Python Environment

- Python SciPy environment installed, ideally with Python 3.
- Keras installed with the TensorFlow backend.
- NumPy and NLTK installed.
- Python Environment for Deep Learning with Anaconda

Photo and Caption Dataset

- Download the datasets and unzip them into your current working directory.

Prepare Photo Data

- Visual Geometry Group, or VGG, model that won the ImageNet competition in 2014 in use.
- Keras will download the model weights from the Internet, which are about 500 Megabytes. This may take a few minutes depending on your internet connection.
- Pre-compute the "photo features" using the pre-trained model and save them to file.
- Remove the last layer from the loaded model, as this is the model used to predict a classification for a photo.
- Keras also provides tools for reshaping the loaded photo into the preferred size for the model (e.g. 3 channel 224 x 224 pixel image).
- Function named *extract_features()* that, given a directory name, will load each photo, prepare it for VGG, and collect the predicted features from the VGG model. The image features are a 1-dimensional 4,096 element vector.
- Running this data preparation step may take a while depending on your hardware, perhaps one hour on the CPU with a modern workstation.
- At the end of the run, you will have the extracted features stored in '*features.pkl*' for later use. This file will be about 127 Megabytes in size.

Prepare Text Data

- Each photo has a unique identifier. This identifier is used on the photo filename and in the text file of descriptions.
- *load_descriptions()* that, given the loaded document text, will return a dictionary of photo identifiers to descriptions. Each photo identifier maps to a list of one or more textual descriptions.
- The descriptions are already tokenized and easy to work with. *clean_descriptions()* function that, given the dictionary of image identifiers to descriptions, steps through each description and cleans the text.
    1. Convert all words to lowercase.
    2. Remove all punctuation.
    3. Remove all words that are one character or less in length (e.g. 'a').
    4. Remove all words with numbers in them.

A smaller vocabulary will result in a smaller model that will train faster. Save the dictionary of image identifiers and descriptions to a new file named *descriptions.txt*, with one image identifier and description per line. *save_descriptions()* function that, given a dictionary containing the mapping of identifiers to descriptions and a filename, saves the mapping to file. Finally, the clean descriptions are written to '*descriptions.txt*'.

Develop Deep Learning Model

This section is divided into the following parts:
Loading Data.

- Train the data on all of the photos and captions in the training dataset.
- Function *load_set()* below will load a pre-defined set of identifiers given the train or development sets filename.
- *load_clean_descriptions()* that loads the cleaned text descriptions from '*descriptions.txt*' for a given set of identifiers and returns a dictionary of identifiers to lists of text descriptions.The model we will develop will generate a caption given a photo, and the caption will be generated one word at a time. The sequence of previously generated words will be provided as input. Therefore, we will need a '*first word*' to kick-off the generation process and a '*last word*' to signal the end of the caption.
- The strings '*startseq*' and '*endseq*' for this purpose. These tokens are added to the loaded descriptions as they are loaded. It is important to do this now before we encode the text so that the tokens are also encoded correctly.
- *load_photo_features()* that loads the entire set of photo descriptions, then returns the subset of interest for a given set of photo identifiers.
- The first step in encoding the data is to create a consistent mapping from words to unique integer values. Keras provides the *Tokenizer* class that can learn this mapping from the loaded description data.
- *to_lines()* to convert the dictionary of descriptions into a list of strings and the *create_tokenizer()* function that will fit a Tokenizer given the loaded photo description text.
- Each description will be split into words. The model will be provided one word and the photo and generate the next word. Then the first two words of the description will be provided to the model as input with the image to generate the next word. This is how the model will be trained.

```
1 X1,     X2 (text sequence),                      y (word)
2 photo   startseq,                                little
3 photo   startseq, little,                        girl
4 photo   startseq, little, girl,                  running
5 photo   startseq, little, girl, running,         in
6 photo   startseq, little, girl, running, in,     field
7 photo   startseq, little, girl, running, in, field, endseq
```

- *create_sequences()*, given the tokenizer, a maximum sequence length, and the dictionary of all descriptions and photos, will transform the data into input-output pairs of data for training the model. 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.

- The input text is encoded as integers, which will be fed to a word embedding layer. The photo features will be fed directly to another part of the model. The model will output a prediction, which will be a probability distribution over all words in the vocabulary.
- The output data will therefore be a one-hot encoded version of each word, representing an idealized probability distribution with 0 values at all word positions except the actual word position, which has a value of 1.

Defining the Model.

- **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.
- **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** (for lack of a better name). 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.

Fitting the Model.

- The saved model with the best skill on the training dataset as our final model.
- a *ModelCheckpoint* in Keras and specifying it to monitor the minimum loss on the validation dataset and save the model to a file that has both the training and validation loss in the filename.Checkpoint in the call to *fit()* via the *callbacks* argument.

Train with Progressive Loading

- The code in the previous section is not memory efficient and assumes you are running on a large EC2 instance with 32GB or 64GB of RAM. If you are running the code on a workstation of 8GB of RAM, you cannot train the model.
- *data_generator()* will be the data generator and will take the loaded textual descriptions, photo features, tokenizer and max length.
- *create_sequence()* function to create a batch worth of data for a single photo rather than an entire dataset. This means that we must update the *create_sequences()* function to delete the "iterate over all descriptions" for-loop.
- *fit_generator()* function on the model to train the model with this data generator.

Evaluate Model

- *generate_desc()* implements this behavior and generates a textual description given a trained model, and a given prepared photo as input. It calls the function *word_for_id()* in order to map an integer prediction back to a word.
- *evaluate_model()* will evaluate a trained model against a given dataset of photo descriptions and photo features. The actual and predicted descriptions are collected and evaluated collectively using the corpus BLEU score that summarizes how close the generated text is to the expected text.

Citation:

Md. Mijanur Rahman, Ashik Uzzaman, Sadia Islam Sami, "Image captioning using deep neural network based model", Department of Computer Science and Engineering, Jatiya Kabi Kazi Nazrul Islam University. Github Repository, 2021. https://github.com/mijancse/image-captioning-using-deep-neural-network-based-model