DOI: 10.2352/ISSN.2470-1173.2016.2.VIPC-237

Optimizing Transcoder Quality Targets Using a Neural Network with an Embedded Bitrate Model

Michele Covell, Martín Arjovsky†, Yao-chung Lin, Anil Kokaram
Google Inc., Mountain View, California 94043, USA

Abstract

Like all modern internet-based video services, YouTube employs adaptive bitrate (ABR) streaming. Due to the computational expense of transcoding, the goal is to achieve a target bitrate for each ABR segment, without requiring multi-pass encoding. We extend the content-dependent model equation between bitrate and frame rate [6] to include CRF and frame size. We then attempt to estimate the content-dependent parameters used in the model equation, using simple summary features taken from the video segment and a novel neural-network layout. We show that we can estimate the correct quality-control parameter on 65% of our test cases without using a previous transcode of the video segment. If there is a previous transcode of the same segment available (using an inexpensive configuration), we increase our accuracy to 80%.

Motivation and Background

Like other over-the-top (OTT) media services, YouTube uses ABR streaming to make the most of the available bandwidth at the user's client player. ABR streaming (e.g., MPEG DASH [9]) allows the client to switch between alternate streams (*representations* in DASH) which encode the same content at different bitrates. Each representation is encoded such that short duration temporal *segments* are independently decodable. The option to switch between the different bitrate streams is therefore available only at the end of these segments. Target bitrates for each representation are chosen so that the user perceives a smoothly varying stream quality as the bitrate varies.

For large OTT sites, transcoding to support these schemes is extremely resource intensive. This is due both to the sheer volume of video that must be transcoded[1] and to the fact that multiple representations must be created from each single input file.

A simple codec-agnostic technique for increasing throughput in proportion to available computational resources is to split each input clip into a number of segments which are then encoded in parallel. For DASH compliant streams, each encoder operates under the constraint that the bitrate is less than some specified maximum. Unfortunately, this parallel encoding process can result in artifacts that manifest as a large discontinuity between the picture quality at the start and at the end of the segment. The changes in picture quality is a result of a single pass attempt to achieve a given average bitrate over each segment independently. The transcoder is unable to correctly estimate the quality settings for the required bitrate and adjusts the encoding as it goes. As this happens on each segment, the viewer observes this as a cycle of picture quality from bad to good at intervals equal to the segment duration.

The problem is exacerbated when segments are short (on the order of seconds) which is vital for low-latency cloud-based applications like YouTube. At the core, this problem is due to issues with rate control in the encoding process. This could be mitigated by propagating transcoding statistics (features) for each segment through the system. However, in a parallel encoding system, we wish to minimize or eliminate the need for information to be communicated between processing nodes, to allow wide (and independent) deployment across general-purpose CPU farms. Communication between these separate jobs would greatly complicate their deployment and would increase the impact of isolated transcoding slow-downs or failures. By deploying completely independent transcoding jobs, each job can be restarted without any of the other transcoded segments being affected.

Equally important to our effort is to avoid making deep changes in any part of the codec implementation. We need to be able to treat our codec implementation as a commodity, to be replaced and upgraded as better versions or implementations become available. For this reason, our approach sets a single segment-level rate-control parameter. If we were, instead, to modify with picture- or macroblock-level quantization processes, our system would require continual maintenance as the deployed codec changed. By remaining at the level of a single external parameter, one that has a clear tie to a basic property of the transcode (the quality/bitrate control), the most that we will need to do upon changing codecs will be to retrain our neural-network-based control process.

Within those constraints, our goal is to achieve stable frame quality throughout each segment as well as the desired bitrate for the segment as a whole. We consider the use of the x264.org codec to produce DASH compliant streams in a cloud-based environment. x264 is now the most common open source codec used in the video streaming industry, has a high throughput and is a reference for high performance. Multi-pass constant bitrate encoding would appear to satisfy the quality requirement. But, as we have shown in previous work [5], this is only successful if some effort is expended on optimising codec-specific quality settings at each pass. Given that multi-pass encoding clearly increases the computational resources required to encode a segment, we wish to reduce or eliminate the need for multiple passes.

This paper explores the use of a neural network for predicting the parameters of a model that relates bitrate to various video properties. We show that we can estimate the correct codec parameter on 65% of our test cases without using a previous transcode of the video segment. If there is a previous transcode of the same segment available (using an inexpensive codec configu-

†Martín Arjovsky is currently at the University of Buenos Aires, Argentina.

[1]At YouTube, 300 hours of video is uploaded every minute of every day [11].

