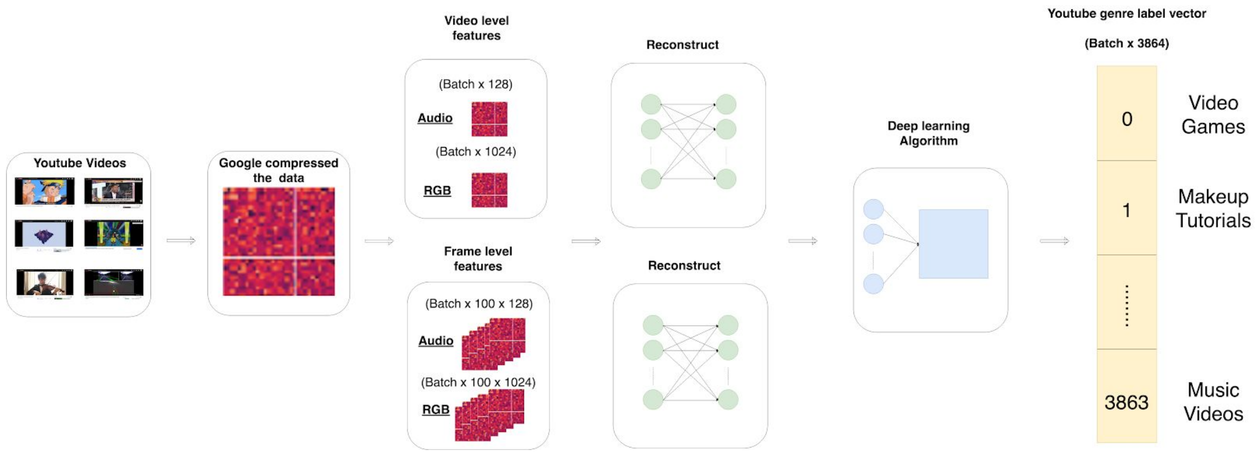
Portfolio: https://github.com/rchavezj/Label_YT_Videos



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 see which among is most feasible for research and scaled into production. 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 petabyte big, compressed down to 1.5 terabyte due to download limitations for researchers.

The pipeline would be:
(1) Youtube videos uploaded from users
(2) Google compressing the content
(3) Researchers 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) See which model from their respected framework is most feasible

Contribution: After the data was processed and visualized (histogram & similarity matrix) with the help of Kaggle peers [46], I contributed the following: compute power, compare models, feature engineering, integration, algorithms, optimization, CPU/GPU parallel distribution (PyTorch), and evaluation 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. My attention "*comparing two frameworks"* is from reading a report on algorithms crashing in production [49]. Since most machine/deep learning models are coded in Tensorflow (Keras) [37, 50], I want to see the benchmarks comparing it with PyTorch: 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]. More info behind the difference between each framework and their performance are written on my paper and check out my experiments: I ended up with surprising results.

Integration: There needs to be three algorithms integrated in order to aggregate yt8m dataset:

(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 info under Feature Engineering section.
(2) An autoencoder would help (more info under methods). Output the reconstructed features into their respected models in step (2) and (3).
(3) Compute sequential data for each video gathered from yt8m. Each video at least 100 second long is going to be utilized within the dataset. We can use temporal models (Recurrent neural Net).
(4) Find patterns within compressed pixels [1]: non-linear model.
(5) Concatenate the output of step (2) and (3) algorithms into a softmax approximator.

Methods: Google compressed their yt8m data using a Principle Component Analysis (PCA) so the dimension our input features, 100 for rgb content and 1028 for audio, are both less than 3864 which is the vector size video label genre (E.g., Games, Art & Entertainment, etc.) (3864). Thergo I need to propose a method to add more dimensions on the input data in order to fit the output genre-label vector. The first pipeline will need to be a unsupervised learning method to increase the number of features in order to match the number of class labels. After the number of 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 a large amount of data: not unless we use popular deep learning algorithms [2] that have been getting a lot of attraction in the past decade. 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 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 in the next section.

Algorithms: Below are four algorithms I coded to aggregate my data. Later compare each algorithm to see which one was most efficient for research and or production.

