DEMO 05

Joint text and image representations

Prashanth ASOK KUMAR
Annanya CHITRA KANNAN

Introduction:

In this Demo we will be making a model which places an image and it's caption close to each other in a space, such that the image and it's caption would be close to each other and the dissimilar caption would be far away. This closeness is determined by making use of Dot product between the space of image and caption.

We can use this model to get the image which is close to an inserted caption.

This Demo will be making use of the MSCOCO image captioning dataset from the below link:

http://mscoco.org/

we will not be making use of all the data set from MSCOCO, istead we will be making use of only 10,000 image with 1 caption per image.

We will be making use of the image processing model from http://www.image-net.org/ for processing the feature extraction from the images.

Pre-requisites:

We will be working on this demo using the keras 2.2.4 and TensorFlow 2.2.0. So we need to have these libraries installed in the system running this Demo. The code to install them are as follows:

```
pip install keras==2.2.4

pip install tensorflow==2.2.0
```

We have also updated the python files resnet50.py and imagenet_utils.py , to make it compatible with our environment.

Data Sets:

Since, it is difficult to process all the 10,000 images to generate it's feature representation, we will be making use of the numpy matrix which is the feature vector of the image dataset created based on the ResNet50 model. The vector is of the size 2048 and it is present in the file "*resnet50-features.10k.npy*". It can be loaded with the help of the below command:

```
import numpy as np
features = np.load('resnet50-features.10k.npy')
print(features.shape)
```

The caption data of the images are stored in the "*annotations.10k.txt*" file. This file has the data in such a way that each line has the image name followed by the caption for the image. The data in this file has already been tokenized and cleaned up. Also the entries correspond to the rows of matrix of the image features in the other file.

Extract Features of New Image:

We can extract the features from a new image using ResNet50 model by making use of the files *resnet50.py* and *imagenet_utils.py*

This can be done by making use of the below commands and the function:

```
from resnet50 import ResNet50
from keras.preprocessing import image
from imagenet_utils import preprocess_input

resnet_model = ResNet50(weights='imagenet', include_top=False)
```

Function:

```
def extract_features(img_path):
    img = image.load_img(img_path, target_size=(224, 224))
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)
    features = resnet_model.predict(x)
    return np.expand_dims(features.flatten(), axis=0)
```

The below command will help to get the features of the image you want to get the features of, let's consider the name of the image file as "*COCO_test2014_000000005583.jpg*"

```
features = extract_features('COCO_test2014_000000005583.jpg')
print(features.shape)
```

```
(1, 2048)
```

Loading the dataset:

Both the dataset can be loaded into the model for determining the similarity by making use of the below function:

```
def load(captions_filename, features_filename):
    features = np.load(features_filename)
    images = []
    texts = []
    with open(captions_filename) as fp:
        for line in fp:
            tokens= line.strip().split()
            images.append(tokens[0])
            texts.append(' '.join(tokens[1:]))
    return features, images, texts
```

This function outputs a tuple containing the 3 elements;

- Features of the Image
- Image Names
- Caption texts of the Images

We will be calling this load function and having the values stored in the following variables to use them later.

```
features, images, texts= load('annotations.10k.txt','resnet50-features.10k.npy')
```

3

Vocabulary generation:

Next, we need to convert the captions (stored in "texts")from list to integer sequences of same size as the features. This is achieved with the help of Tokenizer and pad_sequences from keras. We will also be setting the max length of the caption to 16.

The conversion of captions from list to integer sequence is achieved with the below code.

