Life-Long Learning Framework from Unsupervised Image Embeddings on Twitch Content

Prepared by Essam Sleiman - Last updated on 08/16/2022

Summary

As a result of high costs of manually labeled data and constantly changing Twitch content, Twitch Machine Learning Teams can benefit from 1.) Training an image embedding on unlabeled Twitch content using self-supervised learning computer vision methods and 2.) A pipeline which automatically and effectively continuously trains these embeddings on new stream data. If a model learns one set of parameters to learn a feature in one dataset, when it updates its parameters while training on a different dataset, it overwrites the original parameters, forgetting the original feature representations to some extent. We seek to mitigate these effects by applying a method called Learning Without Forgetting. In this work, we build an end-to-end Self-Supervised Continual Learning Framework for Twitch ML teams to continuously train an Image Embedding on new Twitch streams. This Image Embedding can then be finetuned on small labeled datasets to solve a variety of downstream tasks.

Background

Twitch utilizes intelligent machine learning systems to solve a multitude of tasks in the arena of content detection, intelligent search, recommendations, safety, etc. To solve a task such as moderating abusive content, a dataset of labeled content violations is gathered and used to train a machine learning model to correctly detect violations of future unseen content.

As a result of costly computation for model training and the increased size and diversity of public datasets, transfer learning, the process of fine-tuning an already pretrained model on a new task, has become incredibly popular and successful in the ML community. Furthermore, to combat expensive manual labeling of data, recent research introduced self-supervised learning methods in which feature representations can be learned without the presence of labels. This embedding doesn't solve a particular task itself, but can be fine-tuned (trained on a small labeled dataset) to solve a variety of tasks. This process is especially useful for Twitch, as Twitch has an abundance of stream data that it could otherwise not benefit from without labels. Image Embedding pretrained on Twitch content can be used as a strong backbone for efficiently fine tuning to solve downstream tasks Twitch Machine Learning teams are interested in.

Twitch content is ever-changing; new content is constantly released and old content diminishes. Image Embeddings need to be trained on new streams to account for the changing data distribution. To achieve this, there are two options: 1. Retrain embedding on a combination of old and new data, or 2. Train the old model on new data. Option 1 requires the storage of all old data over time which can be costly and high resource usage as the model needs to be retrained from scratch every iteration. Option 2 solves Option 1's problems but suffers from a complication called catastrophic forgetting.

When a model is first trained on one task, then is sequentially trained on another different task, the model performance on the first task decreases substantially, called catastrophic forgetting. Correspondingly, we will face this problem as we seek to continually learn new Twitch stream content representations, while also retaining prior knowledge. In our experiment setup, the first task is training on a dataset of one time period and the second is training on a dataset of a future time period. Recent research in the field of Supervised Continual Learning has made strides in mitigating this problem and we seek to translate these methods to the Unsupervised setting. More concretely, we seek to continually train an Image embedding in a self-supervised learning fashion on new data collected in set time periods (ex: every year). To the best of our knowledge, we are the first to do unsupervised continual learning to train Image Embeddings.

Method

SimCLR [1] is a contrastive self-supervised learning method. Through this framework, an encoder such as a CNN can be trained on unlabeled image data to learn rich feature representations. The architecture is as follows:

1. For each image in the dataset, make a copy and augment each image in a different manner.
2. Pass both images into same encoder (CNN) and projection head (MLP)
3. The distance between the output feature vectors is minimized in the embedding space, and the distance between these feature vectors and all other feature vectors from images in the batch is maximized by minimizing the NT-Xent [1] loss.

Through this process, the encoder learns to cluster images with similar semantics thus learning effective feature representations of the dataset. Next we explore methods for training an encoder using SimCLR on continuous data.

