Submitted by

Harish Tata

Class ID: 25

Team 4

Objectives:

Image Caption Generator:

1. Create your own Show and Tell Model using your dataset. (Describe why did you choose the dataset for creating the model)
2. Generate captions for your own dataset using the Show and Tellmodel.
3. Report your accuracy in BLEU, CIDER, METEOR and ROGUE measures.

Data Analytics based on Unsupervised Learning

1. Perform analytics on your own dataset using a machine learning and Deep Learning based Unsupervised Learning.

Dataset

- “Google Conceptual Captions” data set.

Create your own Show and Tell Model using your dataset

Extract Features

- First, give the images as “input” to the pre-trained model -“VGG MODEL” to get the image features out of it.
- We have removed the last layer for the extracting of the features not for the classification
- Store the image features and it is saved as “features.pkl”
  Reference link:- https://machinelearningmastery.com/use-pre-trained-vgg-model-classify-objects-photographs/

Code

```python
# extract features from each photo in the directory
def extract_features(directory):
    # load the model
    model = VGG16()
    # re-structure the model
    model.layers.pop()
    model = Model(inputs=model.inputs, outputs=model.layers[-1].output)
    # summarize
    print(model.summary())
    # extract features from each photo
    features = dict()
    for name in listdir(directory):
        # 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

# extract features from all images
directory = 'images'
features = extract_features(directory)
print('Extracted Features: %d' % len(features))
# save to file
dump(features, open('features.pkl', 'wb'))
```

Output

```
_________________________________________________________________
(type)                 Output Shape              Param #
=================================================================
1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
```

Preprocess the caption data

- First, take the input i.e caption.txt file.
- Convert the text into lower and remove the punctuations

Code

```
import string

desc_list = []
# save descriptions to file, one per line
def clean_descriptions():
    table = str.maketrans('', '', string.punctuation)
    index = 0
    with open("captions/captions.txt") as openfile:
        for line in openfile:
            index = index+1
            desc = line
            # 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.append(' '.join(desc))
```

- Save the image id and the description in a text file

```
def save_descriptions(descriptions, filename,_list):
    total_list = []
    for index in range(len(_list)):
        total_list.append(str(index+1) + ' '+str(_list[index]))
    data = '\n'.join(total_list)
    file = open(filename, 'w')
    file.write(data)
    file.close()
```

- Make two different text file for train and validation

- The following sets are divided into 70:30 ratio.

```
def generate():
    length = len(desc_list)
    train = .7 * length
    validate = .3 * length
    train_list =[]
    validate_list=[]
    for i in range(int(train)):
        train_list.append(str(i+1)+".jpg")
    train_data = '\n'.join(train_list)
    file = open("train/train.txt", 'w')
    file.write(train_data)
    file.close()
    for i in range(int(validate)):
        validate_list.append(str(1+i+int(train))+".jpg")
    validate_data = '\n'.join(validate_list)
    file = open("validation/validation.txt", 'w')
    file.write(validate_data)
    file.close()
```

Training a Deep Learning Model

- Give the train images, output features.pkl file and description.txt as inputs for the model.
- Load the datasets load_set()(this is used for loading)
- Give the descriptions.txt to “load_clean_descriptions()” which parses and makes a dictionary of caption with a startseq and endseq.It is used as a signal i.e end of the process.

```
# load clean descriptions into memory
def load_clean_descriptions(filename, dataset):
    # load document
    doc = load_doc(filename)
    descriptions = dict()
    for line in doc.split('\n'):
        # split line by white space
        tokens = line.split()
        # split id from description
        image_id, image_desc = tokens[0], tokens[1:]
        # skip images not in the set
        if image_id in dataset:
            # create list
            if image_id not in descriptions:
                descriptions[image_id] = list()
            # wrap description in tokens
            desc = 'startseq ' + ' '.join(image_desc) + ' endseq'
            # store
            descriptions[image_id].append(desc)
    return descriptions
```

- load_photo_features() will load the photo features and filter only what we needed

- to_lines() will convert the dictionary into a list of strings and create_tokenizer to fit a tokenizer.

```
# load photo features
def load_photo_features(filename, dataset):
    # load all features
    all_features = load(open(filename, 'rb'))
    # filter features
    features = {k: all_features[k] for k in dataset}
    return features

# covert 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

# fit a tokenizer given caption descriptions
def create_tokenizer(descriptions):
    lines = to_lines(descriptions)
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(lines)
    return tokenizer
```

- create_sequences() will transform the data into two input arrays.

