Video Description

Antonia Zeibel

Technical University of Cluj-Napoca

June 4, 2025

Summary

Introduction

The goal of this project is to generate a caption given a video.

Road-map:

1. **Data Collection & Preprocessing**: Acquire MSVD corpus and video clips. Extract, select, and preprocess video frames. Clean and tokenize captions.
2. **Feature Extraction**: Extract visual features from selected frames using a pre-trained CNN (VGG16).
3. **Model Architecture Design**: Implement an Encoder-Decoder model using LSTMs for video feature encoding and caption generation.
4. **Model Training**: Train the LSTM Encoder-Decoder model on the prepared video features and captions.
5. **Model Evaluation**: Perform inference to generate captions and evaluate model performance using metrics like BLEU score.

Data Sources

MSVD Dataset Corpus

- primarily used for **video description** or **video captioning** tasks in machine learning and natural language processing
- contains information regarding the video files: VideoID, Start, End, WorkerID, Source, AnnotationTime, Language, and **Description**

MSVD Clips

- contains the video snippets associated to the annotations in the MSVD Corpus

Pre-processing Steps

Video Corpus Clean-Up

- keep only the annotations in English
- drop all the other irrelevant columns and keep only the **VideoID** and **Description**
- prepare the **Description** column for tokenizer
  - transform to lowercase
  - de-contract words
  - remove punctuation, numbers and unnecessary white-spaces
  - adds <BOS>and <EOS>

Frame Extraction and Selection

Frame Extraction

- The video file is transformed into a list of frames
- The frames are saved under specific directories using the unique identifier (VideoID)

Frame Selection

- The frames are categorized as **key-frames** and **in-between frames**
- Key-frames → frames with significant change in the scene
- In-between frames → frames that are between these key-frames

Frame Selection using Scene Detection

The **key-frames** can actually be chosen using a **scene detection technique** which works directly on the video files.

Scene Detection with `scenedetect API`

- Implemented various detectors:
  - ContentDetector (fast cuts)
  - ThresholdDetector (slow transitions)
  - AdaptiveDetector (adjacent frame differences)
  - HistogramDetector (histograms)
  - HashDetector (perceptual hashing)
- **Chosen**: **AdaptiveDetector**
  - Effective for adjacent frame analysis.
  - Chosen via trial-and-error due to varied video content.

Frame Selection

Process After Detector Selection

- Parsed video directories to select **key frames**.
- Set a parameter for the **required number of frames**.
- If detected key scenes did not match, **padded with in-between frames**.

Output

- Selected frames saved in video-specific directories.
- Created `frames_metadata.csv` with:
  - `VideoID`
  - `Key_Frames` (number of detected key frames)
  - `Total_Frames` (final number after padding)
- **Result**: No failed frame extraction or selection.

Feature Extraction using the Selected Frames

Before extracting the features using a pre-trained model, I had to transform the frames into suitable inputs:

Frame Preprocessing

- Ensure all selected frames have consistent size.
- **Resizing**: 224x224 pixels (common for VGG16).
- Convert PIL Image Object to **3D NumPy array** (height, width, channels).
- Apply `preprocess_input` (from `Keras`) for proper VGG16 handling.

Feature Extraction using the Selected Frames

Now that the frames are ready for the model, the features can be extracted:

Feature Extraction with VGG16

- Loaded **pre-trained VGG16 model** on ImageNet dataset.
- Access the **second-to-last Dense layer** (4096 units) since interested in **feature representation**.
- 4096-dimensional feature vector for each 224x224x3 frame.
- All resulting feature vectors per video are appended to a NumPy file, saved under its `VideoID`.

Captions Processing: Tokenization & Analysis

Tokenization with Keras Tokenizer

- Uses a **Word-level tokenization** which basically splits text by word separators.
- Builds vocabulary by assigning unique integers to words.
- **Out-of-Vocabulary (OOV)** words: Replaced with an OOV token (e.g., `<unk>`) or skipped.
- **Vocabulary Limit**: Initialized to use only top 1500 most frequent words.

Captions Processing: Tokenization & Analysis

Token Length Analysis

- Performed analysis on tokenized description lengths.
- **Key Statistics**:
  - Max Length: 139
  - Min Length: 3
  - Mean Length: 9.10
  - 90th Percentile: 12
- Filtered data by max/min length to discourage outliers.
- Sequences were padded to a fixed maximum length to ensure consistent model input.

Caption Processing: Example

Table 1: Video Description and Padded Sequence Examples

Description	**Padded Sequence**
⟨bos⟩ a bird is bathing in a sink ⟨eos⟩	[3, 2, 253, 5, 554, 9, 2, 465, 4, 0, 0, 0, 0, 0]
⟨bos⟩ a bird is splashing around under a running faucet ⟨eos⟩	[3, 2, 253, 5, 1, 81, 318, 2, 47, 903, 4, 0, 0, 0, 0]
⟨bos⟩ a bird is bathing in a sink ⟨eos⟩	[3, 2, 253, 5, 554, 9, 2, 465, 4, 0, 0, 0, 0, 0]

Caption Generator: LSTM Encoder

The first block of the architecture is the **encoder**.

