Image Caption Model

# Image Caption Model[¶](#Image-Caption-Model)

Algoritmul de descriere a imaginilor poate fi vazut ca o rețea de tipul encoder-decoder. Reteaua convolutionala codeaza imaginea intr-un format pe care reteaua recurenta il va putea folosi pentru a genera o descriere textuala.

#### Encoder[¶](#Encoder)

Vom folosi un model de VGG16 preantrenat pentru clasificarea imaginilor. Vom elimina unltimul layer de clasificare pentru a redirectiona iesirea stratului anterior. Stratul anterior are 4096 de neuroni, ale caror iesiri vor reprezenta un vector de 4096 de elemente ce descriu continutul imaginii. Acest vector va servi ca intrarea initiala a retelei neuronale recurente (Gated Recurrent Units sau GRU). Starea interna a retelei recurente are 512 elemente asa ca vectorul nostru va trebui redus la aceasta dimensiune cu ajutorul unui strat de 512 neuroni conectat complet la stratul anterior.

#### Decoder[¶](#Decoder)

Decodorul foloseste starea initiala impreauna cu un marker de start "ssss" pentru a produce cuvintele ce descriu imaginea. In prima iteratie speram ca va da la iesire cuvantul "big". Imaginea este apoi servita din nou la intrarea retelei de data aceasta cu cuvintele obtinute in iteratiile anterioare pentru a putea obtine o propozitie. În final vom obține "big brown bear sitting eeee", unde "eeee" marchează finalul propozitiei.

O schemă a algoritmului arată in felul următor:

(de moficat aceasta poza)

Flowchart

## Importuri[¶](#Importuri)

In [2]:

%matplotlib inline  
import matplotlib.pyplot as plt  
import tensorflow as tf  
import numpy as np  
import sys  
import os  
from PIL import Image  
from cache import cache  
from tensorflow.keras import backend as K  
from tensorflow.keras.models import Model  
from tensorflow.keras.layers import Input, Dense, GRU, Embedding  
from tensorflow.keras.applications import VGG16  
from tensorflow.keras.optimizers import RMSprop,Adam  
  
from tensorflow.keras.callbacks import ModelCheckpoint, TensorBoard  
from tensorflow.keras.preprocessing.text import Tokenizer  
from tensorflow.keras.preprocessing.sequence import pad\_sequences

Am folosit Python 3.6 (Anaconda) and si urmatoarele versiuni de tensorflow si keras:

In [3]:

tf.\_\_version\_\_

Out[3]:

'2.2.0'

In [4]:

tf.keras.\_\_version\_\_

Out[4]:

'2.3.0-tf'

### Importul bazei de date[¶](#Importul-bazei-de-date)

In [5]:

import coco

#### Descarcarea bazei de date[¶](#Descarcarea-bazei-de-date)

In [6]:

coco.maybe\_download\_and\_extract()

Downloading http://images.cocodataset.org/zips/train2017.zip  
Data has apparently already been downloaded and unpacked.  
Downloading http://images.cocodataset.org/zips/val2017.zip  
Data has apparently already been downloaded and unpacked.  
Downloading http://images.cocodataset.org/annotations/annotations\_trainval2017.zip  
Data has apparently already been downloaded and unpacked.

Obtinerea numelor imaginilor si a descrierilor din setul de antrenare

In [7]:

\_, filenames\_train, captions\_train = coco.load\_records(train=True)

- Data loaded from cache-file: data/coco/records\_train.pkl

Numarul de imagini din setul de antrenare.

In [8]:

num\_images\_train = len(filenames\_train)  
num\_images\_train

Out[8]:

118287

Obtinerea numelor fisierelor si a descrierilor pentru imaginile din setul de validare

In [74]:

\_, filenames\_val, captions\_val = coco.load\_records(train=False)

- Data loaded from cache-file: data/coco/records\_val.pkl

In [75]:

num\_images\_val = len(filenames\_val)  
num\_images\_val

Out[75]:

5000

### Functii ajutatoare[¶](#Functii-ajutatoare)

Incarcarea si redimensionarea imaginilor

In [10]:

def load\_image(path, size=None):  
 """  
 Load the image from the given file-path and resize it  
 to the given size if not None.  
 """  
  
 # Load the image using PIL.  
 img = Image.open(path)  
  
 # Resize image if desired.  
 if not size is None:  
 img = img.resize(size=size, resample=Image.LANCZOS)  
  
 # Convert image to numpy array.  
 img = np.array(img)  
  
 # Scale image-pixels so they fall between 0.0 and 1.0  
 img = img / 255.0  
  
 # Convert 2-dim gray-scale array to 3-dim RGB array.  
 if (len(img.shape) == 2):  
 img = np.repeat(img[:, :, np.newaxis], 3, axis=2)  
  
 return img

Afisarea imaginii si descrierilor aferente

In [11]:

def show\_image(idx, train):  
 """  
 Load and plot an image from the training- or validation-set  
 with the given index.  
 """  
  
 if train:  
 # Use an image from the training-set.  
 dir = coco.train\_dir  
 filename = filenames\_train[idx]  
 captions = captions\_train[idx]  
 else:  
 # Use an image from the validation-set.  
 dir = coco.val\_dir  
 filename = filenames\_val[idx]  
 captions = captions\_val[idx]  
  
 # Path for the image-file.  
 path = os.path.join(dir, filename)  
  
 # Print the captions for this image.  
 for caption in captions:  
 print(caption)  
   
 # Load the image and plot it.  
 img = load\_image(path)  
 plt.imshow(img)  
 plt.show()

### Example[¶](#Example)

In [12]:

show\_image(idx=1, train=True)  
show\_image(idx=11, train=True)  
show\_image(idx=111, train=True)

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.