1. photo feature
2. encoded text

- The above output is sent for next word sequence.
- Input text is encoded as integers ad fed into word embedding layer.
- The model gives the output i.e prediction as a probability distribution over other words.
- output data is encoded using one-hot.

```
# create sequences of images, input sequences and output words for an image
def create_sequences(tokenizer, max_length, descriptions, photos):
    X1, X2, y = list(), list(), list()
    # walk through each image identifier
    for key, desc_list in descriptions.items():
        # walk through each description for the image
        for desc in desc_list:
            # encode the sequence
            seq = tokenizer.texts_to_sequences([desc])[0]
            # 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_length)[0]
                # encode output sequence
                out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
                # store
                X1.append(photos[key][0])
                X2.append(in_seq)
                y.append(out_seq)
    return array(X1), array(X2), array(y)
```

Model

- The feature extractor model needs input photo features of a vector of 4096 elements.

- The sequence processor model needs input sequences with a predefined length which is fed to the embedding layer and followed by an LSTM layer with 256 memory units.

- Both of the above models produce 256 element vector and 50% dropout to avoid overfitting.

- At last decoder, the model combines both using an addition operation and fed to a dense 256 neuron layer that can predict the entire output vocabulary for the next word.

```
# define the captioning model
def define_model(vocab_size, max_length):
    # feature extractor model
    inputs1 = Input(shape=(4096,))
    fe1 = Dropout(0.5)(inputs1)
    fe2 = Dense(256, activation='relu')(fe1)
    # sequence model
    inputs2 = Input(shape=(max_length,))
    se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
    se2 = Dropout(0.5)(se1)
    se3 = LSTM(256)(se2)
    # decoder model
    decoder1 = add([fe2, se3])
    decoder2 = Dense(256, activation='relu')(decoder1)
    outputs = Dense(vocab_size, activation='softmax')(decoder2)
    # tie it together [image, seq] [word]
    model = Model(inputs=[inputs1, inputs2], outputs=outputs)
    model.compile(loss='categorical_crossentropy', optimizer='adam')
    # summarize model
    print(model.summary())
    plot_model(model, to_file='model.png', show_shapes=True)
    return model
```

- Saving the best-trained model

```
# define the model
model = define_model(vocab_size, max_length)
# define checkpoint callback
filepath = 'model-ep{epoch:03d}-loss{loss:.3f}-val_loss{val_loss:.3f}.h5'
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=True, mode='min')
# fit model
model.fit([X1train, X2train], ytrain, epochs=20, verbose=2, callbacks=[checkpoint], validation_data=([X1test, X2test], ytest))
model.save("final.h5")
```

Output

```
/home/harish/anaconda3/envs/mypy36/bin/python "/home/harish/UMKC SPRING/BDAA/CS5542_LAB25_3/SourceCode/show_and_tell/model.py"
Using TensorFlow backend.
Dataset: 140
Descriptions: train=140
Photos: train=140
Vocabulary Size: 385
Description Length: 21
Dataset: 60
Descriptions: test=60
Photos: test=60
```

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	(None, 21)	0	
input_1 (InputLayer)	(None, 4096)	0	
embedding_1 (Embedding)	(None, 21, 256)	98560	input_2[0][0]
dropout_1 (Dropout)	(None, 4096)	0	input_1[0][0]
dropout_2 (Dropout)	(None, 21, 256)	0	embedding_1[0][0]
dense_1 (Dense)	(None, 256)	1048832	dropout_1[0][0]
lstm_1 (LSTM)	(None, 256)	525312	dropout_2[0][0]
add_1 (Add)	(None, 256)	0	dense_1[0][0] lstm_1[0][0]
dense_2 (Dense)	(None, 256)	65792	add_1[0][0]
dense_3 (Dense)	(None, 385)	98945	dense_2[0][0]

```
Total params: 1,837,441
Trainable params: 1,837,441
```

Generate the captions

Give the description.txt and tokenizer.pkl as input for generating the caption for the image

Code

```python
def generate_desc(model, tokenizer, photo, max_length):
    # seed the generation process
    in_text = 'startseq'
    # iterate over the whole length of the sequence
    for i in range(max_length):
        # integer encode input sequence
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        # pad input
        sequence = pad_sequences([sequence], maxlen=max_length)
        # predict next word
        yhat = model.predict([photo,sequence], verbose=0)
        # convert probability to integer
        yhat = argmax(yhat)
        # map integer to word
        word = word_for_id(yhat, tokenizer)
        # stop if we cannot map the word
        if word is None:
            break
        # append as input for generating the next word
        in_text += ' ' + word
        # stop if we predict the end of the sequence
        if word == 'endseq':
            break
    return in_text

# load the tokenizer
tokenizer = load(open('tokenizer.pkl', 'rb'))
# pre-define the max sequence length (from training)
max_length = 21
# load the model
model = load_model('final.h5')
image_list = [f for f in listdir("testimages") if isfile(join("testimages", f))]

for i in range(len(image_list)):
    photo = extract_features("testimages/"+image_list[i])
    description = generate_desc(model, tokenizer, photo, max_length)
    print(description)
```