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences


tokenizer = Tokenizer()
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
captions = pad_sequences(sequences, maxlen=16)
```

We will now be making use of the tokenized texts stored as "tokenizer" to generate the vocabulary file which can be reloaded in the later point of time when required.

This is done with the following code:

```
import json

vocab = tokenizer.word_index
vocab['<eos>'] = 0 # add word with id 0


with open('vocab.json', 'w') as fp: # save the vocab
    fp.write(json.dumps(vocab))
```

Vocab embedding:

We will be embedding the weights to the vocabulary with the help of the vocabularies from "*glove.twitter.27B.100d.filtered.txt*" file for gated recurrent units (GRU) and placing the vocabulary in the 100-dimensional space.

This is done with the help of the below code :

```
import embedding
embedding_weights = embedding.load(vocab, 100,'glove.twitter.27B.100d.filtered.txt')
```

```
loading embeddings from "glove.twitter.27B.100d.filtered.txt"
```

Setting up Layers:

The model we are going to make will have 3 inputs and will have the shapes as follows:

- image features (2048,)
- caption (16,)
- noise caption (16,)

The code would be as follows:

```
from keras.layers import Input, Dense, Embedding, GRU
image_input = Input(shape=(2048,))
caption_input = Input(shape=(16,))
noise_input = Input(shape=(16,))
```

We will be making use of these inputs to form the individual layers which will act as places in the model. The 3 layers created and has the following function:

- Embedding layer – with vocab embeddings
- GRU layer – text representation
- Dense layer -image representation

These are done as follows:

```
caption_embedding = Embedding(len(vocab), 100, input_length=16, weights=[embedding_weights])
caption_rnn = GRU(256)
image_dense = Dense(256, activation='tanh')
```

Pipeline and Model Creation:

We will be creating the pipeline by making use of the layers created earlier. By making use of the below code, weights of the correct caption and noise pipelines are shared.

```
image_pipeline = image_dense(image_input)
caption_pipeline = caption_rnn(caption_embedding(caption_input))
noise_pipeline = caption_rnn(caption_embedding(noise_input))
```

Now, we will be making use of these pipelines to create the dot product between the image and caption to get the distance in space.

```
from keras.layers import dot, concatenate
positive_pair = dot([image_pipeline, caption_pipeline], axes=1)
negative_pair = dot([image_pipeline, noise_pipeline], axes=1)
output = concatenate([positive_pair, negative_pair])
```

Next we will be creating the Models, there are 3 models that can be created using the inputs and the output created earlier .

- *Training Model- it outputs the concatenation of positive & negative pairs*
- *Image Model – used during prediction*
- *Caption Model – used during prediction*

```
from keras.models import Model

training_model = Model([image_input, caption_input, noise_input], output)
image_model = Model(image_input, image_pipeline)
caption_model = Model(caption_input, caption_pipeline)
```

Metrics:

The metrics used for evaluation of the model are the loss and accuracy.

Keras doesn't have a built in loss function metric. So,we will be generating the loss using keras.backend functions from Theano and Tensorflow. By making use of the loss function formula below:

$$loss = \sum_i max(0, 1 - p_i + n_i)$$

We will be creating a function "*custom_loss*" to perform the loss evaluation, as follows:

```
from keras import backend as K
def custom_loss(y_true, y_pred):
    positive = y_pred[:,0]
    negative = y_pred[:,1]
    return K.sum(K.maximum(0., 1. - positive + negative))
```

Similarly we will be calculating the Mean Accuracy using the below function "*accuracy*" :

```
def accuracy(y_true, y_pred):
    positive = y_pred[:,0]
    negative = y_pred[:,1]
    return K.mean(positive > negative)
```

Now, we will be compiling the model to set the loss and accuracy model as metrics using the below code:

```
training_model.compile(loss=custom_loss, optimizer='adam', metrics=[accuracy])
```

Training the Model:

Now, that the Model has been created we will be training the model using our 10,000 image dataset. We will be splitting the dataset into 9,000 for Training and the rest 1,000 for validation, inorder to avoid overfitting of the model.

This is achieved by making use of the below code and the output of the training with the metrics and loss function in displayed, we will be training the model for 10 epochs:

```
noise = np.copy(captions)
fake_labels = np.zeros((len(features), 1))
X_train = [features[:9000], captions[:9000], noise[:9000]]
Y_train = fake_labels[:9000]
X_valid = [features[-1000:], captions[-1000:], noise[-1000:]]
Y_valid = fake_labels[-1000:]
# actual training
for epoch in range(10):
    np.random.shuffle(noise) # don't forget to shuffle mismatched captions
    training_model.fit(X_train, Y_train, validation_data=[X_valid, Y_valid], epochs=1, batch_size=64)
```

