DL Assignment 2 - Image Captioning - Set 9 : Encoder - Inception ; Decoder - GRU

Submitted by - GROUP 29

AMITABH GUPTA (2019ab04170), DEBORIMA SENAPATI(2019ab04184) , PRIYADARSHI MISHRA (2019ab04133) ----** **bold text**

Prepare python notebook to build, train and evaluate model.

Question:Image Captioning is the process of generating textual description of an image. It uses both Natural Language Processing and Computer Vision to generate the captions. The dataset will be in the form [image → captions]. The dataset consists of input images and their corresponding output captions.

1 : Import Libraries/Dataset

a. Import the required libraries

```
# import tensorflow as tf
# import keras
# print(tf.__version__)
# print(keras.__version__)


import string
import numpy as np
import PIL.Image
import pickle
import matplotlib.pyplot as plt
import os
import pandas as pd
import numpy as np
from collections import Counter
import numpy as np
from numpy import array
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import string
import os
from PIL import Image
import glob
from pickle import dump, load
from time import time
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.optimizers import Adam, RMSprop
from keras.layers.wrappers import Bidirectional
from keras.layers.merge import add
from keras.applications.inception_v3 import InceptionV3
from keras.preprocessing import image
from keras.models import Model
from keras import Input, layers
from keras import optimizers
from keras.applications.inception_v3 import preprocess_input
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import pickle
from tqdm import tqdm
import os
from tensorflow.keras.optimizers import Adam, RMSprop
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.preprocessing import image
from tensorflow.keras.layers import LSTM, GRU,Embedding, Input,Dense,Activation, Flatten, RepeatVector,TimeDistributed
from os import listdir
from pickle import dump, load
from numpy import array
from numpy import argmax
#from keras.applications.inception_v3  import inception_v3 , preprocess_input
from keras.preprocessing.image import load_img, img_to_array
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
```

```
#from keras.utils import to_categorical
from keras.layers.merge import add
from keras.models import Model, load_model
from keras.layers import Input, Dense, GRU, Embedding, Dropout
from keras.callbacks import ModelCheckpoint
from nltk.translate.bleu_score import corpus_bleu
from tqdm import tqdm_notebook as tqdm
```

b. Check the GPU available (recommended- use free GPU provided by Google Colab).

```
from google.colab import drive
drive.mount('/content/drive')
```

```
     Mounted at /content/drive
```

```
#Load Data
url = '/content/drive/MyDrive/Colab Notebooks/set_3.pkl'
with open(url, 'rb') as f:
  caption_text = pickle.load(f)

datatxt = []
for text in caption_text :
  for line in text.split('\n'):
    col = line.split('\t')
    if len(col) == 1:
        continue
    w = col[0].split("#")
    datatxt.append(w + [col[1].lower()])

df_txt = pd.DataFrame(datatxt,columns=["filename","index","caption"])
np.savetxt(r'/content/drive/MyDrive/Colab Notebooks/set_3.txt', df_txt.values, fmt='%s')
uni_filenames = np.unique(df_txt.filename.values)
print("The number of unique file names : {}".format(len(uni_filenames)))
print("The distribution of the number of captions for each image:")
Counter(Counter(df_txt.filename.values).values())
```

```
     The number of unique file names : 8017
     The distribution of the number of captions for each image:
     Counter({1: 522, 2: 1700, 3: 2854, 4: 2189, 5: 752})
```

```
### load the images
dir_jpg = '/content/drive/MyDrive/Colab Notebooks/Flicker8k_Dataset'
jpgs = os.listdir(dir_jpg)
#print(jpgs)
# Below path contains all the images
#images = '../../storage/image_caption/dataset/Flicker8k_Dataset/'
# Create a list of all image names in the directory
img = glob.glob('/content/drive/MyDrive/Colab Notebooks/Flicker8k_Dataset/' + '*.jpg')
print (img)
print("The number of jpg flies in Flicker8k_Dataset: {}".format(len(jpgs)))
```

```
     ['/content/drive/MyDrive/Colab Notebooks/Flicker8k_Dataset/382090166_be2c2c63e1.jpg', '/content/drive/MyDrive/Colab Notebooks/F
     The number of jpg flies in Flicker8k_Dataset: 8092
```

