} I(i-k, j-l) \tag{2}$$

$$\sigma(i,j) = \sqrt{\sum_{k=-K}^{K} \sum_{l=-L}^{L} [w_{k,l} I(i-k, j-l) - \mu(i,j)]^2}, \tag{3}$$

where w is a 2D gaussian weighting function (Tu et al. 2021b). The MSCN method has been shown to quantify local irregularities in single-channel intensity maps (Galdran et al. 2017). The code to compute the coefficient map is contained in Listing 1 and is found in `extractors.extract_nss_feats.get_nss_feats.py`.

A total of 34 features are computed by the NSS feature extractor all of which are computed directly from the MSCN

Dataset	KoNViD-1k			LIVE-VQC			YouTube-UGC			All-Combined		
Model	PLCC ↑	SRCC ↑	RMSE ↓	PLCC ↑	SRCC ↑	RMSE ↓	PLCC ↑	SRCC ↑	RMSE ↓	PLCC ↑	SRCC ↑	RMSE ↓
RAPIQUE	0.7548	0.7863	10.518	0.8030	0.8175	0.3623	0.7591	0.7684	0.4060	0.8070	0.8229	0.3968
RAPIQUE-Python (Ours)	0.7447	0.7642	10.926	0.7815	0.7928	0.3866	0.7923	0.7979	0.3907	0.8160	0.8238	0.4096

Table 2: Implementation validation results. Official implementation (RAPIQUE) vs our implementation (RAPIQUE-Python) on three Video Quality Assement (VQA) datasets (KoNViD-1k (Hosu et al. 2017), LIVE-VQC (Sinno and Bovik 2018), and YouTube-UGC (Wang, Inguva, and Adsumilli 2019)) and one composite dataset (All-Combined). Metrics from left to right Pearson Linear Correlation Coefficient (PLCC), Spearman Rank-order Correlation Coefficient (SRCC) and Root Mean Squared Error (RMSE). Arrow direction indicates which is better (higher or lower).

Listing 1: MSCN source code to compute Equation 1

```
filtlength = 7
window = utils.gaussian_filter(
    filtlength, filtlength/6)
window = window / (np.sum(window))
mu = scipy.ndimage.correlate(frame,
    window, mode='nearest')
mu_sq = mu * mu
sigma = np.sqrt(np.abs(correlate(frame *
     frame, window, mode='nearest') -
    mu_sq))
struct = (frame - mu) / (sigma + 1)
```

coefficient map. A summary of the NSS features are contained in Table 1. The first two features, α and σ, from the zero-mean Generalized Gaussian Distribution (GGD) are shown in Equation 4 (Mittal, Moorthy, and Bovik 2012).

$$f(x;\alpha,\sigma^2) = \frac{\alpha}{2\beta\Gamma(1/\alpha)} \exp\left(-\left(\frac{|x|}{\beta}\right)^{\alpha}\right) \tag{4}$$

where $\beta = \sigma\sqrt{\frac{\Gamma(1/\alpha)}{\Gamma(3\alpha)}}$ and $\Gamma(\cdot)$ is the gamma function. The α and σ parameters are estimated in the same manner as the official release, using a moment matching technique (Sharifi and Leon-Garcia 1995). The source code for this step is shown in Listing 2 The next two features we extract from the variance estimated in Equation 3, the mean ϕ_σ and the square of the reciprocal of the coefficient of variation (CoV) where CoV : $\rho = (\phi_\sigma/\omega_\sigma)^2$.

$$\phi_\sigma = \frac{1}{MN}\sum_{i=0}^{M-1} \sigma_{j=0}^{N-1}\sigma(i,j) \tag{5}$$

$$\omega_\sigma = \sqrt{\frac{1}{MN}\sum_{i=0}^{M-1}[\sigma_{j=0}^{N-1}\sigma(i,j) - \phi_\sigma]^2} \tag{6}$$

