Video Captioning Project

Chao Fang How, Gabriel; Jovi Loo Lek Yang; Wu Shifan

Introduction

In this project, our goal is to produce a deep learning model that, given a sequence of frames from a video, generates three-word captions in the form $object_1$-*relationship*-$object_2$. Given that the three words come from a predefined list of 35 objects and 82 relationships, this is a three-way classification problem.

Proposed Model

Network Structure

We use a sequence to sequence model for caption generation. The input is the sequence of video frames $X = (x_1, x_2 \ldots, x_t)$, where $t = 30$ by default as we are provided with 30 frames per sample. The output sequence is $Y = (y_1, y_2, y_3)$ as we are only predicting three words.

The model seeks to find a set of parameters that maximizes the probability of a given triplet Y given input video frames X.

The temporal nature of the input data makes RNN an excellent choice for modelling, since the prediction of the three words are not independent of each other, but are likely to have some sort of temporal relationship. For instance, it is very unlikely to have labels such as *snake-fly_next_to-skateboard*. This makes the RNN even more relevant for generation of the captions, as it can also learn these relationships.

LSTM was eventually chosen because it is much abler to capture long-term dependencies. In terms of structuring the layers, we took reference from Venugopalan et al. (2015) as well as the model suggested by the teaching assistant. We also experimented with GRU, which we found to be more time efficient due its relatively simpler structure, but displayed inferior performance relative to LSTM.

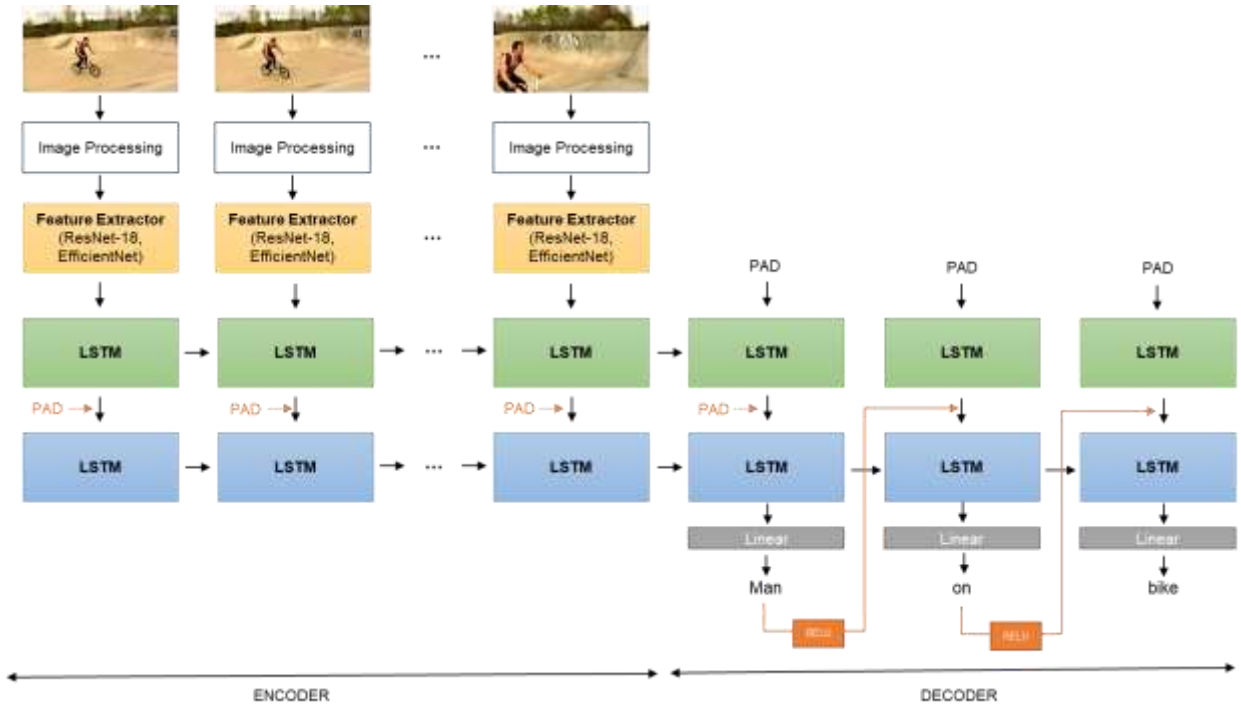

Figure 1: ***Model architecture.*** *There are 4 LSTM cells in total: top encoder, bottom encoder, top decoder and bottom decoder. The object$_1$ and relationship predictions are transformed via a relu hidden layer before being fed back into the bottom decoder.*

Data Processing

Before feeding the image into the network, we did basic image processing, including resizing the rectangular image into 240 by 240. This is an important step to ensure that inputs into the feature extractor have the same shape and similar data distribution so that convergence is sped up while training.