## Pre-Trained Image Model (VGG16)[¶](#Pre-Trained-Image-Model-(VGG16))

Vom crea o instanta de VGG16 folosind API-ul de Keras. Modelul a fost deja preantrenat pe setul de date ImageNet. VGG16 contine un segment de straturi convolutionale urmata de staturi fully-connected, folosite pentru clasificarea imaginilor.

include\_top=True va descarca intreaga retea ce are 528 de MB. Avem nevoie de intreaga retea deoarce vom folosi si o parte din straturile interconectate pentru descrierea imaginilor.

In [13]:

image\_model = VGG16(include\_top=True, weights='imagenet')

Print a list of all the layers in the VGG16 model.

In [14]:

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  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Folosim penultimul layer de clasificare, numit fc2. Acesta e un strat dens/complet conecatat.

In [15]:

transfer\_layer = image\_model.get\_layer('fc2')

Il numim "transfer-layer" deoarece va transfera informatiile venite de la reteaua convolutionala.

Cream un nou model ce are aceeasi intrare ca reteaua de mai sus insă iesirea este penultimul strat

In [16]:

image\_model\_transfer = Model(inputs=image\_model.input,  
 outputs=transfer\_layer.output)

Modelul se asteapta ca imaginile sa aiba urmatoarea dimensiune

In [17]:

img\_size = K.int\_shape(image\_model.input)[1:3]  
img\_size

Out[17]:

(224, 224)

Pentru fiecare imagine modelul va obtine un vector de 4096 valori

In [18]:

transfer\_values\_size = K.int\_shape(transfer\_layer.output)[1]  
transfer\_values\_size

Out[18]:

4096

### Procesarea tuturor imaginilor[¶](#Procesarea-tuturor-imaginilor)

Vom crea functii ce au rol sa preproceseze imaginile din seturile de date folosind modelul preantrenat si salvand valorile intr-un fisier pentru a putea fi reincarcate rapid.

Obtinem astfel un nou set de date de intrare cu valorile preprocesate. Nu vom schimba toți parametrii modelului VGG16. Avem nevoie de valorile de transfer pentru a instrui modelul de subtitrare a imaginii pentru mai multe epoci, astfel încât economisim mult timp calculând valorile de transfer o dată și salvându-le într-un fișier \*.pkl in prealabil.

Aceasta este o funcție de ajutor pentru afisarea progresului.

In [19]:

def print\_progress(count, max\_count):  
 # Percentage completion.  
 pct\_complete = count / max\_count  
  
 # Status-message. Note the \r which means the line should  
 # overwrite itself.  
 msg = "\r- Progress: {0:.1%}".format(pct\_complete)  
  
 # Print it.  
 sys.stdout.write(msg)  
 sys.stdout.flush()

Aceasta este funcția pentru procesarea fișierelor date folosind modelul VGG16 și returnarea valorilor de transfer ale acestora.

In [20]:

def process\_images(data\_dir, filenames, batch\_size=32):  
 """  
 Process all the given files in the given data\_dir using the  
 pre-trained image-model and return their transfer-values.  
   
 Note that we process the images in batches to save  
 memory and improve efficiency on the GPU.  
 """  
   
 # Number of images to process.  
 num\_images = len(filenames)  
  
 # Pre-allocate input-batch-array for images.  
 shape = (batch\_size,) + img\_size + (3,)  
 image\_batch = np.zeros(shape=shape, dtype=np.float16)  
  
 # Pre-allocate output-array for transfer-values.  
 # Note that we use 16-bit floating-points to save memory.  
 shape = (num\_images, transfer\_values\_size)  
 transfer\_values = np.zeros(shape=shape, dtype=np.float16)  
  
 # Initialize index into the filenames.  
 start\_index = 0  
  
 # Process batches of image-files.  
 while start\_index < num\_images:  
 # Print the percentage-progress.  
 print\_progress(count=start\_index, max\_count=num\_images)  
  
 # End-index for this batch.  
 end\_index = start\_index + batch\_size  
  
 # Ensure end-index is within bounds.  
 if end\_index > num\_images:  
 end\_index = num\_images  
  
 # The last batch may have a different batch-size.  
 current\_batch\_size = end\_index - start\_index  
  
 # Load all the images in the batch.  
 for i, filename in enumerate(filenames[start\_index:end\_index]):  
 # Path for the image-file.  
 path = os.path.join(data\_dir, filename)  
  
 # Load and resize the image.  
 # This returns the image as a numpy-array.  
 img = load\_image(path, size=img\_size)  
  
 # Save the image for later use.  
 image\_batch[i] = img  
  
 # Use the pre-trained image-model to process the image.  
 # Note that the last batch may have a different size,  
 # so we only use the relevant images.  
 transfer\_values\_batch = \  
 image\_model\_transfer.predict(image\_batch[0:current\_batch\_size])  
  
