Computer Vision Techniques for Game Flow Tracking

Sean Yamamoto

December 16, 2022

Abstract

In this project, a transformer neural network is used to track game flow in tennis matches. The pre-trained TimeSformer [BWT21] model is loaded and fine-tuned to classify tennis videos as either "play time" or "dead time". A custom dataset is also constructed to train and evaluate the transformer model. The goal of this project is to automate the process of manually going through tennis match videos and removing the "dead time".

1 Introduction

1.1 Problem Breakdown

The problem can be visualized with Figure 1. We start with a full tennis match. Matches are usually around 1.5 hours long unedited. A tennis match with its dead time taken out is typically around 40 minutes. The second step is to classify the segments of the tennis match as either play time (green) or dead time (red). Segments do not have uniform lengths. Finally, we remove all the dead time from the video and are left with the concatenation of all the play time, leaving us with all the important parts of the video.

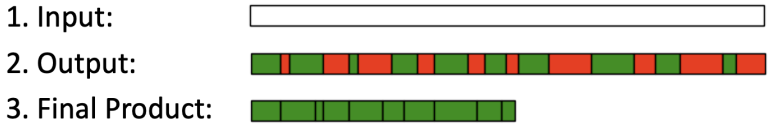

Figure 1: High Level Overview of Classification Problem

1.2 Classification Model

The TimeSformer [BWT21] model is used as the classification tool in this project. The model was published in a paper here and its code can be found here.

The GitHub repository contains pre-trained models on a number of popular video classification datasets. The model used in this project is pre-trained on the Kinetics 600 [CNHZ19] dataset. This dataset is comprised of around 480,000 YouTube videos of people doing different things. This dataset has 600 different classes. Examples of classes are "bodysurfing", "head stand", "pillow fight", and "playing ping pong". The TimeSformer achieved state-of-the-art performance on this dataset with a Top-1 accuracy of 79.1% and a Top-5 accuracy of 94.4%.

The Kinetics 600 TimeSformer model was loaded and then fine-tuned on a custom tennis dataset that will be described in Section 3.

2 TimeSformer

TimeSformer stands for Time-Space Transformer. The model used in this project uses a transformer attention scheme that the TimeSformer authors call "divided space-time attention".

2.1 Patches

Transformers are used to track sequential relationships. In this instance, the elements of the "sequence" are frame patches. Given an input video, the model breaks each of the video's frames into patches. Figure 3 shows the frames of a video broken up into 9 patches each.

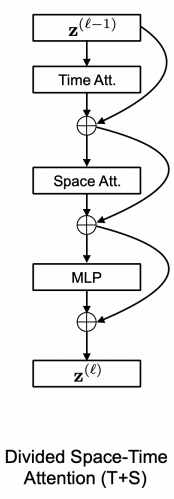

Figure 2: Divided Space-Time Attention Encoder Block [BWT21]

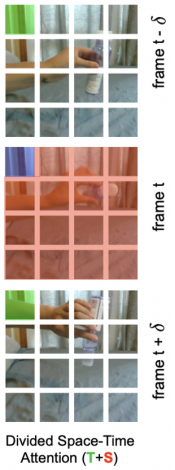

Figure 3: Divided Space-Time Attention Intuition [BWT21]. 3 frames of a video.

2.2 Video Embedding

Calculation Each patch is given an embedding z. The embedding for patch x at position p and timestamp t is calculated using $z_{(p,t)} = Ex_{(p,t)} + e_{(p,t)}$. The same learnable E embedding matrix is applied to each patch. Additionally, a learnable spatio-temporal positional embedding is added to the patch's embedding. This tells the model where the patch is located in its frame (top left, bottom right, etc..) and which timestamp the patch belongs to. All of the embedded patches are concatenated and passed into the model at once.

Reasoning Some of the main reasons transformers use positional encodings is to retain the sequential properties of the data, enable both long and short distance dependencies, and allow all of the "data" to be passed into the model at once, speeding up training and testing of the model.