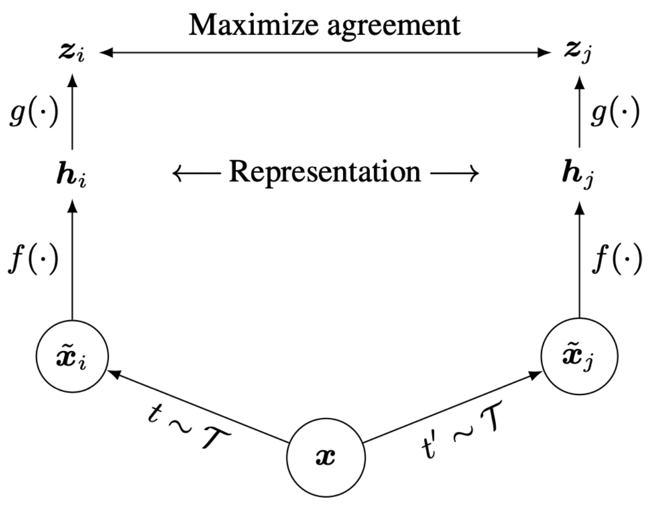


Figure 1: SimCLR [1] Architecture. Image is augmented twice, passed into **f** and **g**, which are trained to minimize the NT-Xent [1] loss of the output feature vectors.

Learning Without Forgetting [2] has achieved significant success in mitigating Catastrophic Forgetting. Its setup is as follows:

1. Train shared encoder on dataset 1 with output head 1
2. Evaluate on dataset 2 with output head 1
3. Train shared encoder on dataset 2 with output head 2.
    a. While training, the goal is to minimize shared encoder model loss on dataset 2, while also penalizing if the model's performance (from step 2) changes when training each step. This process is called Knowledge Distillation, which works to encourage the outputs of one model to approximate that of another. This solves our problem, as we wish for our model to train on the new dataset while at the same time not forget old knowledge (through Knowledge Distillation).

```
LEARNINGWITHOUTFORGETTING:
  Start with:
      $\theta_s$: shared parameters
      $\theta_o$: task specific parameters for each old task
      $X_n, Y_n$: training data and ground truth on the new task
  Initialize:
      $Y_o \leftarrow \mathrm{CNN}(X_n, \theta_s, \theta_o)$    // compute output of old tasks for new data
      $\theta_n \leftarrow \mathrm{RANDINIT}(|\theta_n|)$    // randomly initialize new parameters
  Train:
      Define $\hat{Y}_o \equiv \mathrm{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_o)$    // old task output
      Define $\hat{Y}_n \equiv \mathrm{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_n)$    // new task output
      $\theta_s^*,\ \theta_o^*,\ \theta_n^* \leftarrow \underset{\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n}{\mathrm{argmin}} \left( \lambda_o \mathcal{L}_{old}(Y_o, \hat{Y}_o) + \mathcal{L}_{new}(Y_n, \hat{Y}_n) + \mathcal{R}(\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n) \right)$
```

The *Learning Without Forgetting* paper solves this problem in the context of being trained with supervision (labeled data). We seek to translate their method to an unsupervised task (training Twitch Image Embeddings with unlabeled data). We accomplish this by modifying the Knowledge Distillation portion of the architecture. Knowledge distillation is the process of distilling the knowledge from a “parent” model to a “child” model. When training on labeled data, the student model is trained to match the softmax output distribution of the parent model. Since we don’t have labels, instead of ensuring the outputs of teacher and student models are the same, one might naively consider clustering the embedding space of the parent, and encourage a similar clustering of the student. Upon further thought, this would limit the student’s learning, as learned feature representations aren't defined by the location of feature vectors in the embedding space, but in their relative relationships/similarities with one another. Accordingly, we encourage the similarity/distance of feature vectors in the embedding space of the teacher to be close to that of the student model.

Dataset & Preprocessing

We gather two datasets of unlabeled data containing frames (150k each) from randomly sampled Twitch streams. The first dataset is pulled in the 6 month time frame Jan-June 2020 and the second is sampled from streams one year later in the 6 month time frame July-Dec 2021.

We built a script that takes a csv file obtained from a query of VOD metadata between a specified time period as input, downloads all VODs in csv file, parses them according to user provided specifications (fps, which time periods, etc), and stores them locally, ready to be used to train an Image Embedding. It also stores a corresponding csv file which is required by the Pytorch Dataloader to access the train data. This script can be run, for example, every 6 months to download and preprocess the data required to continually train the embedding.

We also gather two labeled downstream task datasets, one containing images with and without IP violations with the task of detecting IP Violations in an image. The second contains images from 10 different game streams with the task of classifying which game is being played.

For our final experiment, we gather and preprocess the following 2 datasets: ImageNet and CelebA. We reduced the ImageNet dataset size such that each dataset contains ~200k images.