In addition to modeling the distribution at the center of the region, we model the distribution of the surrounding pixels: Horizontal (H) (Eq 7), Vertical (V) (Eq 8), Main-diagonal (MD) (Eq 9) and secondary diagonal (SD) (Eq 10) using a zero mode asymmetric generalized Gaussian Distribution (AGGD) (Mittal, Moorthy, and Bovik 2012; Tu et al. 2021b).

$$H(i,j) = \hat{I}(i,j)\hat{I}(i,j+1) \tag{7}$$

$$V(i,j) = \hat{I}(i,j)\hat{I}(i+1,j) \tag{8}$$

Listing 2: Source code to compute α and σ GGD paramters. Adapted from the code used in the BRISQUE paper (Mittal, Moorthy, and Bovik 2012)

```
def est_GGD_param(vec):
    # moment matching to estimate \beta
    gam = np.arange(0.1, 6, 0.001)
    gam = np.append(gam, 6)
    r_gam = (gamma(1 / gam) *gamma(3 /
        gam)) / ((gamma(2 / gam)) ** 2)
    sigma_sq = np.mean((vec) ** 2)
    alpha_par = np.sqrt(sigma_sq)
    E = np.mean(abs(vec))
    rho = sigma_sq  / (E ** 2)
    array_position = np.argmin(abs(rho -
         r_gam))
    beta_par = gam[array_position]
    return beta_par, alpha_par
```

$$MD(i,j) = \hat{I}(i,j)\hat{I}(i+1,j) \tag{9}$$

$$SD(i,j) = \hat{I}(i,j)\hat{I}(i+1,j+1) \tag{10}$$

The AGGD with zero mode is expressed by four parameters: the mean, η, the shape, v, the left variance, σ_l^2 and right variance, σ_r^2 and is given by Equation 11. For each direction, we extract these four parameters, resulting in 16 features.

$$f(x;v,\sigma_l^2,\sigma_r^2) = \begin{cases} \frac{v}{(\beta_l+\beta_r)\Gamma\left(\frac{1}{v}\right)} \exp\left(-\left(\frac{-x}{\beta_l}\right)^v\right) & x < 0 \\ \frac{v}{(\beta_l+\beta_r)\Gamma\left(\frac{1}{v}\right)} \exp\left(-\left(\frac{-x}{\beta_r}\right)^v\right) & x \geq 0 \end{cases} \tag{11}$$

where

$$\beta_l = \sigma_l\sqrt{\frac{\Gamma\left(\frac{1}{v}\right)}{\Gamma\left(\frac{3}{v}\right)}} \tag{12}$$

$$\beta_r = \sigma_r\sqrt{\frac{\Gamma\left(\frac{1}{v}\right)}{\Gamma\left(\frac{3}{v}\right)}} \tag{13}$$

and the mean is given by Equation 14.

$$\eta = (\beta_r - \beta_l)\frac{\Gamma\left(\frac{2}{v}\right)}{\Gamma\left(\frac{1}{v}\right)} \tag{14}$$

Lastly the log-derivative statistics of the six paired orientations (Equations 7-10) are computed by first transforming

Training Scheme	Trained On				Finetuned From			
Metric	LIVE-VQC	KoNViD-1k	YouTube- UGC	All-Combined	LIVE-VQC	KoNViD-1k	YouTube- UGC	All-Combined
PLCC↑	0.65	0.67	0.58	0.71	0.8	0.8	0.8	0.8
SRCC↑	0.64	0.64	0.52	0.68	0.79	0.8	0.79	0.8
RMSE↓	0.58	0.57	0.63	0.54	0.44	0.43	0.44	0.43

Table 3: Results from testing on the internal dataset. The trained on columns represent testing against the internal dataset using models obtained by training on the datasets. The finetuned from columns represent testing against the internal dataset using models obtained by performing 5-fold cross validation on the internal dataset starting from the parameters obtained in the trained on columns. By finetuning from the original models, we can obtain the same scores shown in Table 2.

