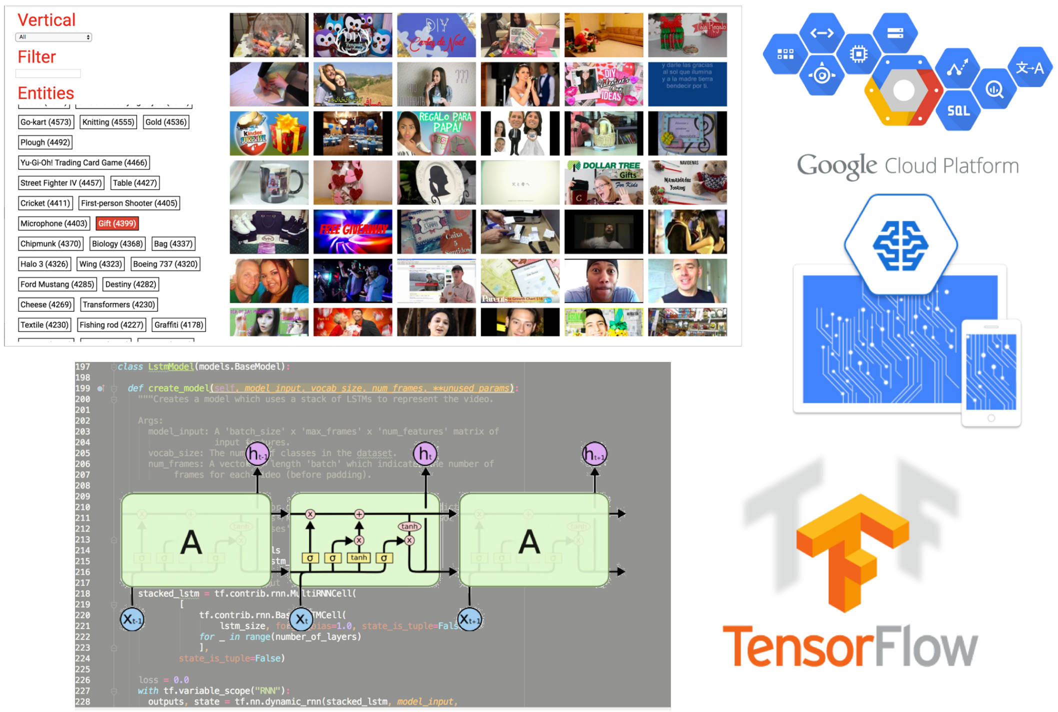
Large-Scale Video Classification with Cloud Computing

EECS 351 Final Project Report

Yunsheng Bai
University of Michigan, Ann Arbor
yba@umich.edu

Tyler Kohan
University of Michigan, Ann Arbor
tkohan@umich.edu

Murat Turkeli
University of Michigan, Ann Arbor
mturkeli@umich.edu



```
class LstmModel(models.BaseModel):

  def create_model(self, model_input, vocab_size, num_frames, **unused_params):
    """Creates a model which uses a stack of LSTMs to represent the video.

    Args:
      model_input: A 'batch_size' x 'max_frames' x 'num_features' matrix of
                   input features.
      vocab_size: The num  f classes in the dataset.
      num_frames: A vecto   length 'batch' which indicat    e number of
          frames for each  ideo (before padding).









    stacked_lstm = tf.contrib.rnn.MultiRNNCell(
            [
                tf.contrib.rnn.Bas    TMCell(
                    lstm_size, for    bias=1.0, state_is_tuple=Fals
                for _ in range(number_of_layers)
                ],
            state_is_tuple=False)

    loss = 0.0
    with tf.variable_scope("RNN"):
      outputs, state = tf.nn.dynamic_rnn(stacked_lstm, model_input,
```

Fig. 1: Overview. This project involves a large-scale video dataset, deep learning with TensorFlow, Google Cloud Platform, etc.

