

MACHINE LEARNING TECHNIQUES FOR COMPONENT-BASED IMAGE PREPROCESSING

CROSS-REFERENCE TO RELATED APPLICATIONS

5 [0001] This application claims benefit of United States patent application serial number 17/551,086 filed December 14, 2021, which is hereby incorporated herein by reference.

BACKGROUND

Field of the Various Embodiments

10 [0002] The various embodiments relate generally to computer science and to video and video encoding technologies and, more specifically, to machine learning techniques for component-based image preprocessing.

Description of the Related Art

15 [0003] Efficiently and accurately encoding video data is an important aspect of delivering high-quality videos in real-time. In a typical system, an encoded video is transmitted to the playback device. As the playback device receives the corresponding encoded video data, a video decoder incrementally generates a reconstructed video that is played back via the playback device. To reduce the bitrate and, accordingly, the size of an encoded video, a video encoder usually implements some type of lossy encoding algorithm to eliminate selected information from a source video. In general, eliminating information during video encoding can lead to visual quality impairments or “artifacts” that can reduce the visual quality of a reconstructed video derived from an encoded video.

25 [0004] To reduce visual quality degradation attributable to video encoding, a sequence of one or more preprocessing algorithms can be deployed prior to a lossy encoding algorithm to create a conventional video encoding pipeline. Each of the preprocessing algorithms can be instantiated externally to or embedded within a video encoder that includes the lossy encoding algorithm. In a typical video encoding pipeline, each preprocessing algorithm independently implements a different type of image preprocessing to preemptively eliminate information from images or “frames” that make up a source video in an effort to increase the effectiveness of the lossy encoding algorithm and reduce the overall amount of visual quality degradation resulting from the related encoding operations.

[0005] Generally speaking, each type of image preprocessing implements a different strategy for identifying and eliminating less important or distorted information from one or more components of an image. Some common types of image preprocessing operations include, without limitation, chroma subsampling, downscaling, and spatial
5 denoising. Chroma subsampling refers to reducing the resolution of data corresponding to color or “chroma” components, to which the human visual system is less sensitive, in favor of the resolution of data corresponding to a brightness or “luma” component, to which the human visual system is more sensitive. Downscaling refers to equitably reducing the resolution of data corresponding to all image
10 components. Spatial denoising refers to suppressing noise that contaminates images relative to the original objects (e.g., that are filmed, photographed, etc., to generate the images).

[0006] One drawback of conventional video encoding pipelines is that the preprocessing algorithms implemented within those pipelines typically are not
15 optimized to reduce the amount of visual quality degradation arising from the aggregated preprocessing operations and/or arising from subsequent encoding operations. In that regard, a given preprocessing algorithm is usually designed and optimized independently of the video encoding pipeline in which the algorithm is eventually implemented and is not altered when deployed with other preprocessing
20 algorithms within the actual pipeline. The current piecemeal approach to designing and optimizing preprocessing algorithms individually can limit the effectiveness of the aggregated preprocessing operations being performed across multiple preprocessing algorithms within a video encoding pipeline, which can unnecessarily increase the number of artifacts in a decoded image and/or increase the severity of those artifacts.
25 For example, when a lossy encoding algorithm encodes a preprocessed video to generate an encoded video having a given bitrate, the overall visual quality of an associated reconstructed video could be sub-optimally low. Conversely, the total number of bits that a lossy encoding algorithm uses to encode a preprocessed video to achieve a given target visual quality level for an associated reconstructed video
30 could be sub-optimally high.

[0007] As the foregoing illustrates, what is needed in the art are more effective techniques for preprocessing videos within video encoding pipelines.

SUMMARY

[0008] One embodiment sets forth a computer-implemented method for training a machine learning model to preprocess images. The method includes computing a chroma sampling factor based on a downscaling factor and a chroma subsampling ratio; executing a first machine learning model that is associated with the chroma sampling factor on first data that corresponds to both a first image and a first chroma component to generate first preprocessed data corresponding to the first chroma component; and updating at least one parameter of the first machine learning model based on the first preprocessed data to generate a first trained machine learning model that is associated with the first chroma component.

[0009] At least one technical advantage of the disclosed techniques relative to the prior art is that the disclosed techniques can be implemented in a video encoding pipeline to mitigate the visual quality reductions typically experienced with conventional video encoding pipelines. In that regard, a trained machine learning model implemented in a video encoding pipeline learns to reduce reconstruction errors when preprocessing data corresponding to at least one component in a luma-chroma color space. The reconstruction errors correlate to decreases in the visual quality of reconstructed images attributable to preprocessing source images within a video encoding pipeline. Thus, with the disclosed techniques, the visual quality of a reconstructed video can be increased for a given bitrate relative to what can be achieved using prior art techniques. Conversely, with the disclosed techniques, the number of bits used when encoding a source video to achieve a given target visual quality can be reduced relative to what can be achieved using prior art techniques. These technical advantages provide one or more technological improvements over prior art approaches.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] So that the manner in which the above-recited features of the various embodiments can be understood in detail, a more particular description of the inventive concepts, briefly summarized above, may be had by reference to various embodiments, some of which are illustrated in the appended drawings. It is to be noted, however, that the appended drawings illustrate only typical embodiments of the inventive concepts and are therefore not to be considered limiting of scope in any way, and that there are other equally effective embodiments.

[0011] Figure 1 is a conceptual illustration of a system configured to implement one or more aspects of the various embodiments;

[0012] Figure 2 is an exemplar illustration of the training application of Figure 1, according to various embodiments;

- 5 [0013] Figures 3A-3B set forth a flow diagram of method steps for training one or more machine learning models to preprocess images, according to various embodiments;

[0014] Figure 4 is an exemplar illustration of the image preprocessing application of Figure 1, according to various embodiments;

- 10 [0015] Figure 5 is a flow diagram of method steps for preprocessing a source video, according to various embodiments; and

[0016] Figure 6 is a flow diagram of method steps for preprocessing source images, according to various embodiments.

DETAILED DESCRIPTION

- 15 [0017] In the following description, numerous specific details are set forth to provide a more thorough understanding of the various embodiments. However, it will be apparent to one skilled in the art that the inventive concepts may be practiced without one or more of these specific details.

- [0018] A typical video streaming service provides access to a library of media titles
20 that can be viewed via different playback devices. To enable real-time delivery of the media titles to playback devices via limited-bandwidth networks, video encoders are configured to execute lossy encoding algorithms that eliminate selected information from corresponding source videos to generate corresponding encoded videos having reduced bitrates. In the context of a media title, a "bitrate" is a total number of bits
25 used to represent the media title divided by the total playback time. To playback a given media title via a given playback device, a corresponding encoded video is streamed to the playback device. As the playback device receives the encoded video data, a video decoder generates a reconstructed video that is played back via the playback device.

[0019] For a typical video encoder, as the number of bits used to encode a media title decreases, the visual quality of the reconstructed video also decreases. In an attempt to mitigate visual quality reductions attributable to video encoding, oftentimes one or more types of image preprocessing algorithms are deployed before or at the
5 beginning of a video encoder to generate a conventional video encoding pipeline. Each type of image preprocessing algorithm systematically removes information from the images included in a source video in an effort to increase the effectiveness of the video encoder based on a different strategy. For example, a downscaling algorithm reduces the resolution of data corresponding to all components, a chroma
10 subsampling algorithm reduces the resolution of data corresponding to chroma components, and a spatial denoising algorithm reduces noise.

[0020] One drawback of conventional video encoding pipelines is that each preprocessing algorithm is typically designed and optimized independently of the other preprocessing algorithms. Therefore, a preprocessed video generated by the
15 final preprocessing algorithm in the video encoding pipeline is unlikely to be properly optimized to increase the overall effectiveness of the aggregated preprocessing operations and/or subsequent encoding operations performed within the conventional video encoding pipeline. As a result, for a given bitrate, the visual quality of a reconstructed video could be needlessly low. Conversely, the number of bits used
20 during encoding to achieve a given visual quality level for a reconstructed video could be inordinately high.

[0021] With the disclosed techniques, however, a training application uses machine learning techniques to generate one or more trained preprocessors that reduce reconstruction errors when preprocessing images represented in a luma-chroma color
25 space. The reconstruction errors correlate to an end-to-end loss of visual quality between a given source image and a reconstructed image derived from a preprocessed image. During training, the reconstruction errors can optionally be adjusted to account for any intentional modifications (e.g., removal of noise). designed to increase the effectiveness of lossy encoding algorithms. A luma-chroma color
30 represents colors numerically via a luma component that represents brightness, and two chroma components (denoted herein as chroma₁ and chroma₂) that represent color. The training application can jointly or independently train one or more

preprocessors to preprocess data corresponding to mutually exclusive sets of components.

- [0022]** In some embodiments, the training application generates a trained luma preprocessor, a trained chroma₁ preprocessor, and a trained chroma₂ preprocessor based on a downscaling factor and a chroma subsampling ratio. Notably, downscaling is akin to downsampling chroma₁ data, chroma₂ data, and luma data. By contrast, chroma subsampling is akin to downsampling chroma₁ data and chroma₂ data while leaving luma data unchanged. To coalesce downscaling and chroma subsampling, the training application sets an initial version of a luma preprocessor equal to an untrained convolutional neural network (CNN) that downsamples by the downscaling factor. By contrast, the training application sets initial versions of a chroma₁ preprocessor and a chroma₂ preprocessor equal to two instances of a untrained CNN that downsamples by a multiple of the downscaling factor, where the multiplier reflects the chroma subsampling ratio.
- [0023]** The training application then trains the luma preprocessor, the chroma₁ preprocessor, and the chroma₂ preprocessor based on discrete portions or “image patches” of any number of training images included in a training database. In some embodiments, the training images are frames of videos that reflect the primary use of a target video encoding pipeline (e.g., encoding animated high-resolution movies or encoding low-resolution video clips). If the training images are not in the luma-chroma color space, then the training application converts the training images to the luma-chroma color space when determining the image patches. The training application can perform any number and/or type of machine learning training operations on the luma preprocessor, the chroma₁ preprocessor, and the chroma₂ preprocessor to reduce model-specific reconstruction errors for luma data, chroma₁ data, and chroma₂ data, respectively, or joint reconstruction errors for image patches. In some embodiments, and as described in greater detail below in conjunction with Figure 2, the training application computes any number and/or types of reconstruction errors based on any number and/or types of loss functions, cost functions, objective functions, or any combination thereof. In the same or other embodiments, the training application computes any number and/or types of reconstruction errors based on any number and/or **types of differentiable** loss functions, cost functions, objective functions, or any combination thereof that incorporate directionality and/or regional

information to reduce any number and/or types of distortions, such as color bleeding, edge distortions, etc.