DOI: 10.2352/ISSN.2470-1173.2016.2.VIPC-237

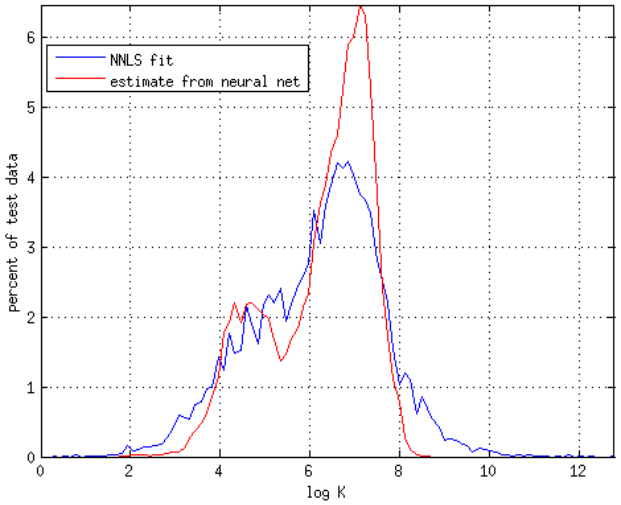


Figure 1: Distributions of $\log K$ estimates, as found by NNLS fit per video block (blue) and by the mid-network layer (red).
(mean: 6.15; std: 1.44; min: 0.22; max: 12.84)

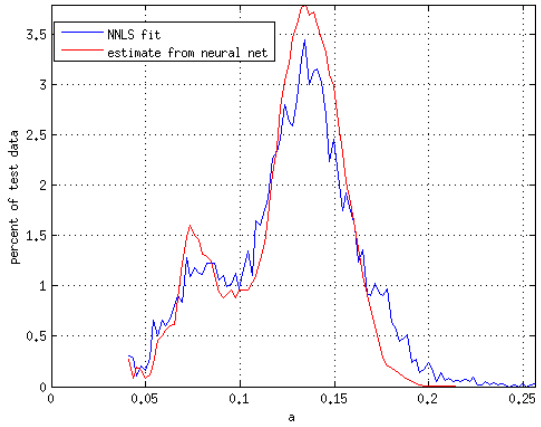


Figure 2: Distributions of a estimates, as found by NNLS fit per video block (blue) and by the mid-network layer (red).
(mean: 0.126; std: 0.034; min: 0.04; max: 0.257)

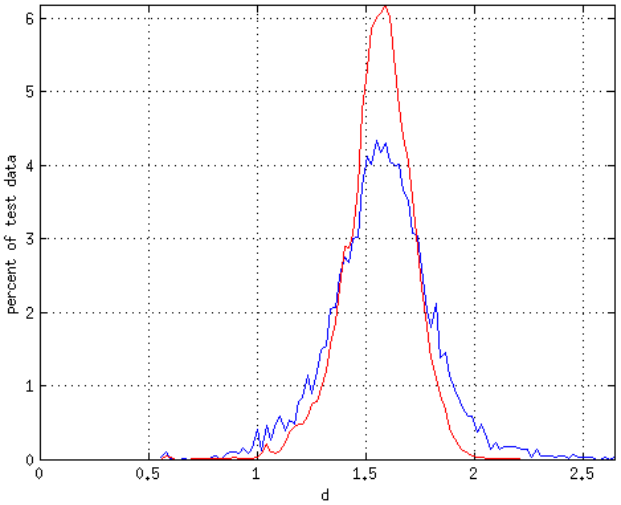


Figure 3: Distributions of d estimates, as found by NNLS fit per video block (blue) and by the mid-network layer (red).
(mean: 1.57; std: 0.23; min: 0.55; max: 2.65)

ration), we increase our accuracy to 80%.

The next section discusses the key points with respect to rate control for each segment. We then go on to introduce the models used and to present the results of our neural-network training.

Transcoding Configurations and Parameters

We explore resolution-dependent ABR in which each representation is at a different resolution and a different bitrate. In this formulation when the client player switches representations, the stream changes resolution and bitrate.

The spatial resolution of each video representation is easy to control: we simply tell the transcoder our target resolution. The bitrate of each video segment is more difficult to control in a way that provides the best output quality. This is especially true when doing single-pass compression. In x264 single-pass transcoding, the best (perceptual) video quality (for the bandwidth) is achieved by controlling the quantization levels indirectly, using what is called the *Constant Rate Factor (CRF)*.[2] Using CRF has the advantage that it adjusts the quantization parameters to take advantage of motion masking. The general idea is that mistakes are most noticable in smooth regions with good inter-frame prediction, so the CRF spends more of its bits on these regions [7].

Unfortunately, with CRF, there is no direct control of the actual bitrate that is used over the segment of video that we are transcoding. The same CRF parameter settings will yield widely different bitrates, when applied to different videos or even to different segments within a single video. Since we require a single transcode per resolution and a known target bitrate ($\pm$20%), we need to estimate the relationship between the bitrate and CRF for each video segment *before* we start transcoding that segment. In the next section, we discuss a model to relate CRF to bitrate, and help us towards this goal.

