A Report

On

Feature Extraction:

Text, Image, Video, and Audio Data

By

Yash Gupta 2019A7PS1138P

Prepared in Partial Fulfillment of the Course

CS F320 Foundations of Data Science

Birla Institute of Technology and Science (BITS), Pilani

November, 2021

Table of Contents

Introduction

We are living in a data-driven world to an extent that one might argue that data science has become the fourth paradigm of science - the other three being, experimental, theoretical, and computational sciences. According to Peter Sondergaard of Gartner Research, 'information' is the *'oil'* of the present century, and 'analytics' is the *'combustion engine.'*

Much of the data, however, rarely is in a form that is conveniently processed by the existing machine learning and data mining algorithms or statistical procedures. Furthermore, data today exists in multiple modalities, rarely appearing as a nice table of numbers. Data scientists must be braced to face text, audio, image, and video data on a daily basis. Oftentimes finding the best representation of data for the task is the most arduous and important step in the overall process, more so than the algorithm used for analysis later. Thus, it only makes sense to develop proper feature extraction strategies for the best utilization of data - the veritable *treasure* of the big-tech companies today.

As is the case with most fields of science and technology, data science demands its own set of arsenals, tensors being one of the most basic and versatile of them. The idea of tensors, along with convenient resources and tools for processing and working with them - Tensor Processing Units and Tensorflow - will be briefly explored in the report.

1. Feature Extraction

The creation of a new set of features from the original raw data is known as feature extraction [1]. The process of feature extraction aims at obtaining the maximum amount of useful information from the data, while keeping the computation expenses at the minimum. It involves reducing the number of resources required to describe a large set of data [2].

Often, feature extraction not only simplifies the data from a resource expense point of view, but it also renders it in a form that is easy-to-analyze for human experts. For instance, audio data, when represented as a time series, can be analyzed through the rich array of statistical methods for time series analysis.

It begins with an initial set of raw data and derives informative and non-redundant attributes, suitable for the analysis or learning job at hand. As such it can be considered the part of data preparation/preprocessing phase in the general data mining job cycle, as shown in Figure 1.1 below.

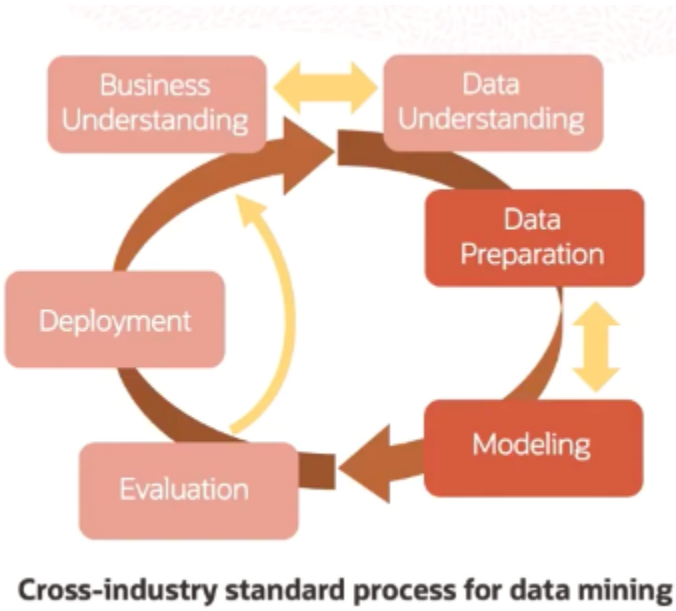

Figure 1.1 Feature Extraction can be put under the Data Preparation phase of the cycle.
Image Credit: Oracle Developers - Machine Learning 101

While related to dimensionality reduction, feature extraction is not the same as it. It may involve a feature selection step to determine a subset of features that are non-redundant and relevant so that the input data size becomes manageable for the algorithm [2]. However, feature extraction not only aims to reduce the number of features but also builds a representation that will be useful to the task, and it may happen that the most relevant representation is not the smallest one. For instance, we may have to continue working with an RGB representation of an image in

favour of a grayscale one when making sense out of traffic light signals, even though the latter is a more concise representation.

In the following sections, some useful representations and feature extraction strategies for text, image, video, and audio have been explored.

2. Text Data

Feature extraction for text data is the first step in a variety of natural language processing (NLP) and machine learning tasks, including sentiment analysis, language translation, and spam filtering. However, raw text can hardly be used for analysis directly. It is imperative to render it into a numerical or categorical dataframe to find patterns in a seemingly unorganized corpus. The most common strategies employed for text data are Bag-of-words, TF-IDF vectorization, and Word Embedding as discussed below.

2.1 Bag of Words

The bag-of-words model for text is often used for document classification, each word, or groups thereof, helping us to make a decision about the category of the document. We make a list of unique words in the text corpus called vocabulary. We represent each document as a vector, where 0 and 1 respectively represent the absence and presence of a particular word from the vocabulary. We can use a non-zero value instead of 1 for a word to count its number of occurrences in the document.

Let us take an example to illustrate the step-wise process of creating a bag-of-words. Suppose the document contains a single sentence, *"Every morning, Dave runs on the running track ."*

- *Tokenize* the document, that is, convert it into building blocks like words and punctuation.

Every	morning	Dave	runs	on	the	running	track	.

- Remove the *punctuation* marks and - in case of HTML documents - the tags (such as `<br>`). Also convert the words into *lowercase*.

every	morning	dave	runs	on	the	running	track

- Remove the *stopwords* - such as articles and auxiliary verbs - that do not contribute to the semantics.

every	morning	dave	runs	running	track

- Reduce the related words to a common stem *(stemming)* or a common form *(lemmatization)*, such as noun. This will result in a smaller vocabulary and a representation with less redundant features.

every	morning	dave	run	run	track

- Develop the vocabulary: the distinct words or groups thereof of interest that occur in the document(s).
  every, morning, dave, run, track

- Assign the values of the number of occurrences of various words in the document to the feature vector for the document.

every	morning	dave	run	track
1	1	1	2	1

Note that if there are more than one document or if we are using an external vocabulary, it is possible that some of the columns will have the value 0, indicating the absence of the word from the document.