 # Save the transfer-values in the pre-allocated array.  
 transfer\_values[start\_index:end\_index] = \  
 transfer\_values\_batch[0:current\_batch\_size]  
  
 # Increase the index for the next loop-iteration.  
 start\_index = end\_index  
  
 # Print newline.  
 print()  
  
 return transfer\_values

Procesarea imagnilor de antrenare

In [21]:

def process\_images\_train():  
 print("Processing {0} images in training-set ...".format(len(filenames\_train)))  
  
 # Path for the cache-file.  
 cache\_path = os.path.join(coco.data\_dir,  
 "transfer\_values\_train.pkl")  
  
 # If the cache-file already exists then reload it,  
 # otherwise process all images and save their transfer-values  
 # to the cache-file so it can be reloaded quickly.  
 transfer\_values = cache(cache\_path=cache\_path,  
 fn=process\_images,  
 data\_dir=coco.train\_dir,  
 filenames=filenames\_train)  
  
 return transfer\_values

Procesarea imagnilor de validare

In [22]:

def process\_images\_val():  
 print("Processing {0} images in validation-set ...".format(len(filenames\_val)))  
  
 # Path for the cache-file.  
 cache\_path = os.path.join(coco.data\_dir, "transfer\_values\_val.pkl")  
  
 # If the cache-file already exists then reload it,  
 # otherwise process all images and save their transfer-values  
 # to the cache-file so it can be reloaded quickly.  
 transfer\_values = cache(cache\_path=cache\_path,  
 fn=process\_images,  
 data\_dir=coco.val\_dir,  
 filenames=filenames\_val)  
  
 return transfer\_values

Procesam toate imaginile din setul de antrenament și le salvam valorile de transfer într-un fișier cache. Acest proces a durat aproximativ 90 de minute pentru o GPU 1060 GTX.

In [23]:

%%time  
transfer\_values\_train = process\_images\_train()  
print("dtype:", transfer\_values\_train.dtype)  
print("shape:", transfer\_values\_train.shape)

Processing 118287 images in training-set ...  
- Data loaded from cache-file: data/coco/transfer\_values\_train.pkl  
dtype: float16  
shape: (118287, 4096)  
Wall time: 1.42 s

Procesam toate imaginile din setul de validare și le salvam valorile de transfer într-un fișier cache. Acest proces a durat aproximativ 10 minute pentru un GPU 1060 GTX.

In [24]:

%%time  
transfer\_values\_val = process\_images\_val()  
print("dtype:", transfer\_values\_val.dtype)  
print("shape:", transfer\_values\_val.shape)

Processing 5000 images in validation-set ...  
- Data loaded from cache-file: data/coco/transfer\_values\_val.pkl  
dtype: float16  
shape: (5000, 4096)  
Wall time: 61 ms

## Tokenizer[¶](#Tokenizer)

Retelele neuronale nu pot lucra direct pe cuvinte. Pentru a putea obtine o descriere textuala vom folosi un procedeu denumit tokenizare. Vom desemna un vocabular de 10000 de cuvinte si fiecarui cuvant ii vom atribui un numar intreg. Inceputul si finalul propozitiei vor fi formate din 2 cuvinte care sunt putin probabile sa faca parte din setul de antrenare.

In [25]:

mark\_start = 'ssss '  
mark\_end = ' eeee'

Functia va adauga markerul de inceput si final capturilor din setul de antrenare si verificare

In [26]:

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

In [27]:

captions\_train\_marked = mark\_captions(captions\_train)  
captions\_train\_marked[0]

Out[27]:

['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']

In [28]:

def flatten(captions\_listlist):  
 captions\_list = [caption  
 for captions\_list in captions\_listlist  
 for caption in captions\_list]  
   
 return captions\_list

Transformam setul de antrenare intr-un vector de numere intregi

In [29]:

captions\_train\_flat = flatten(captions\_train\_marked)

Setam numarul maxim de cuvinte distincte cu care vom antrena.

In [30]:

num\_words = 10000

Un wrapper asupra functiei de tokenizare deja existenta in Keras

In [31]:

class TokenizerWrap(Tokenizer):  
 """Wrap the Tokenizer-class from Keras with more functionality."""  
   
 def \_\_init\_\_(self, texts, num\_words=None):  
 """  
 :param texts: List of strings with the data-set.  
 :param num\_words: Max number of words to use.  
 """  
  
 Tokenizer.\_\_init\_\_(self, num\_words=num\_words)  
  
 # Create the vocabulary from the texts.  
 self.fit\_on\_texts(texts)  
  
 # Create inverse lookup from integer-tokens to words.  
 self.index\_to\_word = dict(zip(self.word\_index.values(),  
 self.word\_index.keys()))  
  
 def token\_to\_word(self, token):  
 """Lookup a single word from an integer-token."""  
  
 word = " " if token == 0 else self.index\_to\_word[token]  
 return word   
  
 def tokens\_to\_string(self, tokens):  
 """Convert a list of integer-tokens to a string."""  
  
