

Image Captioning using Flickr8k Dataset

GROUP 6 - Final Project
DATS 6203(10)
Machine Learning II

Instructor: Prof. Amir Jafari
By: Hemanth K., Madhuri Y., Sharmin K.

Introduction

This project aims to achieve Image Captioning using Deep Learning Techniques, mainly Convolutional Neural Networks and Long Short-Term Memory. The dataset used for this project is the Flickr8k dataset, found easily on Kaggle. Flickr8k, and others like it, such as the COCO dataset, have created the benchmark for sentence-based image captioning.

The scope of the project incorporates different domains under Artificial Intelligence such as Computer Vision, Deep Learning, and Natural Language Processing. Applications and business value of this project include:

- ❑ Image Search in Search Engines
- ❑ Image Segregation and Classification
- ❑ Text-to-Speech to aid visually impaired individuals
- ❑ Automatic Image annotation in Facial Recognition, E-commerce, etc.

The project is divided into 5 main components:

1. Data Splitting – generating the train and test ids of the images
2. Text Preprocessing – preprocessing the captions before being fed into the LSTM model
3. Image Preprocessing – generating image features using CNN models
4. LSTM – Caption generator – generates captions
5. Model Evaluation – using various performance metrics

Dataset Description

The Flickr8k dataset has 2 main components – Images and their associated Captions. There are 8092 images, each having 5 captions in the captions.txt file. Hence, there are a total of 40,460 captions. The dataset can be found [here](#).

Train-Test Splitting

The Dataset list of images is taken from the Images folder of the dataset and the list is divided into 80%-20% train and test respectively. The code generates two files train_ids.txt and test_ids.txt which have the list of image ids/names.

Deep Learning Algorithms and Techniques

Text Pre-Processing

The text preprocessing was done to generate clean captions and enumerated caption sequences which will be fed into the LSTM model for training. The captions were transformed as follows. The image captions were loaded as follows and converted to a dictionary with image id as key and a list of all corresponding captions as values.

```
image,caption
1000268201_693b08cb0e.jpg,A child in a pink dress is climbing up a set of stairs in an entry way .
1000268201_693b08cb0e.jpg,A girl going into a wooden building .
1000268201_693b08cb0e.jpg,A little girl climbing into a wooden playhouse .
1000268201_693b08cb0e.jpg,A little girl climbing
```

The captions were then cleaned as follows. The unique words in the captions were identified as follows. There are 8680 unique words.

```
'99679241_adc853a5c0': ['grey bird stands majestically on a beach while waves roll in .', 'large bird stands in the water on the beach .', 'tall bird is standing on the sand beside the ocean .', 'water bird standing at the ocean 's edge .', 'white crane stands tall as it looks out upon the ocean .'], '997338199_7343367d7f': ['person stands near
```

```
mountain scene', 'children getting ready to sled', 'people are sitting together in the snow'], '99679241_adc853a5c0': ['grey bird stands majestically on beach while waves roll in', 'large bird stands in the water on the beach', 'tall bird is standing on the sand beside the ocean', 'water bird standing at the ocean edge', 'white crane stands tall as it looks out upon the ocean'], '997338199_7343367d7f': ['person stands near golden walls', 'woman behind scrolled wall is
```

```
{ 'trips', 'kidsized', 'banner', 'counter', 'rabbits', 'creating', 'argues', 'cardigan', 'scooter',
Original Vocabulary Size: 8680
```

The image ids of the train set (split beforehand) are loaded. There are 6468 train images.

```
'3265162450_5b4e3c5f1b', '2220175999_081aa9cce8', '3556598205_86c180769d', '2672354635_3a03f76486',
'103195344_5d2dc613a3', '2695085448_a11833df95', '2278110011_ba846e7795', '3038941104_17ee91fc03',
'2480668859_6f9b46be6a', '2704257993_d485058a5f', '242558556_12f4d1cab', '2451988767_244bff98d1',
'1394620454_bf708cc501'}
```

```
Dataset: 6468
```