Experimentation

Our goal is to train our Image Embedding continuously on new image data while retaining prior knowledge. We run two different experiments to evaluate the effectiveness of our method in mitigating catastrophic forgetting. For all experiments, we utilize a ResNet-18 as our encoder, SimCLR framework for training our model on the unlabeled datasets, and *Learning Without Forgetting* to mitigate catastrophic forgetting. Time Period 1 dataset is called D1 and Time Period 2 dataset is D2. Each Time Period dataset has 90/10% train/test split.

Experiment 1:

1. Train CNN on D1 (train set) using SimCLR.
2. Evaluate CNN on D1 (test set)
    a. Since our dataset is unlabeled, evaluation is a calculation of SimCLR loss on D1 (test set). This loss is a measurement of similarity of augmentations of the same image and dissimilarity between different images in the same batch. This loss can be used as an indicator of learned feature representations.
3. Train CNN on D2 (train set) using SimCLR both with and without applying *Learning Without Forgetting*.
4. Evaluate CNN on D1 (test set)

We compare the loss produced in step 4 for both when applying LWF and when not.

Table 1: Loss of model on D1 Test set using and without applying LWF method.

	Train D1 Eval D1	Train D1+D2 Eval D1
Regular	46122.72	46085.47
LWF	46122.72	45898.14

Given there is great overlap between our two datasets, catastrophic forgetting is not as clearly apparent. This is because, say only 90-95% of the same semantic data in Time Period 1 is in Time Period 2. As a result, Catastrophic Forgetting will only take place on the 5-10% of non-overlapping content. At the same time, there is 90-95% of overlapping content in both time frames, which means the feature representations of this content in the embedding space only improves with training. As a result, the improvement made by the large overlap is larger than the loss taken by catastrophic forgetting. We see an improvement in evaluation on dataset 1 with our model trained with Learning Without Forgetting applied vs without it applied.

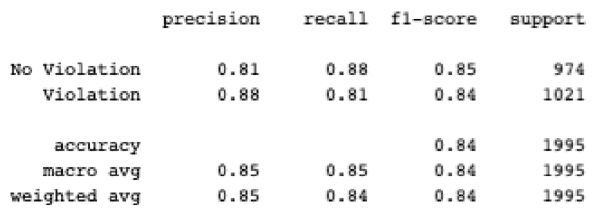
Experiment 2: similar to experiment 1, but instead of evaluating on the unlabeled test sets, we fine-tune on a downstream task. The experiment structure is as follows:

1. Train CNN on D1 (train set) using SimCLR.
2. Freeze CNN, attach MLP, and train/evaluate MLP on downstream task
3. Unfreeze and Train CNN on D2 (train set) using SimCLR both with and without applying *Learning Without Forgetting*.
4. Freeze CNN, attach MLP, and train/evaluate MLP on same downstream task

We compare the difference in performance evaluating on the downstream task from step 4 for both when applying LWF and when not.

Results for Downstream task of IP Violation Detection.

Without LWF

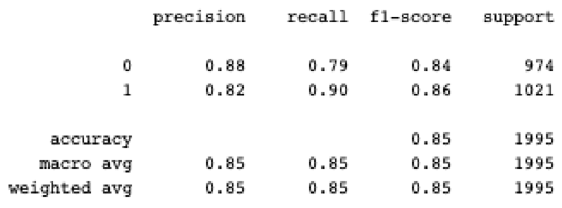


```
              precision    recall  f1-score   support

No Violation       0.81      0.88      0.85       974
   Violation       0.88      0.81      0.84      1021

    accuracy                           0.84      1995
   macro avg       0.85      0.85      0.84      1995
weighted avg       0.85      0.84      0.84      1995
```

With LWF



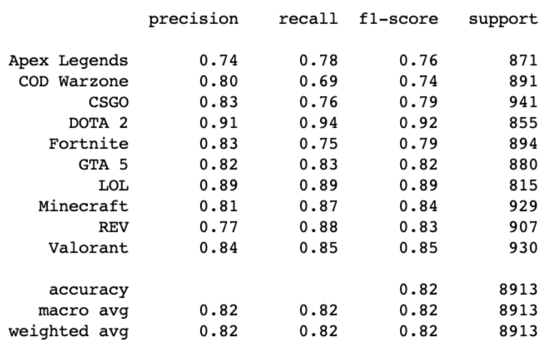
```
              precision    recall  f1-score   support

           0       0.88      0.79      0.84       974
           1       0.82      0.90      0.86      1021

    accuracy                           0.85      1995
   macro avg       0.85      0.85      0.85      1995
weighted avg       0.85      0.85      0.85      1995
```