```
141/141 [==============================] - 10s 69ms/step - loss: 34.4825 - accuracy: 0.8184 - val_loss: 0.0000e+00
- val_accuracy: 0.0000e+00
141/141 [==============================] - 9s 65ms/step - loss: 13.8319 - accuracy: 0.9162 - val_loss: 0.0000e+00 -
val_accuracy: 0.0000e+00
141/141 [==============================] - 9s 66ms/step - loss: 11.6600 - accuracy: 0.9256 - val_loss: 0.0000e+00 -
val_accuracy: 0.0000e+00
141/141 [==============================] - 10s 69ms/step - loss: 10.4369 - accuracy: 0.9344 - val_loss: 0.0000e+00
- val_accuracy: 0.0000e+00
141/141 [==============================] - 9s 67ms/step - loss: 8.0597 - accuracy: 0.9477 - val_loss: 0.0000e+00 -
val_accuracy: 0.0000e+00
141/141 [==============================] - 10s 69ms/step - loss: 7.6363 - accuracy: 0.9543 - val_loss: 0.0000e+00 -
val_accuracy: 0.0000e+00
141/141 [==============================] - 11s 75ms/step - loss: 7.1619 - accuracy: 0.9535 - val_loss: 0.0000e+00 -
val_accuracy: 0.0000e+00
141/141 [==============================] - 10s 70ms/step - loss: 6.1744 - accuracy: 0.9625 - val_loss: 0.0000e+00 -
val_accuracy: 0.0000e+00
141/141 [==============================] - 10s 74ms/step - loss: 6.6632 - accuracy: 0.9574 - val_loss: 0.0000e+00 -
val_accuracy: 0.0000e+00
141/141 [==============================] - 10s 73ms/step - loss: 6.5364 - accuracy: 0.9601 - val_loss: 0.0000e+00 -
val_accuracy: 0.0000e+00
```

Saving & Loading Model:

Once the model has been trained, we can save the model for future while we need to use them for prediction, with the below code:

```
image_model.save('model.image')
caption_model.save('model.caption')
```

The same can be done to the representations as well:

```
np.save('caption-representations',caption_model.predict(captions))
np.save('image-representations', image_model.predict(features))
```

When we are in need of the models and representation during the prediction we can load them with the below code:

```
# load models
from keras.models import load_model
image_model = load_model('model.image')
caption_model = load_model('model.caption')
# load representations (you could as well recompute them)
import numpy as np
caption_representations = np.load('caption-representations.npy')
image_representations = np.load('image-representations.npy')
```

We can also retrieve the vocab by making use of the previously saved "*vocab.json*" file, instead of using the tokenizers again.

```
from keras.preprocessing.sequence import pad_sequences
import json
vocab = json.loads(open('vocab.json').read())
def preprocess_texts(texts):
    output = []
    for text in texts:
        output.append([vocab[word] if word in vocab else 0 for word in text.split()])
    return pad_sequences(output, maxlen=16)
```

Prediction of Captions:

The loaded data can be used to predict the captions of an image file. In order to do so, we will be making a function "*generate_caption*" which will be making use of the Model, Image Model to predict the captions close to it in the space.

The function is made as follows:

```
def generate_caption(image_filename, n=10):
    # generate image representation for new image
    image_representation = image_model.predict(extract_features(image_filename))

    # compute score of all captions in the dataset
    scores = np.dot(caption_representations,image_representation.T).flatten()

    # compute indices of n best captions
    indices = np.argpartition(scores, -n)[-n:]
    indices = indices[np.argsort(scores[indices])]

    for i in [int(x) for x in reversed(indices)]:
            print(scores[i], texts[i])
```

We can test our prediction with the image below and the output received is as follows:

The output received for the image is as shown below:

```
generate_caption('val2014/COCO_val2014_000000301581.jpg')
```

```
16.315874 a skier skiing down a snowy mountain
15.921094 a downhill skier shredding the slopes of snow
15.75975 a person on skis skiing down a mountain
15.396327 a skier is skiing quickly down a mountain
15.36192 a shadowy skier skiing down a snowy mountain
15.331093 a person is snowboarding down a snowy slope
15.112196 a man is skiing on the snow slopes
15.036899 a woman skiing poses on a snow slope
14.847397 this man is skiing down a mountain slope
14.730103 a woman skiing down a ski slope
```

Prediction of Images:

With the loaded data and model, we can predict the images with the caption as input, with our model. This is achieved by making use of the below function "*search_image*".