The captions of the train images were cleaned and appended with a 'startseq' and an 'endseq' token as the start and end of each caption. This is done to provide a uniform starting partial caption to the language model.

```
patterns endseq', 'startseq writing on pad in room with gold decorated walls endseq'], '997722733_0cb5439472':  
['startseq man in pink shirt climbs rock face endseq', 'startseq man is rock climbing high in the air endseq',  
'startseq person in red shirt climbing up rock face covered in assist handles endseq', 'startseq rock climber in red  
shirt endseq', 'startseq rock climber practices on rock climbing wall endseq']}]  
Length of all train captions : 32340
```

Input = Image_1 + 'startseq'; Output = 'the'

We then generate a list of all cleaned descriptions alone, which is used to find the length of the longest caption. This maximum length is taken into account while padding the partial caption sequence. The 'ixtoward' and 'wordtoix' dictionaries contain words and corresponding indices. The sequence of indices is passed to the model, rather than the actual text caption.

```
'tall bird is standing on the sand beside the ocean', 'water bird standing at the ocean edge', 'white crane stands tall  
as it looks out upon the ocean', 'person stands near golden walls', 'woman behind scrolled wall is writing', 'woman  
standing near decorated wall writes', 'walls are covered in gold and patterns', 'writing on pad in room with gold  
decorated walls', 'man in pink shirt climbs rock face', 'man is rock climbing high in the air', 'person in red shirt  
climbing up rock face covered in assist handles', 'rock climber in red shirt', 'rock climber practices on rock climbing  
wall']  
Description Length: 31
```

Preprocessed words 7838 -> 1725
Vocab size : 1726
Description Length: 31

Image Pre-Processing

Before passing the images to the encoder images are preprocessed. The train image ids received from the previous step are read id by id, and each image is then resized to the input size used by CNN model (244X244 for VGG-16). The resized image is then converted to array and reshaped to pass it to the CNN model which returns the encoded image feature to pass to the LSTM model. Similar process is followed with the test id's and the encoded features are passed to the prediction model.

Image Model:

Convulated Neural Networks work here as an encoder model. As we know since CNN uses Kernels, the model could be used to extract features from the images and this step we denote

as an image encoding. There are a lot of models that we can use like VGG-16, InceptionV3, ResNet, etc. Which are generally used for image captioning. We tried All the three models for our project to see which encoder gives the best results. Since we found VGG-16 gives the best results with our data set let us discuss the details of the model.

About Best feature extractor - VGG-16:

VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman. The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes.

