CS6301 MACHINE LEARNING – MINI PROJECT

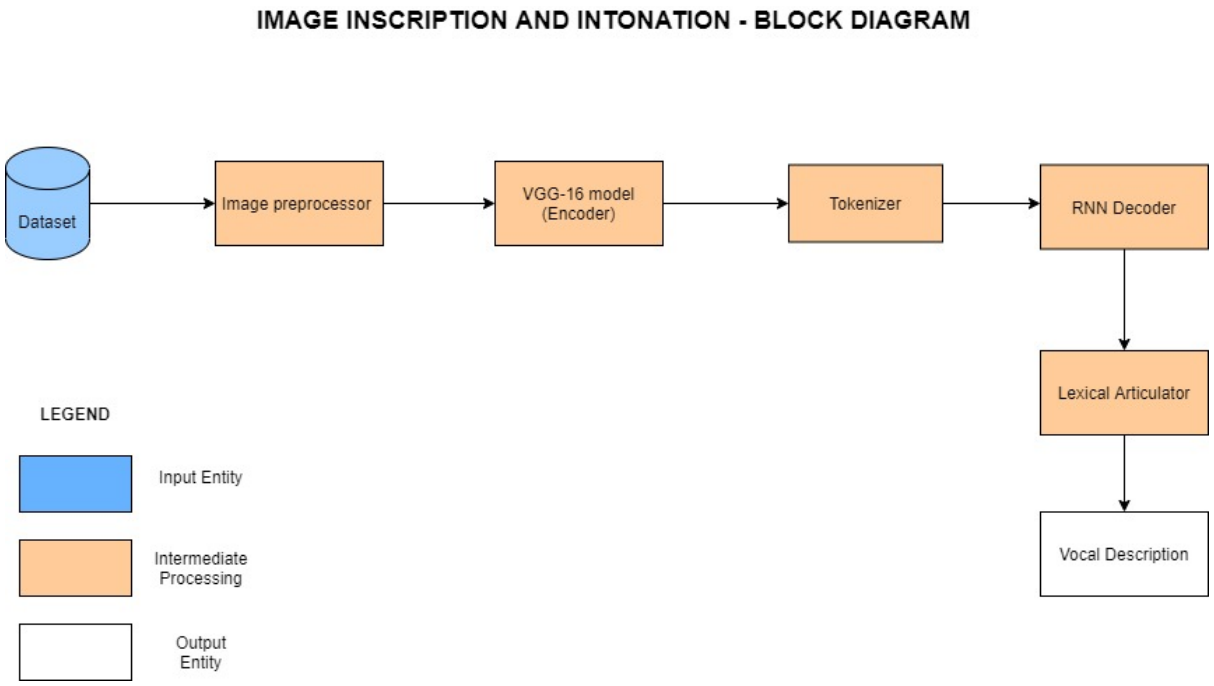
SRIHARI. S T.K.S. ARUNACHALAM

2018103601 2018103616

IMAGE INSCRIPTION AND INTONATION - A NEURAL NETWORK APPROACH

50% IMPLEMENTATION UPDATE DOCUMENTATION

DATASET USED: MS-COCO Dataset (https://cocodataset.org/#download)



IMPLEMENTATION PROGRESS	
Completed Modules	Image Pre-processor VGG-16 Model (Encoder) Tokenizer
Ongoing Modules	RNN Decoder Lexical Articulator

INDIVIDUAL CONTRIBUTION	
SRIHARI.S	**ARUNACHALAM.T.K.S.**
Caption Pre-processing	Image Pre-processing
Transfer Learning on the dataset using VGG16	Incorporating PyCoCo Tools for dealing with the data
Vocabulary construction	Embeddings productions

MODULE - 1 PRE-PROCESSOR

```
%matplotlib inline
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
import sys
import requests
import urllib.request
import tarfile
import zipfile
import json
import os
import pickle
import numpy as np
from PIL import Image
from tensorflow.keras import backend as K
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.applications import VGG16
```

```
data_dir = "/content/data/coco"
train_dir = "/content/data/coco/train2014"
val_dir = "/content/data/coco/val2014"
data_url = "http://images.cocodataset.org/"
```

