

This is a Python script that imports several libraries, including TensorFlow, NumPy, and NLTK. The script is likely used for image captioning using a deep learning approach.

Here's a brief summary of what the code does:

* Imports necessary libraries for deep learning, image processing, and natural language processing.
* Defines several functions and classes for image and caption preprocessing, model building, and evaluation.
* Loads pre-trained VGG16 model for feature extraction from images.
* Loads pre-processed captions and images.
* Tokenizes captions and creates a word-to-index dictionary.
* Builds a deep learning model using LSTM layers for sequence generation.
* Trains the model on the pre-processed data.
* Evaluates the trained model using BLEU score metric.
* Saves the trained model.
* The code also includes some visualization of the training process and model architecture.

Overall, this code is likely used for training an image captioning model using a deep learning approach and evaluating its performance.

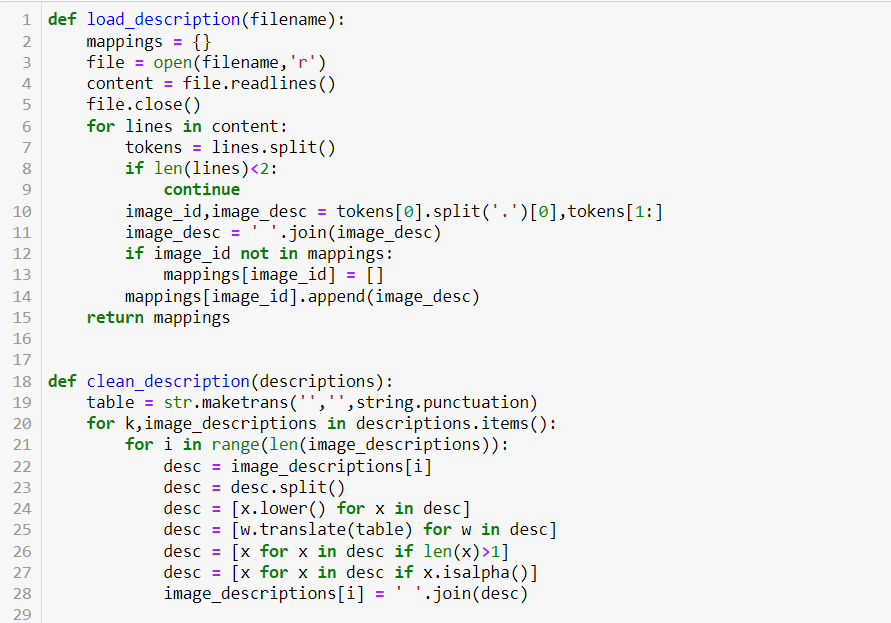


This is a Python function that uses a pre-trained VGG16 model to extract features from images in a directory. The function takes in the directory path as an argument.

Here's a brief summary of what the code does:

* Loads the pre-trained VGG16 model.
* Removes the last layer of the model (the classification layer) to get the feature extraction layer.
* Creates a new model with the feature extraction layer as the output.
* Loads images from the specified directory and extracts features from them using the VGG16 model.
* Reshapes and preprocesses the images before feeding them into the model.
* Saves the extracted features in a dictionary, where each key is the image ID and the value is its corresponding features.
* The returned features dictionary can then be used to train an image captioning model or for any other image-related tasks that require feature extraction.

The last two lines of the code call the extract\_features function and pass in the directory path. The returned features are saved to the features variable.



This block of code defines four Python functions: load\_description, clean\_description, create\_corpus, and save\_descriptions.

The load\_description function takes in the filename of a text file containing image IDs and corresponding image descriptions, reads in the file, and parses the image IDs and descriptions into a Python dictionary where each key is an image ID and the corresponding value is a list of its image descriptions.

