EEE543 Final Project Report

Doruk Çakmakçı 21502293
Furkan Özden 21502397
Mert Albaba 21501950
Tahir Ahmet Gölge 21501627
Elif Engin 21302193

Bilkent University — March 7, 2020

Abstract

Image captioning is a popular research area in recent years. In this project, an image captioning model is presented as the term project of EEE543 course at Bilkent University. Our architecture, which is inspired by state-of-art methods in the literature, combines deep, pre-trained convolutional neural networks and long short term memory networks in order to generate captions of given figures. As convolutional networks, InceptionV3 and VGG-16 are used to encode images. For both, an LSTM network is utilized to produce captions from given image embeddings. Proposed model is accurate in most of the images. In order to show the performance of the model, test results are presented for different images for our best working model with InceptionV3.

1 Introduction

Human may easily understand and describe the scene, or images, with a quick glance [1]. To make artificially intelligent systems have the same visual capability, there is a huge amount of research and interest in this area. Previously, image classification was the leading research area in this field. In order to find features and classify images correctly, several architectures have been proposed as a result of the ImageNet challenge. Stacked convolutional layers along with pooling layers are proposed as an architecture in [2] and became the baseline model in this area. In [3], rectified linear units are introduced to a convolutional network. Much deeper networks are proposed and utilized in [4]. In [5], a deep network that stacks blocks containing convolutional layers instead of stacking layers is proposed and achieved a great success.

In recent years, the main field of research in image understanding area is generation of descriptions of images. Primitive works, [6] and [7], in this area utilized hard coded mechanism to generate descriptions of images. On the other hand, most of the recently proposed architectures utilizes convolutional networks or blocks for image feature extraction, which have been developed for classification of images. In [8], an architecture that contains two sub-architectures for different purposes are proposed. The main goal of the first of these sub-architectures is detecting sets of words found in images and this architecture has utilized deep convolutional networks. Second sub-architecture utilizes a maximum entropy language model for generating sentences. Object recognizing deep convolutional networks are used as feature extractors in [9], which are explained before. For sentence generation and as a language model, a recurrent neural network is utilized in this work. In [10], previous two works are compared and advenced with the usage of BLEU scores on a large dataset. In [11], deep convolutional neural networks are combined with recurrent neural networks for generating image desricptions, i.e. captions. In this work, a deep convolutional neural network is used as an image encoder, which is pretrained for ImageNet challenge, and outputs of this deep network is used as inputs to caption generating architecture. To generate captions, a long short term memory networks are utilized. This architecture is simply presented in Figure 1 [11].

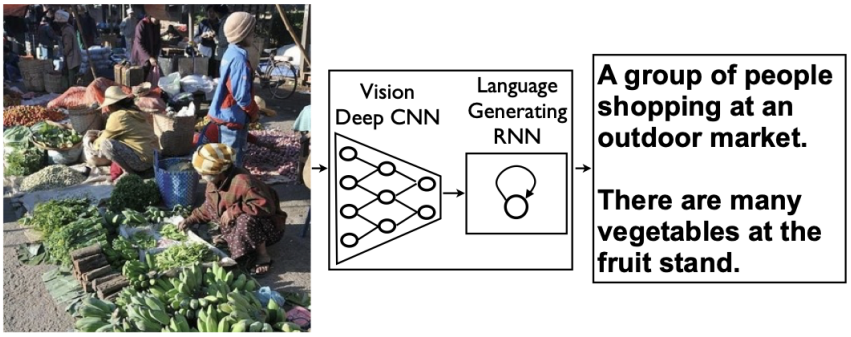


Figure 1: Simplified architecture used in [11].

In this project, an image captioning, i.e. description, program is developed. Dataset used in this work contains images and is provided by our professor and explained in Section 2. As explained before, [11] proposes an end-to-end system and achieved great BLEU scores, for instance 59 in Pascal dataset, thus in this project, we have used a very similar approach to [11]. For generating captions, an LSTM network is utilized. For image encoder convolutional neural network, we have tried two different pretrained networks: VGG16 [4] and Inception V3 [5]. Utilized architecture is detailly explained in Section 3 and for both of these, results are presented in Section 4. Codes for both architecture can be found in Section 5.

2 Methods

2.1 Dataset Description

Dataset used for this project contains real life images portraying interactions between different objects in an environment or interactions between an object and an environment. The objects of interest can vary from a human to a teddy bear. The environment can vary from a natural scene to indoors of an household. The wide array of object-environment combinations present in the dataset preserve an important situational information. Our goal in this project is to generate a model trained on the dataset which can describe the scene in an image

The dataset contains 82783 URLs from an FTP server with images and a total of 400135 captions for the images. Images may have more than one caption. Only 73,659 of the URLS given were valid. Hence, the number of images present in the dataset decreased to 73,659 and the number of captions decreased correspondingly. Captions are combination of words in a vocabulary of size 1004. Maximum length of a caption in the dataset is 17.

Glove word embedding is used to project vocabulary from word space to a vector space. Glove is an unsupervised learning algorithm for obtaining vector representation of words [12]. We used 200-dimensional vector representations of a vocabulary of size 400K, pretrained on Wikipedia. We used the subset of vector representations corresponding to our vocabulary. Nine words in our vocabulary were not present in the pretrained vectors. We set the word embeddings of missing words to 200-dimensional zero vectors. Word embedding is discussed in detail in the section InceptionV3.

After generating word embeddings of the vocabulary, we used t-SNE to visualize vocabulary space. t-SNE reduces the dimensionality of the word embedding space for visualization. In the leftmost figure below, we can see that words related to color are clustered. In the middle figure below, we can see that words related to cooking and food ingredients are clustered. In the rightmost figure below, we can see the missing words, which are set to zero vectors, are clustered. This visualization is a validation of the projection of semantic space to a space where distance between points represent semantic closeness.

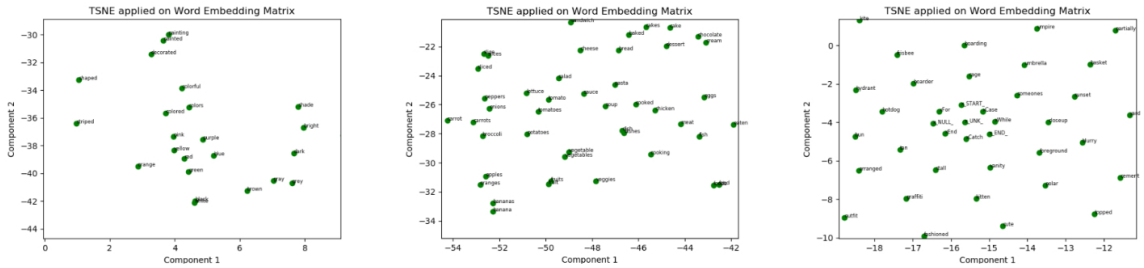


Figure 2: t-SNE visualizations of word embedding space for example clusters

2.2 Training Methodology

It is very well known that CNN's can encode images to a subspace that can efficiently encode the given image with its features. This is crucial since the images are embedded by the CNN's to a fixed length vector, that can later be used for visual tasks such as image captioning. Thus, it is very common to use a pre-trained CNN classifier with its final softmax layer popped out in order to encode the images efficiently for image captioning task. Our approach to the problem is as follows. Since the task includes both a computer vision and a natural language processing part, we combined two state-of-the-art models with their pre-trained weights for encoding purposes. After encoding processes are done, dropout layers are used to prevent overfitting. The encoded features are then decoded with dense layers with learnable parameters. In the final layer, our model chooses the next word in the caption sequence with a softmax activated dense layer with proper dimensions.

We chose Inception V3 [5] as an image feature extractor for our architecture. For encoding of the caption sequences, we used GLOVE pre-trained word embedding matrix. With these extracted features as inputs to our model architecture, our model design is summarized in the figures below.