Listing 3: Source code to compute η, v, σ_l^2, and σ_r^2 AGGD parameters. Adapted from the source code in BRISQUE (Mittal, Moorthy, and Bovik 2012; Tu et al. 2021b)

```
def est_AGGD_param(vec):
    gam = np.arange(0.1, 6, 0.001)
    gam = np.append(gam, 6)
    r_gam = ((gamma(2 / gam)) ** 2) / (
        gamma(1 / gam) * gamma(3 / gam))
    leftstd = np.sqrt(np.mean((vec[vec
        <0]) ** 2))
    rightstd = np.sqrt(np.mean((vec[vec
        >0]) ** 2))
    gammahat = leftstd / rightstd
    rhat = (np.mean(abs(vec))) ** 2 / np
        .mean((vec) ** 2)
    rhatnorm = (rhat * (gammahat ** 3
        +1) * (gammahat + 1)) / ((
        gammahat ** 2 + 1) ** 2)
    array_position = np.argmin((r_gam -
        rhatnorm) ** 2)
    alpha = gam[array_position]
    return alpha, leftstd, rightstd
```

the MSCN map using the log transform shown in Equation 15 (Zhang and Chandler 2013).

$$Z(i,j) = \log\left[|\hat{I}(i,j)| + 0.1\right] \tag{15}$$

The shape and variance parameters are estimated for each of the seven log-derivatives shown in Equation 16 using the same GGD method used above (Zhang and Chandler 2013), resulting in 14 features.

$$\begin{aligned}
D_1 : \nabla_x Z(i,j) &= Z(i,j+1) - Z(i,j) \\
D_2 : \nabla_y Z(i,j) &= Z(i+1,j) - Z(i,j) \\
D_3 : \nabla_{xy} Z(i,j) &= Z(i+1,j+1) - Z(i,j) \\
D_4 : \nabla_{yx} Z(i,j) &= Z(i+1,j-1) - Z(i,j) \\
D_5 : \nabla_x \nabla_y Z(i,j) &= Z(i-1,j) - Z(i+1,j) \\
&\quad - Z(i,j-1) - Z(i,j+1) \\
D_6 : \nabla_{cx} \nabla_{cy} Z(i,j)_1 &= Z(i,j) + Z(i+1,j+1) \\
&\quad - Z(i,j+1) - Z(i+1,j) \\
D_7 : \nabla_{cx} \nabla_{cy} Z(i,j)_2 &= Z(i-1,j-1) + Z(i+1,j+1) \\
&\quad - Z(i-1,j+1) - Z(i+1,j-1)
\end{aligned} \tag{16}$$

Spatial Extractor

The main idea behind the spatial extractor is to transform the image into various color/chromatic spaces and use the NSS-34 extractor on each channel of the transformed image. The original authors of RAPIQUE select seven transforms and their combinations to extract a total of 680 features (Tu et al. 2021b).

The first of these transforms converts the image to gray-scale, denoted as Y, in Equation 17. The subsequent three transforms, which consist of a number of filters, are applied to the gray-scale image. These first four transformations are referred to as the *luminance* transforms.

$$Y = (R \times 0.299) + (G \times 0.587) + (B \times 0.114) \tag{17}$$

Next, we take the Gradient Magnitude (GM) of the gray-scale image by convolving the image with a Sobel kernel as shown in Equations 18 and 19 (Tu et al. 2021b).

$$h_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \text{ and } h_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{18}$$

$$GM = \sqrt{(Y * h_x)^2 + (Y * h_y)}, \tag{19}$$

where Y is a single image channel and $*$ denotes the convolution operator. In addition to Sobel kernels, the original implementation considered Laplacian of Gaussian (LoG) (Eqn 20), and Difference of Gaussian (DoG) (Eqn 21), for their ability to characterize the receptive fields of retinal cells (Campbell and Robson 1968; Tu et al. 2021b).