 # Create a list of the individual words.  
 words = [self.index\_to\_word[token]  
 for token in tokens  
 if token != 0]  
   
 # Concatenate the words to a single string  
 # with space between all the words.  
 text = " ".join(words)  
  
 return text  
   
 def captions\_to\_tokens(self, captions\_listlist):  
 """  
 Convert a list-of-list with text-captions to  
 a list-of-list of integer-tokens.  
 """  
   
 # Note that text\_to\_sequences() takes a list of texts.  
 tokens = [self.texts\_to\_sequences(captions\_list)  
 for captions\_list in captions\_listlist]  
   
 return tokens

Cream o instanta a clasei

In [32]:

%%time  
tokenizer = TokenizerWrap(texts=captions\_train\_flat,  
 num\_words=num\_words)

Wall time: 8.5 s

Obtinem indexul markerului de start('ssss').

In [33]:

token\_start = tokenizer.word\_index[mark\_start.strip()]  
token\_start

Out[33]:

2

Obtinem indexul markerului de final('eeee').

In [34]:

token\_end = tokenizer.word\_index[mark\_end.strip()]  
token\_end

Out[34]:

3

In [35]:

%%time  
tokens\_train = tokenizer.captions\_to\_tokens(captions\_train\_marked)

Wall time: 8.43 s

Cuvintele convertite in numere intregi

In [36]:

tokens\_train[0]

Out[36]:

[[2, 841, 5, 2864, 5, 61, 26, 1984, 238, 9, 433, 3],  
 [2, 1, 429, 10, 3310, 7, 1025, 390, 501, 1110, 3],  
 [2, 63, 19, 993, 143, 8, 190, 958, 5, 743, 3],  
 [2, 299, 725, 25, 343, 208, 264, 9, 433, 3],  
 [2, 1, 170, 5, 1110, 26, 446, 190, 61, 3]]

Propozitiile corespondente

In [37]:

captions\_train\_marked[0]

Out[37]:

['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']

## Generatorul de date[¶](#Generatorul-de-date)

Fiecare imagine are cel putin 5 descrieri ale continutului. Reteaua neuronala va fi antrenata cu batch-uri ale valorilor intermediare venite de la vgg16 si o secvente de numere intregi reprezentand descrierea lor. Dacă ar trebui să avem matricile de numpy potrivite pentru setul de antrenament (adica ce obtinem la intrarea retelei recurente cu cele cel putin 5 descrieri valide), fie ar trebui să folosim o singură legendă pentru fiecare imagine și să ignorăm restul acestor date valoroase, fie ar trebui să repetăm valorile de transfer de imagine pentru fiecare dintre legendele, care ar irosi multă memorie.

O soluție mai bună este crearea unui generator de date personalizat pentru Keras care va crea un lot de date cu valori de transfer selectate aleatoriu și secvențe de simboluri.

Această funcție returnează o listă de secvențe ale tokenizer-ului aleatoriu pentru imagini cu indicii date în setul de antrenament.

In [38]:

def get\_random\_caption\_tokens(idx):  
 """  
 Given a list of indices for images in the training-set,  
 select a token-sequence for a random caption,  
 and return a list of all these token-sequences.  
 """  
   
 # Initialize an empty list for the results.  
 result = []  
  
 # For each of the indices.  
 for i in idx:  
 # The index i points to an image in the training-set.  
 # Each image in the training-set has at least 5 captions  
 # which have been converted to tokens in tokens\_train.  
 # We want to select one of these token-sequences at random.  
  
 # Get a random index for a token-sequence.  
 j = np.random.choice(len(tokens\_train[i]))  
  
 # Get the j'th token-sequence for image i.  
 tokens = tokens\_train[i][j]  
  
 # Add this token-sequence to the list of results.  
 result.append(tokens)  
  
 return result

This generator function creates random batches of training-data for use in training the neural network.

In [39]:

def batch\_generator(batch\_size):  
 """  
 Generator function for creating random batches of training-data.  
   
 Note that it selects the data completely randomly for each  
 batch, corresponding to sampling of the training-set with  
 replacement. This means it is possible to sample the same  
 data multiple times within a single epoch - and it is also  
 possible that some data is not sampled at all within an epoch.  
 However, all the data should be unique within a single batch.  
 """  
  
 # Infinite loop.  
 while True:  
 # Get a list of random indices for images in the training-set.  
 idx = np.random.randint(num\_images\_train,  
 size=batch\_size)  
   
 # Get the pre-computed transfer-values for those images.  
 # These are the outputs of the pre-trained image-model.  
 transfer\_values = transfer\_values\_train[idx]  
  
 # For each of the randomly chosen images there are  
 # at least 5 captions describing the contents of the image.  
 # Select one of those captions at random and get the  
 # associated sequence of integer-tokens.  
 tokens = get\_random\_caption\_tokens(idx)  
  
 # Count the number of tokens in all these token-sequences.  
 num\_tokens = [len(t) for t in tokens]  
   
 # Max number of tokens.  
 max\_tokens = np.max(num\_tokens)  
   
 # Pad all the other token-sequences with zeros  
 # so they all have the same length and can be  
 # input to the neural network as a numpy array.  
 tokens\_padded = pad\_sequences(tokens,  
 maxlen=max\_tokens,  
 padding='post',  
 truncating='post')  
   
