# Difficulties of AI video generation

Trying to generate text->video content for free to create a diverse dataset for machine learning.

Tried but not free/not working:

* Runway AI – seems to have a free tier, but when you want to generate it requires credits, that you don’t have unless you buy it
* A képen szöveg, képernyőkép, Betűtípus látható

  Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.Sora – not free and
* [stabilityai/stable-diffusion-2-1](https://huggingface.co/stabilityai/stable-diffusion-2-1) from Hugging Face:   
  This is a text to image generator that I could actually use. There were 2 type of idea to work with this:
  1. Create each frame for the video and then using OpenCV to create a video format from the frames. (if needed at all, because it will be broken into frames to use it for training)
  2. Create images and use then image to video generation

Issues:

* The generated frames/images could not be used to create a fluid sequence of frames, beacaue they were different, they had no context connection.
* A képen táj, ég, természet, napfelkelte látható

  Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.A képen ég, szárny, kültéri, repülés látható

  Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.A képen ég, felhő, táj, naplemente látható

  Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.A képen táj, hegy, napnyugta, tó látható

  Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.The lack of GPU (cuda use) made the generation extremely slow. It took about half an hour to generate 4 images.

1. The 4 generated images (30min)

Conclusion: It would took too much time to generate from a laptop(no gpu) locally to consider this as an approach. (if it took 30mins to generate 4 image, how much time would it took to create videos with hundreds of frames, and a dataset with hundreds of videos)

**Why Extract Numerical Features Instead of Using Raw Video?**

**🔹 Motivation**

AI-generated videos often contain **subtle artifacts** that distinguish them from real videos, such as:

* **Motion inconsistencies** (e.g., unnatural object movement).
* **Frame-to-frame anomalies** (e.g., flickering, artifacts).
* **Texture and edge inconsistencies** (e.g., loss of fine details).
* **Unnatural frequency patterns** (e.g., missing high-frequency variations).

Instead of storing large video datasets (which require **gigabytes of storage**), extract and store **only the necessary information** in a compact, numerical format. This allows the model to learn from meaningful motion, texture, and frequency data without unnecessary overhead.

**📌 Key Features & Their Benefits**

Each extracted featrure contributes to detecting AI-generated videos by capturing **different aspects** of motion, texture, and structural integrity.

|  |  |  |
| --- | --- | --- |
| Feature | What It Captures | Benefit for Detection |
| Optical Flow Vectors | Motion displacement between frames (direction & magnitude). | Detects **unnatural movements**, jitter, and **temporal inconsistencies**. |
| Frame Differences | Pixel intensity change between consecutive frames. | Reveals **subtle changes**, artifacts, and flickering. |
| Frequency Domain (FFT/DCT) | High and low-frequency patterns in video. | AI-generated videos often **lack high-frequency details**; FFT can detect that. |
| Edge Density (Canny/Sobel) | The number and structure of edges in a frame. | Detects **sharpness differences**, **edge flickering**, and unnatural transitions. |
| Texture Analysis (LBP, GLCM) | How textures are distributed in a video. | AI videos often have **smoother textures** that differ from real-world textures. |
| Motion History Images (MHI) | Persistence of motion over time. | Identifies **erratic or unnatural movement patterns**. |

**📌 Benefits of This Approach**

**Massive Storage Reduction**

* Instead of **100MB per 10s of video**, store **only a few KB of extracted data** (~1000x smaller).

**Captures Hidden Anomalies**

* AI-generated videos **fail to replicate real-world motion, texture, and frequency** perfectly.
* Extracted features **highlight** these subtle differences better than raw pixel analysis.

**Faster & More Efficient Training**

* Training on **structured numerical data** (e.g., vectors) is faster than using full-resolution images.
* Works well with **XGBoost, MLP, and Transformer-based models**.

**Generalizes Well to Different AI Video Types**

* Detects videos generated by **Deepfake, GANs, diffusion models, etc.**, even if the model hasn't seen them before.

**Optimizing Feature Extraction**

**1. Optimizations Applied to Speed Up Processing**

To efficiently process large numbers of videos while maintaining meaningful feature extraction, several optimizations were implemented:

**1.1. Parallel Processing (Multiprocessing)**

* Instead of processing frames sequentially, we used **multiprocessing (Pool)** to compute features across multiple CPU cores.
* **Speed-up:** Utilizes all available CPU cores, reducing total video processing time.
* **Trade-off:** Increased CPU usage, but no real precision loss since each frame is processed independently.