2: Data Visualization and augmentation

```
from keras.preprocessing.image import load_img, img_to_array

npic = 5
npix = 224
target_size = (npix,npix,3)

count = 1
fig = plt.figure(figsize=(10,20))
for jpgfnm in uni_filenames[-npic:]:
    filename = dir_jpg + '/' + jpgfnm
    captions = list(df_txt["caption"].loc[df_txt["filename"]==jpgfnm].values)
    image_load = load_img(filename, target_size=target_size)

    ax = fig.add_subplot(npic,2,count,xticks=[],yticks=[])
    ax.imshow(image_load)
    count += 1

    ax = fig.add_subplot(npic,2,count)
```

```
    plt.axis('off')
    ax.plot()
    ax.set_xlim(0,1)
    ax.set_ylim(0,len(captions))
    for i, caption in enumerate(captions):
        ax.text(0,i,caption,fontsize=20)
    count += 1
plt.show()
```

man on a bicycle ride on only one wheel .

a man do a wheelie on his bicycle on a sidewalk .

a man on a bicycle be on only the back wheel .

a man be do a wheelie on a mountain bike .

a group of person sit in the snow overlook a mountain scene .

a group be sit around a snowy crevasse .

five person be sit together in the snow .

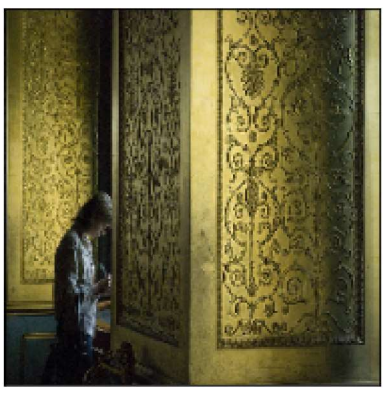
a group of person sit atop a snowy mountain .

a large bird stand in the water on a beach .

a water bird stand at the ocean 's edge .

a gray bird stand majestically on a beach while wave roll in .

a white crane stand tall as it look out upon the ocean .



a woman stand near a decorate wall write .

wall be cover in gold and pattern .

woman writing on a pad in room with gold , decorate wall .

a rock climber practice on a rock climb wall .

a rock climber in a red shirt .

a man in a pink shirt climb a rock face

a man be rock climb high in the air .

3 - **Bringing the train and test data in the required format & Model Building**

Image Preparation

Extract Freatures of each image using Transfer Learning Inception Model

```
from keras.models import Model
from keras.utils.np_utils import to_categorical
```

```
from keras.utils.np_utils import to_categorical
# Extract features from each photo in the directory
def extract_features(directory):
    # Loading the model
    inception_model = tf.keras.applications.InceptionV3(weights='imagenet')
    # Removing the last layer from the loaded model as we require only the features not the classification
    inception_model.layers.pop()
    inception_model = Model(inputs=inception_model.inputs, outputs=inception_model.layers[-1].output)
    # Summarizing the model
    print(inception_model.summary())
    # Extracting features from each photo and storing it in a dictionary
    features = dict()
    for name in tqdm(listdir(directory)):
        # Defining the path of the image
        filename = directory + '/' + name
        # Loading an image and converting it into size 299 * 299
        image = load_img(filename, target_size=(299, 299)) #for inception
        # Converting the image pixels into a numpy array
        image = img_to_array(image)
        # Reshaping data for the model
        image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))

        # Preprocessing the images for the Inception Model
        image = preprocess_input(image)

        # Getting features of an image
        feature = inception_model.predict(image, verbose=0)

        # Getting the image name
        image_id = name.split('.')[0]

        # Storing the feature corresponding to the image in the dictionary
        features[image_id] = feature

        # print('>%s' % name)

    return features


# Defining the directory we are using
directory = '/content/drive/MyDrive/Colab Notebooks/Flicker8k_Dataset'

# Extracting features from all the images
features = extract_features(directory)
print('Extracted Features: ', len(features))
# Dumping the features in a pickle file for further use
dump(features, open('features.pkl', 'wb'))
```

```
Model: "model_11"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
Total params: 23,851,784
Trainable params: 23,817,352
Non-trainable params: 34,432
__________________________________________________________________________________________________
None
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:15: TqdmDeprecationWarning: This function will be removed in tqdm=
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  from ipykernel import kernelapp as app
100%                                      8092/8092 [27:00<00:00, 4.99it/s]


Extracted Features:  8092
```