Abstract—This report will detail a project derived from a contest available from Kaggle. Kaggle is a platform for predictive modelling and analytics competitions, and the company collects data from companies and data miners all over the world. The particular contest that inspired this project was Google Cloud & Youtube-8M Video Understanding Challenge [1]. This contest challenges the community to help with large-scale video understanding using Google's Machine Learning Platform [2]. **Keywords—video classification; cloud computing; machine learning; deep learning**

I. Introduction

Video captures a cross-section of our society, and major advances in analyzing and understanding video have the potential to touch all aspects of life from learning and communication to entertainment and play. In this project, we participate in a public Kaggle competition sponsored by Google. Google is inviting the Kaggle community to join efforts to accelerate research in large-scale video understanding, while giving participants early access to the Google Cloud Machine Learning (Cloud ML) platform. In this competition, we develop classification algorithms which accurately assign video-level labels using the new and improved YouTube-8M V2 dataset [3] [4].

The YouTube-8M dataset (Fig. 2) contains over 7 million videos, and is split into 3 parts: training data, evaluation data, and testing data. All videos have ids that can be used to retrieve the original videos on YouTube. Videos in the training and evaluation data contain labels that correspond to video classes. The YouTube-8M dataset comes with frame-level features (1.7TB) including the RGB and audio features, and video-level

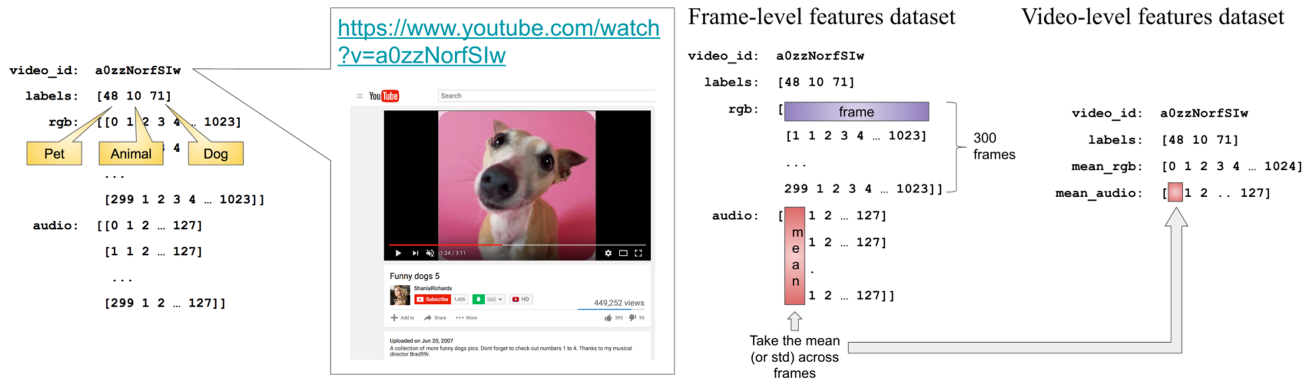


Fig. 2: Dataset illustration.

features (30GB) including the mean RGB and mean audio features. Each video is decoded at 1 frame per second, and is capped at the first 300 frames. The RGB feature matrices are created by feeding raw pixels through an Inception network trained on ImageNet, and taking the ReLU activation of the last hidden layer. The video-level data is created by taking the mean (or the standard deviation) of the features across frames.

A large amount of research on video understanding and machine learning is available online and points to many different methods to solving video understanding problems. The most common approach seems to be the use of a Convolutional Neural Network (CNN). The following papers show that CNN is a state-of-the-art approach for video and image classification. Wu [5] proposed a three-layer neural network that fuses information from three CNN streams: object stream, scene stream, and generic feature stream, for video understanding. Zhang [6] proposed a CNN architecture for sketch classification. Wang [7] suggests combining CNN and Recurrent Neural Network (RNN) for the purpose of image classification. Wang suggests using this combined NN for the correlation between multiple labels of an image, which is highly relevant to the Y-8M dataset. That paper also details the Long Short-Term Memory (LSTM) method, a particular kind of RNN. Feichtenhofer [8] uses CNN for human action recognition in videos. Murthy [9] combines neural networks and decision trees for multi-class image classification. Huang [10] proposed an algorithm to improve classification using CNN with imbalanced datasets. Ng [11] proposed two methods to classify videos using CNNs: the first method is to divide the video into temporal events (e.g. frames) and look at them individually; the second method is to look at videos in an ordered sequence of frames. Xiao [12] proposed a method to apply visual attention to fine-grained classification using CNN. Perronnin and Larlus [13] proposed a combination of CNN and Fisher Vector (FV) to have both their strength: CNN for accuracy and FVs for low training cost. Tang [14] proposed a method to enhance image classification using CNN by integrating GPS coordinates into the model. Donahue [15] proposed a novel recurrent convolutional architecture based on RNN to learn on sequential data sets to investigate events that are recurrent or temporally deep.