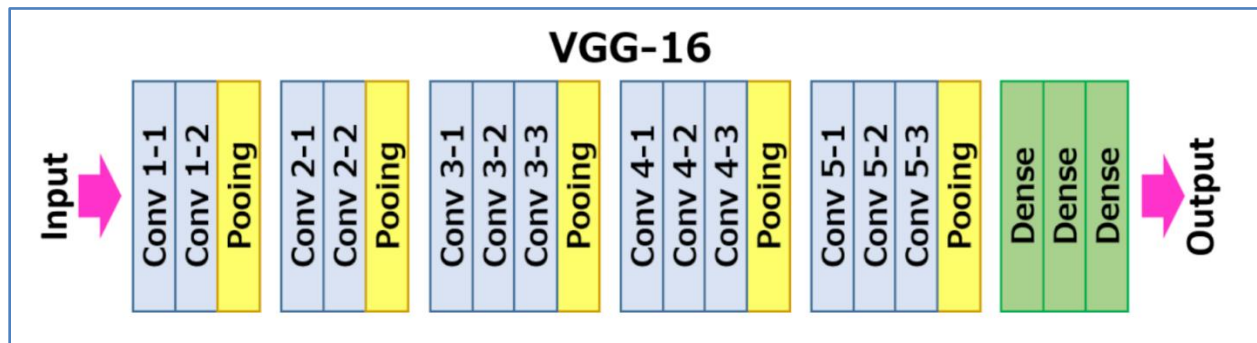


Figure 2: Layers of VGG-16

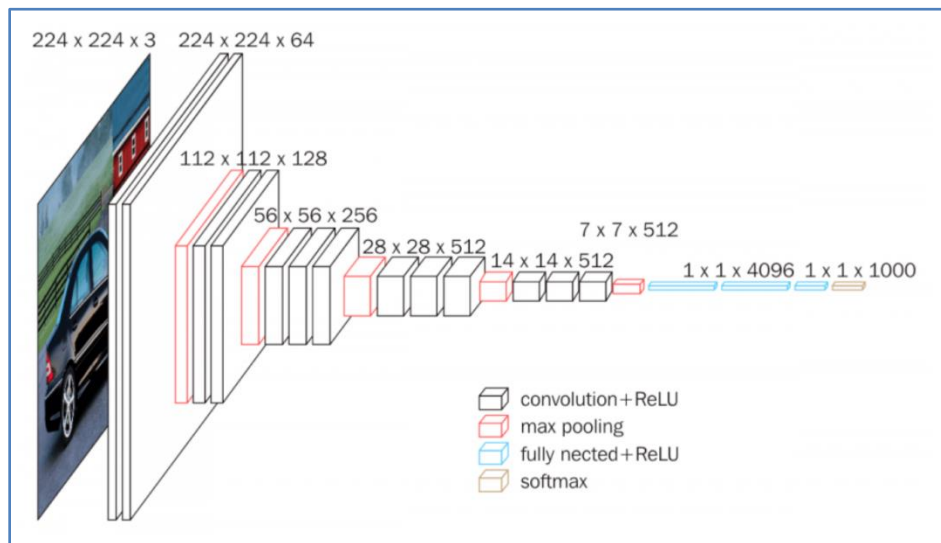


Figure 3: The architecture of the VGG-16.

The Model is named as VGG-16 as it uses 16 hidden layers in its architecture. The input to the cov1 layer is of fixed size 224 x 224 RGB image. The image is passed through a stack of convolutional layers, where the filters with very small receptive fields: 3×3 & 1×1 are used. The network uses Max-pooling with stride of 2. It has Three Fully Connected (FC) layers; the first

two have 4096 channels each. The third layer is the soft-max layer containing 1000 channels. The model has 138,357,544 parameters in total. The summary of the model is as follows.

Model: "vgg16"

Layer (type)	Output Shape	Param #
=====		
=====		
input_1 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808

block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000

=====

=====

Total params: 138,357,544
 Trainable params: 138,357,544
 Non-trainable params: 0

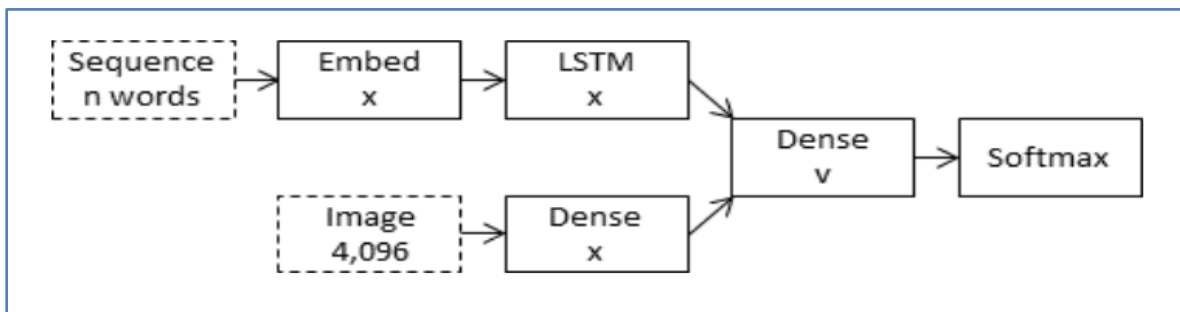
As the purpose of using CNN here is not for image classification and is for feature extraction, we eliminate the last 2 dense layers to get the flattened features which could be passed to our Decoder model.

Other than VGG-16 we tried to use our own networks, which was not a good encoder, hence based on our research on image captioning we also tried InceptionV3 (Size: 92MB, Top-5 Acc: 0.937, Parameters: 23,851,784), ResNet50V2 (Size: 98MB, Top-5 Acc: 0.93, Parameters: 25,613,800). These models are also well-known feature extractors for image captioning purposes.

The `extract_features` method implemented generates a pickle file. The pickle file contains all the images features as a dictionary with image name as key and features generated by the CNN model (size 4096 each) as its value. The Pickle file is then extracted as a dictionary before being passed to the decoder model.