[0024] In some embodiments, to compute a joint reconstruction error for a given image patch, the training application inputs the corresponding luma data, chroma₁ data, and chroma₂ data into the luma preprocessor, the chroma₁ preprocessor, and the chroma₂ preprocessor, respectively. In response, the luma preprocessor, the chroma₁ preprocessor, and the chroma₂ preprocessor generate preprocessed luma data, preprocessed chroma₁ data, and preprocessed chroma₂ data, respectively. Subsequently, the training application configures and applies different instances of a configurable upsampling algorithm to the preprocessed luma data, the preprocessed chroma₁ data, and the preprocessed chroma₂ data. The resulting reconstructed luma data, reconstructed chroma₁ data, and reconstructed chroma₂ data have the same resolution as the luma data, the chroma₁ data, and the chroma₂ data, respectively. The training application aggregates the reconstructed luma data, the reconstructed chroma₁ data, and the reconstructed chroma₂ data to generate a reconstructed patch. In some embodiments, the training application sets a joint reconstruction error for the image patch equal to the mean squared error of the reconstructed patch relative to the image patch.

[0025] After computing an average joint reconstruction error based on the reconstruction errors for a batch of image patches, the training application individually updates the parameters of each of the preprocessors based on the average joint reconstruction loss. After the training is complete, the training application stores the final versions of the chroma₁ preprocessor, the chroma₂ preprocessor, and the luma preprocessor in a trained machine learning (ML) model database as a trained chroma₁ preprocessor, a trained chroma₂ preprocessor, and a trained luma preprocessor, respectively.

[0026] In some embodiments, an image preprocessing application can use one or more of any number and/or types of trained preprocessors and optionally any number and/or types of non-ML preprocessors to perform component-based preprocessing of source images. As referred to herein, in “component-based preprocessing,” preprocessed luma data is generated separately from preprocessed chroma data, and preprocessed chroma₁ data and preprocessed chroma₂ data can be generated jointly or separately. In operation, if a source image is not in the luma-chroma color

space, the image preprocessing application converts the source image into the luma-chroma color space. In some embodiments, the image preprocessing application inputs the luma data, the chroma₁ data, and the chroma₂ data corresponding to the source image into the trained luma preprocessor, the trained chroma₁ preprocessor, and the trained chroma₂ preprocessor, respectively. In response, the trained luma preprocessor, the trained chroma₁ preprocessor, and the trained chroma₂ preprocessor generate preprocessed luma data, preprocessed chroma₁ data, and preprocessed chroma₂ data, respectively. The image preprocessing application aggregates the preprocessed luma data, the preprocessed chroma₁ data, and the preprocessed chroma₂ data to generate a preprocessed image. The image preprocessing application then stores the preprocessed image in any memory and/or transmits the preprocessed image to any number of software applications (e.g., an image encoder or a video encoder).

[0027] At least one technical advantage of the disclosed techniques relative to the prior art is that the disclosed techniques can be implemented in a video encoding pipeline to mitigate the visual quality reductions typically experienced with conventional video encoding pipelines. In that regard, the training application trains one or more ML models to reduce reconstruction errors when preprocessing at least one of chroma₁ data, chroma₂ data, or luma data. The reconstruction errors correlate to decreases in the visual quality of reconstructed images included in reconstructed videos attributable to preprocessing source images within a video encoding pipeline. Thus, relative to prior art techniques, these trained ML models(s) can be used by the image preprocessing application within a video encoding pipeline to increase the effectiveness of the video encoding pipeline. Increasing the effectiveness of a video encoding pipeline can lead to an increase in the visual quality level of a reconstructed video derived from an encoded video having a given bitrate. Conversely, increasing the effectiveness of a video encoding pipeline can decrease the minimum bitrate required to achieve a target visual quality level. Another technical advantage of the disclosed techniques is that the training application can generate different trained preprocessor(s) that are fine-tuned for different types of images (e.g., images in an animated film). Accordingly, when preprocessing images of a given type, the image preprocessing application can use the trained preprocessor(s) that are fine-tuned for that image type, which increases the effectiveness of the related preprocessing

operations. These technical advantages provide one or more technological improvements over prior art approaches.

System Overview

[0028] Figure 1 is a conceptual illustration of a system 100 configured to implement one or more aspects of the various embodiments. For explanatory purposes, multiple instances or versions of like objects are denoted with reference numbers identifying the object and parenthetical alphanumeric character(s) identifying the instance or version where needed. As shown, in some embodiments, the system 100 includes, without limitation, a compute instance 110(1), a compute instance 110(2), an untrained/pre-trained model database 102, a training image database 104, and a trained model database 140.

[0029] In some other embodiments, the system 100 can omit the compute instance 110(1), the compute instance 110(2), the untrained/pre-trained model database 102, the training image database 104, the trained model database 140, or any combination thereof. In the same or other embodiments, the system 100 can include, without limitation, one or more other compute instances, any number and/or types of other databases, or any combination thereof. The components of the system 100 can be distributed across any number of shared geographic locations and/or any number of different geographic locations and/or implemented in one or more cloud computing environments (i.e., encapsulated shared resources, software, data, etc.) in any combination.

[0030] As shown, the compute instance 110(1) includes, without limitation, a processor 112(1) and a memory 116(1), and the compute instance 110(2) includes, without limitation, a processor 112(2) and a memory 116(2). The compute instance 110(1) and the compute instance 110(2) are also referred to herein individually as “the compute instance 110” and collectively as “the compute instances 110.” The processor 112(1) and the processor 112(2) are also referred to herein individually as “the processor 112” and collectively as “the processors 112.” The memory 116(1) and the memory 116(2) are also referred to herein individually as “the memory 116” and collectively as “the memories 116.” Each of the compute instances 110 can be implemented in a cloud computing environment, implemented as part of any other distributed computing environment, or implemented in a stand-alone fashion.

[0031] The processor 112 can be any instruction execution system, apparatus, or device capable of executing instructions. For example, the processor 112 could comprise a central processing unit, a graphics processing unit, a controller, a microcontroller, a state machine, or any combination thereof. The memory 116 of the compute instance 110 stores content, such as software applications and data, for use by the processor 112 of the compute instance 110. The memory 116 can be one or more of a readily available memory, such as random-access memory, read-only memory, floppy disk, hard disk, or any other form of digital storage, local or remote.

[0032] In some other embodiments, each of any number of compute instances 110 can include any number of processors 112 and any number of memories 116 in any combination. In particular, any number of compute instances 110 (including one) can provide a multiprocessing environment in any technically feasible fashion.

[0033] In some embodiments, a storage (not shown) may supplement or replace the memory 116 of the compute instance 110. The storage may include any number and type of external memories that are accessible to the processor 112 of the compute instance 110. For example, and without limitation, the storage can include a Secure Digital Card, an external Flash memory, a portable compact disc read-only memory, an optical storage device, a magnetic storage device, or any suitable combination of the foregoing

[0034] In general, each compute instance 110 is configured to implement one or more software applications. For explanatory purposes only, each software application is described as residing in the memory 116 of a single compute instance 110 and executing on the processor 112 of the single compute instances 110. However, in some embodiments, the functionality of each software application can be distributed across any number of other software applications that reside in the memories 116 of any number of compute instances 110 and execute on the processors 112 of any number of compute instances 110 in any combination. Further, the functionality of any number of software applications can be consolidated into a single software application.

[0035] In some embodiments, a video preprocessing application 180 and optionally a video encoder 196 reside in the memory 116(2) of the compute instance 110(2) and execute on the processor 112(2) of the compute instance 110(2). In some

embodiments, the video preprocessing application 180 and the video encoder 196 are collectively referred to as a “video encoding pipeline.”

[0036] As shown, in some embodiments, the video encoding pipeline generates an encoded video 198 based on a source video 170. The source video 170 includes, without limitation, any amount and/or types of video content. Some examples of video content are any portion (including all) of feature-length films, episodes of television programs, and music videos, to name a few. The video content of source video 170 is organized as a sequence of source images 172. Each of the source images 172 is an image or “frame” that includes, without limitation any amount and/or type of image content.

[0037] Although not shown, in some embodiments, the encoded video 198 is transmitted to a playback device. An endpoint application executing on the playback device decodes the encoded video 198 to generate a decoded video. The endpoint application performs any number (including none) and/or types of sampling operations (e.g., upsampling operations or downsampling operations) on the encoded video 198 to generate a reconstructed video having the same resolution as a display device. The endpoint application plays back the reconstructed video.

[0038] As described previously herein, a typical video encoding pipeline includes, without limitation, a sequence of one or more preprocessing algorithms followed by a lossy encoding algorithm that is implemented in the video encoder. Each of the preprocessing algorithms in the video encoding pipeline independently implements a different type of image preprocessing to preemptively eliminate information from the images in a video in an effort to increase the effectiveness of the lossy encoding algorithm. Some examples of types of image preprocessing include, without limitation, downscaling, chroma subsampling, and denoising. One drawback of such a video encoding pipeline is that the preprocessing algorithms are neither designed nor optimized to operate as part of the video encoding pipeline. As a result, a reduction in the visual quality of a reconstructed video that is attributable to the video encoding pipeline can be unnecessarily high.

Using Machine Learning Techniques to Preprocess Images

[0039] To address the above problems, the system 100 includes, without limitation, a training application 130 that uses machine learning techniques to generate trained

preprocessors that reduce reconstruction errors when preprocessing components of images. In some embodiments, a reconstruction error is a value for any type of metric that correlates to an end-to-end loss of visual quality between a given source image and a reconstructed version of the source image. In the same or other

5 embodiments, a reconstruction error is determined based on any number and/or types of metrics that take into account one or more aspects of visual quality. In some embodiments, one goal is to reduce color bleeding when preprocessing components of images, and a reconstruction error is determined, at least in part, based on a metric that measures color bleeding resulting from color subsampling. In the same or other

10 embodiments, one goal is to reduce edge distortions when preprocessing components of images, and a reconstruction error is determined, at least in part, based on a metric that measures edge distortions.

[0040] Because the human visual system is less sensitive to color than to brightness, the training application 130 enables component-based image preprocessing. More

15 specifically, the training application 130 can train different machine learning (ML) models to preprocess data corresponding to different subsets of the components of a luma-chroma color space. Furthermore, each trained ML model or “trained preprocessor” generated by the training application 130 can consolidate any amount and/or types of preprocessing on data corresponding to the associated component

20 subset. And the training images used to train a given ML model can be selected such that the resulting trained preprocessor is fine-tuned for images and/or videos having specific characteristics.

[0041] The luma-chroma color space can be any color space that represents colors numerically via a luma component that represents brightness, and two chroma

25 components that represent color. For explanatory purposes, “luma component,” “chroma₁ component,” and “chroma₂ component” are used herein to denote the components of the luma-chroma color space. Similarly, the terms “luma data,” “chroma₁ data,” and “chroma₂ data” are used herein to refer to data corresponding to the luma component, the chroma₁ component, and the chroma₂ component,

30 respectively. Chroma₁ data and chroma₂ data are also referred to herein individually and collectively as “chroma data.”

[0042] An image represented in a luma-chroma color space includes, without limitation, at least one of luma data, chroma₁ data, and chroma₂ data. An image that

includes only luma data, only chroma₁ data, or only chroma₂ data is a monochrome image. Luma data, chroma₁ data, and chroma₂ data can correspond to any portion (including all) of any type of image (e.g., one of the source images 172) and/or any type of video.