An iterative approach has been chosen for the implementation of the problem statement. The first goal was to understand the nature of the dataset and pre-process it. The input MS-COCO 2014 dataset is of size 25 GB. In-order to deal with this huge dataset and the constrained computing resources we make use of the dynamic programming paradigm by caching the values, the first time the dataset is downloaded, in-order to make access faster the subsequent times. The dataset consisting of both images and captions together is loaded. The images and the corresponding captions are then segregated and stored separately. The images then undergo normalization followed by scaling to finish the pre-processing. On the other hand, the captions are encoded in a dictionary and are thus pre-processed so that it could be used by the tokenizer.

The cached file is stored as a pickle object and the function to accomplish this is defined as below. This is used to persist the data so it can be reloaded very quickly and easily. If the cache-file exists then the data is reloaded and returned, otherwise the function is called and the result is saved to cache.

```
def cache(cache_path, fn, *args, **kwargs):
    if os.path.exists(cache_path):
        with open(cache_path, mode='rb') as file:
            obj = pickle.load(file)
        print("Data loaded from cache-file: " + cache_path)
    else:
        obj = fn(*args, **kwargs)

        with open(cache_path, mode='wb') as file:
            pickle.dump(obj, file)

        print("Data saved to cache-file: " + cache_path)

    return obj
```

We facilitate the download of the dataset in the desirable format with the aid of the below functions.

```python
def maybe_download_and_extract_2():
    filenames = ["zips/train2014.zip", "zips/val2014.zip",
                 "annotations/annotations_trainval2014.zip"]
    for filename in filenames:
        url = data_url + filename
        print("Downloading " + url)
        maybe_download_and_extract(url=url, download_dir=data_dir)
```

```python
def _print_download_progress(count, block_size, total_size):
    pct_complete = float(count * block_size) / total_size
    pct_complete = min(1.0, pct_complete)
    msg = "\r- Download progress: {0:.1%}".format(pct_complete)
    sys.stdout.write(msg)
    sys.stdout.flush()
```

```python
def maybe_download_and_extract(url, download_dir):
    filename = url.split('/')[-1]
    file_path = os.path.join(download_dir, filename)
    if not os.path.exists(file_path):
        if not os.path.exists(download_dir):
            os.makedirs(download_dir)
        file_path, _ = urllib.request.urlretrieve(url=url, filename=file_path,
                                                  reporthook=_print_download_progress)

        print()
        print("Download finished. Extracting files.")

        if file_path.endswith(".zip"):
            zipfile.ZipFile(file=file_path, mode="r").extractall(download_dir)
        elif file_path.endswith((".tar.gz", ".tgz")):
            tarfile.open(name=file_path, mode="r:gz").extractall(download_dir)
        print("Done.")
    else:
        print("Data has apparently already been downloaded and unpacked.")
```

Downloading the dataset:

```
1 maybe_download_and_extract_2()
```

```
Downloading http://images.cocodataset.org/zips/train2014.zip
- Download progress: 100.0%
Download finished. Extracting files.
Done.
Downloading http://images.cocodataset.org/zips/val2014.zip
- Download progress: 67.7%
```