 # Further prepare the token-sequences.  
 # The decoder-part of the neural network  
 # will try to map the token-sequences to  
 # themselves shifted one time-step.  
 decoder\_input\_data = tokens\_padded[:, 0:-1]  
 decoder\_output\_data = tokens\_padded[:, 1:]  
  
 # Dict for the input-data. Because we have  
 # several inputs, we use a named dict to  
 # ensure that the data is assigned correctly.  
 x\_data = \  
 {  
 'decoder\_input': decoder\_input\_data,  
 'transfer\_values\_input': transfer\_values  
 }  
  
 # Dict for the output-data.  
 y\_data = \  
 {  
 'decoder\_output': decoder\_output\_data  
 }  
   
 yield (x\_data, y\_data)

### Pasi per epoca[¶](#Pasi-per-epoca)

Dat fiind modul de generare al datelor putem intalni cazuri in care in aceeasi epoca se repeta o imagine de mai multe ori cu descrieri diferite, sau o imagine nu este utilizata deloc. Vom folosi totusi termenul de epoca pentru a masura timpul de antrenament al modelului

In [40]:

num\_captions\_train = [len(captions) for captions in captions\_train]

Numarul total de descrieri din setul de antrenare

In [41]:

total\_num\_captions\_train = np.sum(num\_captions\_train)

## Crearea Retelei Recurente[¶](#Crearea-Retelei-Recurente)

Vom crea o retea recurenta ce va primi la intrare valorile de transfer de la reteaua convolutionala ce are ca scop sa determine obiectele din imagine, cat si propozitia construita deja pentru a genera o descriere textuala a imaginii de la intrare. Aceasta retea este decoder-ul arhitecturii noastre.

Acest model va fi insa detasat de primul model. El va fi compilat si antrenat separat. Acesta este alcatuit din 3 straturi de

Gated Recurent Units a caror stare interna este:

In [42]:

state\_size = 512

Stratul de embedding converteste valorile intregi de la intrare in vectori de lungimea:

In [43]:

embedding\_size = 128

Valorile de intrare pentru decoder

In [44]:

transfer\_values\_input = Input(shape=(transfer\_values\_size,),  
 name='transfer\_values\_input')

La iesirea penultimului strat din VGG16 avem 4096 valori dar GRU-ul nostru are doar 512 stari interne. Vom reduce numarul de valori cu ajutorul unui strat dens intermediar de 512 neuroni ce va preceda arhitectura recurenta.

Folosim functia de activare tanh pentru a avea valori cuprinse intre -1 si 1.

In [45]:

decoder\_transfer\_map = Dense(state\_size,  
 activation='tanh',  
 name='decoder\_transfer\_map')

Intrarea in acest decoder poate avea lungimi arbitrare

In [46]:

decoder\_input = Input(shape=(None, ), name='decoder\_input')

In [47]:

decoder\_embedding = Embedding(input\_dim=num\_words,  
 output\_dim=embedding\_size,  
 name='decoder\_embedding')

Cream secventa de straturi recurente.

In [48]:

decoder\_gru1 = GRU(state\_size, name='decoder\_gru1',  
 return\_sequences=True)  
decoder\_gru2 = GRU(state\_size, name='decoder\_gru2',  
 return\_sequences=True)  
decoder\_gru3 = GRU(state\_size, name='decoder\_gru3',  
 return\_sequences=True)

La iesire vom avea un vector de forma [batch\_size, sequence\_length, state\_size]. Fiecare cuvant ce va alcatui descrierea va veni sub forma unui vector(format din activarile lui GRU3). Aceste secvente trebuie convertite in numere intregi pentru a putea obtine din maparea numar-cuvant anterior definita, o propozitie.

Aceasta mapare se va face cu ajutorul unui strat de iesire ce va avea numarul de neuroni egal cu numarul vocabularului nostru

In [49]:

decoder\_dense = Dense(num\_words,  
 activation='softmax',  
 name='decoder\_output')

### Conectarea intrarii si crearea modelului de antrenare[¶](#X36f58b3a4b61913c27c4f6d69c99a1f150c781f)

Conectarea retelei recurente la encoder ul nostru(VGG16)

In [50]:

def connect\_decoder(transfer\_values):  
 # Map the transfer-values so the dimensionality matches  
 # the internal state of the GRU layers. This means  
 # we can use the mapped transfer-values as the initial state  
 # of the GRU layers.  
 initial\_state = decoder\_transfer\_map(transfer\_values)  
  
 # Start the decoder-network with its input-layer.  
 net = decoder\_input  
   
 # Connect the embedding-layer.  
 net = decoder\_embedding(net)  
   
 # Connect all the GRU layers.  
 net = decoder\_gru1(net, initial\_state=initial\_state)  
 net = decoder\_gru2(net, initial\_state=initial\_state)  
 net = decoder\_gru3(net, initial\_state=initial\_state)  
  
 # Connect the final dense layer that converts to  
 # one-hot encoded arrays.  
 decoder\_output = decoder\_dense(net)  
   
 return decoder\_output

In [51]:

decoder\_output = connect\_decoder(transfer\_values=transfer\_values\_input)  
  
decoder\_model = Model(inputs=[transfer\_values\_input, decoder\_input],  
 outputs=[decoder\_output])

### Compilarea modelului[¶](#Compilarea-modelului)

Stratul de iesire va fi reprezentat de un set de neuroni peste care se va aplica o decodare de tipul 'one-hot'. Cu toate aceastea validarea contine un set de valori tokenizate. Ca functie loss, vom utiliza sparse cross-entropy ce face o conversie interna intre o valoare intreaga si un vector cu codare one hot pentru fiecare cuvant.