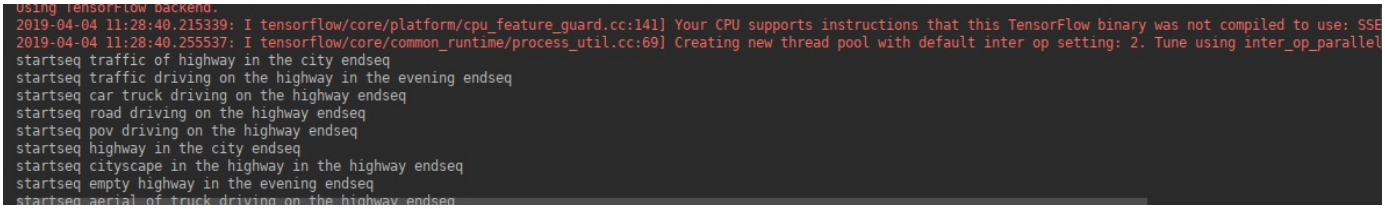
for i in range(len(image_list))

Output

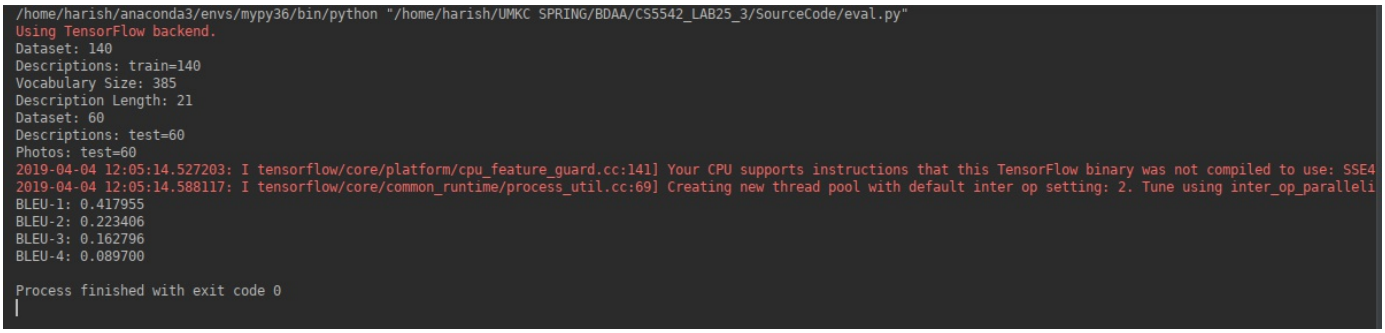
www.alamy.com - EC4747

www.alamy.com - ARTM0C

Captions



Measure the accuracy of the generated captions in BLEU



Data Analytics based on Unsupervised Learning

- Text classification is done with the help of K-Means Clustering.
- Give the captions data as input to the k-means.
- The number of clusters = 5 is given to the code.
- The code will form 5 clusters on the caption data
- Test a caption to which cluster it belongs

Code

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score

captions = []
caption_file = open("highway_caption.txt", encoding="utf8")
for caption in caption_file:
   captions.append(caption.split(' ', 1)[1])

vectorizer = TfidfVectorizer(stop_words='english')
X = vectorizer.fit_transform(captions)

true_k = 5
model = KMeans(n_clusters=true_k, init='k-means++', max_iter=100, n_init=1)
model.fit(X)

print("Top terms per cluster:")
order_centroids = model.cluster_centers_.argsort()[:, ::-1]
terms = vectorizer.get_feature_names()
for i in range(true_k):
    print("Cluster %d:" % i),
    for ind in order_centroids[i, :10]:
        print(' %s' % terms[ind]),

print("\n")
print("Prediction")

Y = vectorizer.transform(["high speed train passing by on a bridge at a highway with traffic"])
prediction = model.predict(Y)
print("high speed train passing by on a bridge at a highway with traffic")
print(prediction)
```

Output

Clusters

```
Top terms per cluster:
Cluster 0:
 mountain
 shot
 highway
 drone
 range
 tunnel
 angle
 view
 dessert
 wintry
Cluster 1:
 jam
 night
 highway
 yellow
 follows
 filled
 filming
 flat
 flood
 flow
Cluster 2:
 truck
 car
 driving
 view
 highway
 traveling
 video
 road
 light
 pass
Cluster 3:
 highway
```

Prediction

```
Prediction
high speed train passing by on a bridge at a highway with traffic
[3]
```

REFERENCES

https://machinelearningmastery.com/use-pre-trained-vgg-model-classify-objects-photographs/