Modeling the Effect of Transcoding Parameters on Bitrates

In their 2012 article, Ma et al. [6] found that, for a given segment of video and a fixed frame size, they could accurately predict the bitrate by relating it to the frame rate and the quantization step size using the equation

$$R(q,t,v) = R_{\max}(q_{\min},t_{\max},v)\left(\frac{q}{q_{\min}}\right)^{-\alpha(v)}\left(\frac{t}{t_{\max}}\right)^{\beta(v)}$$

where $q_{\min}$, $t_{\max}$, and $R_{\max}(q_{\min},t_{\max},v)$ are all taken from a previous transcode of the same video material, v, and are the previous transcode's quantization step size, frame rate, and bitrate, respectively. The values of $\alpha(v)$, $\beta(v)$, and $R_{\max}(q_{\min},t_{\max},v)$ are transcoder and content dependent but independent of the desired quantization step size (q) and frame rate (t). On the test sets used in [6], the values of $\alpha(v)$ and $\beta(v)$ generally varied by 70% or less, while the value for $R_{\max}$ changed by as much as 13 times, depending on content.

We have discovered a similar relationship between bitrate, CRF, frame resolution, and frame rate:

$$\log R(c,t,h,v) = \log K(v) - a(v)c + b(v)\log t + d(v)\log h \quad (1)$$

[2] Throughout this paper, the acronym CRF will be used used to refer to this compression parameter and *not* Conditional Random Field.

DOI: 10.2352/ISSN.2470-1173.2016.2.VIPC-237

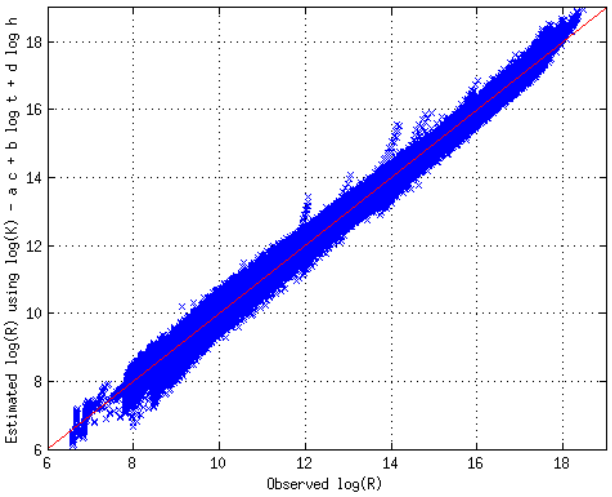


Figure 4: Quality of fit of estimated $\log R(v)$ using NNLS-fit values for $\log K(v)$, $a(v)$, and $d(v)$.
(Pearson's correlation: 0.9984. Error std.: 0.1. Max error: 1.41)

where $R(c,t,h,v)$ is the predicted bitrate for a segment of video v, given the requested CRF setting, c; the requested frame rate, t; the requested frame height, h,[3] and where $K(v)$, $a(v)$, $b(v)$, and $d(v)$ are hidden variables that are dependent only on the video content v.

We tested this relationship on 9,250 5-second video segments, sampled from 1000 of the videos that had been recently uploaded to YouTube. We selected up to 5 different frame heights (from 240 to 1080 but omitting heights that are larger than the original video height) and 29 different CRF settings (from 12 - 40), resulting in up to 145 different samples of each video segment.[4] For each of the 9,250 video segments, we found the best fit for $\log K$, a and d, using the non-negative least squares routine that is included in SciPy [4, 8]. None of the bitrate-model parameters ($\log K$, a, b, and d) can be valid when less than zero.[5] Since Ma et. al [6] had already shown the log-linear relationship between bitrate and frame rate, we did not increase our testing data by repeating that part of the experiment. The blue curves in Figures 1 through 3 show the distributions that we found from the NNLS fit to our training data.

We show the quality of the bitrate estimation in Figure 4. We get the same Pearson correlation as was found in that earlier work [6], but with a more complex quantization process (CRF instead of quantization step size) and with more factors accounted for (including frame size). Our Pearson coefficient was 0.9984 (compared to 0.9985-0.9991 reported in [6]).

The blue curve in Figure 9 shows the cumulative distribution across our test set of the percentage error in the predicted bitrate. This uses the NNLS-fit bitrate-model parameter values for $\log K(v)$, $a(v)$, and $d(v)$. With these bitrate-model parameters, we are able to estimate the correct CRF for our target bitrate (within $\pm 20\%$) on 95% of our test cases. While this approach is not possible in practice (it requires information from 145 transcodes of the same content for the fitting process), it provides a guideline for the best possible performance using this approach.

Understanding that these hidden parameters model the relationship between bitrate and CRF values is not immediately useful. Instead, we would like to estimate these bitrate-model parameters as part of a process for estimating the correct CRF to use for our target bitrate using video features that are available to use from earlier processing in the upload pipeline. We discuss this possibility in the next section.