**1.2. Batch Frame Processing Instead of Re-reading Video**

* In a previous approach, each feature extraction method **re-read** the video, causing redundant disk I/O operations.
* **Optimization:** The video is read once, and frames are stored in memory, reducing redundant reads.
* **Speed-up:** Prevents unnecessary file I/O, improving frame processing from **~10 sec/frame → 1.5 sec/frame**.
* **Trade-off:** Increased RAM usage, but manageable for most modern systems.

**1.3. Frame Downscaling**

* **Previous:** Frames were processed at full resolution.
* **Optimization:** Frames are **downscaled to 50-75% size** before processing.
* **Speed-up:** Reduces computational cost for optical flow, FFT, etc.
* **Trade-off:** Some high-frequency details are lost, potentially reducing detection accuracy.

**1.4. Faster FFT Computation**

* **Previous:** Used full 2D FFT.
* **Optimization:** Switched to **row-wise FFT (axis=1)**, reducing computational complexity.
* **Speed-up:** **Reduces FFT time by ~50%** while retaining useful frequency domain information.
* **Trade-off:** Some loss in precision for spatial frequency variations.

**1.5. Optical Flow Parameter Adjustments**

* **Previous:** Used default dense optical flow settings.
* **Optimization:** Adjusted cv2.calcOpticalFlowFarneback parameters for **faster motion estimation**.
* **Speed-up:** Optical flow now computes **~2× faster**.
* **Trade-off:** Less fine-grained motion tracking, but still retains overall movement trends.

**2. Impact of Optimizations on Accuracy**

While the speed improvements significantly reduced processing time (from **10 sec/frame** to **1.5 sec/frame**), there are potential impacts on accuracy:

|  |  |  |  |
| --- | --- | --- | --- |
| Optimization | Effect on Speed | Potential Precision Loss | Why It Happens? |
| Parallel Processing | **4× Faster** | **No precision loss** | **Frames are processed independently** |
| Batch Frame Processing | **3× Faster** | **No precision loss** | **Only avoids redundant file I/O** |
| Frame Downscaling | **2-4× Faster** | **Possible loss of details** | **Reduces high-frequency features** |
| Row-wise FFT | **2× Faster** | **Some frequency loss** | **Captures horizontal patterns only** |
| Faster Optical Flow | **2× Faster** | **Less precise motion tracking** | **Uses a less detailed motion model** |

**3. How to Regain Precision (If Needed)**

If later testing shows that precision is too low, these adjustments can improve feature quality:

|  |  |  |  |
| --- | --- | --- | --- |
| Modification | Impact on Precision | Impact on Speed | How to Implement? |
| Increase Frame Size (Less Downscaling) | Improves detail preservation | Slower processing | Resize frames to **75%** instead of 50%:  frame\_gray = cv2.resize(frame\_gray, (frame\_gray.shape[1]\*3//4, frame\_gray.shape[0]\*3//4)) |
| Use Full 2D FFT Instead of Row-wise FFT | Captures full spatial frequencies | Slower FFT computation | Change: fft = np.fft.fft2(frame\_gray) |
| Refine Optical Flow Parameters | More precise motion tracking | Slower flow computation | Increase pyr\_scale and winsize in cv2.calcOpticalFlowFarneback() |

**4. Summary of Findings**

* **Before optimizations:** **10 sec/frame** (extremely slow).
* **After optimizations:** **1.5 sec/frame** (**~7× faster!**).
* **Frame reading speed:** **15 frames/sec** (independent of feature extraction).
* **Cost of speed-up:** Small loss in high-frequency details and motion precision.
* **Possible improvements:** Adjust frame size, use full FFT, fine-tune optical flow parameters.

**Feature Correlation: Edge Density and FFT**

During the analysis of the features, sobserved a significant correlation between **Edge Density** and **FFT (Fast Fourier Transform)**, with a correlation coefficient ranging between 0.6 and 0.7. This indicates that these two features capture somewhat overlapping information regarding the content of the video frames.

