Opioid Trafficking Detection on Social Media via Computer Vision Approaches

Author:

Jacob Thrasher

Submitted To

Dr. Xin Li

Computer Science and Electrical Engineering 481
Statler College of Engineering and Mineral Resources
West Virginia University
Morgantown, WV

12/7/2022

Abstract

This report intends to expand upon previously published work, “Detection of Illicit Drug Trafficking Events on Instagram: A Deep Multimodal Multilabel Learning Approach” by analyzing drug trafficking efforts on the social media platform TikTok. As TikTok is a relatively new platform, there are not many robust datasets available, especially those relating to drug habits. We begin by introducing a TikTokScraper Python API that can easily search, download, and label videos from the platform. Furthermore, we apply Vision Transformers to this data in a binary classification task to determine the presence of potentially illegal substances in these videos. It should be noted that the work presented in this report acts as a steppingstone to further exploration of the data in the multimodal space by including audio and text information in future classification tasks.

Contents

1.0 Problem Statement 4
2.0 Background 5
2.1 Transformers 5
2.2 Vision Transformers (ViT) 8
2.3 Video Vision Transformers (ViViT) 9
2.4 Multimodal Data Fusion 11
3.0 Methodology 13
3.1 TikTokScraper Python API 13
3.2 Data Collection 13
3.2.1 CAPTCHA 16
3.3 Models 17
3.3.1 Video Vision Transformer (ViViT) 18
3.3.2 Vision Transformer (ViT) 18
3.4 Training 19
4.0 Results 20
5.0 Future Works 23
References 24

Figures

Figure 1 Transformer architecture 5
Figure 2 Text embedding pipeline 6
Figure 3 Example of self-attention 8
Figure 4 Example of patching mechanism 9
Figure 5 (top) uniform frame sampling (bottom) tubelet embedding method 10
Figure 6 4 encoder implementations for ViViT (top) spatial-temportal attention 11
Figure 7 Common space projection for video-audio-text triplet 12
Figure 8 Example of puzzle piece CAPTCHA used by TikTok 17
Figure 9 Confusion matrices (top) and accuracy over time (bottom) for top performing model 21
Figure 10 Model overfitting 21
Figure 11 Best performing models by learning rate scheduler. (left) stepwise decay scheduler (middle) OneCycle scheduler (right) constant learning rate 22

Tables

Table 1 Label mapping from inital collection effort 14
Table 2 Detailed breakdown from initial collection effort 15
Table 3 Updated label mapping 16
Table 4 Optimal hyperparameter configuration of ViT 20

1.0 Problem Statement

Social media connects people seamlessly and instantly over the internet. This is great for individuals to stay in touch over long distances, but simultaneously acts as a medium for bad actors to abuse its anonymous nature for illegal activities. In recent years, there has been a surge of illicit drug trafficking efforts in the form of opioids, hallucinogens, and other narcotics, as demonstrated in [4]. These platforms should be a safe place for its users, so it is vital to have a mechanism that can automatically detect and report these actors. We aim to develop a robust multimodal model to consider vision, audio, and text information to make a final, accurate, classification.

2.0 Background

2.1 Transformers

Transformers were initially introduced by Google in 2017 [8] as a sequence-to-sequence learning model to improve upon existing methods such as recurrent neural networks (RNNs) and long short-term memory (LSTM). It was originally created for the purpose of natural language processing, namely language translation, but can be modified to add support for other modalities such as image processing, discussed in **2.2 Vision Transformers.**

The transformer works by first embedding an input sequence into a high dimensional domain. It then encodes the embedded input using multi-headed self-attention and a simple feedforward neural network. This step can be thought of as translating the input sequence into a "language" the model can understand. From here, the encoded input can either be passed into a multi-layer perceptron (MLP) for classification tasks such as sentiment analysis or it can be fed through a decoder network for sequence translation. In this study, we are primarily focused on classification, so we will not discuss the decoder in detail. **Figure 1** provides an overview of the encoder/decoder architecture, where the encoder is pictured on the left and the decoder on the right.

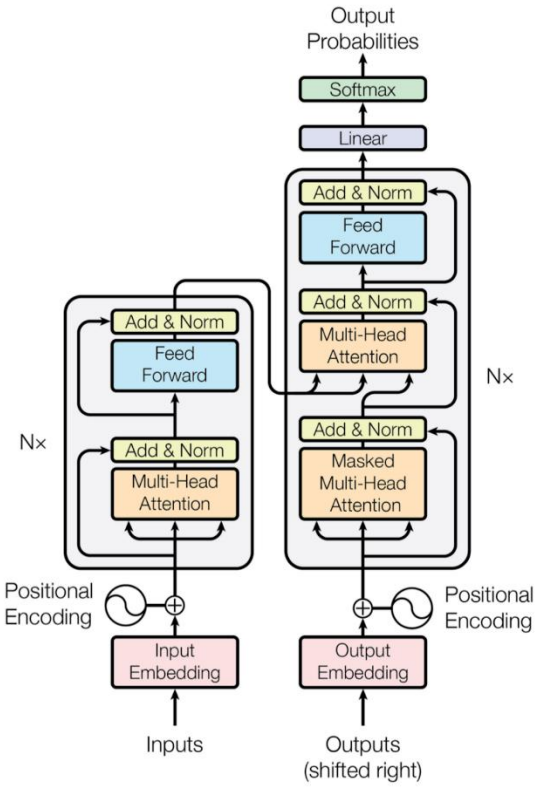

Figure 1 Transformer architecture

The first step to a transformer task is input embedding, which is fully illustrated in **Figure 2.** Consider a sentence in plain English such as “The animal did not cross the street because it was too tired”. As humans, this is an easy sentence to understand, but the computer does not speak English. The first step to input embedding is to translate the input sequence into a language the computer understands: numbers. We begin by creating a vocabulary which acts dictionary that maps every unique word in the entire dataset to a number. The English words in the input are simply replaced by their corresponding number. This process is called *tokenization*, and each word in the sequence is a *token*.

Now the sequence is in a language the computer understands. Even so, the tokens have no meaning. The next step is to embed the inputs. This process uses a simple linear layer that maps each token in the vocabulary to a vector with a length equal to predetermined *hidden dimension*. The hidden dimension is a hyperparameter that defines the embedded dimension space. The use of a linear layer for embedding allows the model to learn a given token’s meaning over the course of training.