Features from the Mezz

Files uploaded to YouTube are completely unconstrained. Even though unusual filetypes and bitstreams (e.g. variable frame rates, colorspaces, incorrect containers etc) are encountered for a small fraction of these uploads, the volume of ingest means that these edge cases can challenge the robustness of the transcoding system. Therefore, we normalize files before transcoding by creating a high-bitrate, constant-frame-rate mezzanine. We refer to the output of this re-encode as our *mezz*. The process of mezz creation gives us access to encoding properties that characterise the video stream. We collect the following features (accumulated over each video segment) from the creation of the mezz bitstream, for prediction of each segment's bitrate-model parameters:

- average number of bits used for the move vector (MV) per predicted (P) macroblock (MB)
- average number of bits used for the texture (i.e., pixel values or prediction error) per MB
- average number of bits used for the texture per intraframe (I) MB
- average number of bits used for the texture per P MB
- percentage of I MBs
- percentage of skipped MBs
- a score on the complexity of the video encoding for this segment
- average quantization-parameter setting used in the mezz transcode

We also include properties of the original upload: bitrate, frame size (width and height), and frame rate of the original upload. In addition, we include 4 features relevant to our target output: target bitrate, target frame size (width and height), and target frame rate.

As we will describe in the next section, we use a compact, shallow neural network (only about 200 hidden units) for our estimation process. We work to keep our network small, since we expect to run it on all videos uploaded to YouTube: the smaller the network, the lower the expense of estimating the correct CRF. However, this small size suggests that we provide as rich an input feature set as we can. In addition to the 16 features, extracted directly from the mezz transcode and the target settings, we include values derived from these original features.

We provide as input the logarithm of original and target bitrates, frame rates, frame heights, and frame widths. The selection of these features for a log operation is suggested by the appearance of those same logs in Equation 1.

[3] We assume that the width is set from the requested height, using the same aspect ratio as the input video.

[4] Just to be clear, the 29 different CRF settings per frame resolution is for training and testing purposes only. Normally, there are only one transcode per segment, per frame resolution.

[5] We tested for the possibility that K could be less than one, by running the NNLS with a shifted parameterization of $\log K$ but we did not find any cases of the fit being better for such small values of K.

DOI: 10.2352/ISSN.2470-1173.2016.2.VIPC-237

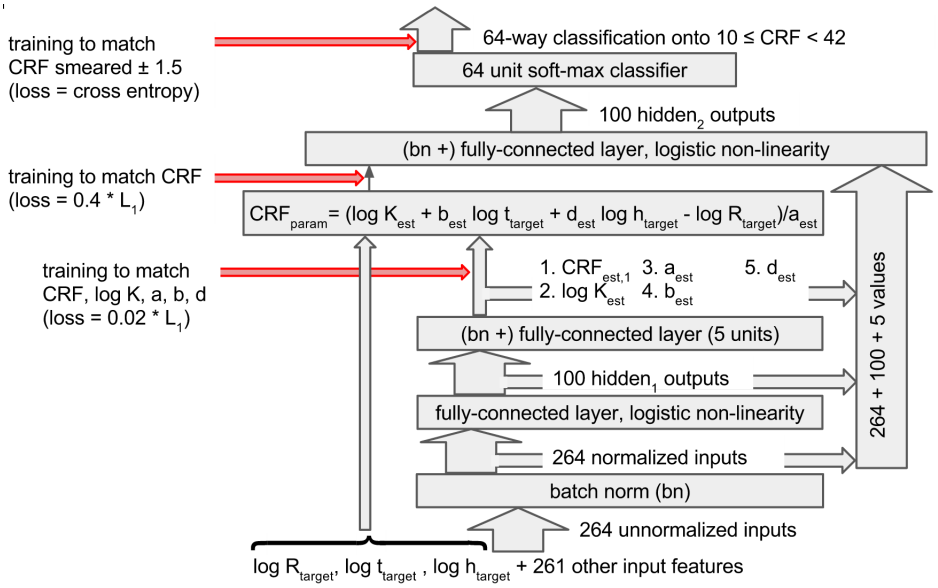


Figure 5: Layout of neural network used to estimate CRF

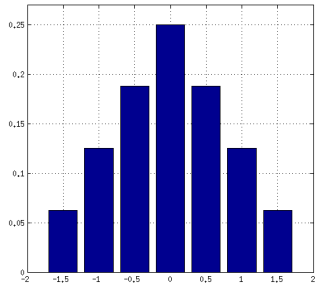


Figure 6: Label smearing used for going from actual CRF of the transcode to a multi-class label. By smearing over ± 1.5 CRF units, we lessen the categorization penalty for bitrate errors that are less than 20%

Finally, we repeatedly cross multiply the target frame rate, the target height, the percentage skipped MB, and the percentage I MB with the other 12 features. Conceptually, we do this up to three times (creating terms of a fourth order dependency on the original features) but remove the terms where any of the input features, other than target height, are repeated more than once. We allow target height to repeat up to two times in a single term, as an approximation to frame size. For example, this set will include (target height squared) times (average texture bits per I MP) times (percentage I MB). We do this to allow the neural network with easy access to the volume measures (in addition to the average measures listed in the original input features). After removing duplicates, our final input vector size is 264 entries.

In the next section, we discuss the neural net that we use to process these inputs, as well as how we use our model equation (Equation 1) to guide its training.