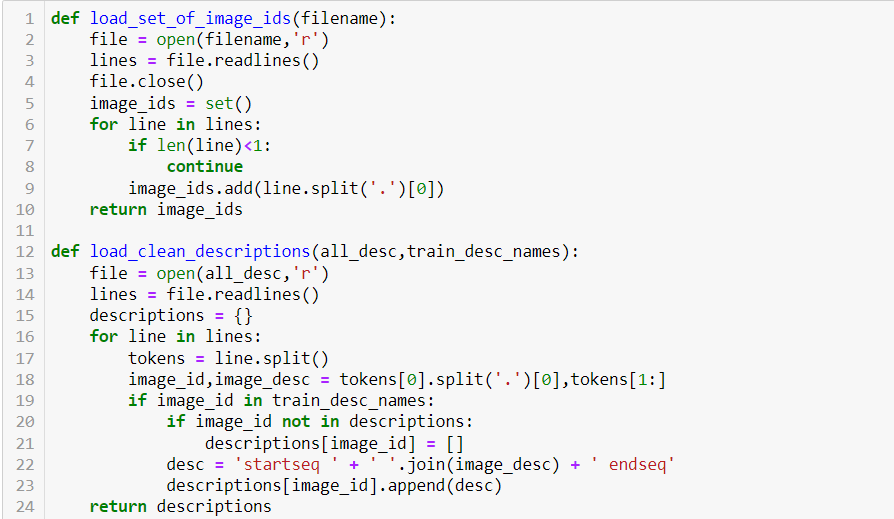
The clean\_description function takes in the dictionary of image descriptions and preprocesses the descriptions by converting all words to lowercase, removing punctuation, and removing any words that are not alphabetic or have a length of one.

The create\_corpus function takes in the preprocessed dictionary of image descriptions and creates a set of unique words (the "corpus") that appear in the descriptions.

The save\_descriptions function takes in the preprocessed dictionary of image descriptions and saves them to a text file where each line contains an image ID and its corresponding preprocessed image description.

The remaining lines of code call these functions in sequence to load the image descriptions from the text file, preprocess them, create the corpus of words, and save the preprocessed descriptions to a text file.

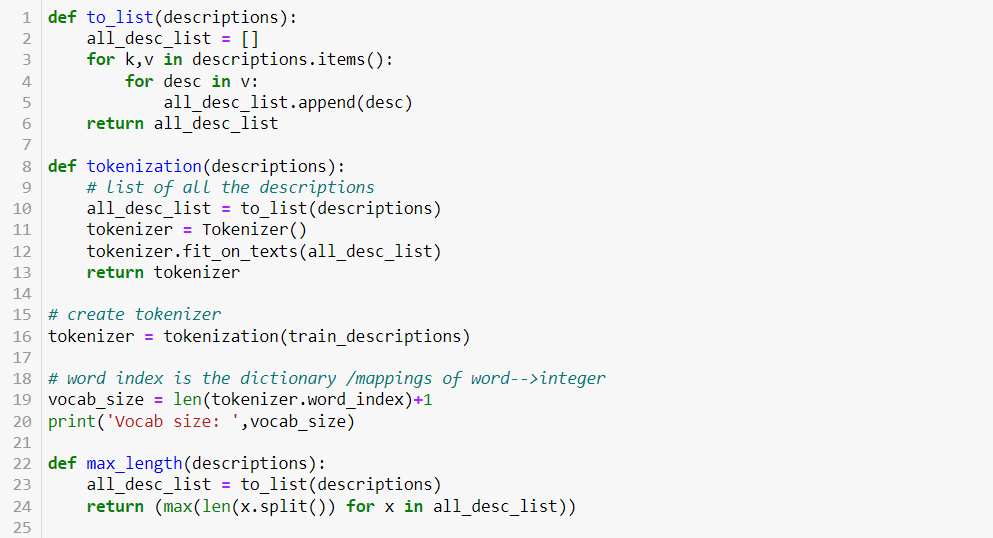
The output of the code includes the number of descriptions loaded, the length of the vocabulary, and a message indicating that the preprocessed descriptions have been saved to a text file.