$$\begin{aligned}
h_{LoG} &= \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) g_\sigma(x,y) \\
&= \frac{x^2 + y^2 - 2\sigma^2}{2\pi\sigma^6} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \\
LoG &= Y * h_{LoG}
\end{aligned} \tag{20}$$

$$DoG = Y * g_{\sigma_1} - Y * g_{\sigma_2} = Y * (g_{\sigma_1} - g_{\sigma_2}) \tag{21}$$

where $g_\sigma(x,y)$ is an isotropic Gaussian function with scale σ (Tu et al. 2021b). The code for generating these filters is contained in the `util.utils.py` file and the code has been included here in Listings 4, 5 and 6

Listing 4: Code to generate and convolve an image with a Sobel filter. Equivalent to MATLAB imgradient

```
def imgradient(frame_gray):
    sobelx = cv2.Sobel(frame_gray, cv2.CV_64F, 1, 0) # Find x and y gradients
    sobely = cv2.Sobel(frame_gray, cv2.CV_64F, 0, 1)
    # Find magnitude and angle
    magnitude = np.sqrt(sobelx**2.0 + sobely**2.0)
    angle = np.arctan2(sobely, sobelx) * (180 / np.pi)
    return magnitude, angle
```

Listing 5: Code to generate LoG filter. Equivalent to MATLAB 'fspecial'

```
def log_filter(p2, p3):
    p2 = np.array([p2, p2])
    # Translated from matlab fspecial LoG
    siz   = (p2 - 1) / 2
    std2   = p3**2
    x, y = np.meshgrid(np.arange(-siz[1], siz[1]+1), np.arange(-siz[0], siz[0]+1))
    arg = -(x*x + y*y)/(2*std2)
    h = np.exp(arg)
    eps = np.finfo(float).eps   # machine epsilon: flop acc
    h[h<eps*max(h.reshape(-1, 1))] = 0
    sumh = np.sum(h)
    if sumh != 0:
        h  = h / sumh
    # now calculate Laplacian
    h1 = h*(x*x + y*y - 2*std2)/(std2**2)
    h = h1 - np.sum(h1)/np.prod(p2) # make the filter sum to zero
    return h
```

The second set of transforms are called the chromatic transforms. These transforms shift the original RGB image into other color spaces such as $O_1O_2O_3$ (Eqn 22), red-green (RG), blue yellow (BY) (Eqn 23) and A, B from the CIELAB color space (LAB transform available in OpenCV) (Bradski 2000).

$$\begin{bmatrix} O_1 \\ O_2 \\ O_3 \end{bmatrix} = \begin{bmatrix} 0.06 & 0.63 & 0.27 \\ 0.30 & 0.04 & -0.35 \\ 0.34 & -0.60 & 0.17 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{22}$$

$$\begin{aligned} \mathcal{R}(i,j) &= \log\left[R(i,j)+0.1\right] - \mu_R \\ \mathcal{G}(i,j) &= \log\left[G(i,j)+0.1\right] - \mu_G \\ \mathcal{B}(i,j) &= \log\left[B(i,j)+0.1\right] - \mu_B \\ \hat{L} &= (\mathcal{R}+\mathcal{G}+\mathcal{B})/\sqrt{3} \\ BY &= (\mathcal{R}+\mathcal{B}-2\mathcal{B})/\sqrt{6} \\ RG &= (\mathcal{R}-\mathcal{G})/\sqrt{2} \end{aligned} \tag{23}$$