- 5 **[0043]** In some embodiments, each image and each video can be associated with any technically feasible combination of any number and/or types of color-related characteristics. Some examples of color-related characteristics include, without limitation, dynamic range, color gamut, and bit-depth. For instance, in some
- 10 embodiments, the source video 170 is a High Dynamic Range (HDR)/Wide Color Gamut (WCG) video having a bit-depth of 16 bits per component, and each of the source images 172 is an HDR/WCG image having a bit-depth of 16 bits per component. In some other embodiments, the source video 170 is a Standard
- 15 Dynamic Range (SDR) video having a bit-depth of 8 bits per component, and each of the source images 172 is an SDR image having a bit-depth of 8 bits per component.
- 20 **[0044]** The luma-chroma color space can provide a non-constant luminance representation or a constant luminance representation. In a non-constant luminance representation, modifying the chroma data of an image is relatively likely to perceptibly impact the brightness of the image. An example of a luma-chroma color space that provides a non-constant luminance representation is the Y'CbCr color
- 25 space. If the luma-chroma color space is the Y'CbCr color space, then the luma component is a gamma-compressed brightness component denoted "Y'," the chroma₁ component is the blue-difference chroma component denoted "Cb," and the chroma₂ component is the red-difference chroma component denoted "Cr."
- 30 **[0045]** By contrast, in a constant luminance representation modifying the chroma data of an image is relatively unlikely to perceptibly impact the brightness of the image. Accordingly, some embodiments implement a luma-chroma color space providing a constant luminance representation to increase the effectiveness of chroma-specific preprocessing, such as chroma subsampling. An example of a luma-chroma color space that provides a constant luminance representation is the ICtCp color space. If
- 35 the luma-chroma color space is the ICtCp color space, then the luma component is the intensity component denoted "I," the chroma₁ component is the blue-yellow chroma component denoted "Ct," and the chroma₂ component is the red-green component denoted "Cp."

[0046] As shown, in some embodiments, the training application 130 resides in the memory 116(1) of the compute instance 110(1) and executes on the processor 112(1) of the compute instance 110(1). In the same or other embodiments, the training application 130 generates a trained luma preprocessor 142, a trained chroma₁ preprocessor 144, and a trained chroma₂ preprocessor 146 based on an untrained/pre-trained model database 102, a training image database 104, and a preprocessing specification 120(0). The preprocessing specification 120(0) describes, without limitation, any amount and/or types of preprocessing that is to be implemented in one or more associated trained preprocessors.

[0047] As shown, in some embodiments, the preprocessing specification 120(0) includes, without limitation, a downscaling factor 122(0), a chroma subsampling ratio 124(0), optionally a denoising type 126(0), and optionally any amount and/or type of other data associated with any preprocessing. The downscaling factor 122(0) specifies a factor by which the resolutions of luma data, chroma₁ data, and chroma₂ data are to be decreased. In some embodiments, the chroma subsampling ratio is a triplet that is denoted "4:h:v," where h and v specify the relative number of values corresponding to each chroma component in rows of a conceptual 4x2 pixel block. In some embodiments, the denoising type 126(0) specifies whether spatial denoising is to be applied to all data, only luma data, only chroma data, or no data.

[0048] The training image database 104 includes, without limitation, any number and/or types of training images. In some embodiments, the training images can be represented in any type of color space, and the training application 130 performs any number and or types of operations on the training images to determine any amount and/or types of training data in the luma-chroma color space. For instance, in some embodiments, the training images are represented in a linear RGB color space or a non-linear RGB color space and the training application 130 determines luma data, chroma₁ data, and chroma₂ data corresponding to multiple "patches" of each training image.

[0049] In some embodiments, the training images in the training image database 104 share one or more characteristics. In the same or other embodiments, the training application 130 selects any number of training images from the training image database 104 and/or any number of other training databases based on one or more target characteristics and disregards the remaining training images. In some

embodiments, the training application 130 trains one or more untrained or pre-trained ML models based on a selected subset of available training images to generate corresponding trained preprocessors that are fine-tuned for images and/or videos similar to the selected subset of training images.

- 5 **[0050]** The untrained/pre-trained model database 102 includes, without limitation, any number and/or types of ML models. For instance, in some embodiments, the untrained/pre-trained model database 102 includes any number of convolutional neural networks (CNNs). In the same or other embodiments, any number of CNNs can be configurable, and the architecture of any number of CNNs can vary based on
10 associated types of preprocessing. In some embodiments, the training application 130 is configured to generate multiple trained preprocessors corresponding to different downsampling factors. In the same or other embodiments, multiple trained luma preprocessors corresponding to different downsampling factors are implemented in an adaptive multi-network architecture in an image preprocessing
15 application 150.

- [0051]** Each of the trained luma preprocessor 142, the trained chroma₁ preprocessor 144, and the trained chroma₂ preprocessor 146 can implement any number and/or types of preprocessing. For instance, in some embodiments, the trained luma preprocessor 142, the trained chroma₁ preprocessor 144, and the trained chroma₂
20 preprocessor 146 each implement both downsampling and spatial denoising. In some other embodiments, the trained luma preprocessor 142 implements both downscaling and spatial denoising, and each of the trained chroma₁ preprocessor 144 and the trained chroma₂ preprocessor 146 implement both downscaling and chroma subsampling.

- 25 **[0052]** Although not shown, in some embodiments, the training application 130 generates a joint chroma preprocessor that jointly preprocesses chroma₁ data and chroma₂ data. In some other embodiments, the training application 130 generates a single chroma preprocessor that can be used to independently process chroma₁ data and chroma₂ data.

- 30 **[0053]** In some embodiments, the trained luma preprocessor 142 generates preprocessed luma data for a preprocessed image based on luma data, optionally chroma₁ data, and optionally chroma₂ data for an image. In the same or other

embodiments, the trained chroma₁ preprocessor 144 generates preprocessed chroma₁ data for a preprocessed image based on chroma₁ data, optionally luma data, and optionally chroma₂ data for an image. In some embodiments, the trained chroma₂ preprocessor 146 generates preprocessed chroma₂ data for a preprocessed
5 image based on chroma₂ data, optionally luma data, and optionally chroma₁ data for an image.

[0054] The training application 130 can implement any amount and/or type of machine learning techniques to train any type of untrained or pre-trained machine learning models based on any criteria in any technically feasible fashion. As described in
10 greater detail below in conjunction with Figure 2, in some embodiments, the training application 130 implements an iterative training process based on any number and/or types of loss functions, cost functions, objective functions, or any combination thereof.

[0055] In some embodiments, after generating a trained preprocessor, the training application 130 stores the trained preprocessor in a trained model database 140 and
15 optionally any number and/or types of associated characteristics in any technically feasible fashion. In the same or other embodiments, the training application 130 stores the trained preprocessor in any number and/or types of other memories and/or transmits the trained preprocessor to any number and/or types of software application.

[0056] As shown, in some embodiments, the training application 130 stores the trained luma preprocessor 142, the trained chroma₁ preprocessor 144, and the trained chroma₂ preprocessor 146 in a trained model database 140. In the same or other
20 embodiments, the training application optionally associated the trained luma preprocessor 142, the trained chroma₁ preprocessor 144, and the trained chroma₂
25 preprocessor 146 with the luma component, the chroma₁ component, and the chroma₂ component, respectively.

[0057] In some embodiments, an image preprocessing application 150 and optionally an image encoder 192 reside in the memory 116(2) of the compute instance 110(2) and execute on the processor 112(2) of the compute instance 110(2). In the same or
30 other embodiments, the image preprocessing application 150 uses any number and/or types of trained preprocessors and/or non-ML preprocessors 106 to perform

component-based preprocessing on any number of source images as per a preprocessing specification 120(1) to generate corresponding preprocessed images.

[0058] In some embodiments, the image preprocessor performs component-based image preprocessing based on one or more mutually exclusive component set(s) that collectively specify the luma, chroma₁, and chroma₂ components of the luma-chroma color space. The image preprocessor can determine the component set(s) in any technically feasible fashion. In some embodiments, the image preprocessor determines the component set(s) based on the preprocessing specification 120(1). In some other embodiments, the image preprocessing application 150 implements predefined component set(s).

[0059] For each component set, the image preprocessing application 150 determines an associated spatial preprocessor (not shown) that is capable of preprocessing data corresponding to the component set per the preprocessing specification 120(1). In some embodiments, each spatial preprocessor includes, without limitation, a trained preprocessor, one or more non-ML preprocessors 106, or any combination thereof. The image preprocessing application 150 can determine the spatial preprocessor(s) in any technically feasible fashion.

[0060] In some embodiments, the image preprocessing application 150 implements predefined spatial preprocessor(s). In some other embodiments, the image preprocessing application 150 selects and aggregates one or more preprocessor(s) to generate each spatial preprocessor. In the same or other embodiments, the image preprocessing application 150 selects the preprocessor(s) from the trained model database 140 and optionally any number of non-ML preprocessors 106 based on the preprocessing specification 120(1). In some embodiments, the image preprocessing application 150 uses the same spatial preprocessor(s) to preprocess any number of source images to generate corresponding encoded source images.

[0061] For explanatory purposes, the operations the image preprocessing application 150 performs on each of any number of source images are described and depicted (via dashed arrows) herein in the context of preprocessing a source image 108. In some embodiments, the source image 108 can be represented in any type of color space, and the image preprocessing application 150 performs any number and or

types of operations on the source image 108 to determine luma data, chroma₁ data, and chroma₂ data corresponding to the source image 108.

[0062] The image preprocessing application 150 causes each spatial preprocessor to generate preprocessed data corresponding to the associated component set based on at least one of the luma data, the chroma₁ data, or the chroma₂ data. In some embodiments, the training application 130 aggregates the preprocessed luma data, the preprocessed chroma₁ data, and the preprocessed chroma₂ data to generate a preprocessed image 158 that is a preprocessed version of the source image 108. In some embodiments, the image preprocessing application 150 stores preprocessed image 158 in any number and/or types of memories. In the same or other embodiments, the image preprocessing application 150 transmits the preprocessed image 158 to any number and/or types of software applications.

[0063] In some embodiment, the image preprocessing application 150 transmits the preprocessed image 158 to the image encoder 192. The image encoder 192 performs any number and/or types of operations on the preprocessed image 158 to generate an encoded image 194. In some embodiments, the image encoder 192 stores the encoded image 194 in any number and/or types of memories. In the same or other embodiments, the image encoder 192 transmits the encoded image 194 to any number and/or types of software applications. In some embodiments, the image preprocessing application 150 and the image encoder 192 are collectively referred to as an “image encoding pipeline.”

[0064] In some embodiments, the video preprocessing application 180 uses the image preprocessing application 150 to perform component-based encoding on a source video 170 based on the preprocessing specification 120(1). As shown, in some embodiments, the video preprocessing application 180 includes, without limitation, a spatial preprocessing engine 182 and optionally a temporal preprocessing engine 184.

[0065] In operation, the spatial preprocessing engine 182 splits the source video 170 into source images 172. In some embodiments, the source video 170 can be represented in any type of color space. and therefore the source images 172 are not represented in the luma-chroma color space. In the same or other embodiments, and as described previously herein, the image preprocessing application 150 can

preprocess source images in any type of color space. The spatial preprocessing engine 182 transmits the source images 172 and the preprocessing specification 120(1) to the image preprocessing application 150. In response, the image preprocessing application 150 preprocesses the source images 172 to generate
5 preprocessed images 178.