3 (II) . Preparing Captions Data

```
# Loading the file containg all the descriptions into memory

def load_doc(filename):
    # Opening the file as read only
    file = open(filename, 'r')
    # Reading all text and storing it.
    text = file.read()
    # Closing the file
    file.close()
    return text
```

```
def photo_to_description_mapping(descriptions):

    # Dictionary to store the mapping of photo identifiers to descriptions
    description_mapping = dict()

    # Iterating through each line of the descriptions
    for line in descriptions.split('\n'):

        # Splitting the lines by white space
        words = line.split()

        # Skipping the lines with length less than 2
        if len(line)<2:
            continue

        # The first word is the image_id and the rest are the part of the description of that image
        image_id, image_description = words[0], words[1:]

        # Retaining only the name of the image and removing the extension from it
        image_id = image_id.split('.')[0]

        # Image_descriptions contains comma separated words of the description, hence, converting it back to string
        image_description = ' '.join(image_description)

        # There are multiple descriptions per image,
        # hence, corresponding to every image identifier in the dictionary, there is a list of description
        # if the list does not exist then we need to create it

        if image_id not in description_mapping:
            description_mapping[image_id] = list()

        # Now storing the descriptions in the mapping
        description_mapping[image_id].append(image_description)

    return description_mapping


def clean_descriptions(description_mapping):

    # Preapring a translation table for removing all the punctuation
    table = str.maketrans('','', string.punctuation)

    # Traversing through the mapping we created
    for key, descriptions in description_mapping.items():
        for i in range(len(descriptions)):
            description = descriptions[i]
            description = description.split()

            # Converting all the words to lower case
            description = [word.lower() for word in description]

            # Removing the punctuation using the translation table we made
            description = [word.translate(table) for word in description]

            # Removing the words with length =1
            description = [word for word in description if len(word)>1]

            # Removing all words with number in them
            description = [word for word in description if word.isalpha()]

            # Converting the description back to string and overwriting in the descriptions list
            descriptions[i] = ' '.join(description)


# Converting the loaded descriptions into a vocabulary of words

def to_vocabulary(descriptions):

    # Build a list of all description strings
    all_desc = set()

    for key in descriptions.keys():
        [all_desc.update(d.split()) for d in descriptions[key]]

    return all_desc


# save descriptions to file, one per line
def save_descriptions(descriptions, filename):
    lines = list()
```

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


#converted .pkl to .txt for smooth processing
filename = '/content/drive/MyDrive/Colab Notebooks/set_3.txt'
# Loading descriptions
doc = load_doc(filename)

# Parsing descriptions
descriptions = photo_to_description_mapping(doc)
print('Loaded: %d ' % len(descriptions))

# Cleaning the descriptions
clean_descriptions(descriptions)

# Summarizing the vocabulary
vocabulary = to_vocabulary(descriptions)
print('Vocabulary Size: %d' % len(vocabulary))

# Saving to the file
save_descriptions(descriptions, 'descriptions.txt')
```

```
Loaded: 8017
Vocabulary Size: 5452
```

Exploratory Analysis

Cleaning captions for further analysis

A.) Loading the data

```
# Function for loading a file into memory and returning text from it
def load_file(filename):
    file = open(filename, 'r')
    text = file.read()
    file.close()
    return text

# Function for loading a pre-defined list of photo identifiers
def load_photo_identifiers(filename):

    # Loading the file containing the list of photo identifier
    file = load_file(filename)

    # Creating a list for storing the identifiers
    photos = list()

    # Traversing the file one line at a time
    for line in file.split('\n'):
        if len(line) < 1:
            continue

        # Image name contains the extension as well but we need just the name
        identifier = line.split('.')[0]

        # Adding it to the list of photos
        photos.append(identifier)

    # Returning the set of photos created
    return set(photos)

def load_clean_descriptions(filename, photos):

    #loading the cleaned description file
    file = load_file(filename)

    #creating a dictionary of descripitions for storing the photo to description mapping of train images
    descriptions = dict()
```

```
    #traversing the file line by line
    for line in file.split('\n'):
        # splitting the line at white spaces
        words = line.split()

        # the first word will be the image name and the rest will be the description of that particular image
        image_id, image_description = words[0], words[1:]

        # we want to load only those description which corresponds to the set of photos we provided as argument
        if image_id in photos:
            #creating list of description if needed
            if image_id not in descriptions:
                descriptions[image_id] = list()


            desc = 'startseq ' + ' '.join(image_description) + ' endseq'
            descriptions[image_id].append(desc)

    return descriptions

# function to load the photo features created using the VGG16 model
def load_photo_features(filename, photos):

    #this will load the entire features
    all_features = load(open(filename, 'rb'))

