Edge Intelligence DA-19.02.2026

D. Sai Mohith, 25MML0032

Video Classification:

Detailed Code implementation and explanation:

Using a deep learning model that has already been trained, the application is intended to classify videos. Instead of starting from zero when training a new network, it makes advantage of a model that has previously gained knowledge from a sizable dataset. The basic concept behind the implementation is straightforward: a movie is just a series of pictures, or frames. A strong image model may be used to extract and classify each frame independently, resulting in a composite output that can reflect the video's overall content. This method uses information about picture categorisation to comprehend videos.

The first step in the implementation is to import the required libraries. Reading frames from a video file is one of the video processing operations that are handled by the OpenCV library. Working with picture data requires the use of NumPy for array manipulation and numerical calculations. MobileNetV2 provides the pre-trained model, while TensorFlow serves as the deep learning framework. ImageNet has previously been used to train this model, and it comprises millions of labelled photos from various object categories. The model already understands how to recognise common items like cars, tools, furniture, animals, and many more because of this pre-training.

When weights='imagenet' is used to initialise the model, it indicates that previously learnt weights are being loaded. This makes it possible to utilise it for prediction right away and does away with the requirement for additional training. The model requires input photos to have a certain size—224 by 224 pixels—and in a format that corresponds to the way it was trained.

Frame extraction from a video file is the responsibility of the program's initial function. To access the video, use cv2.VideoCapture(). This object examines the video file frame per frame on the inside. A check is done to make sure the video file has been successfully opened before continuing. An empty array is returned if the file is not accessible. This stops the software from crashing later.

A property flag is used to determine the total number of frames in the video. Videos can include hundreds or thousands of frames, so processing them all would be needless and computationally costly. Consequently, a step size is determined. Only equally spaced frames are chosen depending on the maximum number requested (20 by default) if the movie contains a large number of frames. This guarantees that all of the video's frames are sampled consistently, collecting material from start to finish rather than just the opening few seconds.

Every frame is read in turn within the loop. The loop ends if the frame reading is unsuccessful. A frame goes through preprocessing once it satisfies the sampling requirement. Because MobileNetV2 requires that set input size, it is first scaled to 224 × 224 pixels. After that, a colour conversion from BGR to RGB is carried out. Since most deep learning models (including MobileNetV2) demand photos to be in RGB format, but OpenCV reads images in BGR format, this step is crucial. Without this conversion, the colour channels would be misread, resulting in inaccurate forecasts.

A list of the processed frames is created till the maximum frame limit is achieved. To free up system resources, the video capture object is removed after extraction is finished. The list of frames is then transformed into a NumPy array, the format needed for input in deep learning.

The work of categorisation is completed by the second function. **First, the function mentioned above is used to extract frames. If the function returns no frames**, it does so securely. The frames go through a preprocessing function otherwise. In this preprocessing, pixel values are scaled to match the model's initial training. Matching the training settings is essential for accurate results since neural networks are sensitive to input scale.

The model then makes predictions for each extracted frame. The model produces a prediction vector for every frame since several frames are analysed collectively. Since **ImageNet has 1000 classes**, each vector provides probabilities for 1000 potential object categorisation. The implementation averages the prediction probability across all frames rather than choosing the forecast from a single frame. A more stable outcome is produced by smoothing out noisy forecasts through this averaging procedure. When paired with other frames, for instance, an unclear or blurry frame's inaccurate prediction will have less of an impact.

The **probabilities are averaged**, and then the decoding utility converts them into labels. They retrieve the top five anticipated categories. The confidence percentage and label name are provided for each of these categories. The final prediction for the video is shown as the category with the highest likelihood.

Overall, by **transforming the video into representative frames, the application applies an image classification model to the video data**. Instead of using temporal modelling methods like transformers or recurrent networks, it treats the video as a collection of separate pictures and mathematically aggregates their predictions. This method works well for novices learning the fundamentals of video analysis since it is simple to use and computationally efficient.

This implementation serves as a practical example of transfer learning from a design standpoint. Reusing a previously trained convolutional neural network for a similar but distinct job eliminates the need for retraining. Resizing guarantees model compatibility, colour conversion avoids wrong channel interpretation, frame sampling lowers computing costs, preprocessing aligns input scaling, and probability averaging stabilises final predictions. Every stage is crucial to making sure the method for video categorisation operates effectively and dependably.

Error: The main reason the implementation does not perform well on videos

1. Images, not videos, are used to train the model.

MobileNetV2, the backbone network utilised here, was trained using the ImageNet dataset. Instead of continuous video sequences, ImageNet includes individual, independent pictures. Videos include scene transitions, motion, and temporal shifts. The model is unable to accurately comprehend actions or events since it has never learnt motion patterns. Without context, it merely perceives each frame as a distinct image.