Results for Downstream task of Game Stream Classification:

Without LWF

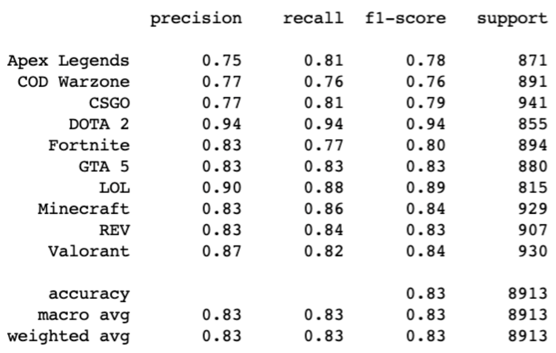


```
              precision    recall  f1-score   support

 Apex Legends       0.74      0.78      0.76       871
  COD Warzone       0.80      0.69      0.74       891
         CSGO       0.83      0.76      0.79       941
       DOTA 2       0.91      0.94      0.92       855
     Fortnite       0.83      0.75      0.79       894
        GTA 5       0.82      0.83      0.82       880
          LOL       0.89      0.89      0.89       815
    Minecraft       0.81      0.87      0.84       929
          REV       0.77      0.88      0.83       907
     Valorant       0.84      0.85      0.85       930

     accuracy                           0.82      8913
    macro avg       0.82      0.82      0.82      8913
 weighted avg       0.82      0.82      0.82      8913
```

With LWF



```
              precision    recall  f1-score   support

 Apex Legends       0.75      0.81      0.78       871
  COD Warzone       0.77      0.76      0.76       891
         CSGO       0.77      0.81      0.79       941
       DOTA 2       0.94      0.94      0.94       855
     Fortnite       0.83      0.77      0.80       894
        GTA 5       0.83      0.83      0.83       880
          LOL       0.90      0.88      0.89       815
    Minecraft       0.83      0.86      0.84       929
          REV       0.83      0.84      0.83       907
     Valorant       0.87      0.82      0.84       930

     accuracy                           0.83      8913
    macro avg       0.83      0.83      0.83      8913
 weighted avg       0.83      0.83      0.83      8913
```

Our results are as we expected. We do not see a significant difference in performance between utilizing LWF and without for our downstream tasks. We hypothesize this is the case because there isn't a large data distribution shift between the streams in the two time periods.

To validate the following hypothesis, we experiment with two unrelated publicly available datasets: ImageNet and CelebA. We follow the same structure as Experiment 1, with ImageNet replacing Time Period 1 and CelebA replacing Time Period 2.

Table 2: Loss of model on ImageNet Test set using and without applying LWF method on the ImageNet & CelebA Datasets.

	Train D1 Eval D1	**Train D1+D2 Eval D1**
Regular	**31856.58**	**39943.75**
LWF	**31856.58**	**39188.78**

As a result, we see 25% catastrophic forgetting when we do not apply our continual learning method. When we do apply LWF, forgetting decreases by 9% to 23%.

These results support our hypothesis that there is a low data distribution shift between the streams in Time Period 1 and Time Period 2 and as a result, Image Embeddings do not need to be updated frequently on Twitch streams. We also prove the validity of our novel unsupervised continual learning method by decreasing catastrophic forgetting by 9%. To our knowledge, we are the first to present an unsupervised continual learning framework.

Note: Training Image Embeddings using unsupervised methods like SimCLR are extremely data hungry and need a LOT of data to learn proper representations. If I had more time in my internship, I would have had a better experiment if I had say 1 million+ images per time period instead of 150k.

References

1. Chen, T., Kornblith, S., Norouzi, M. &amp; Hinton, G.. (2020). A Simple Framework for Contrastive Learning of Visual Representations. ICML

2. Ting Chen, Simon Kornblith, Mohammad Norouzi, Geoffrey Hinton Proceedings of the 37th International Conference on Machine Learning, PMLR 119:1597-1607, 2020.