Neural-Network Estimation of x264 CRF

With our study of Equation 1, we showed that, if we somehow had access to the values for $\log K(v)$, $a(v)$, $b(v)$, and $d(v)$ for a given video block v, we should be able to predict the correct setting for CRF for a desired frame rate and our prediction would accurate, to within our 20% bitrate margin, 95% of the time. However, we do not have the results of other transcodes of v, other than the features coming from the mezz transcode, as described in the previous section. In this section, we describe the neural network that we created to try to predict the needed CRF, starting from the mezz-transcode features.

We use the layout shown in Figure 5. Starting from the bottom of the figure, we input the 264 features discussed in the previous section. These include the logarithms of the target bitrates, frame rate, and frame height, values that we use deeper in our network.

Our first "layer" completes batch normalization on all 264 inputs. Batch normalization [2] is a technique that improves and accelerates training convergence by estimating the mean and variance of each input and outputting an (approximately) zero mean and unit variance version of the input activation. While this normalization could be done as a pre-processing step on the input features, we avoid a separate pre-processing step by including it in our network. Also, we will use this batch-normalization technique on the internal network activations that act as inputs to learning layers.

The normalized input features are fed into a 100-unit, fully-connected layer with a logistic-function nonlinearity on the outputs. We feed those output activations through batch normalization and then into a 5-unit, fully-connected linear-response layer. Up to this point, our neural network structure is fairly conventional.

At this point, during training, we introduce a loss "side-training" layer [10] to train the outputs of these 5 linear units. Our training uses an L_1 error measure between the units and our training labels for CRF, $\log K$, a, b, and d. We also form the compound terms that are needed for estimating the CRF using Equation 1 and the three un-normalized inputs that we feed to this layer ($\log R_{\text{target}}$, $\log t_{\text{target}}$, and $\log h_{\text{target}}$). We add an L_1-loss side-training layer for each of the terms ($\log K/a$, $b \log t_{\text{target}}/a$, and $d \log h_{\text{target}}/a$, and $\log R_{\text{target}}/a$) and for their CRF-estimating combination ("$\text{CRF}_{\text{param}}$" in Figure 5). For all of these side-training layers, we downweight the L_1 error terms so that they are, in aggregate, less important in training than the final categorization error that we discuss below.

This type of mid-network interpretation of unit outputs is unusual. Most neural networks work best if left to create their own intermediate variables. However, in our experiments with networks that did not include these extra constraints based on our bitrate model, the networks did not converge to a solution, even after a week of training. With the mid-network training, we see 95% covergence in less than a day (with continued improvement for about 4 days of training total). Also, by feeding not only these values but all the inputs and the previous hidden-layer outputs into our next layer, we allow the network the freedom to still create intermediate variables that are not constrained by the terms that we have explicitly trained.

DOI: 10.2352/ISSN.2470-1173.2016.2.VIPC-237

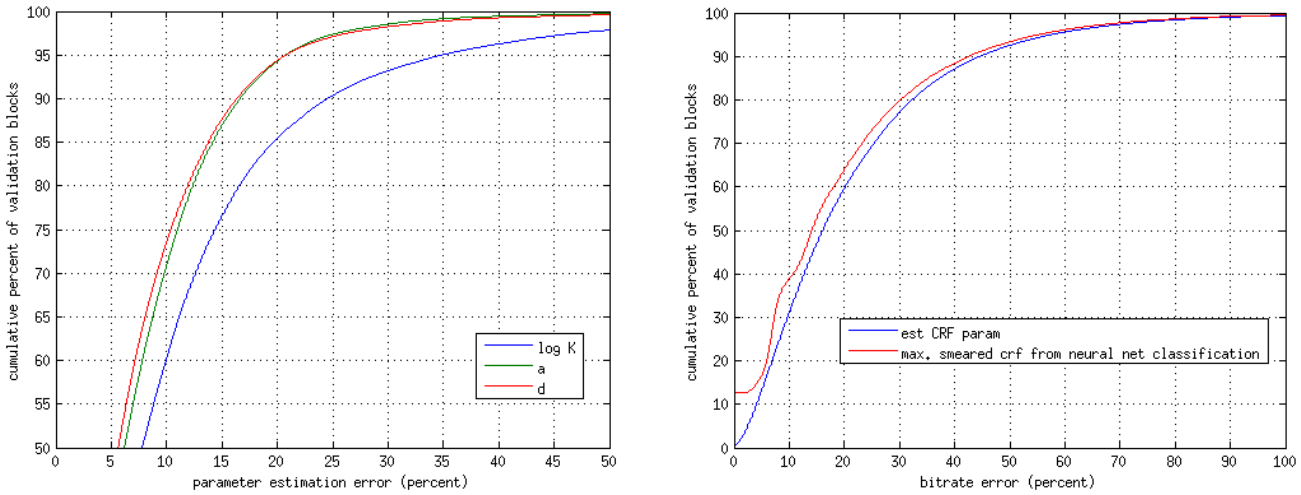


Figure 7: Both: cumulative distributions of percent errors across the test set. Left: Errors in the bitrate-model parameter estimation. Right: Bitrate error of the mid-network "$\text{CRF}_{\text{param}}$" (see Figure 5), with the final error curve shown in red for reference.

