Large-Scale Multi-Label Video Classification Using QRNNs

To Isaac Zachary

ysto@connect.ust.hk

COMP 4971C (Summer 2018), Supervised by Dr. David Rossiter

1 Introduction

Video is one of the most prominent forms of content that we consume. With multiple video-centric platforms on the rise, from Snapchat to Twitch, video has become an integral part of our lives. It is crucial for the proprietors to understand the content and trends of the video content being curated. That said, with the huge amount of videos being processed through these aforementioned platforms, it has become a task that humans alone cannot keep up.

In this paper, we introduce a method to predict the relevant labels to any given video based on Quasi-Recurrent Neural Networks (Bradbury & Merity et al., 2016 [1]) architecture. Quasi-Recurrent Neural Networks (QRNNs) alternates between Convolutional layers and Pooling layers. This architecture is superior in accuracy and speed.

The experiment presented in this paper is based on the "Google Cloud & YouTube-8M Video Understanding Challenge" held on Kaggle in 2017 [8], follows the same protocol for evaluation, and compares to the winners of the presented challenge as a benchmark.

This priorities in the selection the model architecture for this task is having a light-weight model, meaning the model being small in file size, as well as require the least possible computation resources for training and deployment. This is due to many current models having high accuracies but require multiple Graphics Processing Units (GPUs) for training and deployment. This paper aims to provide a model at a reasonable accuracy that uses a small amount of computation resources.

2 Technical Methodology

2.1 Dataset

We will use the Youtube-8M dataset [2] for experimentation. The Youtube-8M dataset is the largest publicly available multi-label video classification dataset, with approximately 8 Million videos annotated with 3862 classes of labels [3]. The videos within the dataset averages 3.01 labels per video, where the number of labels per video ranges from 1 to 23. As this dataset covers over 500,000 hours of video, 2.6 billion audio and visual features have been extracted and pre-processed in advance by the Google Research Team as it would be infeasible for research teams to train hundreds of Terabytes worth of video for their model. The next section of this paper will outline the feature extraction process of the Youtube-8M dataset.

2.2 Youtube-8M Feature Extraction

A deep model, namely an Inception network[2], trained on Imagenet was used to pre-process the videos and extract frame-level features. An Inception Network is like a Convolutional Neural Network except it has been heavily modified to boost results and performance. The following points are the procedures and specifications of the feature extraction process of this dataset:

- Each video was decoded at 1 frame-per-second up to the first 360 seconds (6 minutes). This cap was implemented for storage and computation reasons
- Decoded frames are fed into the Inception Network
- The Rectified Linear Unit (ReLU) activation function of the last hidden layer before the classification layer is fetched
- The feature vector is 2048-dimensional per second of video
- Principal Component Analysis (PCA) along with whitening was applied to scale down the feature dimensions to 1024, followed by quantization (1 byte per coefficient)
- PCA and Quantization downsamples the size of the data by the factor of 8
- The mean vector and covariancce matrix for PCA was computed on all frames from the training partition
- Each 32-bit float was quantized into 256 distinct values (8 bits) using optimally computed (non-uniform) quantization bin boundaries
- The dataset is then broken into 3844 shards each for the training, validation, and testing partitions
- The dataset is separated into the Frame-level features dataset, which is approximately 1.53 Terabytes, and the Video-level features dataset, which is approximately 31 Gigabytes

The features extracted were also split into two types and two different datasets. Namely frame-level features and video-level representations. For frame-level features, 20 random frames are randomly sampled from each video, then the ground truth labels are associated with each frame. Video-level representations are simply fixed-length video features being extracted using the above procedures.

2.3 Evaluation

In order to prevent bias in the evaluation process, the evaluation protocol[2] provided by Google will be used to calculate the accuracy of a given model. The two protocols that are used to evaluate the accuracy of the model are Hit @ k, where k = 1, and Precision at equal recall rate (PERR).