Research points toward CNN and LSTM being the most efficient models to approach this problem, and were the first solutions we tried. We also found an ensemble model called Mixture of Expert (MOE) in Abu-El-Haija's [16] research on the Y-8M dataset. In that paper, the MOE model was found to be the most efficient and yielded the best results, and so was our primary focus.

II. METHODS

Our code is written in Python with TensorFlow. The reasons are: (1) TensorFlow is an open source software library that makes it to perform complex machine learning concepts, especially deep learning; (2) Kaggle provides starter code that performs training and inference, and that code uses TensorFlow.

We stored and processed the data on Google Cloud Platform. It allows users to store large amount of data and easily create and train machine learning models. There is a large amount of existing framework available on Google's platform using TensorFlow. This framework gave us a basis to build off of and significantly reduced the amount of time required to build our models.

We train the following models: Mixture of Expert (MOE) model (Fig. 3), Long Short-Term Memory (LSTM) model, and some variants of the MOE model, one of which is CNN. CNN and LSTM are two popular approaches, and can be found in the research literature mentioned in the previous section. MOE model is a less popular one, and its description can be found in [17].

The diagram shows the how an input matrix is transformed into an output matrix by a MOE model with two experts. The entire process starts with the input matrix. Each row represents a video, so in this example, 100 videos are processed as a batch. The first 1024 elements of the video are its mean rgb features. They are appended by is 128 mean audio features. The actual matrix calculation starts in the expert network. The first layer of the expert network has 4716*2=9432 neurons., numbered as 0, 1, 2, …, 9432 from the top to the bottom. Each neuron conceptually corresponds to a video class. For example, neuron 0 corresponds to class "Cat," neuron 1 corresponds to class "Dog," etc. The top 4716 neurons conceptually belong to expert 1, and the bottom 4716 ones conceptually belong to expert 2. Each expert looks at the input matrix independently, so if neuron 4716 corresponds to the same class as neuron 0, 4717 the same class 1, etc. Each neuron has weight vector w, and a bias b, and uses the ReLU activation function f. W and b are randomly initialized. Each neuron produces one output number o from a video vector v as in equation (1).

$$o = f(w * v + b) \qquad (1)$$

However, packages like TensorFlow process the entire input matrix rather than one video at a time for optimization opportunity. In other words, the actual calculation is performed at the matrix level. In this example, the 9432 neurons together produce a matrix with shape (batch size, 9432), which is shown below the first layer. A reshape operation is performed before the data is sent to a second layer, so that each row represents a particular class for a particular video. It has only two numbers, and no longer (and cannot) represent an entire video. The first video becomes the yellow vector plus the blue one, a rectangle block. Now this matrix is sent to the second layer, which is essentially applying a sigmoid function to each element as described on equation (2).

$$\sigma(x) = \frac{1}{1+e^{x}} \qquad (2)$$

The dimension does not change, but each element is "squashed" to a number between 0 and 1. In machine learning, this is essentially logistic regression, where the number between 0 and 1 gets a probabilistic interpretation: it can be treated as a probability of something! In this case, the top left element, 0.1, for example, can be thought of as the probability that video 0 belongs to class "Cat" is 0.1, which is assigned by expert 1, because of the way it was reshaped previously. Its next element, 0.2, means that expert 2 believes that the probability that video 0 belongs to class "Cat" is 0.2. They are usually different because neuron parameters are