We feed all of our layers' outputs as well as our input features into our final hidden layer (after batch normalization) and follow that by a classifier. These two additional layers are an effort to improve our final CRF, over what we would otherwise get, given the mistakes we make in estimating $\log K$, a, b, and d from the features observed from the mezz-transcode features. Our final 100-unit hidden layer allows us to try for this by getting all of the information from all the previous layers (as well as our CRF, term, and bitrate-model parameter estimates from the layers with side training).

We use a 64 unit soft-max classifier as our final layer for two reasons. The first is that it allows the neural net more freedom in how it estimates each section of the CRF range: each of the units is only responsible for deciding if the target CRF is likely to be within its 0.5-unit range, instead of trying to come up with an accurate numerical estimate for values that range from 10 to 42. The second reason is one that we will discuss more in our future-work section but is basically that this output architecture will give us easy ways to estimate our certainty in our answer. In the simplest case, we can look at the activation of the soft-max output layer as a probability distribution for the different CRF values.

To create output classification labels for training and testing, we represented each transcode as having a multi-class label around the actual CRF value. The smearing uses the triangle smearing shown in Figure 6. This label smearing allows for small errors in CRF estimates at less-than-full penalty, up to an error of 1.5 units. Since we can miss our target bitrate by up to a 20% and remain within our acceptable zone, we allow the neural network some of that leeway, even in training, by smearing our target labels. For our training and test data, we found that 2 CRF units corresponds to about 20% bitrate error: the content-specific relationship between absolute CRF error and percentage bitrate error will depend on the exact value for a but that distribution is reasonably compact around $a = 0.1$ (see the blue curve in Figure 2), so this content-independent approximation should be reasonable for label smearing.

Results from Neural Network CRF Estimation

To collect the training and test data for the network described above, we pulled segments from a total of 15,000 distinct videos: 14,000 of them were used for training and 1000 for testing. The clips were at least 20 seconds long but not more than 100 seconds (to avoid over-representation). Also, we were careful to keep all of the segments in each video together, either in the training data or in the testing data but not split across both. Without this strict segregation, our test results would have been overly optimistic, since neighboring segments often have very similar bitrate-model parameter values. These groups gave us over 137 thousand distinct training video segments and over 9 thousand testing video segments. On each block, we computed up to 145 distinct transcodes (up to 5 different spatial resolutions and 29 different CRF settings), giving over 15 million training transcodes and over 1 million testing transcodes. On each block of the training data, we separately completed NNLS fits to the available transcodes, to determine "ground-truth" values for $\log K$, a, and d, to be used in training. Since our training data did not include frame-rate variation within a single block of data, we pre-fit a single value for b, for all video segments, by linear regression across all of the training data.

We train the network using Adagrad [1] adaptive learning rates. To help reduce over-fitting, we include a small amount of weight decay [3] in the training process (with an L_2 penalty weight of 10^{-5}).

The red line in Figure 9 shows the results from our neural network on the test set. We keep the bitrate error below 20% on 65% of our test-set samples. This is a huge improvement (more than four fold) over the content-independent approach to setting CRF. In addition, it does not cost more than the content-independent approach (other than the very low computational cost of running the neural network): there is no added dependencies and no additional, supporting transcoding.

Figure 7 shows that, while we do fairly well at estimating a and d (about 95% of these estimates are within 20% of truth), we are much worse at estimating $\log K$. The histograms of estimated and actual bitrate-model parameter distributions reinforce this finding: while the distributions estimated for a (Figure 2) and

©2016 Society for Imaging Science and Technology
DOI: 10.2352/ISSN.2470-1173.2016.2.VIPC-237

