1. Importing Libraries and Packages.

In [1]:

```
# Installing the packages for the metric
!pip install strsim

# Installing Fasttext for unsupervised embeddings
!pip install fasttext

# Downloading fasttext embeddings and keeping the '.bin' file whicl
!wget https://dl.fbaipublicfiles.com/fasttext/vectors-english/crawl
!unzip crawl-300d-2M-subword.zip
!rm -r /content/crawl-300d-2M-subword.vec
!rm -r /content/crawl-300d-2M-subword.zip
```

In [42]:

```
#Importing libraries
import glob
import json
import re
import collections
import random
import os
import time
import warnings
import fasttext
import datetime
import pickle
import pandas as pd
import numpy as np
import tensorflow as tf
import tensorflow_addons as tfa
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import matplotlib.image as mpimg
from PIL import Image
from tqdm import tqdm
from similarity.normalized_levenshtein import NormalizedLevenshtein
warnings.filterwarnings('ignore')
```

In [3]:

```
#Deleting unnecessary files if needed.
#!rm -r /content/drive/MyDrive/Data/training_checkpoints
#!rm /content/Data/train/documents/*.npy
#!rm -r /content/drive/MyDrive/Data/tensorboard_logs
```

2. Data Collection.

In [1]:

```
# Data Collection
!wget http://datasets.cvc.uab.es/rrc/DocVQA/test.tar.gz
!wget http://datasets.cvc.uab.es/rrc/DocVQA/train.tar.gz
!wget http://datasets.cvc.uab.es/rrc/DocVQA/val.tar.gz
!mkdir /content/Data
!tar -xf train.tar.gz -C /content/Data
!tar -xf test.tar.gz -C /content/Data
!tar -xf val.tar.gz -C /content/Data
!rm test.tar.gz
!rm train.tar.gz
```

```
!rm val.tar.gz
```

```
--2021-02-08 17:12:24--  http://datasets.cvc.uab.es/rrc/DocVQA/te
st.tar.gz (http://datasets.cvc.uab.es/rrc/DocVQA/test.tar.gz)
Resolving datasets.cvc.uab.es (datasets.cvc.uab.es)... 158.109.8.
18
Connecting to datasets.cvc.uab.es (datasets.cvc.uab.es)|158.109.
8.18|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 921292800 (879M) [application/x-gzip]
Saving to: ‘test.tar.gz’

test.tar.gz         100%[===================>] 878.61M  15.7MB/s
in 91s

2021-02-08 17:13:55 (9.69 MB/s) - ‘test.tar.gz’ saved [921292800/
921292800]

--2021-02-08 17:13:55--  http://datasets.cvc.uab.es/rrc/DocVQA/tr
ain.tar.gz (http://datasets.cvc.uab.es/rrc/DocVQA/train.tar.gz)
Resolving datasets.cvc.uab.es (datasets.cvc.uab.es)... 158.109.8.
18
Connecting to datasets.cvc.uab.es (datasets.cvc.uab.es)|158.109.
8.18|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 7122739200 (6.6G) [application/x-gzip]
Saving to: ‘train.tar.gz’

train.tar.gz        100%[===================>]   6.63G  4.27MB/s
in 13m 0s

2021-02-08 17:26:55 (8.71 MB/s) - ‘train.tar.gz’ saved [712273920
0/7122739200]

--2021-02-08 17:26:55--  http://datasets.cvc.uab.es/rrc/DocVQA/va
l.tar.gz (http://datasets.cvc.uab.es/rrc/DocVQA/val.tar.gz)
Resolving datasets.cvc.uab.es (datasets.cvc.uab.es)... 158.109.8.
18
Connecting to datasets.cvc.uab.es (datasets.cvc.uab.es)|158.109.
8.18|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 864788480 (825M) [application/x-gzip]
Saving to: ‘val.tar.gz’

val.tar.gz          100%[===================>] 824.73M  4.18MB/s
in 2m 32s

2021-02-08 17:29:28 (5.41 MB/s) - ‘val.tar.gz’ saved [864788480/8
64788480]
```

3. Getting the datasets ready.

3.1. Defining the Function.

The **"check_data"** function is used to check whether all the lists corresponding to the columns are of equal length or not.

The **"prepare_df"** function returns a dataframe with 6 columns(Image-Link of the image, OCR-Link of the OCR result of the respective images, Questions-Questions asked on the image, Answers-Answers with respect to the question, Answers_List-List of correct Answers)

In [ ]:

```
def check_data(image,ocr,questions,questionid,answers,answers_list)
  '''
    The function "check_data" takes 5 inputs(lists) which are :
      image = Links of the images of where the image is present,
      ocr = Links of the OCR data of where the OCR data is present,
      questions = Questions corresponding to the Image and the OCR,
      questionId = Question ID of the Question,
      answers = Answers of those questions,
      answers_list = All the right answers corresponding to the Que
    It Returns a Boolean value on whether all the lengths of the li
  '''
  if answers is not None and answers_list is not None:
    assert(len(image)==len(ocr) and
           len(image)==len(questions) and
           len(image)==len(questionid) and
           len(image)==len(answers) and
           len(image)==len(answers_list))
  else:
    assert(len(image)==len(ocr) and
           len(image)==len(questions) and
           len(image)==len(questionid))
  return "\nData is loaded and is correct"

def prepare_df(df_type):
  '''
    The function "prepare_df" takes 1 input(String):
      df_type = The type of dataset(String value) whether it be 'tr
    It Returns a Dataframe with columns:
      Image(train, val, test) = Links of the images of where the im
      OCR(train, val, test) = Links of the OCR data of where the OC
      Questions(train, val, test) = Questions corresponding to the 
      QuestionId(train, val, test) = Question ID of the Question,
      Answers(train, val) = Answers of those questions,
      Answers_list(train, val) = All the right answers correspondin
  '''
  # STEP 1 - Getting all the paths of the images based on the datas
  image_paths = glob.glob("/content/Data/" + df_type + "/documents/
  # STEP 2 - Getting all the paths of the ocr text data based on th
  ocr_paths = glob.glob("/content/Data/" + df_type + "/ocr_results/

  # STEP 3 - Code for creating a column in the dataframe of the ocr
  temp_image_paths_strings = []
  for i in image_paths:
    temp_image_paths_strings.append(i.split('/')[-1].replace('.png'

  final_ocr_paths = []
  for image_str in tqdm(temp_image_paths_strings):
    for ocr_str in ocr_paths:
      if image_str == ocr_str.split('/')[-1]:
        final_ocr_paths.append(ocr_str)

  # STEP 4 - Reading the json file corresponding to each dataset_ty
  with open("/content/Data/" + df_type + "/" + df_type + "_v1.0.jso
    annotations = json.load(f)

  # STEP 5 - Getting the lists ready with respect to each column.
  new_image_paths = []
  new_ocr_paths = []
  new_questions = []
  new_answers = []
  new_answers_list = []
  new_question_id = []

  #The test dataset_type doesn't has answers for each question.
  #So, it doesn't return columns, Answers and Answers_List, in the
  if df_type=='test' :
    for image, ocr in tqdm(zip(image_paths,final_ocr_paths)):
      count_of_questions = 0
      for i in annotations['data']:
        if image.split('/')[-1] in i['image']:
          new_questions.append(i['question'].lower())
          new_question_id.append(i['questionId'])
          count_of_questions+=1

      new_image_paths.extend([image]*(count_of_questions))
      new_ocr_paths.extend([ocr]*(count_of_questions))
    new_answers = new_answers_list = None

  # The else condition is for train and validation datasets where i
  else:
    for image, ocr in tqdm(zip(image_paths,final_ocr_paths)):
      count_of_questions = 0
      for i in annotations['data']:
        if image.split('/')[-1] in i['image']:
          more_answers = []
          answer = i['answers'][0]
          #If the answer has n/a present in it replace it with "no
          if i['answers'][0].lower()=="n/a" or i['answers'][0].lowe
            new_answers.append("no answer")
            for answer in i['answers']:
              more_answers.append("no answer")
          else:
            new_answers.append(i['answers'][0].lower())
            for answer in i['answers']:
              more_answers.append(answer.lower())

          new_questions.append(i['question'].lower())
          new_question_id.append(i['questionId'])
          new_answers_list.append(np.array(more_answers))
          count_of_questions+=1

      new_image_paths.extend([image]*(count_of_questions))
      new_ocr_paths.extend([ocr]*(count_of_questions))

  # STEP 6 - Check if all the lists corresponding to the columns ar
  print(check_data(new_image_paths,
                   new_ocr_paths,
                   new_questions,
                   new_question_id,
                   new_answers,
                   new_answers_list))

  # STEP 7 - Create a Dataframe with respect to all the lists creat
```

```
  if new_answers is not None:
    df = pd.DataFrame(list(zip(new_image_paths,
                              new_ocr_paths,
                              new_questions,
                              new_question_id,
                              new_answers,
                              new_answers_list)),
                     columns =['Image', 'OCR', 'Question', 'Questio
  else:
    df = pd.DataFrame(list(zip(new_image_paths,
                              new_ocr_paths,
                              new_questions,
                              new_question_id)),
                     columns =['Image', 'OCR', 'Question', 'Questio

  # STEP 8 - Return the Dataframe.
  return df
```

3.2. Train

In [ ]:

```
# Getting the Train data
train = prepare_df('train')
```

```
100%|██████████| 10194/10194 [00:35<00:00, 290.25it/s]
10194it [02:49, 59.97it/s]


Data is loaded and is correct
```

In [ ]:

```
print("Number of rows : {}".format(train.shape[0]))
print("Number of columns : {}".format(train.shape[1]))
```

```
Number of rows : 39463
Number of columns : 6
```

In [ ]:

```
train.head(60)
```

Out[7]:

	Image	OCR	Question	Question_Id	Answer
0	/content/Data/train /documents /xrcw0217_2.png	/content/Data/train /ocr_results /xrcw0217_2.json	what is the project # number?	45894	8910
1	/content/Data/train /documents /xrcw0217_2.png	/content/Data/train /ocr_results /xrcw0217_2.json	what is the date mentioned in the document?	45895	may 2002
2	/content/Data/train /documents /nhng0227_1.png	/content/Data/train /ocr_results /nhng0227_1.json	what is the date on the document?	31505	november 7, 1961
3	/content/Data/train /documents /nhng0227_1.png	/content/Data/train /ocr_results /nhng0227_1.json	what is the department?	31506	wohl clinic
4	/content/Data/train /documents /nhng0227_1.png	/content/Data/train /ocr_results /nhng0227_1.json	what is the suggested source?	31508	holscher-wernig

3.3. Validation.

In [ ]:
```
# Getting the Validation data
val = prepare_df('val')
```

```
100%|██████████| 1286/1286 [00:00<00:00, 2380.08it/s]
1286it [00:02, 448.34it/s]


Data is loaded and is correct
```

In [ ]:
```
print("Number of rows : {}".format(val.shape[0]))
print("Number of columns : {}".format(val.shape[1]))
```

```
Number of rows : 5349
Number of columns : 6
```

In [ ]:
```
val.head(60)
```

Out[11]:

	Image	OCR	Question	Question_Id	Answer	
0	/content/Data/val /documents /njdv0228_13.png	/content/Data/val /ocr_results /njdv0228_13.json	what is the form number?	63133	164	
1	/content/Data/val /documents /njdv0228_13.png	/content/Data/val /ocr_results /njdv0228_13.json	what is the date of deposit?	63135	august 9, 1976	[a
2	/content/Data/val /documents /njdv0228_13.png	/content/Data/val /ocr_results /njdv0228_13.json	what is the amount of deposit?	63137	100.00	[1
3	/content/Data/val /documents /jjvg0227_5.png	/content/Data/val /ocr_results /jjvg0227_5.json	what is the page number?	32998	page 5	
4	/content/Data/val /documents /pmmg0227_2.png	/content/Data/val /ocr_results /pmmg0227_2.json	what is the building name written at the right	62358	david p. wohl jr. memorial	[d