    #we are interested in loading the features of the required photos only
    features = {k: all_features[k] for k in photos}

    return features
```

Splitting data

```
from sklearn.model_selection import train_test_split
token=[]
for i in uni_filenames:
     i=i.split(".jpg")
     token.append(i)

print(len(token))
token_1=[]
for k in range(len(token)):
     token_1.append(token[k][0])

print(len(token_1))
train_ratio = 0.75
validation_ratio = 0.125
test_ratio = 0.125
```

```
x_train, x_test = train_test_split(token_1,test_size=1 - train_ratio)

# # test is now 10% of the initial data set
# # validation is now 15% of the initial data set
x_val, x_test = train_test_split(x_test, test_size=test_ratio/(test_ratio + validation_ratio))

print("Training data length:",len(x_train))
print("validation data length:",len( x_val))
print("Testing data length:",len(x_test))

train = x_train
print('Dataset: ',len(train))

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

train_features = load_photo_features('features.pkl', train)
print('Photos: train=', len(train_features))
```

```
8017
8017
Training data length: 6012
validation data length: 1002
Testing data length: 1003
Dataset:  6012
Descriptions: train= 6012
Photos: train= 6012
```

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

# Given the descriptions, fit a tokenizer

# TOKENIZER CLASS:
# This class allows to vectorize a text corpus,
def create_tokenizer(descriptions):
    lines = to_lines(descriptions)
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(lines)
    return tokenizer


tokenizer = create_tokenizer(train_descriptions)

vocab_size = len(tokenizer.word_index) + 1
print('Vocabulary Size: ', vocab_size)
```

```
 Vocabulary Size:  4752
```

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

4. Model Compilation

a. Compile the model with the appropriate loss function.

Categorical crossentropy is a loss function that is used in multi-class classification tasks. These are tasks where an example can only belong to one out of many possible categories, and the model must decide which one.The categorical crossentropy is well suited to image classification tasks, since one image can be considered to belong to a specific category with probability 1, and to other categories with probability 0.Based on category captioning is done

b. Use an appropriate optimizer. Give reasons for the choice of learning rate and its value.

We have chosen "Adam" which belongs to the family of Adaptive optimizers. This family of optimizers solve the issues of the gradient descent's algorithms. Their most important feature is that they don't require a tuning of the learning rate value.It adapts the learning rate to the parameters performing small updates for frequently occurring features and large updates for the rarest ones.In this way, the network is able to capture information belonging to features that are not frequent, putting them in evidence and giving them the right weight.

Training & Test dataset prepration for model learning

```
from keras.utils.np_utils import to_categorical

#data generator, intended to be used in a call to model.fit_generator()
def data_generator(descriptions, photos, tokenizer, max_length):
    while 1:
        for key, description_list in descriptions.items():
            #retrieve photo features
            photo = photos[key][0]
            input_image, input_sequence, output_word = create_sequences(tokenizer, max_length, description_list, photo)
            yield [[input_image, input_sequence], output_word]

def create_sequences(tokenizer, max_length, desc_list, photo):
    X1, X2, y = list(), list(), list()
    # 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(photo)
            X2.append(in_seq)
            y.append(out_seq)
    return array(X1), array(X2), array(y)


# Function to encode a given image into a vector of size (2048, )
def encode(image):
    image = preprocess(image) # preprocess the image
    fea_vec = model_new.predict(image) # Get the encoding vector for the image
    fea_vec = np.reshape(fea_vec, fea_vec.shape[1]) # reshape from (1, 2048) to (2048, )
    return fea_vec


# data generator, intended to be used in a call to model.fit_generator()
def data_generator1(descriptions, photos, tokenizer, max_length, num_photos_per_batch):
    X1, X2, y = list(), list(), list()
    n=0

    # loop for ever over images
    while 1:
        for key, desc_list in descriptions.items():
            n+=1

#             print("Key:", key)
#             print("desc list: ", desc_list)
            # retrieve the photo feature
            photo = photos[key+'.jpg']
            for desc in desc_list:
                # encode the sequence
                seq = [wordtoix[word] for word in desc.split(' ') if word in wordtoix]
                # split one sequence into multiple X, y pairs
                for i in range(1, len(seq)):
                    # split into input and output pair
                    in_seq, out_seq = seq[:i], seq[i]
                    # pad input sequence
                    in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                    # encode output sequence
                    out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
                    # store
                    X1.append(photo)
                    X2.append(in_seq)
                    y.append(out_seq)
            # yield the batch data
            if n==num_photos_per_batch:
                yield ([array(X1), array(X2)], array(y))
                X1, X2, y = list(), list(), list()
                n=0
```

We are choosing Adam Optimizer & 3 GRY layer with L2 regularization.