The COCO data-set contains a large number of images and various data for each image stored in a JSON-file. The load_records function provides the functionality to get a list of image-filenames (but not actually loading the images) along with their associated data such as text-captions describing the contents of the images.

```
1  def load_records(train=True):
2      if train:
3          cache_filename = "records_train.pkl"
4      else:
5          cache_filename = "records_val.pkl"
6
7      cache_path = os.path.join(data_dir, cache_filename)
8      records = cache(cache_path=cache_path,
9                      fn=_load_records,
10                     train=train)
11     return records
```

```python
def _load_records(train=True):
    if train:
        filename = "captions_train2014.json"
    else:
        filename = "captions_val2014.json"

    path = os.path.join(data_dir, "annotations", filename)
    with open(path, "r", encoding="utf-8") as file:
        data_raw = json.load(file)

    images = data_raw['images']
    annotations = data_raw['annotations']
    records = dict()

    for image in images:
        image_id = image['id']
        filename = image['file_name']
        record = dict()
        record['filename'] = filename
        record['captions'] = list()
        records[image_id] = record

    for ann in annotations:
        image_id = ann['image_id']
        caption = ann['caption']
        record = records[image_id]
        record['captions'].append(caption)

    records_list = [(key, record['filename'], record['captions'])
                    for key, record in sorted(records.items())]

    ids, filenames, captions = zip(*records_list)
    return ids, filenames, captions
```

```python
_, filenames_train, captions_train = load_records(train=True)
_, filenames_val, captions_val = load_records(train=False)
num_images_train = len(filenames_train)
num_images_train
```

The below given load_image function accomplishes the job of image pre-processing. It loads the image from the given file-path and resizes it to the given size. The images are scaled so that their pixels fall between 0.0 and 1.0. It is then plotted with the show_image function.

```
def load_image(path, size=None):
    img = Image.open(path)
    if not size is None:
        img = img.resize(size=size, resample=Image.LANCZOS)

    img = np.array(img)
    img = img / 255.0
    if (len(img.shape) == 2):
        img = np.repeat(img[:, :, np.newaxis], 3, axis=2)

    return img
```

```
def show_image(idx, train):
    if train:
        dir = train_dir
        filename = filenames_train[idx]
        captions = captions_train[idx]
    else:
        dir = val_dir
        filename = filenames_val[idx]
        captions = captions_val[idx]

    path = os.path.join(dir, filename)
    for caption in captions:
        print(caption)

    img = load_image(path)
    plt.imshow(img)
    plt.show()
```

The pycocotools has been put into use. It is a Python API that assists in loading, parsing and visualizing the annotations in COCO. We instantiate the COCO class by passing the json file as an argument.

```
%matplotlib inline
from pycocotools.coco import COCO
import numpy as np
import skimage.io as io
import matplotlib.pyplot as plt
import pylab
pylab.rcParams['figure.figsize'] = (8.0, 10.0)
```

```
capFile='/content/drive/MyDrive/data/coco/annotations/captions_val2014.json'
coco1=COCO(capFile)

```

```
loading annotations into memory...
Done (t=3.15s)
creating index...
index created!
```

Sample of the Annotations:

Annotations is a list of dictionaries. The dictionary contains the image_id, id(caption id), caption as the keys. Here id is the primary key and is used to retrieve a unique caption

```
annIds = coco1.getAnnIds()
annotations = coco1.loadAnns(annIds)

for i in range(5):
  print(annotations[i])
```

```
{'image_id': 203564, 'id': 37, 'caption': 'A bicycle replica with a clock as the front wheel.'}
{'image_id': 179765, 'id': 38, 'caption': 'A black Honda motorcycle parked in front of a garage.'}
{'image_id': 322141, 'id': 49, 'caption': 'A room with blue walls and a white sink and door.'}
{'image_id': 16977, 'id': 89, 'caption': 'A car that seems to be parked illegally behind a legally parked car'}
{'image_id': 106140, 'id': 98, 'caption': 'A large passenger airplane flying through the air.'}
```

img is a dictionary with the following keys. We use the coco_url to load and display the image

```
# load and display image
img = coco1.loadImgs(imgIds[np.random.randint(0,len(imgIds))])[0]
print(img)

```

```
{'coco_url': 'http://images.cocodataset.org/val2014/COCO_val2014_000000260818.jpg',
 'date_captured': '2013-11-25 21:07:37',
 'file_name': 'COCO_val2014_000000260818.jpg',
 'flickr_url': 'http://farm3.staticflickr.com/2023/1685630518_00b15897ab_z.jpg',
 'height': 500,
 'id': 260818,
 'license': 5,
 'width': 375}
```