3.4. Test

In [ ]:
```
# Getting the Test data
test = prepare_df('test')
```

```
100%|██████████| 1287/1287 [00:00<00:00, 2244.22it/s]
1287it [00:02, 475.59it/s]


Data is loaded and is correct
```

In [ ]:
```
print("Number of rows : {}".format(test.shape[0]))
print("Number of columns : {}".format(test.shape[1]))
```

```
Number of rows : 5188
Number of columns : 4
```

In [ ]:
```
test.head(60)
```

Out[14]:

	Image	OCR	Question	Question_Id
0	/content/Data/test/documents/grvv0228_28.png	/content/Data/test/ocr_results/grvv0228_28.json	what is the value in y' axis?	49359
1	/content/Data/test/documents/grvv0228_28.png	/content/Data/test/ocr_results/grvv0228_28.json	how much is the 'scn direct to unmc' amount in...	49389
2	/content/Data/test/documents/grvv0228_28.png	/content/Data/test/ocr_results/grvv0228_28.json	how much is the highest grant amount of unmc -...	49385
3	/content/Data/test/documents/trgj0223_91.png	/content/Data/test/ocr_results/trgj0223_91.json	what is the page number?	57806
4	/content/Data/test/documents/trgj0223_91.png	/content/Data/test/ocr_results/trgj0223_91.json	what is the name of the company in the logo?	57812
5	/content/Data/test/documents/trgj0223_91.png	/content/Data/test/ocr_results/trgj0223_91.json	what is the name of the small scale/ ancillary...	57813
6	/content/Data/test/documents/hlvj0223_46.png	/content/Data/test/ocr_results/hlvj0223_46.json	what is the page number?	55263
7	/content/Data/test/documents/qpvw0217_5.png	/content/Data/test/ocr_results/qpvw0217_5.json	what is the pub. no. given in this document?	15354
8	/content/Data/test/documents/qpvw0217_5.png	/content/Data/test/ocr_results/qpvw0217_5.json	what is the ms topic/no. mentioned in this doc...	15355
9	/content/Data/test/documents/qpvw0217_5.png	/content/Data/test/ocr_results/qpvw0217_5.json	who is the author of the article "can a health...	15356
10	/content/Data/test/documents/qpvw0217_5.png	/content/Data/test/ocr_results/qpvw0217_5.json	in which journal, the article is published?	15357
11	/content/Data/test/documents/qpvw0217_5.png	/content/Data/test/ocr_results/qpvw0217_5.json	when is the manuscript sent to author for subm...	15358
12	/content/Data/test/documents/qpvw0217_5.png	/content/Data/test/ocr_results/qpvw0217_5.json	when is the publication target as per the docu...	15359
13	/content/Data/test/documents/hghf0227_1.png	/content/Data/test/ocr_results/hghf0227_1.json	what is the name of the individual completing ...	29023
14	/content/Data/test/documents/hghf0227_1.png	/content/Data/test/ocr_results/hghf0227_1.json	how many sessions are devoted to mental health...	29024

	Image	OCR	Question	Question_Id
15	/content/Data/test/documents/tynx0037_4.png	/content/Data/test/ocr_results/tynx0037_4.json	whose contact is given at the end?	262
16	/content/Data/test/documents/tynx0037_4.png	/content/Data/test/ocr_results/tynx0037_4.json	since when is camel available in the market?	263
17	/content/Data/test/documents/tynx0037_4.png	/content/Data/test/ocr_results/tynx0037_4.json	"pleasure to burn since 1913"; which cigarette...	264
18	/content/Data/test/documents/rzjh0227_1.png	/content/Data/test/ocr_results/rzjh0227_1.json	what is the name mentioned in the top of the d...	5229
19	/content/Data/test/documents/rzjh0227_1.png	/content/Data/test/ocr_results/rzjh0227_1.json	what is the date of birth of milton ?	5232
20	/content/Data/test/documents/rzjh0227_1.png	/content/Data/test/ocr_results/rzjh0227_1.json	what is the public school number ?	5234
21	/content/Data/test/documents/rzjh0227_1.png	/content/Data/test/ocr_results/rzjh0227_1.json	what is the date mentioned in the bottom of th...	5238
22	/content/Data/test/documents/lmyc0227_2.png	/content/Data/test/ocr_results/lmyc0227_2.json	what is the number of live births for alabama?	40745
23	/content/Data/test/documents/lmyc0227_2.png	/content/Data/test/ocr_results/lmyc0227_2.json	which is the first state listed?	40740
24	/content/Data/test/documents/lmyc0227_2.png	/content/Data/test/ocr_results/lmyc0227_2.json	which is the last state listed?	40743
25	/content/Data/test/documents/lmyc0227_2.png	/content/Data/test/ocr_results/lmyc0227_2.json	what is the number of live births for tennesee?	40749
26	/content/Data/test/documents/lmyc0227_2.png	/content/Data/test/ocr_results/lmyc0227_2.json	what is the number of live births for texas?	40751
27	/content/Data/test/documents/lmyc0227_2.png	/content/Data/test/ocr_results/lmyc0227_2.json	what is the rate per 100,000 live births for k...	40755
28	/content/Data/test/documents/lmyc0227_2.png	/content/Data/test/ocr_results/lmyc0227_2.json	what is the rate per 100,000 live births for g...	40758
29	/content/Data/test/documents/lmyc0227_2.png	/content/Data/test/ocr_results/lmyc0227_2.json	what is the number of maternal deaths for n.ca...	40762
30	/content/Data/test/documents/lmyc0227_2.png	/content/Data/test/ocr_results/lmyc0227_2.json	what is the number of maternal deaths for n.ha...	40764

	Image	OCR	Question	Question_Id
31	/content/Data/test/documents/lzbw0217_4.png	/content/Data/test/ocr_results/lzbw0217_4.json	what is the name of the council ?	58433
32	/content/Data/test/documents/lzbw0217_4.png	/content/Data/test/ocr_results/lzbw0217_4.json	what is the rate for 1 room/night?	58435
33	/content/Data/test/documents/lzbw0217_4.png	/content/Data/test/ocr_results/lzbw0217_4.json	what is the estimated budget for site visit?	58438
34	/content/Data/test/documents/lzbw0217_4.png	/content/Data/test/ocr_results/lzbw0217_4.json	what is the estimated budget for 22 rooms/3 ni...	58440
35	/content/Data/test/documents/lzbw0217_4.png	/content/Data/test/ocr_results/lzbw0217_4.json	what is the estimated budget for 150 rooms/2ni...	58442
36	/content/Data/test/documents/lzjf0226_2.png	/content/Data/test/ocr_results/lzjf0226_2.json	what does this document relate to ?	996
37	/content/Data/test/documents/lzjf0226_2.png	/content/Data/test/ocr_results/lzjf0226_2.json	what is the heading of second paragraph?	999
38	/content/Data/test/documents/kzbn0226_41.png	/content/Data/test/ocr_results/kzbn0226_41.json	what 'is common in the eu'?	51163
39	/content/Data/test/documents/kzbn0226_41.png	/content/Data/test/ocr_results/kzbn0226_41.json	what is the page number?	51164
40	/content/Data/test/documents/phvd0227_14.png	/content/Data/test/ocr_results/phvd0227_14.json	what is the page number?	60728
41	/content/Data/test/documents/phvd0227_14.png	/content/Data/test/ocr_results/phvd0227_14.json	what is the underlined word in the last line?	60730
42	/content/Data/test/documents/kmmw0228_2.png	/content/Data/test/ocr_results/kmmw0228_2.json	what is the date on the check?	56444
43	/content/Data/test/documents/kmmw0228_2.png	/content/Data/test/ocr_results/kmmw0228_2.json	what is the name of the bank?	56445
44	/content/Data/test/documents/rjxg0227_4.png	/content/Data/test/ocr_results/rjxg0227_4.json	what is the day and date on page 6?	60708
45	/content/Data/test/documents/rjxg0227_4.png	/content/Data/test/ocr_results/rjxg0227_4.json	what is the day and date on page 7?	60709
46	/content/Data/test/documents/rjxg0227_4.png	/content/Data/test/ocr_results/rjxg0227_4.json	what is the timing for the morning session?	60710
47	/content/Data/test/documents/rjxg0227_4.png	/content/Data/test/ocr_results/rjxg0227_4.json	what is the timing for the afternoon session?	60711

	Image	OCR	Question	Question_Id
48	/content/Data/test/documents/ktgn0226_2.png	/content/Data/test/ocr_results/ktgn0226_2.json	when is the knee panel meeting held in chicago...	41928
49	/content/Data/test/documents/ktgn0226_2.png	/content/Data/test/ocr_results/ktgn0226_2.json	who is presenting the 'hot topics and future t...	41931
50	/content/Data/test/documents/ktgn0226_2.png	/content/Data/test/ocr_results/ktgn0226_2.json	what time is the working lunch on friday, octo...	41933
51	/content/Data/test/documents/ktgn0226_2.png	/content/Data/test/ocr_results/ktgn0226_2.json	when is the meeting adjourned?	41935
52	/content/Data/test/documents/xkbw0217_50.png	/content/Data/test/ocr_results/xkbw0217_50.json	what is the y-axis of the plot?	60174
53	/content/Data/test/documents/xkbw0217_50.png	/content/Data/test/ocr_results/xkbw0217_50.json	what is the % change from baseline for 36 mont...	60456
54	/content/Data/test/documents/tggg0227_2.png	/content/Data/test/ocr_results/tggg0227_2.json	what is the page number?	62518
55	/content/Data/test/documents/tggg0227_2.png	/content/Data/test/ocr_results/tggg0227_2.json	who is the letter to ?	62519
56	/content/Data/test/documents/qhng0227_1.png	/content/Data/test/ocr_results/qhng0227_1.json	apart from formica what does they distribute m...	62467

3.5. Saving the datasets

In [ ]:

```
# Saving the datasets in a csv format.
train.to_csv('/content/drive/MyDrive/Data/train.csv', index=False)
val.to_csv('/content/drive/MyDrive/Data/val.csv', index=False)
test.to_csv('/content/drive/MyDrive/Data/test.csv', index=False)
```

4. Preprocessing and Modeling.

4.1. Model 1. (Baseline Model) (Image VQA)

4.1.1. Data Preprocessing

4.1.1.1. Importing Data.

Over here I have appended a new datapoint in the train and 3 datapoints validation dataset to avoid an error while training.

Since the train data has a shape of '(39463,6)', concatenating a datapoint(duplicate of the last datapoint in the dataset) turns it into a shape of '(39464,6)'. This makes it easy to create batch sizes in accordance to multiples of of the batch size.