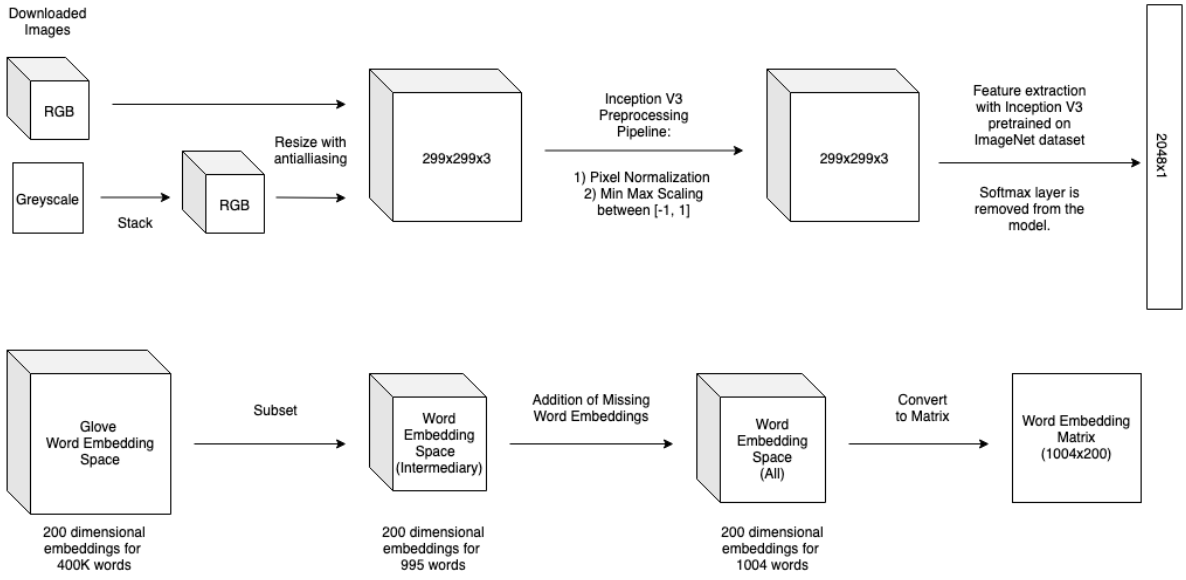


Figure 3: Data pre-processing with pre-trained models

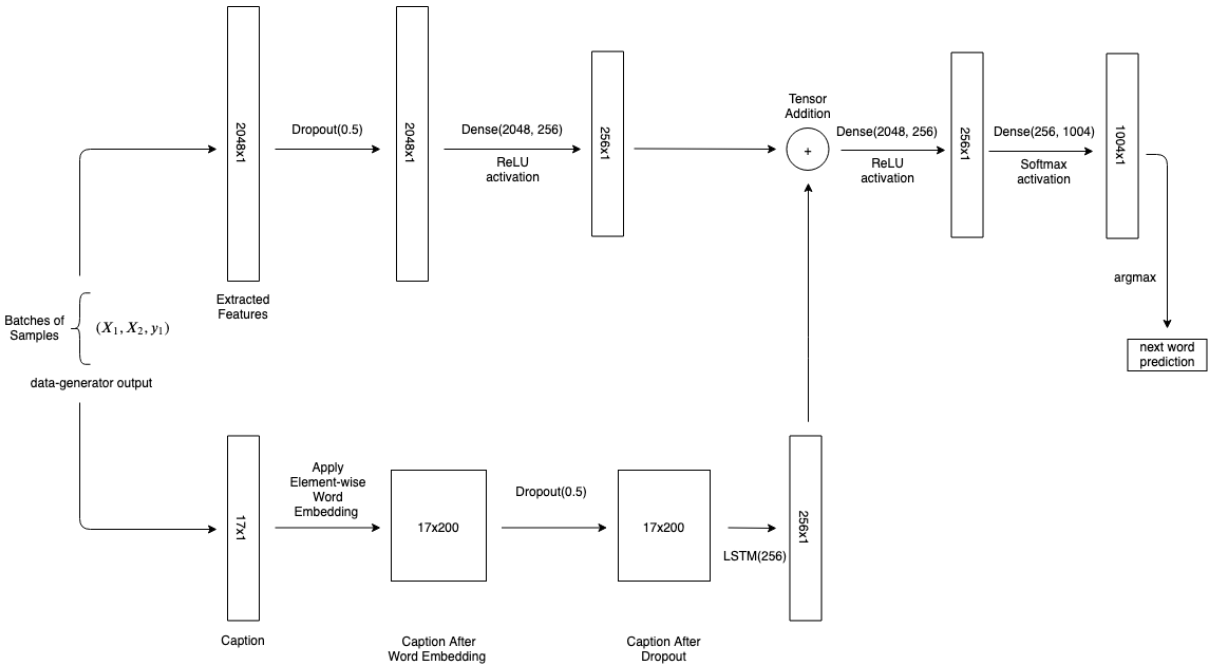


Figure 4: Model architecture

Although it can be seen from the figures above, we also discuss the model training pipeline in detail with the following items.

- Process all images in the dataset such that each of them is of shape $299 \times 299 \times 3$ (i.e, the input shape of InceptionV3 pre-trained model). If any image is gray-scale, we add another dummy axis to convert it to 3-channel rgb image. The images are pre-processed with reshaping using keras pre-process input method.
- After pre-processing step, we normalize each image between $[-1, 1]$ range for better feature extraction.
- All images are then encoded using InceptionV3 with final shape of 2048×1. The encodings of images are stored separately.
- Our captions are in the form of integer sequences as it is explained in the Dataset Description section. First off, using GLOVE, we extracted pre-trained word embeddings for each word in the vocabulary of the given dataset. These word embeddings are also stored seperately. We used GLOVE with embedding size 200. Some of the words in the vocabulary were not present in the GLOVE vocabulary. These words are:
  - x_START_
  - x_END_
  - x_UNK_
  - x_NULL_
  - xFor
  - xWhile
  - xCase
  - xCatch
  - xEnd

  The first four words that are not present in the GLOVE vocabulary are understandable since they are tokens for start of sequence, end of sequence, out of vocabulary word (i.e, o.o.v) and sequence padding encodings respectively. However all of the other words that are not present in the GLOVE dictionary are not of big importance in terms of captioning a given image. All of the o.o.v images are encoded with all zero vectors of size 200 (i.e, embedding size).

- With the above feature extractions are performed and saved, we feed them to our data-generator function.
- The data-generator function takes an image and a captioning sequence, it generates 3-tuples in the form $(X1, X2, Y1)$, where $X1$ is always the input image encoding. $X2$ is a fixed length integer encoding of a caption sequence. For an input image features, $X1$, assume that its caption sentence has n words. The generator function generates sequences with first i words from the caption sentence of $X1$ where $i \in [0, n]$ with $n < 17$ where 17 is the maximum length of captions in the dataset. The generated sequences are then padded with 0's (i.e, integer encoding of x_NULL_) for fixed length mapping. For each of the generated $X2$ sequence with first i words are taken from the original caption sequence, Y is assigned as the target variable with the next word after the last word in the $X2$ sequence, that is $(i + 1)$'st word. Hence, the data-generator function produces n training examples from an image with caption length n.
- The training samples generated by the data-generator function are then fed to our model. Hence, our model has two inputs and a single output word prediction.
- The encoded image features, $X1$'s, are directly fed to a Dropout layer with drop rate of 0.5 to prevent over-fitting. The drop rate choice of 0.5 is result of trial-error search. Different drop rates are tried.
- After the Dropout layer, we feed the outputs to a Dense layer with 256 neurons, each of which has ReLU activation, for further caption-specific feature extraction.
- The integer caption encodings, $X2$'s, are directly fed to the pre-trained and weight-frozen Embedding layer to generate corresponding embedding matrix of the sequence.
- Obtained embeddings are fed to a Dropout layer with drop rate of 0.5 to prevent over-fitting. The drop rate choice is again result of a trial-error search.
- After the Dropout layer, obtained outputs are fed to a LSTM layer with dimensionality of the output space 256, to match with the output dimensionality of the feature extractions of the image input.
- The 256 dimensional sequence features obtained from LSTM layer are than added element-wise to the 256 dimensional image features obtained from the dense layer. Let's call the result of this addition as total features.
- Total features are fed to a dense layer with 256 neurons each of which has ReLU activation again, to learn the mapping between prediction layer and total features of the training sample.
- Finally, the output is fed to a dense layer with 1004 neurons with softmax activation in order to make prediction of the next word in the captioning sequence, namely Y.
- Our architecture combines categorical cross-entropy loss with the softmax outputs since the problem is now expressed as a supervised classification problem. The initial learning rate of the model is 0.001. We used Adam optimizer [13] for optimization. We used a batch size of 6 images initially. Notice that, batch size of 6 images does not mean 6 training examples. Since each image has multiple captions with each length n caption having n training samples.
- Also we realized that model gets into a plateau after 4 epochs where the loss does not significantly decrease over 1 full epoch. Thus, we manually set learning rate of the model to 0.0001 after 4 epochs and increased the batch size to 12 for better loss function approximation. After 10 epochs, we set learning rate to 0.00001 and increased the batch size to 18 with a similar reasoning.
- For validation and test measures, we splitted the dataset with $70\%, 20\%, 10\%$ ratios for training, validation and testing respectively. We used image count while performing the split.