2.2 TF-IDF Vectorization

TF-IDF stands for *term frequency-inverse document frequency*. The TF–IDF value increases proportionally with the number of times a word appears in the document and decreases with the number of documents in the corpus that contain the word [3]. It helps us to account for cases where a very common word (such as the word *movie* in movie reviews) may not hold any significant importance due to occurring in almost all documents. Also, it helps to deal with cases wherein a less common word, due to occurring in few special documents only, may carry significance for a classification model.

The TF-IDF value for a particular word for a document is defined as

$tfidf(t, d, D) = tf(t, d) * idf(t, d, D)$ where t is the term, d is the document, and D is the whole group of documents;

term frequency $tf(t, d) = 1 + ln\, f(t, d)$, where $f(t, d)$ is the frequency of the term in the document; and

inverse document frequency $idf(t, d, D) = ln \frac{|D|}{\{d \in D | t \in d\}}$.

Note that idf is high when the word occurs in only a few documents, and tf is high when word occurs frequently in a document. So the product tf-idf is high when a word, which occurs in only a select few documents, occurs frequently in the document, signalling something special about the document in question.

A feature vector can be created for each document containing the tf-idf scores of the various words.

Note that sometimes, term frequency tf is instead defined as
$tf(t, d) = f(t, d)$ / |d|, where $|d|$ is the document length, or the total words in the document.

2.3 Word Embedding

This representation aims to give words with the same meaning a similar representation. It represents words in a coordinate system where related words are placed in close proximity. It captures context of a word in a document, semantic and syntactic similarity, relation with other words.

Word2Vec is a common method to implement word embedding. It assigns a vector to each word, with similar words given vectors that are similar (have high values of similarity metrics, such as cosine similarity $cosineSimilarity = \bar{a} \bullet \bar{b} \ / \ (||\bar{a}|| \ ||\bar{b}||)$. For instance, similar words (albeit with slightly different connotations) *mother* and *mommy* may have the vectors [1, 2, 2, 2] and [1, 2, 1, 2] respectively, whereas as starkly different word, say *car*, may have the vector [3, 0, 0, 1].

2.4 Which Representation to Use?

As is the case with almost any type of data, the feature extraction technique to be employed for the text data will depend heavily on the problem we are trying to solve. For instance, Word2Vec technique easily answers the analogy completion questions of the form *a:b :: c:?,* exploiting the similarity between the vectors assigned to the words in the data.

Also the TF-IDF approach solves problems associated with frequently occurring unimportant terms and infrequently occurring important terms, but the simple Bag-of-words model might be easier to interpret for human users.

Furthermore, the choice for terms in the vocabulary to build a bag-of-words will depend on the level of performance we are trying to achieve and the computation resources available. For instance, using groups of two or more words, such as *good movie* and *full of disappointing cliches*, might capture the crux of sentiments expressed in movie reviews. At the same time, however, the vocabulary size increases dramatically, thereby increasing the computation cost.

The relatively new methods - using autoencoders, deep belief networks, and RNNs - for feature extraction may find use in certain tasks in natural language processing, such as semantic parsing, semantic role labeling, and summarization [4].

2.5 Implementation

The IMDb dataset consisting of 50,000 reviews on movies has been used to illustrate the process of feature extraction with Bag-of-words model and TF-IDF vectorization. The source code may be found in the file `TextualData.ipynb` accompanying this report. The code has been written in `python` and developed in the `Google Colab` environment equipped with `nltk` and `scikit-learn` libraries.

3. Image Data

The field of computer vision has seen a boom in recent years, thanks to both advances in convolutional neural networks and the voluminous image data generated by the internet and app users everyday. At its heart, the field tries to mimic the function of one of the most intricate sensory data perception systems of the human brain - the visual cortex. The applications are myriad, ranging from self-driving cars to forest canopy modelling using data from airborne remote sensing.

A recent trend observed in the field is to *"throw the data at a CNN"*. Deep learning techniques, however, require high computational resources. As such, some problems may be solved more quickly and efficiently with simpler ML models - like SVMs - if we use a proper feature extraction strategy. Even when using neural networks, feature extraction strategies can make a difference on the process cost.

Following are some representations and feature extraction strategies that are commonly used for image data.

3.1 Three Channel RGB Tensor

Unlike a human being, a computer can not simply “see” an image. Numbers, specifically binary numbers, form the language of computers; images, therefore, must be represented in a form intelligible by a computer. For colored images, this is achieved by creating three 2-d matrices, or a 3-d tensor, where each cell represents a *pixel* value. A pixel denotes the brightness of the corresponding color for that particular tiny box in the image. Each channel represents one of the three colors - red, green, and blue - giving the image its color via additive coloring - that is the colors are added (for example, red + blue = magenta).