CLS Token A CLS token is added the the end of the patch embeddings. It has the same functionality as the CLS token used in certain natural language transformer models. This token has the same shape as the embedding of a single patch. It initially contains no information. In the transformer output, the CLS token contains the classification of the video. All the other patches contain information about the input video so the CLS token is a required input so that model has somewhere to store broad information about the video, including its classification.

2.3 Encoder

Overview After the input video is split into patches, each patch is given an embedding z, and the CLS token is added, the embedding fed into the model encoder. The encoder uses attention to encode the video's "meaning" and its output is used to produce a classification.

Encoder Architecture The encoder consists of 16 encoder blocks stacked on top of each other with each encoder's output used as the next encoder's input. Each encoder block has 8 independent self-attention heads that receive the same input but contain different weights. The output of the attention heads is aggregated and used as the next encoder block's input.

Figure 2 the general architecture of an encoder block. Each encoder block is essentially the same and the reason we have so many of the same thing is to create a model with sufficiently many parameters. In general, classifier models with more parameters are more robust since they can learn more complicated decision boundaries.

2.4 Calculating Attention

Attention Overview Transformer encoders work by calculating attention, which track the meaning of input. The attention value of a patch describes the patch's information in the context of its environment. In this model, divided space-time attention has the patches learn their contexts in space and then time.

Divided Space-Time Attention Divided space-time attention is the mechanism described in Figures 2 and 3. Looking at Figure 2 an encoder block input z, the model first calculates time attention, uses its output to calculate space attention, and finally applies a multi-layer perceptron to form its output.

Figure 3 is a visual description of divided space-time attention. To calculate attention for the blue patch b at frame t, we calculate time attention and then space attention. Time attention uses the green patches - these are the patches with the same position as b (top left) but at different timestamps. Space attention for b uses the red patches - these are the patches with the same timestamp as b but at different positions.

Calculation Each patch's attention is calculated using a key, query, and value vector. Since there are 8 self-attention heads in each encoder block, there are 8 key vectors, 8 query vectors, and 8 value vectors for each patch in every encoder block. These key, query, and value vectors are calculated using a key, query, and value matrix and there is one of these matrices for each attention head for every encoder. Exact equations can be found at [BWT21].

3 Dataset

3.1 Source Videos

The source videos for the dataset were 7 different tennis matches, each around 1.5 to 2 hours. Each match was recorded on a different court using a different camera and a different camera angle. There were a total of 10 different players in the 7 matches. Each of the videos are 30 frames per second and have 3 channels (RBG).

3.2 Ground Truth Timestamps

Using the source videos, human-selected play time and dead time timestamps were recorded from each match. A total of 500 ground truth timestamps were recorded.

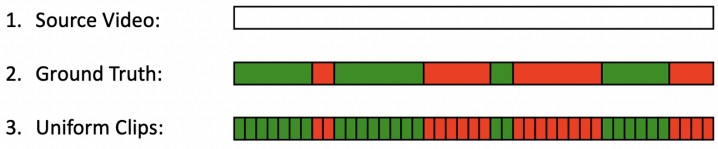

Figure 4: Dataset Creation Pipeline

3.3 Uniform Dataset

Transformers require uniform data input. This means the input videos need to be the same size and length. In this project, the uniform video length was selected to be 88 frames. The exact TimeSformer model used in this project takes in just 8 frames that have a height and width of 244 pixels. The

model's sampling rate is set to $\frac{1}{11}$ so that 8 of the 88 total frames are selected. This allows our videos to cover more time (2.93 seconds) while not expanding the model size.

Type	# Videos	% of Total	Time Span (minutes)
Total	2851	100%	139.38
Play time	864	30.31%	42.24
Dead time	1987	69.69%	97.14

Table 1: Dataset Statistics

3.4 Creating Uniform Dataset

The ground truth timestamps are mapped onto timestamps that have uniform length. This is shown in Figure 4. Timestamps of uniform length are created and then mapped to their corresponding label using the ground truth timestamps. Once all the uniform timestamps have been assigned a label, an *ffmpeg* script is used to cut the source videos at the specified timestamps and corresponding video labels are stored in a .csv file. The final dataset consisted of 2851 uniform .mp4 clips, each 88 frames long, 244 pixels tall and wide, 30 frames per second, and 3 channels. Table 1 contains broad statistics on the uniform dataset.