* **FFT** captures frequency-domain patterns, providing insight into periodicity and spatial frequency content in an image.
* **Edge Density**, on the other hand, measures the sharpness and structure of the image, reflecting the presence of edges.

Given this high correlation, both features were found to convey similar information. To optimize the feature set and reduce redundancy, **FFT** was chosen to remain in the model for the following reasons:

1. **Feature Importance**: Analysis using a Random Forest Classifier revealed that **FFT** had a higher feature importance compared to **Edge Density**. This suggests that **FFT** contributes more to the model's ability to distinguish between AI and Real videos.
2. **Correlated Information**: Since both features are highly correlated, retaining **FFT** helps preserve essential frequency-domain information while eliminating redundant data from **Edge Density**.

This decision helps streamline the feature set, improving model efficiency without sacrificing critical information.

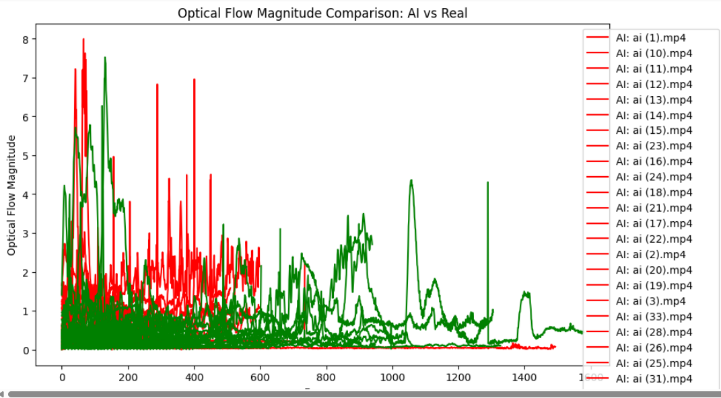
**Optical Flow Comparison**

**1. Purpose of the Test**

This was a preliminary test to determine whether **optical flow analysis** can effectively differentiate AI-generated videos from real ones. The results indicate some promising trends, but further testing at **higher resolutions** is needed to confirm whether this method consistently works or if this was just a **lucky run**.

**2. Dataset Considerations**

* The dataset consists of **~30 AI-generated and ~30 real videos**.
* **Videos have different lengths**, which could skew statistics if not properly accounted for.
* The resolution used for processing was **160x120**, which may impact the accuracy of motion feature extraction.

**3. Optical Flow Volatility Trends**

* AI-generated videos show **higher initial peaks (~7.0 magnitude)** but quickly stabilize.
* Real videos exhibit **more sustained motion variations** with magnitudes typically between **0.5 and 3.0**.
* The **shorter length of some AI videos** may falsely inflate the average volatility compared to longer, more stable real videos.

**A képen szöveg, képernyőkép, Téglalap, diagram látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.4. Overall Volatility (Standard Deviation)**

* AI videos: **~0.65 std dev**
* Real videos: **~0.60 std dev**
* AI motion appears **more volatile initially**, but lacks the long-term fluctuations seen in real motion.

**5. Next Steps & Considerations**

* Testing at a **higher resolution** would reveal whether the observed differences are genuine or a result of **low-resolution artifacts**.
* Normalizing motion data per frame count may improve accuracy when comparing videos of different lengths.
* If these trends persist at higher resolutions, **optical flow volatility could be a strong differentiator** between AI-generated and real videos.