2.3 Training Hardware, Software Specifications & Training Times

The described model is implemented in Python language, with Keras Deep Learning framework. We have used Pillow library in order to process, save, load images and for BLEU score metric calculations we have used NLTK library. The model is then trained on a above average laptop computer. The specifications of the computer is listed below:

- Model : Dell XPS9570
- Processor : Intel Core i7 8th Gen. 8750H 2.2GHz
- GPU : NVIDIA Geforce GTX1050Ti
- RAM : 8GB

We have used GPU support for training, with the GPU specified above. We trained the model with VGG-16 as feature extractor on 5 epochs with a total training time of ~ 1 hour. We chose the model with InceptionV3 as our best model and we trained it 32 epochs with a total training time of ~ 8 hours. The increase in batch size has a positive effect on training time, the batch size specifications are discussed in the above subsection.

3 Results

In this section we report following performance measures regarding our models:

- Learning curves (i.e, validation and loss graphs)
- Sample test caption predictions from our model
- BLEU score [14]

As we have already mentioned in the Introduction section, we have tried two different major architectures to the problem. The two approaches are listed below:

- CNN + LSTM architecture with CNN transfer learning using VGG-16
- CNN + LSTM architecture with CNN transfer learning using InceptionV3

Both of the above approaches had identical inner layers with different input shapes to the models. Since VGG-16 extracts features with shape 4096×1 and InceptionV3 extracts features with shape 2048×1. In subsection Results VGG-16, we share the results regarding the model with VGG-16 as a feature extractor and in subsection Results InceptionV3, we share the results of the model with InceptionV3 as a feature extractor. Note that, in each of the models GLOVE word embedding is used.

3.1 Results InceptionV3

The training procedure of the model is discussed in detail in the Methods section. We chose this model as best model from our model inventory. Hence, we examined this models output in great detail.

3.1.1 Learning curve and BLEU score

Figure below shows the learning curves of the model. Since the dataset is quite big with around 72000 images, batch-wise loss decrease is a more smooth decrease. However we present here average of batch losses over 1 full epoch. The graph below illustrates the validation and training losses separately.

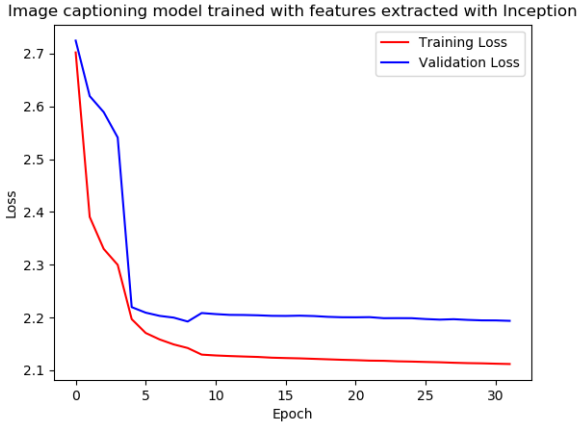


Figure 5: Learning curve of model with InceptionV3

As we have mentioned in the Methods section, we perform learning rate manipulations at epochs 4 and 10. The effects of these manipulations can be seen from the loss graphs. We also calculated BLEU score of our model using 10% of the given dataset. The BLEU score is calculated as 28.83.

3.1.2 Selected caption predictions from test set

We present some selected caption predictions from test set with their real captions in the below grid figures. We can see that model learned to generalize caption generation task very well in overall. It can recognize variety of actions along with environmental effects. It uses x_UNK_ token very appropriately when generating a caption.

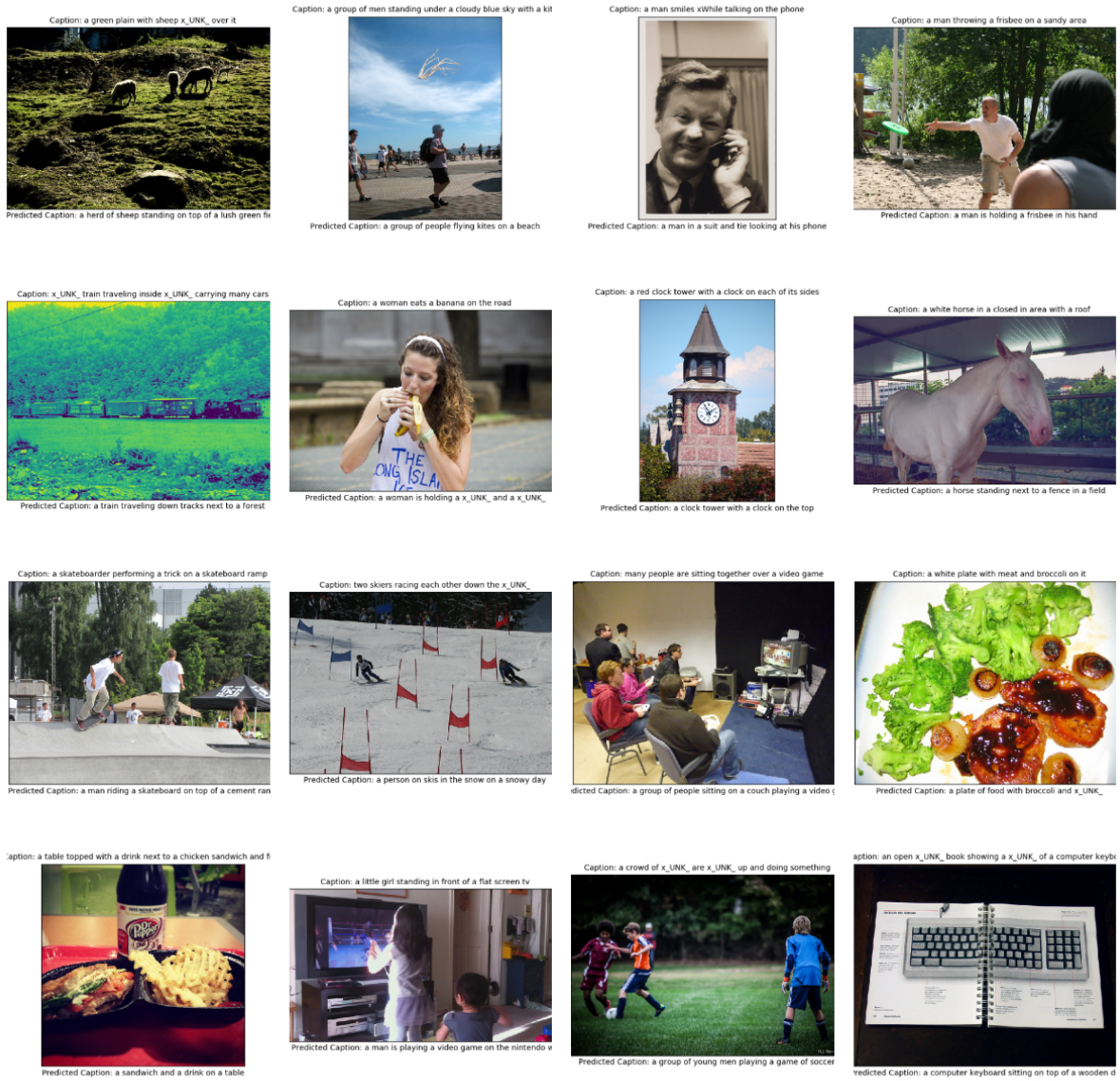
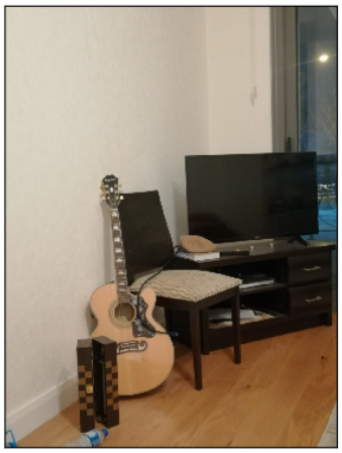


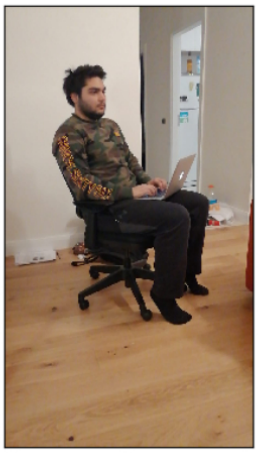
Figure 6: Some selected caption predictions of our model, for display purposes we took the first label caption of each image (i.e, images have multiple label captions we took first one)

3.1.3 Real life caption predictions (images are taken by our team members)

In this subsection, we wanted to test our model with pictures that are taken by our team members. Below pictures are taken using our cell-phone's camera, and tested with the final weights of our model. The predictions somewhat resembled to the real scenes in the photos.