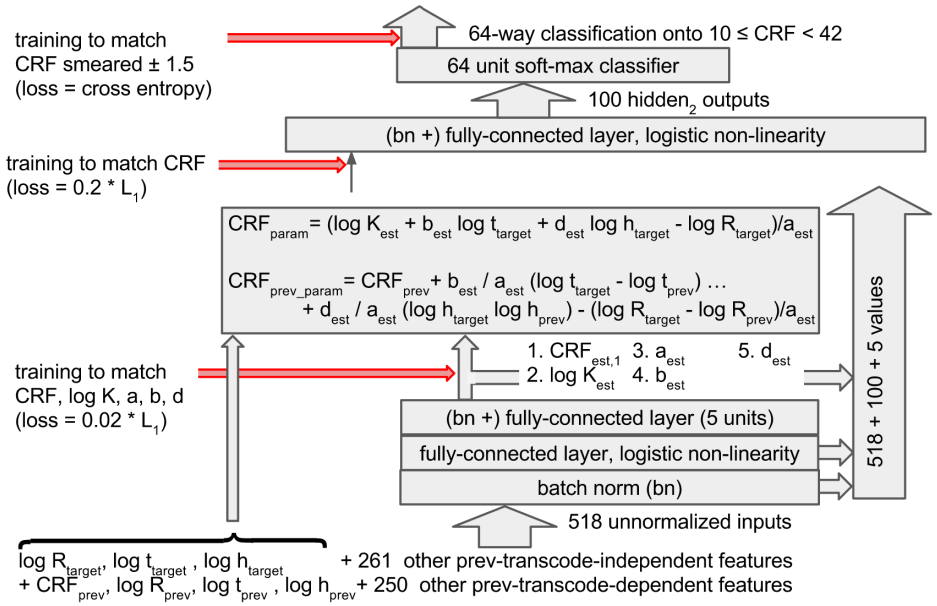


Figure 8: Layout of neural network used to estimate, when a previous transcode is available

d (Figure 3) have the same general shape as the NNLS-fit values, the estimated distribution for $\log K$ (Figure 1) is qualitatively different from that of the ground truth.

In our next section, we will discuss the results of using information from a single previous transcode of the same video segment as a way to lessening our reliance on estimating $\log K$.

CRF Estimation With Previous Transcode

As we saw in the previous section, our ability to correctly estimate CRF is limited (at least in part) by our difficulty in estimating $\log K$ for Equation 1. We can avoid complete reliance on that estimate by providing, as inputs, the features from a single previous transcode of the same video segment. In that case, we can subtract two instances (from the same v) of Equation 1, which removes $\log K$ from the equation and instead gives:

$$\begin{aligned}\log R(c,t,h,v) = \log R(c_{\text{prev}},t_{\text{prev}},h_{\text{prev}},v)\\ -a(v)(c-c_{\text{prev}})+b(v)(\log t-\log t_{\text{prev}})\\ +d(v)(\log h-\log h_{\text{prev}})\end{aligned} \quad (2)$$

We extended our neural net to allow this approach, as shown in Figure 8. For the input layer, we added another 254 input features, including $\log R(c_{\text{prev}},t_{\text{prev}},h_{\text{prev}},v)$, c_{prev}, t_{prev}, and h_{prev}.[6] All of these 518 (= 264 + 254) inputs are fed, after batch-norm processing to the two fully-connected, logistic-activation layers. In the mid-network layer, where previously we estimated "$\text{CRF}_{\text{param}}$", we add separate, parallel estimate of CRF, "$\text{CRF}_{\text{prev\_param}}$", using the formula given in Equation 2. We then train both of these estimates for CRF using an L_1 loss between each estimate and the training labels and fed both estimates upward to the next layer. The rest of the network remains the same.

Figure 9 shows our results when we use the lowest-resolution, most quantized version of the transcode set that we have available (240 pixel frame height and CRF = 40) as our previous transcode. We picked this transcode configuration since it is the least expensive to complete and so should add the least delay to the other transcodes. To avoid positive bias in our results, we do not include the transcodes that were used as the previous transcodes in our test set: they are not included in the target situation that we are trying estimate.

With this extra data, the CRF estimates are correct, to within 20% bitrate error, for 80% of the test set. This halves the distance between the NNLS curve and our no-previous-transcode estimated curve. The gain over the content-independent approach is even more impressive. Instead of needing to re-transcode to get a correct bitrate on 85% of the configurations (according to the content-independent approach), we have this issue on only 20% of the configurations.

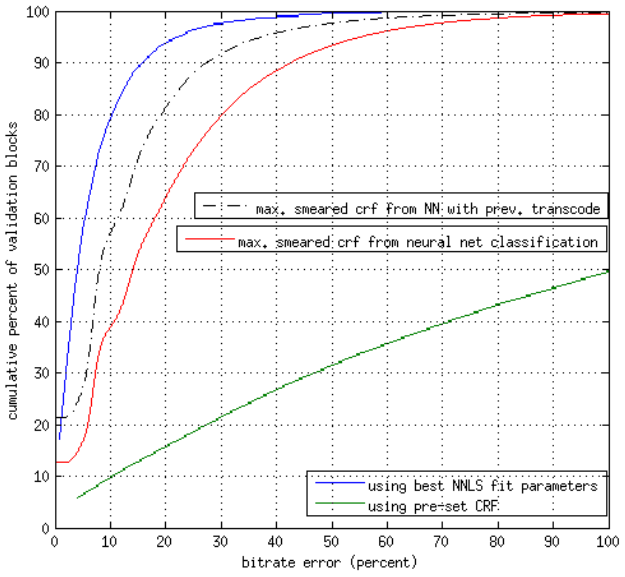
Conclusions

As shown in Figure 4, Equation 1 predicts bitrate quite well, using four content-dependent but compression-configuration-independent parameters ($\log K$, a, b, and d). Having discovered this, we can use the structure provided by that model to guide our neural-network training to estimate the correct CRF value for a desired bitrate. Without knowledge of this bitrate model, our attempts at training a neural network for this task failed to converge: it could not find the correct transformation from our input features to an accurate estimate for CRF.