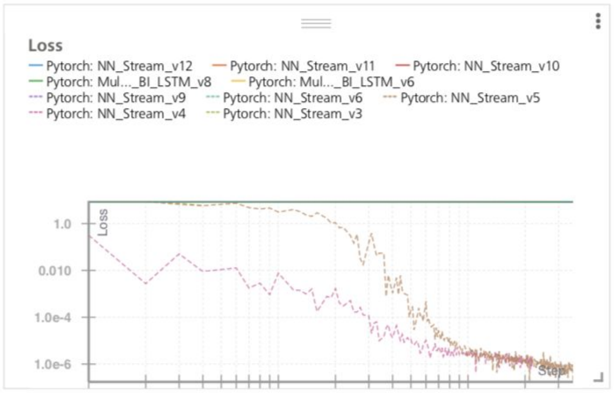
(1) Neural net: Aggregating compressed spatial features from youtube videos

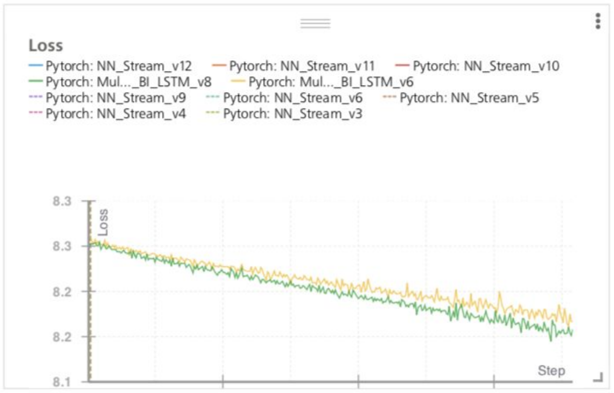
(2) Bidirectional LSTM: We are using a Bidirectional LSTM to aggregate sequential content of video frames. Each youtube video has been cut down to 100 seconds of frames. Any video less than 100 seconds is not part of the dataset to balance the distribution. For example, if there was a 10 second youtube video of nascar-racing inside the dataset while another video with 100 seconds of a video game of cars, the algorithm will likely have a bias prediction labeling a nascar-race as a video game. Both videos have cars inside a video so any efficient algorithm would favor the majority of sequential content. Longer videos have more content to provide a higher distribution to influence the algorithms decision. Nascar-racing would be the minority class and the video games of cars would be the majority.

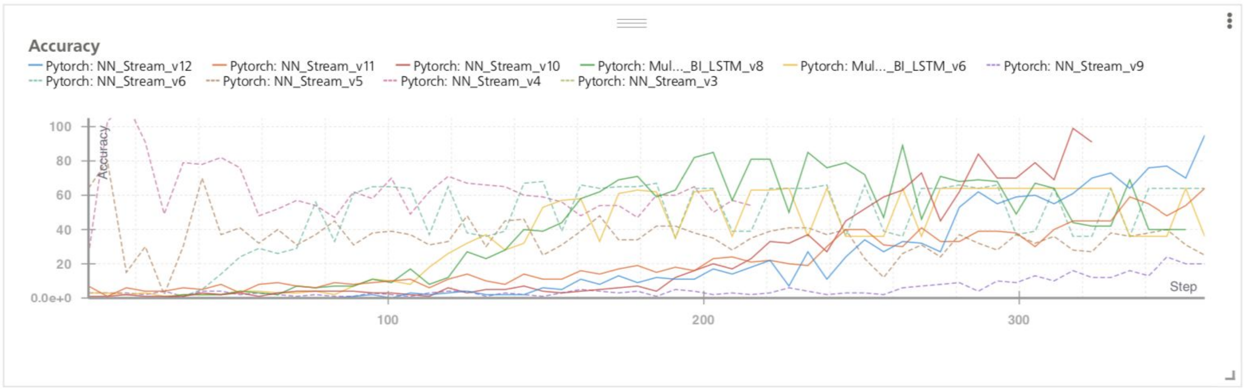
(3) Stream-LSTM is similar to Bidirectional except we have the pipeline for audio and RGB. No concatenation until we reached the sigmoid approximation which is the final output.