[0066] As shown, in some embodiments, the video preprocessing application 180 does not implement temporally preprocessing, and the video preprocessing application 180 aggregates the preprocessed images 178 to generate the preprocessed video 188. In some other embodiments, the video preprocessing
10 application 180 aggregates the preprocessed images 178 to generate a spatially preprocessed video (not shown). The temporal preprocessing engine 184 performs any number and/or types of temporal preprocessing (e.g., temporal denoising, interlace removal, etc.) on the spatially preprocessed video to generate the preprocessed video 188.

[0067] In some embodiments, the video preprocessing application 180 stores the preprocessed video 188 in any number and/or types of memories. In the same or other embodiments, the video preprocessing application 180 transmits the preprocessed video 188 to any number and/or types of software applications. As shown, in some embodiments, the video preprocessing application 180 transmits the
20 preprocessed video 188 to the video encoder 196, and the video encoder 196 encodes the preprocessed video 188 to generate the encoded video 198. In some embodiments, the video encoder 196 stores the encoded video 198 and/or transmits the encoded video 198 to any number of software applications.

[0068] Note that the techniques described herein are illustrative rather than restrictive
25 and may be altered without departing from the broader spirit and scope of the invention. Many modifications and variations on the functionality provided by the training application 130, the image preprocessing application 150, the video preprocessing application 180, the spatial preprocessing engine 182, the temporal preprocessing engine 184, the image encoder 192, the video encoder 196, the trained
30 preprocessors, the untrained ML models, and the pre-trained ML models will be apparent to those of ordinary skill in the art without departing from the scope and spirit of the described embodiments.

[0069] It will be appreciated that the system 100 shown herein is illustrative and that variations and modifications are possible. For instance, the connection topology between the various components in Figure 1 may be modified as desired. For instance, in some embodiments, the image preprocessing application 150 is
5 integrated into the video preprocessing application 180. In some other embodiments, the video preprocessing application 180 and the video encoder 196 are omitted from the system 100.

Generating Different Trained Preprocessors for Different Components

[0070] Figure 2 is an exemplar illustration of the training application 130 of Figure 1,
10 according to various embodiments. More specifically, the training application 130 depicted in Figure 2 generates the trained luma preprocessor 142, the trained chroma₁ preprocessor 144, and the trained chroma₂ preprocessor 146 based on the untrained/pre-trained model database 102, the training image database 104, and the preprocessing specification 120(0). In some other embodiments, the training
15 application 130 can generate any number and/or types of preprocessors.

[0071] For explanatory purposes only, in the context of Figures 2 and 5, the symbol “↓” denotes downsampling, “2↓” denotes 2x downsampling, and “4↓” denotes 4x downsampling. In a complementary fashion, “↑” denotes upsampling, “2↑” denotes 2x upsampling, and “4↑” denotes 4x upsampling. As shown, in some embodiments,
20 the untrained/pre-trained model database 102 includes, without limitation, a 2x downsampling CNN 212, a 4x downsampling CNN 214, a non-integer downsampling CNN 216, and a 2x downsampling/denoising CNN 218, zero or more other CNNs, and zero or more other types of ML models. In some embodiments, each of the 2x downsampling CNN 212, the 4x downsampling CNN 214, the non-integer
25 downsampling CNN 216, and the 2x downsampling/denoising CNN 218 are untrained.

[0072] In some embodiments, the architecture of the ML models included in the untrained/pre-trained model database 102 can vary. As visually depicted via thumbnails, in some embodiments, the 2x downsampling CNN 212 has fewer layers
30 than the 4x downsampling CNN 214. In some embodiments, any number of the ML models included in the untrained/pre-trained model database 102 can be configured and/or customized prior to training in any technically feasible fashion. For instance, in

some embodiments, the non-integer downsampling CNN 216 can be configured via an associated non-integer downsampling factor.

[0073] In some embodiments, each ML model included in the untrained/pre-trained model database 102 includes, without limitation, any number and/or types of parameters that the training application 130 can update during training to reduce reconstruction errors when preprocessing components of images. For instance, in some embodiments, each of the 2x downsampling CNN 212, the 4x downsampling CNN 214, the non-integer downsampling CNN 216, and the non-integer downsampling CNN 216 includes, without limitation, an upsampling/identity residual block and a downsampling residual block. As referred to herein, a “residual block” is a portion of a CNN that maps the input of the residual block to a residual and then adds the residual to a function of the input of the residual block to generate the output of the residual block. Both the upsampling/identity residual block and the downsampling residual block include, without limitation, one or more parameters that the training application can modify to mitigate a reduction in visual quality associated with a loss of data that is inherent in downsampling.

[0074] As shown, in some embodiments, the training application 130 includes, without limitation, a network generation engine 220, and an iteration engine 250. In some embodiments, the network generation engine 220 configures one or more training networks based on the untrained/pre-trained model database 102 and the preprocessing specification 120(0). In the same or other embodiments, the network generation engine 220 includes, without limitation, component set(s) 222, a luma sampling factor 224, and a chroma sampling factor 226.

[0075] The component set(s) 222 include, without limitation, one or more mutually exclusive component sets, where each component set specifies one or more of the luma component, the chroma₁ component, and the chroma₂ component. The network generation engine 220 can determine the component set(s) 222 in any technically feasible fashion. For instance, in some embodiments, the network generation engine 220 determines the component set(s) 222 based on the preprocessing specification 120(0). In some embodiments, for each component set included in the component set(s) 222, the training application 130 generates a different trained preprocessor that preprocesses data corresponding to the one or more components specified in the component set.

[0076] As shown, in some embodiments, including the embodiment depicted in Figure 2, each component set is a singleton specifying a different component of the luma-chroma color space and therefore the component set(s) 222 include, without limitation, {luma}, {chroma₁}, and {chroma₂}. In the same or other embodiments, if the component set(s) 222 are {luma}, {chroma₁}, and {chroma₂}, then the training application 130 generates the trained luma preprocessor 142, the trained chroma₁ preprocessor 144, the trained chroma₂ preprocessor 146. By contrast, if the component set(s) 222 are {chroma₁} and {chroma₂}, then the training application 130 generates the trained chroma₁ preprocessor 144 and the trained chroma₂ preprocessor 146 but does not generate the trained luma preprocessor 142. If the component set(s) 222 are the component set {chroma₁, chroma₂}, then the training application 130 generates a trained joint chroma preprocessor (not shown) that jointly preprocesses chroma₁ data and chroma₂ data.

[0077] In some embodiments, each trained preprocessor generates preprocessed data corresponding to the components specified in the associated component set based on at least source data corresponding to at least the components specified in the associated component set and optionally any number of other source data. For instance, in some embodiments, the trained chroma₂ preprocessor 146 generates preprocessed data corresponding to the chroma₁ component based on data corresponding to the chroma₁ component and optionally the chroma₂ component and/or the luma component. In some embodiments, a joint chroma preprocessor generates preprocessed data corresponding to the chroma₁ component and the chroma₂ component based on data corresponding to the chroma₁ component and the chroma₂ component, and optionally the luma component.

[0078] In some embodiments, the luma sampling factor 224 specifies consolidated downsizing that is to be applied to luma data, and the chroma sampling factor 226 specifies consolidated downsizing that is to be applied to chroma data. In the same or other embodiments, because downsizing but not chroma subsampling reduces the size of luma data, the network generation engine 220 sets the luma sampling factor 224 equal to the downscaling factor 122(0). In some embodiments, because downscaling and chroma subsampling both reduce the size of chroma data, the training application 130 sets the chroma sampling factor 226 equal to an overall downsizing of chroma data that reflects both downscaling and chroma subsampling.

[0079] For instance, as depicted in *italics*, in some embodiments, the downscaling factor 122(0) is 2 and the chroma subsampling ratio 124(0) is 4:2:0. The downscaling factor 122(0) of 2 corresponds to a downsizing of luma data by a factor of 2 and the chroma subsampling rate does not impact the size of luma data. Accordingly, the network generation engine 220 set the luma sampling factor 224 equal to the downscaling factor 122(0) of 2. The downscaling factor 122(0) of 2 corresponds to a downsizing of chroma data by a factor of 2. And as persons skilled in the art will recognize, an image having equal resolutions for all components corresponds to a chroma subsampling ratio of 4:4:4. Relative to a “default” source image having a chroma subsampling ratio of 4:4:4, the chroma subsampling ratio 124(0) of 4:2:0 corresponds to a downsizing of chroma data by a factor of two. In some embodiments, to reflect default source images, the network generation engine 220, therefore, sets the luma sampling factor 224 equal to 4.

[0080] As the above example illustrates, in some embodiments, the network generation engine 220 can implement arbitrarily different downscaling factors for luma data and chroma data via the luma sampling factor 224 and the chroma sampling factor 226. For instance, in some embodiments, the network generation engine 220 can consolidate any downscaling factor (including none) with any conversion from one chroma subsampling ratio to another chroma subsampling ratio. Some examples of chroma subsampling factors include 2:0, 4:2:2, 4:2:1, 4:1:1, 4:0:1:0, and 3:1:1.

[0081] For each of the component set(s) 222, the network generation engine 220 selects and optionally configures an initial version of a corresponding ML model from the untrained/pre-trained model database 102 based on the luma sampling factor 224, the chroma sampling factor 226, and any other relevant preprocessing-related data (*e.g.*, the denoising type 126(0)). For explanatory purposes, and as depicted in *italics*, in some embodiments, the network generation engine 220 sets a luma ML model 242 equal to the 2x downsampling CNN 212 based on the luma sampling factor 224 of 2. In the same or other embodiments, the network generation engine 220 sets each of a chroma₁ ML model 244 and a chroma₂ ML model 246 equal to a different instance of the 4x downsampling CNN 214 based on the chroma sampling factor 226 of 4.

[0082] In some embodiments, for each component set, the network generation engine 220 implements a reconstruction function based on the corresponding ML model and

appends the reconstruction function to the ML model to generate the associated training network. In some embodiments, each reconstruction function is a differentiable upsampling algorithm. The network generation engine 220 can implement any differentiable upsampling algorithm in any technically feasible fashion.

- 5 Some examples of differentiable upsampling algorithms include, without limitation, bicubic upsampling algorithms and bilinear upsampling algorithms. In some embodiments, because endpoint applications executing on playback devices often perform bicubic or bicubic-like upsampling on encoded videos, the differentiable upsampling algorithm is a differentiable bicubic or a differentiable bicubic-like
- 10 upsampling algorithm. In the same or other embodiments, for each component set, the network generation engine 220 configures the differentiable upsampling algorithm to upsample by the sampling factor associated with the corresponding ML model.