Adam combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm. It can handle sparse gradients on noisy problems. This algorithm is used to accelerate the gradient descent algorithm by taking into consideration the 'exponentially weighted average' of the gradients.

Adam is and adaptive optimizer, and it learns the learning rates itself, on a per-parameter basis. So we are not specifying learning rate

For Loss we have choosen categorical_crossentropy as it is the best choice for the Multiclasss classificiation.

```
from keras.utils.vis_utils import plot_model
from keras.utils.np_utils import to_categorical
from keras.regularizers import l2
# Model Working Version

from tensorflow.keras import regularizers
# define the captioning model
def define_model(vocab_size, max_length):
    # feature extractor model
    inputs1 = Input(shape=(2048,)) #for inception
    fe1 = Dropout(0.4)(inputs1)
    fe2 = Dense(256, activation='relu')(fe1)

    # sequence model
    inputs2 = Input(shape=(max_length,))
    se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2) #Masking and padding in Keras reshape the variable length input sequenc
    se2 = Dropout(0.5)(se1)
#     se3 = LSTM(512)(se2)
```

```
se3 = GRU(256, kernel_regularizer = regularizers.l2(0.01), return_sequences=True)(se2)
se4 = GRU(256, kernel_regularizer = regularizers.l2(0.01), return_sequences=True)(se3)
se5 = GRU(256, kernel_regularizer = regularizers.l2(0.01))(se4)

# decoder model
decoder1 = layers.Concatenate()([fe2, se5])
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',metrics=['accuracy'])

# summarize model
print(model.summary())
plot_model(model, to_file='model.png', show_shapes=True,dpi=1000)

return model
```

5. Model Training

a. Train the model for an appropriate number of epochs. Print the train and validation loss for each epoch. Use the appropriate batch size.

```
train = x_train
print('Dataset: ', len(train))
train_descriptions = load_clean_descriptions('descriptions.txt', train)
print('Descriptions: train=', len(train_descriptions))
train_features = load_photo_features('features.pkl', train)
print('Photos: train=', len(train_features))
tokenizer = create_tokenizer(train_descriptions)
vocab_size = len(tokenizer.word_index) + 1
print('Vocabulary Size:', vocab_size)
max_length = max_lengthTEMP(train_descriptions)
print('Description Length: ', max_length)
model = define_model(vocab_size, max_length)
```

```
Dataset:  6012
Descriptions: train= 6012
Photos: train= 6012
Vocabulary Size: 4752
Description Length:  31
Model: "model_25"
```

Layer (type)	Output Shape	Param #	Connected to
input_36 (InputLayer)	[(None, 31)]	0	
embedding_17 (Embedding)	(None, 31, 256)	1216512	input_36[0][0]
dropout_35 (Dropout)	(None, 31, 256)	0	embedding_17[0][0]
input_35 (InputLayer)	[(None, 2048)]	0	
gru_30 (GRU)	(None, 31, 256)	394752	dropout_35[0][0]
dropout_34 (Dropout)	(None, 2048)	0	input_35[0][0]
gru_31 (GRU)	(None, 31, 256)	394752	gru_30[0][0]
dense_41 (Dense)	(None, 256)	524544	dropout_34[0][0]
gru_32 (GRU)	(None, 256)	394752	gru_31[0][0]
concatenate_7 (Concatenate)	(None, 512)	0	dense_41[0][0] gru_32[0][0]
dense_42 (Dense)	(None, 256)	131328	concatenate_7[0][0]
dense_43 (Dense)	(None, 4752)	1221264	dense_42[0][0]

```
Total params: 4,277,904
Trainable params: 4,277,904
Non-trainable params: 0
```

```
None
```

Created 3 layered GRU layer model and other relevant layers for image caption generation. a. Create 3 layered GRU layer model and other relevant layers for image caption generation.