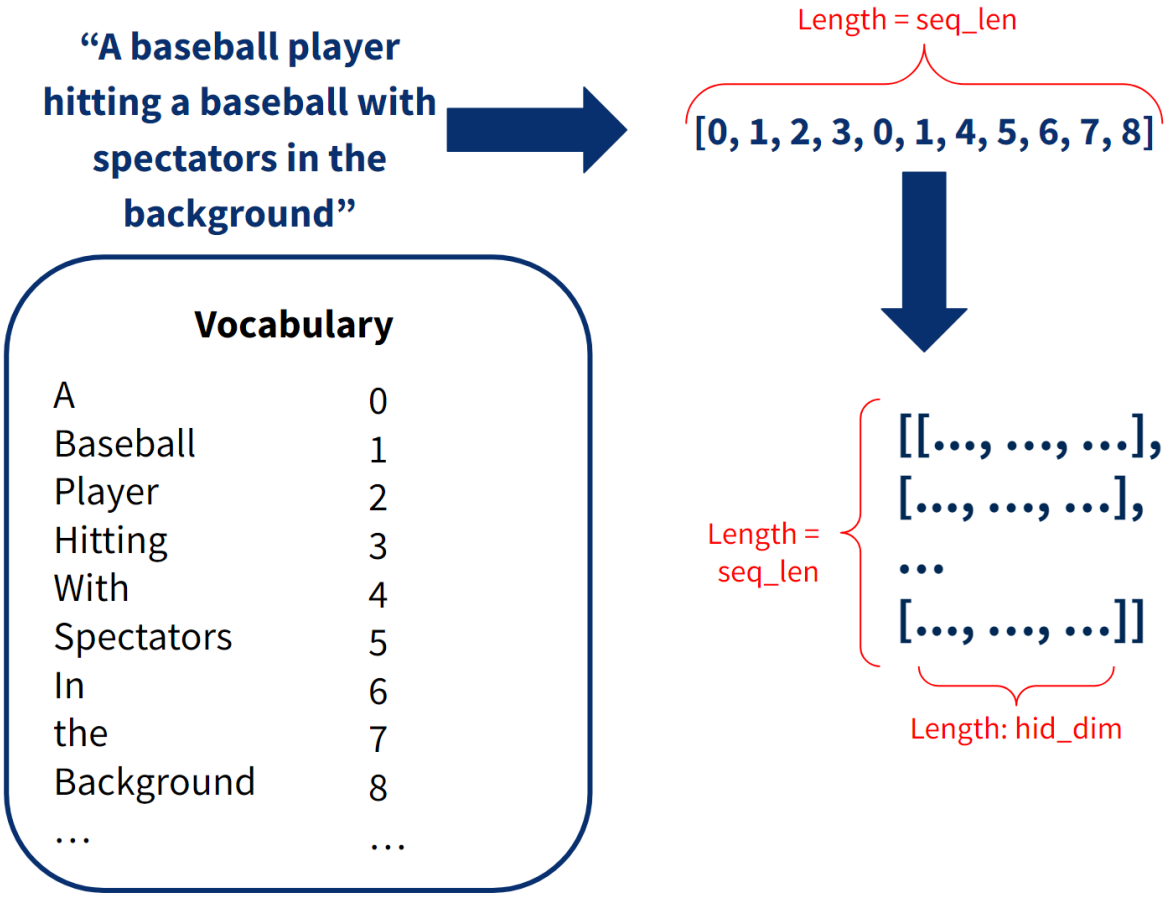

Figure 2 Text embedding pipeline

At this point, the sequence is ready to be fed into the encoder block. One advantage of transformers vs older methods such as recurrent neural networks is that the entire sequence is fed into the network as one large matrix, rather than looking at it one token at time. This helps speed up the process by avoiding redundant tasks but has the adverse side effect of losing positional awareness. Since the entire sequence is analyzed in one step, there is nothing distinguishing a sentence such as “the animal was tired” from “tried the animal was” or any of its other permutations. To account for this, a positional encoding can be added to the sequence to hard code positional awareness using the following functions:

$$P(k, 2i) = \sin\left(\frac{k}{n^{2i/d}}\right)$$

$$P(k, 2i+1) = \cos\left(\frac{k}{n^{2i/d}}\right)$$

Where:

- k = position within the sequence $0 < k < L/2$
- L = Sequence length
- d = hidden dimension
- i = mapping for column indices in range $0 < I < d/2$
- n = user define scaler (default 10000)

The transformer encoder block uses self-attention to learn token correlation within a sequence. Consider again the sentence, “the animal did not cross the street because it was too tired”. The word “it” clearly refers to the animal. However, “it” is a pronoun, which takes the place of a noun, and this sentence contains two separate nouns: “animal” and “street”. It may be obvious to humans that “it” refers to the animal, but it is not so clear to a computer. Moreover, how should the computer know “it” is a pronoun at all and not a third noun of equal importance, or a verb, adjective, etc.? This is the goal of self-attention. This layer in the encoder analyzes each token in a sequence relative to the rest of the tokens to determine which ones are most highly correlated.