Using the load_img method to display an image from the url:

```
# load and display image
img = coco1.loadImgs(imgIds[np.random.randint(0,len(imgIds))])[0]
# Or use url to load image
I = io.imread(img['coco_url'])
plt.axis('off')
plt.imshow(I)
plt.show()


```

MODULE – 2 VGG 16 MODEL ENCODER

```
1 image_model = VGG16(include_top=True, weights='imagenet')
2 image_model.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
_________________________________________________________________
```

We instantiate the VGG16 architecture by importing it from tensorflow.keras.applications. It loads weights pre-trained on ImageNet. The default

input size for this model is 224x224. We remove the last predictions layer and transfer the values of the second fully connected layer.

```
transfer_layer = image_model.get_layer('fc2')
```

The get_layer() method takes as parameters name of the specific layer which we want and retrieves the layer correspondingly.The transfer_layer variable has the fc2 layer stored. We make use of the K.int_shape() function which returns the shape of tensor or variable as a tuple of int or None entries.

i. K.int_shape(image_model.input) – Denotes shape of input vector to the model having value (None, 224, 224, 3).
ii. K.int_shape(transfer_layer.output) – Denotes shape of output vector of fc2 layer having value (None, 4096).

Thus, we assign img_size and transfer_values_size the values (224,224) and 4096 respectively.

```
transfer_layer = image_model.get_layer('fc2')
image_model_transfer = Model(inputs=image_model.input, outputs=transfer_layer.output)
img_size = K.int_shape(image_model.input)[1:3]
print(img_size)
transfer_values_size = K.int_shape(transfer_layer.output)[1]
print(transfer_values_size)
```

Next step is to process all the images with the vgg16 model and cache the values.

In-order to cache the transfer-values, the cache function is called upon with the path as data/coco/transfer_values_train.pkl .

If the cache path i.e. the transfer_values_train.pkl file exists, we return the obj. If it doesn't exist the caching of the transfer-values occurs in batches of images. During this process, we load the image using load_image() function defined previously and the images are resized to meet the expected format for the vgg16 architecture. Thus, we have cached the transfer values for the 82783 images in the training dataset by having the features extracted from the image from the output of the fc2 layer of vgg16 model.

```
def process_images(data_dir, filenames, batch_size=32):
    num_images = len(filenames)
    shape = (batch_size,) + img_size + (3,)
    image_batch = np.zeros(shape=shape, dtype=np.float16)

    shape = (num_images, transfer_values_size)
    transfer_values = np.zeros(shape=shape, dtype=np.float16)
    start_index = 0

    while start_index < num_images:
        print_progress(count=start_index, max_count=num_images)
        end_index = start_index + batch_size
        if end_index > num_images:
            end_index = num_images
        current_batch_size = end_index - start_index

        for i, filename in enumerate(filenames[start_index:end_index]):
            path = os.path.join(data_dir, filename)
            img = load_image(path, size=img_size)
            image_batch[i] = img

        transfer_values_batch = \
            image_model_transfer.predict(image_batch[0:current_batch_size])
        transfer_values[start_index:end_index] = \
            transfer_values_batch[0:current_batch_size]
        start_index = end_index

    print()
    return transfer_values
```

```
def process_images_train():
    print("Processing {0} images in training-set ...".format(len(filenames_train)))
    cache_path = os.path.join(data_dir, "transfer_values_train.pkl")
    transfer_values = cache(cache_path=cache_path, fn=process_images,
                            data_dir=train_dir, filenames=filenames_train)
    return transfer_values
```

```
def process_images_val():
    print("Processing {0} images in validation-set ...".format(len(filenames_val)))
    cache_path = os.path.join(data_dir, "transfer_values_val.pkl")
    transfer_values = cache(cache_path=cache_path, fn=process_images,
                            data_dir=val_dir, filenames=filenames_val)
    return transfer_values
```

```
%%time
transfer_values_train = process_images_train()
print("dtype:", transfer_values_train.dtype)
print("shape:", transfer_values_train.shape)
```

```
Processing 82783 images in training-set ...
- Progress: 100.0%
- Data saved to cache-file: data/coco/transfer_values_train.pkl
dtype: float16
shape: (82783, 4096)
CPU times: user 26min 30s, sys: 5min 17s, total: 31min 48s
Wall time: 34min 50s
```

```
%%time
transfer_values_val = process_images_val()
print("dtype:", transfer_values_val.dtype)
print("shape:", transfer_values_val.shape)
```

```
Processing 40504 images in validation-set ...
- Progress: 99.9%
- Data saved to cache-file: data/coco/transfer_values_val.pkl
dtype: float16
shape: (40504, 4096)
CPU times: user 12min 56s, sys: 2min 32s, total: 15min 29s
Wall time: 16min 29s
```

```
def print_progress(count, max_count):
    pct_complete = count / max_count
    msg = "\r- Progress: {0:.1%}".format(pct_complete)
    sys.stdout.write(msg)
    sys.stdout.flush()
```