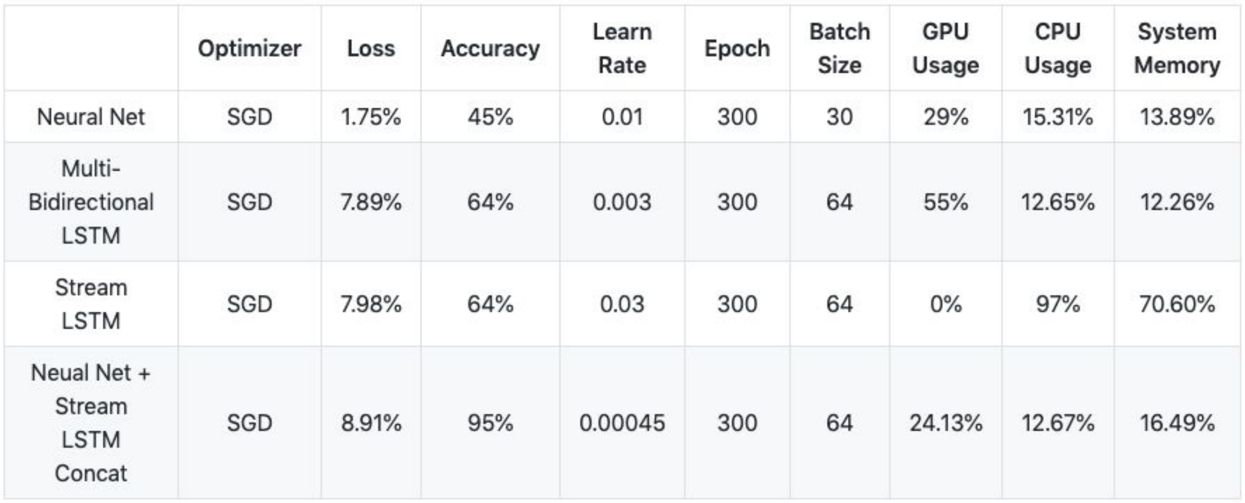
(4) A neural net (Video-level) concatenated with a Stream-LSTM (frame-level). It's a combination of Algorithm (1) & (3).

PyTorch Report:



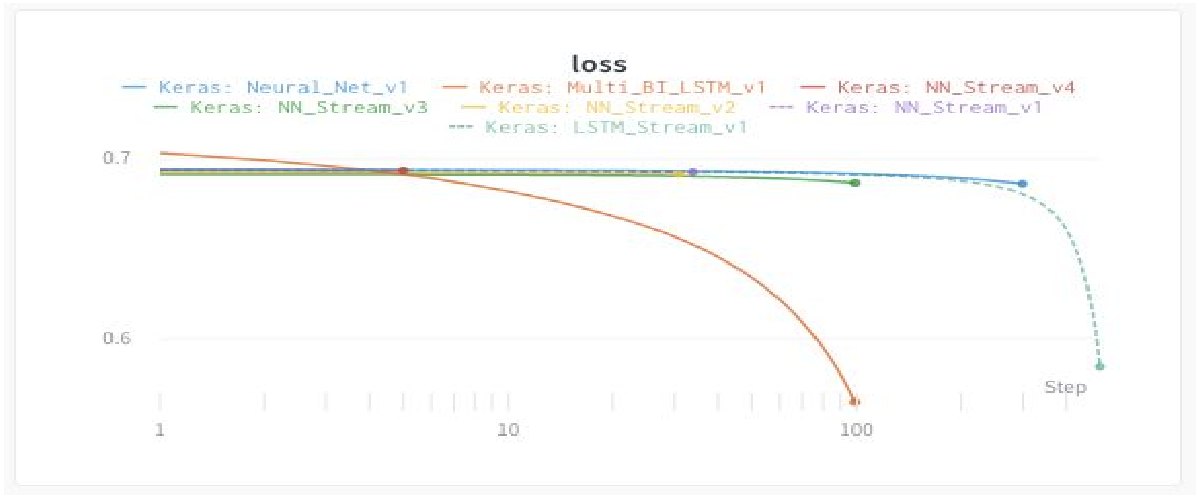


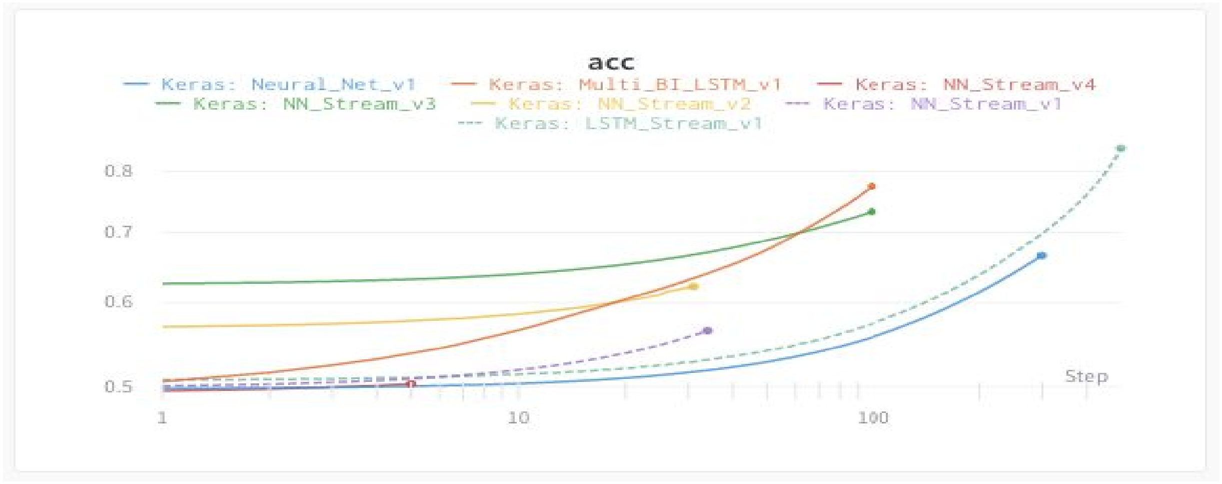




	Optimizer	Loss	Accuracy	Learn Rate	Epoch	Batch Size	GPU Usage	CPU Usage	System Memory
Neural Net	SGD	1.75%	45%	0.01	300	30	29%	15.31%	13.89%
Multi-Bidirectional LSTM	SGD	7.89%	64%	0.003	300	64	55%	12.65%	12.26%
Stream LSTM	SGD	7.98%	64%	0.03	300	64	0%	97%	70.60%
Neual Net + Stream LSTM Concat	SGD	8.91%	95%	0.00045	300	64	24.13%	12.67%	16.49%

Keras Report:



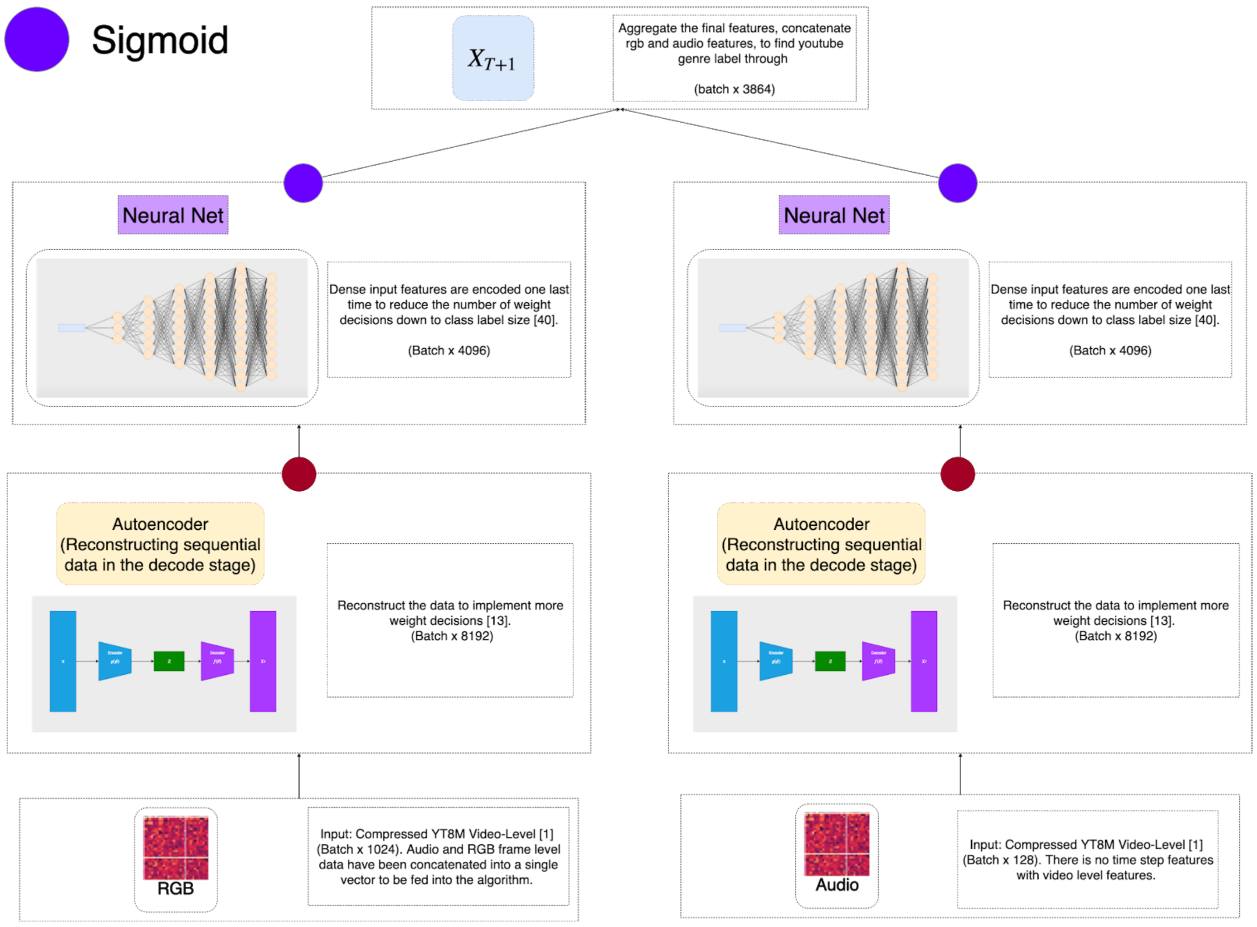


	Opt	Loss	Accuracy	Epoch	Batch Size	GPU Usage	CPU Usage	System Memory
Neural Net	Adam	68.5%	66%	300	84	0.13%	64.97%	14.37%
Multi-Bidirectional LSTM	Adam	56%	77%	300	64	0%	99.17%	31.19%
Stream LSTM	SGD	58.6%	84%	500	64	0%	93.37%	26.07%
Neual Net + Stream LSTM Concat	Adam	68.6%	73%	100	20	0%	96.84%	28.45%

Neural Net

● Relu

● Sigmoid


X_{T+1}
Neural Net
Neural Net
Autoencoder (Reconstructing sequential data in the decode stage)
Autoencoder (Reconstructing sequential data in the decode stage)
RGB
Audio

Aggregate the final features, concatenate rgb and audio features, to find youtube genre label through

(batch x 3864)

Dense input features are encoded one last time to reduce the number of weight decisions down to class label size [40].

(Batch x 4096)

Dense input features are encoded one last time to reduce the number of weight decisions down to class label size [40].

(Batch x 4096)