For feature extraction, we experimented with EfficientNet and ResNet-18 pre-trained on ImageNet. For both, we stripped the last layer and extracted feature maps, which were then average pooled across the height and width dimensions. Our final model uses EfficientNetB1 which outputs features of dimension 1280 for each frame.

Encoding and Decoding

We tried various combinations of stacking the LSTMs, with some architecture having parameter sharing, while others having completely distinct LSTMs. Eventually, a two-layer LSTM was chosen for both the encoder and decoder, with each LSTM having 512 units. The architecture is summarized in Figure 1.

In general, the architecture is split into two stages:

The *encoding* stage first encodes the feature vectors into a fixed length vector using top-layer LSTM, and then uses the bottom-layer LSTM to map the vector to a sequence of outputs. Here in the encoding stage, the outputs of the bottom-layer LSTM are not used.

The *decoding* stage inherits the hidden and cell state from the encoder and outputs a latent vector which is then fed into the linear layer. The linear layer then produces a vector of dimension 35 or 82, corresponding to the number of objects and relationships respectively.

Our target is to generate 3 words. Step by step, the first layer of the decoder takes zero paddings as input and its output is concatenated with the <bos> padding token to be fed into the second decoder layer. The output of the first time step of this second decoder is connected to a size 35 softmax output layer. The result of this softmax is taken as the prediction for $object_1$.

To allow the network to learn to generate better predictions of the *relationship* and $object_2$, we feed the $object_1$ prediction into a fully connected layer with relu to obtain a 128-dimensional representation. We then concatenate this representation with the first decoder layer's second time step output before inputting it into the second layer to produce the second time step. The output of the second time step is given to a size 82 softmax output layer to produce the prediction for the *relationship*, which is again converted to a 128-dimensional space via its own hidden layer before being concatenated back into the decoder to finally produce the $object_2$ prediction. The intermediate hidden layers thus allow both $object_1$ and *relationship* predictions to be represented in a dimension of the same size 128.

During training, categorical cross-entropy loss is calculated for each of the three output layers and summed with equal weight. This loss is then back-propagated to tune the parameters.

Training Strategies

Over the course of training, we employed a couple of strategies either to speed up training, or to improve predictive accuracy.

First, instead of using all 30 frames in the video sample, we explored using either only 10 frames or 15 frames. This strategy takes advantage of the high temporal correlation between immediate sequential images. When we used either 10 or 15 frames, we were careful to ensure that the frames selected are spaced equally apart, and that they span the entire video length. To illustrate, if we chose to use 15 frames, we selected either ($x_1, x_3 \ldots, x_{29}$) or ($x_2, x_4 \ldots, x_{30}$) with equal probability. Hence, different epochs may receive either the first or second set of images. Beyond the obvious benefit of speeding up training, we see the different sets of video frames as perturbations of each other. Since the dataset is small for our task, data augmentation is required to train a model which can distinguish 35 objects and 82 relationships.

Second, during image processing, we perform random image perturbations, such as horizontal flips applied consistently to all frames from the same video, as well as per-frame rotation and height/width shifts. We observed that these augmentations helped the model generalize better and alleviate overfitting.

We employed the Adam optimizer with a learning rate of 0.001, which we found to significantly improve convergence over standard SGD with momentum. Our final model uses 10 frames per video with a batch size of 4.

Results

The main concern was indeed overfitting. While training loss would steadily decrease until accuracy on the training set was 90% or greater, the validation loss would reach a minimum after ~2-3 epochs before rising again, as shown in Figure 2. We tried various regularization strategies such as L2 on the LSTM and/or fully connected layers, dropout, label smoothing, stronger data augmentation, reducing model capacity, early stopping, and different learning rates, optimizers, batch sizes, and number of input frames. However, none of these techniques produced a measurable or consistent improvement in test accuracy compared to the first version of our model.