In-order to keep track of the progress, the percentage of completed downloads is constantly updated.

MODULE - 3 TOKENIZER

Neural Networks cannot work directly on text-data. We use a two-step process to convert text into numbers that can be used in a neural network. The first step is to convert text-words into so-called integer-tokens. The second step is to convert integer-tokens into vectors of floating-point numbers using a so-called embedding-layer.

```
mark_start = 'ssss '
mark_end = ' eeee'
```

Before we can start processing the text, we first need to mark the beginning and end of each text-sequence with unique words that most likely aren't present in the data.

```
def mark_captions(captions_listlist):
    captions_marked = [[mark_start + caption + mark_end
                        for caption in captions_list]
                        for captions_list in captions_listlist]

    return captions_marked
```

The mark_captions function wraps all text-strings in the above markers. Since the captions are a list of lists, we use a nested for-loop to process using list-comprehension in python. In the inner loop we iterate over all the captions as we have four to six captions for each image. For each caption we append the start and end marker. Thus, captions_marked is a list of lists which is returned by this function to the variable captions_train_marked.

```
captions_train[0]
```

```
['Closeup of bins of food that include broccoli and bread.',
 'A meal is presented in brightly colored plastic trays.',
 'there are containers filled with different kinds of foods',
 'Colorful dishes holding meat, vegetables, fruit, and bread.',
 'A bunch of trays that have different food.']
```

captions_train is obtained from the pre-processor module. It’s a tuple of lists. Thus, each element of the tuple captions_train is as shown above.

```
captions_train_marked = mark_captions(captions_train)
```

```
captions_train_marked[0]
```

```
['ssss Closeup of bins of food that include broccoli and bread. eeee',
 'ssss A meal is presented in brightly colored plastic trays. eeee',
 'ssss there are containers filled with different kinds of foods eeee',
 'ssss Colorful dishes holding meat, vegetables, fruit, and bread. eeee',
 'ssss A bunch of trays that have different food. eeee']
```

captions_train_marked is a list of lists where each caption has the start and end markers added. Thus, each element of the tuple captions_train_marked is like shown above.

```
def flatten(captions_listlist):
    captions_list = [caption
                     for captions_list in captions_listlist
                     for caption in captions_list]

    return captions_list
```

```
captions_train_flat = flatten(captions_train_marked)
```

Now process all the captions in the training-set. Next we make a call to flatten() function to which we pass as parameters the marked captions. The functionality achieved here is pretty straightforward and simple.

Basically, we're converting list of lists into a single list. But now we unwrap the inner lists to provide a single flattened list. Thus, captions_list is a python list having all the captions in the training dataset. This is returned to captions_train_flat.

```
num_words = 10000
```

```
%%time
tokenizer = TokenizerWrap(texts=captions_train_flat,
                          num_words=num_words)
token_start = tokenizer.word_index[mark_start.strip()]
token_end = tokenizer.word_index[mark_end.strip()]
print("Start Token: ", token_start)
print("End Token: ", token_end)
```

```
Start Token:  2
End Token:  3
CPU times: user 6.69 s, sys: 1.76 ms, total: 6.69 s
Wall time: 6.75 s
```

The TokenizerWrap is inherited from the Tokenizer class in tensorflow.keras.preprocessing.text as we need more functionality than provided by this Tokenizer class so we wrap it.