- [0083]** For instance, as depicted in italics, in some embodiments, the network generation engine 220 implements a differentiable upsampling algorithm that is
- 15 configured to upsample by a factor of 2 as a luma reconstruction function 238. The network generation engine 220 appends the luma reconstruction function 238 to the luma ML model 242 to generate a luma training network 230. The network generation engine 220 implements two instances of a differentiable upsampling algorithm that is configured to upsample by a factor of 4 as a chroma reconstruction function 248(1)
- 20 and a chroma reconstruction function 248(2). The network generation engine 220 appends the chroma reconstruction function 248(1) to the chroma₁ ML model 244 to generate a chroma₁ training network 232. The network generation engine 220 appends the chroma reconstruction function 248(2) to the chroma₂ ML model 246 to generate a chroma₂ training network 234.

- 25 **[0084]** The iteration engine 250 implements any number and/or types of machine learning techniques to train the ML models included in the ML training networks. In some embodiments, the iteration engine 250 includes, without limitation, image patches 260 and an update engine 290. In some embodiments, the iteration engine 250 performs any number and/or types of operations on any number of on training
- 30 images included in the training image database 104 to determine luma data 262, chroma₁ data 264, and chroma₂ data 266 corresponding to any number of image patches 260. Each image patch can be any portion (including all) of any type of image.

[0085] In some embodiments, the iteration engine 250 distributes the image patches 260 across any number of batches and executes batch iterations for any number of epochs. To execute a batch iteration for a given batch, the iteration engine 250 selects the image patches in the batch. For each selected image patch, the iteration engine 250 configures each training network to generate reconstructed data corresponding to the associated component set based on data corresponding to the selected image patch and at least the associated component set.

[0086] As shown, in some embodiments, the iteration engine 250 inputs the luma data 262, optionally the chroma₁ data 264, and optionally the chroma₂ data 266 corresponding to the selected image patch into the luma ML model 242. In response, the luma ML model 242 downsamples the luma data 262 by a factor of 2 to generate preprocessed luma data (not shown). The luma reconstruction function 238 upsamples the preprocessed luma data by a factor of 2 to generate reconstructed luma data 272 derived from the selected image patch.

[0087] In the same or other embodiments, the iteration engine 250 inputs the chroma₁ data 264, optionally the luma data 262, and optionally the chroma₂ data 266 corresponding to the selected image patch into the chroma₁ ML model 244. In response, the chroma₁ ML model 244 downsamples the chroma₁ data 264 by a factor of 4 to generate preprocessed chroma₁ data (not shown). The chroma reconstruction function 248(1) upsamples the preprocessed chroma₁ data by a factor of 4 to generate reconstructed chroma₁ data 274 derived from the selected image patch.

[0088] In the same or other embodiments, the iteration engine 250 inputs the chroma₂ data 266, optionally the luma data 262, and optionally the chroma₁ data 264 corresponding to the selected image patch into the chroma₂ ML model 246. In response, the chroma₂ ML model 246 downsamples the chroma₂ data 266 by a factor of 4 to generate preprocessed chroma₂ data (not shown). The chroma reconstruction function 248(2) upsamples the preprocessed chroma₂ data by a factor of 4 to generate reconstructed chroma₂ data 276 derived from the selected image patch.

[0089] In some embodiments, the reconstructed luma data 272, the reconstructed chroma₁ data 274, and reconstructed chroma₂ data 276 derived from a given selected image patch define a reconstructed patch corresponding to the selected image patch.

In some embodiments, the reconstructed patches 270 includes, without limitation, one or more of the reconstructed patches.

[0090] Based on the reconstructed data corresponding to the selected image patch, the update engine 290 computes a different reconstruction error for each of the ML models or a single joint reconstruction error. In the same or other embodiments, a reconstruction error is a value for any type of metric that correlates to an end-to-end loss of visual quality between a given source image and a reconstructed version of the source image. In the same or other embodiments, the reconstruction error is modified to take into account the impact of denoising or any other type of preprocessing that intentionally modifies the source image to increase the effectiveness of encoding.

[0091] In some embodiments, the update engine 290 computes a batch reconstruction error for each batch based on the reconstruction errors for the selected images in the batch. In the same or other embodiments, the update engine 290 can update the parameters of the ML models independently based on different batch reconstruction errors or jointly based on joint batch reconstruction errors. As persons skilled in the art will recognize, independently updating the parameters of the ML models corresponds to independently training the ML models, while jointly updating the parameters of the ML models corresponds to jointly training the ML models.

[0092] The update engine 290 can compute the reconstruction error(s) and the batch reconstruction error(s) in any technically feasible fashion. For instance, in some embodiments, the update engine 290 can compute reconstruction error(s) and/or batch reconstruction error(s) using one or more loss functions, one or more cost functions, one or more objective functions, one or more other functions, or any combination thereof.

[0093] As shown, in some embodiments, the update engine 290 computes reconstruction error(s) and batch reconstruction error(s) using loss/cost function(s) 292. The loss/cost function(s) 292 include, without limitation, any number of loss functions and any number of cost functions, where each loss function and each cost function is associated with one or more of the ML models. In some embodiments, each loss function and an associated cost function can be used compute reconstruction error(s) known as "loss(es)" and associated batch reconstruction

error(s) known as “cost(s)” corresponding to one or more of the luma component, the chroma₁ component, or the chroma₂ component.

[0094] For instance, in some embodiments, the loss/cost functions(s) 292 include, without limitation, a luma loss function, a luma cost function, a chroma₁ loss function, a chroma₁ cost function, a chroma₂ loss function, and a chroma₂ cost function. For each selected image patch, the update engine 290 uses the luma loss function to set a luma loss for the selected image patch equal to the mean squared error of the corresponding reconstructed luma data relative to the corresponding luma data. And, for each selected image patch, the update engine 290 uses the chroma₁ loss function to set a chroma₁ loss for the selected image patch equal to the mean squared error of the corresponding reconstructed chroma₁ data relative to the corresponding chroma₁ data. Similarly, for each selected image patch, the update engine 290 uses the chroma₂ loss function to set a chroma₂ loss for the selected image patch equal to the mean squared error of the corresponding reconstructed chroma₂ data relative to the corresponding chroma₂ data.

[0095] In the same or other embodiments, after computing the luma loss for the last selected image patch in each batch, the update engine 290 sets a luma cost equal to the average of the luma losses of the selected image patches in the batch. And after computing the chroma₁ loss for the last selected image patch in each batch, the update engine 290 sets a chroma₁ cost equal to the average of the chroma₁ losses of the of the selected image patches in the batch. Similarly, after computing the chroma₂ loss for the last selected image patch in each batch, the update engine 290 sets a chroma₂ cost equal to the average of the chroma₂ losses of the of the selected image patches in the batch.

[0096] In some other embodiments, the loss/cost functions(s) 292 include, without limitation, a joint loss function and a joint cost function. For each selected image patch, the update engine 290 uses the joint loss function to set a joint loss for the selected image patch equal to the mean squared error of the corresponding reconstructed patch relative to the selected image patch. In the same or other embodiments, after computing the joint loss for the last selected image patch in each batch, the update engine 290 sets a joint cost equal to the average of the joint losses of the selected image patches in the batch.

[0097] In the same or other embodiments, the update engine 290 updates the parameters of the ML models included in the training networks based on any number and/or types of operations performed in a color space other than the luma-chroma color space in which the preprocessing is performed. For instance, in some
5 embodiments, the preprocessing is performed in a non-linear luma-chroma color space, and the update engine 290 computes losses based on an objective function that is defined with respect to a linear luminance-chrominance color space, a linear RGB color space, or a non-linear RGB color space.

[0098] To complete each batch iteration, the update engine 290 individually updates
10 the parameters of each ML model included in the training networks based on the associated batch reconstruction error to generate a new version of the ML model that is better optimized to the goal of reducing the associated batch reconstruction error. For instance, in some embodiments, the update engine 290 updates the parameters of the luma ML model 242, the chroma₁ ML model 244, and the chroma₂ ML model
15 246 based on the luma cost, the chroma₁ cost, and the chroma₂ cost, respectively, for the batch. In some other embodiments, the update engine 290 updates the parameters of the luma ML model 242, the chroma₁ ML model 244, and the chroma₂ ML model 246 based on the joint cost for the batch.

[0099] The update engine 290 can update the parameters for each ML model based
20 on a corresponding batch reconstruction error or a joint batch reconstruction error in any technically feasible fashion. For instance, in some embodiments, the update engine 290 performs any number of backpropagation operations and any number of gradient descent operations on an ML model to update any number of the parameters of the ML model based on a corresponding batch reconstruction error or a joint batch
25 reconstruction error.

[0100] In some embodiments, after the update engine 290 finishes executing the last epoch, the training engine stores the most recent version of each of the ML models included in the training networks in the trained model database 140 as a trained
30 preprocessor for the associated component set that is optionally associated with any number and/or types of characteristics. For instance, as shown, in some embodiments, the update engine 290 stores the luma ML model 242, the chroma₁ ML model 244, and the chroma₂ ML model 246 in the trained model database 140 as a

trained luma preprocessor 142, a trained chroma₁ preprocessor 144, and a trained chroma₂ preprocessor 146, respectively.

5 [0101] Figures 3A-3B set forth a flow diagram of method steps for training one or more machine learning models to preprocess images, according to various embodiments. Although the method steps are described with reference to the systems of Figures 1 and 2, persons skilled in the art will understand that any system configured to implement the method steps, in any order, falls within the scope of the embodiments.

10 [0102] As shown, a method 300 begins at step 302, where the training application 130 determines one or more mutually exclusive component set(s) 222 associated with a luma-chroma color space. The training application 130 can determine the component set(s) 222 in any technically feasible fashion. For instance, in some embodiments, the training application 130 determines the component set(s) 222 based on the preprocessing specification 120. At step 304, the training application 130 determines
15 at least one of a luma sampling factor 224 or a chroma sampling factor 226 based on a preprocessing specification.

[0103] At step 306, for each component set, the training application 130 determines an initial untrained or pre-trained version of an associated ML model based on the corresponding scaling factor. At step 308, for each component set, the training
20 application 130 determines and appends a reconstruction function to the associated ML model to generate an associated training network.

[0104] At step 310, the training application 130 performs any number and/or types of operations on any number of training images to determine image patches 260 represented in the luma-chroma color space. At step 312, the training application 130
25 distributes the image patches 260 across any number of batches and selects the first batch. At step 314, the training application 130 selects the image patches included in the selected batch. At step 316, for each selected image patch, the training application 130 configures each training network to generate reconstructed data corresponding to the associated component set based on the selected image patch.

30 [0105] At step 318, the training application 130 determines whether the training application 130 is to perform joint training of the ML model(s). At step 318, if the

training application 130 determines that the training application 130 is not to perform joint training, then the method 300 proceeds to step 320. At step 320, for each ML model, the training application 130 computes a loss for each selected image patch based on the reconstructed data corresponding to the associated component set and
5 derived from the selected image patch. At step 322, for each ML model, the training application 130 computes a cost based on the associated losses. At step 324, for each ML model, the training application 130 updates any number of the parameters of the ML model based on the associated iteration cost. The method 300 then proceeds directly to step 332.