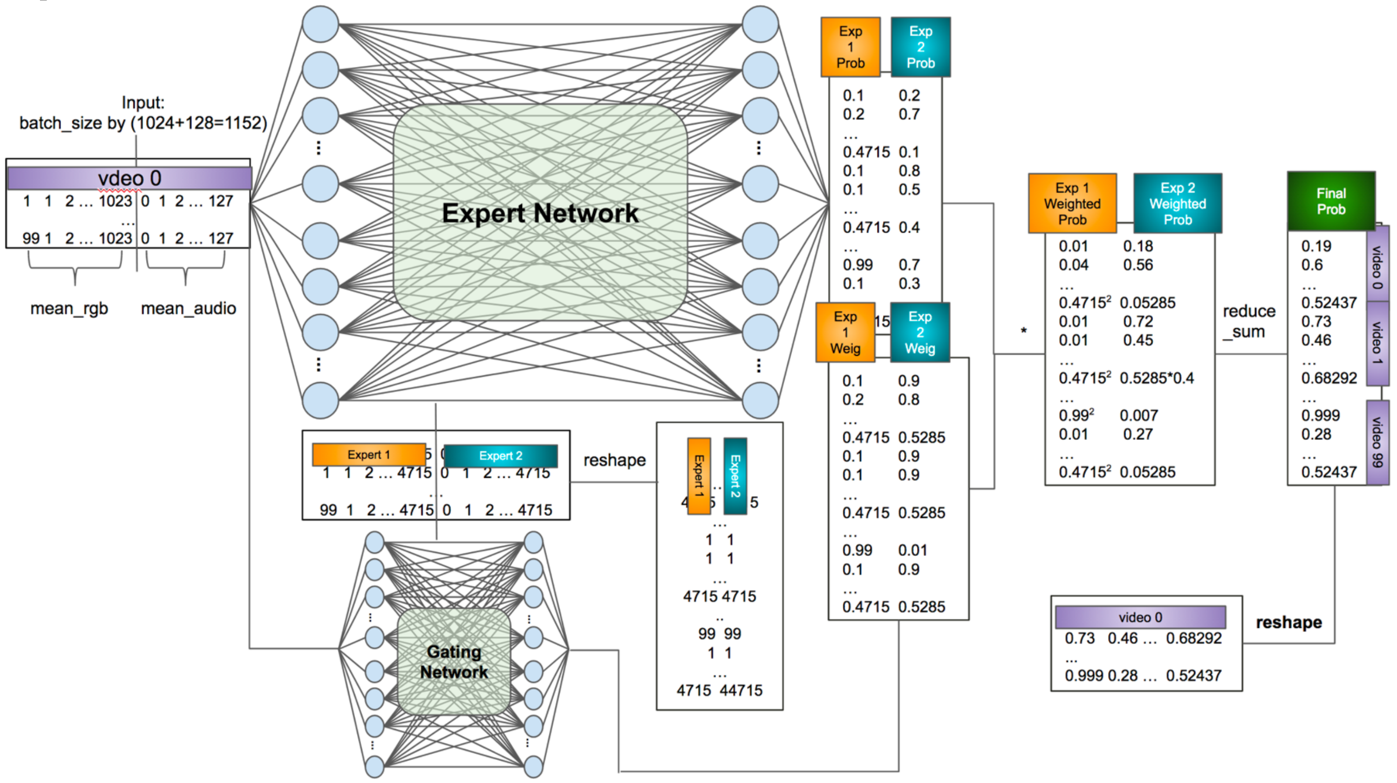


Fig. 3: Mixture of experts (MOE) overview

initialized randomly. However, a single probability is needed both for human understanding and for the final submission purpose: it is not acceptable to say that video 0 belongs to class "Cat" with probability 0.1 and 0.2. A naive approach to combine them is give each expert 50% of weight, so the final submission is 0.15. However, the MOE model is more than that: it introduces another neural network, called the gating network, to give each expert a weight on each class for each video. That is the bottom neural network in the diagram. The only difference between the expert network and the gating network is the second layer; instead of a sigmoid function, the gating network uses a softmax function (3).