Experimental Setup

We tackled this problem using an Encoder-Decoder model. Here our encoder model will combine both the encoded form of the image and the encoded form of the text caption and feed to the decoder. Our model will treat CNN as the 'image model' and the RNN/LSTM as the 'language model' to encode the text sequences of varying length. The vectors resulting from both the encodings are then merged and processed by a Dense layer to make a final prediction.



To encode our image features, we made use of transfer learning. We used VGG-16, InceptionV3, ResNet pre-trained models. Out of these, VGG-16 gave better results compared to others.

To encode our text sequence, we will map every word to a 200-dimensional vector. For this will use a pre-trained Glove model. This mapping will be done in a separate layer after the input layer called the embedding layer. Glove Embedding basically maps words to a vector space, where similar words are clustered together, and different words are separated. The advantage of using Glove over Word2Vec is that GloVe does not just rely on the local context of words, but it incorporates global word co-occurrence to obtain word vectors. For our model, we will map all the words in our 31-word long caption to a 200-dimension vector using Glove.

We created a Merge model where we combined the image vector and the partial caption. Therefore, our model will have 3 major steps:

1. Processing the sequence from the text
2. Extracting the feature vector from the image
3. Decoding the output using softmax by concatenating the above two layers

Summary of our model is attached below:

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	[(None, 31)]	0	
input_1 (InputLayer)	[(None, 4096)]	0	
embedding (Embedding)	(None, 31, 200)	345200	input_2[0][0]
dropout (Dropout)	(None, 4096)	0	input_1[0][0]
dropout_1 (Dropout)	(None, 31, 200)	0	embedding[0][0]
dense (Dense)	(None, 256)	1048832	dropout[0][0]
lstm (LSTM)	(None, 256)	467968	dropout_1[0][0]
add (Add)	(None, 256)	0	dense[0][0] lstm[0][0]
dense_1 (Dense)	(None, 256)	65792	add[0][0]
dense_2 (Dense)	(None, 1726)	443582	dense_1[0][0]
Total params: 2,371,374			
Trainable params: 2,026,174			
Non-trainable params: 345,200			

Input_2 is the partial caption of max length 31 which is fed into the embedding layer. This is where the words are mapped to the 200-d Glove embedding. It is followed by a dropout of 0.5 to avoid overfitting. This is then fed into the LSTM for processing the sequence.

Input_1 is the image vector extracted by our VGG-16 network. It is followed by a dropout of 0.5 to avoid overfitting and then fed into a Fully Connected layer.

Both the Image model and the Language model are then concatenated by adding and fed into another Fully Connected layer. The layer is a softmax layer that provides probabilities to our 1726-word vocabulary.

We compiled the model using Categorical_Crossentropy as the Loss function and Adam as the optimizer with a learning rate of 0.001. We tried different optimizers with different learning rates and Adam with a learning rate of 0.001 gave best results. Since our dataset has 6468 images and 32340 captions, we created a function that can train the data in batches.

Next, we trained our model for 15 epochs with a batch size of 3 and we saved the model at each epoch. Initially, we trained the model using conventional batch sizes of 16,32,64,128 and then we trained with much smaller batch sizes such as 1,3,5. Smaller batches yielded better results and a batch size of 3 gave best results.

Results

As the model generates a 1725 long vector with a probability distribution across all the words in the vocabulary, we greedily pick the word with the highest probability to get the next word prediction. This method is called Greedy Search. Below are some of the captions generated by our model for different images.



dog is jumping into the air to catch a frisbee



man in red shirt is climbing up waterfall

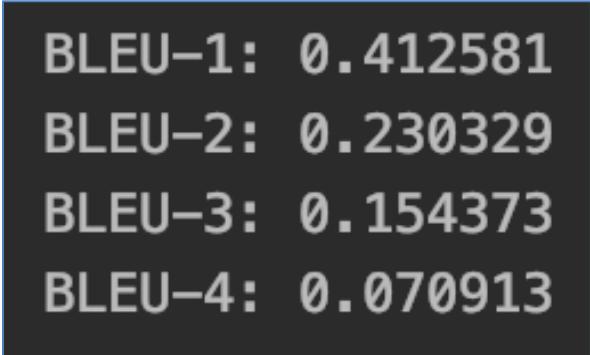


A surfer in blue shorts surfs over water

Model Evaluation

We used the BLEU score metric to evaluate performance of our model. The Bilingual Evaluation Understudy Score, or BLEU for short, is a metric for evaluating a generated sentence to a reference sentence. A perfect match results in a score of 1.0, whereas a perfect mismatch results in a score of 0.0. The approach works by counting matching n-grams in the candidate translation to n-grams in the reference text, where 1-gram or unigram would be each token and a bigram comparison would be each word pair. The comparison is made regardless of word order. The counting of matching n-grams is modified to ensure that it takes the occurrence of the words in the reference text into account, not rewarding a model generated