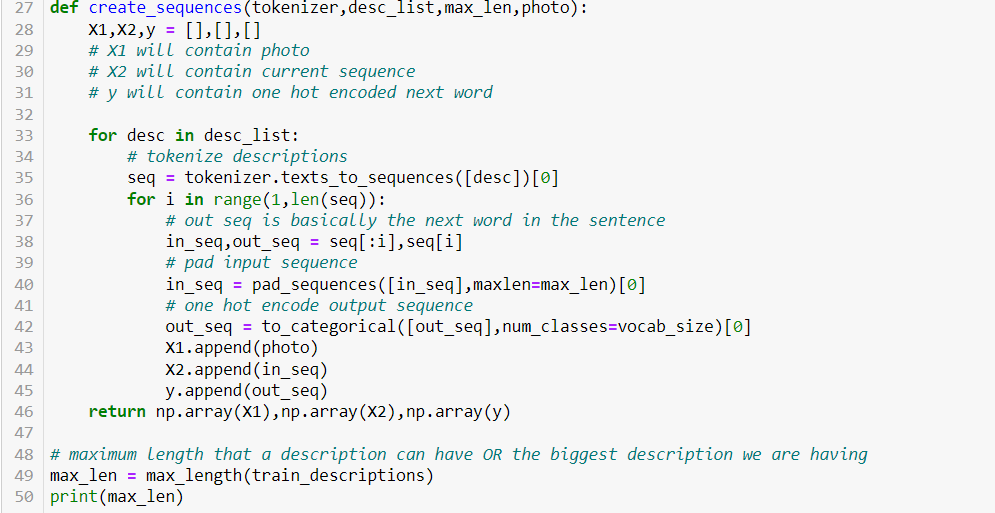




This code loads the training image ids from the file "Flickr\_8k.trainImages.txt". Then, it loads the descriptions of all the images from the file "descriptions.txt" and filters out only the training image descriptions. These descriptions are preprocessed by adding a start token "startseq" at the beginning of each description and an end token "endseq" at the end of each description. Finally, it loads the pre-extracted features for the training images from the file "Flicker\_dataset\_image\_features.pkl" and filters out only the features corresponding to the training images.

The output of the last line of code shows the loaded training descriptions.





The to\_list() function is used to convert the dictionary of image IDs and their corresponding descriptions into a single list of all the descriptions. This is useful when we want to tokenize all the descriptions together as a single corpus.

The tokenization() function takes in the dictionary of image IDs and their corresponding descriptions, and returns a Tokenizer object that has been fit on all the descriptions. This tokenizer is used to convert the textual descriptions into sequences of integers, which can then be used as input to the model.

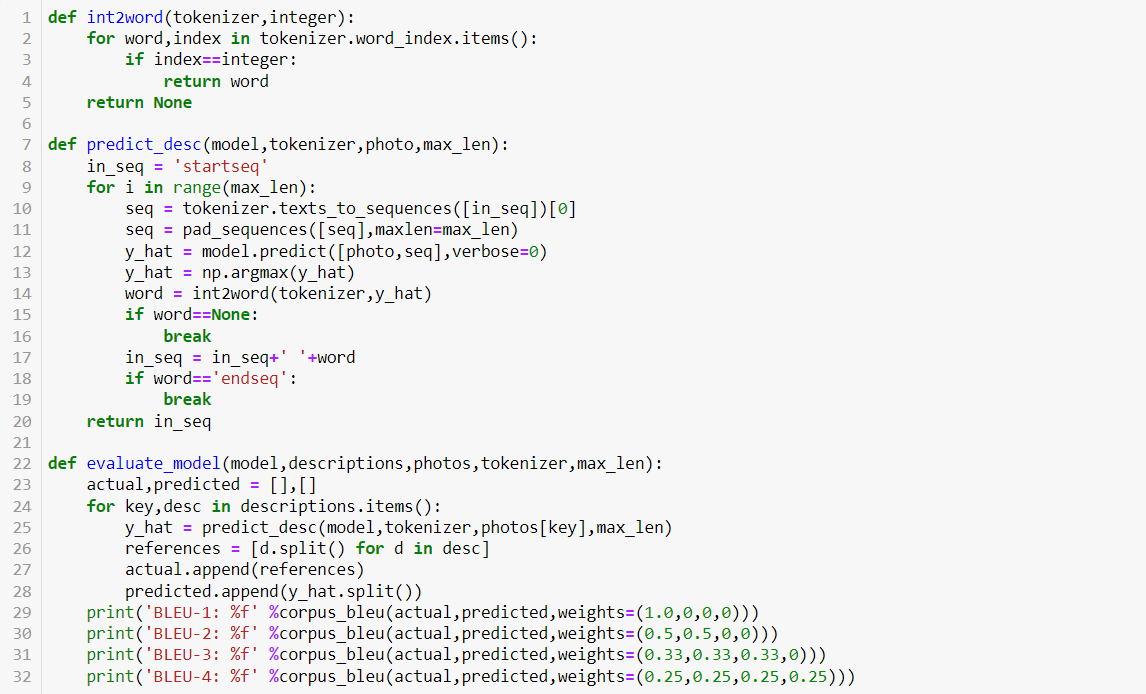
The max\_length() function calculates the maximum length of any description in the dataset. This is used to pad the descriptions to a fixed length when creating sequences of input-output pairs for the model.