Since I have a batch size of 8, it creates batch size of 4,933 for each batch and also it

In [ ]:

```
#Importing datasets
train = pd.read_csv('/content/drive/MyDrive/Data/train.csv')
val = pd.read_csv('/content/drive/MyDrive/Data/val.csv')
test = pd.read_csv('/content/drive/MyDrive/Data/test.csv')

#Appending an extra datapoint in the train dataset.
image = list(train.Image)[-1]
ocr = list(train.OCR)[-1]
question = list(train.Question)[-1]
question_id = list(train.Question_Id)[-1]
answer = list(train.Answer)[-1]
answer_list = list(train.Answer_list)[-1]
new_record = pd.DataFrame([[image,ocr,question,question_id,answer,a
                          columns=['Image', 'OCR', 'Question', 'Que
train = pd.concat([train,new_record])

#Appending an extra datapoint in the validation dataset.
for i in range(3):
  image = list(val.Image)[i]
  ocr = list(val.OCR)[i]
  question = list(val.Question)[i]
  question_id = list(val.Question_Id)[i]
  answer = list(val.Answer)[i]
  answer_list = list(val.Answer_list)[i]
  new_record = pd.DataFrame([[image,ocr,question,question_id,answer
                          columns=['Image', 'OCR', 'Question', 'Que
  val = pd.concat([val,new_record])
```

In [ ]:

```
print("Shape of Train data : \t\t",train.shape)
print("Shape of Validation data : \t",val.shape)
print("Shape of Test data : \t\t",test.shape)
```

```
Shape of Train data :            (39464, 6)
Shape of Validation data :       (5352, 6)
Shape of Test data :             (5188, 4)
```

4.1.1.2. Questions and Answers.

The function **'preprocess_qa'**, preprocesses the questions as well as the answers by adding a space between the punctuations and alphanumeric characters and also gets rid of the redundant spaces.

It also adds a '<'start>' and '<'end>' token to each text and lowers the characters in the texts.

In [ ]:

```
def preprocess_qa(w):
  '''
    The function 'preprocess_qa' preprocesses both the questions as
  '''
  # creating a space between a word and the punctuation following i
  # eg: "he is a boy." => "he is a boy ."
  # Reference:- https://stackoverflow.com/questions/3645931/python
  w = re.sub('([!"#$%&()*+.,-/:;=?@[\]<>?^_`{|}~])', r' \1 ', w)
  w = re.sub('\s{2,}', ' ', w)

  # replacing everything with space except (a-z, A-Z, ".", "?", "!
  w = re.sub('(?<=[A-Za-z])(?=[0-9])|(?<=[0-9])(?=[A-Za-z])',' ', w

  w = ' '.join(e.lower() for e in w.split())

  w = w.strip()

  # adding a start and an end token to the sentence
  # so that the model know when to start and stop predicting.
  w = '<start> ' + w + ' <end>'
  return w
```

In [ ]:

```
eg_question = train['Question'][1]
print(preprocess_qa(eg_question))
del eg_question
```

```
<start> what is the date mentioned in the document ? <end>
```

In [ ]:

```
eg_answer = train['Answer'][1]
print(preprocess_qa(eg_answer))
del eg_answer
```

```
<start> may 2002 <end>
```

The function **'create_dataset'** returns, preprocessed questions for text dataset, and, preprocessed questions and answers for train and validation datasets.

The function has been used for test and validation sets as well, because the unsupervised tokens and embeddings can be created for all the dataset types together.

In [ ]:

```
def create_dataset(df, type_of_dataset):
  '''
    The function 'create_dataset' takes 2 paramaeters:
      df = The dataframe of the dataset type.
      type_of_dataset = The type of dataset

    It returns list/s of preprocessed questions and/or answers base
  '''
  if type_of_dataset=='test':
    questions = []
    for i in tqdm(range(len(df['Question']))):
      questions.append(preprocess_qa(list(df['Question'])[i]))
    return questions

  else:
    questions = []
    answers = []
    for i in tqdm(range(len(df['Question']))):
      questions.append(preprocess_qa(list(df['Question'])[i]))
      answers.append(preprocess_qa(list(df['Answer'])[i]))
    return questions,answers
```

In [ ]:
```
train_questions, train_answers = create_dataset(train,'train')
print("\n",train_questions[-1])
print(train_answers[-1])
```

```
100%|██████████| 39464/39464 [03:58<00:00, 165.69it/s]


 <start> when is the memo dated on ? <end>
<start> october 22 , 1970 <end>
```

In [ ]:
```
val_questions, val_answers = create_dataset(val,'val')
print("\n",val_questions[-1])
print(val_answers[-1])
```

```
100%|██████████| 5352/5352 [00:04<00:00, 1251.08it/s]


 <start> what is the amount of deposit ? <end>
<start> 100 . 00 <end>
```

In [ ]:
```
test_questions = create_dataset(test, 'test')
print("\n",test_questions[-1])
```

```
100%|██████████| 5188/5188 [00:02<00:00, 2469.36it/s]


 <start> what is the p . o . box number ? <end>
```

In [ ]:
```
def tokenize(tr_inp_q, tr_inp_a, v_inp_q, v_inp_a, te_inp_q):
  '''
    The function "tokenize" takes 5 parameters :
      tr_inp_q = list of the questions corresponding to the train c
      tr_inp_a = list of the answers corresponding to the train dat
      v_inp_q = list of the questions corresponding to the validati
      v_inp_a = list of the answers corresponding to the validation
      te_inp_q = list of the questions corresponding to the test da
    It returns :
      tr_q_tensor = padded tokenized sequences of the train questio
      tr_a_tensor = padded tokenized sequences of the train answers
      v_q_tensor = padded tokenized sequences of the validation que
      v_a_tensor  = padded tokenized sequences of the validation an
      tokenizer = tokenizer which has been fit on all the tokens in
  '''
  tr_q = np.array(tr_inp_q)
  tr_a = np.array(tr_inp_a)
  v_q = np.array(v_inp_q)
  v_a = np.array(v_inp_a)

  tr_inp_q.extend(tr_inp_a)
  del tr_inp_a
  tr_inp_q.extend(v_inp_q)
  del v_inp_q
  tr_inp_q.extend(v_inp_a)
  del v_inp_a
  tr_inp_q.extend(te_inp_q)
  del te_inp_q
  #Fitting the tokenizer on texts with all the unsupervised tokens
  tokenizer = tf.keras.preprocessing.text.Tokenizer(
      filters='')
  tokenizer.fit_on_texts(tr_inp_q)

  #========== Train ==========
  #Replacing text with tokens in the data
  tr_q_tensor = tokenizer.texts_to_sequences(tr_q)
  #Padding the tokenized sequences
  tr_q_tensor = tf.keras.preprocessing.sequence.pad_sequences(tr_q_
                                                         padding='p

  #Replacing text with tokens in the data
  tr_a_tensor = tokenizer.texts_to_sequences(tr_a)
  #Padding the tokenized sequences
  tr_a_tensor = tf.keras.preprocessing.sequence.pad_sequences(tr_a_
                                                         padding='p

  #========== Validation ==========
  #Replacing text with tokens in the data
  v_q_tensor = tokenizer.texts_to_sequences(v_q)
  #Padding the tokenized sequences with padding size equal to trair
  v_q_tensor = tf.keras.preprocessing.sequence.pad_sequences(v_q_te
                                                             maxler
                                                         padding='p

  #Replacing text with tokens in the data
  v_a_tensor = tokenizer.texts_to_sequences(v_a)
  #Padding the tokenized sequences with padding size equal to trair
  v_a_tensor = tf.keras.preprocessing.sequence.pad_sequences(v_a_te
                                                             maxler
                                                         padding='p

  return tr_q_tensor, tr_a_tensor, v_q_tensor, v_a_tensor, tokenize
```

In [ ]:

```
# Tokenizing the Train and Validation's, Questions, Answers and OCF
train_input_tensor, train_target_tensor, val_input_tensor, val_targ
                                             train_answers,
                                             val_questions,
                                             val_answers,
                                             test_questions)
del train_questions, train_answers, val_questions, val_answers, tes
```

In [ ]:

```
# These results are inclusive of the unsupervised tokens.
print("Number of unique words in questions and answers :",len(text.
```

```
Number of unique words in questions and answers : 23482
```

In [ ]:

```
def convert(lang, tensor):
  for t in tensor:
    if t!=0:
      print ("%d ----> %s" % (t, lang.index_word[t]))
```

In [ ]:

```
print ("Input Text; index to word mapping")
convert(text, train_input_tensor[0])
print ()
print ("Target Text; index to word mapping")
convert(text, train_target_tensor[0])
```

```
Input Text; index to word mapping
1 ----> <start>
6 ----> what
5 ----> is
3 ----> the
123 ----> project
85 ----> #
16 ----> number
4 ----> ?
2 ----> <end>

Target Text; index to word mapping
1 ----> <start>
```

The .bin which was downloaded earlier from the FastText website (https://fasttext.cc/) contains the model weigths of the fasttext model.

This model creates embedding for any type of text in the universe.

So, in this way I am creating an embedding matrix of the embeddings of all the tokens where each index represents the embedding array of length 300.

In [ ]:

```
#importing fasttext model
model = fasttext.load_model("crawl-300d-2M-subword.bin")
```

```
Warning : `load_model` does not return WordVectorModel or Supervi
sedModel any more, but a `FastText` object which is very similar.
```

In [ ]:

```
#Initiating the embedding matrix with respect to the tokens in the
embedding_matrix = np.zeros((len(text.word_index)+1, 300))
for word, i in text.word_index.items():
    embedding_vector = model[word]
    embedding_matrix[i] = embedding_vector
```

In [ ]:

```
# Deleting the model to save RAM
del model
```

4.1.1.3. Images.

I have used InceptionV3 (which is pretrained on Imagenet) to classify each document image. I have extracted features from the last convolutional layer.

At First, I converted the images into InceptionV3's expected format by:

- Resizing the image to 299px by 299px.
- Preprocessing the images using the preprocess_input method to normalize the image so that it contains pixels in the range of -1 to 1, which matches the format of the images used to train InceptionV3.

In [ ]:

```
def load_image(image_path):
    img = tf.io.read_file(image_path)
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.image.resize(img, (299, 299))
    img = tf.keras.applications.inception_v3.preprocess_input(img)
    return img, image_path
```

Now I created a tf.keras model where the output layer is the last convolutional layer in the InceptionV3 architecture. The shape of the output of this layer is 8x8x2048. I used the last convolutional layer because I was using attention in the baseline architecture.

I forwarded each image through the network and stored the resulting vector in a dictionary (image_name --> feature_vector).

In [ ]:

```
image_model = tf.keras.applications.InceptionV3(include_top=False,
                                                weights='imagenet')
new_input = image_model.input
hidden_layer = image_model.layers[-1].output

image_features_extract_model = tf.keras.Model(new_input, hidden_lay
```

```
Downloading data from https://storage.googleapis.com/tensorflow/k
eras-applications/inception_v3/inception_v3_weights_tf_dim_orderi
ng_tf_kernels_notop.h5 (https://storage.googleapis.com/tensorflow
/keras-applications/inception_v3/inception_v3_weights_tf_dim_orde
ring_tf_kernels_notop.h5)
87916544/87910968 [==============================] - 1s 0us/step
```

I pre-processed each image with InceptionV3 and cached the output to the disk. Caching the output in RAM would be faster but also memory intensive, requiring 8 * 8 * 2048 floats per image.