Predicted Caption: a living room with a television and a television



Predicted Caption: a man sitting on a couch with a laptop

Predicted Caption: a car is parked in front of a x_UNK_

Predicted Caption: a man is sitting in front of a tv

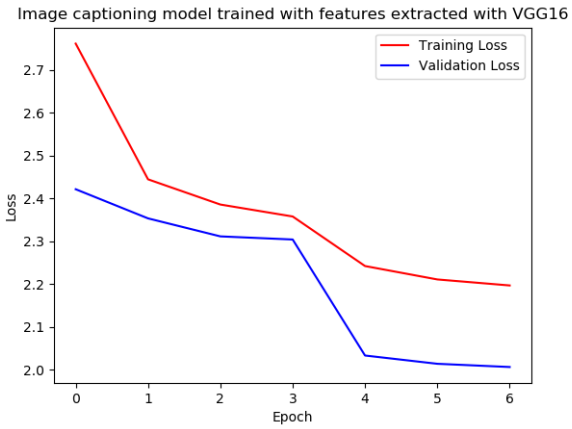
Figure 7: Caption predictions of photos taken by our team

3.2 Results VGG-16

In this sub-section we briefly summarize results obtained from the model uses VGG-16 as feature extractor.

3.2.1 Learning curve and BLEU score

The learning curve regarding the model with VGG-16 is given in the below figure. We calculated BLEU score in the test set also with this model. The BLEU score is calculated as 26. Because of faster training, fewer model parameters we chose to train the model with InceptionV3 as our main model. We trained VGG-16 only for 6 epochs. This model has a lot more learnable parameters when compared to the model with InceptionV3 as a feature extractor because VGG-16 gives features of images with 4096 dimensional vectors which drastically increases parameters required in the architecture.



Learning curve of model with VGG-16

4 Discussion

In this work, an image captioning algorithm, which is a combination of deep pre-trained convolutional neural network and long short term memory network, is proposed and implemented. As explained before, there is a huge amount of interest to this area, generation of image descriptions. Our model has been inspired by one of the state-of-art methods in the literature, [11], which also combines an LSTM network with a deep CNN, Inception V3. In [11], it is stated that the proposed model, which is very similar to our model, has achieved 63 and 66 as BLEU scores in Flickr 8k and Flickr 30k datasets, respectively. In this work, BLEU scores are also calculated as 26 and 28.83, for both of the proposed models, model with VGG16 and with Inception V3, respectively. In order to understand the reason of the difference between BLEU scores of [11] and our Inception V3 model, we have analyzed our dataset first and compared it with Flickr 8k dataset. According to our analysis, number and variety of figures in our dataset are enough, however, some captions are very short. As the result of the comparison with Flickr 8k dataset, it is deduced that although the dataset used in this work has more samples than the Flickr 8k dataset, captions are very limited. In other words, vocabulary of the dataset used is very limited when compared with the Flickr 8k dataset. To clarify, our dataset has a vocabulary formed by 1004 words and Flickr 8k dataset's vocabulary contains more than 4000 words. Furthermore, captions in our dataset are generally shorter than the ones in Flickr 8k dataset. To exemplify, the maximum caption length in Flickr 8k dataset was 34 and it is 17 in our dataset. Thus, we believe that the main reason of the difference in BLEU scores is captions in our dataset. In order to understand if Inception V3 fails in our dataset, VGG 16 is also utilized, however, produced worse results in terms of BLEU scores. Also along the training process of model with InceptionV3, we realized that the model was able to generate object names but without a proper natural language. As a solution, we added Dropout layers after the image feature extraction part, we increased the drop rate iteratively by trial-error search in order to achieve most robust results.

In order to reach a higher BLEU scores and produce better captions, image descriptions, i.e. captions, provided in our dataset may be enriched. Furthermore, this dataset may be combined with another dataset containing large vocabulary to generate better captions. Lastly, attention mechanism may also be introduced to our caption generating network in order to achieve better results.

All in all, the main purpose stated in the introduction in this work is generating meaningful captions for the images in the provided dataset. As presented results validates, we have reached the goals we have set and generated meaningful captions for the most of the images in the dataset. Thus, our proposed model generated the expected outputs, which are meaningful and accurate captions.

5 Appendix

Please note that, for easy running of all the codeblocks, we uploaded a .zip file containing a directory structure. We also included a README file which explains all of the code design structure of the project. One can run the main training file discriptor.py directly from the directory that was downloaded. All result

generating codes are included in the .zip file, explained in the README file and also given in the below listings.