We instantiate a TokenizerWrap object as below by passing the flat list of captions and num_words as parameters. TokenizerWrap is used to convert a text into sequence of integers. The maximum number of words in the vocabulary is set using the num_words variable to 10000. This means that we will only use the 10000 most frequent words in the captions from the training-data.

```
%%time
tokens_train = tokenizer.captions_to_tokens(captions_train_marked)
```

```
CPU times: user 6.14 s, sys: 101 ms, total: 6.24 s
Wall time: 6.27 s
```

Next we pass the flattened caption to the fit_on_texts() function of Tokenizer class. This function updates internal vocabulary based on a list of texts. Then we iterate over the items of the list.

Now create a tokenizer using all the captions in the training-data. The flattened list of captions is used to create the tokenizer because it cannot take a list-of-lists.

```python
class TokenizerWrap(Tokenizer):
    def __init__(self, texts, num_words=None):
        Tokenizer.__init__(self, num_words=num_words)
        self.fit_on_texts(texts)
        self.index_to_word = dict(zip(self.word_index.values(),
                                      self.word_index.keys()))

    def token_to_word(self, token):
        word = " " if token == 0 else self.index_to_word[token]
        return word

    def tokens_to_string(self, tokens):
        words = [self.index_to_word[token]
                 for token in tokens
                 if token != 0]
        text = " ".join(words)

        return text

    def captions_to_tokens(self, captions_listlist):
        tokens = [self.texts_to_sequences(captions_list)
                  for captions_list in captions_listlist]

        return tokens
```

```python
def text_to_word_sequence(text,
                          filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n',
                          lower=True, split=" "):
    if lower:
        text = text.lower()

    translate_dict = {c: split for c in filters}
    print("translate_dict = ",translate_dict)
    translate_map = str.maketrans(translate_dict)
    print("translate_map = ",translate_map)
    text = text.translate(translate_map)
    print(text)
    seq = text.split(split)
    print(seq)
    print([i for i in seq if i])
    return [i for i in seq if i]
```

```python
rval = text_to_word_sequence('That #book is Amazing!! && incredible%%')
print(rval)
```

```
translate_dict =  {'!': ' ', '"': ' ', '#': ' ', '$': ' ', '%': ' ', '&': ' ', '(': '
translate_map =  {33: ' ', 34: ' ', 35: ' ', 36: ' ', 37: ' ', 38: ' ', 40: ' ', 41:
that  book is amazing      incredible
['that', '', 'book', 'is', 'amazing', '', '', '', '', '', 'incredible', '', '']
['that', 'book', 'is', 'amazing', 'incredible']
['that', 'book', 'is', 'amazing', 'incredible']
```

This is the sample execution of text_to_word_sequence(). We only pass the text parameter which is each string in the flattened list. Initially we convert it to lowercase. translate_dict is a dictionary where keys are the punctuations and values are space. The maketrans() method returns a mapping table that can be used with the translate() method to replace specified characters.

In the mapping table we replace the keys in translate_dict dictionary with their corresponding ASCII values. Using translate function we replace the instances of the key in the text with space. i.e We turn all the punctuators into spaces. Now we split the text having space as the delimiter. In the end we return a list from the split text consisting of words which are of size at least 1.

```
1 tokenizer.word_index
```

```
{'a': 1,
 'ssss': 2,
 'eeee': 3,
 'on': 4,
 'of': 5,
 'the': 6,
 'in': 7,
 'with': 8,
 'and': 9,
 'is': 10,
 'man': 11,
 'to': 12,
 'sitting': 13,
 'an': 14,
 'two': 15,
 'standing': 16,
```

