COMSC 448 Final Project: Video Classification

Yuchen Wei, COMSC 448 Section 1, yxw5769@psu.edu, Team WYC

December 3, 2023

My name is Yuchen Wei and my team name is Team WYC. I'm the only member of my team. This is the report of my final project.

1 Introduction to the Video Classification

Video classification is a technique in computer vision and machine learning that categorizes videos into predefined classes based on their content. It involves extracting features from video frames using methods like Convolutional Neural Networks (CNNs) and analyzing the temporal sequence of these frames with models like Recurrent Neural Networks (RNNs) or Long Short-Term Memory networks (LSTMs). The goal is to understand the video's context and subject matter, enabling applications in areas such as surveillance, content filtering, and activity recognition.

In this project, I explored the application of various neural network architectures, specifically Recurrent Neural Networks (RNN), Convolutional Neural Networks (CNN), and Transformers, to the task of video classification. The primary dataset employed for this study was UCF101, a well-known collection of videos focused on sports movement classification.

2 Dataset: UCF101

The UCF101 dataset is a pivotal resource in computer vision research, particularly for the development and evaluation of action recognition algorithms. It encompasses a rich collection of 13,320 videos, meticulously categorized into 101 distinct action classes. These classes cover a broad spectrum of activities, from simple daily tasks to complex sports movements, offering a comprehensive challenge for algorithms in accurately recognizing and classifying human actions.

What sets UCF101 apart is its emphasis on real-world scenarios. The videos are predominantly sourced from YouTube, ensuring a realistic and diverse representation of actions in various settings and contexts. This diversity is critical in training algorithms that are robust and effective in practical, real-life situations. Variations in lighting, camera angles, background environments, and the spontaneity of human actions in these videos add to the complexity and richness of the dataset.

UCF101 has been instrumental in the field of computer vision, serving as a benchmark for researchers and developers. Its extensive use in academic and research settings has propelled numerous advancements in video processing, analysis techniques, and machine-learning models focused on understanding human actions from visual inputs.

In this project, because of limited computer resources, I only tried a subset of UCF101. The subset includes 5 classes from UCF101. They are action videos of cricket shooting, playing cello, punching, shaving the beard, and tennis swing. Figure 1 shows one example of cricket shooting in the subset dataset. I used the official split of the dataset, which splits the dataset into 594 training videos and 224 testing videos.

3 Preprocessing

The dataset for this video classification project consists of two components: CSV files and videos. The CSV files contain data frames that provide labels for each video, which are essential for supervised learning. Upon loading each video and its corresponding label, a specific number of image frames are

Figure 1: The example of cricket shooting.

extracted from each video, with variations of 10, 15, and 20 frames to assess the impact on model performance. To extract features from these frames, two different pre-trained image embedding generators are employed. The first is InceptionV3, a convolutional neural network (CNN) architecture trained on the ImageNet dataset, renowned for its efficiency in image classification tasks. By excluding the top classification layer, InceptionV3 is repurposed as a feature extractor, generating 2048-dimensional feature vectors for each frame. The second is CLIP, developed by OpenAI, a versatile encoder capable of generating feature vectors for both images and text. Unlike traditional image encoders, CLIP is trained on a variety of image and text pairs, enabling it to understand and represent a wide range of visual concepts in a manner that is effectively aligned with human descriptions. This dual capability of CLIP makes it particularly suitable for tasks that require a deep understanding of both visual and textual content, providing a rich, context-aware representation of each frame in the video. Finally, the final feature tensor for each video is [number of frames, number of dimensions].

4 Recurrent Neural Network

In this project, the first deep learning algorithm I tried was recurrent neural networks. I tried two different kinds of recurrent neural networks: Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM). I implemented them on Keras.

4.1 GRU

The architecture of GRU model is sequential and layered, beginning with an input layer that accepts frame features and a masking sequence. The masking sequence allows the model to handle variable-length sequences efficiently (some videos may not have enough frames). Following the input, there are two GRU layers, with the first layer having 80 units and returning sequences, and the second layer comprising 40 units. GRUs are a type of recurrent neural network (RNN) known for their efficiency in modeling temporal dynamics by maintaining a memory of previous inputs.

To mitigate the risk of overfitting, a dropout layer with a 50% dropout rate is incorporated after the second GRU layer. This layer randomly sets input units to 0 at each step during training, helping the model to generalize better.

Subsequent to the dropout layer is a dense layer with 40 units and ReLU (Rectified Linear Unit) activation. This layer is instrumental in introducing non-linear transformations to the network, enabling it to learn more complex patterns.

The final layer of the model is a dense layer with softmax activation, tailored for multi-class classification. The number of units in this layer corresponds to the number of classes (5) in the dataset. Finally, the model is compiled with the 'sparse_categorical_crossentropy' loss function and the 'adam' optimizer, making it apt for classification tasks.

4.2 LSTM

In the LSTM model, the network architecture is similarly structured to handle sequence classification tasks. It starts with input layers that receive frame features along with a corresponding mask. The masking functionality is crucial for efficiently managing variable-length sequences in the data.

At the heart of the network are two LSTM layers, with the first layer comprising 160 units and returning sequences. This design allows for the capturing of temporal dependencies across the entire sequence of data. The use of two LSTM layers in succession enables the network to learn more complex patterns in the data. Each LSTM layer is followed by a dropout layer with a 50% dropout rate, serving as a regularization technique to prevent overfitting. This is important for maintaining the model's generalization capabilities on unseen data.

After the LSTM and dropout layers, there's a dense layer with 80 units and ReLU (Rectified Linear Unit) activation. This layer introduces non-linearity to the network, allowing it to learn and represent more complex relationships in the data.