Listing 6: Code to generate DoG filter

```
def gen_DoG(frame_gray, kband):
    h, w = frame_gray.shape
    kval = 1.6
    gspace_img = np.zeros((h, w, kband))
    ksplit_img = np.zeros((h, w, kband))
    gspace_img[:,:,1] = frame_gray
    # gen gassuan pyramids
    for band in range(1, kband):
        sigma = kval ** (band - 2)
        ws = ceil(2*(3*sigma + 1))
        h = gaussian_filter(ws, sigma)
        gspace_img[:,:,band] = correlate(gspace_img[:,:,1], h, mode='nearest')
    ksplit_img[:,:,kband - 1] = gspace_img[:,:,kband - 1]
    for band in range(kband - 1):
        ksplit_img[:,:,band] = gspace_img[:,:,band] - gspace_img[:,:,band+1]
    return gspace_img, ksplit_img
```

To summarize, we have 9 image channels used in the final algorithm: Y, LoG, DoG, O1, O2, BY, RG, A and B and their respective gradient magnitudes: GMY, GMO1, GMO2, GMBY, GMRG, GMA, and GMB. For each of these image channels we extract 34 features except for the luminance maps (Y, GMY, LoG, DoG) where we extract 34 features each at the full resolution and half the original resolution. The final number of features computed for each frame of video is $(34 \times 4 \times 2) + (34 \times 12) = 680$. Recall that the total number of features extracted by the spatial extractor is 1360 due to sampling the adjacent frames (`cur_frame` $\pm$ 1) to the compute mean and difference across the features extracted from both frames.

Temporal Extractor

Recall that for the temporal extractor we sample 8 frames of the video around the sampling frame depending on its location. If the sampling frame is within 8 frames of the beginning, we sample the first 8 frames, if the sampling frame is within 8 frames of the end of the video we sample the last 8 frames. Otherwise, we sample 8 frames centered around the sampling frame. Each frame is converted to the YUV image format and only the Y channel of each frame is used in the following steps.

To extract temporal dependent features, the original method utilizes simple bandpass statistics. The original method uses a 7x8 Haar wavelet filter convolved along the time axis (Equation 24) resulting in 7 transformed frames with the same shape as the input images.

$$Y_k(x,t) = F(x,t) * h_k(t), \quad k = 0, ..., K-1, \tag{24}$$

where K is the number of filters, 7 in this case, and k indicates the current subband index (Tu et al. 2021b). For each $Y_k(x,t)$ we extract the NSS-34 at two scales just as we did with the spatial extractor resulting in $34 \times 7 \times 2 = 476$ features (Tu et al. 2021b).

Experiment	No Spatial			No Temporal			No Deep Learning		
Dataset	PLCC↑	SRCC↑	RMSE↓	PLCC↑	SRCC↑	RMSE↓	PLCC↑	SRCC↑	RMSE↓
All-Combined	0.7822	0.7947	0.4379	0.8106	0.8215	0.4104	0.8101	0.8201	0.4139
Performance Difference	**0.0337**	**0.0291**	**0.0282**	0.0053	0.0022	0.0007	0.0058	0.0036	0.0042

Table 4: Result of our ablation study on the All-Combined dataset. The first row shows the results for all metrics and the second row shows the difference between the results show in Table 2 and the first row of this table. The largest performance differences for each metric are bolded.

Deep Learning Extractor

It is well known that deep Convolutional Neural Networks (CNNs) capture informative features from images especially in the domain of image quality prediction (Ying et al. 2020; Zhang et al. 2018). The original authors of this method proposed to combine deep learned features from a general performance image classifier trained on ImageNet (Deng et al. 2009) with NSS features (Tu et al. 2021b).

For each sampled frame, the sampling frame is scaled and normalized according to the ImageNet training scheme, that is, resized to 256x256, center-cropped to 224x224 and normalized using the statistics derived from the ImageNet dataset (Deng et al. 2009). The model used is a ResNet50 and the bottleneck features (after the flatten and before the first fully connected layer) are extracted resulting in 2048 total features (Tu et al. 2021b).