text that generates an abundance of reasonable words. Below are the cumulative BLEU scores when we predicted on our test set.



```
BLEU-1: 0.412581
BLEU-2: 0.230329
BLEU-3: 0.154373
BLEU-4: 0.070913
```

Summary & Conclusion

Our model gave a decent BLEU score of 0.41 and we got pretty good captions for certain images. While doing this project, we learned how to incorporate the field of Computer Vision and Natural Language Processing together to make predictions. And also, we have learned architectures of different pre-trained models. One of the main improvements that can be made to make the model more robust is by adding additional images and captions by trying the Flickr30K dataset which has a lot of images and captions. And also, we can make use of the beam search instead of greedy search at the output softmax. Beam Search is where we take top k predictions, feed them again to the model and then sort them using the probabilities returned by the model.

References

1. [Learning CNN-LSTM Architectures for Image Caption Generation](#)
2. [Image Captioning](#)
3. [Show and Tell: A Neural Image Caption Generator](#)
4. [A Hierarchical Approach for Generating Descriptive Image Paragraphs](#)
5. [VizSeq: A Visual Analysis Toolkit for Text Generation Tasks](#)
6. <https://www.analyticsvidhya.com/blog/2020/11/create-your-own-image-caption-generator-using-keras/>
7. <https://neurohive.io/en/popular-networks/vgg16/>