5.1 Codes used for both models, result generating & dictionary preparing codes

```
import pickle
import os
from nltk.translate.bleu_score import corpus_bleu

# test dataset (10% of the data)
with open("./data/dataset/test_dataset_inception.pkl", "rb") as f:
    test_features = pickle.load(f)
    test_encoded_images = preprocess_dataset(test_features)
with open("./data/dataset/test_captions_inception.pkl", "rb") as f:
    test_captions = pickle.load(f)
    test_dataset = test_captions

# relevant dictionaries for word conversion
with open("./utils/word_to_idx.pkl", "rb") as f:
    word_to_idx = pickle.load(f)
with open("./utils/idx_to_word.pkl", "rb") as f:
    idx_to_word = pickle.load(f)

# other
inception = InceptionV3(weights='imagenet', include_top=True)
inception = Model(inception.input, inception.layers[-2].output)
model = load_model('./model_weights/model_inception_epoch1.h5')
max_length = 17
corpus_bleu_score = 0
references = []
candidates= []
for key in test_captions.keys():
    candidate = ['x_START_']
    reference = test_captions[key]
    for i in range(max_length):
        sequence = [word_to_idx[w] for w in candidate if w in word_to_idx.keys()]
        sequence = pad_sequences([sequence], maxlen=max_length)
        yhat = model.predict([img,sequence], verbose=0)
        yhat = np.argmax(yhat)
        word = idx_to_word[str(yhat)]
        candidate.append(word)
        if word == 'x_END_':
            while len(candidate) != 17:
                candidate.append('x_NULL_')
            break
    candidates.append(candidate)
    references.append(reference)

bleu = corpus_bleu(references, candidates)
print("Bleu Score: ", bleu)
```

```
import pickle
import h5py
import os
import pdb

import numpy as np

from gensim.models import Word2Vec

# input is a list of integers or words, and corresponding lookup table
def convert_caption(caption, dct):
    result = []
    for item in caption:
        result.append(dct[str(item)])
    return result

# import dataset
with h5py.File('./data/eee443_project_dataset_train.h5', 'r') as f:
    captions = f['train_cap'][:]
```

```
    word_codes = f['word_code'][:]
    image_ids = f['train_imid'][:]

vocabulary_size = len(word_codes[0].dtype)

# create dictionaries for conversion between words and corresponding indices
idx_to_word = {}
word_to_idx = {}
for i in range(vocabulary_size):
    idx_to_word[str(int(word_codes[0][i]))] = word_codes[0].dtype.descr[i][0]
    word_to_idx[word_codes[0].dtype.descr[i][0]] = str(int(word_codes[0][i]))

pdb.set_trace()

# save all dictionaries
with open('./utils/word_to_idx.pkl', 'wb') as f:
    pickle.dump(word_to_idx, f)
with open('./utils/idx_to_word.pkl', 'wb') as f:
    pickle.dump(idx_to_word, f)
```

```
import h5py
import os
import pdb
import urllib.request

import numpy as np

from multiprocessing.pool import ThreadPool



with h5py.File('./data/eee443_project_dataset_train.h5', 'r') as f:
    captions = f['train_cap'][:]
    image_ids = f['train_imid'][:]
    images = f['train_ims'][:]
    urls = f['train_url'][:]
    word_codes = f['word_code'][:]

print("Dataset Fields")
print("Captions: ", captions.shape)
print("Image IDs: ", image_ids.shape)
print("Images: ", images.shape)
print("URLs: ", urls.shape)
print("Word Codes: ", word_codes.shape)
print("Min image id:", np.min(image_ids))
print("Max image id:", np.max(image_ids))

dataset_path = "./data/dataset/"
images_path = os.path.join(dataset_path, "images/")

# first create a directory named "dataset/images/" to be used as download destination
# uncomment to download images
# error_count = 0
urls = list(enumerate(urls))
def download(idx, url):
    try:
        urllib.request.urlretrieve(url.decode('UTF-8'), os.path.join('./data/dataset/
    images/', str(idx + 1) + ".jpg"))
    except urllib.request.HTTPError:
        pass
with ThreadPool(20) as p:
    p.starmap(download, urls)


# for idx, url in enumerate(urls):
#     try:
#         urllib.request.urlretrieve(url.decode('UTF-8'), os.path.join(images_path, str(
    idx) + ".jpg"))
#     except urllib.request.HTTPError as e:
#         error_count += 1
#         print("Error occured at idx: ", idx, " with source: ", url)
#         print("Error Status Code: ", e.code)
```

```
#         print("Error Message: ", e.read())

# print(error_count, "images were not downloaded due to errors.")
```

```
import pdb
import h5py
import pickle

import numpy as np

with h5py.File('./data/eee443_project_dataset_train.h5', 'r') as f:
    captions = f['train_cap'][:]
    image_ids = f['train_imid'][:]
    images = f['train_ims'][:]
    urls = f['train_url'][:]
    word_codes = f['word_code'][:]

glove_embedding_dict = {}
with open("./utils/glove.6B.200d.txt", 'r', encoding="utf-8") as f:
    for line in f:
        values = line.split()
        word = values[0]
        vector = np.asarray(values[1:], "float32")
        glove_embedding_dict[word] = vector

# check if all vocabulary is present on glove embeddings
with open("./utils/word_to_idx.pkl", "rb") as f:
    word_to_idx = pickle.load(f)
    vocabulary = list(word_to_idx.keys())

embedding_dict = {}
error_cnt = 0
successful_cnt = 0
for word in vocabulary:
    try:
        # print(word, "\t", glove_embedding_dict[word].shape)
        embedding_dict[word] = glove_embedding_dict[word]
        successful_cnt += 1
    except:
        print("glove embedding not found for: ", word)
        print("replaced with zeros")
        embedding_dict[word] = np.zeros(200)
        error_cnt += 1


print("Error count: ", error_cnt)
print("Successful count: ", successful_cnt)


embedding_matrix_size = len(list(embedding_dict.keys()))
embedding_matrix = np.zeros((embedding_matrix_size, 200))
for word in embedding_dict.keys():
    embedding_matrix[int(word_to_idx[word]), :] = embedding_dict[word]

# set embedding of unknown tag as average
# embedding_matrix[int(word_to_idx["x_UNK_"]), :] = np.mean(np.array(list(embedding_dict.
    values())), axis=0)

np.save("./utils/embedding_matrix.npy", embedding_matrix)
```

```

import pickle
from keras.preprocessing.sequence import pad_sequences
import numpy as np
import pdb
import random
from keras.applications.inception_v3 import InceptionV3, preprocess_input
import sys
from PIL import Image
from keras.models import load_model
from keras.models import Model
import os
```

```
import matplotlib.pyplot as plt

# import conversion dictionaries
with open("./utils/word_to_idx.pkl", "rb") as f:
    word_to_idx = pickle.load(f)
with open("./utils/idx_to_word.pkl", "rb") as f:
    idx_to_word = pickle.load(f)

    # all dataset
with open("./data/dataset/processed_dataset_inception.pkl", "rb") as f:
    features = pickle.load(f)
with open("./utils/captions.pkl", "rb") as f:
    captions = pickle.load(f)

count = 0
for key in captions.keys():
    for capt in captions[key]:
        count+=1
pdb.set_trace()
inception = InceptionV3(weights='imagenet', include_top=True)
inception = Model(inception.input, inception.layers[-2].output)

model = load_model('./model_weights/model_inception_epoch1.h5')
max_length = 17

def get_image(image_id):
    feature = features[str(image_id)]
    caps = captions[str(image_id)]
    cap = caps[0] # use first caption
    return feature, cap

# generate caption for a given sample
def describe_image(img, model):
    input_string = 'x_START_'
    for i in range(max_length):
        sequence = [word_to_idx[w] for w in input_string.split() if w in word_to_idx.keys
    ()]
        sequence = pad_sequences([sequence], maxlen=max_length)
        yhat = model.predict([img,sequence], verbose=0)
        yhat = np.argmax(yhat)
        word = idx_to_word[str(yhat)]
        input_string += ' ' + word

        if word == 'x_END_':
            break

    final = input_string.split()
    final = final[1:-1]
    final = ' '.join(final)
    return final

def get_image_caption(image_id):
    features, caption = get_image(image_id)
    result = [idx_to_word[str(x)] for x in caption if idx_to_word[str(x)] != "x_NULL_"]
    result = result[1:-1]
    result = ' '.join(result)
    return result

images = ["car_1", "doruk_1", "ev_1", "furkan_laptop"]

for id in images:
    path= os.path.join('./data/real_life_images/'+id+".jpeg")
    img_ = Image.open(path)
    processed_img = img_.resize((299,299), resample=Image.ANTIALIAS)
    temp = np.array(processed_img)
    if temp.shape == (299,299):
        temp_ = [temp, temp, temp]
        temp = np.stack(temp_, axis=2)
    img= np.array(temp, dtype=np.uint8)
    img = preprocess_input(img)
    img = np.expand_dims(img,axis=0)
    res = inception.predict(img)
```

```
    res = np.reshape(res, res.shape[1])
    img_features = np.expand_dims(res, axis=0)

    pred_cap = describe_image(img_features, model)
    print(pred_cap)
    fig, ax = plt.subplots()
    ax.get_xaxis().set_ticks([])
    ax.get_yaxis().set_ticks([])
    ax.imshow(img_)
    ax.axis('on')
    ax.set_xlabel("Predicted Caption: " + pred_cap, fontsize=10)
    plt.savefig(id+"_capt")
    plt.close()
```

```

import pickle
from keras.preprocessing.sequence import pad_sequences
import numpy as np
import pdb
import random
from keras.applications.inception_v3 import InceptionV3, preprocess_input
import sys
from PIL import Image
from keras.models import load_model
from tensorflow.keras.models import Model

# import conversion dictionaries
with open("./utils/word_to_idx.pkl", "rb") as f:
    word_to_idx = pickle.load(f)
with open("./utils/idx_to_word.pkl", "rb") as f:
    idx_to_word = pickle.load(f)

    # all dataset
with open("./data/dataset/processed_dataset_vgg.pkl", "rb") as f:
    features = pickle.load(f)
with open("./utils/captions.pkl", "rb") as f:
    captions = pickle.load(f)

model = load_model('./model_weights/model_inception_epoch1.h5')
max_length = 17

def get_image(image_id):
    feature = features[str(image_id)]
    caps = captions[str(image_id)]
    cap = caps[0] # use first caption
    return feature, cap

# generate caption for a given sample
def describe_image(img, model):
    input_string = 'x_START_'
    for i in range(max_length):
        sequence = [word_to_idx[w] for w in input_string.split() if w in word_to_idx.keys
    ()]
        sequence = pad_sequences([sequence], maxlen=max_length)

        yhat = model.predict([img,sequence], verbose=0)
        yhat = np.argmax(yhat)
        word = idx_to_word[str(yhat)]
        input_string += ' ' + word

        if word == 'x_END_':
            break

    final = input_string.split()
    final = final[1:-1]
    final = ' '.join(final)
    return final

def get_image_caption(image_id):
    features, caption = get_image(image_id)
    result = [idx_to_word[str(x)] for x in caption]
    result = result[1:-1]
```

```
    result = ' '.join(result)
    return result

images = [24182, 137, 30296, 10518, 25178, 16968, 21062, 30707, 27506, 18408, 39625,
    860,6086, 40825, 8690, 32552]

for id in images:
    image = Image.open(os.path.join('./data/dataset/images/'+str(id)+".jpeg"))
    img, cap = get_image(id)
    pred_cap = describe_image(features[str(id)], model)
    pdb.set_trace()
```

```
import os
import pdb
import h5py
import pickle

import numpy as np

from tqdm import tqdm

with h5py.File('./data/eee443_project_dataset_train.h5', 'r') as f:
    captions = f['train_cap'][:]
    image_ids = f['train_imid'][:]

min_id = np.min(image_ids)
max_id = np.max(image_ids)

dct = {}
for i in tqdm(range(min_id, max_id + 1)):
    idx = np.where(image_ids == i)
    dct[str(i)] = [x.tolist() for x in captions[idx]]

pdb.set_trace()

with open("./utils/captions.pkl", "wb") as f:
    pickle.dump(dct, f)
```

```
import pdb
import h5py
import pickle

import numpy as np
import matplotlib.pyplot as plt

from sklearn.manifold import TSNE
from mpl_toolkits.mplot3d import Axes3D

embedding_matrix = np.load("./utils/embedding_matrix.npy")
with open("./utils/idx_to_word.pkl", "rb") as f:
    idx_to_word = pickle.load(f)

vocabulary_size = 1004
tsne = TSNE(n_components=2, n_iter=3000, perplexity=30, learning_rate=200)
embedding = tsne.fit_transform(embedding_matrix)


figure = plt.figure()
ax = figure.add_subplot(111)
ax.set_title("TSNE applied on Word Embedding Matrix")
ax.scatter(embedding[:,0], embedding[:,1], c='g')
ax.set_xlabel("Component 1")
ax.set_ylabel("Component 2")
for idx in range(vocabulary_size):
    plt.annotate(idx_to_word[str(idx)], (embedding[idx,0], embedding[idx,1]), fontsize=6)

plt.savefig('./plots/tsne_word_embedding.png')
```

```
import os
import pdb
import h5py
import pickle
```

```

import numpy as np

from tqdm import tqdm

with h5py.File('./data/eee443_project_dataset_train.h5', 'r') as f:
    captions = f['train_cap'][:]
    image_ids = f['train_imid'][:]
    images = f['train_ims'][:]
    urls = f['train_url'][:]
    word_codes = f['word_code'][:]

with open('./utils/word_to_idx.pkl', 'rb') as f:
    word_to_idx = pickle.load(f)
with open('./utils/idx_to_word.pkl', 'rb') as f:
    idx_to_word = pickle.load(f)

word_frequency = {}
for word in list(word_to_idx.keys()):
    word_frequency[word] = 0

for caption in tqdm(captions):
    for word in caption:
        word_frequency[idx_to_word[str(word)]] += 1

low_frequency_word_count = 0
for word in word_frequency.keys():
    if word_frequency[word] <= 10:
        print(word, "\t", word_frequency[word])
        low_frequency_word_count += 1

print("Low Frequency Word Count: ", low_frequency_word_count)

words = list(word_frequency.keys())
frequencies = list(word_frequency.values())
frequencies, words = (list(t) for t in zip(*sorted(zip(frequencies, words))))
print(words[-10:])
print(frequencies[-10:])
```