2. Inability to comprehend temporal information (motion)

A video consists of movement and changes throughout time, making it more than just a collection of pictures. The implementation merely averages the predictions after classifying

each frame separately. This disregards the motion or interaction of objects between frames. It is impossible for the system to fully comprehend video dynamics without temporal modelling (e.g., LSTM, GRU, or 3D CNN).

3. Frame sampling could overlook significant events.
From the full video, only a certain number of frames (say 20) are taken out. The model will never see crucial material if it comes in frames that are missed. This is particularly problematic in videos that are brief or change quickly. Despite uniformly sampling frames, important events may still be overlooked.

4. Strong signals are diminished by averaging forecasts
All of the frames' forecasts are averaged. This lowers noise, but it may also make bold predictions from significant frames less reliable. For instance, averaging will dilute the accurate forecast and may result in an ambiguous or inaccurate outcome if just a few frames clearly display the principal item while other frames display background stuff.

5. Background Dominance Issue
In a lot of videos, the backdrop takes up a lot of the frame. The model may pay more attention to dominating background aspects than the main subject since it analyses full-frame images. The program could predict "forest" rather than "bicycle," for instance, if a person is riding a bike in a forest, as trees take up more pixels.
6. Video material may not be compatible with ImageNet classes.
ImageNet's 1000 categories mostly consist of object classes rather than activities or intricate scenarios. As action-based categories rather than static objects, the model can have trouble if the video includes an activity (such as cooking, dancing, or playing football). Because of this, the predictions might not match the actual content of the video.

Key points of Video Analysis is explained in detailed:

The second approach does not do categorisation but instead concentrates on comprehending and investigating video features. It shows how to examine a video file frame by frame with analysis and visualisation tools based on Python.

Common data science libraries like pandas and NumPy are imported at the start of the program to handle structured and numerical data. Images and frames can be shown in a notebook environment by importing Matplotlib for visualisation. In addition to importing the notebook-friendly progress tool from tqdm, the glob module is provided for file handling tasks. In addition, the movie is rendered right within the notebook using IPython's display tool.Video. As a result, the video may be examined visually without ever leaving the coding environment.

OpenCV opens the video file using the cv2.VideoCapture() method. The software and the video file are connected via this object. Upon successfully opening the video, a number of significant attributes are retrieved.

To find the total number of frames, **use cv2.CAP_PROP_FRAME_COUNT**. This number is the number of individual photos that comprise the full video. Since a video

is just a fast-paced series of still pictures, knowing how many frames there are in a video aids in organising processing techniques like memory allocation and sampling intervals.

The video frames' height and width are then measured using **cv2.CAP_PROP_FRAME_HEIGHT cv2.CAP_PROP_FRAME_WIDTH**. Each frame's resolution is determined by these characteristics. Resolution has an impact on both visual clarity and computational expense. Although they demand more processing power, higher resolution frames have more pixel information.

Using cv2.CAP_PROP_FPS, the frames per second (FPS) number is also obtained. FPS is a measure of the number of frames that are seen per second during video playback. Because it establishes the motion's smoothness, this value is significant. For instance, motion at 30 frames per second is usually smooth.

Once the metadata has been extracted, the captured item is released to free up system resources. Frame extraction is then shown by reopening the video. One frame is read at a time using the **cap.read() method**. **Two items are returned: the frame picture itself and a boolean (ret)** indicating if the frame was properly read. Frames are accessible as long as ret remains true.

When Matplotlib is used to **display a frame, a colour format problem arises. Matplotlib requires RGB format, however OpenCV imports pictures in BGR format.** Consequently, a helper function is written to use cv2 to convert BGR pictures to **RGB.cvtColor().** By doing this, accurate colour representation is guaranteed during display.

A grid of subplots (5 × 5) is created in the next step. Several frames can be shown at once thanks to this framework. After then, the application goes over every frame of the video. It use a sampling approach rather than showing every frame. Only the 100th frame is chosen by determining whether frame % 100 == 0. The modulo operator, which verifies divisibility, is used for this. Sampling eliminates repetition and gives an overview of the evolution of the video's content.

The picture is converted from BGR to RGB and added to the subplot grid for each frame that is chosen. In order to aid identify its place in the video timeline, the frame number is appended as a title. For a clearer visualisation, axis marks are disabled. Lastly, to avoid overlap, plt.tight_layout() automatically modifies the distance between subplots.