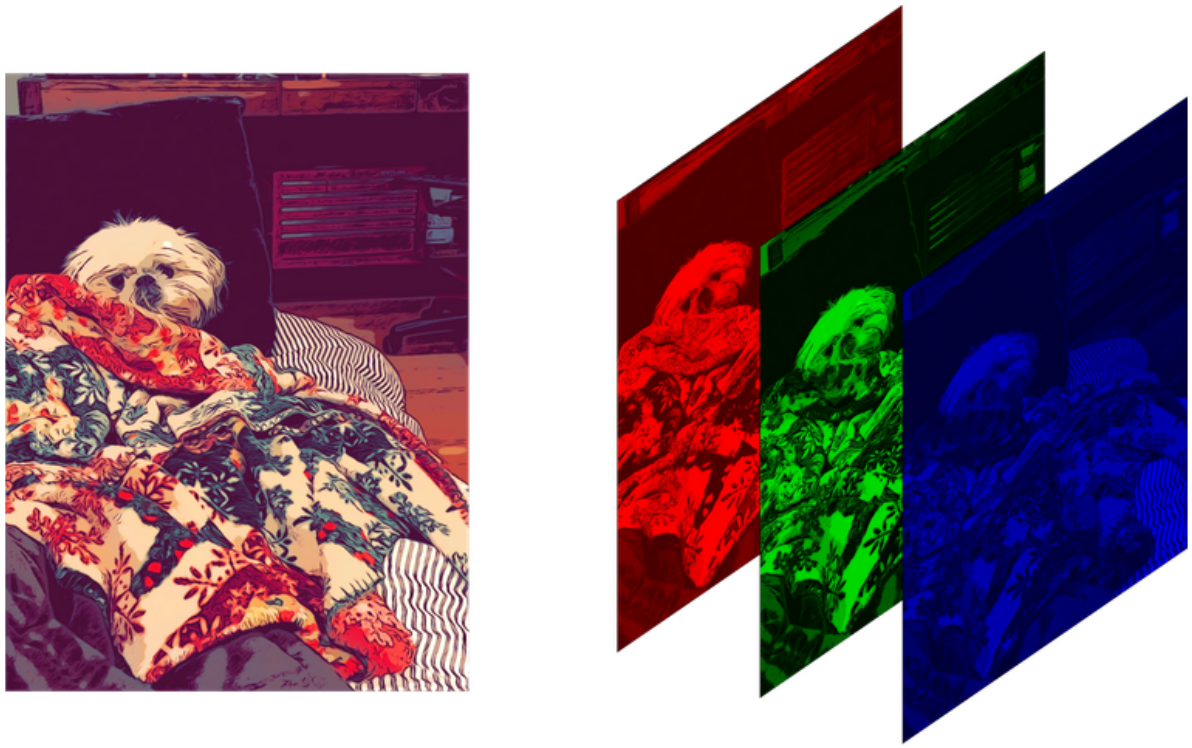

Figure 3.1: A colored image can be decomposed into 3 channels - red, green, and blue.
Image Credit: Brandon Rohrer, KDnuggets.

RGB images, however, are not the only type of images, as shown in Figure 3.2 below. If there is a single channel, where each pixel value can be either ‘high’ or ‘low’, it is a binary image. If there is one channel, but the brightness represented by a pixel can vary in a range, say 0 to 255, it becomes a grayscale image. And when

there are three channels, each with pixel values between 0 to 255, we obtain a colored image.

Figure 3.2: Types of images. (Creative Commons License)

3.2 Reduction to a Single Channel

While carrying a lot of information, working with RGB images directly can be computationally expensive. We can reduce the number of features in many cases by reducing it to a single channel, without sacrificing any relevant information. There are two common ways of achieving this reduction, given as follows.

- Generate a single channel whose each cell stores the arithmetic mean of the pixel values in the R, G, and B channels of the original image, that is $[a_{i,j}]_{mXn} = ([r_{i,j}]_{mXn} + [g_{i,j}]_{mXn} + [b_{i,j}]_{mXn}) / 3$

- Convert the RGB image to a grayscale image. Most library packages employed for this task do not just take the mean of the corresponding RGB values. Rather, they also correct perceptual luminance and apply linear approximation of gamma correction [5].

The single channel image, thus obtained, retains useful information for tasks like object recognition, segmentation, and object detection. Color dependent nuances, however, are lost.

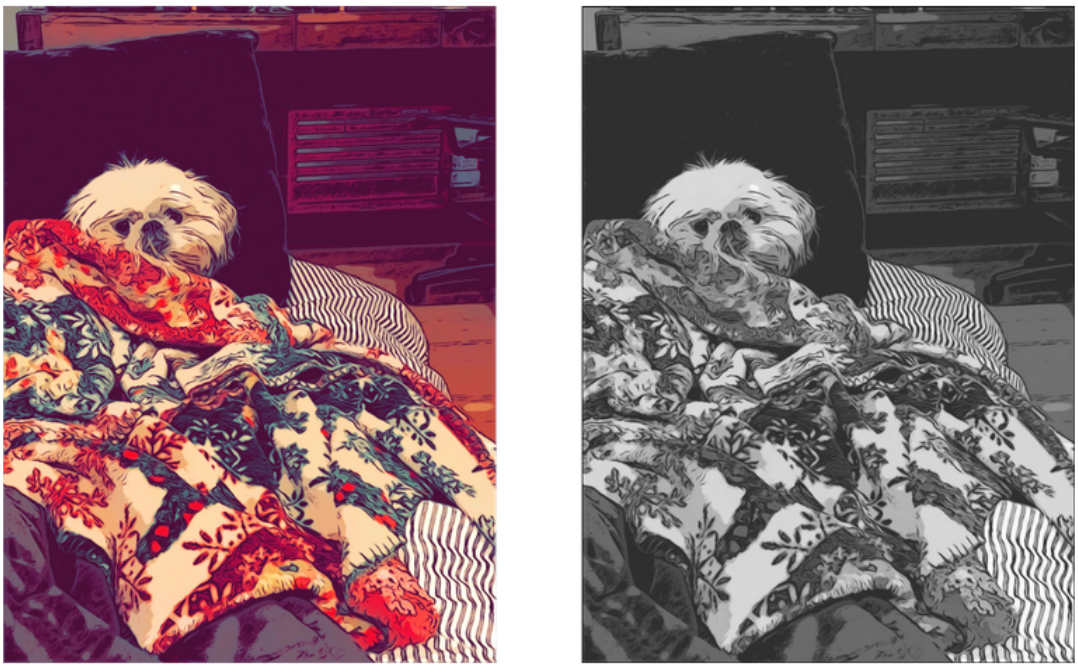

Figure 3.3: RGB image and its corresponding grayscale counterpart.
Image Credit: Brandon Rohrer, KDnuggets.