8. <https://towardsdatascience.com/review-resnet-winner-of-ilsvrc-2015-image-classification-localization-detection-e39402bfa5d8>
9. <https://cloud.google.com/tpu/docs/inception-v3-advanced>
10. [Recurrent Neural Network Regularization](#)
11. [Image Captioning - A Deep Learning Approach](#)
12. <https://hagan.okstate.edu/NNDesign.pdf>
13. <https://kharshit.github.io/blog/2019/01/11/image-captioning-using-encoder-decoder>
14. <https://towardsdatascience.com/image-captioning-in-deep-learning-9cd23fb4d8d2>
15. <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Appendix

```
import os
import pickle
import string
import glob
import numpy as np
from numpy import array
from pickle import dump, load
from keras.applications.vgg16 import VGG16
from keras.applications.inception_v3 import InceptionV3
from keras.applications.resnet_v2 import ResNet50V2
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input
from keras.models import Model, load_model
from keras.layers import LSTM, Embedding, Dense, Dropout
from keras.layers.merge import add
from keras import Input
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from nltk.translate.bleu_score import corpus_bleu
```

```
# -----
```

```
# Function to Split Data in Train set (80%) and Test set (20%)
# Create text files with training and testing ids
```

```
def data_split(data, a=''):
    f = open('{}_ids.txt'.format(a), 'w+')
    for i in range(len(data)):
        f.write(data[i] + '\n')
    f.close()
    f = open('{}_ids.txt'.format(a), 'r')
```

```
f1 = f.readlines()
return len(f1)
```

```
# -----
-----
```

```
# extract features from each photo in the directory
```

```
def extract_features(directory, textfile, m):
```

```
    if m == 1:
```

```
        # load the model
```

```
        model = VGG16()
```

```
        # re-structure the model
```

```
        model = Model(inputs=model.inputs, outputs=model.layers[-2].output)
```

```
        # summarize
```

```
        print(model.summary())
```

```
        m = "vgg16"
```

```
    elif m == 2:
```

```
        # load the model
```

```
        model = InceptionV3(weights='imagenet')
```

```
        # re-structure the model
```

```
        model = Model(inputs=model.inputs, outputs=model.layers[-2].output)
```

```
        # summarize
```

```
        print(model.summary())
```

```
        m = "IV3"
```

```
    else:
```

```
        # load the model
```

```
        model = ResNet50V2()
```

```
        # re-structure the model
```

```
        model = Model(inputs=model.inputs, outputs=model.layers[-2].output)
```

```
        # summarize
```

```
        print(model.summary())
```

```
        m = "ResNet50V2"
```

```
# Read Train ids as a list for feature extraction
```

```
with open(textfile) as f:
```

```
    train_ids = f.read().splitlines()
```

```
# extract features from each photo
```

```
features = dict()
```

```
for name in train_ids:
```

```
    # load an image from file
```

```
    filename = directory + '/' + name
```

```
    image = load_img(filename, target_size=(224, 224))
```

```
    # convert the image pixels to a numpy array
```

```
    image = img_to_array(image)
```

```
    # reshape data for the model
```

```
    image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
```

```
    # prepare the image for the VGG model
```

```
    image = preprocess_input(image)
```

```
    # get features
```

```
    feature = model.predict(image, verbose=0)
```

```
    # get image id
```

```
    image_id = name.split('.')[0]
```

```
    # store feature
```

```
    features[image_id] = feature
```

```
    print('>%s' % name)
```

```
return features, m
```

```

# -----
# Set Directory Path
DATA_DIR = os.getcwd() + "Images"
# -----

data = os.listdir(DATA_DIR) # 8680

train_size = int(0.8 * len(data))

train = data[:train_size]
train_len = data_split(train, a='train') # 6944

test = data[train_size:]
test_len = data_split(test, a='test') # 1736

# -----
# 1 for VGG16, 2 for InceptionV3, 3 for ResNet50V2
features, m = extract_features(DATA_DIR, 'train_ids.txt', 1) # (output size 4096)
trainfile = "train_features8k_" + m + ".pkl"
file = open(trainfile, "wb")
dump(features, file)
file.close()

features, m = extract_features(DATA_DIR, 'test_ids.txt', 1) # (output size 4096)
testfile = "test_features8k_" + m + ".pkl"
file = open(testfile, "wb")
dump(features, file)
file.close()
# -----

file_to_read = open(trainfile, "rb")
loaded_dictionary = pickle.load(file_to_read)
print(loaded_dictionary)

file_to_read = open(testfile, "rb")
loaded_dictionary = pickle.load(file_to_read)
print(loaded_dictionary)

# -----

def load_doc(filename):
    file = open(filename, 'r')
    text = file.read()
    file.close()
    return text

filename = "captions.txt"
doc = load_doc(filename)
# print(doc[:350])

```

```

def load_descriptions(doc):
    mapping = dict()
    for line in doc.split('\n'):
        tokens = line.split()
        if len(line) < 2:
            continue
        image_id, image_desc = tokens[0], tokens[1:]
        image_id = image_id.split('.')[0]
        image_desc = ' '.join(image_desc)
        if image_id not in mapping:
            mapping[image_id] = list()
        mapping[image_id].append(image_desc)
    return mapping

descriptions = load_descriptions(doc)
print('Loaded: %d ' % len(descriptions))

def clean_descriptions(descriptions):
    table = str.maketrans('', '', string.punctuation)
    for key, desc_list in descriptions.items():
        for i in range(len(desc_list)):
            desc = desc_list[i]
            # tokenize
            desc = desc.split()
            # convert to lower case
            desc = [word.lower() for word in desc]
            # remove punctuation from each token
            desc = [w.translate(table) for w in desc]
            # remove hanging 's' and 'a'
            desc = [word for word in desc if len(word)>1]
            # remove tokens with numbers in them
            desc = [word for word in desc if word.isalpha()]
            # store as string
            desc_list[i] = ' '.join(desc)

clean_descriptions(descriptions)

# convert the loaded descriptions into a vocabulary of words
def to_vocabulary(descriptions):
    all_desc = set()
    for key in descriptions.keys():
        [all_desc.update(d.split()) for d in descriptions[key]]
    return all_desc

vocabulary = to_vocabulary(descriptions)
print('Original Vocabulary Size: %d' % len(vocabulary))

def save_descriptions(descriptions, filename):
    lines = list()
    for key, desc_list in descriptions.items():
        for desc in desc_list:
            lines.append(key + ' ' + desc)
    data = '\n'.join(lines)
    file = open(filename, 'w')
    file.write(data)
    file.close()