In [ ]:

```
# Get unique images of Train and Validation
encode_train = list(train['Image'])
encode_val = list(val['Image'])

train_image_dataset = tf.data.Dataset.from_tensor_slices(encode_tra
del encode_train
train_image_dataset = train_image_dataset.map(
  load_image, num_parallel_calls=tf.data.experimental.AUTOTUNE).bat

val_image_dataset = tf.data.Dataset.from_tensor_slices(encode_val)
del encode_val
val_image_dataset = val_image_dataset.map(
  load_image, num_parallel_calls=tf.data.experimental.AUTOTUNE).bat

for img, path in tqdm(train_image_dataset):
  batch_features = image_features_extract_model(img)
  batch_features = tf.reshape(batch_features,
                              (batch_features.shape[0], -1, batch_f

  for bf, p in zip(batch_features, path):
    path_of_feature = p.numpy().decode("utf-8")
    np.save(path_of_feature, bf.numpy())

for img, path in tqdm(val_image_dataset):
  batch_features = image_features_extract_model(img)
  batch_features = tf.reshape(batch_features,
                              (batch_features.shape[0], -1, batch_f

  for bf, p in zip(batch_features, path):
    path_of_feature = p.numpy().decode("utf-8")
    np.save(path_of_feature, bf.numpy())
```

```
del train_image_dataset
del val_image_dataset
```

```
100%|██████████| 2467/2467 [13:18<00:00,  3.09it/s]
100%|██████████| 335/335 [01:35<00:00,  3.53it/s]
```

In [ ]:

```
train_img_to_cap_vector_inp = collections.defaultdict(list)
train_img_to_cap_vector_tar = collections.defaultdict(list)
val_img_to_cap_vector_inp = collections.defaultdict(list)
val_img_to_cap_vector_tar = collections.defaultdict(list)


for img, inp, tar in zip(train['Image'], train_input_tensor, train_
  train_img_to_cap_vector_inp[img].append(inp)
  train_img_to_cap_vector_tar[img].append(tar)
for img, inp, tar in zip(val['Image'], val_input_tensor, val_target
  val_img_to_cap_vector_inp[img].append(inp)
  val_img_to_cap_vector_tar[img].append(tar)


train_img_keys = list(train_img_to_cap_vector_inp.keys())
random.shuffle(train_img_keys)
val_img_keys = list(val_img_to_cap_vector_inp.keys())
random.shuffle(val_img_keys)

img_name_train = []
input_train = []
target_train = []
for imgt in train_img_keys:
  inp_len = len(train_img_to_cap_vector_inp[imgt])
  img_name_train.extend([imgt] * inp_len)
  input_train.extend(train_img_to_cap_vector_inp[imgt])
  target_train.extend(train_img_to_cap_vector_tar[imgt])

img_name_val = []
input_val = []
target_val = []
for imgt in val_img_keys:
  inp_len = len(val_img_to_cap_vector_inp[imgt])
  img_name_val.extend([imgt] * inp_len)
  input_val.extend(val_img_to_cap_vector_inp[imgt])
  target_val.extend(val_img_to_cap_vector_tar[imgt])


del train_img_to_cap_vector_inp
del train_img_to_cap_vector_tar
del train_img_keys
del val_img_to_cap_vector_inp
del val_img_to_cap_vector_tar
del val_img_keys
```

In [ ]:

```
len(img_name_train), len(input_train), len(target_train)
```

Out[25]:

```
(39464, 39464, 39464)
```

In [ ]:

```
len(img_name_val), len(input_val), len(target_val)
```

Out[26]:

```
(5352, 5352, 5352)
```

4.1.1.4. Creating tf.data dataset for training.

In [ ]:

```
# Load the numpy files
def map_func(img_name, inp, targ):
  img_tensor = np.load(img_name.decode('utf-8')+'.npy')
  return img_tensor, inp, targ
```

In [ ]:

```
train_dataset = tf.data.Dataset.from_tensor_slices((img_name_train,
                                                    input_train,
                                                    target_train))
val_dataset = tf.data.Dataset.from_tensor_slices((img_name_val,
                                                  input_val,
                                                  target_val))
```

In [ ]:

```
# Creating tf.data Dataset
TRAIN_BUFFER_SIZE = len(train_input_tensor)
VAL_BUFFER_SIZE = len(val_input_tensor)
max_length_inp = train_input_tensor.shape[1]
max_length_targ = train_target_tensor.shape[1]
del train_input_tensor
del train_target_tensor
del val_input_tensor
del val_target_tensor
BATCH_SIZE = 8

#========== Train ==========
# map to load the numpy files in parallel
train_dataset = train_dataset.map(lambda item1, item2, item3: tf.nu
                                                  [item1, item2, item3
                                                  [tf.float32, tf.int3
                      num_parallel_calls=tf.data.experimental.AUTOT

# Shuffle and batch
train_dataset = train_dataset.shuffle(TRAIN_BUFFER_SIZE).batch(BATC
train_dataset = train_dataset.prefetch(buffer_size=tf.data.experime


#========== Validation ==========
# map to load the numpy files in parallel
val_dataset = val_dataset.map(lambda item1, item2, item3: tf.numpy_
                                                  [item1, item2, item3
                                                  [tf.float32, tf.int3
                      num_parallel_calls=tf.data.experimental.AUTOT

# Shuffle and batch
val_dataset = val_dataset.shuffle(VAL_BUFFER_SIZE).batch(BATCH_SIZE
val_dataset = val_dataset.prefetch(buffer_size=tf.data.experimental
```

In [ ]:

```
example_input_img, example_input_batch, example_target_batch = next
example_input_img.shape, example_input_batch.shape, example_target
```

Out[30]:

```
(TensorShape([8, 64, 2048]), TensorShape([8, 41]), TensorShape
([8, 39]))
```

In [ ]:

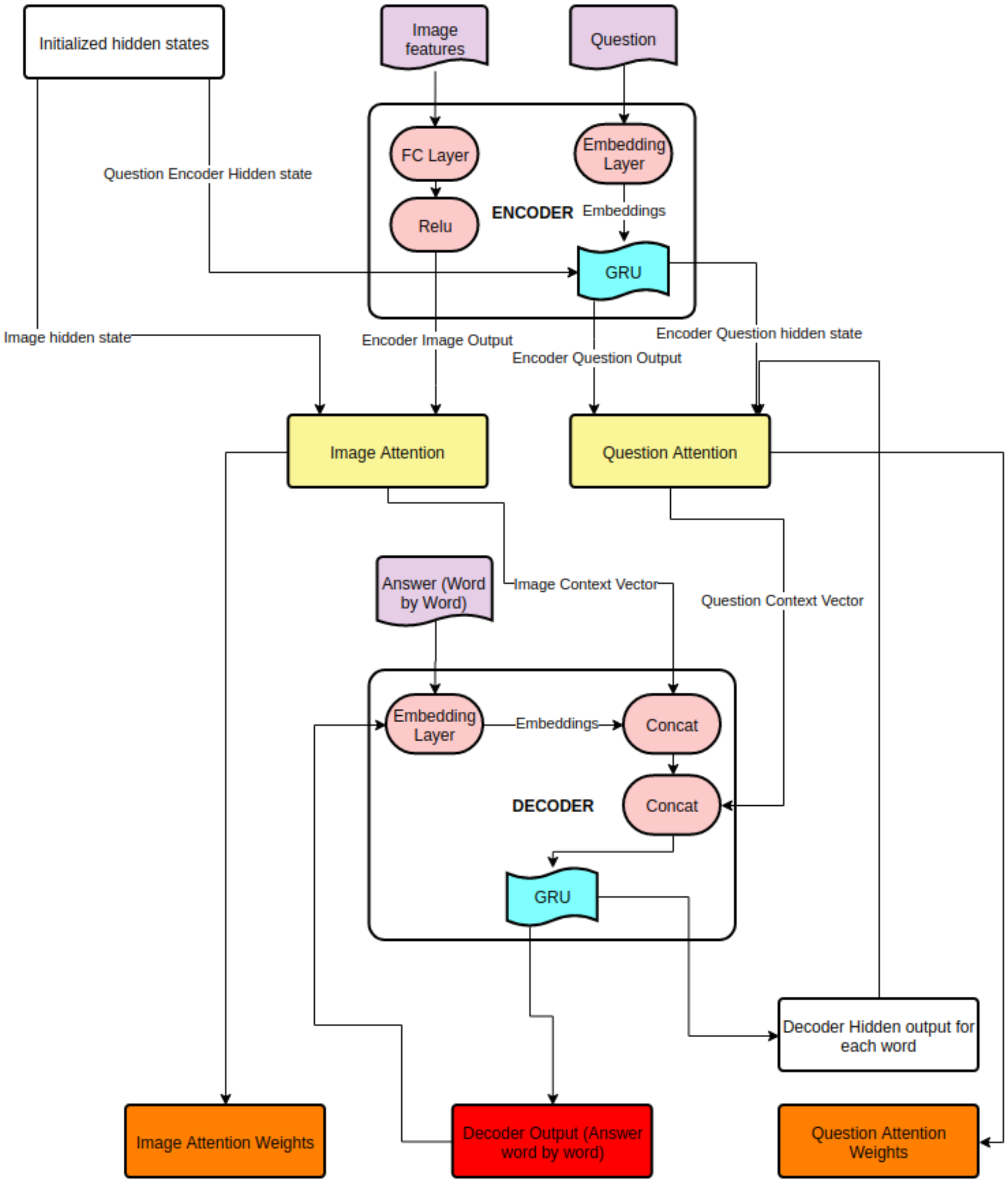
```
example_input_img, example_input_batch, example_target_batch = next
example_input_img.shape, example_input_batch.shape, example_target
```

Out[31]:

```
(TensorShape([8, 64, 2048]), TensorShape([8, 41]), TensorShape
([8, 39]))
```

4.1.2. Model.

4.1.2.1. Model Architecture.



4.1.2.2. Encoder.

In [ ]:

```
class Encoder(tf.keras.Model):
  def __init__(self, vocab_size, embedding_dim, enc_units, batch_sz
    super(Encoder, self).__init__()
    self.batch_sz = batch_sz
    self.enc_units = enc_units
    self.embedding = tf.keras.layers.Embedding(vocab_size,
                                               embedding_dim,
                                               weights=[inp_embed_m
                                               trainable=False)
    self.question_gru = tf.keras.layers.GRU(self.enc_units,
                                            return_sequences=True,
                                            return_state=True,
                                            recurrent_initializer='
    self.fc = tf.keras.layers.Dense(embedding_dim)

  def call(self, image, question, hidden):
    #==================== Image Encoding ====================
    #Squashing the output shape of the InceptionV3 model to (batch_
    image = self.fc(image)
    #features shape after passing through the Dense Layer == (batch
    image = tf.nn.relu(image)

    #==================== Question Encoding ====================
    # question shape after passing through embedding == (batch_size
    question = self.embedding(question)
    # output shape == (batch_size, input_length(41), gru_size(enc_u
    # state shape == (batch_size, gru_size(enc_units))
    question_output, question_state = self.question_gru(question, i

    return image, question_output, question_state

  def initialize_hidden_state(self):
    return tf.zeros((self.batch_sz, self.enc_units))
```

In [ ]:

```
def grader_check_encoder():
    vocab_size=len(text.word_index)+1
    embedding_size=300
    lstm_size=128
    input_length_q=41
    input_length_o=2293
    batch_size=BATCH_SIZE
    encoder=Encoder(vocab_size,
                    embedding_size,
                    lstm_size,
                    batch_size,
                    embedding_matrix)
    input_question=tf.random.uniform(shape=[batch_size,input_length
                                     maxval=vocab_size,minval=0,
                                     dtype=tf.int32)
    input_image=tf.random.uniform(shape=[batch_size,
                                        64,
                                        2048],
                                 maxval=vocab_size,minval=0,dtype=t
    initial_state=encoder.initialize_hidden_state()

    encoder_output_img,encoder_output_q,state_q=encoder(input_image
                                                        input_quest
                                                        initial_sta
    assert(encoder_output_q.shape==(batch_size,input_length_q,lstm_
           encoder_output_img.shape==(batch_size,64,embedding_size)
           state_q.shape==(batch_size,lstm_size))
    return True
print(grader_check_encoder())
```

```
True
```