3.3 Other Formats

While the RGB format is by far the most common format for colored images, formats like BGR, HSV, and HSL still find use in certain applications. The BGR format just uses a different ordering of the three colors - blue then green then red rather than red then green then blue. It is often a default format returned by certain modules in the openCV library.

HSV format is a cylindrical format, mapping RGB to a more human-understandable format, as shown in Figure 3.4. It consists of 3-tuples of hue, saturation and value. Reading the *hue angle* on the cylinder gives the color - for instance, 240 deg is the blue color; *saturation* gives the amount of color used - 0% being grayscale and 100% being pure color; and *value* gives the color brightness - 0% being black and 100% being no black mixed.

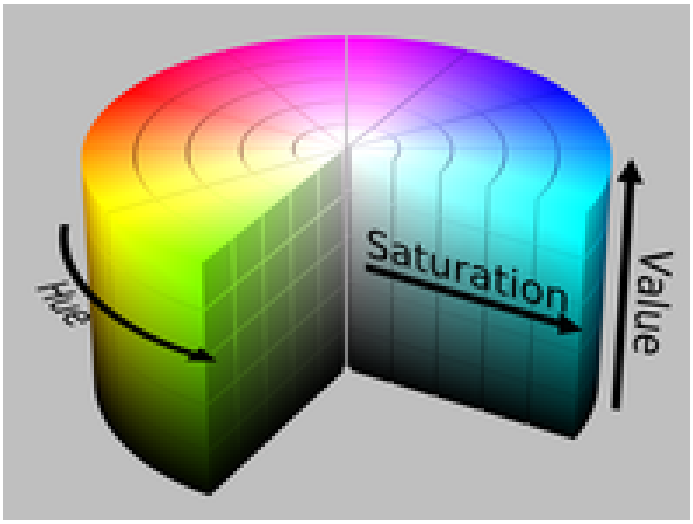

Figure 3.4: HSV Cylinder. Image Credit: HSL & HSV, Wikipedia.

3.5 Edge Extraction

The bottom-up theory of human visual perception suggests that our ability to recognize complex entities like alphabets and even faces of people is based on simpler capacities to recognize and correctly combine lower-level features of objects like lines, edges, corners, and angles [6].

In the same spirit, edges can be used as features for further processing by a model. To identify edges, we may look for sharp changes in pixel values. This can be implemented in practice by using kernels such as the Prewitt and the Sobel kernels. Figure 3.5 below shows the result of edge extraction with the Prewitt kernel.

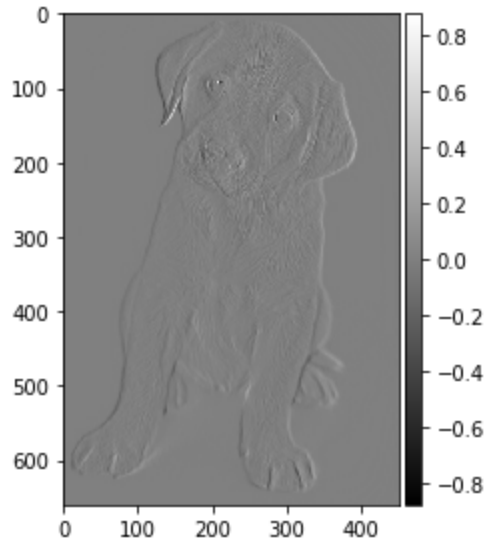

Figure 3.5: Edges for an image of a dog.
Image Credit: AnalyticsVidhya.

3.6 Which Representation to Use?

Again the representation to use will depend on the problem and the computation resources available. For instance, if there is no dearth of computational resources, one strategy could be to use a convolutional neural network on a 3-channel RGB tensor. The bottom layers in a CNN collect the low-level features, while high-level layers collect and learn features with more abstract information, useful for classification tasks [7].

Suppose, we are dealing with traffic signals, such as those used by a traffic light. In such a situation, even though using grayscale images can reduce the number of features, the information about color we lose is important, and thus we stick to RGB images.

Due to various corrections applied - such as the correction for perceptual luminance - using a standard conversion tool for obtaining grayscale image should be preferred to simply taking the mean of the three channels.

3.7 Implementation

The code file `ImageData.ipynb` illustrates the various image data representations using an image of Himalayan ranges visible from Shimla I photographed. RGB, BGR, HSV, and Grayscale formats have been obtained for the image using tools available via `openCV`. Further, edge extraction has been done for vertical and horizontal edges using the Pewitt kernels and the Canny Algorithm.

4. Video Data

With increasing popularity of video upload and sharing platforms such as YouTube, increasing acceptance of online lectures, and increasing storage capacities available via increasingly cheaper semiconductor technologies and cloud to the masses, video data has exploded in recent years, warranting skilled data scientists to handle and analyze it for useful patterns. For instance, analysis of famous speeches can immensely help to a compilation of successful oratory strategies.

While video data can seem very daunting at first, it can easily be understood in terms of its constituent frames, which are just images. Videos typically 20 to 30 images (frames) per second (fps). 20 fps roughly represents the threshold at which humans perceive a sequence of frames as fluid motion [8]. The video file can be sampled periodically to obtain an image frame. These image frames can then be analyzed for various features. For instance, the screen time of an actor in an image can be estimated by recognizing the frames in which the actor appears.

Figure 4.1: Individual Image Frames Extracted from a Video
Image Credit: Nerd for Tech, Medium.

The video features extracted can be divided into structural features and content features. Structural features include lower-level features, such as colors, scene cuts, and duration, which can be typically altered while editing the footage. These features determine *how* the video is presented. Content features determine *what* is presented and include visual cues - such as faces and objects - and automated predictions of viewer perceptions - such as emotions and story [8].

