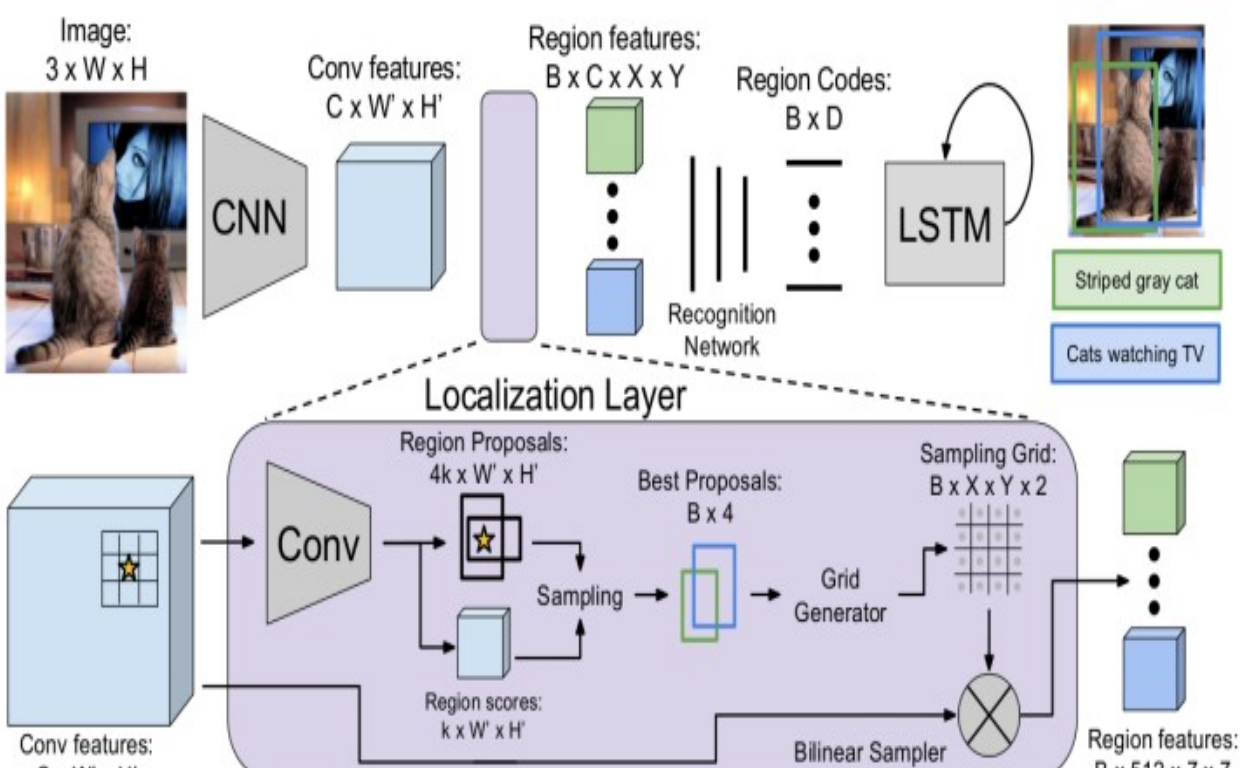
A quick introduction to dense captioning mechanism

by

Ali Sharifi Boroujerdi



Introduction

DenseCap is a system to localize and describe salient regions in the images in natural language. It is a combination of object detection and image captioning tasks that has these important properties:

1. It processes each image with a single and efficient forward pass.
2. It needs no external region proposals (e.g. EdgeBoxes).
3. It can be trained end-to-end with a single round of optimization because the localization parts are fully differentiable.
4. It is a combination of a CNN, a novel dense localization layer and an RNN language model based on torch-rnn implementation.

The data set that is used to train the whole network is VisualGenome which is prepared by a team in Stanford University leading by Prof. Fei-Fei Lee containing more than 100k images which are annotated with more than 4M region-grounded captions.

