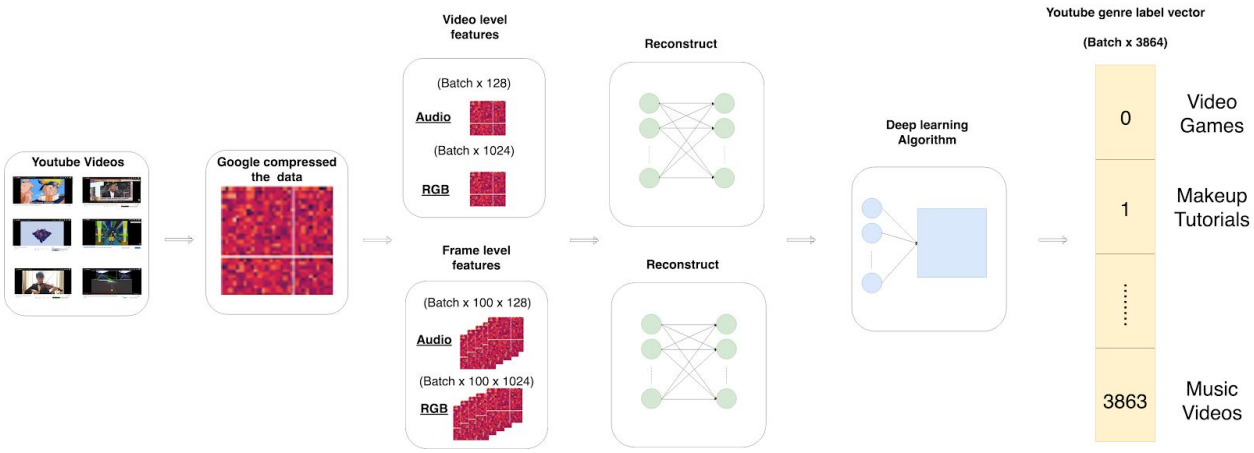
Portfolio: https://github.com/rchavezj/Label_YT_Videos



Goal for my thesis: Teach an AI to label a genre (E.g., Makeup, Games, Art & Entertainment, etc.) on a youtube video using a series of deep learning algorithms and compare each one to understand their strengths and weaknesses, and scalability properties. This algorithm can potentially automate repetitive labor organizing youtube videos with similar content in a recommendation search engine and classify copyright material. I will be using Google's yt8m dataset [1] initially 0.5 petabytes of data, compressed to 1.5 terabytes for the research dataset. There will be two types of features: video-level (2-dimensions) and frame-level (3-dimensions) data. I coded eight deep learning models: 4 in Keras and the same models in PyTorch to compare not only models but frameworks against each other. More details provided under **Comparing models & frameworks** section.

The pipeline would be:
(1) Youtube videos uploaded from users
(2) Google compressing the content
(3) Reconstructing the data and features for a machine learning pipeline
(4) Output a vector from a model with one element containing the highest probability for the predicted genre label.
(5) Perform a comparison of both models and frameworks

Contribution: Based on interactions and learning from other Kaggle competitors, I applied a best-practices a set of feature engineering and visualizations to the dataset. I contributed the following: feature engineering, comparing models, comparing frameworks (Keras & PyTorch), integration, optimization, implementing distributed

training in PyTorch, and evaluating performance (lost, accuracy, CPU/GPU/Hardware usage).

Comparing models & frameworks: I coded eight deep learning models: 4 in Keras and the same models in PyTorch. I chose to compare not only models but frameworks as well based on research I did on algorithms crashing in production [49]. Tensorflow (Keras) has the largest user base and most traction currently. I want to compare it with PyTorch which is an imperative framework that performs computation as you type it. Tensorflow (Keras) uses symbolic programming: only computing your code at the end of each graph session [38]. Tensorflow is evolving to become more "PyTorch" like with eager execution [51], however that's still in alpha and didn't exist at the time time my project started. The value of comparing frameworks was also recognized by google engineers as they were working along a similar path in parallel to my work [51]. More information about the difference between each framework and their performance is written in my paper. My experiments reveal surprising results.

Integration: We are dealing with compressed data containing two important features we need to aggregate: video-level and frame-level content. We need to integrate three different algorithms into the pipeline:

(1) Reconstruct the compressed content [1] with initialized parameters. The number of features is smaller than the number of class labels (output genre labels) which is why I had to reconstruct the data. Add more dimensions into the input data. More details are available in the paper under the **Feature Engineering** section. The output of the reconstructed features in step (1) are separately sent into their respective models in (2) and (3).

(2) Compute sequential data (frame-level) for each video gathered from yt8m. Each video is at least 100 second long to be utilized within the dataset. We can use temporal models (Recurrent Neural Net).

(3) Instead of 100 second per video, Google created video-level features extracting a task-independent fixed-length vector per frame. In other words video-level features have one less dimension from compressing frame-level sequential content. We can train this data using classifiers like logistic regression or any non-linear design.

Both models **(2)** and **(3)** models contain a softmax approximator model to determine the label of a genre from a youtube video. You can develop an **algorithm (section below)** with both models separately or combined (concatenate).

Methods: Google compressed their yt8m data using Principle Component Analysis (PCA) so the dimension of our input features is 100 for RGB content and 1028 for audio, both of which are less than 3864 which is the vector size of the class labels for the video genre (E.g., Games, Art & Entertainment, etc.). We need to come up with an unsupervised learning technique to reconstruct the data with additional weight parameters to make sure the compressed input content could fit the output dimension. After the number of input features for both RGB and audio has increased, we can now send both features into their respected deep learning algorithm. After reconstructing the data we are still given a challenging problem to select an algorithm that can scale to large amounts of data. Fortunately, deep learning algorithms have been getting remarkable results in the AI community for scalability [2]. For video-level we can use logistic regression or a deep neural net to compute compressed pixels. As for frame-level, sequential features (100 second videos) can also be scaled through temporal algorithms with: RNN's, LSTM, Self Attention, GRU, and Markov chains. Each has their own limitations computing long sequences of video frames.

Feature engineering: I reconstructed the data using an autoencoder with additional weight parameters to make sure the compressed input content could fit a large genre label. The output vector mapping the relationship between each genre (E.g., Games, Art & Entertainment, etc.) to their respected youtube video. Once the data has been reconstructed with additional parameters to match the same dimension as the output, we can then choose a series of deep learning algorithms, which is covered in the next section.

Algorithms: Below are four algorithms I coded to aggregate my data. Later I compare each framework (PyTorch & Keras) with the same models to see which one was most efficient for research and or production.

(1) Fully Connected Net: Aggregating compressed spatial features from youtube videos
(2) Bidirectional LSTM: We are using a Bidirectional LSTM to aggregate sequential content of video frames. Each youtube video has been cut down to 100 seconds of frames. Any video less than 100 seconds is not part of the dataset to balance the distribution. For example, if there was a 10 second youtube video of nascar-racing inside the dataset while another video with 100 seconds of a video

game of cars, the algorithm would likely have a bias prediction labeling a nascar-race as a video game.

(3) Stream-LSTM is similar to Bidirectional except we have the approximators for audio and rgb separately sent into their own fully connected nets. The approximators are later combined into a concatenate function to find the average. For example if the approximator for one genre label in a rgb vector was 79% while the audio was 99%, concatenating both would be 89%.

(4) Fully Connected Net (Video-level) concatenated with a Stream-LSTM (frame-level). It's a combination of Algorithm (1) & (3) softmax approximation using concatenation.