4.1.2.3. Attention.

In [ ]:

```
class Image_Attention(tf.keras.Model):
  def __init__(self, units):
    super(Image_Attention, self).__init__()
    self.W1 = tf.keras.layers.Dense(units)
    self.W2 = tf.keras.layers.Dense(units)
    self.V = tf.keras.layers.Dense(1)

  def call(self, features, hidden):
    # features(CNN_encoder output) shape == (batch_size, 64, embedd

    # hidden shape == (batch_size, hidden_size)
    # hidden_with_time_axis shape == (batch_size, 1, hidden_size)
    hidden_with_time_axis = tf.expand_dims(hidden, 1)

    # attention_hidden_layer shape == (batch_size, 64, units)
    attention_hidden_layer = (tf.nn.tanh(self.W1(features) +
                                         self.W2(hidden_with_time_a

    # score shape == (batch_size, 64, 1)
    # This gives you an unnormalized score for each image feature.
    score = self.V(attention_hidden_layer)

    # attention_weights shape == (batch_size, 64, 1)
    attention_weights = tf.nn.softmax(score, axis=1)

    # context_vector shape after sum == (batch_size, hidden_size)
    context_vector = attention_weights * features
    context_vector = tf.reduce_sum(context_vector, axis=1)

    return context_vector, attention_weights
```

In [ ]:
```
class Question_Attention(tf.keras.layers.Layer):
  def __init__(self, units):
    super(Question_Attention, self).__init__()
    self.W1 = tf.keras.layers.Dense(units)
    self.W2 = tf.keras.layers.Dense(units)
    self.V = tf.keras.layers.Dense(1)

  def call(self, query, values):
    # query hidden state shape == (batch_size, question hidden size
    # query_with_time_axis shape == (batch_size, 1, hidden size)
    # values shape == (batch_size, max_len, question hidden size)
    # we are doing this to broadcast addition along the time axis t
    query_with_time_axis = tf.expand_dims(query, 1)

    # score shape == (batch_size, max_length, 1)
    # we get 1 at the last axis because we are applying score to se
    # the shape of the tensor before applying self.V is (batch_size
    score = self.V(tf.nn.tanh(
        self.W1(query_with_time_axis) + self.W2(values)))

    # attention_weights shape == (batch_size, max_length, 1)
    attention_weights = tf.nn.softmax(score, axis=1)

    # context_vector shape after sum == (batch_size, question_hidde
    context_vector = attention_weights * values
    context_vector = tf.reduce_sum(context_vector, axis=1)

    return context_vector, attention_weights
```

4.1.2.4. Decoder.

In [ ]:

```
class Decoder(tf.keras.Model):
  def __init__(self, vocab_size, embedding_dim, dec_units, batch_sz
    super(Decoder, self).__init__()
    self.batch_sz = batch_sz
    self.dec_units = dec_units
    self.embedding = tf.keras.layers.Embedding(vocab_size,
                                               embedding_dim,
                                               weights=[targ_embed_
                                               trainable=False)
    self.gru = tf.keras.layers.GRU(self.dec_units,
                                   return_sequences=True,
                                   return_state=True,
                                   recurrent_initializer='glorot_ur
    self.fc = tf.keras.layers.Dense(vocab_size)

    # used for attention
    self.question_attention = Question_Attention(self.dec_units)
    self.image_attention = Image_Attention(self.dec_units)

  def call(self, x, hidden_q, hidden_i, enc_output_q, enc_output_i)
    #==================== Question Attention ====================
    # defining question attention as a separate model
    # enc_output shape == (batch_size, input_length(41), hidden_siz
    context_vector_q, attention_weights_q = self.question_attention


    #==================== Image Attention ====================
    # defining image attention as a separate model
    context_vector_i, attention_weights_i = self.image_attention(en


    #==================== Concat Image Context Vector ============
    # Here x represents Decoder Input
    # x shape after passing through embedding == (batch_size, 1, em
    x = self.embedding(x)
    # x shape after concatenation == (batch_size, 1, embedding_dim
    x = tf.concat([tf.expand_dims(context_vector_i, 1), x], axis=-1


    #==================== Concat Question Context Vector =========
    # x shape after concatenation == (batch_size, 1, embedding_dim
    x = tf.concat([tf.expand_dims(context_vector_q, 1), x], axis=-1


    #==================== Decoding Answer ====================
    # passing the concatenated vector to the GRU
    output, state = self.gru(x)

    # output shape == (batch_size * 1, hidden_size)
    output = tf.reshape(output, (-1, output.shape[2]))

    # output shape == (batch_size, vocab)
    x = self.fc(output)

    return x, state, attention_weights_q, attention_weights_i

  def reset_state(self, batch_size):
    return tf.zeros((batch_size, self.dec_units))
```

In [ ]:
```
def grader_check_decoder():
    vocab_size=len(text.word_index)+1
    embedding_size=300
    lstm_size=32
    input_length_q=example_input_batch.shape[1]
    targ_length=example_target_batch.shape[1]
    batch_size=BATCH_SIZE

    encoder=Encoder(vocab_size,
                    embedding_size,
                    lstm_size,
                    batch_size,
                    embedding_matrix)

    input_question=tf.random.uniform(shape=[batch_size,input_length
                                    maxval=vocab_size,
                                    minval=0,
                                    dtype=tf.int32)
    input_image=tf.random.uniform(shape=[batch_size,
                                         example_input_img.shape[1]
                                         example_input_img.shape[2]
                                  maxval=vocab_size,
                                  minval=0,
                                  dtype=tf.float32)
    initial_state=encoder.initialize_hidden_state()
    encoder_output_img,encoder_output_q,state_q=encoder(input_image
                                                         input_quest
                                                         initial_sta

    decoder = Decoder(vocab_size,
                     embedding_size,
                     lstm_size,
                     batch_size,
                     embedding_matrix)

    image_hidden = decoder.reset_state(batch_size)

    sample_decoder_output, state, attention_weights_q, attention_we





    assert(sample_decoder_output.shape==(batch_size,vocab_size) and
           state.shape==(batch_size,lstm_size) and
           attention_weights_q.shape==(batch_size,example_input_bat
           attention_weights_i.shape==(batch_size,example_input_img
    return True
print(grader_check_decoder())
```

```
True
```

In [ ]:
```
#del example_input_batch, example_input_img, example_target_batch
```

4.1.2.5. Optimizer and loss function.

In [ ]:
```
vocab_size=len(text.word_index)+1
```

```
steps_per_epoch_train=len(input_train)//BATCH_SIZE
steps_per_epoch_val=len(input_val)//BATCH_SIZE
embedding_dim=300
units=128
batch_size=BATCH_SIZE
attention_features_shape = 64
```

In [ ]:

```
encoder = Encoder(vocab_size,
                  embedding_dim,
                  units,
                  batch_size,
                  embedding_matrix)
decoder = Decoder(vocab_size,
                  embedding_dim,
                  units,
                  batch_size,
                  embedding_matrix)
```

In [ ]:

```
optimizer = tf.keras.optimizers.Adam()
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_lo

def loss_function(real, pred):
  mask = tf.math.logical_not(tf.math.equal(real, 0))
  loss_ = loss_object(real, pred)

  mask = tf.cast(mask, dtype=loss_.dtype)
  loss_ *= mask

  return tf.reduce_mean(loss_)
```

4.1.2.6. Training

Steps involved while training :

1. Pass the input(Inceptionv3 image features and tokens of question) through the encoder. The encoder then returns encoder outputs(image and question) and the encoder question hidden state.
2. The encoder outputs, encoder question hidden state, encoder image hidden state and the decoder input (which is the start token) is passed to the decoder.
3. The decoder returns the predictions and the decoder hidden state.
4. The decoder hidden state is then passed back into the model and the predictions are used to calculate the loss.
5. teacher forcing method is used to decide the next input to the decoder.
6. Teacher forcing is the technique where the target word is passed as the next input to the decoder.
7. The final step is to calculate the gradients and apply it to the optimizer and backpropagate.

In [ ]:

```
checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
checkpoint = tf.train.Checkpoint(optimizer=optimizer,
                                 encoder=encoder,
                                 decoder=decoder)
```

In [ ]:

```
@tf.function
def train_step(train_inp_image, train_inp_quest, train_targ_ans, tr
  train_loss = 0
  # initializing the hidden state for each batch
  # because the ANSWERS as well as QUESTIONS are not related from i
  train_img_hidden = decoder.reset_state(batch_size=train_inp_image

  with tf.GradientTape() as tape:
    train_enc_output_i, train_enc_output_q, train_enc_hidden_q = en



    train_dec_hidden = train_enc_hidden_q

    train_dec_input = tf.expand_dims([text.word_index['<start>']] *

    # Teacher forcing - feeding the target as the next input
    for t in range(1, train_targ_ans.shape[1]):
      # passing encoder_output(enc_output) and image_features(image
      train_predictions, train_dec_hidden, _, _ = decoder(train_dec
                                                          train_dec
                                                          train_img
                                                          train_enc
                                                          train_enc

      train_loss += loss_function(train_targ_ans[:, t], train_predi

      # using teacher forcing
      train_dec_input = tf.expand_dims(train_targ_ans[:, t], 1)

  train_batch_loss = (train_loss / int(train_targ_ans.shape[1]))

  train_variables = encoder.trainable_variables + decoder.trainable

  train_gradients = tape.gradient(train_loss, train_variables)

  optimizer.apply_gradients(zip(train_gradients, train_variables))

  return train_batch_loss
```

In [ ]:

```
@tf.function
def cv_step(inp_image, inp_quest, targ_ans, enc_hidden):
  loss = 0

  # initializing the hidden state for each batch
  # because the ANSWERS as well as QUESTIONS are not related from i
  img_hidden = decoder.reset_state(batch_size=inp_image.shape[0])

  enc_output_i, enc_output_q, enc_hidden_q = encoder(inp_image,
                                                     inp_quest,
                                                     enc_hidden)

  dec_hidden = enc_hidden_q

  dec_input = tf.expand_dims([text.word_index['<start>']] * BATCH_S

  for t in range(1, targ_ans.shape[1]):
    # passing encoder_output(enc_output) and image_features(image_i
    predictions, dec_hidden, _, _ = decoder(dec_input,
                                              dec_hidden,
                                              img_hidden,
                                              enc_output_q,
                                              enc_output_i)
    loss += loss_function(targ_ans[:, t], predictions)

    # using teacher forcing
    dec_input = tf.expand_dims(targ_ans[:, t], 1)

  batch_loss = (loss / int(targ_ans.shape[1]))

  return batch_loss
```

In [ ]:

```
current_time = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
train_log_dir = 'logs/gradient_tape/' + current_time + '/train'
train_summary_writer = tf.summary.create_file_writer(train_log_dir)
```

In [ ]:

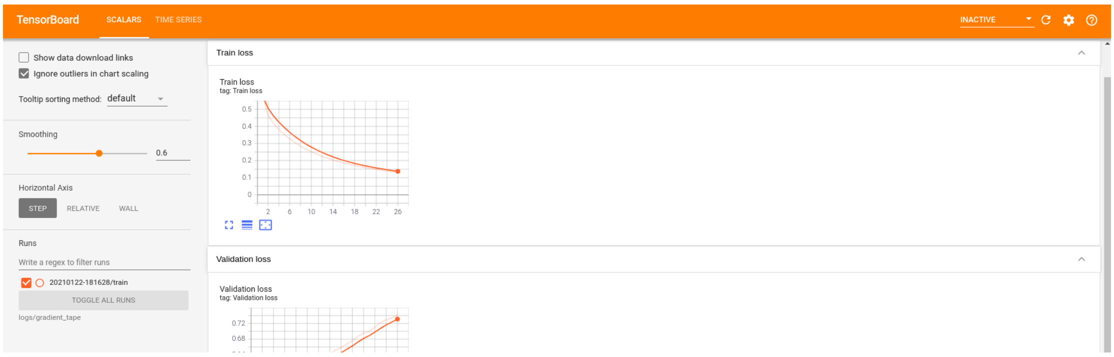
```
EPOCHS = 26
# http://teleported.in/posts/cyclic-learning-rate/
# https://github.com/bckenstler/CLR/blob/master/clr_callback_tests.

for epoch in range(EPOCHS):
  start = time.time()

  enc_hidden = encoder.initialize_hidden_state()

  train_total_loss = 0
  val_total_loss = 0
  for train_image_inp, train_inp_q, train_targ_a in train_dataset:
    train_batch_loss = train_step(train_image_inp, train_inp_q, tra
    train_total_loss += train_batch_loss

  for val_image_inp, val_inp_q, val_targ_a in val_dataset:
    val_batch_loss = cv_step(val_image_inp, val_inp_q, val_targ_a,
    val_total_loss += val_batch_loss

  # saving (checkpoint) the model every even numbered epochs
  if (epoch + 1) % 2 == 0:
    checkpoint.save(file_prefix = checkpoint_prefix)

  #Savings logs for tensorboard
  with train_summary_writer.as_default():
    tf.summary.scalar('Train loss', train_total_loss/steps_per_epoc
    tf.summary.scalar('Validation loss', val_total_loss/steps_per_e

  if (epoch + 1) % 2 == 0:
    print('Epoch {} ====> Train Loss : {:.4f}, Validation loss : {:



  else:
    print('Epoch {} ====> Train Loss : {:.4f}, Validation loss : {:
```

```
Epoch 1 ====> Train Loss : 0.5780, Validation loss : 0.5484, Time
taken : 699.7667 seconds
Epoch 2 ====> Train Loss : 0.4634, Validation loss : 0.5367, Time
taken : 642.8017 seconds (Checkpoint saved)
Epoch 3 ====> Train Loss : 0.4179, Validation loss : 0.5425, Time
taken : 640.3573 seconds
Epoch 4 ====> Train Loss : 0.3821, Validation loss : 0.5458, Time
taken : 631.7270 seconds (Checkpoint saved)
Epoch 5 ====> Train Loss : 0.3521, Validation loss : 0.5515, Time
taken : 627.7188 seconds
Epoch 6 ====> Train Loss : 0.3262, Validation loss : 0.5612, Time
taken : 633.0728 seconds (Checkpoint saved)
Epoch 7 ====> Train Loss : 0.3040, Validation loss : 0.5760, Time
taken : 644.1986 seconds
Epoch 8 ====> Train Loss : 0.2843, Validation loss : 0.5799, Time
taken : 650.4563 seconds (Checkpoint saved)
Epoch 9 ====> Train Loss : 0.2667, Validation loss : 0.5933, Time
taken : 633.6469 seconds
Epoch 10 ====> Train Loss : 0.2513, Validation loss : 0.6040, Tim
e taken : 632.0195 seconds (Checkpoint saved)
Epoch 11 ====> Train Loss : 0.2379, Validation loss : 0.6117, Tim
e taken : 629.2547 seconds
Epoch 12 ====> Train Loss : 0.2247, Validation loss : 0.6199, Tim
e taken : 620.0600 seconds (Checkpoint saved)
Epoch 13 ====> Train Loss : 0.2137, Validation loss : 0.6303, Tim
e taken : 626.9388 seconds
Epoch 14 ====> Train Loss : 0.2032, Validation loss : 0.6399, Tim
e taken : 647.7314 seconds (Checkpoint saved)
Epoch 15 ====> Train Loss : 0.1942, Validation loss : 0.6519, Tim
e taken : 634.3432 seconds
Epoch 16 ====> Train Loss : 0.1858, Validation loss : 0.6582, Tim
e taken : 638.3561 seconds (Checkpoint saved)
Epoch 17 ====> Train Loss : 0.1783, Validation loss : 0.6678, Tim
e taken : 663.7941 seconds
Epoch 18 ====> Train Loss : 0.1712, Validation loss : 0.6756, Tim
e taken : 662.7872 seconds (Checkpoint saved)
Epoch 19 ====> Train Loss : 0.1648, Validation loss : 0.6872, Tim
e taken : 652.1875 seconds
Epoch 20 ====> Train Loss : 0.1587, Validation loss : 0.6957, Tim
e taken : 622.4939 seconds (Checkpoint saved)
Epoch 21 ====> Train Loss : 0.1531, Validation loss : 0.6997, Tim
e taken : 626.0968 seconds
Epoch 22 ====> Train Loss : 0.1479, Validation loss : 0.7110, Tim
e taken : 668.3430 seconds (Checkpoint saved)
Epoch 23 ====> Train Loss : 0.1421, Validation loss : 0.7218, Tim
```

In [ ]:

```
%load_ext tensorboard
%tensorboard --logdir logs/gradient_tape
```

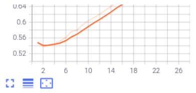
```
<IPython.core.display.Javascript object>
```



□ Show data download links
☑ Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing 0.6
Horizontal Axis: STEP RELATIVE WALL
Runs
Write a regex to filter runs
☑ ○ 20210122-181628/train
logs/gradient_tape

Train loss
tag: Train loss

Validation loss
tag: Validation loss



4.1.3. Attention Plot.

In [ ]:
```
# restoring the latest checkpoint in checkpoint_dir
print("checkpoint directory : ", checkpoint_dir)
checkpoint.restore(tf.train.latest_checkpoint(checkpoint_dir))
```

```
checkpoint directory :  ./training_checkpoints
```

Out[48]:
```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at
0x7ff56f7fac18>
```

In [ ]:
```
def evaluate(sentence, image):
  '''
    The function 'evaluate' takes 2 parameters:
      sentence = preproccessed Question tokens with <start> and <e
      image = the image features from the InceptionV3 model.

    It returns
      result=the predicted answer is in a sentence format for the
      img_result=the predicted answer is in the form of list for th
      sentence=passing the same sentence which has been passed as a
      attention_plot=these are the attention weights for the QUESTI
      image_attention_plot=these are the attention weights for the
  '''
  #=====Question Input Preprocess=====
  question_attention_plot = np.zeros((max_length_targ, max_length_i
  image_attention_plot = np.zeros((max_length_targ, attention_featu

  sentence = preprocess_qa(sentence)

  inputs = [text.word_index[i] for i in sentence.split(' ')]
  inputs = tf.keras.preprocessing.sequence.pad_sequences([inputs],
                                                         maxlen=max
                                                         padding='p
  inputs = tf.convert_to_tensor(inputs)

  result = ''
  img_result = []

  hidden = tf.zeros((1, units))

  #=====Image Input Preprocess=====
  temp_input = tf.expand_dims(load_image(image)[0], 0)
  img_tensor_val = image_features_extract_model(temp_input)
  img_tensor_val = tf.reshape(img_tensor_val, (img_tensor_val.shape

  #=====Encoder Phase=====
  enc_out_i, enc_out_q, enc_hidden_q = encoder(img_tensor_val, inpu

  dec_hidden = enc_hidden_q
  dec_input = tf.expand_dims([text.word_index['<start>']], 0)

  #=====Decoder Phase=====
  img_hidden = decoder.reset_state(batch_size=1)
  for t in range(max_length_targ):
    predictions, dec_hidden, question_attention_weights, image_atte




    # storing the attention weights to plot later on
    question_attention_weights = tf.reshape(question_attention_weig
    question_attention_plot[t] = question_attention_weights.numpy()
    image_attention_plot[t] = tf.reshape(image_attention_weights, (

    predicted_id = tf.argmax(predictions[0]).numpy()

    result += text.index_word[predicted_id] + ' '
    img_result.append(text.index_word[predicted_id] + ' ')

    if text.index_word[predicted_id] == '<end>':
      return result, img_result, sentence, question_attention_plot,

    # the predicted ID is fed back into the model
    dec_input = tf.expand_dims([predicted_id], 0)

  image_attention_plot = image_attention_plot[:len(result), :]
  return result, img_result, sentence, question_attention_plot, ima
```

In [ ]:

```
# function for plotting the question attention weights
def plot_attention(attention, sentence, predicted_sentence):
  fig = plt.figure(figsize=(10,10))
  ax = fig.add_subplot(1, 1, 1)
  ax.matshow(attention, cmap='viridis')

  fontdict = {'fontsize': 14}

  ax.set_xticklabels([''] + sentence, fontdict=fontdict, rotation=9
  ax.set_yticklabels([''] + predicted_sentence, fontdict=fontdict)

  ax.xaxis.set_major_locator(ticker.MultipleLocator(1))
  ax.yaxis.set_major_locator(ticker.MultipleLocator(1))

  plt.show()
```

In [ ]:

```
# function for plotting the image attention weights
def image_plot_attention(image, result, attention_plot):
    temp_image = np.array(Image.open(image))

    fig = plt.figure(figsize=(10, 10))

    len_result = len(result)
    for l in range(len_result):
        temp_att = np.resize(attention_plot[l], (8, 8))
        ax = fig.add_subplot(len_result, len_result, l+1)
        ax.set_title(result[l])
        img = ax.imshow(temp_image)
        ax.imshow(temp_att, cmap='gray', alpha=0.6, extent=img.get_

    plt.tight_layout()
    plt.show()
```

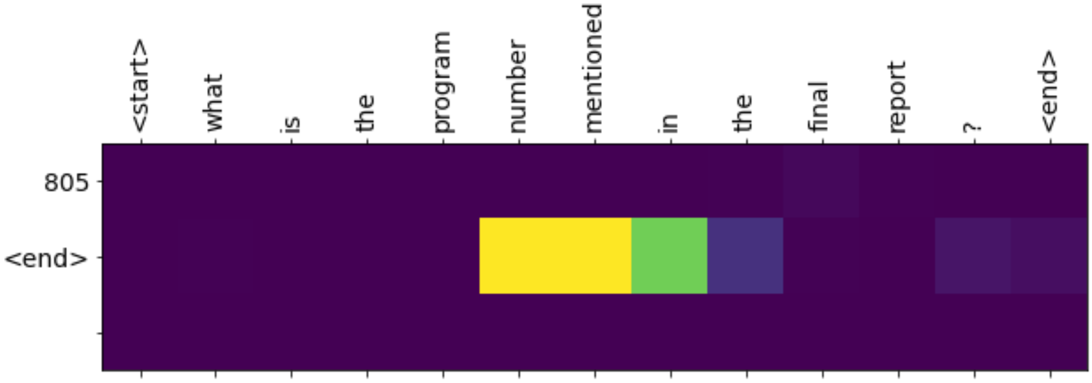
In [ ]:

```
def predict_answer(sentence,image_input):
  result, img_result, sentence, attention_plot, image_attention_plo
  print('\n\nQuestion: %s' % (sentence))
  print('Predicted Answer: {}'.format(result))

  attention_plot = attention_plot[:len(result.split(' ')), :len(ser
  plot_attention(attention_plot, sentence.split(' '), result.split(
  image_plot_attention(image_input_img, result, image_attention_plot)
```

In [ ]:

```
indices = random.sample(range(1, train.shape[0]), 20)
for idx in indices:
  predict_answer(list(train['Question'])[idx],list(train['Image'])[
  print("\nActual Answer : ",list(train['Answer'])[idx])
  print("\n\n===============================================
```

```
Question: <start> what is the program number mentioned in the fin
al report ? <end>
Predicted Answer: 805 <end>
```

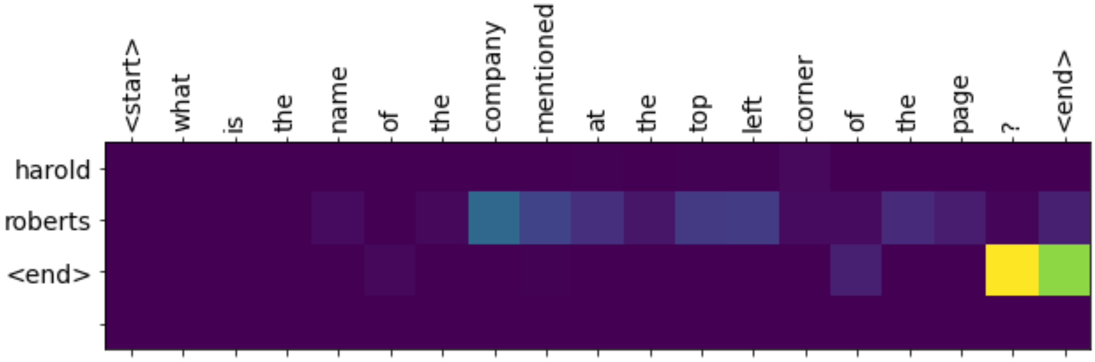


Even if most of the predicted answers are wrong, the model is still able to pertain the format of how the answers should be with respect to the questions.