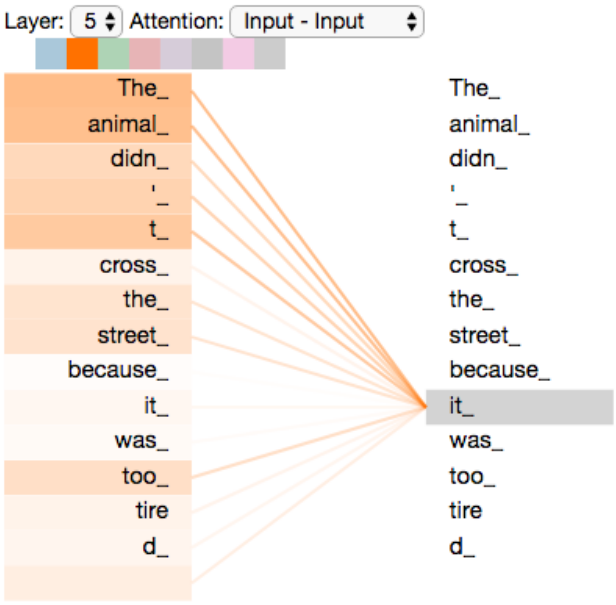

Figure 3 Example of self-attention

Each token in the input sequence is passed through three individual linear layers, generating *Query (Q), Key (K),* and *Value (V)* vectors. Each token's Q vector is multiplied by each other token's K vector via dot product multiplication. Mathematically, given a sequence S, each token S_i generates a list $L_i = \{S_i * S_j \mid \forall S_j \in S\}$ containing scalar outputs. Softmax activation is then applied to L_i which generates a series of scores for each token. As Softmax scales all values into a range [0, 1], irrelevant tokens are left with an extremely low value such as 0.0001 while highly correlated tokens will have a value closer to 1. These scores are multiplied by the corresponding V vectors to drown out the values of irrelevant words. Finally, the weighted sum of each V vector is returned as the self-attention score.

Residual connections are added to the self-attention scores and then passed through a simple feed forward network, adding the residuals yet again for stability. This process repeated N times, where N is a hyperparameter defined by the user. In a typical implementation, N can range from 3-12 encoder layers. The final output of the transformer encoded is passed to an MLP head, which handles the final classification task using standard methods.

2.2 Vision Transformers (ViT)

Since input sequences are cast to a high dimensional domain before being given to the transformer layers, extending the model to support other modalities such as vision is relatively simple. Vision Transformers (ViT) mostly operate identically to the standard implementation. The only difference is the manner with which input sequences are embedded. The tokenization method discussed in **2.1 Transformers** will not work here, so an analog was developed by [2] to address this issue.

We begin by dividing the input image into square patches with a dimension p. This generates $\frac{w*h}{p^2}$ patches, where w and h correspond to the image width and height respectively. These patches can be considered the image tokens, and the number of patches will be the sequence length. The tokens are embedded by flattening each patch and feeding them through a fully connected layer, much like the aforementioned embedding method. Finally, positional embedding can be added in the same manner.

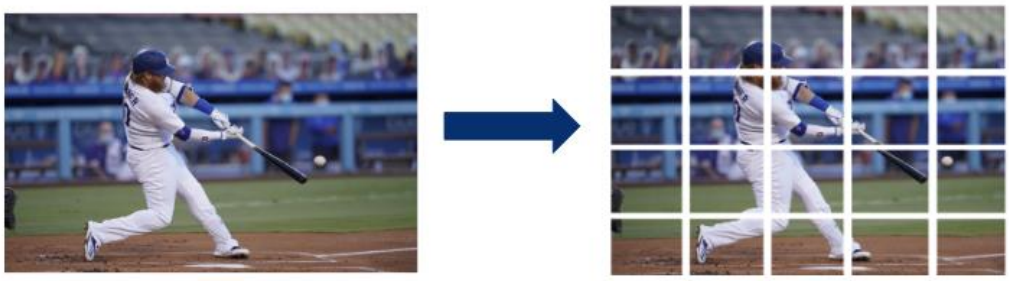

Figure 4 Example of patching mechanism

At this point, the embedded tokens function identically to the language ones from before. Therefore, the encoder layer can function identically. Attention is computed the same way, but instead of locating important words in a sentence, it highlights important elements in the image such as the subject vs the background.

2.3 Video Vision Transformers (ViViT)

We can further expand on the ViT architecture by supporting video inputs. Videos in this context can be thought of as a collection of images, which represent the spatial dimension, but there is an added temporal dimension that should be considered. As such, video data are loaded as four dimensional objects in the shape [T, C, W, H], where T, C, W, and H represent the time, channel, width, and height dimensions respectively. The added T dimension complicates both the embedding and encoding tasks. [3] introduces various techniques to address each of these issues.

There are two proposed methods to tokenize video inputs: uniform frame sampling and tubelet embedding. To reduce computational complexity, both methods begin by sampling n_t frames from the video. Most frames are redundant as there is very little difference between an image at time $t = i$ and $t = i + 1$. Sampling a fraction of the total frames drastically saves on computational cost and memory allocation without sacrificing much information.

Unform frame sampling essentially treats the 4D video as one large 3D image by combining the temporal domain to one token. Each of the n_t sampled frames are patched via the method introduced in **2.2 Vision Transformers (ViT)**. This generates $n_w * n_h$ tokens per frame. Each of these tokens are simply concatenated with the corresponding tokens from the remaining frames. The result is a total of $n_t * n_w * n_h$ tokens to be passed to the transformer encoder.

Tubelet embedding can be thought of as a 3D extension to the standard 2D patching technique. This is accomplished by defining spatio-temporal "tubes". Given a tube dimension of $t \times h \times w$, $n_t = \left\lfloor \frac{T}{t} \right\rfloor$, $n_h = \left\lfloor \frac{H}{h} \right\rfloor$, and $n_w = \left\lfloor \frac{W}{w} \right\rfloor$ tokens are generated from the temporal, height, and width respectively. A visual demonstration of each of these embedding techniques can be view in **Figure 5.**

