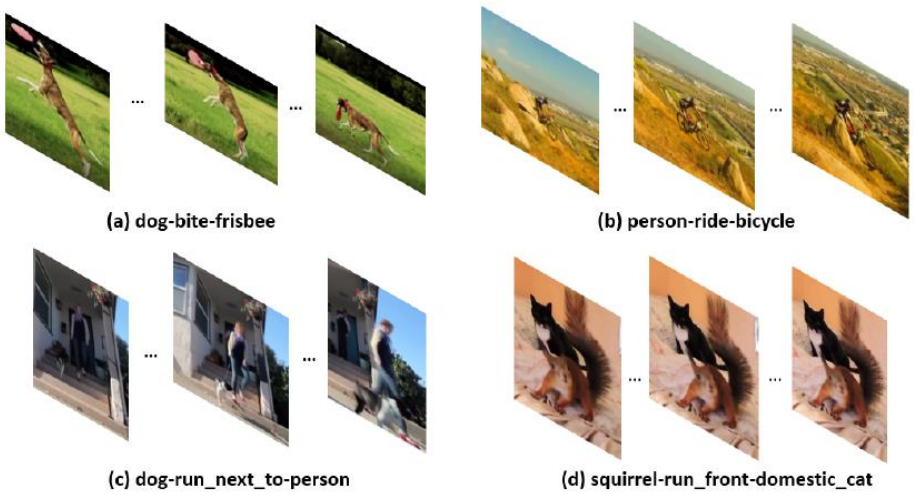
Video Captioning Using CNN-LSTM Architecture (Kaggle Challenge)

Mittal Sidharth (A0218815B) Suen Ao Xiang (A0113842M)
Nguyen Huy Anh (A0088386L)



1. Abstract

In this project, our task was to perform the video visual relationship prediction on a dataset of the video files. There is a plethora of works around detecting different object and relationships in videos, wherein recognition of each object and relationship is treated as a separate classification problem; however, in our approach we propose an integrated model using an encoder-decoder architecture. The encoder is tasked with thorough understanding and learning of visual representation, and the decoder is entrusted with caption generation, which decodes the learned representation into a sequential sentence, word by word. Hence, our project integrates two broad areas in machine learning – computer vision and natural language processing.

2. Introduction

The goal of video captioning is to automatically describe the visual content of videos with natural language. In visual relationship detection, for each input, the task is to generate a triplet of (object1 – relationship – object2), instead of a single label prediction in object simple object detection. In addition, compared to image captioning, video visual relationship detection also provides a more comprehensive description of the objects' relationship due to the sequential order.

Our proposed framework for this task is shown in the chart below:

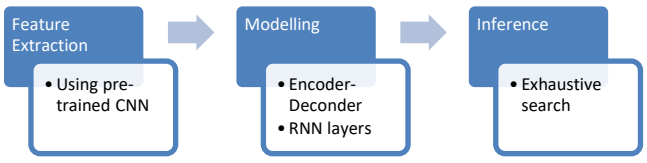


Figure 1: General Methodology Framework

3. Exploratory Data Analysis

Our data contains 566 videos selected from the ILVSRC2016-VID dataset, of which 447 are used as train set, and the remaining 119 are used as test set. Each video is already processed into 30 still frames.

There are a total of 35 different objects and 82 different relationships in the dataset.

Figure 2: Frames from train video 000006: "bicycle move_beneath person"

4. Modelling

4.1 Feature Extraction

We extracted features from the images using the InceptionV3 pre-trained model.

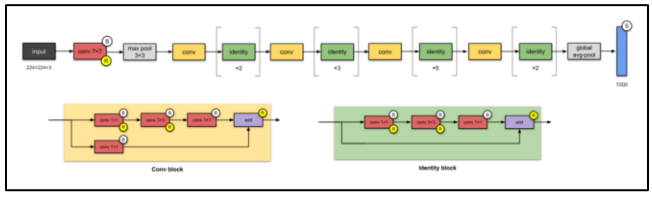


Figure 3: InceptionV3 Model Architecture

Since the default input size for InceptionV3 is (299, 299, 3), we resized our image inputs accordingly. The feature map of the image was extracted by removing the final SoftMax layer of the pre-trained model meant for classification.

The output for each extracted image has the shape 2048, thus for each video of 30 frames, the output shape was (30, 2048).

4.2 Model Building and Training

We randomly split our train data into train and validation set with a ratio of 85:15.

The text data consisting of all object and relationship words were tokenized and formed the vocabulary for this project. There were a total of 121 words in our vocabulary, including 32 objects, 82 relationships, one empty token " ", one unknown token "unk", one beginning token "<BOS>" and one ending token "<EOS>". Each training label was then translated into a vector of 5 words "<BOS> – object1 – relationship – object2 – <EOS>".

We adopted an encoder-decoder architecture, where the encoder consist of a RNN layer mainly to encode the sequential information of each frames in the video. While the decoder RNN layer would generate video captions based on the input features from the encoder.