2.3.1 Hit @ k

This is the percentage of test samples that contain at least one correct label in the top k predictions made by the model. In this case we will be using k = 1 to evaluate the accuracy of the model. Where $\text{rank}_{v,e}$ is the rank of entity e on video v, where the label with the highest confidence score is rank 1. G_v is the set of ground-truth labels for v, Hit@K can be represented as:

$$\frac{1}{|V|}\sum_{v\in V}\vee_{e\in G_v}\mathbb{I}(\text{rank}_{v,e}\leq k),$$

where $\vee$ is logical OR

2.3.2 Precision at equal recall rate (PERR)

For Precision at equal recall rate (PERR), we retrieve the same number of labels per video as there are in the ground-truth to measure the video-level annotation precision. With the same notation as Hit@k, PERR can be represented as:

$$\frac{1}{|V:|G_v|>0|}\sum_{v\in V:|G_v|>0}\left[\frac{1}{|G_v|}\sum_{e\in G_v}\mathbb{I}(\text{rank}_{v,e}\leq|G_v|)\right].$$

3 Model Architecture

3.1 Background

First, we shall explore the typical architecture of commonly seen Neural Networks, namely Convolutional Neural Networks (CNNs)[4] and Long Short Term Memory Units (LSTMs) [6].

A Neural Network is neurons arranged in layers. Generally speaking there are 3 types of layers; input layer, hidden layers and output layer. The neurons in the input layer are connected to neurons in the first hidden layer et cetera. Each neuron has learnable weights and biases. When an input is received, the neuron computes a dot product and follows up with non-linearity if necessary.

For Convolution Neural Networks (CNNs) as shown in figure 1, they are actually quite similar to the typical neural network. For example, most tips and tricks applicable to Neural Networks still apply, a loss function such as a normalized exponential function (Softmax Function) or a support vector machine still exists. The key difference is that CNNs make an explicit assumption that the input is a fixed sized vector, like an image.

This assumption made by CNNs is the reason why CNNs are generally chosen for object detection, text recognition tasks

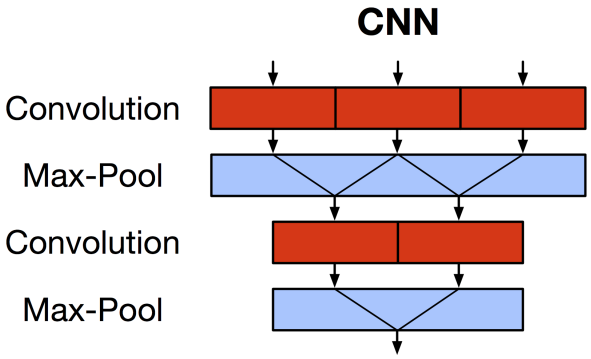

Figure 1: Typical CNN Architecture

etc. Regular Neural Networks don't scale well to full images as a fully connected architecture would result in wasteful results due to a huge number of parameters in the Neural Network. As CNNs only process fixed sized vectors, we can encode certain properties into the architecture, which vastly reduces the amount of parameters in the Neural Network, handling the overfitting issues that normal Neural Networks have with images. CNNs generally learn to recognize patterns in a space. For example, recognizing components of an image like lines and curves, then combining these components to recognize larger structures such as objects and faces.

Only accepting fixed sized vectors as an input and producing a fixed-sized vector as an output is also the most substantial limitation of Convolutional Neural Networks. In addtion, CNNs only perform mapping with a fixed amount of steps, determined by the number of layers in the CNN. This bounds the efficiency of CNNs in handling other types of data. Recurrent Neural Networks (RNNs) [5], on the other hand, are a more effective architecture in applications relating to sequential data. RNNs operates over sequences of vectors: sequences in the input, the output, or in the most general case both. However, RNNs are not capable of handling long-term dependencies as in most cases, it only looks at recent information to perform the present task. Hence Long Short Term Memory (LSTMs), which are units of a RNN, are introduced.