The create\_sequences() function is used to create input-output sequences that can be used to train the model. It takes in the Tokenizer object, a list of descriptions, the maximum length of any description, and the corresponding image features. It then tokenizes each description, creates input-output pairs of sequences and their corresponding one-hot encoded output words, pads the input sequences to the maximum length, and returns these as numpy arrays.



The data\_generator function is a generator function that creates data batches for training the model. It takes in the descriptions of the images, their corresponding features, the tokenizer object, and the maximum length of the sequence. It then creates a generator object that yields batches of input-output pairs for the model.

The define\_model function defines the neural network model for the image captioning task. It takes in the vocabulary size and the maximum length of the sequence as input. The model has two input layers - one for the image features and one for the sequence of words in the description. The image features are passed through a dropout layer and a dense layer to reduce their dimensions. The sequence of words is passed through an embedding layer, a dropout layer, and an LSTM layer to capture the temporal dependencies between the words. The output of the LSTM layer is concatenated with the output of the dense layer for the image features and passed through two dense layers to output the predicted next word. The model is compiled with the categorical cross-entropy loss function and the Adam optimizer.



These are additional functions for the image captioning model:

* int2word(tokenizer,integer): This function takes a tokenizer and an integer as input and returns the corresponding word in the tokenizer's word\_index dictionary. It is used to map the integer predictions from the model back to their original words.
* predict\_desc(model,tokenizer,photo,max\_len): This function takes the trained model, tokenizer, image, and maximum sequence length as input, and returns the predicted description for the given image. It starts with the input sequence "startseq" and generates words one by one until the maximum length is reached or the "endseq" token is generated.
* evaluate\_model(model,descriptions,photos,tokenizer,max\_len): This function takes the trained model, descriptions, photos, tokenizer, and maximum sequence length as input, and prints the BLEU-1, BLEU-2, BLEU-3, and BLEU-4 scores for the predicted captions compared to the actual captions in the test set. The BLEU score is a metric used to evaluate the quality of machine-translated text, where higher scores indicate better quality translations. The BLEU score ranges from 0 to 1.

BLEU score measures the similarity between the generated captions and the reference captions. It does this by comparing n-grams (contiguous sequences of n words) in the generated caption to n-grams in the reference captions. BLEU-1 measures the similarity of unigrams (one-word sequences), BLEU-2 measures the similarity of bigrams (two-word sequences), BLEU-3 measures the similarity of trigrams (three-word sequences), and BLEU-4 measures the similarity of four-gram sequences.

We need four BLEU scores because each score gives us a different level of granularity in evaluating the quality of the generated captions. For example, if the model generates captions that are very similar to the reference captions in terms of one-word sequences (BLEU-1), but not very similar in terms of two- or three-word sequences (BLEU-2 and BLEU-3), it might indicate that the model is good at generating individual words, but not so good at generating longer phrases or sentences. The four BLEU scores together give us a more comprehensive view of the model's performance.