During training, the features extracted from the pre-trained CNN model were inputs to the encoder RNN. The outputs of the RNN layer were ignored as we are interested in the hidden states of the RNN, which we would use as the initial state for the decoder RNN.

At the decoder RNN layer, the input would be a vector consisting of the one-hot encoded ground truth caption, and the hidden states initialised from the hidden state of the encoder RNN layer. This approach of having to input the ground truth labels is known as the teacher forcing method.

Finally, the outputs of the decoder RNN were passed through to a Dense layer using SoftMax activation function to classify each of the 5 words output.

With this general process, we attempted several different RNN architectures and observed their performances on the validation set.

As a baseline model, we implemented a single LSTM layer each in the encoder and decoder model, with the architecture described above. The model is summarized in the figure below.

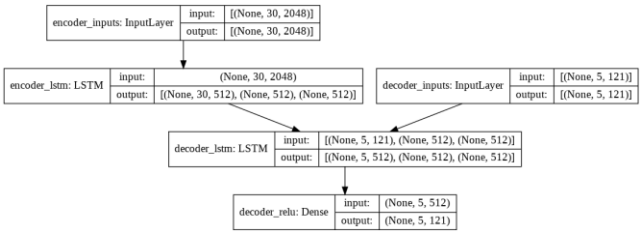


Figure 4: CNN + LSTM Model Architecture

Then, we replaced the LSTM layers with bi-directional LSTM (BiLSTM) layers. As LSTM only considers sequential information in one direction, BiLSTM utilizes information from both directions, learning from the input sequence in normal and reversed order.

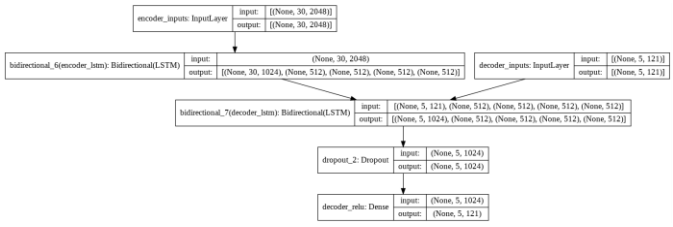


Figure 5: CNN + BiLSTM Model Architecture

Next, we experimented with stacking multilple layers of LSTM and implemented non-teacher forcing by not including the ground truth labels as the input to the decoder layers. This was different from the previous models, which could potentially reduce overfitting and help the model to generalize better on test data.

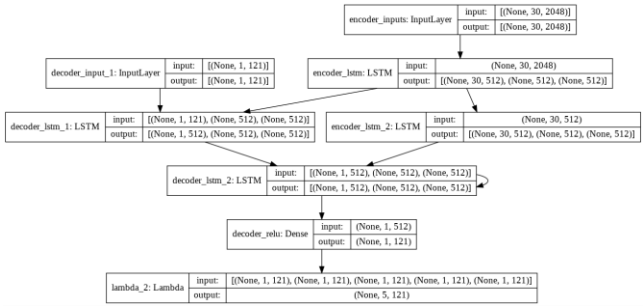


Figure 6: Stacked LSTM with Non-Teacher Forcing Model Architecture

Finally, as our models were mainly trained on the features extracted from the pre-trained InceptionV3 model, we attempted to train our model end-to-end including the last few layers of the CNN model. As an alternative to InceptionV3, we also trained on the ResNet50 and MobileNetV2 models.

4.3 Inference Models

For inference, we retrained the model on the whole training set (447 samples) instead of a subset of it due to the small training dataset available.

Our inference models consist of two parts, first being the encoder inference model, where the inputs to the encoder LSTM model are the resized images or extracted image features, whichever applicable. The output of the LSTM model will likewise be ignored as we are interested in the hidden and cell states, which would be used as the initial state to the decoder LSTM model.

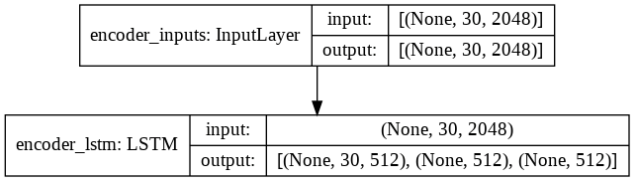


Figure 7: Example of Inference Encoder Model Architecture

The second part of the inference model would be the decoder inference model. A vector consisting of the one-hot encoded text would be the input to the decoder LSTM model, with the encoder LSTM hidden and cell states as the initial state. The output of the decoder inference model would be a one-hot encoded vector representing the predicted word in the next time-step.