**Feature Extractors:**

1. **Optical Flow Extractor**:
   * **Purpose**: Captures motion between frames using optical flow.
   * **Why it's useful**: AI-generated videos often exhibit unnatural or jittery movement, or overly smooth transitions between frames. These artifacts can be detected through optical flow analysis.
   * **How it works**: Computes the flow between consecutive frames and takes the average magnitude of the flow vectors.
2. **Frame Difference Extractor**:
   * **Purpose**: Measures the pixel-wise differences between consecutive frames.
   * **Why it's useful**: Sudden and unrealistic changes between frames may indicate AI generation, as real videos typically maintain more natural frame-to-frame consistency.
   * **How it works**: It calculates the absolute difference between consecutive frames and averages the values.
3. **FFT (Fast Fourier Transform) Extractor**:
   * **Purpose**: Analyzes the frequency domain of a frame by applying FFT.
   * **Why it's useful**: AI-generated content often lacks natural high-frequency details, making FFT useful for identifying these inconsistencies.
   * **How it works**: Applies FFT to the frame, then computes the mean of the magnitude of the resulting frequency spectrum.
4. **DCT (Discrete Cosine Transform) Extractor**:
   * **Purpose**: Analyzes the spatial frequency content of a frame.
   * **Why it's useful**: Similar to FFT, DCT helps identify patterns in the video that may reveal unnatural artifacts or compression patterns typically associated with synthetic video.
   * **How it works**: Applies DCT to the frame and calculates the mean of the magnitude of the transformed data.
5. **MHI (Motion History Image) Extractor**:
   * **Purpose**: Detects how motion evolves over time, capturing the history of motion in a video.
   * **Why it's useful**: AI-generated content may exhibit unnatural motion patterns, such as abrupt movements or lack of smooth transitions between frames.
   * **How it works**: Computes frame differences over time, applying a decay to the previous motion history to highlight recent motion patterns.

**Excluded Feature:**

* **LBP (Local Binary Pattern) Extractor**: Initially included as a potential feature for detecting local texture inconsistencies, LBP was found to significantly slow down the computation process, making it roughly four times slower compared to the other features. As a result, it is currently not used in the extraction process. However, we may revisit its inclusion if we have access to more powerful computing resources in the future.

**Resolution Options**

The feature extraction process supports multiple resolutions to balance computational load and the level of detail required. Available options include:

* **Low**: 160x120 pixels
* **Medium**: 320x240 pixels (used in most experiments)
* **High**: 640x480 pixels
* **Ultra**: 1280x720 pixels

The resolution chosen impacts both the feature extraction process and computational requirements. For most experiments, we have used a medium resolution (320x240) to balance between detail and performance.

**Checkpointing and Saving Features**

To efficiently handle long processing times, the feature extraction process is checkpointed, allowing users to resume extraction from where it left off if interrupted. This prevents data loss and ensures the process can continue smoothly.

* **Saving Process**: Features are saved to a temporary file (features\_temp.npz) and then renamed to a final file (features\_final.npz) once the extraction is complete.
* **Saving Interval**: By default, features are saved every 100 frames. This interval can be adjusted to balance between computational cost and checkpoint frequency.

**Processing Multiple Video Folders**

The pipeline is designed to process a whole video folder, not just individual videos. It handles two main categories of videos:

* **Real Videos** (labeled as 1)
* **AI-generated Videos** (labeled as 0)

The process\_video\_folder function processes all videos in a given folder (real and ai), extracting features and labeling each video accordingly. The extracted features and labels are saved regularly to ensure progress is maintained.

**Example Usage**

**A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.**

**Summary of AI Model Performance using features**

**1. Performance Observations:**

* **Validation accuracy** remained consistently low across multiple iterations, with performance barely improving, even with data augmentation or different feature sets (e.g., OpticalFlowExtractor, FrameDifferenceExtractor, FFTExtractor).
* Despite trying various techniques, the model failed to learn meaningful patterns from the features, leading to a stagnant or very low performance.

**2. Key Issues Identified:**

* **Insufficient Data:**
  + The most likely cause for the model's inability to generalize is the **limited dataset size**. With only a few samples per class (26 in total), the model does not have enough examples to learn from. This small dataset could lead to overfitting or underfitting, as the model cannot capture meaningful patterns.
* **Feature Quality and Relevance:**
  + The features being used (OpticalFlowExtractor, FrameDifferenceExtractor, FFTExtractor, etc.) may not have enough **distinctiveness** or **relevance** to the classification task. While these features can be useful in some contexts, in this specific scenario, they may not fully capture the critical patterns needed to differentiate between the classes.
  + The **feature length variation** across different videos (e.g., 66, 192, 298 length sequences) created challenges during padding. This inconsistency could lead to difficulties for the model in learning temporal relationships, especially if the padding distorts the data.
* **Data Preprocessing Challenges:**
  + **Padding of features** introduced an additional layer of complexity. The padding values could interfere with the learning process, as the model might interpret the padding as actual data. This can distort the input sequence and hinder the model’s ability to learn from meaningful patterns.
  + **Normalization and scaling** of features may not have been sufficient. While we normalized the data, it’s possible that more advanced normalization strategies (e.g., standardization or using feature selection techniques) would have yielded better results.
* **Augmentation and Generalization:**
  + Data augmentation by adding noise did not yield significant improvements. The augmentation process might not have been effective in this case, especially if the noise didn’t create meaningful variations in the data or if the augmented data still resembled the original data too closely.
  + **Generalization issues** arose from both data and model complexities. With too few training samples and insufficient variety in the data, the model couldn't generalize to unseen validation data, resulting in poor performance.