5.2 Codes for Inception V3 + LSTM

```
# from tensorflow.python.framework import ops
# ops.reset_default_graph()

import numpy as np
import tensorflow as tf

from keras.layers import Conv1D
from keras.models import Sequential
from keras.applications.inception_v3 import InceptionV3
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from keras.optimizers import Adam, RMSprop
from keras.layers.merge import add
from keras.layers import Input
from keras.layers import Dropout, Dense, LSTM, Embedding
from keras.models import Model, load_model
import keras.backend as K

import pickle
import pdb

def preprocess_dataset(dct):
    result = {}
    for key, value in zip(dct['keys'], dct['values']):
        result[key] = value
    return result

# # all dataset
# with open("./utils/processed_dataset_vgg.pkl", "rb") as f:
#     all_features = pickle.load(f)
# with open("./utils/captions.pkl", "rb") as f:
```

```
#     all_captions = pickle.load(f)

# train dataset (70% of the data)
with open("./data/dataset/train_dataset_inception.pkl", "rb") as f:
    train_features = pickle.load(f)
    train_encoded_images = preprocess_dataset(train_features)
with open("./data/dataset/train_captions_inception.pkl", "rb") as f:
    train_captions = pickle.load(f)
    train_dataset = train_captions
    # train_dataset = preprocess_dataset(train_captions)

# validation dataset (20% of the data)
with open("./data/dataset/validation_dataset_inception.pkl", "rb") as f:
    validation_features = pickle.load(f)
    validation_encoded_images = preprocess_dataset(validation_features)
with open("./data/dataset/validation_captions_inception.pkl", "rb") as f:
    validation_captions = pickle.load(f)
    validation_dataset = validation_captions
    # validation_dataset = preprocess_dataset(validation_captions)

# test dataset (10% of the data)
with open("./data/dataset/test_dataset_inception.pkl", "rb") as f:
    test_features = pickle.load(f)
    test_encoded_images = preprocess_dataset(test_features)
with open("./data/dataset/test_captions_inception.pkl", "rb") as f:
    test_captions = pickle.load(f)
    test_dataset = test_captions
    # test_dataset = preprocess_dataset(test_captions)

# train_dataset = train_captions
# train_encoded_images = train_features

# test_dataset = test_captions
# test_encoded_images = test_features

# validation_dataset = validation_captions
# validation_encoded_images = validation_features


# model
max_length = 17
embedding_dim = 200
vocabulary_size = 1004

'''
Model takes 2 types of inputs, first being encoded photos by inception to 2048
    dimensional vectors. Second input type to the model are sequences of words (i.e,
    captions).
Captions are all zero padded to max_length. With first item being SOS token and last item
     being EOS token. Dropout layers are used for prevention of overfitting. Photos are
    processed
such that their dimensions are reduced to 256 by dense layer with relu activation.
    Caption sequences are processed by the glove pretrained embedding matrix. Embedding
    dimension
is chosen as 200. Vocabulary size in the problem is 1020.
'''

inputs1 = Input(shape=(2048,))
features1 = Dropout(0.5)(inputs1)
features2 = Dense(256, activation='relu')(features1)
inputs2 = Input(shape=(max_length,))
seqfeatures1 = Embedding(vocabulary_size, embedding_dim, mask_zero=True)(inputs2)
seqfeatures2 = Dropout(0.5)(seqfeatures1)
seqfeatures3 = LSTM(256)(seqfeatures2)

#add two different types of feature extractions into one tensor.
decoder1 = add([features2, seqfeatures3])

decoder2 = Dense(256, activation='relu')(decoder1)
#softmax to predict target word over vocabulary.
outputs = Dense(vocabulary_size, activation='softmax')(decoder2)
#construction of the model.
```

```
model = Model(inputs=[inputs1, inputs2], outputs=outputs)


'''
Data generator function streams the input data to the model in batches. This will be used for fit_generator function with the model.
Note that, batch size is in unit of images. With batch_size = 3, there will be 3 images with corresponding training sample generation.
'''

batch_size = 6
def data_generator(dataset, imgs, max_length, batch_size = batch_size):
    X1, X2, y = list(), list(), list()
    n = 0
    #infinite loop over images
    while True:
        for key, captions_list in dataset.items():
            n += 1
            # retrieve the photo feature
            try:
                image = imgs[key]
            except:
                n -= 1
                continue
            for capt in captions_list:
                # split one sequence into multiple X, y pairs
                for i in range(1, len(capt)):
                    # split into input and output pair
                    in_seq, out_seq = capt[:i], capt[i]
                    if out_seq == 0:
                        break
                    # pad input sequence
                    in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                    # encode output sequence
                    out_seq = to_categorical([out_seq], num_classes=vocabulary_size)[0]
                    # store
                    X1.append(image)
                    X2.append(in_seq)
                    y.append(out_seq)
            # yield the batch data
            if n==batch_size:
                yield [[np.array(X1), np.array(X2)], np.array(y)]
                X1, X2, y = list(), list(), list()
                n=0

#get embedding matrix with shape 1024x200
print(model.summary())
embedding_matrix = np.load("./utils/embedding_matrix.npy")
model.layers[2].set_weights([embedding_matrix])
model.layers[2].trainable = False

#compile the model with categorical cross entropy, combined with softmax
model.compile(loss='categorical_crossentropy', optimizer='adam')

steps = len(train_dataset)//batch_size
# generator = data_generator(train_dataset, train_encoded_images, max_length, batch_size)
# model.optimizer.lr = 0.01
# model.fit_generator(generator, epochs=9, steps_per_epoch=steps, verbose=1)
# model.save('./model_' + '9epochs' + '.h5')
epochs = 50
# # load model
base = 0
# model = load_model('./model_weights/model_9.h5')
# model.optimizer.learning_rate = 0.00001

# print(K.eval(model.optimizer.lr))
# K.set_value(model.optimizer.lr, 0.000001)
# print(K.eval(model.optimizer.lr))


for i in range(1,epochs+1):
    if i == 5:
```

```
        batch_size = 12
        K.set_value(model.optimizer.lr, 0.0001)
    elif i == 10:
        batch_size = 18
        K.set_value(model.optimizer.lr, 0.00001)
    train_generator = data_generator(train_dataset, train_encoded_images, max_length, batch_size)
    validation_generator = data_generator(validation_dataset, validation_encoded_images, max_length, batch_size)
    # pdb.set_trace()
    model.fit_generator(train_generator, epochs=1, steps_per_epoch=steps, verbose=1, validation_data=validation_generator, validation_steps=steps, validation_freq=1)
    model.save('./model_weights/model_inception_epoch' + str(base+i) + '.h5')
```

```
import os
import pdb
import pickle

import pandas as pd
import numpy as np

from keras.applications.inception_v3 import InceptionV3, preprocess_input
from keras.models import Model
from keras import backend as K

from tqdm import tqdm

inception = InceptionV3(weights='imagenet', include_top=True)
model = Model(inception.input, inception.layers[-2].output)

def extract_features(img):
    #img are 299x299x3 images of rgb
    img = preprocess_input(img)
    img = np.expand_dims(img,axis=0)
    res = model.predict(img)
    res = np.reshape(res, res.shape[1])

    return res

# import processed images
with open("./data/dataset/dataset_inception.pkl", "rb") as f:
    dct = pickle.load(f)

images = list(dct.values())
ids = list(dct.keys())

dataset = {}
for (img, id) in tqdm(zip(images, ids)):
    feature = extract_features(img)
    dataset[id] = feature

# save image features
with open("./data/dataset/processed_dataset_inception.pkl", "wb") as f:
    pickle.dump(dataset, f)
```

```
import os
import pdb
import h5py
import re
import pickle

import numpy as np

from PIL import Image
from tqdm import tqdm

with h5py.File('./data/eee443_project_dataset_train.h5', 'r') as f:
    captions = f['train_cap'][:]
    image_ids = f['train_imid'][:]
    images = f['train_ims'][:]
    urls = f['train_url'][:]
    word_codes = f['word_code'][:]
```

```
18
19 # preprocess and save images
20 url_count = 82783
21 filenames = os.listdir('./data/dataset/images/')
22 image_ids = list(range(1, url_count + 1))
23 dataset = {}
24 for filename in tqdm(filenames):
25     img = Image.open(os.path.join('./data/dataset/images/', filename))
26     img_id = int(re.split('\.', filename)[0])
27     try:
28         img = img.convert('RGB')
29         processed_img = img.resize((299,299), resample=Image.ANTIALIAS)
30         # print(filename, " --> ", img_id)
31         # image is greyscale
32         temp = np.array(processed_img)
33         if temp.shape == (299,299):
34             temp_ = [temp, temp, temp]
35             temp = np.stack(temp_, axis=2)
36         dataset[str(img_id)] = np.array(temp, dtype=np.uint8)
37     except:
38         print("Exception --> ", img_id)
39 pdb.set_trace()
40
41 # generate training and test datasets
42 with open('./data/dataset/dataset_inception.pkl', 'wb') as f:
43     pickle.dump(dataset, f)
```

```
1  import os
2  import pickle
3  import pdb
4
5  import numpy as np
6
7  from sklearn.model_selection import train_test_split
8
9  # dummy data
10 # dataset = dict(a=[1], b=[2], c=[3], d=[4], e=[5], f=[6], g=[7], h=[8], i=[9], j=[10])
11 with open("./data/dataset/processed_dataset_inception.pkl", "rb") as f:
12     dataset = pickle.load(f)
13
14 keys = list(dataset.keys())
15 values = list(dataset.values())
16
17 images_train_val, images_test, id_train_val, id_test = train_test_split(values, keys,
       shuffle=True, train_size=9/10)
18 images_train, images_val, id_train, id_val = train_test_split(images_train_val,
       id_train_val, shuffle=True, train_size = 7/9)
19
20 train_dataset = dict(keys=id_train, values=images_train)
21 validation_dataset = dict(keys=id_val, values=images_val)
22 test_dataset = dict(keys=id_test, values=images_test)
23
24 # save datasets
25 with open("./data/dataset/train_dataset_inception.pkl", "wb") as f:
26     pickle.dump(train_dataset, f)
27 with open("./datap/dataset/validation_dataset_inception.pkl", "wb") as f:
28     pickle.dump(validation_dataset, f)
29 with open("./data/dataset/test_dataset_inception.pkl", "wb") as f:
30     pickle.dump(test_dataset, f)
31
32 # load captions and split to train test and validation in a corresponding manner
33 with open("./utils/captions.pkl", "rb") as f:
34     captions = pickle.load(f)
35
36 train_captions = {}
37 for key in id_train:
38     train_captions[key] = captions[key]
39
40 validation_captions = {}
41 for key in id_val:
42     validation_captions[key] = captions[key]
43
```

```
test_captions = {}
for key in id_test:
    test_captions[key] = captions[key]

# save captions
with open("./data/dataset/train_captions_inception.pkl", "wb") as f:
    pickle.dump(train_captions, f)
with open("./data/dataset/validation_captions_inception.pkl", "wb") as f:
    pickle.dump(validation_captions, f)
with open("./data/dataset/test_captions_inception.pkl", "wb") as f:
    pickle.dump(test_captions, f)
```