Modeling and Validation

In order to validate our approach, we replicated the experimental procedures outlined in the original work (Tu et al. 2021b). Three video quality datasets were chosen for performance evaluation: KoNViD-1k (Hosu et al. 2017), LIVE-VQC (Sinno and Bovik 2018), and YouTube-UGC (Wang, Inguva, and Adsumilli 2019). Just as was done in the original work, we also utilized a Combined dataset (All-Combined), that is a concatenation of the three named datasets (Tu et al. 2021b). The main objective of VQA is to predict the Mean Opinion Score (MOS), which represents the mean of a subjective score (typically on a 1-5 scale) given by labelers, therefore, this is a regression problem. Since each dataset uses different scales for MOS (e.g. 1-10 for LIVE-VQC, 1-5 for YouTube-UGC), for the combined dataset, we adjust the labels of LIVE-VQC and KoNViD-1k to YouTube-UGC's labelling convention using Equations 25 and 26 (Tu et al. 2021b).

$$y_{\text{adjusted}} = 5 - 4 \times [(100 - y_{\text{LIVE-VQC}}/100 \times 0.7132 + 0.0253] \tag{25}$$

$$y_{\text{adjusted}} = 5 - 4 \times [(5 - y_{\text{KoNViD-1k}})/4 \times 1.1241 - 0.0993] \tag{26}$$

For each video in the dataset, the RAPIQUE feature vectors were computed. These feature vectors are used as input to a RBF SVM using a 5-fold cross validation training scheme. Missing data was assumed to be missing at random and missing values were replaced with the mean value. Train and test data were standardized using statistics from the training set. The C and γ parameters of the SVM were tuned via random search in a nested 3-fold cross validation scheme. For each parameter, we tested a range of 10 evenly spaced values on a log-2 scale. For C we tested $2^1 - 2^{10}$ and $2^{-8} - 2^1$ for γ. The best scoring parameters were used to refit the model and this model was used to evaluate the test data. This process is contained in the `modelling.fit_regressor.model_selection` function in our package and our results can be reproduced by running the `fit_regressor.py` script with a dataset name as input (e.g. `'LIVE-VQC'`).

To compare our implementation with the official repository, we use the same metrics outlined by Tu *et al.*, Pearson Linear Correlation Coefficient (PLCC), Spearman Rank-order Correlation Coefficient (SRCC), and Root Mean Squared Error (RMSE). All were computed directly after inference besides PLCC and RMSE which were computed after performing a nonlinear four-parametric logistic regression as is standard in the literature (Seshadrinathan et al. 2010; Tu et al. 2021b). To determine statistical significance, we repeated the same process using features extracted by the official method and obtained p-values using a t-test. For our experiment to be a success, our goal is to fail to reject the null hypothesis ($p > \alpha$), which is to say, there is no significant differences in the metrics achieved by the official implementation compared to ours. We choose $\alpha = 0.01$ to be our significance level. We also ran ablation studies where the process described above was repeated holding out one set of features each time (e.g. spatial features removed, trained and tested using only temporal and deep learning features).

For use in production, we sought to evaluate performance on an internal dataset consisting of videos users record in natural settings (e.g. home) using a mobile device while performing a prescribed behavioral task (Abbas et al. 2021). In accordance with the HIPPA privacy rule, this dataset will not be publicly available because it contains Protected Health Information. To this end, six labelers labelled a set of 200 videos with their opinion score. The distribution of labels for each dataset are shown in Figure 1. The labelers were "trained" using videos from KoNViD-1k, LIVE-VQC, and YouTube-UGC, where they were provided labeled examples to learn from. The opinion scores for each video were averaged to obtain a MOS for each video and then evaluated using the best performing model from each of the prior experiments (training on KoNViD-1k, LIVE-VQC, YouTube-UGC, All-Combined) following the same preprocessing steps as described above. After evaluating the performance we fine-tuned each model using the same 5-fold cross validation scheme starting from the optimal parameters found in the prior experiments and re-evaluated the per-

formance.