In [ ]:

```
indices = random.sample(range(1, val.shape[0]), 20)
for idx in indices:
  predict_answer(list(val['Question'])[idx],list(val['Image'])[idx]
  print("\nActual Answer : ",list(val['Answer'])[idx])
  print("\n\n===============================================
```

```
Question: <start> what is the name of the company mentioned at th
e top left corner of the page ? <end>
Predicted Answer: harold roberts <end>
```



4.1.4. ANLS Score.

These scores have been calculated where the actual answers and the predicted outputs have whitespaces present between the punctuations and the characters.

In [ ]:
```
def anls_score(df, df_type):
  images_list = list(df['Image'])
  questions_list = list(df['Question'])
  _, answers_list = create_dataset(df,df_type)
  rho=0.5

  nl = NormalizedLevenshtein()
  nl_scores = []
  for image_idx in tqdm(range(len(images_list))):
    result,_,_,_,_ =  evaluate(questions_list[image_idx],images_lis

    actual_answer = answers_list[image_idx]
    actual_answer = actual_answer.replace('<end>','')
    actual_answer = actual_answer.replace('<start>','')
    actual_answer = actual_answer.strip()

    result = result
    result = result.replace('<end>','')
    result = result.strip()

    if round(nl.distance(result,actual_answer),1)<rho:
      nls = nl.similarity(result,actual_answer)
      nl_scores.append(nls)
    else:
      nls = 0.0
      nl_scores.append(nls)
  anl_score = (sum(nl_scores))/(len(nl_scores))
  return anl_score
```

In [ ]:
```
# Printing the Train ANLS score
print("\nTrain ANLS Score :",anls_score(train,'train'))
```

```
100%|██████████| 39464/39464 [04:07<00:00, 159.41it/s]
100%|██████████| 39464/39464 [3:24:39<00:00,  3.21it/s]


Train ANLS Score : 0.06073730708515422
```

In [ ]:
```
# Printing the Validation ANLS score
print("\nTest ANLS Score :",anls_score(val,'val'))
```

```
100%|██████████| 5352/5352 [00:04<00:00, 1209.22it/s]
100%|██████████| 5352/5352 [28:38<00:00,  3.11it/s]


Test ANLS Score : 0.018013496713653238
```

4.1.5. Predict Test answers

In [ ]:
```
def predict(df):
  images_list = list(df['Image'])
  questions_list = list(df['Question'])
  results = []
  for image_idx in tqdm(range(len(images_list))):
    result,_,_,_,_ =  evaluate(questions_list[image_idx],images_lis
    results.append(result)
  return results
```

In [ ]:
```
lst = predict(test)
```

```
100%|██████████| 5188/5188 [24:00<00:00,  3.60it/s]
```

In [ ]:
```
pred = pd.DataFrame(lst)
pred.head()
```

In [ ]:
```
pred.to_csv('/content/drive/MyDrive/Data/test_results_1.csv', index
```

4.1.6. Submitting results.

In [ ]:
```
answer_df = pd.read_csv('/content/drive/MyDrive/Data/test_results_1
test = pd.read_csv('/content/drive/MyDrive/Data/test.csv')
```

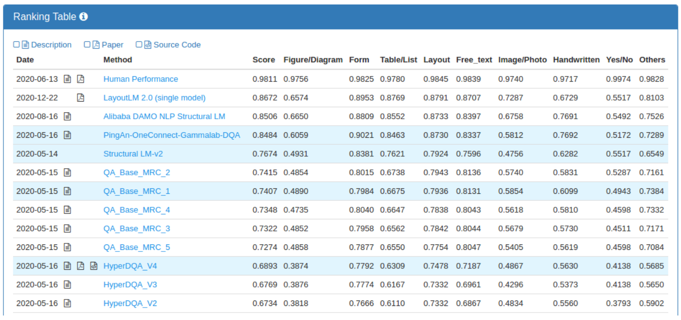
In [ ]:
```
results_list = []
for i in tqdm(range(len(test['Question_Id']))):
  temp_dict = {}
  temp_answer = list(answer_df['0'])[i]
  temp_answer = temp_answer.replace('<end>','')
  temp_answer = ' '.join(temp_answer.split())
  temp_dict['answer'] = temp_answer
  temp_dict['questionId'] = list(test['Question_Id'])[i]
  results_list.append(temp_dict)
```

```
100%|██████████| 5188/5188 [00:04<00:00, 1194.63it/s]
```

In [ ]:
```
# Writing to sample.json
with open("/content/drive/MyDrive/Data/model_1_result.json", "w") a
    json.dump(results_list, outfile)
```

4.1.7. Test Result.

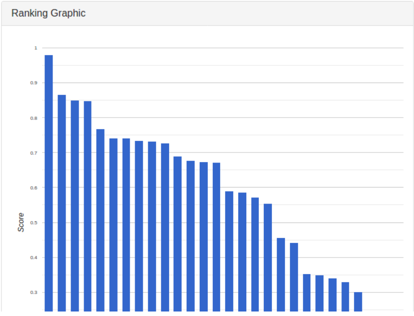
The baseline model achieved a Test ANLS score of 0.0552 after submitting the results on the RRC (https://rrc.cvc.uab.es/?ch=17&com=evaluation&task=1) website.

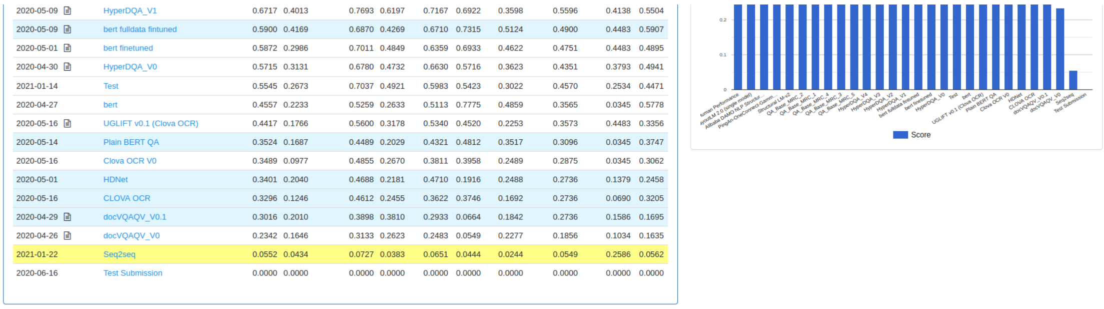


Ranking Table

□ Description □ Paper □ Source Code

Date	Method	Score	Figure/Diagram	Form	Table/List	Layout	Free_text	Image/Photo	Handwritten	Yes/No	Others
2020-06-13	Human Performance	0.9811	0.9756	0.9825	0.9780	0.9845	0.9839	0.9740	0.9717	0.9974	0.9828
2020-12-22	LayoutLM 2.0 (single model)	0.8672	0.6574	0.8953	0.8769	0.8791	0.8707	0.7287	0.6729	0.5517	0.8103
2020-08-16	Alibaba DAMO NLP Structural LM	0.8506	0.6650	0.8809	0.8552	0.8733	0.8397	0.6758	0.7691	0.5492	0.7526
2020-05-16	PingAn-OneConnect-Gammalab-DQA	0.8484	0.6059	0.9021	0.8463	0.8730	0.8337	0.5812	0.7692	0.5172	0.7289
2020-05-14	Structural LM-v2	0.7674	0.4931	0.8381	0.7621	0.7924	0.7596	0.4756	0.6282	0.5517	0.6549
2020-05-15	QA_Base_MRC_2	0.7415	0.4854	0.8015	0.6738	0.7943	0.8136	0.5740	0.5831	0.5287	0.7161
2020-05-15	QA_Base_MRC_1	0.7407	0.4890	0.7984	0.6675	0.7936	0.8131	0.5854	0.6099	0.4943	0.7384
2020-05-15	QA_Base_MRC_4	0.7348	0.4735	0.8040	0.6647	0.7838	0.8043	0.5618	0.5810	0.4598	0.7332
2020-05-15	QA_Base_MRC_3	0.7322	0.4852	0.7958	0.6562	0.7842	0.8044	0.5679	0.5730	0.4511	0.7171
2020-05-15	QA_Base_MRC_5	0.7274	0.4858	0.7877	0.6550	0.7754	0.8047	0.5405	0.5619	0.4598	0.7084
2020-05-16	HyperDQA_V4	0.6893	0.3874	0.7792	0.6309	0.7478	0.7187	0.4867	0.5630	0.4138	0.5685
2020-05-16	HyperDQA_V3	0.6769	0.3876	0.7774	0.6167	0.7332	0.6961	0.4296	0.5373	0.4138	0.5650
2020-05-16	HyperDQA_V2	0.6734	0.3818	0.7666	0.6110	0.7332	0.6867	0.4834	0.5560	0.3793	0.5902

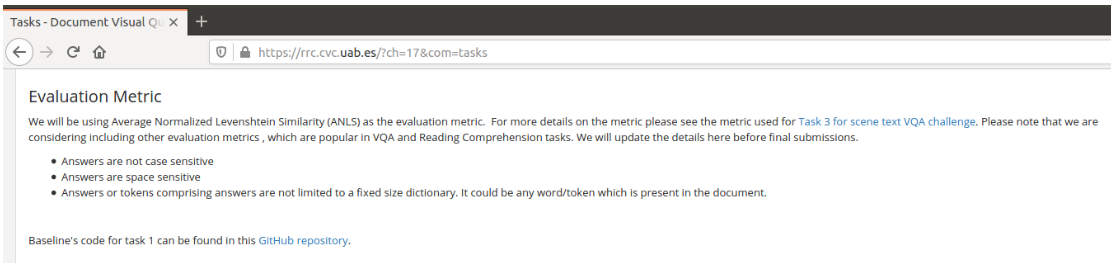




2020-05-09	HyperDQA_V1	0.6717	0.4013	0.7693	0.6197	0.7167	0.6922	0.3598	0.5596	0.4138	0.5504
2020-05-09	bert fulldata fintuned	0.5900	0.4169	0.6870	0.4269	0.6710	0.7315	0.5124	0.4900	0.4483	0.5907
2020-05-01	bert finetuned	0.5872	0.2986	0.7011	0.4849	0.6359	0.6933	0.4622	0.4751	0.4483	0.4895
2020-04-30	HyperDQA_V0	0.5715	0.3131	0.6780	0.4732	0.6630	0.5716	0.3623	0.4351	0.3793	0.4941
2021-01-14	Test	0.5545	0.2673	0.7037	0.4921	0.5983	0.5423	0.3022	0.4570	0.2534	0.4471
2020-04-27	bert	0.4557	0.2233	0.5259	0.2633	0.5113	0.7775	0.4859	0.3565	0.0345	0.5778
2020-05-16	UGLIFT v0.1 (Clova OCR)	0.4417	0.1766	0.5600	0.3178	0.5340	0.4520	0.2253	0.3573	0.4483	0.3356
2020-05-14	Plain BERT QA	0.3524	0.1687	0.4489	0.2029	0.4321	0.4812	0.3517	0.3096	0.0345	0.3747
2020-05-16	Clova OCR V0	0.3489	0.0977	0.4855	0.2670	0.3811	0.3958	0.2489	0.2875	0.0345	0.3062
2020-05-01	HDNet	0.3401	0.2040	0.4688	0.2181	0.4710	0.1916	0.2488	0.2736	0.1379	0.2458
2020-05-16	CLOVA OCR	0.3296	0.1246	0.4612	0.2455	0.3622	0.3746	0.1692	0.2736	0.0690	0.3205
2020-04-29	docVQAQV_V0.1	0.3016	0.2010	0.3898	0.3810	0.2933	0.0664	0.1842	0.2736	0.1586	0.1695
2020-04-26	docVQAQV_V0	0.2342	0.1646	0.3133	0.2623	0.2483	0.0549	0.2277	0.1856	0.1034	0.1635
2021-01-22	Seq2seq	0.0552	0.0434	0.0727	0.0383	0.0651	0.0444	0.0244	0.0549	0.2586	0.0562
2020-06-16	Test Submission	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

