**Image Captioning**

Collector: 1305095

Coder: 1305115

Trainer: 1305105

Writer: 1305110

Leader: 1305059

**Project Overview**

Image captioning (automatically generating natural language image descriptions) is useful for the visually impaired and natural language based image search. It is significantly more challenging than classic vision tasks such as object recognition and image classification. The structured output space of well-formed natural language sentences is significantly more challenging to predict over than just a set of class labels. To overcome this challenge, Recurrent Neural Network (RNN) is used. These methods use a convolutional neural network (CNN) to encode the input image into a compact representation. A recurrent neural network (RNN) is used to decode this representation word-by-word into a natural language description of the image.

**Problem statement**

The image dataset contains description for each image. Our goal is to predict realistic captions for new images. Deep learning based techniques (CNNs) has been very popular in the last few years where they consistently outperformed traditional approaches for feature extraction to the point of winning image-net challenges. We used CNN to extract features. For generating captions we have to predict the words of the caption sequentially. So it needs to take the previous words as input to predict the next word. For this task we used long short-term memory (LSTM). In neural image captioning systems, a LSTM is typically viewed as the primary generation component. This view suggests that the image features should be injected into the LSTM. This is in fact the dominant view in the literature. Alternatively, the LSTM can instead be viewed as only encoding the previously generated words. This view suggests that the LSTM should only be used to encode linguistic features and that only the final representation should be merged with the image features at a later stage.

**Performance metrics**

One measure that can be used to evaluate the skill of the model is BLEU (bilingual evaluation understudy) scores. BLEU was one of the first [metrics](https://en.wikipedia.org/wiki/Metric_%28mathematics%29) to claim a high [correlation](https://en.wikipedia.org/wiki/Correlation) with human judgments of quality. BLEU’s output is always a number between 0 and 1. This value indicates how similar the candidate text is to the reference texts, with values closer to 1 representing more similar texts. Few human translations will attain a score of 1, since this would indicate that the candidate is identical to one of the reference translations. For this reason, it is not necessary to attain a score of 1.

**Data description**

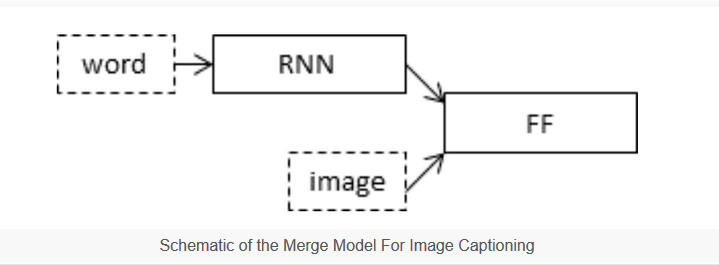
For image captioning we used Flickr8K dataset. The reason is because it is realistic and relatively small. This contains 8092 images in JPG format. There is also a token.txt file that contains image id and description for that image. We extracted a subset sized 150 from the dataset. For this subset we have trained, validated and tested our model.

**Data preprocessing**

We have cleaned the text in order to reduce the size of the vocabulary of words we need to work with. First we have normalized text description by converting them all in lower case. Then removed all punctuation, removed all words that are one character or less in length, removed all words with numbers in them. These cleaned descriptions are then saved in description.txt file. This task is done in “clean\_descriptions**”** function.

**Model description**

We have used a deep learning based on the “merge-model” described by Marc Tanti, et al. in their 2017 paper “[Where to put the Image in an Image Caption Generator](https://arxiv.org/abs/1703.09137)”.

****

Our model has three parts:

**Photo Feature Extractor**: This is a 16-layer VGG model pre-trained on the ImageNet dataset. We have pre-processed the photos with the VGG model (without the output layer) and will use the extracted features predicted by this model as input.