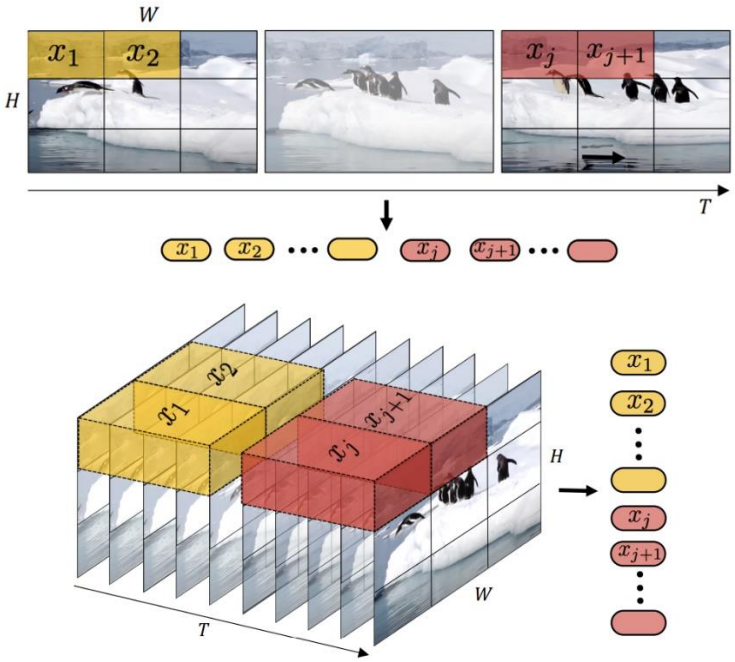

Figure 5 (top) uniform frame sampling (bottom) tubelet embedding method

The transformer encoder must be able to account for both spatial and temporal information when performing self-attention. As such, four models were proposed by [3] to solve this issue. **Figure 6** illustrates each of these approaches. The most intuitive approach is *spatial-temporal attention* whereby the spatial-temporal tokens generated previously are simply passed through the transformer. During the attention phase, the transformer models all pair-wise interactions between all tokens. This is problematic as Multi-headed self-attention has a time complexity of $\mathcal{O}((n_t * n_h * n_w)^2)$ making it very inefficient for video attention.

The second model, *factorized encoder*, segments the spatial and temporal domains by creating an individual encoder for each. The first encoder fully models the spatial interactions for each temporal index. Once the spatial interactions are fully computed, the temporal information across each frame in the same spatial location are modeled. This approach is much more efficient than method 1 as it was a complexity of $\mathcal{O}((n_h * n_w)^2 + n_t^2)$.

Model 3, *Factorized self-attention*, models both spatial and temporal interactions within the same encoding step, rather than fully modeling spatial, then temporal information. The

encoder models interactions that occur on frames within the same temporal index, then models interactions that occur within across all frames in the same spatial index. This is accomplished by reshaping the input tokens from $\mathbb{R}^{1\times n_t * n_h * n_w * d}$ to $\mathbb{R}^{n_t \times n_h * n_w * d}$ for spatial attention and $\mathbb{R}^{n_h * n_w \times n_t * d}$ for temporal attention, where *d* represents the hidden dimension. It should be noted that there is no difference in performance between spatial-then-temporal and temporal-then-spatial implementations. The downside of this implementation is the number of parameters is increased compared to Model 1 since there are two attention layers per transformer block.

The final model, *factorized dot-product attention*, is functionally the same as Model 3, but rather than reshaping the tokens for two individual attention layers, the multi-headed self-attention layer itself is modified to compute the spatial and temporal attention weights using different heads. The advantage to this approach is that it achieves the same computational complexity of Models 2 and 3, while maintaining the same number of parameters as Model 1.

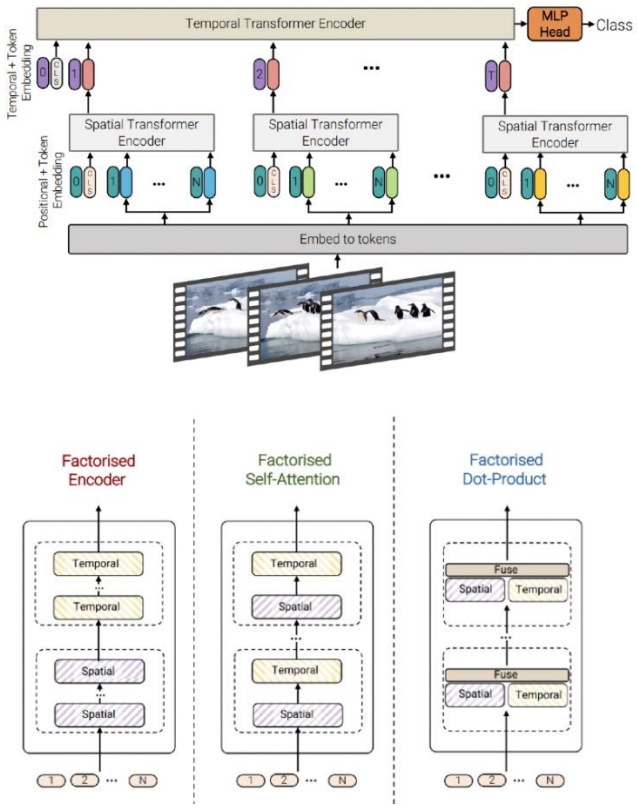

Figure 6 4 encoder implementations for ViViT (top) spatial-temportal attention

2.4 Multi-modal Data Fusion

Videos themselves contain two modalities: vision and audio. Additionally, there is frequently important text data that serves to contribute extra information about the video contents. These three forms of data are all stored in very different ways, so in order to make use

of each one simultaneously, we must first project them into a common space. [1] introduces a common space projection solution that allows video, audio, and text data to be fused in transformer applications.

The video and text information are embedded via the previously discussed methods. Audio data are treated as a one-dimensional vector of length T' and is patched into $\left\lfloor \frac{T'}{t'} \right\rfloor$ segments of length t'. These patches are encoded by normal methods. Common space token projection is defined as follows:

$$\boldsymbol{z_{v,va} = g_{v\to va}\left(z_{out}^{video}\right),\ z_{a,va} = g_{a\to va}\left(z_{out}^{audio}\right)}$$

$$\boldsymbol{z_{t,vt} = g_{t\to vt}\left(z_{out}^{text}\right),\ z_{v,vt} = g_{v\to vt}\left(z_{v,va}\right)}$$

Here, $g_{x\to y}$ is a linear projection head that maps input x to the common space S_y. This acts as a hierarchical system whereby video-audio pairs are projected to a common space first, then fused with text information for the final mapping. The intuition for this stems from the idea that different modalities possess different levels of semantic granularity. Therefore, this is imposed as an "inductive bias" in the projections [4]. **Figure 7** demonstrates this behavior in application.