Am testat aici atat cu Adam optimizer cat si cu RMSprop si am ajuns la concluzia ca cel din urma are perfomantele cele mai mari. Adam functioneaza bine in primele epoci insa esueaza in a face o convergenta dupa ceva vreme.

In [52]:

decoder\_model.compile(optimizer=tf.keras.optimizers.RMSprop(  
 learning\_rate=0.001, rho=0.9, momentum=0.0, epsilon=1e-07, centered=False),  
 loss='sparse\_categorical\_crossentropy')

### Callback Functions[¶](#Callback-Functions)

In [53]:

path\_checkpoint = 'captionThisModel.h5'  
callback\_checkpoint = ModelCheckpoint(filepath=path\_checkpoint,  
 verbose=1,  
 save\_weights\_only=True)

Log-uri de TensorBoard(Inca nu m-am uitat peste ele). Pot fi utile in lucrarea teoretica

In [54]:

callback\_tensorboard = TensorBoard(log\_dir='./captionThis\_logs/',  
 histogram\_freq=0,  
 write\_graph=False)

In [55]:

callbacks = [callback\_checkpoint, callback\_tensorboard]

### Load Checkpoint[¶](#Load-Checkpoint)

Incarcarea ultimului model.

In [56]:

try:  
 decoder\_model.load\_weights(path\_checkpoint)  
except Exception as error:  
 print("Error trying to load checkpoint.")  
 print(error)

### Train the Model[¶](#Train-the-Model)

Vom antrena modelul decoder-ului.

O epoca a durat 4-5 ore pe un GTX 1060 GPU, 6GB cu un batch size de 3000.

In [57]:

batch\_size = 3000  
generator = batch\_generator(batch\_size=batch\_size)  
steps\_per\_epoch = int(total\_num\_captions\_train / batch\_size)  
steps\_per\_epoch

Out[57]:

197

In [141]:

%%time  
decoder\_model.fit(x=generator,  
 steps\_per\_epoch=steps\_per\_epoch,  
 epochs=10,  
 callbacks=callbacks)

Epoch 1/10  
295/295 [==============================] - ETA: 0s - loss: 0.6806   
Epoch 00001: saving model to captionThisModel.h5  
295/295 [==============================] - 17519s 59s/step - loss: 0.6806  
Epoch 2/10  
295/295 [==============================] - ETA: 0s - loss: 0.6770   
Epoch 00002: saving model to captionThisModel.h5  
295/295 [==============================] - 18075s 61s/step - loss: 0.6770  
Epoch 3/10  
295/295 [==============================] - ETA: 0s - loss: 0.6564   
Epoch 00003: saving model to captionThisModel.h5  
295/295 [==============================] - 17881s 61s/step - loss: 0.6564  
Epoch 4/10  
295/295 [==============================] - ETA: 0s - loss: 0.6512   
Epoch 00004: saving model to captionThisModel.h5  
295/295 [==============================] - 19080s 65s/step - loss: 0.6512  
Epoch 5/10  
295/295 [==============================] - ETA: 0s - loss: 0.6546   
Epoch 00005: saving model to captionThisModel.h5  
295/295 [==============================] - 18077s 61s/step - loss: 0.6546  
Epoch 6/10  
295/295 [==============================] - ETA: 0s - loss: 0.6537   
Epoch 00006: saving model to captionThisModel.h5  
295/295 [==============================] - 17695s 60s/step - loss: 0.6537  
Epoch 7/10  
295/295 [==============================] - ETA: 0s - loss: 0.6550   
Epoch 00007: saving model to captionThisModel.h5  
295/295 [==============================] - 18391s 62s/step - loss: 0.6550  
Epoch 8/10  
295/295 [==============================] - ETA: 0s - loss: 0.6620   
Epoch 00008: saving model to captionThisModel.h5  
295/295 [==============================] - 17478s 59s/step - loss: 0.6620  
Epoch 9/10  
295/295 [==============================] - ETA: 0s - loss: 0.6572   
Epoch 00009: saving model to captionThisModel.h5  
295/295 [==============================] - 17055s 58s/step - loss: 0.6572  
Epoch 10/10  
295/295 [==============================] - ETA: 0s - loss: 0.6526   
Epoch 00010: saving model to captionThisModel.h5  
295/295 [==============================] - 17615s 60s/step - loss: 0.6526  
Wall time: 2d 1h 50min 16s

## Generarea descrierilor[¶](#Generarea-descrierilor)

Functia incarca o imagine si genereaza o descriere

In [191]:

def generate\_caption(image\_path, max\_tokens=30, plot=True, printc=True):  
 """  
 Generate a caption for the image in the given path.  
 The caption is limited to the given number of tokens (words).  
 """  
  