Feature extraction is performed at two levels for videos - on the level of frames and on the level of overall video. Features like video duration only make sense for the overall video. In contrast, features like human faces are detected on a frame level and can be aggregated (as shown in Figure 4.2 below) at a video level, say to calculate the share of frames in which human faces appear.

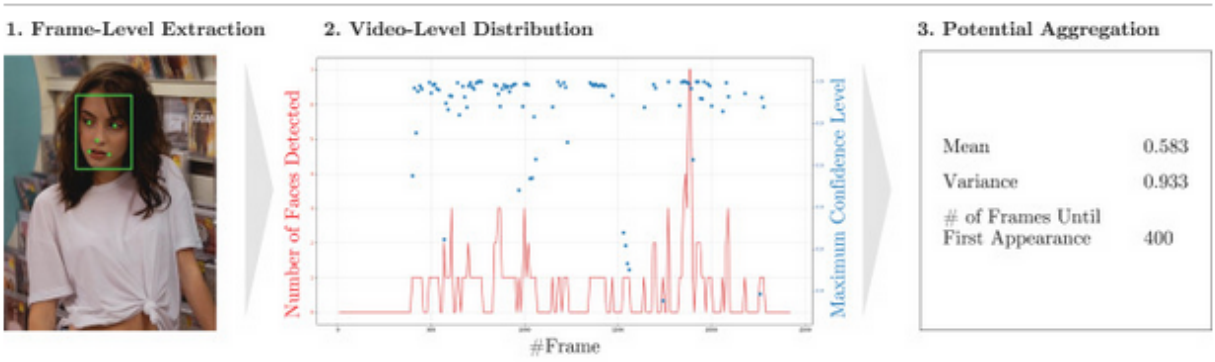

Figure 4.2: Frame level features are aggregated for the overall video.
Image Credit: *See Reference 8.*

Furthermore, as shown in Figure 4.3 below, certain features - such as video duration and resolution of frames - can be extracted using traditional techniques, while others can only be obtained using machine learning - for example, emotions and object types in the video.

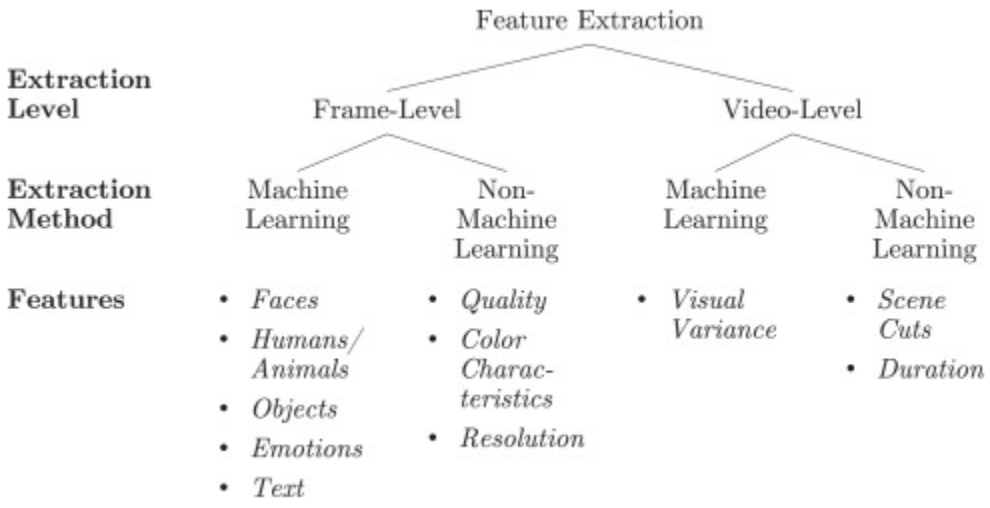

Figure 4.3: Some commonly used features for videos with extraction levels and techniques.
Image Credit: *See Reference 8.*

So how to proceed and which representation to use? At its heart, most video analysis tasks revolve around the analysis of component images. So the most versatile representation of a video is a tensor storing a sequence of frames, which in turn, being image, are tensors. Nevertheless, if all we'll be dealing with is, say,

the text in the video, we may as well extract it first and use it for our models in laters stages, rather than proceeding with a collection of frames.

Specific applications might demand special features. For instance, global motion and object trajectory are described with parametric motion and motion trajectory descriptors to study camera motion and motion activity in video sequences [9].

4.1 Implementation

The code file `VideoData.ipynb` uses a short video from the *Tom & Jerry Tales* to illustrate the nature of video as a collection of frames. It displays the resulting tensor for the video, extracts one sample frame to illustrate its nature as an image, and calculates the structural feature video duration.

5. Audio Data

Without proper strategies and tools to handle audio data, voice assistants like *Alexa* and *Cortana* would not have been possible; neither would have been possible the analysis of trending music playlists to find out what is selling among the music listeners.

Much like text and videos, audio data in its raw form is far from understandable from an analysis point of view. The following sections discuss several strategies to represent and extract features from audio data.

5.1 Time Series

An audio signal is a three-dimensional signal with three components - time, frequency, and amplitude - as illustrated in Figure 5.1 below.