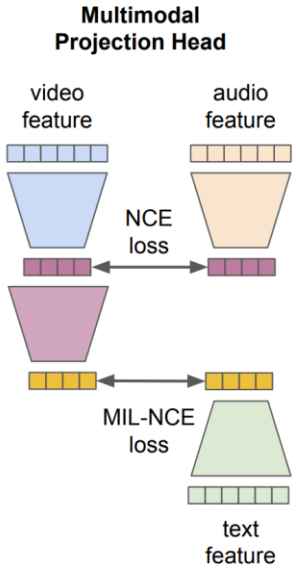

Figure 7 Common space projection for video-audio-text triplet

3.0 Methodology

3.1 TikTokScraper Python API

TikTok is a relatively young platform that has seen incredible growth in the past years. As such, there are few robust datasets available – especially in the domain of drug use. The TikTokScraper API aims to address both problems simultaneously. It is a flexible API that can navigate the TikTok website autonomously, giving the API user the power to search for TikTok users, hashtags, and other relevant terms in mass with ease. This flexibility allows us to specifically target searches related to drug use while allowing others to use the API for their own collection purposes. There are two main tools used to search, store, and save TikTok videos: TikTokScraper and Discover.

The TikTokScraper class is a barebones search tool that deploys Selenium for Python to implement the following functions for website navigation: search, switch_tab, get_videos, goto_user, goto_video, goto_tag, download_video, login, bypass_captcha, solve_puzzle, quit. This class is intended for basic website navigation and bulk video collection. This can be a very slow task up from, so a lightweight Discover class was implemented for quicker data collection.

The Discover class is a lightweight alternative to the original TikTokScraper class that collects and save video links instead of the metadata and mp4 downloads. The purpose of this class is to collect links in one quick run which can later be used to collect metadata as needed.

Metadata is stored in two object classes: Video and User. Deploying these classes on the correct page automatically collects metadata from the video and user respectively. These objects simply store necessary data for future use and implement a pretty print method.

3.2 Data Collection

A comprehensive list of common drug terms from the Instagram trafficking project was provided which included a total of 131 unique tags. To develop an ideal dataset, as many videos related to each tag must be parsed as possible, so the provided terms were divided into 20 batches and annotated over the course of one month.

The data collection and annotation loop employs a combination of the pure TikTokScraper class and the Discover class to parse and annotate data. First a given batch of hashtags is provided to the Discover class which systematically performs a search for each tag. Selenium is only able to collect the videos that are visible on screen, so Discover attempts to load 8 “pages” before collecting links. TikTok loads new videos when the user scrolls to the bottom of the screen, so here “pages” refers to the number of times new videos are loaded by this method. The links for each video are loaded into a set, which helps filter out duplicates between tag searches, and finally saved to a .txt file.

After this is accomplished, the user is left with a list of a few hundred links that must be manually viewed and annotated. This is done using the TikTokScraper class. Each video is loaded by its link, then the user is prompted to provide a label according to a labelling scheme.

Once entered, the metadata is collected and saved to a csv, and the video is downloaded from TikTok and uploaded to a Google Drive folder. TikTokScraper automatically advances to the next video. The metadata that was collected during this process includes: video link, username, user bio, video description, hashtags, mentions, and level 1 comments. “Level 1 comments” refers to all comments that are not replies. In total, 7890 videos were manually reviewed and annotated. A full breakdown can be viewed in the table below. It should be noted that the labelling scheme was later changed, and more videos were downloaded, so this breakdown is not representative of the final dataset.

Labels	
1	Active illegal drug user
2	Drug dealer
3	Video about substance abuse recovery
4	Educational video about drugs
5	Legal drug use

Table 1 Label mapping from inital collection effort

	Labels							
Current count - From folders:	**1**	**2**	**3**	**4**	**5**	Total positive	Total searched (approx)	% positive
(0) All tags	27	3	28	35	31	124	600	20.67%
(1) 100packs to 8oz	5	1	1	0	1	8	300	2.67%
(2) acetylene to activis	29	6	0	1	0	36	380	9.47%
(3) adderall to adds	0	1	0	14	8	23	300	7.67%
(4) Ambien - az	12	0	3	2	6	23	290	7.93%
(5) benzo to benzos	3	1	31	18	5	58	300	19.33%
(6) clonazepam to diazapam	18	1	5	19	2	45	200	22.50%
(7) downers to fourbars	2	0	7	7	0	16	230	6.96%
(8) heroin to hydros	0	0	0	1	0	1	200	0.50%
(9) klonazapem to mdma	8	0	17	16	12	53	260	20.38%
(10) meth to musclerelaxers	3	0	3	5	0	11	380	2.89%
(11) narcos to opiates	4	0	23	25	3	55	550	10.00%
(12) oxy to oxycoti	0	0	0	6	0	6	400	1.50%
(13) painkiller to painmeds	1	0	3	15	9	28	400	7.00%
(14) perc to perks	8	0	4	4	2	18	800	2.25%
(15) phx to promethazin	5	2	0	8	17	32	600	5.33%
(16) promethazine to psil	18	8	1	4	7	38	300	12.67%
(17) rolls to roxys	0	0	3	0	0	3	390	0.77%
(18) sellmeth to trazadone	0	0	0	2	10	12	200	6.00%
(19) uppers to vikes	0	0	7	11	0	18	400	4.50%
(20) xanax to zoloft	2	0	6	9	5	22	410	5.37%
Totals	120	13	125	168	79	505	7890	6.40%

Table 2 Detailed breakdown from initial collection effort

From **Tables 1** and **2**, the initial collection efforts yielded 13 videos of suspected drug dealers, accounting for only 2.5% of the total drug related video sample. It can be reasonably inferred that a suspected drug dealer might post more than one trafficking video to their account, so an additional method was implemented to investigate users that were deemed suspicious. This method automatically downloads all videos from the selected user's account and saves them to the dataset. In err, it was assumed that all videos from these accounts would be related to drug

trafficking, but upon manual review only a fraction were. I took this opportunity to develop a more robust labelling system and manually relabeled all videos in the dataset.

Label	Content	Number samples
1	Suspected Drug Dealer	88
2	Drug present in video - cannot verify legality	132
3	Drug present in video - Likely legal	106
4	Substance abuse recovery	97
5	Educational Content	177
6	Storytime video (Not about recovery)	113
7	User actively high (both legally and illegally)	19

Table 3 Updated label mapping

The final TikTok dataset was relatively small. Transformer networks require a substantial amount of data to perform effectively. To account for this, a sample of Instagram data was added to the dataset. Additionally, an online image generation model, DALL-E Mini was used to generate synthetic data to be added to the final dataset. For training flexibility, each type of data (TikTok, Instagram, DALL-E Synthetic) were stored in separate folders that can be loaded individually in the dataset class. This approach added 2,738 Instagram and 50 synthetic elements to the dataset.