$$\rho(x_{ij}) = \frac{e^{x_{ij}}}{\sum_{j=0}^{n} e^{x_{ij}}} \qquad (3)$$

In this case, n=2 because there are two experts. Notice that each element of the resulting matrix also depends on the other elements in the same row. Specifically, each row of the resulting matrix adds up to 1. In this example, 0.1+0.9=1, 0.2+0.8=1, etc. This nice property of the softmax function gives the resulting matrix the following interpretation: each element is a weight assigned to an expert for a class and a video. Because it has the same dimension as the expert matrix on the top, there is a one-to-one mapping between the expert matrix and the gating matrix. The first row of the gating matrix can be interpreted this way: expert 1 has only 10% weight on the question, while expert has 90% weight on the question, "Is video 0 a video about 'Cat?'" By element-wise matrix multiplication and row summation (called "reduced_sum" in TensorFlow), the final probability that video 0 belongs to "Cat" is 0.1*0.1+0.2*0.9=0.19. Each video and class combination has a probability, which is exactly what is needed. Finally, it is reshaped to the original dimension.

The process described above is only one forward pass of the input matrix. In prediction, the weights of neurons have been fixed, so only one forward pass is necessary to generate the submission file. However, to obtain the optimal parameters, we need to train the neural networks. Training is an iterative process, with each step involving the following operations: forward propagation (forward pass), loss calculation, and backward propagation. The loss function takes the output matrix and the label matrix, and calculates the deviation between the predictions and the ground-truth labels, which is to be minimized. It is chosen to be the cross entropy loss function (4).

$$\text{loss} = -\frac{1}{n}\sum_{i}^{m}(y_i * \ln a_i + (1 - y_i) * \ln(1 - a_i)) \qquad (4)$$

In this function, y_i is the ith row of the output matrix, and a_i is the ith row of the label matrix. [18] has a nice explanation on the advantage of this loss function. The backward propagation minimizes the loss function by updating the neuron parameters using the gradient descent method. Specifically, it calculates the gradients of the loss function with respect to the weights of neurons, from the end to the beginning using the chain rule. Then it updates the weights using the Adam gradient descent method [19]. Training stops when one of the following criterion is met: a predefined maximum number of epochs is reached, the training input files are exhausted, the loss is below a predefined threshold, the process is manually terminated, etc. The neuron parameters are dumped to the cloud storage in the form of checkpoint files as training goes forward, which will be loaded by the inference process to make predictions.

Per Kaggle, submissions are evaluated using the Global Average Precision (GAP) at k, where k=20. For each video, we will submit a list of predicted labels and their corresponding confidence scores. The evaluation takes the predicted labels that have the highest k confidence scores for each video, then treats each prediction and the confidence score as an individual data point in a long list of global predictions, to compute the Average Precision across all of the predictions and all the videos.

If a submission has N predictions (label/confidence pairs) sorted by their confidence score, then the Global Average Precision is computed as seen on equation (5).

$$\text{GAP} = \sum_{i=1}^{N} p(i)\Delta r(i) \qquad (5)$$

where N is the number of final predictions with up to 20 label/confidence pair per video (Kaggle only considers the first 20 label/confidence pairs for each video), p(i) is the precision of the ith pair, and Δr(i) is the recall. Precision is the fraction of correct labels that we predict over the number of labels that we predict. Recall is the fraction of correct labels that we predict over the number of ground truth labels. This corresponds to the area under the curve of Fig. 4, a precision-recall plot.

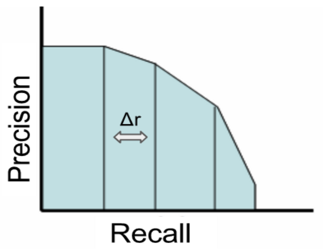


Fig. 4. Precision Recall curve.

III. RESULTS

We have achieved results that range from a GAP score of about 60% to almost 80% using our different methods. The best scores achieved by different models are shown in Table 1.

TABLE I. BEST RESULT BY MODEL

Approach	GAP (Global average precision)
LSTM	77.07%
Mixture of experts (MOE) w/ 9 experts	79.87%
MOE w/ convolutional layers	60.54%
MOE w/ augmented with gradient	71.12%
MOE w/ augmented with std and other stats	77.77%

We have mostly focused our efforts on the MOE model as it is the one yielding the best results. The top score, 79.87%, agrees with the best result achieved in [4], which also uses the MOE model. Therefore, we have essentially reproduced the best result of [4]. On top of being a top performer, the MOE model has a relatively short training time. As can be seen in Fig. 5, we achieve peak performance in about 40,000 training steps. The GAP shown on the graph is based on the training data whose label we know. The results presented in Table 1 are from evaluations ran by Kaggle on the inference data we don't have labels for. The difference between the two is expected due to the overfitting problem of neural network models.