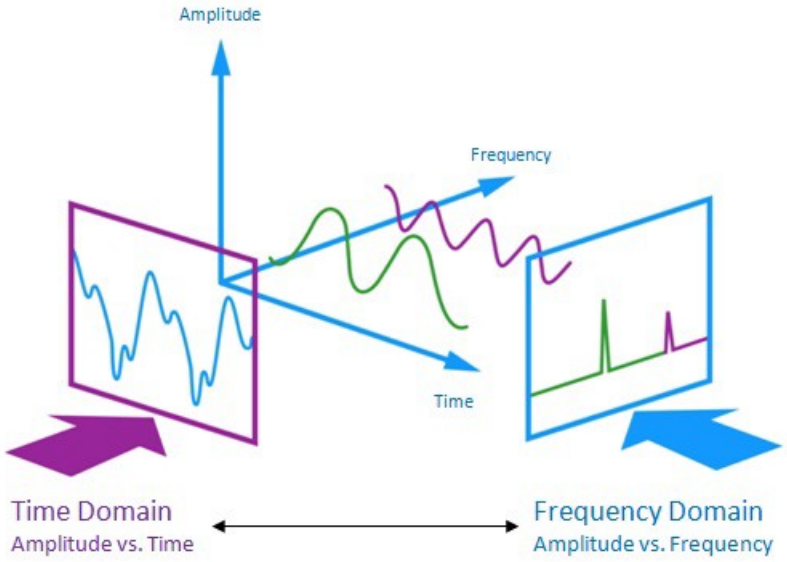

Figure 5.1: Audio signal - time, frequency, and amplitude.
Image Credit: Sanket Joshi, TowardsDataScience.

We humans perceive the amplitude of sound as loudness and the frequency as pitch. It is possible to represent sound as a time series of amplitudes. The frequency(ies) can be inferred by plotting the time series - high frequency sounds will have a large number of crests and troughs in a given time period as compared to a low frequency one. The time series can be plotted as a waveplot as shown Figure 5.2 below.

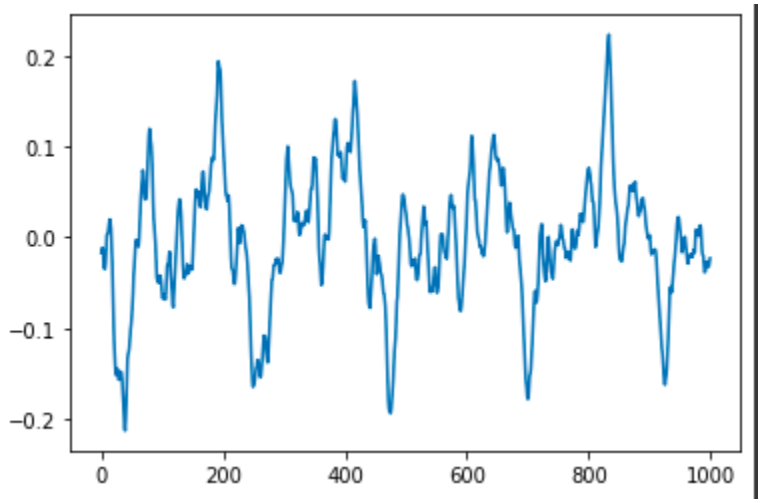

Figure 5.2: A Zoomed-in waveplot for *Darth Vader's Imperial March* from *Star Wars.*
See AudioData.ipynb.

Rendering the audio data into a time series offers several advantages, the foremost being the purely numerical nature of the data. The traditional time series analysis methods from statistics - including time series decomposition to isolate the periodic, trend, and noise components - can be used conveniently. Furthermore, since the data has been converted into an array, efficient processing tools can be used to deal with it.

5.2 Spectrogram

A spectrogram is a visual representation of the spectrum of frequencies in sound or other signals as they vary with time [10]. It shows how the energy - measured by amplitude ($intensity\ \alpha\ amplitude^2$) - is distributed for different frequencies for a time interval.

It is displayed as a two-dimensional graph plotting frequency against time, with the third dimension - amplitude - shown by using a continuum of colors. Generally, blue and red shades represent low and high amplitudes respectively [10]. The frequency-axis can be arithmetic or logarithmic, whichever is found convenient.

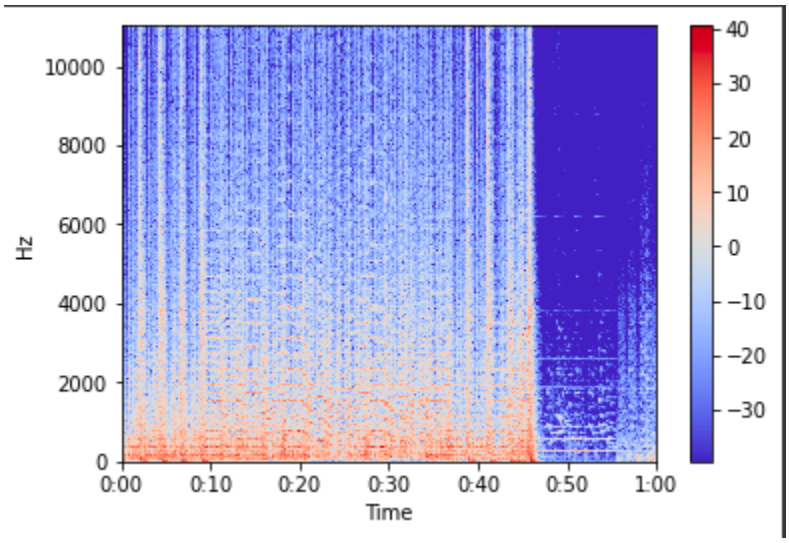

Figure 5.3: Spectrogram for *Darth Vader's Imperial March* from *Star Wars.*
See AudioData.ipynb.

5.3 Zero Crossings

Zero crossings are the points where the waveplot crosses the x-axis, that is, the signed sound amplitude becomes zero (crosses the time-axis) and changes signs. Generally, we are interested in the zero crossing rate - the rate at which the sound signal changes from positive to negative and vice versa. It can be used in genre detection for music, for percussion instruments like drums used in rock music typically give high zero crossing rate.

5.4 Spectral Centroids

The spectral centroid is a measure used in digital signal processing to characterise a spectrum. By measuring the weighted mean of frequencies present in the signal, it indicates where the center of mass of the spectrum is located [11].