3.2.1 CAPTCHA

TikTok uses a CAPTCHA puzzle system to deter bots from crawling its website. CAPTCHAs occur irregularly, but occasionally pop-up during searches. Fortunately, the TikTok uses a puzzle piece captcha, which can be easily defeated using Selenium and OpenCV. Puzzle piece CAPTCHAs are solved by dragging a puzzle piece horizontally to the correct spot in the provided picture using a slider at the bottom of the window as seen in **Figure 8** below. The image and solution location vary with each CAPTCHA, so a flexible algorithm must be developed to account for all possible solutions.

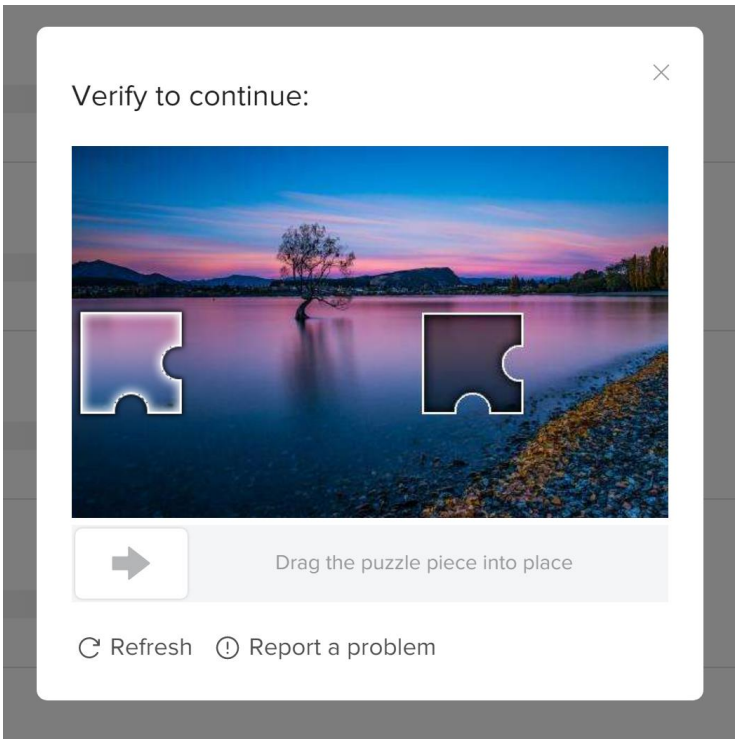

Figure 8 Example of puzzle piece CAPTCHA used by TikTok

The approach is relatively simple, but very effective. First, the provided image is downloaded and converted to greyscale. The puzzle piece that the user slides is a separate element, so the downloaded image will only contain the picture and solution square. It can be observed that the solution square is outlined by a bright white border. This white happens to be the brightest possible with a value of 255. It is reasonable to assume that a majority of the image will have a brightness value less than 255, so each pixel that does not have a value of 255 is converted to 0, leaving only an outline of the solution square's border. From here, we can iterate across the image, summing the value of the columns. If the value is above a given threshold, we take the column index and save it as the solution square's left bound. The approximate center of the solution square is calculated and returned. Selenium then uses this value to move the slider to the calculated location.

3.3 Models

The collected TikTok data is multimodal by nature, including video, audio, and text information. All these modes must be considered to make inferences on each of the aforementioned class labels. As an intermediary step, only the video mode was considered. A video classifier alone cannot accurately distinguish between legal and illegal drug use, as well as the different types of non-drug related videos that appear in the dataset. Instead, two vision models were built on a binary classification task: *drug present* and *drug absent.*

3.3.1 Video Vision Transformer

The binary classification task of identifying drug presence does not require audio or text information, so ViViT can be implemented as an intermediary step. Once it is fully functional and accurate, expanding to the VATT implemented from [1] is relatively easy.

For this experiment, Models 1 and 3 from [3] were implemented using the tubelet embedding technique to tokenize the inputs. Due to memory limitations, videos were down sampled severely to a spatial size of 64x128, sampling 32 frames from the temporal dimension. The inputs were patched into tubes of size [16, 16, 16] for experiments conducted on model 3. To simulate model 1, we simply set the temporal dimension of the patches equal to that of the down sampled video, which treated the entire input as one large 3D image. We also implement the Drop Token technique proposed by [1] by randomly dropping a fraction of the input tokens to avoid redundancies and reduce computational complexity.

Transformers take a massive amount of data to train from scratch. Fine-tuning techniques allows us to bypass this undertaking by initializing the layers with weights from a pre-trained model. This technique of "transfer learning" gives the network a head start since it will not need to learn very low level details such as shapes and color. Instead, it can focus on learning the task at hand. In this implementation, ViViT was initialized via the weights from a pretrained ViT model. This approach works without issue for Model 1 as it contains an identical transformer encoder block to ViT, but fails on Model 3 since the encoder is segmented into spatial and temporal attention sections. When training on based on Model 3, we instead initialize the spatial attention block on ViViT to the pretrained attention block provided by ViT. Then, the temporal attention section is simply initialized to 0.

Initializing ViViT from the pretrained ViT model can help reduce training time and improve performance, but it can have the adverse size effect of allowing the model to easily overfit the data. To accommodate this, *n* layer initialization was implemented to only initialize the first *n* layers of ViViT with the pretrained weights [9]. For example, if ViViT has a total of 12 encoder layers, but $n = 6$, only the first 6 encoder layers will be initialized with the weights from ViT. The remaining layers are left with random initialization. This approach gives us the ability to fine tune the pretraining stage for more flexibility.

3.3.2 Vision Transformer

Upon manual inspection of the positive samples of the data (i.e. the ones which contain drug samples), it was discovered that videos that contain drug use will almost always have the drug as the primary subject throughout the entire video. ViViT was introduced as an action recognition model, but here, there is no action to detect. We only care about the presence of the drug in the video and if it occurs at any one frame, the entire video is positive regardless of any other factors.

With this fact in mind, a simpler Vision Transformer model was implemented. We randomly sample one frame from each video during loading to pass through the model. In theory, this approach will drastically reduce training time, as inputs will be a single image, rather than an entire video. Pretraining was accomplished identically, but without the need to accommodate different types of models.