Next passing captions_train_marked which is a list of lists of marked captions to the function captions_to_tokens, we call the function texts_to_sequences() of Tokenizer class for each list in the variable caption_listlist.

```
1 %%time
2 tokens_train = tokenizer.captions_to_tokens(captions_train_marked)
```

tokens_train displays the output of the tokenizer (sequence of integers) .

```
captions_train_marked[0][0]
```

```
'ssss Closeup of bins of food that include broccoli and bread. eeee'
```

```
tokens_train[0][0]
```

```
[2, 844, 5, 2845, 5, 60, 25, 1933, 248, 9, 438, 3]
```

```
tokenizer.word_index
```

```
{'a': 1,
 'ssss': 2,
 'eeee': 3,
 'on': 4,
 'of': 5,
 'the': 6,
 'in': 7,
```

```
def get_random_caption_tokens(idx):
    result = []
    for i in idx:
        j = np.random.choice(len(tokens_train[i]))
        tokens = tokens_train[i][j]
        result.append(tokens)
    return result
```

```
def batch_generator(batch_size):
    while True:
        idx = np.random.randint(num_images_train, size=batch_size)
        transfer_values = transfer_values_train[idx]
        tokens = get_random_caption_tokens(idx)
        num_tokens = [len(t) for t in tokens]
        max_tokens = np.max(num_tokens)
        tokens_padded = pad_sequences(tokens, maxlen=max_tokens, padding='post', truncating='post')
        decoder_input_data = tokens_padded[:, 0:-1]
        decoder_output_data = tokens_padded[:, 1:]
        x_data = { 'decoder_input': decoder_input_data, 'transfer_values_input': transfer_values }
        y_data = { 'decoder_output': decoder_output_data }
        yield (x_data, y_data)
```

```
batch_size = 1024
generator = batch_generator(batch_size=batch_size)
batch = next(generator)
batch_x = batch[0]
batch_y = batch[1]
```

```
batch_x['decoder_input'].shape
```

```
(1024, 42)
```

```
batch_y['decoder_output'].shape
```

```
(1024, 42)
```

```
batch_x['transfer_values_input'].shape
```

```
(1024, 4096)
```

```
batch_x['decoder_input'][0]
```

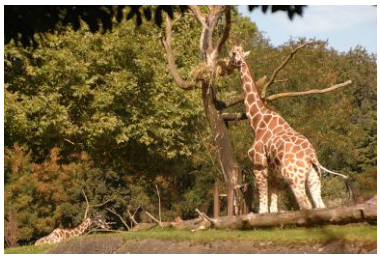
```
array([  2,  15,  18, 593,   1, 901,  48,   1, 334,   3,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0], dtype=int32)
```

```
from tensorflow.keras.layers import Input, Dense, GRU, Embedding

```

```
decoder_embedding = Embedding(input_dim=num_words,
                              output_dim=embedding_size,
                              name='decoder_embedding') (decoder_input)
```