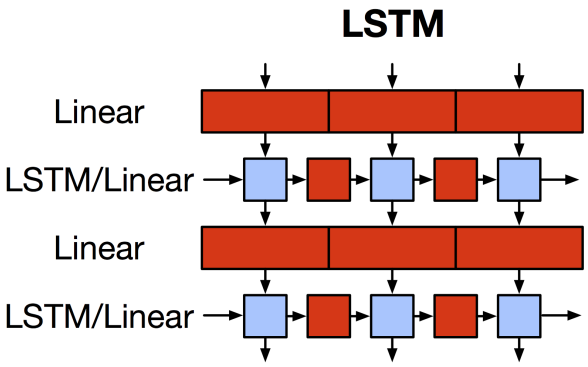

Figure 2: Typical LSTM Architecture

Long Short-Term Memory (LSTM) Hochreiter & Schmidhuber (1997) [6] as shown in figure 2, was originally designed to address the difficulties of training RNNs Bengio

et al (1994) [7]. LSTM units maintains a memory vector, which allows past information to be read, written or reset after a long period of time. This is complemented by a gating mechanism. Generally in an RNN a gradient vanishes or explodes due to backpropagation dyanamics, hence not being able to handle past information that is relevant to the events that are being processed. This is why LSTM units complements RNNs well, as it handles the long-term dependencies issues of RNNs. LSTM units are especially useful where past and present information is very important to the events being processed by the RNN.

3.2 Architecture

For the architecture of the model chosen, we selected a Quasi-Recurrent Neural Network (QRNNs)[2]. QRNNs are built using simple components of Neural Networks: a convolutional layer and a pooling layer. The idea of alternating between convolution layers and pooling layers is to use the convolutional layer to compute and map current representations, then use the pooling layers to handle sequential dependencies. This architecture is represented in Figure 3.

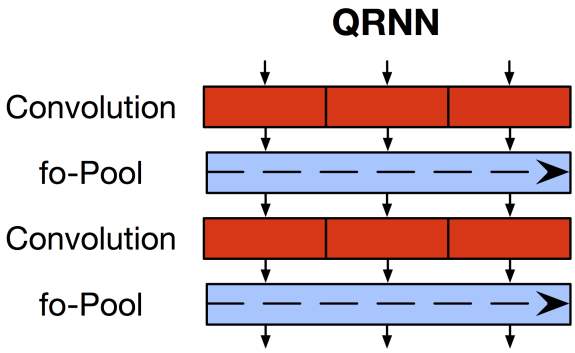

Figure 3: The QRNN Architecture

This architecture is especially effective for tacking this video classification task as videos are sequential data, which plays to the strength of RNNs and LSTMs. On the other hand, videos themselves are composed of multiple frames which are essentially images, which plays to the strength of CNNs. The QRNN architecture basically combines the strengths of both cNNs and RNNs (LSTMs) to classify the large amount of video data being inputted.

3.2.1 Convolutional Layer

QRNNs use convolutional layers in the timestep dimension to compute the intermediate vectors and gating vectors. Each input is computed with 3 vectors: a candidate vector, a forget gate, and an output gate.

When given an input sequence of n-dimensional vectors $x_1, x_2, ...x_T$, the convolution layer for the candidate vectors with m filters produces a sequence of T m-dimensional output vectors $z_1, z_2, ..., z_T$. The same process applies to forget gates and output gates, represented by

$$\mathrm{z}_t = \tanh(conv_{W_z}(x_t, ..., x_{t-k+1}))$$

$$\mathrm{f}_t = \sigma(conv_{W_f}(x_t, ..., x_{t-k+1}))$$

$$\mathrm{z}_t = \sigma(conv_{W_o}(x_t, ..., x_{t-k+1}))$$

where conv represents convolution and k is the filter size. This convolutional layer is no different from a normal convolution with tanh or sigmoid activations. But take note this is masked convolution which means that the output does not have a dependency in the form of a future input.

3.2.2 Pooling

The QRNN layer performs input-dependent pooling, then combines the convolutional features linearly via a gate. As shown in figure 3. Hence with the inputs from the convolutional layer, the QRNN computes the hidden states using a forget gate and an output gate as represented by:

$$\mathrm{c}_t = f_t \odot c_{t-1} + (1 - f_t) \odot z_t$$

$$\mathrm{h}_t = o_t \odot c_t$$

where $\odot$ represents element-wise multiplication.