Lastly, as the dataset is very small, we implemented few-shot learning techniques from [5]. Shifted Patch Tokenization (SPT) attempts to improve the process of embedding by allowing each individual patch to "see" more of the input image simultaneously. The entire image is first shifted by half the patch size in all four diagonal directions: left-up, right-up, left-down, and right-down. This shifting system is notated by $\mathcal{S}$ in the original paper. The four images generated by $\mathcal{S}$ are then cropped to match the original image dimension and concatenated to the input. The resulting object is tokenized much like the unform frame sampling method from [3] and passed to the transformer.

The second few-short learning technique implemented by [5] is Locality Self-Attention (LSA). LSA is comprised of two components: diagonal masking and temperature scaling. First, input tokens are masked on the main diagonal to prevent the model from performing self-token attention. During the attention phase, obviously each token is going to be most highly correlated with itself, so diagonal masking prevents this self-token self-attention setting the values of the main diagonal equal to negative infinity. This is followed by a learnable temperature parameter, which allows the model to determine the Softmax temperature on its own during training.

3.4 Training

The biggest hurdle in training was the lack of data. As mentioned in the previously, transformers take a massive amount of data to train well. The TikTok collection efforts from **3.1 Data Collection** ultimately led to 732 unique TikTok samples, which is far fewer than adequate for a transformer task. The first approach to solve this problem was a random augmentation pipeline in which the user defines an *n_passes* parameter in the dataset class, which corresponds to the number of times each data element should be loaded. For example, *n_passes = 2* doubles the total number of elements using the random augmentation pipeline. This pipeline consisted of the following transformations: random horizontal flip, random color jitter, random invert, random rotation, and random reverse. Here, random reverse simply reverses the order of the frames to reverse the video.

At this point, we still did not feel as though we had enough data, so we incorporated Instagram data collected by [4] as well as synthetic data generated by DALL-E mini to further expand the size of the dataset, resulting in a total of 3,520 unique data elements.

4.0 Results

A total of 30 experiments were conducted, comparing different hyperparameter configurations, optimizers, model types, and learning rate schedulers. It was found during testing that optimal results were attained by training the ViT model with SPT and LSA using Adam optimization and a constant learning rate with the following hyperparameter configuration:

Hyperparameter	Value
Learning rate	0.0008
Batch size	64
Weight decay	0.001
n_layer initialization	6
Dropout	0.2
Video dim	64x128x1
Patch dim	16x16
n_passes	2
Epochs	100

Table 4 Optimal hyperparameter configuration of ViT

This configuration yielded a final average accuracy of 70% of the validation set without overfitting the training data. As 50% is the baseline for random guessing on a binary classification task, 70% accuracy is subpar performance. However, given the very limited data available and considering the minimum requirements for adequate transformer performance, this serves as a valuable proof of concept that can be expanded upon in future work. The figure below outlines the accuracy over time for the training and validation sets as well as the final confusion matrices.

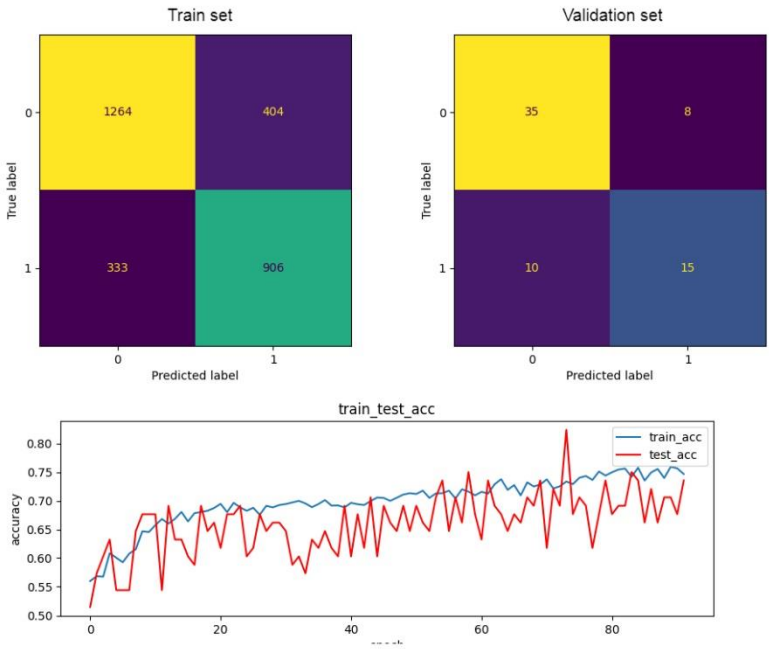

Figure 9 Confusion matrices (top) and accuracy over time (bottom) for top performing model

The shortage of data provided little diversity for the model to generalize. Training for more than 50-100 epochs resulted in severe overfitting of the training data as evident by **Figure 10**. Expanding the dataset with Instagram and synthetic data and random augmentation did not prove to resolve this issue. Few shot learning techniques such as SPT and LSA did, however, help to improve the overfitting, but were not enough to completely eliminate the issue. **5.0 Future works** elaborates on possible methods to alleviate the problem.

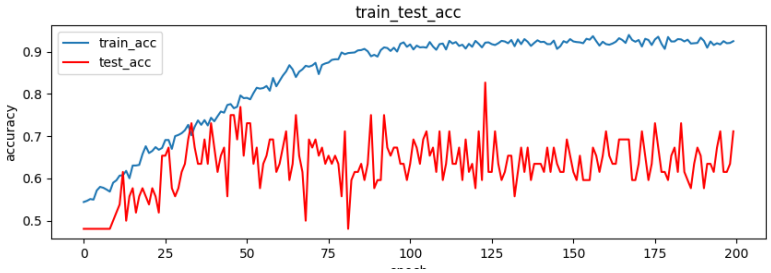

Figure 10 Model overfitting

Optimizer choice turned out to be an important factor in improving the model's performance as well. During experimentation, we found that Adam optimization provided much more stability and reduced overfitting. SGD optimization, on the other hand, allowed the model to collapse very quickly. We also found that ViT outperformed both ViViT Model 1 and Model 3 because it eliminates irrelevant and redundant information from the video, allowing the model

to focus only on the primary subject. Finally, we found that learning rate schedulers have the potential to stabilize performance with the right configuration, but unfortunately, we did not find a configuration that wholly outperformed a constant learning rate. **Figure 11** details the best performing networks using stepwise decay, OneCycle, and constant learning rate scheduling algorithms.