- 10 **[0106]** If, however, at step 318, the training application 130 determines that the training application 130 is to perform joint training, then the method 300 proceeds directly to step 326. At step 326, the training application 130 computes a joint loss for each selected image patch based on the reconstructed data derived from the selected image patch. At step 328, the training application 130 computes a joint cost based on
15 the joint losses. At step 330, the training application 130 updates any number of the parameters of each ML model based on the joint cost.

- [0107]** At step 332, the training application 130 determines whether the selected batch is the last batch. If, at step 332, the training application 130 determines that the selected batch is not the last batch, then the method 300 proceeds to step 334. At
20 step 334, the training application 130 selects the next batch, and the method 300 returns to step 314, where the training application 130 selects the image patches included in the selected batch.

- [0108]** If, however, at step 332, the training application 130 determines that the selected batch is the last batch, then the method 300 proceeds directly to step 336.
25 At step 336, the training application 130 determines whether the current epoch is the last epoch. If, at step 336, the training application 130 determines that the current epoch is not the last epoch, then the method 300 proceeds to step 338. At step 338, the training application 130 selects the first batch, and the method 300 returns to step 314, where the training application 130 selects the image patches included in the
30 selected batch.

[0109] If, however, at step 336, the training application 130 determines that the current epoch is the last epoch, then the method 300 proceeds directly to step 340.

At step 340, the training application 130 stores the more recent version of each ML model in a trained model database as a trained preprocessor for the associated component set that is optionally associated with any number and/or types of characteristics. The method 300 then terminates.

5 Component-Based Image Preprocessing

[0110] Figure 4 is an exemplar illustration of the image preprocessing application 150 of Figure 1, according to various embodiments. More specifically, the image preprocessing application 150 depicted in Figure 5 implements a luma spatial preprocessor 442, a chroma₁ spatial preprocessor 444, and a chroma₂ spatial preprocessor 446 that independently preprocess luma data, chroma₁ data, and chroma₂ data, respectively. In some other embodiments, the image preprocessing application 150 can implement fewer spatial preprocessors, and at least one of the spatial preprocessors jointly preprocesses multiple components of the luma-chroma color space.

15 **[0111]** As shown, in some embodiments, the image preprocessing application 150 includes, without limitation, a composite model engine 420, the luma spatial preprocessor 442, the chroma₁ spatial preprocessor 444, the chroma₂ spatial preprocessor 446, and a luma-chroma image 450. In the same or other
20 embodiments, the composite model engine 420 generates one or more spatial preprocessors based on the preprocessing specification 120(1), the trained model database 140, and optionally the non-ML preprocessors 106.

[0112] The preprocessing specification 120(1) and the preprocessing specification 120(0) described previously herein in conjunction with Figures 1 and 2 are different instances of a generic preprocessing specification that can have different values. In some embodiments, the trained model database 140 includes, without limitation, the trained luma preprocessor 142, the trained chroma₁ preprocessor 144, and the trained chroma₂ preprocessor 146. In the same or other embodiments, the non-ML preprocessors 106 include, without limitation, a non-ML chroma subsampler 402, a non-ML downsampler 404, and a non-ML denoiser 406.

30 **[0113]** As shown, in some embodiments, the composite model engine 420 includes, without limitation, component set(s) 422, a luma sampling factor 424, and a chroma sampling factor 426. The component set(s) 422 include, without limitation, one or

more mutually exclusive component sets that, collectively, specify the luma component, the chroma₁ component, and the chroma₂ component. The composite model engine 420 can determine the component set(s) 422 in any technically feasible fashion. For instance, in some embodiments, the composite model engine 420 can
5 determine the component set(s) 422 based on the preprocessing specification 120(1). In some embodiments, for each component set included in the component set(s) 422, the composite model engine 420 implements a different temporal preprocessor that preprocesses data corresponding to the one or more components specified in the component set.

10 **[0114]** As shown, in some embodiments, including the embodiment depicted in Figure 4, each component set is a singleton specifying a different component of the luma-chroma color space and therefore the component set(s) 422 are {luma}, {chroma₁}, and {chroma₂}. In the same or other embodiments, based on the component set(s) 422 of {luma}, {chroma₁}, and {chroma₂}, the composite model engine 420
15 implements the luma spatial preprocessor 442, the chroma₁ spatial preprocessor 444, and the chroma₂ spatial preprocessor 446. By contrast, in some other embodiments, based on the component set(s) 422 of {luma} and {chroma₁, chroma₂}, the training application 130 implements the luma spatial preprocessor 442 that preprocesses luma data and a joint spatial preprocessor (not shown) that jointly preprocesses
20 chroma₁ data and chroma₂ data.

[0115] In some embodiments, the luma sampling factor 424 specifies consolidated downsizing that is to be applied to luma data, and the chroma sampling factor 426 specifies consolidated downsizing that is to be applied to chroma data. The composite model engine 420 can determine the luma sampling factor 424 and the chroma
25 sampling factor 426 in any technically feasible fashion. In some embodiments, the composite model engine 420 determines the luma sampling factor 424 and the chroma sampling factor 426 based on the preprocessing specification 120(1).

[0116] In the same or other embodiments, because downsizing but not chroma subsampling reduces the size of luma data, the composite model engine 420 sets the
30 luma sampling factor 424 equal to the downscaling factor 122(1). And because downscaling and chroma subsampling both reduce the size of chroma data, the composite model engine 420 sets the chroma sampling factor 426 equal to an overall downsizing of chroma data that reflects downscaling and chroma subsampling.

[0117] In some embodiments, for each of the component set(s) 422, the composite model engine 420 determines a spatial preprocessor that complies with the preprocessing specification 120(1) and includes, without limitation, one or more trained preprocessors and/or one or more non-ML preprocessors. The composite model engine 420 can determine each spatial preprocessor in any technically feasible fashion. In some embodiments, to generate each spatial preprocessor, the composite model engine 420 attempts to determine a trained preprocessor from the trained model database 140 that implements the corresponding sampling factor (e.g., the luma sampling factor 424 or the chroma sampling factor 426) and any other specified preprocessing (e.g., denoising). If the composite model engine 420 fails to determine a single trained preprocessor that complies with the preprocessing specification 120(1), then the composite model engine 420 aggregates zero or more trained preprocessors and zero or more non-ML preprocessors 106 to generate the spatial preprocessor.

[0118] For explanatory purposes, the preprocessing the image preprocessing application 150 performs on each of any number of source images are described and depicted in Figure 4 in the context of preprocessing the source image 108. In some embodiments, the source image 108 can be represented in any type of color space, and the image preprocessing application 150 performs any number and or types of operations on the source image 108 to determine luma-chroma image 450 that represents the source image 108 in the luma-chroma color space. As shown, the luma-chroma image 450 includes, without luma data 452, chroma₁ data 454, and chroma₂ data 456.

[0119] The image preprocessing application 150 configures each of the spatial preprocessor(s) to generate preprocessed data for the corresponding component set based on at least the data corresponding to the corresponding component set. As shown, in some embodiments, the image preprocessing application 150 inputs the luma data 452, optionally the chroma₁ data 454, and optionally the chroma₂ data 456 into the luma spatial preprocessor 442. In response, the luma spatial preprocessor 442 generates preprocessed luma data 492. In the same or other embodiments, the image preprocessing application 150 inputs the chroma₁ data 454, optionally the luma data 452, and optionally the chroma₂ data 456 into the chroma₁ spatial preprocessor 444. In response, the chroma₁ spatial preprocessor 444 generates preprocessed

chroma₁ data 494. In the same or other embodiments, the image preprocessing application 150 inputs the chroma₂ data 456, optionally the luma data 452, and optionally the chroma₁ data 454 into the chroma₂ spatial preprocessor 446. In response, the chroma₂ spatial preprocessor 446 generates preprocessed chroma₂ data 496.

[0120] The image preprocessing application 150 aggregates the preprocessed luma data 492, the preprocessed chroma₁ data 494, and the preprocessed chroma₂ data 496 to generate the preprocessed image 158. The image preprocessing application 150 stores the preprocessed image 158 and/or transmits the preprocessed image 158 to any number of software applications.

[0121] Figure 5 is a flow diagram of method steps for preprocessing a source video, according to various embodiments. Although the method steps are described with reference to the systems of Figures 1, 2, and 4, persons skilled in the art will understand that any system configured to implement the method steps, in any order, falls within the scope of the embodiments.

[0122] As shown, a method 500 begins at step 502, where the video preprocessing application 180 partitions the source video 170 into source images 172. At step 504, based on the preprocessing specification 120(1), the video preprocessing application 180 performs component-based preprocessing of the source images 172 in a luma-chroma color space to generate preprocessed images 178.

[0123] The video preprocessing application 180 can execute and/or cause the execution of any number and/or types of operations on the source images 172 to generate the preprocessed images 178 in any technically feasible fashion. For instance, in some embodiments, the video preprocessing application 180 transmits the source images 172 and the preprocessing specification 120(1) to the image preprocessing application 150 that implements the method steps described below in conjunction with Figure 6 to generate the preprocessed images 178.

[0124] At step 506, the video preprocessing application 180 aggregates the preprocessed images 178 to generate a spatially preprocessed video. At step 508, the video preprocessing application 180 performs zero or more temporal preprocessing operations on the spatially preprocessed video to generate

preprocessed video 188. At step 510, the video preprocessing application 180 stores the preprocessed video 188 and/or transmits the preprocessed video 188 to video encoder 196 and/or any number of other software applications. The method 500 then terminates.

- 5 **[0125]** Figure 6 is a flow diagram of method steps for preprocessing source images according to various embodiments. Although the method steps are described with reference to the systems of Figures 1, 2, and 4, persons skilled in the art will understand that any system configured to implement the method steps, in any order, falls within the scope of the embodiments.
- 10 **[0126]** As shown, a method 600 begins at step 602, where the image preprocessing application 150 determines one or more mutually exclusive component set(s) 422 that collectively specify the luma, chroma₁, and chroma₂ components of a luma-chroma color space. The image preprocessing application 150 can determine the component set(s) 422 in any technically feasible fashion. For instance, in some embodiments,
- 15 the image preprocessing application 150 can determine the component set(s) 422 based on a preprocessing specification. In the same or other embodiments, each component set is a singleton specifying a different component of the luma-chroma color space. In some other embodiments, one component set is a singleton specifying the luma component of a luma-chroma color space and another
- 20 component set specifying the two chroma components of the luma-chroma color space.

- [0127]** At step 604, for each component set, the image preprocessing application 150 determines a spatial preprocessor that complies with a preprocessing specification and includes, without limitation, one or more trained preprocessors and/or one or
- 25 more non-ML preprocessors. At step 606, the image preprocessing application 150 selects a first source image. At step 608, the image preprocessing application 150 performs any number and/or types of operations on the selected source image to determine luma data, chroma₁ data, and chroma₂ data. At step 610, the image preprocessing application 150 causes each spatial preprocessor to generate
- 30 preprocessed data corresponding to the associated component set based on at least one of the luma data, the chroma₁ data, and the chroma₂ data. At step 612, the image preprocessing application 150 aggregates the preprocessed luma data, the

preprocessed chroma₁ data, and the preprocessed chroma₂ data to generate a preprocessed image corresponding to the source image.