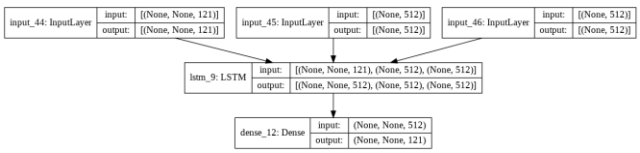


Figure 8: Example of Inference Decoder Model Architecture

In order to optimize the predicted output of our inference model, a naïve approach would be to adopt the greedy search method, whereby the predicted output would be selected from the highest probability output from the final SoftMax layer. However, this approach may result in problematic predictions as the final caption output may not have the highest joint probability, resulting in higher error rate.

Therefore, we adopted the exhaustive search method, whereby the outputs are predicted iteratively, and the joint probabilities of each candidate captions are considered. We noted that while the iterative approach in exhaustive search may be computationally intensive as compared to greedy search, it was still achievable due to the short captions in our project.

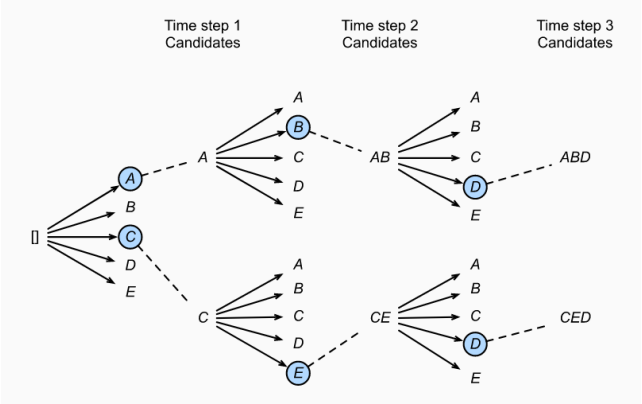


Figure 9: Illustration of Exhaustive Search Approach

In order to optimize on the exhaustive search approach, we avoided redundant predictions wherever possible, such as when "<BOS>" and "<EOS>" tags appear in the middle of the caption, or when object words appear in the middle of the caption where relationship words should be, and vice-versa for relationship words. If such predictions occur, the remaining branches would be ignored and thus reducing the computational cost. The total score for each sequence was calculated as the sum of the negative log probability for each word at each timestep, therefore, the lower the aggregated score is, the better.

As the final submission required the top 5 candidate words for each element "object1", "relationship", "object2", by only selecting 5 candidate captions with the best aggregated score, there would be an inherent issue of multiple appearances of the same word candidate, or insufficient word candidates to fulfill the top 5 criteria. In order to circumvent these issues, we expanded our exhaustive search width to predict up to 50 words per node, in an attempt to obtain more variety of candidate words. During the selection stage, we only selected the top 5 unique words.

5. Results

After setting-up the training and inference models, we trained the above mentioned architecures and tweaked the various hyper-parameters, such as learning rate, hidden layer size, loss function, and also added the regularization terms (such as weight decay) as the model was overfitting on training data. However, despite adding the regularization parameters the training accuracy remained much higher than validation accuracy. Out of all the model architectures, the best performing model on the kaggle dataset was the encoder-decoder LSTM architecure – for which we used extracted features from pre-trained InceptionV3. The hyper-parameters for this model were – hidden layer size (512), kullback_leibler_divergence loss function, weight decay (0.001).

The validation and training results of our models are summarized in the table below.

No.	Model	Training	Validation
1.	**LSTM (tuned)**	100%	69%
2.	**BiLSTM**	100%	75.6%
3.	**Stacked LSTM (non-teacher forcing)**	100%	72%
4.	**End-to-End (Resnet50)**	100%	65.7%

Table 1: Model Performances

All models exhibits overfiting behaviors as their train accuracy are much higher than their validation accuracy.

6. Conclusion and Areas for Imrprovement

As a conclusion, the base LSTM model with hyperparameters tuned had the best performance on the test data.

Some potential areas of improvement would be to include image data augmentation, such as injecting noise to video frame images. It should be noted that some augmentation methods such as horizontal or vertical flipping and rotations up to 90 degrees are not suitable for the context of this project, as the captions include spatial descriptions such as object 1 being on the left or right of object 2. Another potential area to explore would be to include object detection architectures such as RCNNs and YOLO to more accurately identify the objects of interest. Attention layers could be added after feature extraction layers to apply different weights to different areas of the feature map.

7. Statement of Contribution

Ao Xiang and Sidharth contributed to build and tune the different models. Ao Xiang, Sidharth and Huy Anh were involved in writing the project report.

References:

1. https://openaccess.thecvf.com/content_cvpr_2017/papers/Dai_Detecting_Visual_Relationships_CVPR_2017_paper.pdf
2. https://www.ijcai.org/Proceedings/2019/0877.pdf
3. https://cs.stanford.edu/people/ranjaykrishna/vrd/vrd.pdf
4. https://nextcenter.org/wp-content/uploads/2018/04/Video-Visual-Relation-Detection.pdf
5. https://d2l.ai/chapter_recurrent-modern/beam-search.html
6. http://cs231n.stanford.edu/reports/2017/pdfs/31.pdf
7. https://arxiv.org/pdf/1505.00487.pdf