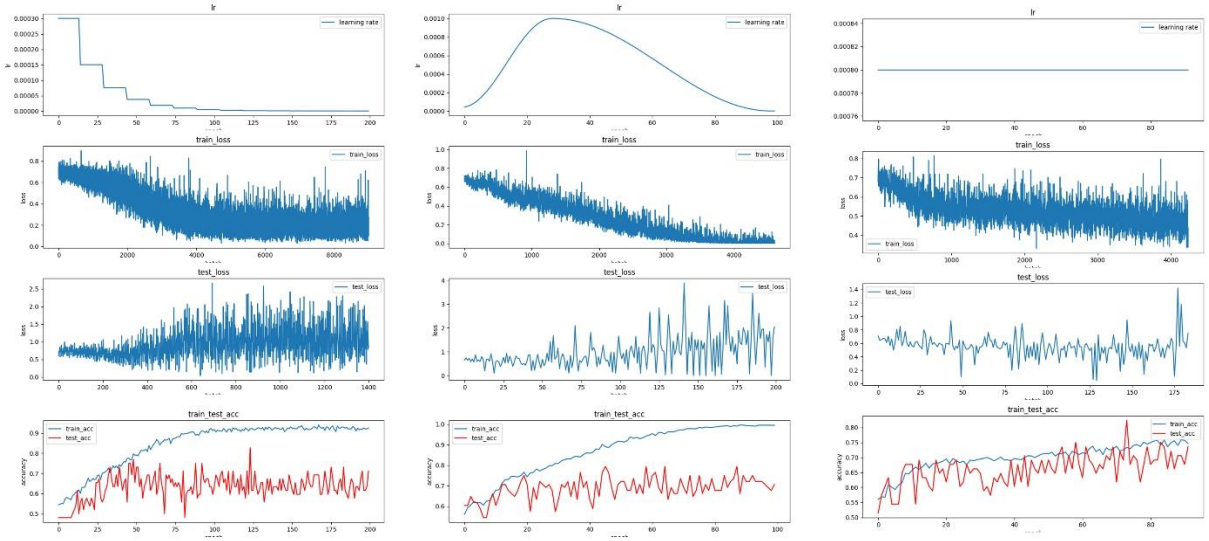

Figure 11 Best performing models by learning rate scheduler. (left) stepwise decay scheduler (middle) OneCycle scheduler (right) constant learning rate

5.0 Future Works

The results from the binary classification experiments were lackluster but serve as a functional proof of concept. We believe much better performance could be achieved with a much larger and more robust dataset collected by a small team of researchers. At the time of collection, DALL-E mini was the most advanced image generation model available and produced subpar images that did not serve to improve performance. Additionally, there was no API available to automatically generate and save images to your device, making synthetic data collection a slow and arduous process. Since then, however, OpenAI has released the DALL-E 2 API to the public, which could serve to generate much more high fidelity images at a quicker pace. In the event that the dataset must be expanded again with synthetic image, the newly available generation techniques could make that process much more painless.

In the future, we would like to expand the ViViT model to a multimodal approach which includes audio and text data in its predictions. Audio is an important aspect of all videos, but more importantly, video descriptions, user bios, and comments are arguably the most vital pieces of information to accurately classify illicit drug trafficking on the app. [1] provides a great starting point for this process.

At this time, it is uncertain whether the single frame sampling technique proposed in **3.3.2 Vision Transformer** is sufficient in accurately detecting drug presence in videos. More testing is required to make a definitive decision on the topic. During this comparison, we would like to implement an *m* frame sampling technique, where we sample *m* frames from an input video to be treated as unique instances in training. This could potentially increase the size of the dataset while maintaining sufficient variability across samples.

References

[1] H. Akbari et al, "VATT: Transformers for Multimodal Self-Supervised Learning from Raw, Audio, Video, and Text", *35th Conference on Neural Information Processing Systems*, Apr. 2021. Doi: 2104.11178. [Online]. Available: https://arxiv.org/abs/2104.11178

[2] A. Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", *International Conference for Deep Learning Representations 2021*, Sept. 2020. Doi: 2010.11929. [Online]. Available: https://arxiv.org/abs/2010.11929

[3] A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lucic, C. Schmid, "ViViT: A Video Vision Transformer", *Proceedings of the IEEE/CVF International Conference on Computer Vision 2021*, Mar. 2021. Doi: 2103.15691. [Online]. Available: https://arxiv.org/abs/2103.15691

[4] C. Hu, M. Yin, B. Liu, X. Li, "Detection of Illicit Drug Trafficking Events on Instagram: A Deep Multimodal Multilabel Learning Approach", *International Conference on Information and Knowledge Management 2021*, Nov. 2021, Doi: 2108.08920, [Online]. Available: https://arxiv.org/abs/2108.08920

[5] S. Lee, S. Lee, B. Song, "Vision Transformer for Small Datasets", Dec. 2021, Doi: 2112.13492. [Online]. Available: https://arxiv.org/abs/2112.13492

[6] E. Cubuk, B. Zoph, J. Shlens, Q. Le, "RandAugment: Practical automated data augmentation with reduced search space". *Advances in Neural Information Processing Systems 33*. Sep. 2019. Doi: 1909.13719. [Online]. Available: https://arxiv.org/abs/1909.13719

[7] Maxime, "What is a Transformer", *Medium*, Jan. 2019. [Online]. Available: https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04

[8] A. Vaswani et al, "Attention is All You Need", *Proceedings of the 31st International Conference on Neural Information Processing Systems*, Dec. 2017. Doi: 10.5555/3295222.3295349. [Online]. Available: https://dl.acm.org/doi/10.5555/3295222.3295349

[9] P. Chang, "Advanced Techniques for Fine-tuning Transformers", *towardsdatascience*, Sep. 2021. [Online]. Available: https://towardsdatascience.com/advanced-techniques-for-fine-tuning-transformers-82e4e61e16e