 # Load and resize the image.  
 image = load\_image(image\_path, size=img\_size)  
   
 # Expand the 3-dim numpy array to 4-dim  
 # because the image-model expects a whole batch as input  
   
 # so we give it a batch with just one image.  
 image\_batch = np.expand\_dims(image, axis=0)  
  
 # Process the image with the pre-trained image-model  
 # to get the transfer-values.  
 transfer\_values = image\_model\_transfer.predict(image\_batch)  
  
 # Pre-allocate the 2-dim array used as input to the decoder.  
 # This holds just a single sequence of integer-tokens,  
 # but the decoder-model expects a batch of sequences.  
 shape = (1, max\_tokens)  
 decoder\_input\_data = np.zeros(shape=shape, dtype=np.int)  
  
 # The first input-token is the special start-token for 'ssss '.  
 token\_int = token\_start  
  
 # Initialize an empty output-text.  
 output\_text = ''  
  
 # Initialize the number of tokens we have processed.  
 count\_tokens = 0  
  
 # While we haven't sampled the special end-token for ' eeee'  
 # and we haven't processed the max number of tokens.  
 while token\_int != token\_end and count\_tokens < max\_tokens:  
 # Update the input-sequence to the decoder  
 # with the last token that was sampled.  
 # In the first iteration this will set the  
 # first element to the start-token.  
 decoder\_input\_data[0, count\_tokens] = token\_int  
  
 # Wrap the input-data in a dict for clarity and safety,  
 # so we are sure we input the data in the right order.  
 x\_data = {  
 'transfer\_values\_input': transfer\_values,  
 'decoder\_input': decoder\_input\_data  
 }  
   
 # Note that we input the entire sequence of tokens  
 # to the decoder. This wastes a lot of computation  
 # because we are only interested in the last input  
 # and output. We could modify the code to return  
 # the GRU-states when calling predict() and then  
 # feeding these GRU-states as well the next time  
 # we call predict(), but it would make the code  
 # much more complicated.  
   
 # Input this data to the decoder and get the predicted output.  
 decoder\_output = decoder\_model.predict(x\_data)  
  
 # Get the last predicted token as a one-hot encoded array.  
 # Note that this is not limited by softmax, but we just  
 # need the index of the largest element so it doesn't matter.  
 token\_onehot = decoder\_output[0, count\_tokens, :]  
  
 # Convert to an integer-token.  
 token\_int = np.argmax(token\_onehot)  
  
 # Lookup the word corresponding to this integer-token.  
 sampled\_word = tokenizer.token\_to\_word(token\_int)  
  
 # Append the word to the output-text.  
 output\_text += " " + sampled\_word  
  
 # Increment the token-counter.  
 count\_tokens += 1  
  
 # This is the sequence of tokens output by the decoder.  
 output\_tokens = decoder\_input\_data[0]  
 if plot:  
 # Plot the image.  
 plt.imshow(image)  
 plt.show()  
 output\_text=output\_text[1].upper()+output\_text[2:-5]+'.'  
   
 if printc== True:  
 # Print the predicted caption.  
 print("Predicted caption:")  
 print(output\_text)  
 print()  
 return output\_text

### Example[¶](#Example)

O poza cu 2 pasari

In [192]:

generate\_caption("test/birds.jpg")

![](data:image/png;base64;base64,)

O poza cu un atlet

In [193]:

generate\_caption("test/athlete.jpg")

![](data:image/png;base64;base64,)

O poza cu un urs

In [194]:

generate\_caption("test/bear.jpg")

![](data:image/png;base64;base64,)

O poza cu o plaja

In [195]:

generate\_caption("test/beach.jpg")

![](data:image/png;base64;base64,)

O poza cu o masina

In [196]:

generate\_caption("test/golf.jpg")

![](data:image/png;base64;base64,)

O poza cu un om

In [197]:

generate\_caption("test/musk.jpeg")

![](data:image/png;base64;base64,)

O poza cu o femeie si un obiect

In [198]:

generate\_caption("test/marion.jpg")

![](data:image/png;base64;base64,)

O poza doar cu o femeie

In [199]:

generate\_caption("test/marion1.jpg")

![](data:image/png;base64;base64,)

Sa incercam cu poza unei capre

In [200]:

generate\_caption("test/deer.jpg")

![](data:image/png;base64;base64,)

Urmeaza sa definim o functie care sa ne afiseze o imagine din setul de antrenare/validare si rezultatul pe care il produce modelul nostru.