5.3 Codes for VGG16 + LSTM

```
# from tensorflow.python.framework import ops
# ops.reset_default_graph()

import numpy as np
import tensorflow as tf

from keras.layers import Conv1D
from keras.models import Sequential
from keras.applications.inception_v3 import InceptionV3
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from keras.optimizers import Adam, RMSprop
from keras.layers.merge import add
from keras.layers import Input
from keras.layers import Dropout, Dense, LSTM, Embedding
from keras.models import Model, load_model
import keras.backend as K

import pickle
import pdb

def preprocess_dataset(dct):
    result = {}
    for key, value in zip(dct['keys'], dct['values']):
        result[key] = value
    return result

# all dataset
# with open("./utils/processed_dataset_vgg.pkl", "rb") as f:
#     all_features = pickle.load(f)
# with open("./utils/captions.pkl", "rb") as f:
#     all_captions = pickle.load(f)

# train dataset (70% of the data)
with open("./data/dataset/train_dataset_vgg.pkl", "rb") as f:
    train_features = pickle.load(f)
    train_encoded_images = preprocess_dataset(train_features)
with open("./data/dataset/train_captions_vgg.pkl", "rb") as f:
    train_captions = pickle.load(f)
    train_dataset = train_captions
    # train_dataset = preprocess_dataset(train_captions)

# validation dataset (20% of the data)
with open("./data/dataset/validation_dataset_vgg.pkl", "rb") as f:
    validation_features = pickle.load(f)
    validation_encoded_images = preprocess_dataset(validation_features)
with open("./data/dataset/validation_captions_vgg.pkl", "rb") as f:
    validation_captions = pickle.load(f)
    validation_dataset = validation_captions
    # validation_dataset = preprocess_dataset(validation_captions)

# test dataset (10% of the data)
with open("./data/dataset/test_dataset_vgg.pkl", "rb") as f:
    test_features = pickle.load(f)
    test_encoded_images = preprocess_dataset(test_features)
with open("./data/dataset/test_captions_vgg.pkl", "rb") as f:
    test_captions = pickle.load(f)
    test_dataset = test_captions
```

```
    # test_dataset = preprocess_dataset(test_captions)

# train_dataset = train_captions
# train_encoded_images = train_features

# test_dataset = test_captions
# test_encoded_images = test_features

# validation_dataset = validation_captions
# validation_encoded_images = validation_features


# model
max_length = 17
embedding_dim = 200
vocabulary_size = 1004

'''
Model takes 2 types of inputs, first being encoded photos by inception to 2048 dimensional vectors. Second input type to the model are sequences of words (i.e, captions).
Captions are all zero padded to max_length. With first item being SOS token and last item being EOS token. Dropout layers are used for prevention of overfitting. Photos are processed
such that their dimensions are reduced to 256 by dense layer with relu activation. Caption sequences are processed by the glove pretrained embedding matrix. Embedding dimension
is chosen as 200. Vocabulary size in the problem is 1020.
'''

inputs1 = Input(shape=(4096,))
features1 = Dropout(0.5)(inputs1)
features2 = Dense(256, activation='relu')(features1)
inputs2 = Input(shape=(max_length,))
seqfeatures1 = Embedding(vocabulary_size, embedding_dim, mask_zero=True)(inputs2)
seqfeatures2 = Dropout(0.5)(seqfeatures1)
seqfeatures3 = LSTM(256)(seqfeatures2)

#add two different types of feature extractions into one tensor.
decoder1 = add([features2, seqfeatures3])

decoder2 = Dense(256, activation='relu')(decoder1)
#softmax to predict target word over vocabulary.
outputs = Dense(vocabulary_size, activation='softmax')(decoder2)
#construction of the model.
model = Model(inputs=[inputs1, inputs2], outputs=outputs)


'''
Data generator function streams the input data to the model in batches. This will be used for fit_generator function with the model.
Note that, batch size is in unit of images. With batch_size = 3, there will be 3 images with corresponding training sample generation.
'''

batch_size = 6
def data_generator(dataset, imgs, max_length, batch_size = batch_size):
    X1, X2, y = list(), list(), list()
    n = 0
    #infinite loop over images
    while True:
        for key, captions_list in dataset.items():
            n += 1
            # retrieve the photo feature
            try:
                image = imgs[key]
            except:
                n -= 1
                continue
            for capt in captions_list:
                # split one sequence into multiple X, y pairs
                for i in range(1, len(capt)):
```

```
                    # split into input and output pair
                    in_seq, out_seq = capt[:i], capt[i]
                    if out_seq == 0:
                        break
                    # pad input sequence
                    in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                    # encode output sequence
                    out_seq = to_categorical([out_seq], num_classes=vocabulary_size)[0]
                    # store
                    X1.append(image)
                    X2.append(in_seq)
                    y.append(out_seq)
            # yield the batch data
            if n==batch_size:
                yield [[np.array(X1), np.array(X2)], np.array(y)]
                X1, X2, y = list(), list(), list()
                n=0

#get embedding matrix with shape 1024x200
print(model.summary())
embedding_matrix = np.load("./utils/embedding_matrix.npy")
model.layers[2].set_weights([embedding_matrix])
model.layers[2].trainable = False

#compile the model with categorical cross entropy, combined with softmax
model.compile(loss='categorical_crossentropy', optimizer='adam')

steps = len(train_dataset)//batch_size
# generator = data_generator(train_dataset, train_encoded_images, max_length, batch_size)
# model.optimizer.lr = 0.01
# model.fit_generator(generator, epochs=9, steps_per_epoch=steps, verbose=1)
# model.save('./model_' + '9epochs' + '.h5')
epochs = 50
# # load model
base = 0
# model = load_model('./model_weights/model_9.h5')
# model.optimizer.learning_rate = 0.00001

# print(K.eval(model.optimizer.lr))
# K.set_value(model.optimizer.lr, 0.000001)
# print(K.eval(model.optimizer.lr))


for i in range(1,epochs+1):
    if i == 5:
        batch_size = 12
        K.set_value(model.optimizer.lr, 0.0001)
    elif i == 10:
        batch_size = 18
        K.set_value(model.optimizer.lr, 0.00001)
    train_generator = data_generator(train_dataset, train_encoded_images, max_length,
    batch_size)
    validation_generator = data_generator(validation_dataset, validation_encoded_images,
    max_length, batch_size)
    # pdb.set_trace()
    model.fit_generator(train_generator, epochs=1, steps_per_epoch=steps, verbose=1,
    validation_data=validation_generator, validation_steps=steps, validation_freq=1)
    model.save('./model_weights/model_vgg_epoch' + str(base+i) + '.h5')
```

```
import os
import pdb
import pickle

import pandas as pd
import numpy as np

from keras.applications.vgg16 import VGG16, preprocess_input
from keras.models import Model
from tqdm import tqdm
from keras import backend as K