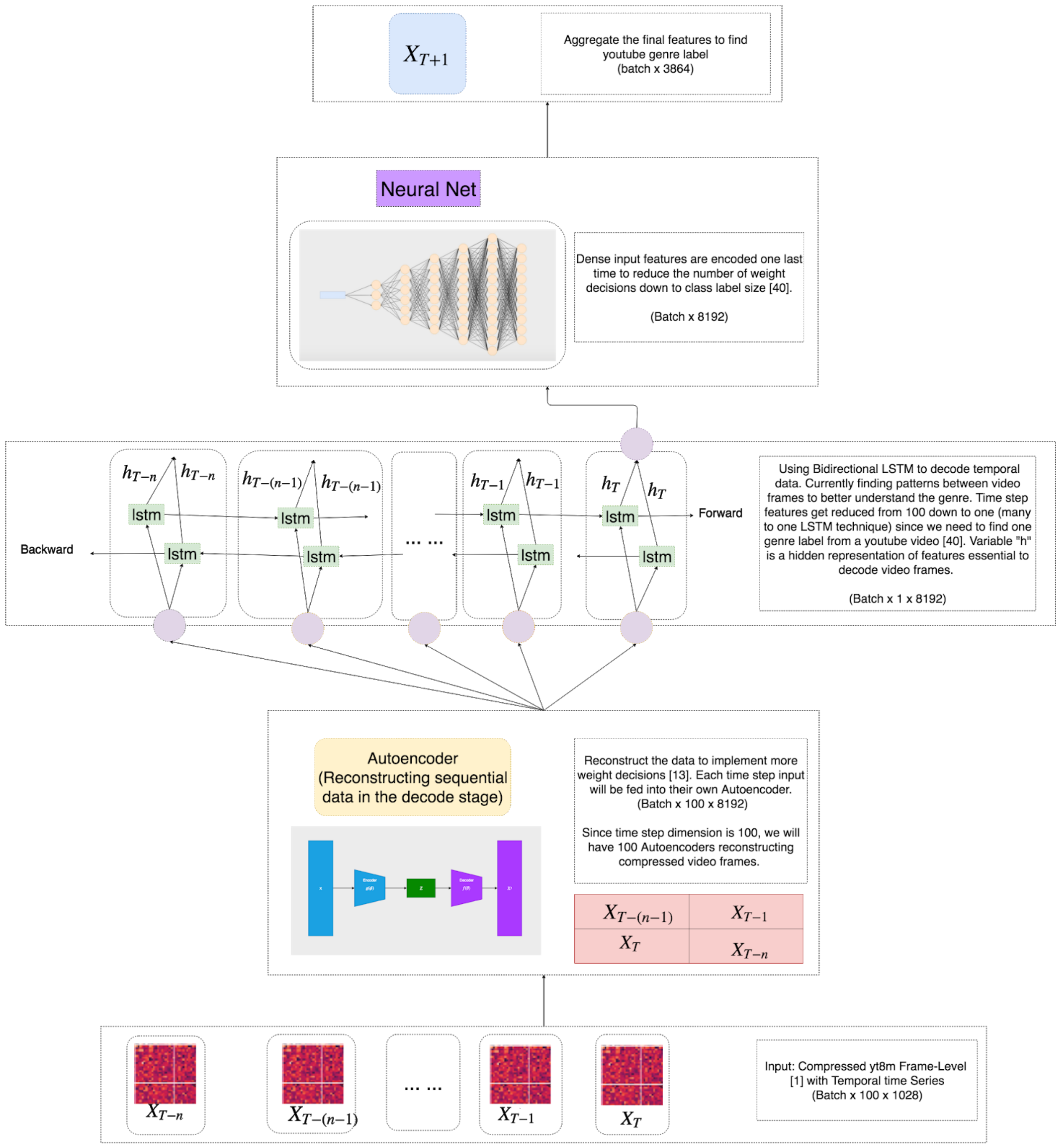
Reconstruct the data to implement more weight decisions [13].
(Batch x 8192)

Reconstruct the data to implement more weight decisions [13].
(Batch x 8192)

Input: Compressed YT8M Video-Level [1] (Batch x 1024). Audio and RGB frame level data have been concatenated into a single vector to be fed into the algorithm.

Input: Compressed YT8M Video-Level [1] (Batch x 128). There is no time step features with video level features.

Bidirectional LSTM



Aggregate the final features to find youtube genre label (batch x 3864)

Dense input features are encoded one last time to reduce the number of weight decisions down to class label size [40].

(Batch x 8192)

Using Bidirectional LSTM to decode temporal data. Currently finding patterns between video frames to better understand the genre. Time step features get reduced from 100 down to one (many to one LSTM technique) since we need to find one genre label from a youtube video [40]. Variable "h" is a hidden representation of features essential to decode video frames.