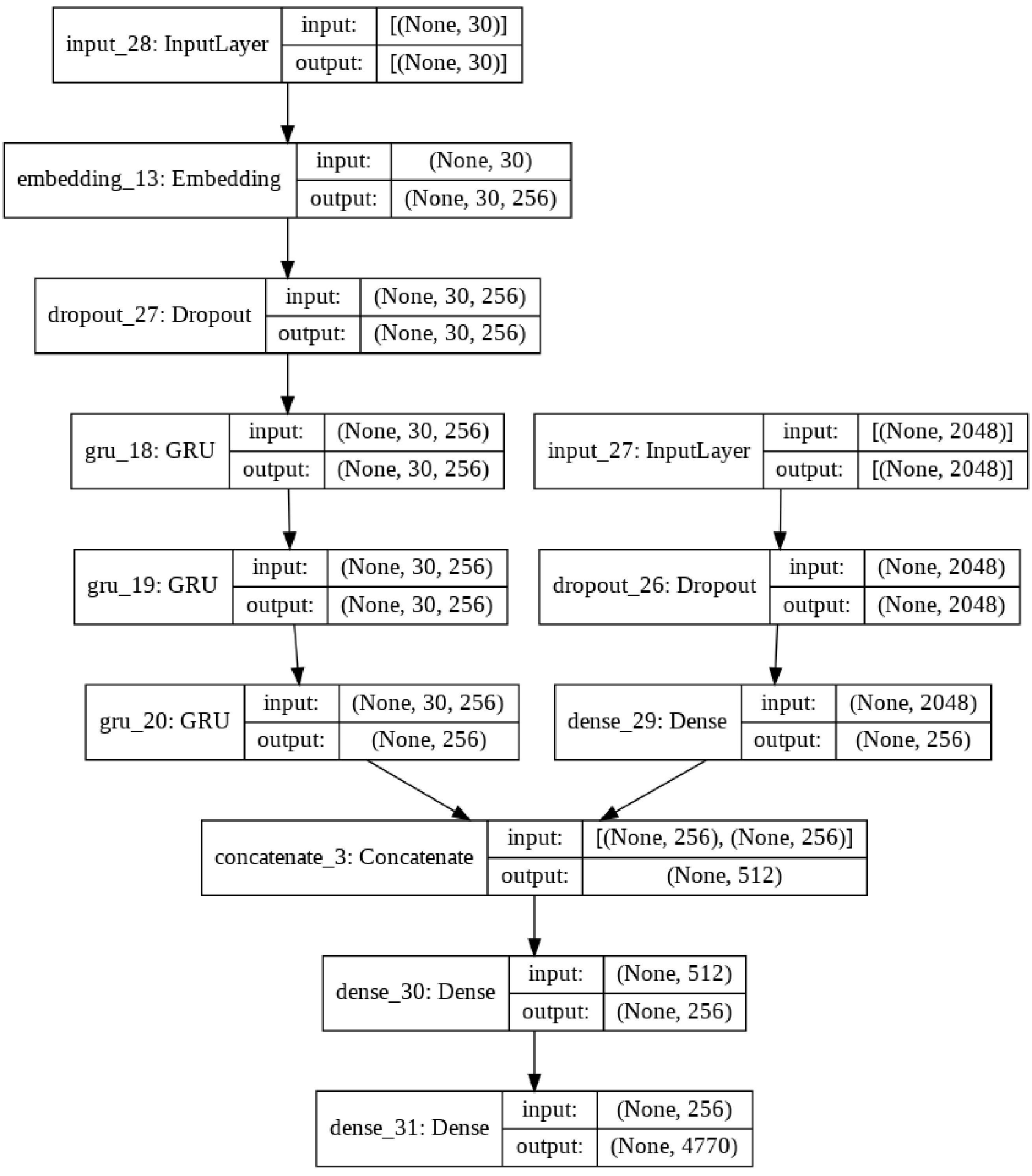
b. Add L2 regularization to all the GRU layers.

c. Add one layer of dropout at the appropriate position and give reasons.

d. Choose the appropriate activation function for all the layers.