```



```

save_descriptions(descriptions, 'descriptions.txt')

# load a pre-defined list of photo identifiers
def load_set(filename):
    doc = load_doc(filename)
    dataset = list()
    for line in doc.split('\n'):
        if len(line) < 1:
            continue
        identifier = line.split('.')[0]
        dataset.append(identifier)
    return set(dataset)

# load training dataset
filename = 'train_ids.txt'
train = load_set(filename)
print('Dataset: %d' % len(train))

# Below path contains all the images
images = 'Images/'
# Create a list of all image names in the directory
img = glob.glob(images + '*.jpg')

# Below file contains the names of images to be used in train data
train_images_file = 'train_ids.txt'
# Read the train image names in a set
train_images = set(open(train_images_file, 'r').read().strip().split('\n'))

# Create a list of all the training images with their full path names
train_img = []
for i in img:
    if i[len(images):] in train_images:
        train_img.append(i)

# Below file contains the names of images to be used in test data
test_images_file = 'test_ids.txt'
# Read the validation image names in a set# Read the test image names in a set
test_images = set(open(test_images_file, 'r').read().strip().split('\n'))

# Create a list of all the test images with their full path names
test_img = []
for i in img:
    if i[len(images):] in test_images:
        test_img.append(i)

# load clean descriptions into memory
def load_clean_descriptions(filename, dataset):
    doc = load_doc(filename)
    descriptions = dict()
    for line in doc.split('\n'):
        tokens = line.split()
        image_id, image_desc = tokens[0], tokens[1:]
        if image_id in dataset:
            if image_id not in descriptions:

```

```

        descriptions[image_id] = list()
        # wrap description in tokens
        desc = 'startseq ' + ' '.join(image_desc) + ' endseq'
        descriptions[image_id].append(desc)
    return descriptions

# descriptions
train_descriptions = load_clean_descriptions('descriptions.txt', train)
print('Descriptions: train=%d' % len(train_descriptions))

train_features = load(open("train_features8k_vgg16.pkl", "rb"))
print('Photos: train=%d' % len(train_features))

# Create a list of all the training captions
all_train_captions = []
for key, val in train_descriptions.items():
    for cap in val:
        all_train_captions.append(cap)
print('Length of all train captions :', len(all_train_captions))

# Consider only words which occur at least 10 times in the corpus
word_count_threshold = 10
word_counts = {}
nsents = 0
for sent in all_train_captions:
    nsents += 1
    for w in sent.split(' '):
        word_counts[w] = word_counts.get(w, 0) + 1

vocab = [w for w in word_counts if word_counts[w] >= word_count_threshold]
print('preprocessed words %d -> %d' % (len(word_counts), len(vocab)))

ixtoword = {}
wordtoix = {}

ix = 1
for w in vocab:
    wordtoix[w] = ix
    ixtoword[ix] = w
    ix += 1

vocab_size = len(ixtoword) + 1 # one for appended 0's
print('Vocab size :', vocab_size)

# convert a dictionary of clean descriptions to a list of descriptions
def to_lines(descriptions):
    all_desc = list()
    for key in descriptions.keys():
        [all_desc.append(d) for d in descriptions[key]]
    return all_desc

# calculate the length of the description with the most words
def max_length(descriptions):
    lines = to_lines(descriptions)
    return max(len(d.split()) for d in lines)

# determine the maximum sequence length

```

```

max_len = max_length(train_descriptions)
print('Description Length: %d' % max_len)

# -----
# -----
# -----

# data generator, intended to be used in a call to model.fit_generator()
def data_generator(descriptions, photos, wordtoix, max_len, num_photos_per_batch):
    X1, X2, y = list(), list(), list()
    n=0
    # loop for ever over images
    while 1:
        for key, desc_list in descriptions.items():
            n+=1
            # retrieve the photo feature
            photo = photos[key][0]
            for desc in desc_list:
                # encode the sequence
                seq = [wordtoix[word] for word in desc.split(' ') if word in
wordtoix]

                # split one sequence into multiple X, y pairs
                for i in range(1, len(seq)):
                    # split into input and output pair
                    in_seq, out_seq = seq[:i], seq[i]
                    # pad input sequence
                    in_seq = pad_sequences([in_seq], maxlen=max_len)[0]
                    # encode output sequence
                    out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
                    # store
                    X1.append(photo)
                    X2.append(in_seq)
                    y.append(out_seq)
                # yield the batch data
            if n==num_photos_per_batch:
                yield ([array(X1), array(X2)], array(y))
                X1, X2, y = list(), list(), list()
                n=0

# Load Glove vectors
embeddings_index = {} # empty dictionary
f = open('glove.6B.200d.txt', encoding="utf-8")

for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs

f.close()
print('Found %s word vectors.' % len(embeddings_index))

embedding_dim = 200

# Get 200-dim dense vector for each of the 1725 words in our vocabulary