```
from IPython.display import Image, display


def search_image(caption, n=10):
    listOfImageNames=[]
    caption_representation = caption_model.predict(preprocess_texts([caption]))
    scores = np.dot(image_representations, caption_representation.T).flatten()
    indices = np.argpartition(scores, -n)[-n:]
    indices = indices[np.argsort(scores[indices])]
    for i in [int(x) for x in reversed(indices)]:
        print(scores[i], images[i])
        image_file='val2014/' + images[i]
        display(Image(filename=image_file))
```

When we call the function by inputing a caption , we get the result as follows:

```
search_image('a man in the snow on some skis')
```

```
20.738012 COCO_val2014_000000228135.jpg
```

```
20.470901 COCO_val2014_000000371241.jpg
```

```
20.41573 COCO_val2014_000000195567.jpg
```

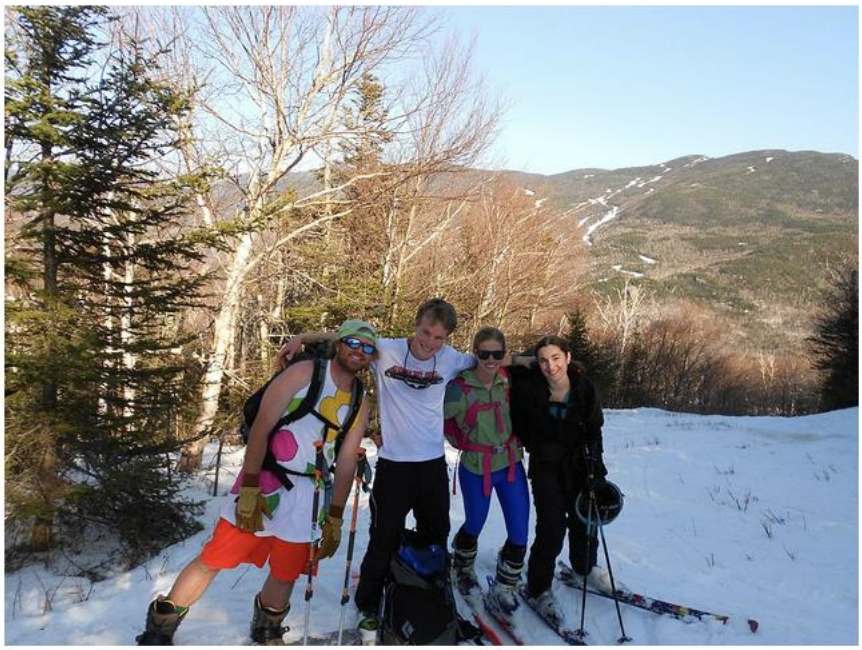
20.41573 COCO_val2014_000000195567.jpg

20.343077 COCO_val2014_000000122161.jpg

20.193474 COCO_val2014_000000351715.jpg

20.18501 COCO_val2014_000000165476.jpg

20.115982 COCO_val2014_000000199989.jpg



20.090803 COCO_val2014_000000032331.jpg

20.090803 COCO_val2014_000000032331.jpg

19.98804 COCO_val2014_000000299261.jpg

19.879683 COCO_val2014_000000116389.jpg

Conclusion:

With this model we will be able to find the image and tags related to it and to get more results with wide vocabulary we can train the model with all the data sets available in MSCOCO. But extracting the feature vectors and training the Model would take a lot of time. That too if the model is aimed to generate an accuracy of more than 90%, it can be trained faster, since as per our training we were able to see that the Model reached 90% in the second epoch itself.