Figure 5: Data Dilemma Example

3.5 Imperfect Data

There was a notable problem with this data creation. The "overlapping" dilemma is shown in Figure 5. There exists scenarios where a uniform clip contains segments of both play time and dead time. A choice was made to prioritize play time in this situation. It is better to "keep" all the play time and watch some dead time than to eliminate play time. The specific condition used was as follows: if a clip contains at least 6 frames (0.2 seconds) of play time (out of 88 frames/2.93 seconds), label the overlapping clip as play time. Else, label the overlapping clip as dead time. This gives our model a worst-case precision of 2.73 seconds, which is the case where a mostly dead time clip has a tiny bit of play time.

3.6 Final Dataset Notes

Figure 6 shows 4 frames from the final dataset. The frames are 244x244 so their resolution is not great by any means. The ball is not always visible (to the human eye), even during play time. One might assume this will make it very difficult for the model to classify clips since most humans would classify the clips based on what the ball is doing. However, the videos still contain tons of information, such as player movement and positioning. Also, just because the ball is not visible to the human eye does not mean that the model will not be able to "see" the ball since the model sees pixel values, which are much more precise than a human eye.

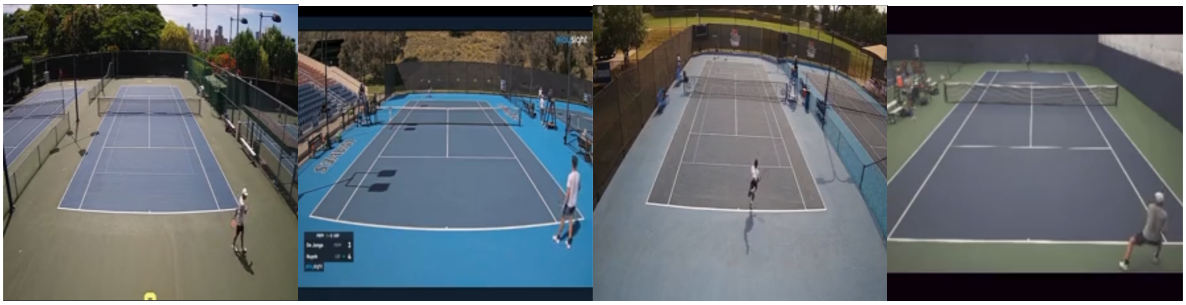

Figure 6: Example 244x244 Frames from Dataset

4 Training

4.1 Pre-Trained Model

The TimeSformer model used 32 Tesla V100 GPUs for Kinetics 600 training. The model required 420 GPU hours to train. Training a model this large is completely unfeasible on a personal laptop. In order to fine-tune the TimeSformer model, resources from Brown Computer Science GridEngine was used.

4.2 Training Method

The final model was trained using two training runs. Each run consisted of 15 training epochs with validation testing every 5 epochs and final training at the very end of training. The TimeSformer trained on Kinetics 600 was used as the starting point for the first model. The final checkpoint of the first train run was used as the starting point for the second run. The two train runs had different train, test, and validation sets but each of the sets were sampled from the same 2851 video dataset. The full dataset was shuffled and redistributed into train, validation, and test sets in between train runs.

4.3 Machine Used for Training

Training for the first run was done on a machine with 8 NVIDIA GTX1080ti GPUs, a Xeon E5-2609, 128GB RAM, 11GB VRAM, and 16 cores. The second run was trained using a machine with 8 NVIDIA GTX 1080ti GPUs, a Xeon Bronze 3106 CPU, 128GB RAM, 11GB VRAM, and 16 cores.

4.4 Training Statistics

Train Run	Runtime	# Total Videos
1	3 hours 8 minutes	2581
2	3 hours 15 minutes	2581

Table 2: Model Train Times

Considering the dataset is 2851 videos, the training times shown in Table 3 were a miracle. Training for just one video on a M1 Macbook Pro took 80 seconds for 1 video. So, it would have taken approximately 945 hours to train on that machine. Brown GridEngine is the reason this model was able to be trained.