```

```

embedding_matrix = np.zeros((vocab_size, embedding_dim))

for word, i in wordtoix.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector

print('Embedding matrix shape :', embedding_matrix.shape)

inputs1 = Input(shape=(4096,))
fe1 = Dropout(0.5)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)
inputs2 = Input(shape=(max_len,))
se1 = Embedding(vocab_size, embedding_dim, mask_zero=True)(inputs2)
se2 = Dropout(0.5)(se1)
se3 = LSTM(256)(se2)
decoder1 = add([fe2, se3])
decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(vocab_size, activation='softmax')(decoder2)
model = Model(inputs=[inputs1, inputs2], outputs=outputs)

print(model.summary())

model.layers[2].set_weights([embedding_matrix])
model.layers[2].trainable = False

print(model.summary())

model.compile(loss='categorical_crossentropy', optimizer='adam')

epochs = 15
photos_per_batch = 3 # Batch size
steps = len(train_descriptions)//photos_per_batch

for i in range(epochs):
    generator = data_generator(train_descriptions, train_features, wordtoix,
max_len, photos_per_batch)
    print('Epoch', i+1, ':')
    model.fit_generator(generator, epochs=1, steps_per_epoch=steps, verbose=1)
    model.save('model_' + str(i) + '.h5')

def greedySearch(model, photo):
    in_text = 'startseq'
    for i in range(max_len):
        sequence = [wordtoix[w] for w in in_text.split() if w in wordtoix]
        sequence = pad_sequences([sequence], maxlen=max_len)
        yhat = model.predict([photo, sequence], verbose=0)
        yhat = np.argmax(yhat)
        word = ixtoword[yhat]
        in_text += ' ' + word
        if word == 'endseq':
            break
    final = in_text.split()
    final = final[1:-1]
    final = ' '.join(final)
    return final

```

```

filename = 'test_ids.txt'
test = load_set(filename)
print('Test Dataset: %d' % len(test))

test_features = load(open("test_features8k_vgg16.pkl", "rb"))
print('Photos: test=%d' % len(test_features))

test_descriptions = load_clean_descriptions('descriptions.txt', test)
print('Descriptions: test=%d' % len(test_descriptions))

# print(test_descriptions)

actual, predicted = [], []
model = load_model('model_14.h5')
for i in range(len(test_images)):
    cap = greedySearch(model, test_features[list(test_images)[i][:4]])
    # Appending actual descriptions
    one_img = []
    for c in test_descriptions[list(test_images)[i][:4]]:
        one_img.append(c.split())
    actual.append(one_img)
    print(cap)
    predicted.append(cap.split())

print('BLEU-1: %f' % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
print('BLEU-2: %f' % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0)))
print('BLEU-3: %f' % corpus_bleu(actual, predicted, weights=(0.33, 0.33, 0.33, 0)))
print('BLEU-4: %f' % corpus_bleu(actual, predicted, weights=(0.25, 0.25, 0.25,
0.25)))

# extract features from new photo
def extract_features(filename):
    # load the model
    model = VGG16()
    # re-structure the model
    model = Model(inputs=model.inputs, outputs=model.layers[-2].output)
    # load the photo
    image = load_img(filename, target_size=(224, 224))
    # convert the image pixels to a numpy array
    image = img_to_array(image)
    # reshape data for the model
    image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
    # prepare the image for the VGG model
    image = preprocess_input(image)
    # get features
    features = model.predict(image, verbose=0)
    return features

# photo = extract_features('#####') # link to new image
# desc = greedySearch(model, photo)
# print('Generated caption : ', desc)

```