[0128] At step 614, the image preprocessing application 150 determines whether the selected source image is the last source image. If, at step 614, the image

5 preprocessing application 150 determines that the selected source image is not the last source image, then the method 600 proceeds to step 616. At step 616, the image preprocessing application 150 selects the next source image, and the method 600 returns to step 608, where the image preprocessing application 150 determines luma data, chroma₁ data, and chroma₂ data corresponding to the newly selected source
10 image.

[0129] If, however, at step 616, the image preprocessing application 150 determines that the selected source image is the last source image, then the method 600 proceeds directly to step 618. At step 618, the image preprocessing application 150 stores each preprocessed image and/or transmits each preprocessed image to any
15 number of software applications. The method 600 then terminates.

[0130] In sum, the disclosed techniques can be used to reduce an overall degradation in visual quality typically associated with streaming videos. In some embodiments, a training application generates a different trained preprocessor for each component of a luma-chroma color space based on a preprocessing specification and a training
20 image database. The preprocessing specification includes, without limitation, a downscaling factor, a chroma subsampling ratio, and optionally specifies any amount and/or type of other preprocessing. The training application sets a luma sampling factor equal to the downscaling factor and computes a chroma sampling factor based on the luma sampling factor and the chroma subsampling ratio. In some
25 embodiments, the training application determines untrained or pre-trained versions of a luma ML model, a chroma₁ ML model, or a chroma₂ ML model that downsample by the luma sampling factor, the chroma sampling factor, and the chroma sampling factor, respectively and optionally implement any other specified preprocessing. The training application generates a training network for each ML model that includes,
30 without limitation, the ML model followed by an upsampling algorithm that upsamples by the scaling factor associated with the ML model.

[0131] In some embodiments, the training application performs any number of conversion and/or extraction operations on training images included in a training image database to generate luma data, chroma₁ data, and chroma₂ data corresponding to each of any number of image patches. The training application

5 distributes the image patches across any number of batches and executes batch iterations for any number of epochs. To execute a batch iteration for a given batch, the training application selects the image patches in the batch. For each selected image patch, the training application inputs at least the corresponding luma data into the luma training network, at least the corresponding chroma₁ data into the chroma₁

10 training network, and at least the corresponding chroma₂ data into the chroma₁ training network. In response, for each selected image patch, the luma ML model, the chroma₁ ML model, and the chroma₂ ML models generate preprocessed luma data, preprocessed chroma₁ data, and preprocessed chroma₂ data, respectively. The upsampling algorithms in the luma training network, the chroma₁ training network, and

15 the chroma₂ training network upsample the preprocessed luma data, preprocessed chroma₁ data, and preprocessed chroma₂ data, respectively, to generate reconstructed luma data, reconstructed chroma₁ data, and reconstructed chroma₂ data, respectively. The reconstructed luma data, reconstructed chroma₁ data, and reconstructed chroma₂ data define a reconstructed patch corresponding to the

20 selected image patch.

[0132] For each of the selected image patches, the training application sets a joint loss equal to the mean squared error of the corresponding reconstructed patch relative to the selected image patch. The training application sets a joint cost equal to the average of the joint losses of the selected image patches. The training application

25 individually updates the parameters of the luma ML model, the chroma₁ ML model, and the chroma₂ ML based on the joint cost to complete the batch iteration. After the training engine finishes executing the last epoch, the training engine stores the most recent version of the luma ML model, the chroma₁ ML model, and the chroma₂ ML model as a trained luma preprocessor, a trained chroma₁ preprocessor, and a trained

30 chroma₂ preprocessor, respectively.

[0133] In the same or other embodiments, a video preprocessing application generates a preprocessed video based on a source video and a preprocessing specification. The video preprocessing application splits the source video into source

images. The video preprocessing application transmits the source images and a preprocessing specification to an image preprocessing application. The image preprocessing application attempts to satisfy the preprocessing specification using a different trained preprocessor for each of a luma spatial preprocessor, a chroma₁ spatial preprocessor, and a chroma₂ spatial preprocessor equal to a trained preprocessor. If the image preprocessing application is unable to satisfy the preprocessing specification for a given component using a single trained preprocessor, then the image preprocessing application aggregates zero or more trained preprocessors and zero or more non-ML preprocessors to generate a corresponding spatial preprocessor. The image preprocessing application performs any number and/or types of conversion and/or extraction operations on each source image to determine luma data, chroma₁ data, and chroma₂ data that collectively represent the image in a luma-chroma color space.

[0134] For each source image, the image preprocessing application configures the luma spatial preprocessor to map at least the luma data corresponding to the source image to preprocessed luma data. The image preprocessing application configures the chroma₁ spatial preprocessor to map at least the chroma₁ data corresponding to the source image to preprocessed chroma₁ data. The image preprocessing application configures the chroma₂ spatial preprocessor to map at least the chroma₂ data corresponding to the source image to preprocessed chroma₂ data. The image preprocessing application aggregates the preprocessed luma data, the preprocessed chroma₁ data, and the preprocessed chroma₂ data to generate a preprocessed image that is derived from the source image. The image preprocessing application transmits the preprocessed images to the video preprocessing application. The video preprocessing application aggregates and optionally performs any number and/or types of temporal preprocessing operations on the preprocessed images to generate a preprocessed video.

[0135] At least one technical advantage of the disclosed techniques relative to the prior art is that the disclosed techniques can be implemented in a video encoding pipeline to mitigate the visual quality reductions typically experienced with conventional video encoding pipelines. In that regard, a trained preprocessor implemented in a video encoding pipeline learns to reduce reconstruction errors when preprocessing data corresponding to at least one component in a luma-chroma color

space. The reconstruction errors correlate to decreases in the visual quality of reconstructed images attributable to preprocessing source images within a video encoding pipeline. Thus, with the disclosed techniques, the visual quality of a reconstructed video can be increased for a given bitrate relative to what can be achieved using prior art techniques. Conversely, with the disclosed techniques, the number of bits used when encoding a source video to achieve a given target visual quality can be reduced relative to what can be achieved using prior art techniques. Another technical advantage of the disclosed techniques is that the training application can generate different trained preprocessor(s) that are fine-tuned for different types of images (e.g., images in an animated film). Accordingly, when preprocessing images of a given type, the image preprocessing application can use the trained preprocessor(s) that are fine-tuned for that image type, which increases the effectiveness of the related preprocessing operations. These technical advantages provide one or more technological improvements over prior art approaches.

[0136] 1. In some embodiments, a computer-implemented method for training a machine learning model to preprocess images comprises computing a chroma sampling factor based on a downscaling factor and a chroma subsampling ratio; executing a first machine learning model that is associated with the chroma sampling factor on first data that corresponds to both a first image and a first chroma component to generate first preprocessed data corresponding to the first chroma component; and updating at least one parameter of the first machine learning model based on the first preprocessed data to generate a first trained machine learning model that is associated with the first chroma component.

[0137] 2. The computer-implemented method of clause 1, wherein a first resolution of the first data is equal to the product of the chroma sampling factor and a second resolution of the first preprocessed data.

[0138] 3. The computer-implemented method of clauses 1 or 2, wherein computing the chroma subsampling factor comprises converting the chroma subsampling ratio to a multiplier; and multiplying the downscaling factor by the multiplier.

[0139] 4. The computer-implemented method of any of clauses 1-3, further comprising executing a second machine learning model that is associated with the downscaling

factor on second data corresponding to both the first image and a luma component to generate second preprocessed data corresponding to the luma component; and updating at least one parameter of the second machine learning model based on the second preprocessed data to generate a second trained machine learning model that
5 is associated with the luma component.

[0140] 5. The computer-implemented method of any of clauses 1-4, wherein, when executing the first machine learning model, the first data and second data corresponding to both the first image and a second chroma component are input into the first machine learning model.

10 **[0141]** 6. The computer-implemented method of any of clauses 1-5, wherein updating the at least one parameter of the first machine learning model comprises executing an upsampling algorithm on the first preprocessed data to generate reconstructed data; computing a first reconstruction error based on the reconstructed data; and performing at least one of a backpropagation operation or a gradient descent
15 operation on the first machine learning model based on the first reconstruction error to update the at least one parameter of the first machine learning model.

[0142] 7. The computer-implemented method of any of clauses 1-6, wherein updating the at least one parameter of the first machine learning model comprises executing an upsampling algorithm on the first preprocessed data to generate first reconstructed
20 data; computing a joint reconstruction error based on the first reconstructed data; performing at least one of a backpropagation operation or a gradient descent operation on the first machine learning model based on the joint reconstruction error to update the at least one parameter of the first machine learning model; and performing at least one of a backpropagation operation or a gradient descent
25 operation on a second machine learning model based on the joint reconstruction error to generate a second trained machine learning model that is associated with a second component.

[0143] 8. The computer-implemented method of any of clauses 1-7, wherein the first machine learning model comprises a convolutional neural network.

30 **[0144]** 9. The computer-implemented method of any of clauses 1-8, wherein the first image comprises at least a portion of a frame included in a video.

[0145] 10. The computer-implemented method of any of clauses 1-9, wherein the first chroma component comprises a blue-difference chroma component or a red-difference chroma component.

[0146] 11. In some embodiments, one or more non-transitory computer readable
5 media include instructions that, when executed by one or more processors, cause the one or more processors to train a machine learning model to preprocess images by performing the steps of identifying a first machine learning model based on a chroma sampling factor that reflects both a downscaling factor and a chroma subsampling ratio; executing the first machine learning model on first data that corresponds to both
10 a first image and a first chroma component to generate first preprocessed data corresponding to the first chroma component; and modifying the first machine learning model based on the first preprocessed data to generate a first trained machine learning model that is associated with the first chroma component.

[0147] 12. The one or more non-transitory computer readable media of clause 11,
15 wherein a first resolution of the first data is equal to the product of the chroma sampling factor and a second resolution of the first preprocessed data.

[0148] 13. The one or more non-transitory computer readable media of clauses 11 or 12, further comprising converting the chroma subsampling ratio to a multiplier; and multiplying the downscaling factor by the multiplier to compute the chroma sampling
20 factor.

[0149] 14. The one or more non-transitory computer readable media of any of clauses 11-13, further comprising executing a second machine learning model that is associated with the downscaling factor on second data that corresponds to both the first image and a luma component to generate second preprocessed data
25 corresponding to the luma component; and modifying the second machine learning model based on the second preprocessed data to generate a second trained machine learning model that is associated with the luma component.

[0150] 15. The one or more non-transitory computer readable media of any of clauses 11-14, wherein, when executing the first machine learning model, the first data and
30 second data corresponding to both the first image and a luma component are input into the first machine learning model.

[0151] 16. The one or more non-transitory computer readable media of any of clauses 11-15, wherein modifying the first machine learning model comprises executing an upsampling algorithm on the first preprocessed data to generate reconstructed data; computing a first reconstruction error based on the reconstructed data; and
5 performing at least one of a backpropagation operation or a gradient descent operation on the first machine learning model based on the first reconstruction error to update at least one parameter of the first machine learning model.