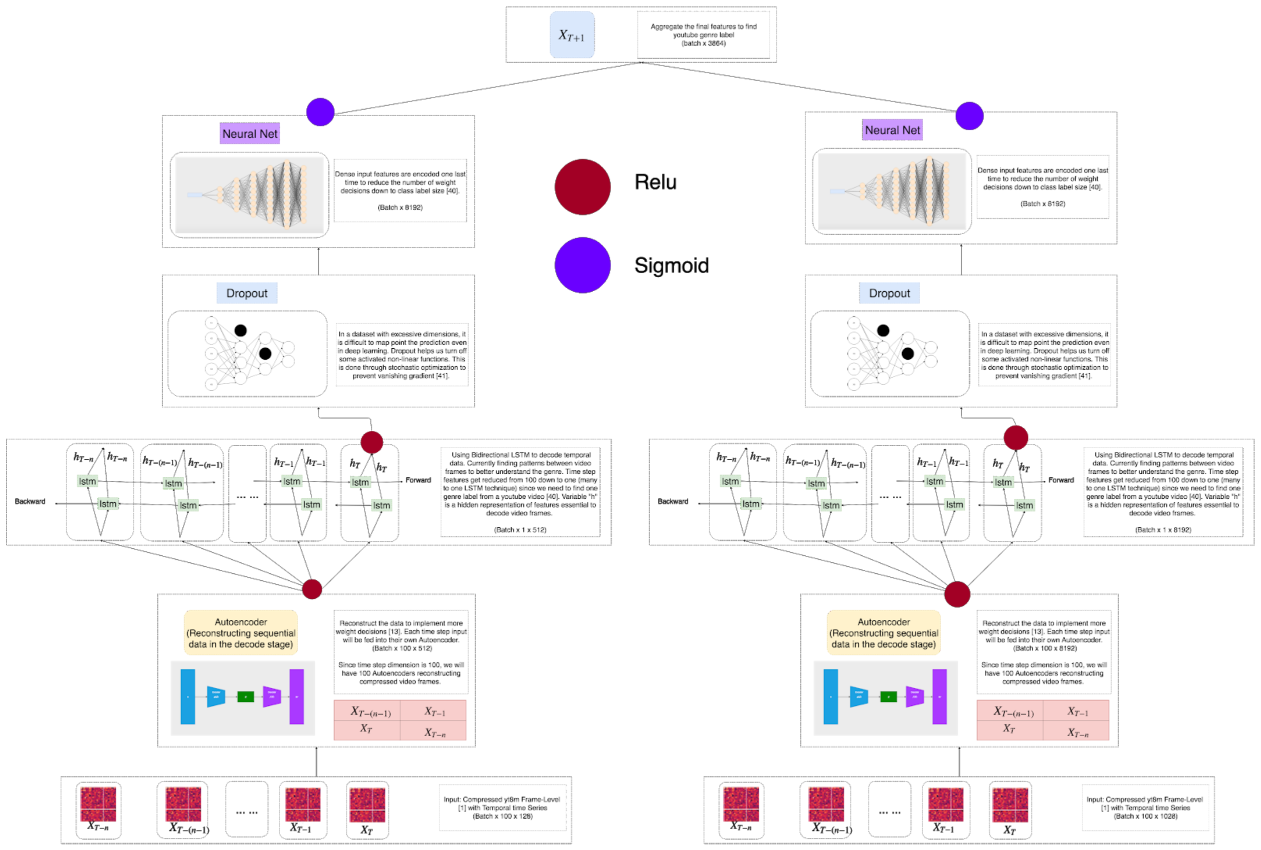
(Batch x 1 x 8192)

Reconstruct the data to implement more weight decisions [13]. Each time step input will be fed into their own Autoencoder. (Batch x 100 x 8192)

Since time step dimension is 100, we will have 100 Autoencoders reconstructing compressed video frames.

Input: Compressed yt8m Frame-Level [1] with Temporal time Series (Batch x 100 x 1028)

Stream LSTM



Aggregate the final features to find youtube genre label
(batch x 3864)

Dense input features are encoded one last time to reduce the number of weight decisions down to class label size [40].

(Batch x 8192)

In a dataset with excessive dimensions, it is difficult to map point the prediction even in deep learning. Dropout helps us turn off some activated non-linear functions. This is done through stochastic optimization to prevent vanishing gradient [41].

Using Bidirectional LSTM to decode temporal data. Currently finding patterns between video frames to better understand the genre. Time step features get reduced from 100 down to one (many to one LSTM technique) since we need to find one genre label from a youtube video [40]. Variable "h" is a hidden representation of features essential to decode video frames.

(Batch x 1 x 512)

Reconstruct the data to implement more weight decisions [13]. Each time step input will be fed into their own Autoencoder.
(Batch x 100 x 512)

Since time step dimension is 100, we will have 100 Autoencoders reconstructing compressed video frames.

Input: Compressed yt8m Frame-Level [1] with Temporal time Series
(Batch x 100 x 128)

Dense input features are encoded one last time to reduce the number of weight decisions down to class label size [40].

(Batch x 8192)

In a dataset with excessive dimensions, it is difficult to map point the prediction even in deep learning. Dropout helps us turn off some activated non-linear functions. This is done through stochastic optimization to prevent vanishing gradient [41].