To put this in perspective with time for this particular example, we were doing 0.9 training steps per second, or close to 14 hours of training to get to 45,000 training steps.

In contrast to the MOE model, the GAP achieved on our LSTM model is shown in Fig. 6. Although we can reach a peak GAP in 16,000 training steps, the training performance of this model is only 0.23 steps per second or a little over 19 hours to reach the 16,000 steps.

In addition, we have tried 3 variants of the MOE model. The MOE model with convolutional layers only yields 60% of GAP score, which is reasonable given the fact that the RGB features are already fetched from the end of an Inception network. The MOE augmented with gradient model is an MOE model whose input contains a gradient vector of the data in addition to the RGB and audio features, doubling the size of the input. The performance, however, drops to 71% as suggested by the noisy GAP graph in Fig. 7. Another data augmentation method involves the standard deviation and other statistics of the frame level features. It gives a lower GAP score, but is better than the previous two variants of the MOE model. We leave the investigation of data augmentation as future work.

Our results are relatively close to the top of the Kaggle ladder where the top performers have GAP scores in the low 80s (by April 21, 2017). This makes us confident about being able to reach a good position on the ladder in future.

IV. FUTURE WORK

In future, we will continue working on the data and the model. For the data, we will further augment the video-level dataset. For the model, we may implement other models including Adaptive Boosting, Decision Tree, etc. The difficulty is that some of the popular machine learning models do not scale well to very large datasets. Besides, we will continue improving the MOE model. For example, Jacobs and Jordan originally proposed the MOE model with a hierarchical structure [20].

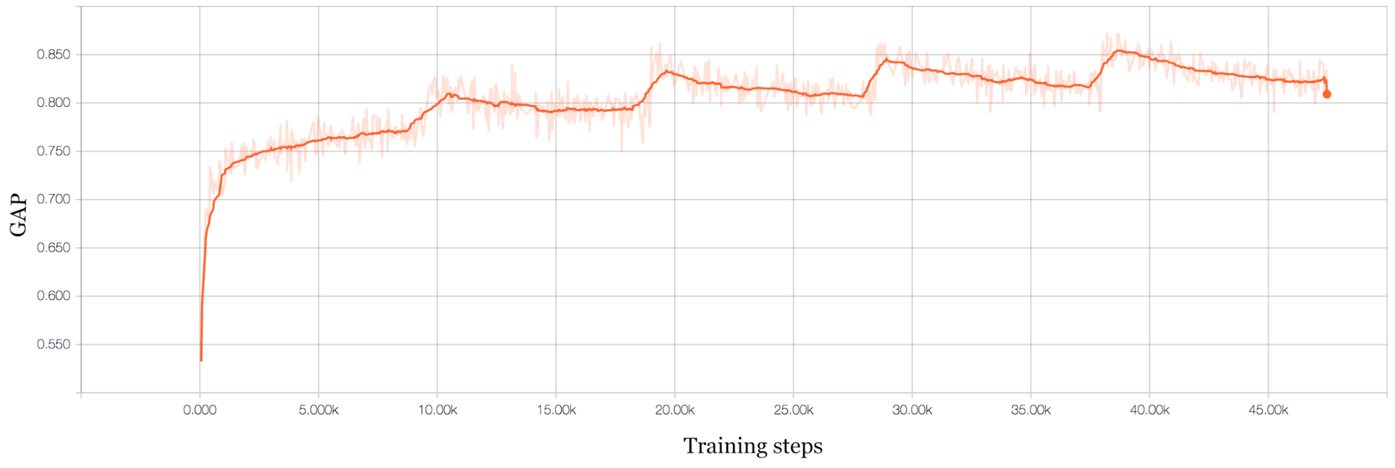


Fig. 5. GAP by training step for the MOE model.