[0152] 17. The one or more non-transitory computer readable media of any of clauses 11-16, wherein modifying the first machine learning model comprises executing an
10 upsampling algorithm on the first preprocessed data to generate first reconstructed data; computing a joint reconstruction error based on the first reconstructed data; performing at least one of a backpropagation operation or a gradient descent operation on the first machine learning model based on the joint reconstruction error to update at least one parameter of the first machine learning model; and performing
15 at least one of a backpropagation operation or a gradient descent operation on a second machine learning model based on the joint reconstruction error to generate a second trained machine learning model that is associated with a second component.

[0153] 18. The one or more non-transitory computer readable media of any of clauses 11-17, wherein the first machine learning model is associated with a non-linear luma-
20 chroma color space, and the joint reconstruction error is associated with a linear luminance-chrominance color space, a linear RGB color space, or a non-linear RGB color space.

[0154] 19. The one or more non-transitory computer readable media of any of clauses 11-18, wherein the chroma subsampling ratio comprises 4 2 0, 4 2 2, 4 2 1, 4 1 1, 4 1
25 0, or 3 1 1.

[0155] 20. In some embodiments, a system comprises one or more memories storing instructions and one or more processors coupled to the one or more memories that, when executing the instructions, perform the steps of computing a chroma sampling factor based on a downscaling factor and a chroma subsampling ratio; executing a
30 first machine learning model that is associated with the chroma sampling factor on first data that corresponds to both a first image and a first chroma component to generate first preprocessed data corresponding to the first chroma component; and

modifying the first machine learning model based on the first preprocessed data to generate a first trained machine learning model that is associated with the first chroma component.

5 [0156] Any and all combinations of any of the claim elements recited in any of the claims and/or any elements described in this application, in any fashion, fall within the contemplated scope of the embodiments and protection.

[0157] The descriptions of the various embodiments have been presented for purposes of illustration but are not intended to be exhaustive or limited to the embodiments disclosed. Many modifications and variations will be apparent to those
10 of ordinary skill in the art without departing from the scope and spirit of the described embodiments.

[0158] Aspects of the present embodiments may be embodied as a system, method, or computer program product. Accordingly, aspects of the present disclosure may take the form of an entirely hardware embodiment, an entirely software embodiment
15 (including firmware, resident software, micro-code, etc.) or an embodiment combining software and hardware aspects that may all generally be referred to herein as a “module,” a “system,” or a “computer.” In addition, any hardware and/or software technique, process, function, component, engine, module, or system described in the present disclosure may be implemented as a circuit or set of circuits. Furthermore,
20 aspects of the present disclosure may take the form of a computer program product embodied in one or more computer-readable media having computer-readable program code embodied thereon.

[0159] Any combination of one or more computer-readable media may be utilized. The computer-readable medium may be a computer-readable signal medium or a
25 computer-readable storage medium. A computer-readable storage medium may be, for example, but not limited to, an electronic, magnetic, optical, electromagnetic, infrared, or semiconductor system, apparatus, or device, or any suitable combination of the foregoing. More specific examples (a non-exhaustive list) of the computer-readable storage medium would include the following: an electrical connection having
30 one or more wires, a portable computer diskette, a hard disk, a random access memory, a read-only memory, an erasable programmable read-only memory, a Flash memory, an optical fiber, a portable compact disc read-only memory, an optical

storage device, a magnetic storage device, or any suitable combination of the foregoing. In the context of this document, a computer-readable storage medium may be any tangible medium that can contain or store a program for use by or in connection with an instruction execution system, apparatus, or device.

5 **[0160]** Aspects of the present disclosure are described above with reference to flowchart illustrations and/or block diagrams of methods, apparatus (systems) and computer program products according to embodiments of the disclosure. It will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, can be
10 implemented by computer program instructions. These computer program instructions can be provided to a processor of a general-purpose computer, special purpose computer, or other programmable data processing apparatus to produce a machine. The instructions, when executed via the processor of the computer or other programmable data processing apparatus, enable the implementation of the
15 functions/acts specified in the flowchart and/or block diagram block or blocks. Such processors may be, without limitation, general purpose processors, special-purpose processors, application-specific processors, or field-programmable gate arrays.

[0161] The flowchart and block diagrams in the figures illustrate the architecture, functionality, and operation of possible implementations of systems, methods, and
20 computer program products according to various embodiments of the present disclosure. In this regard, each block in the flowchart or block diagrams may represent a module, segment, or portion of code, which comprises one or more executable instructions for implementing the specified logical function(s). It should also be noted that, in some alternative implementations, the functions noted in the
25 block may occur out of the order noted in the figures. For example, two blocks shown in succession may, in fact, be executed substantially concurrently, or the blocks may sometimes be executed in the reverse order, depending upon the functionality involved. It will also be noted that each block of the block diagrams and/or flowchart illustration, and combinations of blocks in the block diagrams and/or flowchart
30 illustration, can be implemented by special purpose hardware-based systems that perform the specified functions or acts, or combinations of special purpose hardware and computer instructions.

[0162] While the preceding is directed to embodiments of the present disclosure, other and further embodiments of the disclosure may be devised without departing from the basic scope thereof, and the scope thereof is determined by the claims that follow.

WHAT IS CLAIMED IS:

1. A computer-implemented method for training a machine learning model to preprocess images, the method comprising:
 - 5 computing a chroma sampling factor based on a downscaling factor and a chroma subsampling ratio;
 - executing a first machine learning model that is associated with the chroma sampling factor on first data that corresponds to both a first image and a first chroma component to generate first preprocessed data corresponding to the first chroma component; and
 - 10 updating at least one parameter of the first machine learning model based on the first preprocessed data to generate a first trained machine learning model that is associated with the first chroma component.
2. The computer-implemented method of claim 1, wherein a first resolution of the
15 first data is equal to the product of the chroma sampling factor and a second resolution of the first preprocessed data.
3. The computer-implemented method of claim 1, wherein computing the chroma subsampling factor comprises:
 - 20 converting the chroma subsampling ratio to a multiplier; and
 - multiplying the downscaling factor by the multiplier.
4. The computer-implemented method of claim 1, further comprising:
 - 25 executing a second machine learning model that is associated with the downscaling factor on second data corresponding to both the first image and a luma component to generate second preprocessed data corresponding to the luma component; and
 - updating at least one parameter of the second machine learning model based on the second preprocessed data to generate a second trained machine
30 learning model that is associated with the luma component.
5. The computer-implemented method of claim 1, wherein, when executing the first machine learning model, the first data and second data corresponding to both the first image and a second chroma component are input into the first machine learning

model.

6. The computer-implemented method of claim 1, wherein updating the at least one parameter of the first machine learning model comprises:

- 5 executing an upsampling algorithm on the first preprocessed data to generate reconstructed data;
 computing a first reconstruction error based on the reconstructed data; and
 performing at least one of a backpropagation operation or a gradient descent operation on the first machine learning model based on the first
10 reconstruction error to update the at least one parameter of the first machine learning model.

7. The computer-implemented method of claim 1, wherein updating the at least one parameter of the first machine learning model comprises:

- 15 executing an upsampling algorithm on the first preprocessed data to generate first reconstructed data;
 computing a joint reconstruction error based on the first reconstructed data;
 performing at least one of a backpropagation operation or a gradient descent operation on the first machine learning model based on the joint
20 reconstruction error to update the at least one parameter of the first machine learning model; and
 performing at least one of a backpropagation operation or a gradient descent operation on a second machine learning model based on the joint reconstruction error to generate a second trained machine learning
25 model that is associated with a second component.

8. The computer-implemented method of claim 1, wherein the first machine learning model comprises a convolutional neural network.

- 30 9. The computer-implemented method of claim 1, wherein the first image comprises at least a portion of a frame included in a video.

10. The computer-implemented method of claim 1, wherein the first chroma component comprises a blue-difference chroma component or a red-difference

chroma component.

11. One or more non-transitory computer readable media including instructions that, when executed by one or more processors, cause the one or more processors to
- 5 train a machine learning model to preprocess images by performing the steps of:
- identifying a first machine learning model based on a chroma sampling factor that reflects both a downscaling factor and a chroma subsampling ratio;
 - executing the first machine learning model on first data that corresponds to both a first image and a first chroma component to generate first
 - 10 preprocessed data corresponding to the first chroma component; and
 - modifying the first machine learning model based on the first preprocessed data to generate a first trained machine learning model that is associated with the first chroma component.
- 15 12. The one or more non-transitory computer readable media of claim 11, wherein a first resolution of the first data is equal to the product of the chroma sampling factor and a second resolution of the first preprocessed data.
13. The one or more non-transitory computer readable media of claim 11, further
- 20 comprising:
- converting the chroma subsampling ratio to a multiplier; and
 - multiplying the downscaling factor by the multiplier to compute the chroma sampling factor.
- 25 14. The one or more non-transitory computer readable media of claim 11, further comprising:
- executing a second machine learning model that is associated with the downscaling factor on second data that corresponds to both the first image and a luma component to generate second preprocessed data
 - 30 corresponding to the luma component; and
 - modifying the second machine learning model based on the second preprocessed data to generate a second trained machine learning model that is associated with the luma component.

15. The one or more non-transitory computer readable media of claim 11, wherein, when executing the first machine learning model, the first data and second data corresponding to both the first image and a luma component are input into the first machine learning model.

5

16. The one or more non-transitory computer readable media of claim 11, wherein modifying the first machine learning model comprises:

executing an upsampling algorithm on the first preprocessed data to generate reconstructed data;

10 computing a first reconstruction error based on the reconstructed data; and performing at least one of a backpropagation operation or a gradient descent operation on the first machine learning model based on the first reconstruction error to update at least one parameter of the first machine learning model.

15

17. The one or more non-transitory computer readable media of claim 11, wherein modifying the first machine learning model comprises:

executing an upsampling algorithm on the first preprocessed data to generate first reconstructed data;

20 computing a joint reconstruction error based on the first reconstructed data; performing at least one of a backpropagation operation or a gradient descent operation on the first machine learning model based on the joint reconstruction error to update at least one parameter of the first machine learning model; and

25 performing at least one of a backpropagation operation or a gradient descent operation on a second machine learning model based on the joint reconstruction error to generate a second trained machine learning model that is associated with a second component.

30 18. The one or more non-transitory computer readable media of claim 17, wherein the first machine learning model is associated with a non-linear luma-chroma color space, and the joint reconstruction error is associated with a linear luminance-chrominance color space, a linear RGB color space, or a non-linear RGB color space.

19. The one or more non-transitory computer readable media of claim 11, wherein the chroma subsampling ratio comprises 4:2:0, 4:2:2, 4:2:1, 4:1:1, 4:1:0, or 3:1:1.

20. A system comprising:

- 5 one or more memories storing instructions; and
 one or more processors coupled to the one or more memories that, when
 executing the instructions, perform the steps of:
 computing a chroma sampling factor based on a downscaling factor and
 a chroma subsampling ratio;
10 executing a first machine learning model that is associated with the
 chroma sampling factor on first data that corresponds to both a
 first image and a first chroma component to generate first
 preprocessed data corresponding to the first chroma component;
 and
15 modifying the first machine learning model based on the first
 preprocessed data to generate a first trained machine learning
 model that is associated with the first chroma component.