Using Bidirectional LSTM to decode temporal data. Currently finding patterns between video frames to better understand the genre. Time step features get reduced from 100 down to one (many to one LSTM technique) since we need to find one genre label from a youtube video [40]. Variable "h" is a hidden representation of features essential to decode video frames.

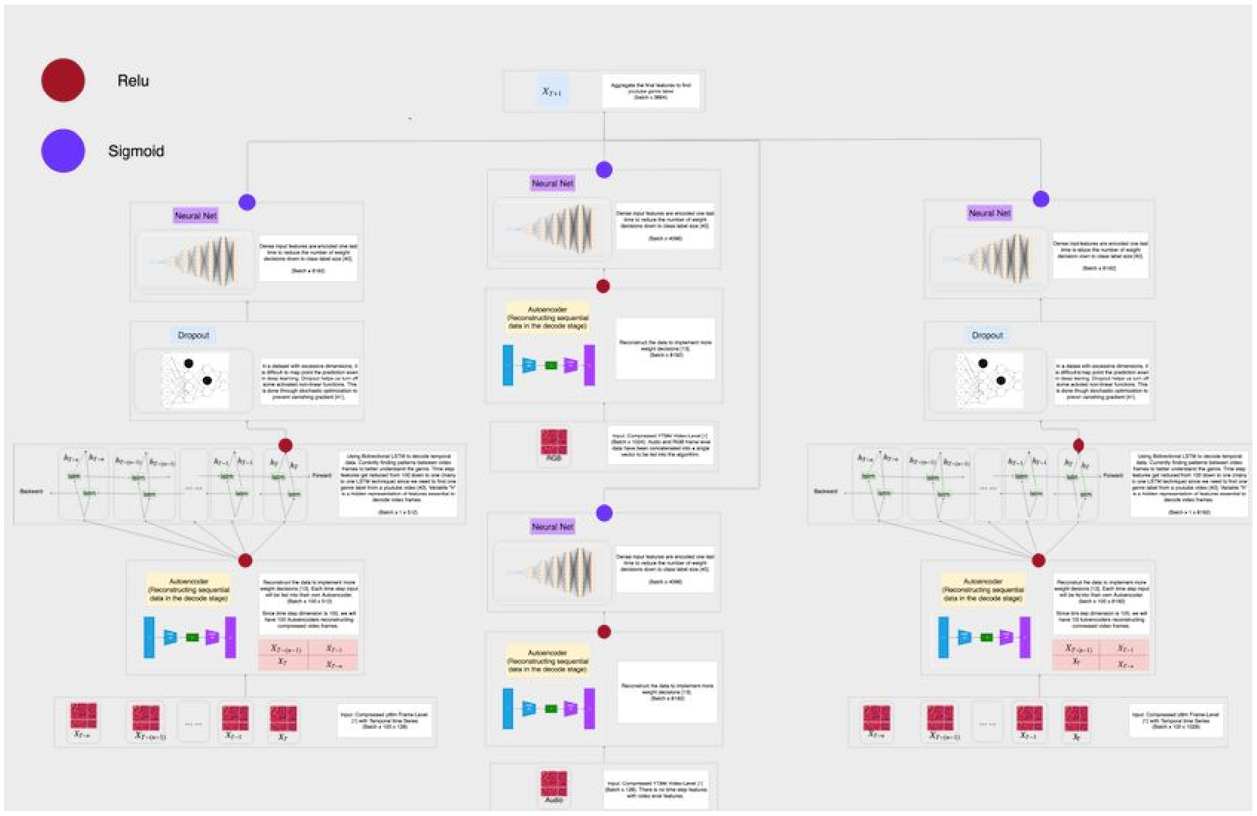
(Batch x 1 x 8192)

Reconstruct the data to implement more weight decisions [13]. Each time step input will be fed into their own Autoencoder.
(Batch x 100 x 8192)

Since time step dimension is 100, we will have 100 Autoencoders reconstructing compressed video frames.

Input: Compressed yt8m Frame-Level [1] with Temporal time Series
(Batch x 100 x 1028)

Neural Net Concat with LSTM Stream


Relu
Sigmoid
X_{T+1}
Neural Net
Dropout
Autoencoder (Reconstructing sequential data in the decode stage)
lstm
Backward
Forward
RGB
Audio