To evaluate the speed of our implementation, we benchmarked our method using a set of five videos from the YouTube-UGC dataset (Wang, Inguva, and Adsumilli 2019). A set of five videos were chosen at each resolution available in the dataset (360p, 480p, 720p and 1080p) and each video had a duration of exactly 20 seconds. Knowing that Python is inherently slower than MATLAB, we chose to implement RAPIQUE-Python with support for multiple threads extracting in parallel. To that end, we measure the extraction time of each video set with various parallel threads (1-5).

For our ablation and finetuning experiments we scaled all labels using Equation 25 and 26 so that the metrics were all on the same scale.

Results

Table 2 shows the evaluation of the same baselines as the official implementation. It is clear our implementation is very close and superior in some instances but it is not obvious if they are significantly different. Using a two-way t-test, it was found that for all tests, $p > 0.01$. In this case we fail to reject the null hypothesis therefore there is no significant difference between the features extracted by our implementation and the official implementation. For each dataset we found the optimal hyperparameters of our SVM to be: LIVE-VQC: $C = 16$, $\gamma = 2^{-8}$, KoNViD-1k: $C = 2$, $\gamma = 2^{-8}$, YouTube-UGC: $C = 32$, $\gamma = 2^{-8}$, All-Combined: $C = 2$, $\gamma = 2^{-8}$,

Table 3 shows our method's performance estimating MOS on our internal dataset. Without finetuning there was a greater than 25% reduction in performance (RMSE) compared to the All-Combined benchmark. By finetuning we were able to bring our performance back up to our benchmark scores from Table 2.

Table 4 shows the result of our ablation study. The spatial features were the component that caused the largest decrease in performance, however, the performance difference is not actually a very large. This indicates that each component contributes meaningfully to the whole method. We postulate that there must be large overlap in the features extracted by each component resulting in redundant features. We leave a feature selection analysis for future work.

The result of our speed benchmarking experiment are shown in Figure 2. These results were obtained using an Intel Xeon E5-2686 CPU and a Nvidia Tesla K80 GPU. The official implementation reports an average extraction time of 17.3 seconds for a second of 1080p video (Tu et al. 2021b). In Figure 2 we show that with multithreading enabled we can perform the same extraction in about 18 seconds with lower performing hardware than was reported in the official work (Tu et al. 2021b).

Conclusion & Future Work

In this paper, we present a Python implementation of the RAPIQUE VQA algorithm. We validated our implementation by replicating experiments from the official release and found that there were no significant differences between the two. Next, we showcased how to adapt general performance

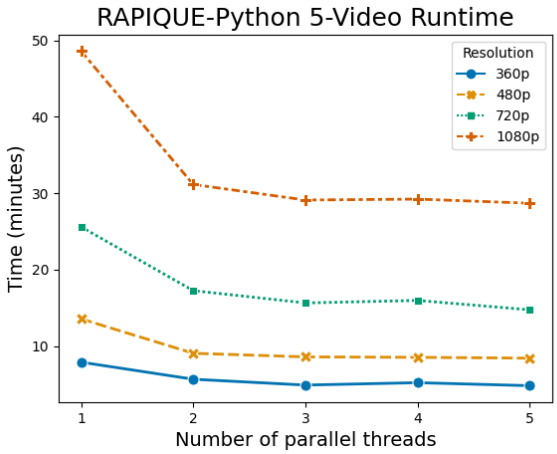


Figure 2: This figure shows the time to complete RAPIQUE feature extraction for a set of 5 videos as a function of the number of parallel processing threads and video resolution. The videos used to benchmark were taken from the YouTube-UGC dataset (Wang, Inguva, and Adsumilli 2019). Each video was approximately 20 seconds long.

MOS prediction models to a very specific subset of data using our internal dataset. We discussed our API and how each component of the method interacts with one another. Finally, in our ablation study, we examined the implications of redundant features and plan to study this in our next work in an effort to speed up the method even further. Upon publication our GitHub and PyPI package will be made public and we will link to them here.