The neural-network layout shown in Figure 5 achieves an accuracy of 20% bitrate error (or better) on 65% of our tests. While this is not as consistently accurate as we would like, it is much more accurate than the obvious alternative (using a content-independent mapping from desired bitrate to CRF): that approach has only 15% of the segments at or below the 20% bitrate-error target. We can improve the quality of our neural-network estimate for CRF, if we introduce a dependence on a previous transcode. Even using the least-expensive transcode (low-resolution and low-bitrate) improves our estimate's accuracy to the point that we have

[6] The number of additional features is less than the original set, since we take out features that depend only on original upload: they would be redundant.

DOI: 10.2352/ISSN.2470-1173.2016.2.VIPC-237



Green: Fixed (content-independent) mapping from CRF to bitrate.
Blue: NNLS-fit for $\log K(v)$, $a(v)$, and $d(v)$ (not possible in practice, included for best-case reference).
Red: From highest-activation CRF label from the network layout shown in Figure 5 *without* a previous transcode.
Black dashed: Using similar network but *with* the features from a previous low-res, low-bitrate transcode of the same content.

Figure 9: Cumulative distribution of bitrate errors (%) across the test set.

80% of the test transcodes at or below the 20% bitrate-error target.

This work suggests several additional avenues of research. One is to use the additional information that is available from the classification layer to provide a confidence measure for the estimated CRF. In our initial (simple) tests, when we limit our estimates to the 5% strongest (that is, the ones with the highest peak soft-max output activation level), we improve our accuracy to 90% of the tests at or below 20% bitrate error. Similarly, the 25% most confident estimates have 80% of the tests at or below 20% bitrate error.

Another interesting area to explore is the correlations amongst the bitrate-model parameter values, the errors that we see in their estimation, and the successful inference of CRF. It could be that segmenting the CRF estimation problem using the estimated bitrate-model parameter values might lead to better performance or, at minimum, provide us with better confidence measures for the estimated CRF.

References

[1] Duchi, J.C., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research 12, 21212159 (2011)

[2] Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: International Conference on Machine Learning (2015)

[3] Krogh, A., Hertz, J.A.: A simple weight decay can improve generalization. In: Advances in Neural Information Processing Systems (1992)

[4] Lawson, C., Hanson, R.: Solving Least Squares Problems. SIAM (1987)

[5] Lin, Y.C., Denman, H., Kokaram, A.: Multipass encoding for reducing pulsing artifacts in cloud-based video transcoding. In: International Conference on Image Processing (2015)

[6] Ma, Z., Xu, M., Ou, Y.F., Wang, Y.: Modeling of rate and perceptual quality of compressed video as functions of frame rate and quantization stepsize and its applications. IEEE Transactions on Circuits and Systems for Video Technology 22(5), 671–682 (May 2012)

[7] Merritt, L.: A qualitative overview of x264's ratecontrol methods. http://akuvian.org/src/x264/ratecontrol.txt (2010)

[8] Scipy community: Non-negative least squares linear solver (nnls). https://github.com/scipy/scipy/blob/v0.16.0/scipy/optimize/nnls.py#L9 (2008)

[9] Sodagar, I.: The mpeg-dash standard for multimedia streaming over the internet. IEEE Transactions on Multimedia 18(4), 62–67 (April 2011)

[10] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Computer Vision and Pattern Recognition 2015 (2015), http://arxiv.org/abs/1409.4842

[11] YouTube Press: Youtube statistics. https://www.youtube.com/yt/press/statistics.html (2015)

Author Biography

Michele Covell earned her PhD from MIT. After working at SRI International, Interval Research, YesVideo, and HP Labs, Michele joined Google, in the research group, in 2005. She focused for several years on large-scale audio and video fingerprinting, identification, and retrieval, both on basic research and on creating YouTube's Video Content Id system, based on her research. More recently, Michele has been working in image, audio, and video analysis and compression.

Martin Arjovsky is a student at the University of Buenos Aires, expecting to graduate in 2016. His research is focused on Deep Learning and Machine Learning. He has worked at Google/YouTube on applications to video compression, and at Microsoft on modelling distributed systems. He is currently a visitor at the MILA Deep Learning team in the Université de Montréal.

Yao-Chung Lin received his Ph.D. degree from the Department of Electrical Engineering at Stanford University in 2010. He now works in Video Infrastructure at Google/Youtube in Mountain View, CA. His research interests include distributed source coding applications, multimedia systems, and video processing and compression.

Anil Kokaram received the PhD in Signal Processing from Cambridge University (1993). He then worked as a research fellow at the Signal Processing group at the Cambridge University Engineering Department until 1998, when he took up a lectureship at Trinity College Dublin. His work is in the broad area of Video Processing. In 2007, he was awarded a Technical Oscar for his work in motion estimation for the movies. In 2011, his startup (GreenParrotPictures) was acquired by Google. He is now a Technical Lead at YouTube's video infrastructure division and also holds a Professorship at Trinity College Dublin.