5 Results

5.1 Loss

Figures 7 and 8 show the model loss throughout training. Loss in this model was calculated using cross-entropy loss. For the first train run, we see the loss decrease throughout training. For certain

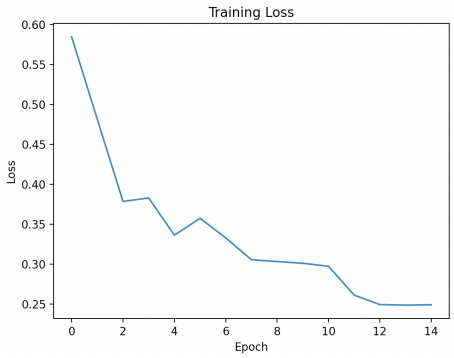

Figure 7: Train Run 1 Loss

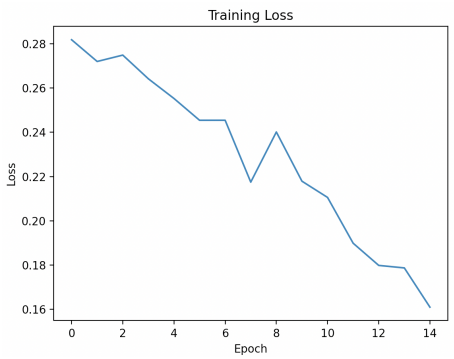

Figure 8: Train Run 2 Loss

epochs, loss increases. The loss resembles an exponential decay function. Loss in the second training run decreases more linearly. It doesn't decrease as much in magnitude.

5.2 Error

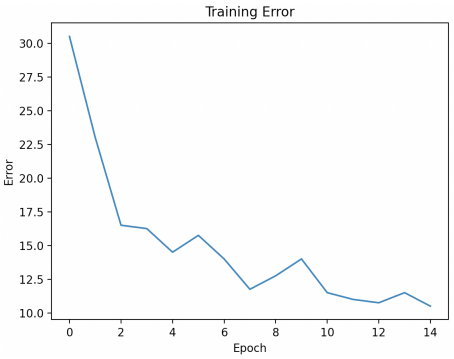

Figure 9: Train Run 1 Error

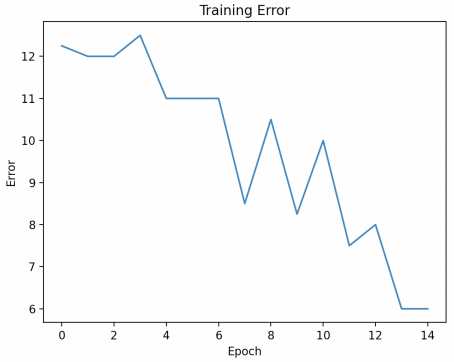

Figure 10: Train Run 2 Error

Training error is calculated using $E = 1 - A$ where A is the training accuracy. The error and loss graphs are highly correlated. Train run 1 has exponential decay while in train run 2, the loss is extremely volatile but does decrease by a significant amount by the end of training.

5.3 Accuracy

Train Run	Accuracy	# Test Videos
1	81.97%	427
2	86.89%	427

Table 3: Model Performance

The first training run achieved an accuracy of 81.97% on the 427 test videos. The second training run achieved an accuracy of 86.89%. The accuracy achieved by this model are quite impressive given the imperfect data it was given. However, this does not mean that the TimeSformer will perform well for its intended task, which is to take out dead time in tennis videos. This is simply a sub-metric of our overall model. This is further explored in 6.1.

6 Future Work

6.1 Model Skepticism

Summary The goal of this project is to automate the process of removing dead time from tennis matches. Right now, the classifier model has been trained to classify 88-frame clips as either play time or dead time. Even though the final model achieved an accuracy of 86.89%, this does not mean it would work well in the overall editing pipeline. It is unknown if the model has been overtrained and overfitted to the limited dataset it has access to.

Volatile Loss and Error The model loss and error in training is highly volatile. This is a bad sign and it can signify that the model is failing to learn suitable classification metrics. Instead, it is possible that the model is just "getting lucky" from iteration to iteration. This volatile performance may also be due to the limited sized dataset used in training or the imperfect data described in 3.5.