MODULE WISE TEST CASES

MODULE 1: PRE-PROCESSOR		
INPUT	**SEPARATED IMAGE**	**SEPARATED CAPTIONS**
{"license": 5,"file_name": "COCO_train2014_000000057870.jpg","coco_url": "http://images.cocodataset.org/train2014/COCO_train2014_000000000025.jpg","height": 480,"width": 640,"date_captured": "2013-11-14 16:28:13","id": 57870}		['A giraffe eating food from the top of the tree.', 'A giraffe standing up nearby a tree ', 'A giraffe mother with its baby in the forest.', 'Two giraffes standing in a tree filled area.', 'A giraffe standing next to a forest filled with trees.']
{"license": 5,"file_name": "COCO_train2014_000000384029.jpg","coco_url": "http://images.cocodataset.org/val2014/COCO_val2014_000000003014.jpg","height": 429,"width": 640,"date_captured": "2013-11-14 16:29:45","id": 384029}		['Three-quarters of a meat-lovers pizza with mushrooms with drink', 'A partially eaten pizza is sitting beside a soda.', 'A table with a partially eaten pizza and blue canned beverage on it.', 'A close up of a pizza and a drink on a table.', 'A pizza with cherry tomatoes has a piece taken out.']
{"license": 1,"file_name": "COCO_train2014_000000222016.jpg","coco_url": "http://images.cocodataset.org/train2014/COCO_train2014_000000000404.jpg","height": 640,"width": 480,"date_captured": "2013-11-14 16:37:59”,"id": 222016}		['a couple of boats that are in some water', 'A pair of boats docked at a pier is shown.', 'Three boats docked in still water with clouds in the sky. ', 'Three boats are docked together on the cloudy day.', 'some colorful boats sitting next to a dock ']
{"license": 3,"file_name": "COCO_val2014_000000391895.jpg","coco_url": "http://images.cocodataset.org/train2014/COCO_train2014_000000000089.jpg","height": 360,"width": 640,"date_captured": "2013-11-14 11:18:45","id": 391895}		['An oven with a stove on top of it in a kitchen.', 'A stove with a lighted hood in the kitchen.', 'A small light is on above the polished stove top.', 'Smooth top stove with exhaust fan that has light turned on.', 'A stove top is cleaned with a set of knives on the wall.']

{"license": 4,"file_name": "COCO_val2014_000000522418.jpg","coco_url": "http://images.cocodataset.org/val2014/COCO_val2014_000000000073.jpg","height": 480,"width": 640,"date_captured": "2013-11-14 11:38:44","id": 522418}		['A motorcycle parked in a parking space next to another motorcycle.', 'An old motorcycle parked beside other motorcycles with a brown leather seat.', 'Motorcycle parked in the parking lot of asphalt.', 'A close up view of a motorized bicycle, sitting in a rack. ', 'The back tire of an old style motorcycle is resting in a metal stand. ']

MODULE 2: ENCODER	
INPUT	**OUTPUT – TRANSFER VALUES**
	[0. 0. 1.556 ... 0. 0. 0.4502] Dimensions: (4096,)
	[0. 0. 1.378 ... 0. 0. 0.5244] Dimensions: (4096,)

	[0. 0.0756 1.33 ... 0. 0. 0.532] Dimensions: (4096,)

MODULE 3: TOKENIZER			
INPUT	**INTERMEDIATE OUTPUT**	**OUTPUT SEQUENCE**	**EMBEDDINGS**
'Closeup of bins of food that include broccoli and bread.'	'ssss Closeup of bins of food that include broccoli and bread. eeee'	[2, 844, 5, 2845, 5, 60, 25, 1933, 248, 9, 438, 3]	[[0.41177 , -2.223 , -1.0756 , -1.0783 ,]...[0.15155 , 0.78321 , -0.91241]]
'A giraffe eating food from the top of the tree.'	'ssss A giraffe eating food from the top of the tree. eeee'	[2, 1, 117, 108, 60, 96, 6, 32, 5, 6, 133, 3]	[[0.50451 , 0.68607 , -0.59517,]...[0.60046,-0.13498,..-0.022801]]
'White vase with different colored flowers sitting inside of it. '	'ssss White vase with different colored flowers sitting inside of it. eeee'	[2, 21, 202, 8, 191, 395, 200, 13, 159, 5, 30, 3]	[[0.32157 , 0.45894 , -0.32014,]...[0.32015,-0.23156,..-0.30215]]
‘A cat is lying on its back in a man's lap’	ssss A cat is lying on its back in a man's lap. eeee	[2, 1, 50, 10, 370, 4, 154, 163, 7, 1, 1248, 584, 3]	[[0.09981 , 0.45477 , -0.87890,]...[0.12344,-0.98757,..-0.00854]]
‘a giraffe standing in tall grass with trees in the background .’	ssss a giraffe standing in tall grass with trees in the background eeee	[2, 6, 117, 10, 16, 683, 7, 6, 1613, 3]	[[0.54687 , 0.63547 , -0.77894,]...[0.30210,-0.65489,..-0.32154]]