This equation is denoted as *fo*-pooling by the authors, as this function contains a forget gate followed by an output gate. Other forms may include an independent input followed by a forget gate, represented by

$$\mathrm{c}_t = f_t \odot c_{t-1} + i_t \odot z_t$$

$$\mathrm{h}_t = o_t \odot c_t$$

where $\odot$ represents element-wise multiplication.

The pooling component of the QRNN architecture is similar to the traditional LSTM unit. The difference is that sequential processes only occur in the pooling layer and the values of $z_t, f_t, \& o_t$ are independent. Heavy computation is done in parallel and sequential processing is strictly done in pooling layers. Other forms of QRNNs were also explored in the paper however it was not tested out. Like conventional Neural Networks, multiple QRNN layers stacked together can create a model that is able to handle more complex functions.

3.3 Why QRNN was chosen

As aforementioned the QRNN architecutre is especially effective for tacking this video classification task as videos are sequential data, composed by a series of images. Hence QRNNs have an advantage in terms of predictive accuracy over traditional CNNs, RNNs or LSTMs.

In addition, another reason why QRNNs were chosen to tackle this task was the computational efficiency. QRNNs have increased parallelism as sequential data is strictly computed in the pooling layers, and heavy computation is done via parallel methods. Therefore, it trains and tests faster than traditional Neural Networks. Bradbury & Merity et al. (2016), claims that QRNNs are up to 16 times faster in both training and testing time, however, this paper has not explored the training time and testing time of QRNNs as there are no competing models at this stage.

4 Experiments

4.1 Training

The model was trained on a Macbook Pro with a 2.8 GHz Intel Core i5 CPU and 8 GB 1600 MHz DDR3 memory with all shrads of the Youtube-8M dataset. The model was built on Keras, an open-sourced Python Neural Network library on top of Tensorflow. Tensorflow, another open-sourced framework was used to train and test the model. The model was exclusively trained on the CPU of the device. The model was trained on 5 epochs to shorten the training time required. The model was built using the console Sublime Text 3.1.1 and trained on the Terminal provided by macOS High Sierra Version 10.13.

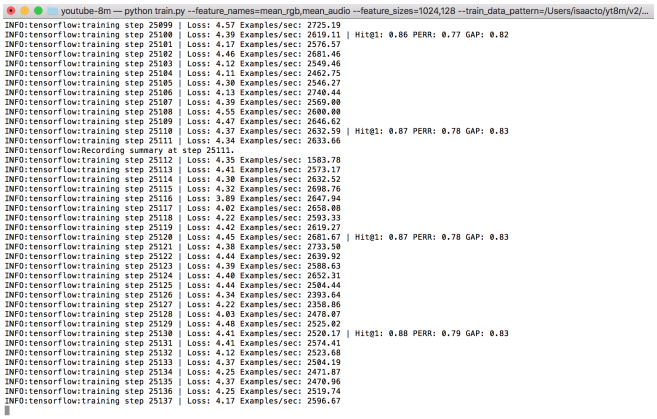

Figure 4: An illustration of the training in progress

4.2 Results

As aforementioned to prevent bias in the evaluation process, Google's evaluation protocol was used to evaluate the accuracy of the model.

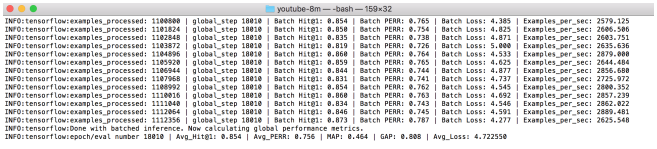

Figure 5: Evaluation

The results yielded:

- Average Hit@1 = 0.854
- Average Precision at equal recall rate = 0.756
- Global Average Precision = 0.808

4.3 Comparison of results

In the technical paper of the Youtube-8M dataset, Frame-level feature models and video-level features models were built and tested on the dataset as a benchmark [2]. We compared the results from the QRNN with the results from the video-level models, namely the Mixture of Experts model and (proposed by Jacobs and Jordan 1994) [12], the Logistic Regression model. For frame level features, the results from the LSTM scored significantly below the QRNN as well.