References

Abbas, A.; Sauder, C.; Yadav, V.; Koesmahargyo, V.; Aghjayan, A.; Marecki, S.; Evans, M.; and Galatzer-Levy, I. R. 2021. Remote digital measurement of facial and vocal markers of major depressive disorder severity and treatment response: a pilot study. *Frontiers in digital health*, 3: 610006.

Bender, E. M.; Gebru, T.; McMillan-Major, A.; and Shmitchell, S. 2021. On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, 610–623.

Bradski, G. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Campbell, F. W.; and Robson, J. G. 1968. Application of Fourier analysis to the visibility of gratings. *The Journal of physiology*, 197(3): 551.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, 248–255. Ieee.

Galdran, A.; Araújo, T.; Mendonça, A. M.; and Campilho, A. 2017. Retinal image quality assessment by mean-subtracted contrast-normalized coefficients. In *European*

Congress on Computational Methods in Applied Sciences and Engineering, 844–853. Springer.

Ghadiyaram, D.; and Bovik, A. C. 2017. Perceptual quality prediction on authentically distorted images using a bag of features approach. *Journal of vision*, 17(1): 32–32.

Hosu, V.; Hahn, F.; Jenadeleh, M.; Lin, H.; Men, H.; Szirányi, T.; Li, S.; and Saupe, D. 2017. The Konstanz natural video database (KoNViD-1k). In *2017 Ninth international conference on quality of multimedia experience (QoMEX)*, 1–6. IEEE.

Kundu, D.; Ghadiyaram, D.; Bovik, A. C.; and Evans, B. L. 2017. No-reference quality assessment of tone-mapped HDR pictures. *IEEE Transactions on Image Processing*, 26(6): 2957–2971.

Meister, S.; Deiters, W.; and Becker, S. 2016. Digital health and digital biomarkers–enabling value chains on health data. *Current Directions in Biomedical Engineering*, 2(1): 577–581.

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

Moorthy, A. K.; and Bovik, A. C. 2011. Blind image quality assessment: From natural scene statistics to perceptual quality. *IEEE transactions on Image Processing*, 20(12): 3350–3364.

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.

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

Sharifi, K.; and Leon-Garcia, A. 1995. Estimation of shape parameter for generalized Gaussian distributions in subband decompositions of video. *IEEE Transactions on Circuits and Systems for Video Technology*, 5(1): 52–56.

Sheng, E.; Chang, K.-W.; Natarajan, P.; and Peng, N. 2020. Towards controllable biases in language generation. *arXiv preprint arXiv:2005.00268*.

Shwartz, V.; Rudinger, R.; and Tafjord, O. 2020. " You are grounded!": Latent Name Artifacts in Pre-trained Language Models. *arXiv preprint arXiv:2004.03012*.

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

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)*, 1–5. IEEE.

Xue, W.; Mou, X.; Zhang, L.; Bovik, A. C.; and Feng, X. 2014. Blind image quality assessment using joint statistics of gradient magnitude and Laplacian features. *IEEE Transactions on Image Processing*, 23(11): 4850–4862.

Ying, Z.; Niu, H.; Gupta, P.; Mahajan, D.; Ghadiyaram, D.; and Bovik, A. 2020. From patches to pictures (PaQ-2-PiQ): Mapping the perceptual space of picture quality. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 3575–3585.

Zhang, Y.; and Chandler, D. M. 2013. No-reference image quality assessment based on log-derivative statistics of natural scenes. *Journal of Electronic Imaging*, 22(4): 043025.

Zhang, Y.; Gao, X.; He, L.; Lu, W.; and He, R. 2018. Blind video quality assessment with weakly supervised learning and resampling strategy. *IEEE Transactions on Circuits and Systems for Video Technology*, 29(8): 2244–2255.