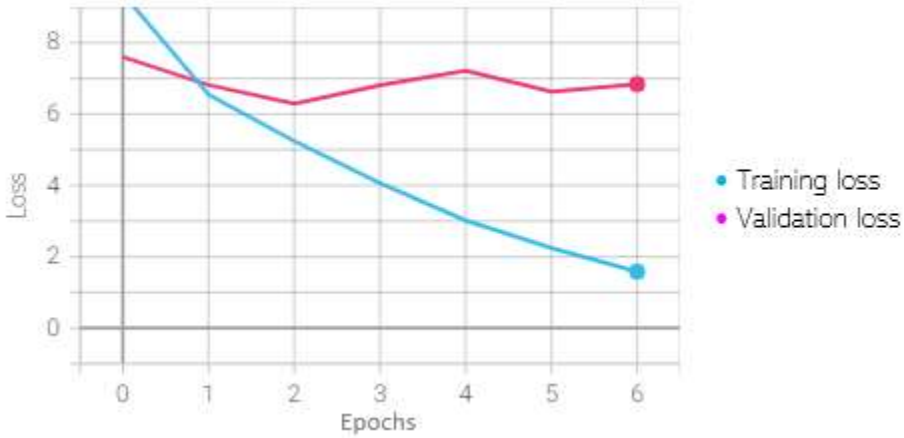

Figure 2: ***Sample train-test accuracies over epochs***

The dearth of training data was probably a significant contributing factor; reducing the size of the validation set from 20% to 10% produced a model with a 0.03 improvement in test score. However, the smaller validation sets gave less reliable estimates of test score, with models having validation scores of 0.80 scoring only 0.67 on the test set. Cross-validation may be a suitable answer.

Interestingly, we were able to significantly reduce model capacity by a factor of more than 8 by trimming the number of units in the LSTM and hidden layer from 512 and 128 to 128 and 32 respectively with only a 0.001 decrease in the best test score, suggesting that many of the parameters in the larger model were redundant. The weights for both the 9M parameter and 1M parameter models are included in the submission, with test scores of 0.68160 and 0.68039 respectively.

Possible other areas of exploration and improvement

The aforementioned model achieves good accuracy and generalizes well on our validation set. Indeed, several distinct models were proposed during our discussion, some of which have even been experimented and proved to be a bad option.

The EfficientNet extractor gave good feature extraction. On a simplified single image classification task, the CNN classifier on its own was able to achieve an accuracy of 96% on the validation set. However, when we tried to predict both the first and the second object, the accuracy decreased. This is actually quite reasonable -- CNN may distinguish the two “main objects” of a picture, but without motion, it can be

difficult to tell which is the subject and which one is the object. As a result, it gives nearly the same output for the two objects. When we add these “extra hints” to LSTM, the model gives worse predictions, which means that we add some noise to the model. Finally, we decided to give up this proposal.

Some further extension of our current model may be to introduce attention modelling -- add skip connections from the hiddenstate to the decoder, and weigh them by weights computed from some learned parameters. The skip connection would in theory bring back more gradients to the encoder and the weighted sum would enable the decoder to focus on different parts of information at each timestep.

The best way, from our perspective, to improve the model accuracy is to introduce gradients back to the CNN part, namely the EfficientNet, and further fine-tune it together with the LSTM. The reason behind this strategy is quite straight forward -- the pre-trained model may give excellent performance on ImageNet owning 1000 classes, our dataset, nevertheless, is much less complicated, with only 35 classes. After fine-tuning the EfficientNet on our dataset, the CNN might give better image feature extraction, with the added benefit of integrating the whole model end-to-end.

The hyperparameter tuning process could be extended to include techniques such as grid search, k-fold cross-validation, or Bayesian optimization for a more thorough approach.

References

[1] Venugopalan, S., Rohrbach, M., Donahue, J., Mooney, R., Darrell, T., & Saenko, K. (2015). Sequence to sequence-video to text. In *Proceedings of the IEEE international conference on computer vision* (pp. 4534-4542).