Role of the LSTM Encoder

- Responsible for processing the input sequence of **video features**.
- Compresses the information into a fixed-size **context vector** (final hidden and cell states).
- This context vector acts as a summary of the entire input sequence.

Caption Generator: LSTM Encoder

Key Aspects of the Encoder Architecture

- **LSTM layer**: The core recurrent unit.
- **Input Size**: Dimensionality of each individual feature in the input sequence (e.g., 4096 for VGG16 features).
- **Hidden Size**: Number of hidden units in the LSTM layer.
- **Outputs**: Output of the LSTM at each time step.
- **Hidden**: Final hidden state (compact representation of the entire input sequence).
- **Cell**: Final cell state (part of the LSTM's internal memory).

Caption Generator: LSTM Decoder

The second block of the architecture is the **decoder**.

Role of the LSTM Decoder

- Takes the **context vector** from the encoder.
- Generates an **output sequence** (the textual caption).
- Predicts one word at a time, using the previously predicted word and the encoder's context.

Caption Generator: LSTM Decoder

Key Aspects of the Decoder Architecture

- **Embedding Size**: Dimensionality of the word embeddings.
- **Hidden Size**: Number of hidden units in the decoder's LSTM layer (typically matches encoder's hidden size).
- **Vocabulary Size**: Total number of unique words in the vocabulary (including special tokens).
- **Outputs, Hidden, Cell**: Predicted word logits for the sequence, along with the final hidden and cell states.

Caption Generator: VideoCaptioningModel

The Combined Model

- Combines both the **LSTM Encoder** and the **LSTM Decoder**.
- **Flow:**
  1. Video features are first passed through the **Encoder**.
  2. Encoder outputs initial **hidden and cell states**.
  3. These initial states are then passed to the **Decoder**, along with the caption tokens.
  4. The Decoder generates the **predicted word logits**.
- **Final Output**: Sequence of predicted word logits.
  - Used to calculate loss during training.
  - Used to determine the most likely words during inference.

Training

Training Setup

- Model trained for **40 epochs**.
- Instances of the model were saved at:
  - Epoch 10
  - Epoch 20
  - Epoch 30
  - Epoch 40

Training Progress

- The training loss consistently **decreased** over the epochs.
- This indicates that the model was **successfully learning** from the training data, continually reducing prediction errors.

Training Loss

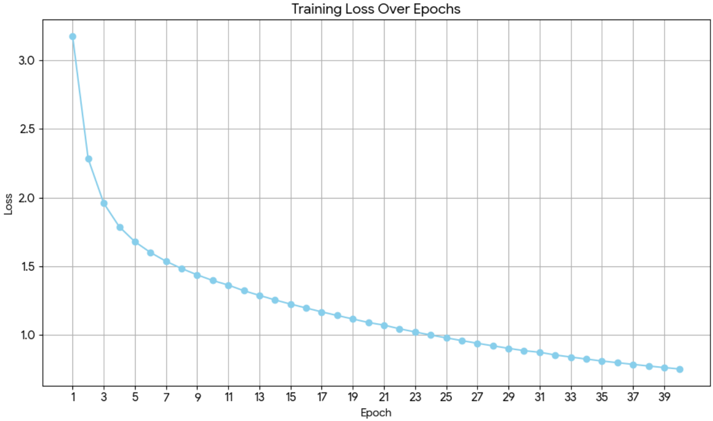

Figure 1: Training Loss

Evaluation: Inference Process

Out of curiosity, I wanted to visualize the results of the model,therefore I have implemented a dedicated method for inference.

Inference Mechanism

- Implemented a dedicated method for the **inference process**.
- Model generates the caption **word by word**.
- Visualized results by displaying actual video frames alongside the generated captions.

Evaluation: BLEU Scores

To measure the scores along the test data, I have used the **BLEU Score**.

BLEU Score

- Widely used metric for evaluating the quality of **machine-generated text**.
- Compares generated text against a set of human-created reference texts.
- **Interpretation**: A higher BLEU score generally indicates better quality and closer resemblance to human references.

Evaluation: BLEU Scores

Table 2: Average BLEU Scores on Test Set

Model trained for	**Average BLEU Score**
10 epochs	0.1734
20 epochs	0.1685
30 epochs	0.1586
40 epochs	0.1548

Conclusions regarding the BLEU Score

- The model trained for **10 epochs showed the highest BLEU score**.
- I have expected to improve the BLEU score if I trained for more than 10 epochs, but the findings show that the score actually decreases after the 10th epoch.
- There is still significant room for improvement in caption quality.

Further Improvements

- Use a different tokenizer for the caption processing.
- Use different vocabulary sizes.
- Explore other architectures for video captioning or even summarization.

Thank you!