e. Print the model summary.

```
from tensorflow.keras.utils import plot_model
plot_model(model, to_file='model.png', show_shapes=True)
```




```
epochs = 10
number_pics_per_bath = 16
steps = len(train_descriptions)//number_pics_per_bath


len(train_descriptions), len(train_features)
train_data_with_jpg = list(set([i+'.jpg' for i in train_descriptions.keys()]).intersection(set(train_features.keys())))
train_data_without_jpg = list(set(train_descriptions.keys()).intersection(set([i.split(".")[0] for i in train_features.keys()])))

train_descriptions_new = {i:train_descriptions[i] for i in train_data_without_jpg}
train_features_new = {i:train_features[i] for i in train_data_with_jpg}
```

```
len(train_descriptions_new), len(train_features)

# type(train_descriptions), type(train_features)
```

```
(6012, 6012)
```

```
import tensorflow as tf
tf.config.run_functions_eagerly(True)

loss_list = []

for i in range(epochs):
    generator = data_generator(train_descriptions_new, train_features_new, wordtoix, max_length, number_pics_per_bath)
    history = model.fit_generator(generator, epochs=1, steps_per_epoch=steps, verbose=1)
    loss_list.append(history.history['loss'])
    model.save('./model_weights/model_' + str(i) + '.h5')
```

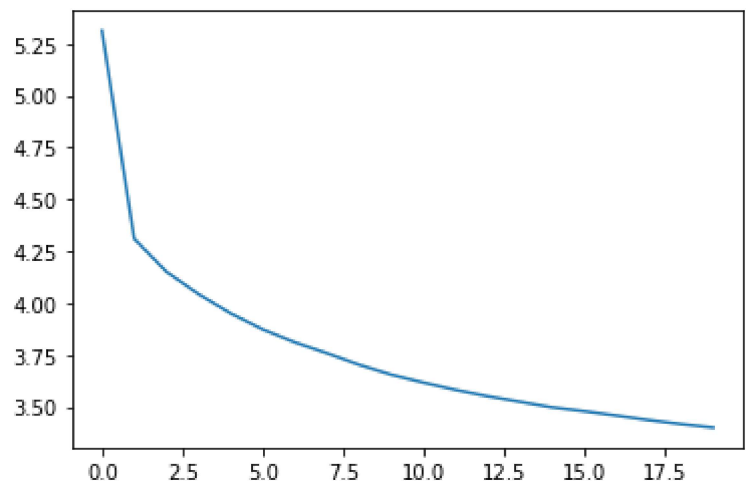
```
/opt/conda/lib/python3.7/site-packages/tensorflow/python/keras/engine/training.py:1844: UserWarning: `Model.fit_generator` is d
  warnings.warn('`Model.fit_generator` is deprecated and '
/opt/conda/lib/python3.7/site-packages/tensorflow/python/data/ops/dataset_ops.py:3504: UserWarning: Even though the tf.config.e
  "Even though the tf.config.experimental_run_functions_eagerly "
376/376 [==============================] - 145s 385ms/step - loss: 7.1528
376/376 [==============================] - 144s 384ms/step - loss: 4.3106
376/376 [==============================] - 146s 389ms/step - loss: 4.1516
376/376 [==============================] - 147s 390ms/step - loss: 4.0434
376/376 [==============================] - 144s 382ms/step - loss: 3.9506
376/376 [==============================] - 145s 385ms/step - loss: 3.8728
376/376 [==============================] - 144s 384ms/step - loss: 3.8105
376/376 [==============================] - 141s 376ms/step - loss: 3.7579
376/376 [==============================] - 142s 379ms/step - loss: 3.7023
376/376 [==============================] - 142s 378ms/step - loss: 3.6547
```

```
loss_list
```

```
[[5.313275337219238],
 [4.310606479644775],
 [4.151629447937012],
 [4.043374538421631],
 [3.9505598545074463],
 [3.8728158473968506],
 [3.810487985610962],
 [3.757880926132202],
 [3.7022626399993896],
 [3.654745578765869],
 [3.616623640060425],
 [3.581441879272461],
 [3.5507917404174805],
 [3.5241613388061523],
 [3.4973294734954834],
 [3.478621482849121],
 [3.4576241970062256],
 [3.436070680618286],
 [3.4167110919952393],
 [3.3999197483062744]]
```

```
# Plotting loss
plt.plot(loss_list)
```

```
[<matplotlib.lines.Line2D at 0x7f7c06716410>]
```



6. Model Evaluation (1 mark)

```
# From Google Image
```

```
import time

source = "https://images-na.ssl-images-amazon.com/images/I/819Tn9I43ZL.png"
file_name = source.split("/")[-1]
show_img = source.split("/")[-1]
!wget $source
time.sleep(2)
file_name = extract_features(file_name).reshape((1,2048))

x = plt.imread(show_img)
plt.imshow(x)
plt.show()

greedySearch(file_name)
```

✓ 11s completed at 4:16 AM