The final layer in the network is a dense layer with softmax activation, configured for multi-class classification. The number of units in this layer corresponds to the number of classes, which is 5. Lastly, the LSTM network is also compiled with the 'sparse_categorical_crossentropy' loss function and the 'adam' optimizer, which are standard choices for multi-class classification tasks.

5 Transformer-based Systems

Besides RNNs, I also tried one Transformer-based system. The Transformer-based model includes two such Transformer blocks, each configured with specific embedding dimensions, number of heads (3), and feed-forward dimensions (32). After processing through these blocks, the sequence data is aggregated using global average pooling (GlobalAveragePooling1D), followed by a dropout layer for additional regularization.

Each Transformer block layer, a critical component of the model, consists of a multi-head attention mechanism (MultiHeadAttention) and a feed-forward network (ffn). The attention mechanism allows the model to focus on different parts of the input sequence for each prediction, enhancing its ability to capture complex dependencies. The feed-forward network further processes the output from the attention mechanism. Both the attention outputs and the feed-forward network outputs undergo layer normalization (LayerNormalization) and dropout (Dropout) for regularization and to mitigate overfitting.

At the head part of the model, there is one PositionalEmbedding layer. The PositionalEmbedding layer is integral to the Transformer's architecture, as it injects position information into the sequence of frame features, allowing the model to consider the order of inputs. This layer uses an embedding matrix to generate positional encodings, which are added to the input features, thereby retaining positional context in the subsequent processing stages.

Finally, the model concludes with a dense layer with ReLU activation and an output dense layer with softmax activation. The output layer's size corresponds to the number of classes in the dataset, which is 5. The model is also compiled with the 'sparse_categorical_crossentropy' loss function and 'adam' optimizer, standard choices for multi-class classification tasks.

6 Training Details

In this project, I used their official split which divides the dataset into 594 training video samples and 224 test video samples. For the training, I tried batch sizes equal to 16, 32, and 64 and finally selected

Model Name	10 frames	15 frames	20 frames
InceptionV3 GRU Network	70.3%	81.9%	97.2%
CLIP GRU Network	61.9%	71.2%	76.7%
InceptionV3 LSTM Network	69.6%	88.6%	95.6%
CLIP LSTM Network	62.5%	70.1%	74.6%
InceptionV3 Transformer Network	55.6%	65.1%	72.2%
CLIP Transformer Network	37.6 %	51.3%	57.2%

Table 1: Models' Performance.

64 as the batch size for all neural network models. I set the number of epochs equal to 15 for GRU and LSTM and 20 for Transformer. As I said before, I also tried different numbers of frames (10, 15, and 20) and image encoders (InceptionV3 and CLIP) to better evaluate models. The validation set comes from the 20% of the training set. I use accuracy as the performance evaluation metric. The optimizer is the Adam algorithm and the loss function is the cross-entropy loss function.

7 Results & Observations

Table 1 shows the result of this project. It shows the performance of each neural network across different numbers of frames and different kinds of image encoders (InceptionV3 and CLIP). The evaluation metric of this project is accuracy.

We can easily find some take-away messages and observations. The first thing is that the GRU network performs best. With 20 frames of each video and InceptionV3 as the image encoder, it achieves an accuracy of 97.2%. Also, almost all GRU networks outperform other networks with the same frame numbers and image encoders (except GRU network with CLIP image encoder and number of frames equal to 10).

The performance of LSTM networks is similar to its corresponding GRU networks. This is mainly because they have similar structures and both belong to RNN. Also, it is clear that Transformer networks' performance is obviously worse. They are lower than corresponding GRU networks with about 15% to 25%. This may mean that transformers (at least my transformers) are not good at this task. I also think that Transformers are mainly used in sequence-2-sequence tasks, not classification tasks.

Using InceptionV3 is obviously better than using CLIP in this project for video classification. The gap between accuracy may even be about 20%. This surprises me because CLIP is the encoder with many more parameters (the version of CLIP I used in this project is Vit-14L). The possible reason is that CLIP focuses more on image and text pairing, not only on images.

8 Challenges & Obstacles

In this project, the main challenges come from two aspects: data preprocessing and architecture construction. Regarding data preprocessing, I spent lots of time finding how to preprocess videos. About image encoders, I tried different kinds of pre-trained CNN architectures like Resnet-50 and VGG, and finally, I selected InceptionV3 which can output better image embeddings for this task. Using CLIP also takes me a lot of time. I think the most challenging part of data preprocessing is not coding and trial but finding how to preprocess. Searching for methods on the Internet takes me a lot of time.

The architecture construction of these deep-learning models is also challenging because of the low explainability of deep-learning algorithms. I spent lots of time tuning hyperparameters. In the beginning, my models only have an accuracy of about 20% on the training set. After lots of trials of different hyperparameters and layers, their performance increased. Modifying the transformer took me lots of time, I tried different numbers of transformer blocks and different numbers of multi-heads.

In summary, in this project, I learned how to solve the video classification task. The key knowledge I learned is video preprocessing and neural network construction. Online searching and more trials are the solutions to challenges in this task.

9 Conclusion

In this project, I tried different deep-learning architectures to solve a video classification task. Because of the limit of computing resources, I use one subset of UCF101 with 5 classes, 594 training videos, and 224 testing videos. The deep-learning architectures I tried in this project include GRU, LSTM, and Transformer. I also tried video preprocessing methods with different image encoders and numbers of video frames. Finally, I observe their performance and show some possible reasons. The GRU with InceptionV3 and 20 frames for each video performs best.