In [201]:

def generate\_caption\_coco(idx, train=False):  
 """  
 Generate a caption for an image in the COCO data-set.  
 Use the image with the given index in either the  
 training-set (train=True) or validation-set (train=False).  
 """  
   
 if train:  
 # Use image and captions from the training-set.  
 data\_dir = coco.train\_dir  
 filename = filenames\_train[idx]  
 captions = captions\_train[idx]  
 else:  
 # Use image and captions from the validation-set.  
 data\_dir = coco.val\_dir  
 filename = filenames\_val[idx]  
 captions = captions\_val[idx]  
  
 # Path for the image-file.  
 path = os.path.join(data\_dir, filename)  
  
 # Use the model to generate a caption of the image.  
 generate\_caption(image\_path=path)  
  
 # Print the true captions from the data-set.  
 print("True captions:")  
 for caption in captions:  
 print(caption)

Alta imagine din setul de antrenare

In [202]:

generate\_caption\_coco(idx=1, train=True)

![](data:image/png;base64;base64,)

Alta imagine din setul de validare

In [203]:

generate\_caption\_coco(idx=4999, train=False)

![](data:image/png;base64;base64,)

# Metrica de acuratețe[¶](#Metrica-de-acuratețe)

![metrica.PNG](data:image/png;base64;base64,)

In [204]:

import nltk  
nltk.data.path=[r'C:\Users\George\Desktop\Dizertatie\Python']  
nltk.download()

showing info https://raw.githubusercontent.com/nltk/nltk\_data/gh-pages/index.xml

In [220]:

def MeteorScore():  
 data\_dir = coco.val\_dir  
 acc=0  
 for idx in range(num\_images\_val):  
 filename = filenames\_val[idx]  
 caption\_val\_set = captions\_val[idx]  
 path=os.path.join(data\_dir, filename)  
 caption=generate\_caption(image\_path=path,plot=False,printc=False)  
 acc+=round(nltk.meteor(caption\_val\_set, caption),4)  
 return acc/num\_images\_val

In [212]:

print('METEOR score is {}'.format(round(MeteorScore(),4)))

METEOR score is 0.3408

![BLEU.PNG](data:image/png;base64;base64,)

In [ ]:

import nltk.translate.bleu\_score as bleu

In [213]:

def BleuScore():  
 data\_dir = coco.val\_dir  
 acc=0  
 for idx in range(num\_images\_val):  
 filename = filenames\_val[idx]  
 caption\_val\_set = captions\_val[idx]  
 path=os.path.join(data\_dir, filename)  
 caption=generate\_caption(image\_path=path,plot=False,printc=False)  
 acc+=round(bleu.sentence\_bleu(caption\_val\_set, caption),4)  
 return acc/num\_images\_val

In [218]:

print('BLEU score is {}'.format(round(BleuScore(),4)))

BLEU score is 0.3408

Salvam Modelul

In [76]:

decoder\_model.summary()

Model: "model\_1"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Layer (type) Output Shape Param # Connected to   
==================================================================================================  
decoder\_input (InputLayer) [(None, None)] 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
transfer\_values\_input (InputLay [(None, 4096)] 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
decoder\_embedding (Embedding) (None, None, 128) 1280000 decoder\_input[0][0]   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
decoder\_transfer\_map (Dense) (None, 512) 2097664 transfer\_values\_input[0][0]   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
decoder\_gru1 (GRU) (None, None, 512) 986112 decoder\_embedding[0][0]   
 decoder\_transfer\_map[0][0]   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
decoder\_gru2 (GRU) (None, None, 512) 1575936 decoder\_gru1[0][0]   
 decoder\_transfer\_map[0][0]   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
decoder\_gru3 (GRU) (None, None, 512) 1575936 decoder\_gru2[0][0]   
 decoder\_transfer\_map[0][0]   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
decoder\_output (Dense) (None, None, 10000) 5130000 decoder\_gru3[0][0]   
==================================================================================================  
Total params: 12,645,648  
Trainable params: 12,645,648  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

In [77]:

decoder\_model.save(r'C:\Users\George\Desktop\Dizertatie\Python\Caption Model\finalModel')

WARNING:tensorflow:From C:\Users\George\Anaconda3\envs\tf2\lib\site-packages\tensorflow\python\ops\resource\_variable\_ops.py:1817: calling BaseResourceVariable.\_\_init\_\_ (from tensorflow.python.ops.resource\_variable\_ops) with constraint is deprecated and will be removed in a future version.  
Instructions for updating:  
If using Keras pass \*\_constraint arguments to layers.  
INFO:tensorflow:Assets written to: C:\Users\George\Desktop\Dizertatie\Python\Caption Model\finalModel\assets

Verificam ca modelul se incă

In [78]:

newModel=tf.keras.models.load\_model('finalModel')  
newModel.summary()

Model: "model\_1"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Layer (type) Output Shape Param # Connected to   
==================================================================================================  
decoder\_input (InputLayer) [(None, None)] 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
transfer\_values\_input (InputLay [(None, 4096)] 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
decoder\_embedding (Embedding) (None, None, 128) 1280000 decoder\_input[0][0]   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
decoder\_transfer\_map (Dense) (None, 512) 2097664 transfer\_values\_input[0][0]   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
decoder\_gru1 (GRU) (None, None, 512) 986112 decoder\_embedding[0][0]   
 decoder\_transfer\_map[0][0]   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
decoder\_gru2 (GRU) (None, None, 512) 1575936 decoder\_gru1[0][0]   
 decoder\_transfer\_map[0][0]   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
decoder\_gru3 (GRU) (None, None, 512) 1575936 decoder\_gru2[0][0]   
 decoder\_transfer\_map[0][0]   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
decoder\_output (Dense) (None, None, 10000) 5130000 decoder\_gru3[0][0]   
==================================================================================================  
Total params: 12,645,648  
Trainable params: 12,645,648  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Sfarsit/Concluzii[¶](#Sfarsit/Concluzii)