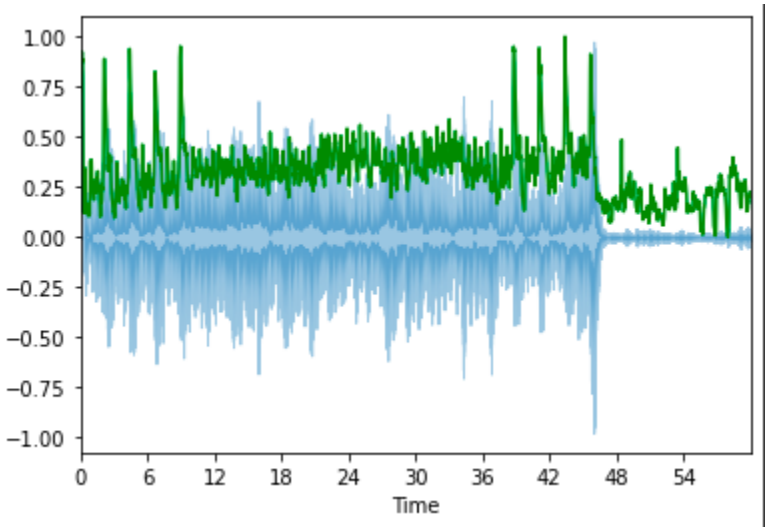

Figure 5.4: A plot of spectral centroids (green) for *Darth Vader's Imperial March* from *Star Wars*. *See AudioData.ipynb.*

A higher value of spectral centroid indicates more energy of the signal being concentrated within higher frequencies. If the energies are distributed uniformly, the spectral centroid is close to the center.

5.5 MFCC

MFCCs stand for Mel-frequency cepstral coefficients. They are a small set of cepstral-based features which can concisely describe the overall shape of the spectral envelope. Here, *cepstrum* is a nonlinear transformation of the spectrum.

The Mel-frequency cepstrum portrays the short-term power spectrum of a sound, based on a linear cosine transform of a log range spectrum on a nonlinear Mel scale of frequency [12]. The cepstrum-based features can be extracted using the process given in Figure 5.5 below.

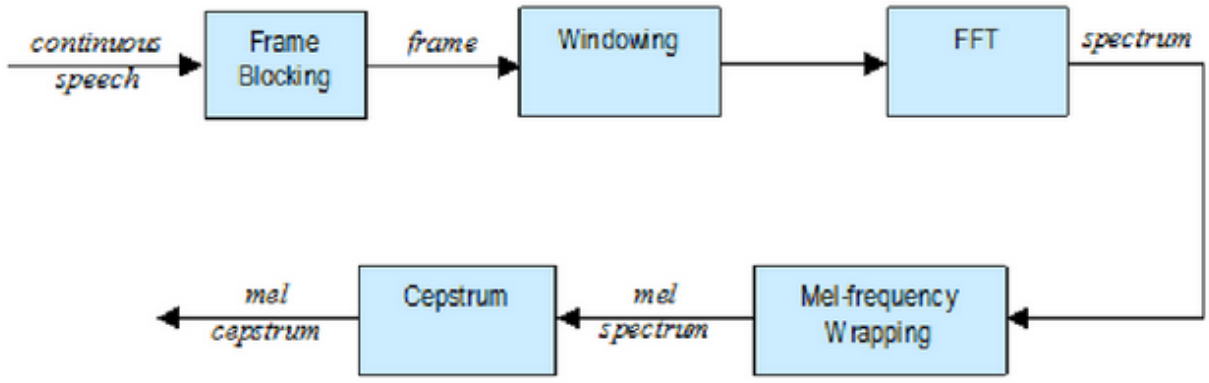

Figure 5.5: An outline of the process to extract MFCCs.
Image Credit: Sanket Joshi, TowardsDataScience.

5.6 Which Representation to Use?

MFCC is a very concise representation, often using just 20 or 13 coefficients instead of about 32 to 64 bands in the Mel spectrogram; hence MFCC is often preferred over a spectrogram. Deep Learning techniques consider unstructured audio representations, such as the spectrogram or MFCCs, extracting the patterns on their own [12].

A time series representation is the most basic representation for audio data, and, as previously mentioned, is suitable for analysis with traditional statistical techniques.

Then there is the question of what level of features we are interested in. High level features such as instrumentation, chords, melody, mood, and genre are best detected by deep learning models and can be used for further analysis in subsequent models. Mid level features - such as MFCCs and fluctuation patterns - are extracted as discussed in previous sections and are often viewed as the aggregation of low level features [12]. Low level features like zero crossing and amplitude, might not be very useful on their own directly but form the backbone of the audio processing.

5.7 Implementation

The code file `AudioData.ipynb` illustrates the feature extraction for audio data - time series, zero crossings, spectral centroids, spectrogram, and MFCCs. The audio file used is a `.wav` music with *Darth Vader's Imperial March* theme from *Star Wars*. The audio processing tools supplied by `librosa` and `scikit-learn` have been used.

6. Tensors

A tensor, to put it simply, is a container which can store data in N dimensions. Scalars, vectors, and matrices are all special instances of tensors in zero, one, and two dimensions respectively. The rank or dimensions of a tensor is the number of directions required to represent the data.

Computer scientists often know tensors with more familiarity as n-dimensional arrays. Indeed, Table 6.1 shows the correspondence between the commonly used terms in computer science and mathematics [13].

Mathematics	Computer Science	Tensor Dimensions
Scalar	Number	0
Vector	1-d array	1
Matrix	2-d array	2
Tensor	n-d array	n

Table 6.1: Commonly used related terms in Mathematics and Computer Science

Tensors are more than simply a data container - they also include descriptions of the valid linear transformations between tensors, such as the cross product and the dot product. Thus, we should think of tensors as objects in an object-oriented sense, as opposed to simply as a data structure.