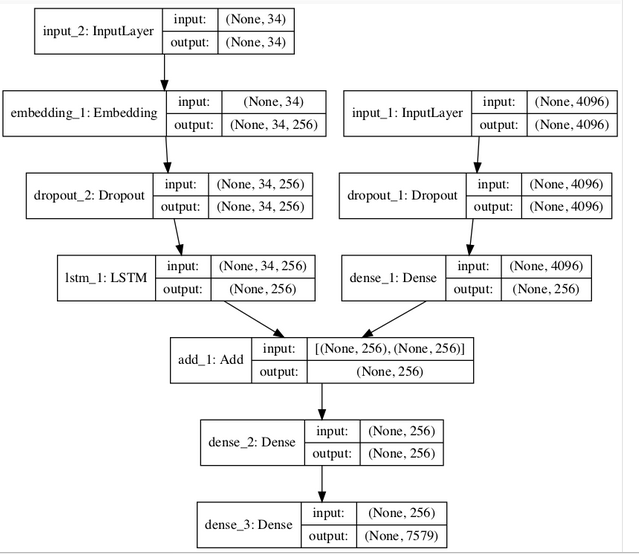
Fig: VGG16 model architecture

**Sequence Processor**: This is a word embedding layer for handling the text input, followed by a Long Short-Term Memory (LSTM) recurrent neural network layer.

**Decoder:** Both the feature extractor and sequence processor output a fixed-length vector. These are merged together and processed by a dense layer to make a final prediction.

Output of the VGG16 model is a vector of 4,096 elements. For regularization purpose we have used a dropout layer. Then it is fed to a dense layer. The Sequence Processor model expects input sequences with a pre-defined length (max length of the descriptions) which are fed into an Embedding layer that uses a mask to ignore padded values. This input vector is then passed through a dropout layer. This is followed by an LSTM layer. The Decoder model merges the vectors from both input models using an addition operation. This is then fed to a dense neuron layer and then to a final output dense layer that makes a softmax prediction over the entire output vocabulary for the next word in the sequence.

**Architecture description**



**Algorithm and techniques**

**Feature extraction:** The VGG architecture is composed entirely of 3x3 convolutional and max-pooling layers, with a fully connected block at the end.

**Layers:**

* **Convolution:** Convolutional layers convolve around the image to detect edges, lines, blobs of colors and other visual elements. Convolutional layer’s hyper-parameters are the number of filters, filter size, stride, padding and activation functions for introducing non-linearity.
* **MaxPooling:** Pooling layers reduces the dimensionality of the images by removing some of the pixels from the image. Maxpooling replaces a n x n area of an image with the maximum pixel value from that area to downsample the image.
* **Dropout:** Dropout is a simple and effective technique to prevent the neural network from overfitting during the training. Dropout is implemented by only keeping a neuron active with some probability p and setting it to 0 otherwise. This forces the network to not learn redundant information.
* **Flatten:** Flattens the output of the convolution layers to feed into the Dense layers.
* **Dense**: Dense layers are the traditional fully connected networks that maps the scores of the convolutional layers into the correct labels with some activation function (softmax , ReLU).

**Activation functions:**

Activation layers apply a non-linear operation to the output of the other layers such as convolutional layers or dense layers.

**ReLU:** The ReLU is the most used activation function in the world right now. It is used in almost all the convolutional neural networks or deep learning. f(z) is zero when z is less than zero and f(z) is equal to z when z is above or equal to zero. We have used ReLU in dense layer.



**Softmax:** [Softmax function](https://en.wikipedia.org/wiki/Softmax_function) is applied to the output layer to convert the scores into probabilities that sum to 1.

**Optimizers:**

**Adam**: [Adam](https://en.wikipedia.org/wiki/Stochastic_gradient_descent#Adam) (Adaptive moment estimation) is an update to RMSProp optimizer in which the running average of both the gradients and their magnitude is used. In practice Adam is currently recommended as the default algorithm to use, and often works slightly better than RMSProp.

**Training procedure:**

At first we have loaded the descriptions of the train images from description.txt. Then we have wrapped each description with two strings “startseq” and “endseq” at the starting and ending position respectively. This is because we needed a ‘first word’ to kick-off the generation process and a ‘last word‘ to signal the end of the caption. This is done in “Load\_clean\_descriptions” function.