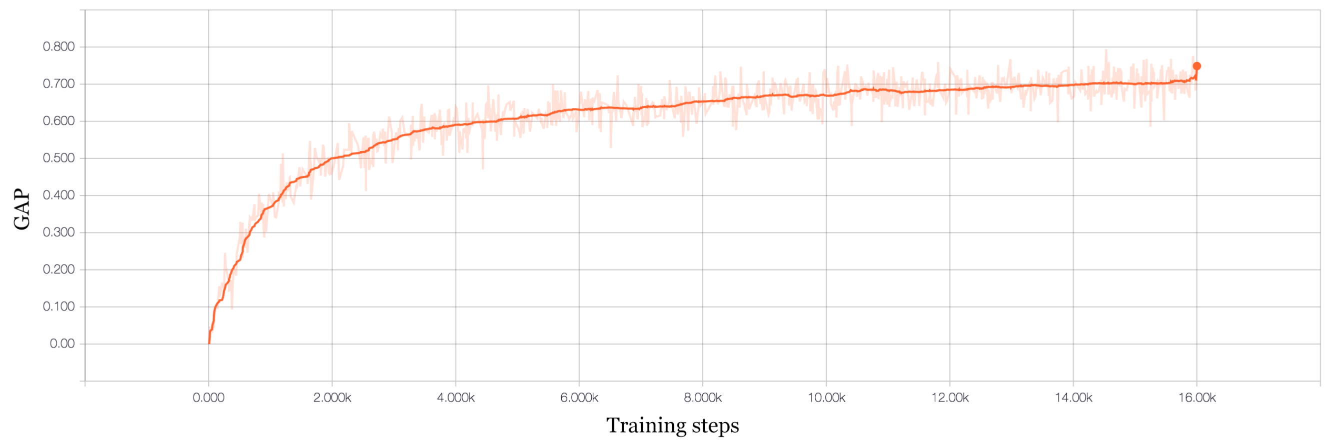


Fig. 6. GAP by training step for LSTM model.

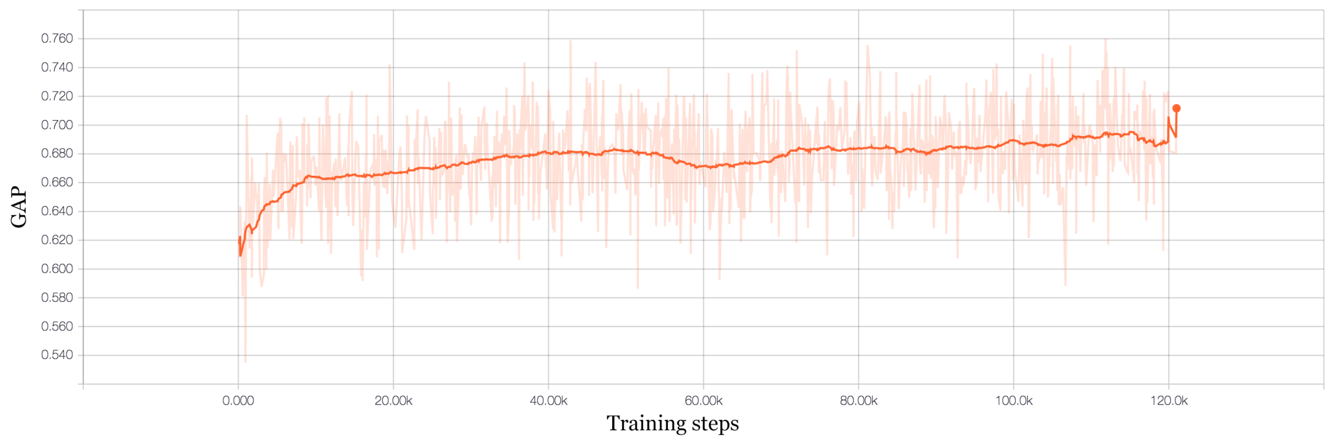


Fig. 7. GAP by training steps for the MOE model with gradient vector.