Level	Model	PERR	Hit@1
Video	This Study (QRNN)	75.6	85.4
Video	Logistic Regression	53.0	60.5
Video	Mixture of Experts	55.8	63.3
Frame	LSTM	57.3	64.5

The results from the QRNN was also compared to the top 3 winners of the Google Cloud YouTube-8M Video Understanding Challenge 2017. They include teams from Baidu and Tsinghua University.

Instead of Hit@k and PERR, Google Cloud YouTube-8M Video Understanding Challenge 2017 uses Global Average Precision (GAP) at k, where k = 20 as an evaluation metric [8]. Each submission contains a list of the top 20 predicted labels for each video and their corresponding confidence scores. The evaluation treats each predicted label and the confidence score as an individual data point to compute the GAP across all the predictions and videos. If there are N predictions sorted by decreasing confidence scores, the Global Average Precision is computed by

$$\text{GAP} = \sum_{i=1}^{N} p(i) \triangle r(i)$$

where N is the number of final predictions made, p(i) is the precision and r(i) is the recall.

Rank	Model	GAP
1	Ensemble of 25 models [9]	84.966
2	Weighted 74 models [10]	84.589
3	Ensemble of of 57 models [11]	84.541
n/a	This Study (QRNN)	80.8

Where "Rank" represents the ranking that model gained in the challenge.

As evident from the Global Average Prediction scores, the QRNN falls shy against the results achieved by the winners from 2017. This may be because the winners selected to use multiple models instead of one model, whereas an ensemble method can be explored in the future.

5 Further Exploration

5.1 Expanding upon the model

In the future I would like to test how the results of an ensemble of models would differ from the result from one model, as evidently one single model may not be able to compete with the results of an ensemble of models.

5.2 Other Possible Contexts

It is possible to expand this model to cover other social networks or platform with a huge set of video data. This can be applied to improve search algorithms, or to inspire creators on the copywriting of their content or even on what content to create next.

References

[1] James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. Quasi-recurrent neural networks. CoRR, abs/1611.01576, 2016.

[2] Sami Abu-El-Haija, Nisarg Kothari, Joonseok Lee, Paul Natsev, George Toderici, Balakrishnan Varadarajan, and Sudheendra Vijayanarasimhan. Youtube-8m:A large-scale video classification benchmark. CoRR, abs/1609.08675, 2016.

[3] YouTube-8M: A Large and Diverse Labeled Video Dataset for Video Understanding Research. (n.d.). Retrieved from https://research.google.com/youtube8m/

[4] Karpathy, A. (n.d.). Convolutional Neural Networks (CNNs / ConvNets). Retrieved from http://cs231n.github.io/convolutional-networks/conv

[5] Andrej Karpathy, Justin Johnson, and Fei-Fei Li. Visualizing and understanding recurrent networks. CoRR, abs/1506.02078, 2015.

[6] Sepp Hochreiter and Jürgen Schmidhuber.Long short-term memory. Neural Comput., 9(8):1735–1780, November 1997.

[7] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks, 5(2), 157–166.

[8] Google Cloud YouTube-8M Video Understanding Challenge. (n.d.). Retrieved from https://www.kaggle.com/c/youtube8mevaluation

[9] Antoine Miech, Ivan Laptev, and Josef Sivic. Learnable pooling with context gating for video classification. CoRR, abs/1706.06905, 2017.

[10] He-Da Wang, Teng Zhang, and Ji Wu. The monkey-typing solution to the youtube-8m video understanding challenge. CoRR, abs/1706.05150, 2017.

[11] Fu Li, Chuang Gan, Xiao Liu, Yunlong Bian, Xiang Long, Yandong Li, Zhichao Li, Jie Zhou, and Shilei Wen. Temporal modeling approaches for large-scale youtube-8m video understanding. CoRR, abs/1707.04555, 2017.

[12] M. I. Jordan. Hierarchical mixtures of experts and the em algorithm. Neural Computation, 6, 1994.