```

```
vgg = VGG16(weights='imagenet', include_top=True)
transfer_layer = vgg.get_layer('fc2') # one previous from last
model = Model(vgg.input, transfer_layer.output)

def extract_features(img):
    #img are 299x299x3 images of rgb
    img = preprocess_input(img)
    img = np.expand_dims(img,axis=0)
    res = model.predict(img)
    res = np.reshape(res, res.shape[1])
    return res

# import processed images
with open("./data/dataset/dataset_vgg.pkl", "rb") as f:
    dct = pickle.load(f)

images = list(dct.values())
ids = list(dct.keys())

dataset = {}
for (img, id) in tqdm(zip(images, ids)):
    feature = extract_features(img)
    dataset[id] = feature

# save image features
with open("./data/dataset/processed_dataset_vgg.pkl", "wb") as f:
    pickle.dump(dataset, f)
```

```
import os
import pdb
import h5py
import re
import pickle

import numpy as np

from PIL import Image
from tqdm import tqdm

with h5py.File('./data/eee443_project_dataset_train.h5', 'r') as f:
    captions = f['train_cap'][:]
    image_ids = f['train_imid'][:]
    images = f['train_ims'][:]
    urls = f['train_url'][:]
    word_codes = f['word_code'][:]

# preprocess and save images
url_count = 82783
filenames = os.listdir('./data/dataset/images/')
image_ids = list(range(1, url_count + 1))
dataset = {}
for filename in tqdm(filenames):
    img = Image.open(os.path.join('./data/dataset/images/', filename))
    img_id = int(re.split('\.', filename)[0])
    try:
        img = img.convert('RGB')
        processed_img = img.resize((224,224), resample=Image.ANTIALIAS)
        # print(filename, " --> ", img_id)
        # image is greyscale
        temp = np.array(processed_img)
        if temp.shape == (224,224):
            temp_ = [temp, temp, temp]
            temp = np.stack(temp_, axis=2)
        dataset[str(img_id)] = np.array(temp, dtype=np.uint8)
    except:
        print("Exception --> ", img_id)
pdb.set_trace()

# generate training and test datasets
with open('./data/dataset/dataset_vgg.pkl', 'wb') as f:
    pickle.dump(dataset, f)
```

```
import os
import pickle
import pdb

import numpy as np

from sklearn.model_selection import train_test_split

# dummy data
# dataset = dict(a=[1], b=[2], c=[3], d=[4], e=[5], f=[6], g=[7], h=[8], i=[9], j=[10])
with open("./data/dataset/processed_dataset_vgg.pkl", "rb") as f:
    dataset = pickle.load(f)

keys = list(dataset.keys())
values = list(dataset.values())

images_train_val, images_test, id_train_val, id_test = train_test_split(values, keys,
    shuffle=True, train_size=9/10)
images_train, images_val, id_train, id_val = train_test_split(images_train_val,
    id_train_val, shuffle=True, train_size = 7/9)

train_dataset = dict(keys=id_train, values=images_train)
validation_dataset = dict(keys=id_val, values=images_val)
test_dataset = dict(keys=id_test, values=images_test)

# save datasets
with open("./data/dataset/train_dataset_vgg.pkl", "wb") as f:
    pickle.dump(train_dataset, f)
with open("./datap/dataset/validation_dataset_vgg.pkl", "wb") as f:
    pickle.dump(validation_dataset, f)
with open("./data/dataset/test_dataset_vgg.pkl", "wb") as f:
    pickle.dump(test_dataset, f)

# load captions and split to train test and validation in a corresponding manner
with open("./utils/captions.pkl", "rb") as f:
    captions = pickle.load(f)

train_captions = {}
for key in id_train:
    train_captions[key] = captions[key]

validation_captions = {}
for key in id_val:
    validation_captions[key] = captions[key]

test_captions = {}
for key in id_test:
    test_captions[key] = captions[key]

# save captions
with open("./data/dataset/train_captions_vgg.pkl", "wb") as f:
    pickle.dump(train_captions, f)
with open("./data/dataset/validation_captions_vgg.pkl", "wb") as f:
    pickle.dump(validation_captions, f)
with open("./data/dataset/test_captions_vgg.pkl", "wb") as f:
    pickle.dump(test_captions, f)
```

References

[1] L. Fei-Fei, A. Iyer, C. Koch, and P. Perona, "What do we perceive in a glance of a real-world scene?" *Journal of vision*, vol. 7, no. 1, pp. 10–10, 2007.

[2] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[5] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[6] G. Kulkarni, V. Premraj, V. Ordonez, S. Dhar, S. Li, Y. Choi, A. C. Berg, and T. L. Berg, "Babytalk: Understanding and generating simple image descriptions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 12, pp. 2891–2903, 2013.

[7] A. Farhadi, M. Hejrati, M. A. Sadeghi, P. Young, C. Rashtchian, J. Hockenmaier, and D. Forsyth, "Every picture tells a story: Generating sentences from images," in *European conference on computer vision*. Springer, 2010, pp. 15–29.

[8] H. Fang, S. Gupta, F. Iandola, R. K. Srivastava, L. Deng, P. Dollár, J. Gao, X. He, M. Mitchell, J. C. Platt *et al.*, "From captions to visual concepts and back," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1473–1482.

[9] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3128–3137.

[10] J. Devlin, H. Cheng, H. Fang, S. Gupta, L. Deng, X. He, G. Zweig, and M. Mitchell, "Language models for image captioning: The quirks and what works," *arXiv preprint arXiv:1505.01809*, 2015.

[11] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3156–3164.

[12] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *In EMNLP*, 2014.

[13] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[14] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics, 2002, pp. 311–318.