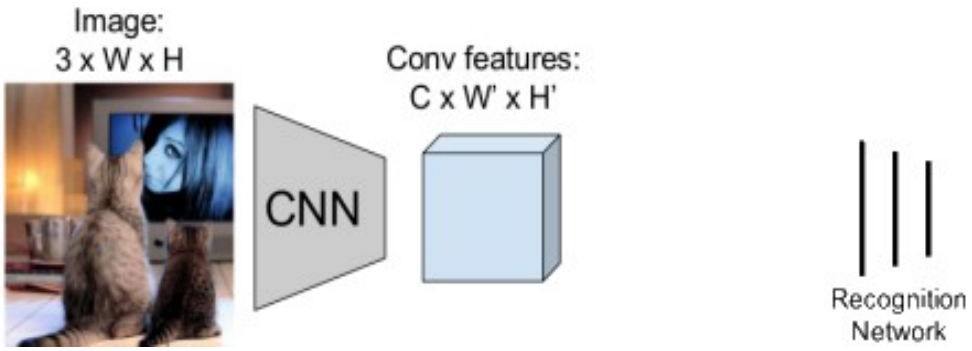
System Components

Convolutional Neural Network (CNN)

Is the VGG-16 architecture that is separated in two parts:

I. 13 convolutional Layers of size 3 x 3 and 5 max pooling layers of size 2 x 2 where the last pooling layer is removed. This part of the CNN plays the role of feature representor of the network input. The output of this part is the input of the *localization layer* which will be called *activations*.

II. The ***Recognition Network*** that consists of Fully-Connected (FC) layers of the VGG Network. Each of FC layers uses ReLU units and will be regularized by *Dropout* technique. This part processes flattened versions of the *region features* from the localization layer and produces the *region codes* which are the inputs of the *Language Model*.



Fully Convolutional Localization Layer (FCLL)

This layer receives an input tensor of *activations*, identify special region of interest and smoothly extracts a fixed-size representation from each region. The baseline approach of this layer is inspired by *Faster R-CNN* model in which the *RoI Pooling Mechanism* is replaced by *Bilinear Interpolation* technique (see Spatial Transformer Networks). This change allows the model to propagate gradients backward through the coordinates of predicted regions that opens up the possibility of predicting *Affine (morphed) regions* instead of bounding boxes.

This layer internally selects B regions of interest by regressing offsets from a set of *translation-invariant anchors* of different aspect ratios. All the information of these anchors (their objectness score and four number to determine their location in the input image namely center x and y coordinates, their width and their height) will be generated by passing the input feature map through a 3 x 3 convolution with 256 filters (this layer maps each sliding window of the conv feature map to a lower dimension vector “256-d in this case”) , a rectified nonlinearity and a 1 x 1 convolution (that acts as an FC layer) with 5k filters. Then the proposed parametrization method of *Fast R-CNN* paper will be used to regress from anchors to the region proposals by generating *four mapping scalars*.

Since the number of *region proposals* is too high to run the *recognition network* and the *language model* on top of them, it is necessary to subsample them (*Box Sampling*). Based on the method proposed in *Faster R-CNN* paper, during the sampling procedure at training time, regions will be divided to *Positive Regions* and

Negative Regions based on their *Intersection of Union (IoU)* with their corresponding *ground-truth regions*. At the test time, subsampling will be done using a greedy *non-maximum suppression (NMS)* on the predicted proposal confidences to select the most confident proposals. Its explanation can be easily found over the Internet.

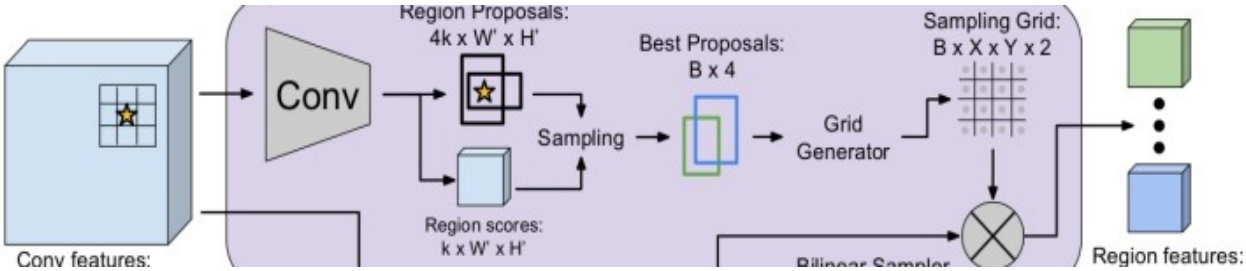
After sampling, we are left with region proposals of varying sizes and aspect ratios. In order to inference with *recognition network* and the *RNN language model*, we must extract a *fixed-size feature representation* for each variably sized region proposal (why?). In *Fast R-CNN* paper, a *RoI polling layer* is proposed to solve this problem. The *Roi Pooling Layer* is a function of two inputs:

1. Convolutional feature map *U* (the original input of the localization layer).
2. Region proposal coordinates.

Where *Gradients* cannot be back-propagated to the input proposal coordinates. (why?) Therefore, *Bilinear Interpolation* techniques is proposed to overcome this limitation in which we interpolate convolutional feature map *U* (input of the localization layer) to produce an output feature map *V* of shape *C* x *X* x *Y* for each region. Based on proposed method we have to compute a sampling grid *G* of shape *X* x *Y* x *2* associating each element of *V* with real-value coordinates into *U.* This computation will be done by a *convolution operation* with a *sampling kernel*. Since the sampling grid is a linear function of the proposal coordinates, so gradients can be propagated backward into predicted region proposal coordinates.

At the end, the localization layer returns three different outputs:

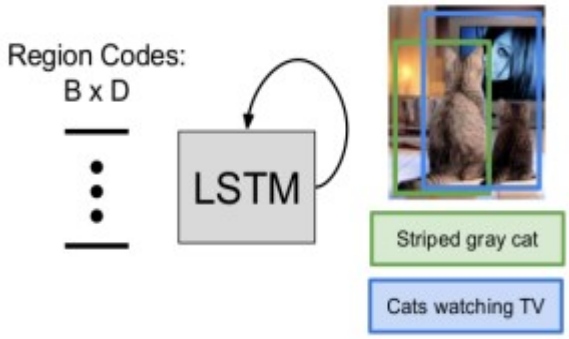
I. **Region Coordinates**: In the shape of B x 4 containing bounding box coordinates (region center coordinates plus width and height of each region) .
II. **Region scores**: A vector of length b giving a *confidence score* for each output region (probability of their correspondence with ground-truth regions).
III. **Region Features**: A tensor of shape B x C x X x Y (*C*=512) representing an X x Y grid of C-dimensional features for each region.



RNN Language Model

The core architectural element of this part is a *Recurrent Neural Network* (RNN) that has the ability of learning powerful long term interactions. This network should be conditioned on image information (actually *region codes* that are produced by the *recognition network*). Based of training sequences of tokens, the network will be fed by some *word vectors* in which the first element is the region code encoded with a linear layer followed by a ReLU non-linearity , the second one is the special START token and the rest are the language tokens. The RNN will generate a sequence of *hidden states* h and *output vectors* y using LSTM recurrence formula. The vector y have size $|V|+1$ where V is the token vocabulary and the additional one is for the spacial END token.

The implementation of the Language model is based on *torch-rnn* implementation by justin johnson (Stanford PhD candidate). torch-rnn provides high-performance, reusable Vanilla RNN (VanillaRNN.lua) and LSTM (LSTM.lua) modules for torch7, and uses these modules for character-level language modeling similar to char-rnn (Implemented by Andrej Karpathy). Although DenseCap RNN Language model acts better at word-level language modeling.



Implementation Details

The proposed model is implemented using Torch framework in *LUA programming Language.* Here, different modules are responsible for different tasks:

I. **Preprocess.py:** The script that is used to preprocess a dataset of images, regions, and captions (VisulaGenome), and convert the entire dataset to an HDF5 file and a JSON file to be read by Lua scripts.
II. **DataLoader.lua:** is the responsible for reading data from HDF5 and JSON files, process them and pack them in a suitable formats.
III. **LocalizationLayer.lua:** that contains all the parts that are necessary to generate region coordinates, scores and features, including Region Proposal Network (RPN), box regression and box sampling procedures and bilinear interpolation consists of a grid generator and a bilinear sampler.
IV. **LanguageModel.lua** : including all the necessary parts of an RNN to model a language which can be run in two different modes: LSTM and Vanilla RNN.
V. **DenseCapModel.lua:** contains the whole components of the final network in the shape of stacked components.
VI. **Train.lua:** which is used to define, initialize and train a *DenseCap* model from scratch while validating the quality of the trained network using validation or the test data.
VII. **Run_model.lua:** which is used to load and run a pre-trained model to generate final outputs that can be used with three different type of inputs:

1. A single image file in png, jpg, ppm or pgm formats.
2. A directory containing several images.
3. An split of MS COCO data set containing training, validation and test portions.

   The output can be visualized in the form of simple images or in an interactive *html representor.*

VIII. **Evaluate_model.lua:** which can be used to evaluate a trained DenseCap model using cross-validation technique.