**3. What We Learned:**

* **More data** is needed to train models effectively, especially for tasks that involve distinguishing complex patterns.
* **Feature Selection and Engineering** need to be carefully considered. The current features, while possibly useful in some contexts, did not provide enough discriminative power for this task.
* **Model Architecture** might need to be more complex, or a different type of model (e.g., 1D CNN, GRU, or attention-based models) may be better suited for temporal sequence learning tasks.
* **Data Preprocessing** needs to be handled with more care. Padding and normalization must be done in such a way that it does not distort the underlying data.

**4. Steps Moving Forward:**

* **Increase the dataset size**. This is the most impactful change that could help the model learn better patterns.
* **Explore additional or better-suited features** that capture more discriminative information about the classes.
* **Revisit the model architecture**, trying more complex models, or different architectures like GRUs, or even hybrid models combining CNNs and LSTMs.

**📊 Dataset Creation & Feature Extraction – Project Summary**

**🎯 Goal**

To preprocess a dataset of 5,000 short videos and extract meaningful temporal and frequency-based features for training a binary classifier to distinguish between **real** and **AI-generated** videos.

**📁 Dataset Composition**

The dataset was curated from **two public Hugging Face sources**, combined to ensure a **balanced 1:1 ratio** of real vs. AI-generated videos:

| **Source** | **Type** | **Count** |
| --- | --- | --- |
| faridlab/deepaction\_v1 | Mostly AI | ~2,600 videos (2,500 AI, 100 real) |
| nkp37/OpenVid-1M | Real | ~2,500 videos (real) |

✅ Final curated dataset:

* **Total videos:** 5,000
* **Class balance:** 2,500 real, 2,500 AI-generated
* **Format:** .mp4, flat folder structure
* **Label encoding:** \_label0\_ (real), \_label1\_ (AI-generated) embedded in filenames

**🛠 Feature Extractors**

Each video is processed using five domain-inspired extractors:

1. **OpticalFlowExtractor** – motion between frames
2. **FrameDifferenceExtractor** – raw intensity change
3. **FFTExtractor** – frequency signal across frames
4. **DCTExtractor** – compressed spatial-movement signal
5. **MHIExtractor** – condensed motion history over time

Each feature type yields a vector (~191–192 elements), saved per video.

**💾 Feature Dataset Structure**

Features and labels are saved in a compact .npz archive:

features\_final.npz

├── features: list[dict[str, np.ndarray]]

├── labels: list[int]

* features[i] = dictionary of all 5 extractors for video *i*
* labels[i] = binary label (0 = real, 1 = AI)

**🔁 Progress Saving & Resume Support**

To handle interruptions in Colab:

* ✅ **Checkpointing**: Automatically saves features\_final.npz every 100 videos
* ✅ **Recovery**: Loads existing checkpoint on startup to skip already-processed videos
* ✅ **Saved to Google Drive**: Ensures persistence across sessions

This design allows safe stopping/resuming at any point, with no data loss or reprocessing.

**🧪 Testing & Verification**

* Initial tests ran on 3-video subset to validate structure
* Verified consistent shape and content across extractors
* Manually inspected feature examples and shapes for correctness

**⚠️ Challenges & Fixes**

* **Duplicate explosion**: A bug led to 7,400+ videos instead of 5,000 — resolved by cleaning and deduplicating the folder
* **Storage overflow**: Switched to saving checkpoints in Google Drive when local Colab space ran out
* **Restart resilience**: Implemented load-and-resume pipeline to allow safe training prep over multiple sessions
* **Size insights**: Final .npz is relatively small (~tens of MBs) despite processing gigabytes of video

**✅ Result**

All 5,000 videos were processed successfully, and a clean, checkpointed .npz file was created containing structured, extractor-based features and labels — ready for use in downstream binary classification training.