Dataset Another aspect to consider is the fact that the dataset was not split evenly between play time and dead time. 30% of the data was play time and 70% of the data was dead time. So, the model could attain high accuracy by labeling most data as dead time and then correctly identifying 50% of the play time, which adds up to $70 + \frac{30}{2} = 85\%$ accuracy, even though half of the play time, which is what we want to conserve, is being falsely classified.

6.2 Evaluation Metrics

In order to ensure that the intended model works well, the model's F1 score should be measured. F1 score F_1 is calculated using precision and recall. Precision P is how often predicted positives are actually positive. When there are no false positives, precision is 1. Recall R is how often actual positives are predicted as positives. When there are no false negatives, recall is 1. A good F1 score is typically greater than 0.8 and a perfect F1 score is 1. The equations for F1, precision, and recall are shown below. TP = true positive. FP = false positive. FN = false negative.

$$F_1 = 2 * P * R/(P + R)$$
$$P = TP/(TP + FP)$$
$$R = TP/(TP + FN)$$

6.3 Final Evaluation

In order to correctly evaluate this model, a new dataset, comprised on matches the model has never seen before, should be made. The videos should be inputted in the order they occur in the match, and the outputs should be analyzed. From there, the sequential results can be compared with the ground truth values.

It is important to figure out what types of videos are being incorrectly classified. If it is the edge videos a.k.a the imperfect data videos, then maybe the model can be used as is. If it is incorrectly classifying segments that are no where near a play time/dead time "edge", then fundamental changes will have to be made to the model.

6.4 Different TimeSformer Models

The TimeSformer pre-trained model used in this project was pre-trained on Kinetics 600. It took in 8 frames from an input video and during training used a sampling rate of $\frac{1}{32}$. The TimeSformer developers also released models that take in more frames. There is a model that takes in 16 frames and uses a sampling rate of $\frac{1}{16}$ and another that takes in 96 frames and uses a sampling rate of $\frac{1}{4}$. These models should be tested on to see if they create better results. There are also models pre-trained on datasets other than Kinetics. Those models can be explored as well.

6.5 Refining Model Outputs

Our model currently has a precision of 2.73 seconds because our video input needs to be uniform in size and length. This precision can be improved using techniques such as temporal saliency detection. Given the video, predicted timestamps of play time and dead time, and the true timestamps, temporal saliency detection can be used to converge the predicted timestamps onto the true timestamps. This technique was not explored in this project but it may be in the future.

6.6 Dataset Improvement

This model was trained on 2851 videos from 7 different matches that were recorded on 7 different cameras with 7 different camera angles. The players in the videos are all high level players who are either current college players or high school players who are on track to play college tennis. In order to make this model more robust, this dataset should be augmented to be as large as possible and include players of different skill levels. However, no matter how many matches are used in the dataset, it is guaranteed that in production this model will be asked to classify videos of courts, players, camera angles, and backgrounds it has never seen.

6.7 Incorporating Audio

One part of the data that was not utilized is the audio in each match. There are definitely audio patterns that occur in tennis matches, whether it be the sound of the ball being hit, a player's grunt, or a player's shoes sliding on the court. If this channel of input was successfully incorporated, the model would likely improve in performance.

7 Conclusion

In this project, a transformer architecture was used to automate the task of removing dead time from tennis videos. The TimeSformer model was fine-tuned on a novel dataset of 2851 88-frame videos from tennis matches. The model was able to attain a testing accuracy of 86.89% on a testing set of 427 videos. However, the loss and error of the model was highly volatile and suggests that further advancements must be made in order to build a sufficient model for this task. The code for building the dataset can be found on GitHub here and the code to train the model can be found here. The .pyth Pytorch models can be found here.

References

[BWT21] Gedas Bertasius, Heng Wang, and Lorenzo Torresani. *Is Space-Time Attention All You Need for Video Understanding?* July 2021.

[CNHZ19] João Carreira, Eric Noland, Chloe Hillier, and Andrew Zisserman. *A Short Note on the Kinetics-700 Human Action Dataset.* July 2019.