V. CONCLUSION

In conclusion, our project was an approach to a Kaggle competition on large-scale video classification. We use Google Cloud Platform to store and process the data. Initial research indicated that CNN may be the best approach, but we eventually determined that MOE was the best model for approaching this problem. Our results show that with the MOE model, we can approach about an 80% accuracy on labeling for the given videos, while LSTM and other variants of the MOE model give lower accuracy. These results show that even on a very large scale, video understanding using neural networks can reach very high levels of accuracy. Given that we still have many future work to do, the capabilities of large-scale video understanding with machine learning can reach even higher accuracy and may translate to large-scale classification of other large datasets. We will continue working on the competition for the upcoming weeks.

REFERENCES

[1] Google Cloud & YouTube-8M Video Understanding Challenge. Accessed April 21, 2017. https://www.kaggle.com/c/youtube8m/.

[2] Google Cloud Machine Learning Engine. Accessed April 21, 2017. https://cloud.google.com/ml-engine/.

[3] YouTube-8M Dataset. Accessed April 21, 2017. https://research.google.com/youtube8m/.

[4] Abu-El-Haija, Sami, Nisarg Kothari, Joonseok Lee, Paul Natsev, George Toderici, Balakrishnan Varadarajan, and Sudheendra Vijayanarasimhan. "Youtube-8m: A large-scale video classification benchmark." arXiv preprint arXiv:1609.08675 (2016).

[5] Wu, Zuxuan, Yanwei Fu, Yu-Gang Jiang, and Leonid Sigal. "Harnessing object and scene

semantics for large-scale video understanding." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3112-3121. 2016.

[6] Zhang, Hua, Si Liu, Changqing Zhang, Wenqi Ren, Rui Wang, and Xiaochun Cao. "Sketchnet: Sketch classification with web images." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1105-1113. 2016.

[7] Wang, Jiang, Yi Yang, Junhua Mao, Zhiheng Huang, Chang Huang, and Wei Xu. "Cnn-rnn: A unified framework for multi-label image classification." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2285-2294. 2016.

[8] Feichtenhofer, Christoph, Axel Pinz, and Andrew Zisserman. "Convolutional two-stream network fusion for video action recognition." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1933-1941. 2016.

[9] Murthy, Venkatesh N., Vivek Singh, Terrence Chen, R. Manmatha, and Dorin Comaniciu. "Deep decision network for multi-class image classification." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2240-2248. 2016.

[10] Huang, Chen, Yining Li, Chen Change Loy, and Xiaoou Tang. "Learning deep representation for imbalanced classification." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 5375-5384. 2016.

[11] Yue-Hei Ng, Joe, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. "Beyond short snippets: Deep networks for video classification." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 4694-4702. 2015.

[12] Xiao, Tianjun, Yichong Xu, Kuiyuan Yang, Jiaxing Zhang, Yuxin Peng, and Zheng Zhang. "The application of two-level attention models in deep convolutional neural network for fine-grained image classification." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 842-850. 2015.

[13] Perronnin, Florent, and Diane Larlus. "Fisher vectors meet neural networks: A hybrid classification architecture." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 3743-3752. 2015.

[14] Tang, Kevin, Manohar Paluri, Li Fei-Fei, Rob Fergus, and Lubomir Bourdev. "Improving image classification with location context." In Proceedings of the IEEE International Conference on Computer Vision, pp. 1008-1016. 2015.

[15] Donahue, Jeffrey, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. "Long-term recurrent convolutional networks for visual recognition and description." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2625-2634. 2015.

[16] Abu-El-Haija, Sami, Nisarg Kothari, Joonseok Lee, Paul Natsev, George Toderici, Balakrishnan Varadarajan, and Sudheendra Vijayanarasimhan. "Youtube-8m: A large-scale video classification benchmark." arXiv preprint arXiv:1609.08675 (2016).

[17] Hatami, Nima. "Some proposals for combining ensemble classifiers." PhD diss., University of Cagliari, 2012.

[18] Michael Nielsen. Neural Networks and Deep Learning. Accessed April 21, 2017. http://neuralnetworksanddeeplearning.com/chap3.html.

[19] Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

[20] Jordan, Michael I., and Robert A. Jacobs. "Hierarchical mixtures of experts and the EM algorithm." *Neural computation* 6, no. 2 (1994): 181-214.