6.1 Tensor Processing Units

Google provides Tensor Processing Units (TPUs), which comprise Application Specific Integrated Circuits (ASICs) specially designed for tensor calculations. It is a coprocessor that cannot execute codes on its own but supports the CPU to increase the computation speed [14]. They are designed to immensely accelerate linear algebra calculations, bringing the time for model training from weeks to hours. They are built to support all standard functionalities provided by Tensorflow.

Ideal situations, according to the Google cloud documentation page, for TPU utilization include models dominated by matrix computations, models which may require a training time of weeks and months, and very large models with very large effective batch sizes. They are not suitable if custom Tensorflow operations - built in C++ - occur in the main training loop. They do not support double precision floating point calculations. Also, they are not suitable for elementwise tensor operations and sparse memory accesses.

Several important services provided by Google, including Google Translate, Google Photos, Gmail, and Google Assistant are powered by TPUs [14].

6.2 Tensorflow

`Tensorflow` is an open-source `Python` library for high-performance tensor processing and machine learning framework provided by Google. It offers multiple levels of abstraction - the low-level `Tensorflow Core` for increased flexibility, the high-level `Keras` API for easy model building, and the Distribution Strategy API for distributed training on different hardware configurations.

It is supplemented by `Tensorflow.js` for deploying models with JavaScript, Tensorflow Lite for mobile and embedded systems - such as Android, iOS, and Raspberry Pi - and Tensorflow Extended (`TFX`) to deploy a production-ready ML pipeline for training and inference.

Tensors form the basic unit of data in `Tensorflow`. As such, Tensorflow allows complete integration with `NumPy`, one of the most popular libraries for linear algebra and array-handling in Python.

Several well-known tech companies, including AirBnB, Google, Intel, PayPal, Qualcomm, and Twitter use Tensorflow for a wide gamut of tasks ranging from AI natural language processing to network anomaly detection.

References

[1] Tan, P.-N., Steinbach, M.,, Kumar, V. (2005). Introduction to Data Mining. Addison Wesley. ISBN: 0321321367.

[2] "Feature extraction," Wikipedia, 20-Oct-2021. [Online]. Available: https://en.wikipedia.org/wiki/Feature_extraction. [Accessed: 08-Nov-2021].

[3] "Feature extraction techniques - NLP," *GeeksforGeeks*, 06-Mar-2020. [Online]. Available: https://www.geeksforgeeks.org/feature-extraction-techniques-nlp/. [Accessed: 08-Nov-2021].

[4] Liang, H., Sun, X., Sun, Y. *et al.* Text feature extraction based on deep learning: a review. *J Wireless Com Network* **2017,** 211 (2017). https://doi.org/10.1186/s13638-017-0993-1

[5] "How to convert an RGB image to grayscale," *KDnuggets*. [Online]. Available: https://www.kdnuggets.com/2019/12/convert-rgb-image-grayscale.html. [Accessed: 08-Nov-2021].

[6] Baron, R.A.(2016). Psychology. Pearson. ISBN: 9789332558540.

[7] A. A. Alani, G. Cosma, A. Taherkhani and T. M. McGinnity, "Hand gesture recognition using an adapted convolutional neural network with data augmentation," 2018 4th International Conference on Information Management (ICIM), 2018, pp. 5-12, doi: 10.1109/INFOMAN.2018.8392660.

[8] J. Schwenzow, J. Hartmann, A. Schikowsky, and M. Heitmann, "Understanding videos at scale: How to extract insights for business research," *Journal of Business Research*, vol. 123, pp. 367–379, 2021.

[9] Chuan Wu, Yuwen He, Li Zhaoe, and Yuzhuo Zhong. "Motion feature extraction scheme for content-based video retrieval", Proc. SPIE 4676, Storage and Retrieval for Media Databases 2002, (19 December 2001); https://doi.org/10.1117/12.451100.

[10] "What is a spectrogram?," *Pacific Northwest Seismic Network*. [Online]. Available: https://pnsn.org/spectrograms/what-is-a-spectrogram. [Accessed: 08-Nov-2021].

[11] "Spectral centroid," *Wikipedia*, 06-Dec-2020. [Online]. Available: https://en.wikipedia.org/wiki/Spectral_centroid. [Accessed: 08-Nov-2021].

[12] Pdmn Arvind, Mahanta S.K. "Audio Feature Extraction," *Devopedia*, 23-May-2021. [Online]. Available: https://devopedia.org/audio-feature-extraction. [Accessed: 08-Nov-2021].

[13] "Tensors explained - data structures of Deep Learning," *YouTube*, 11-Sep-2018. [Online]. Available: https://www.youtube.com/watch?v=Csa5R12jYRg&t=65s. [Accessed: 08-Nov-2021].

[14] Rose S.L., Kumar L.A., Renuka K. (2019) Deep Learning Using Python. Wiley. ISBN: 9788126579914.

Datasets Used

Text Data: IMdB dataset containing 50,000 movie reviews has been used. The review labels have been discarded as the problem tackled is concerned with the text itself to develop a bag-of-words or TF-IDF vector. The dataset is linked to the colab environment directly from https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz.

Image Data: The image used to illustrate the process of numerical representation and edge detection comes from one of the photos I photographed - DSC07921.JPG, which was uploaded to the colab environment from the local machine.

Video Data: The .mp4 is a short video from *Tom and Jerry Tales* obtained from Kaggle available as "Train Tom and Jerry.mp4": https://www.kaggle.com/yashagrawal300/tom-and-jerry. The video was uploaded to the colab environment from the local machine.

Audio Data: The audio file used - ImperialMarch60.wav - containing *Darth Vader's Imperial March* theme from *Star Wars* is available at https://www2.cs.uic.edu/~i101/SoundFiles/. The file was uploaded to the colab environment from the local machine.