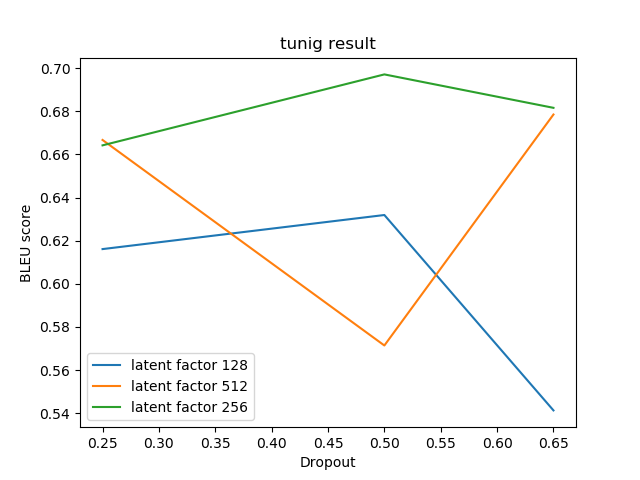
Each caption has been split into words. The model would be provided one word and the photo and generate the next word. Then the first two words of the description would be provided to the model as input with the image to generate the next word. This is how the model would be trained.

For example, the input sequence “little girl running in field” would be split into 6 input-output pairs to train the model:

|  |
| --- |
| X1, X2 (text sequence), y (word)  photo startseq, little  photo startseq, little, girl  photo startseq, little, girl, running  photo startseq, little, girl, running, in  photo startseq, little, girl, running, in, field  photo startseq, little, girl, running, in, field, endseq  This is done in “create\_sequences” function. The output of this function is fed to the model while training.  Later, when the model is used to generate descriptions, the generated words would be concatenated and recursively provided as input to generate a caption for an image. There are two input arrays to the model: one for photo features and one for the encoded text. There is one output for the model which is the encoded next word in the text sequence. |
|  |

**Hyper-parameter tuning**

Latent factor of embedding layer and the number of nodes to be dropped out are the hyper-parameters of our model. We have tuned latent factor for 128,256,512 and dropout for 0.25, 0.5, 0.65.



**Result description**

We have evaluated our model with three types of dataset (50,100,150). Performance has been increased with increasing size of dataset.

**Dataset of length 50:**

BLEU-1: 0.454905

BLEU-2: 0.287979

BLEU-3: 0.173670

BLEU-4: 0.000000

**Dataset of length 100:**

BLEU-1: 0.777778

BLEU-2: 0.512748

BLEU-3: 0.364142

BLEU-4: 0.191760

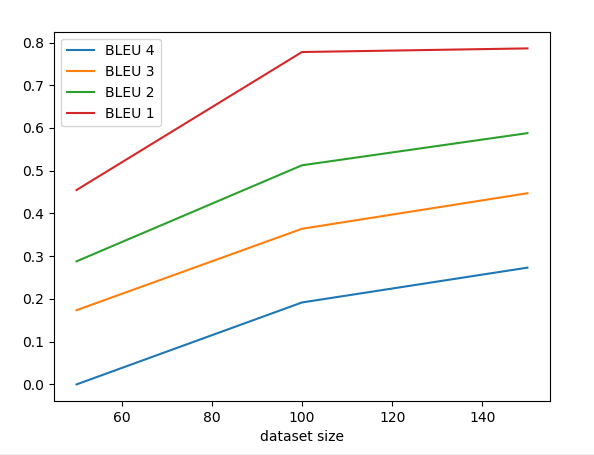
**Dataset of length 150:**

BLEU-1: 0.786260

BLEU-2: 0.587948

BLEU-3: 0.447318

BLEU-4: 0.273210



**Conclusion:**

After training the model with 150 data, the model can generate better caption than training with 100 and 50 data. The generated caption is much closer to the real one. A few examples are given below:



**dog win stick in the water**



**dog running in the grass**

****

**two dogs are running in the grass**