The Test result is again based again on the basis of where the actual test data answers do not have any whitespaces between the punctuations and the predicted answers do have it.

This is because they have mentioned that the ANLS metric used during submission is ofcourse space sensitive.



Tasks - Document Visual Qu ×
+
https://rrc.cvc.uab.es/?ch=17&com=tasks

Evaluation Metric

We will be using Average Normalized Levenshtein Similarity (ANLS) as the evaluation metric. For more details on the metric please see the metric used for Task 3 for scene text VQA challenge. Please note that we are considering including other evaluation metrics , which are popular in VQA and Reading Comprehension tasks. We will update the details here before final submissions.

- Answers are not case sensitive
- Answers are space sensitive
- Answers or tokens comprising answers are not limited to a fixed size dictionary. It could be any word/token which is present in the document.

Baseline's code for task 1 can be found in this GitHub repository.

4.2. Model 2. (Image and OCR VQA)

4.2.1. Data Preprocessing.

4.2.1.1. Importing Data

Over here I have appended a new datapoint in the train dataset to avoid an error while training.

Since the train data has a shape of '(39463,6)', concatenating a datapoint(duplicate of the last datapoint in the dataset) turns it into a shape of '(39464,6)'. This makes it easy to create batch sizes in accordance to multiples of 2.

Since I have a batch size of 8, it creates batch size of 4,933 for each batch.

In [4]:
```
#Importing datasets
train = pd.read_csv('/content/drive/MyDrive/Data/train.csv')
val = pd.read_csv('/content/drive/MyDrive/Data/val.csv')
test = pd.read_csv('/content/drive/MyDrive/Data/test.csv')

#Appending an extra datapoint in the train dataset.
image = list(train.Image)[-1]
ocr = list(train.OCR)[-1]
question = list(train.Question)[-1]
question_id = list(train.Question_Id)[-1]
answer = list(train.Answer)[-1]
```

```
answer_list = list(train.Answer_list)[-1]
new_record = pd.DataFrame([[image,ocr,question,question_id,answer,a
                           columns=['Image', 'OCR', 'Question', 'Que
train = pd.concat([train,new_record])

#Appending an extra datapoint in the validation dataset.
for i in range(3):
  image = list(val.Image)[i]
  ocr = list(val.OCR)[i]
  question = list(val.Question)[i]
  question_id = list(val.Question_Id)[i]
  answer = list(val.Answer)[i]
  answer_list = list(val.Answer_list)[i]
  new_record = pd.DataFrame([[image,ocr,question,question_id,answer
                           columns=['Image', 'OCR', 'Question', 'Que
  val = pd.concat([val,new_record])
```

In [5]:

```
print("Shape of Train data : \t\t",train.shape)
print("Shape of Validation data : \t",val.shape)
print("Shape of Test data : \t\t",test.shape)
```

```
Shape of Train data :            (39464, 6)
Shape of Validation data :       (5352, 6)
Shape of Test data :             (5188, 4)
```

4.2.1.2. OCR, Questions and Answers.

In [ ]:

```
def preprocess_qa(w):
  '''
    The function 'preprocess_qa' preprocesses both the questions as
  '''
  # creating a space between a word and the punctuation following i
  # eg: "he is a boy." => "he is a boy ."
  # Reference:- https://stackoverflow.com/questions/3645931/python-
  w = re.sub('([!"#$%&()*+.,-/:;=?@[\]<>?^_`{|}~])', r' \1 ', w)
  w = re.sub('\s{2,}', ' ', w)

  # replacing everything with space except (a-z, A-Z, ".", "?", "!
  w = re.sub('(?<=[A-Za-z])(?=[0-9])|(?<=[0-9])(?=[A-Za-z])',' ', w

  w = ' '.join(e.lower() for e in w.split())

  w = w.strip()

  # adding a start and an end token to the sentence
  # so that the model know when to start and stop predicting.
  w = '<start> ' + w + ' <end>'
  return w

def preprocess_ocr(w):
  '''
    The function 'preprocess_qa' preprocesses ocr texts.
  '''
  with open(w, 'r') as f:
    annotations = json.load(f)
  w = ''
  for i in range(len(annotations['recognitionResults'][0]['lines'])
    w+= ' '+annotations['recognitionResults'][0]['lines'][i]['text'
  w = w.split(' Source: ', 1)[0]
  w = preprocess_qa(w)
```

```
    return w
```

In [ ]:
```
eg_question = train['Question'][1]
print(preprocess_qa(eg_question))
del eg_question
```

```
<start> what is the date mentioned in the document ? <end>
```

In [ ]:
```
eg_answer = train['Answer'][1]
print(preprocess_qa(eg_answer))
del eg_answer
```

```
<start> may 2002 <end>
```

In [ ]:
```
eg_ocr = train['OCR'][1]
print(preprocess_ocr(eg_ocr))
del eg_ocr
```

```
<start> metabolic effects of menopausal therapies outline n draft
date : may 2002 project # 8910 bush , et al ( 1987 ) circulation
75 : 1102 - 9 de leo , et al ( 2001 ) am j obstet . gynercol . 18
4 : 350 - 3 . perera , et al . ( 2002 ) hum reprod 17 : 497 - 502
khurana , et al . ( 2001 ) curr atherosclerkep . 3 : 399 - simon
, et al . ( 2001 ) circulation 103 : 638 - 642 rosano , et al . (
2000 ) maturitas 34 : ( supply $ 3 - $ 10 smolders , et al . ( 20
02 ) maturitas 41 : 105 - 14 arnal . bayard ( 2001 ) clin exp pha
rmacol physiol . 28 : 1032 - 4 . hulley , et al . ( 1998 ) jama 2
80 : 605 - 613 ( and comments 650 - 2 ; also 1999 ; vol 281 . p 7
94 - 7 ) herrington . et al . ( 1998 ) am heart j . 136 : 115 - 2
4 ( and erratum 1999 ; vol 138 , p 800 ) barrett - connor , et al
. ( 2002 ) jama 287 : 847 - 57 hlatky , et al . ( 2002 ) jama 287
: 591 - 7 ( and comment , 641 - 2 ) andersson , et al . ( 2002 )
j . clin . endocrinol . metab . 87 : 122 - 128 wenger ( 2000 ) am
j geriatrcardiol . 9 : 204 - 209 milner , et al . ( 1996 ) obstet
gynercol . 87 : 593 - 9 wiegratz et al ( 2002 ) matunitas 41 : 13
3 - 41 ginsburg , et al . ( 1995 ) maturitas 21 : 71 - 6 cagnacci
, et al . ( 1997 ) j . clin . endocrinol . metab . 82 : 251 - 3 j
ackson ( 2001 ) eur heart j . ( supply 3 / m , m 17 - m 21 models
ka , cummings ( 2002 ) j . clin . endocrinol . metab . 87 : 16 -
23 . burger ( 2000 ) hormone res . 53 : ( supply 25 - 9 doren , e
t al . ( 2000 ) am j obstet gynercol . 183 : 575 - 82 ginsburg ,
prelevic ( 1999 ) menopause 6 : 87 - 9 ( include comments 92 - 10
4 ) prelevic , et al . ( 1998 ) maturitas 28 : 271 - 6 . prelevic
, et al ( 19980 maturitas 27 : 85 - 90 . meeuwsen , et al . ( 200
1 ) endocrinology 142 : 4813 - 4817 . hanggi . et al . ( 1998 ) c
lin . endocrinol . 48 : 691 - 9 . genazzan ( 1998 ) chin . endocr
inol . 48 : 683 . rymer , et al ( 1997 ) maturitas 27 : ( supply
136 ( abstract ? ) . dwrite 072641 <end>
```

The function **'create_dataset'** returns, preprocessed questions for text dataset, and, preprocessed questions and answers for train and validation datasets.

The function has been used for test and validation sets as well, because the unsupervised tokens and embeddings can be created for all the dataset types together.

I have truncated the ocr texts to a length of only 220 because of hardware limitations.

In [6]:
```
def create_dataset(df, type_of_dataset):
    '''
      The function 'create_dataset' takes 2 paramaeters:
```

```
      df = The dataframe of the dataset type.
      type_of_dataset = The type of dataset

    It returns list/s of preprocessed questions and/or answers base
  '''
  if type_of_dataset=='test':
    questions = []
    ocr = []
    for i in tqdm(range(len(df['Question']))):
      questions.append(preprocess_qa(list(df['Question'])[i]))
      temp_ocr_result = preprocess_ocr(list(df['OCR'])[i])
      if len(temp_ocr_result.split())<221:
        ocr.append(temp_ocr_result)
      else:
        ocr_words = temp_ocr_result.split()
        temp_final_ocr = ' '.join(ocr_words[:219])
        if '<end>' not in temp_final_ocr:
          temp_final_ocr = temp_final_ocr + ' <end>'
        ocr.append(temp_final_ocr)
    return questions, ocr

  else:
    questions = []
    answers = []
    ocr = []
    for i in tqdm(range(len(df['Question']))):
      questions.append(preprocess_qa(list(df['Question'])[i]))
      answers.append(preprocess_qa(list(df['Answer'])[i]))
      temp_ocr_result = preprocess_ocr(list(df['OCR'])[i])
      if len(temp_ocr_result.split())<221:
        ocr.append(temp_ocr_result)
      else:
        ocr_words = temp_ocr_result.split()
        temp_final_ocr = ' '.join(ocr_words[:219])
        if '<end>' not in temp_final_ocr:
          temp_final_ocr = temp_final_ocr + ' <end>'
        ocr.append(temp_final_ocr)
    return questions,answers,ocr
```

In [ ]:

```
train_questions, train_answers, train_ocr = create_dataset(train,'t
print("\n",train_questions[-1])
print(train_answers[-1])
print(train_ocr[-1])
```

```
100%|██████████| 39464/39464 [06:12<00:00, 106.08it/s]
```

```
 <start> when is the memo dated on ? <end>
<start> october 22 , 1970 <end>
<start> memo to : r . j . fisher from : g . f . lachenauer subjec
t : consumer correspondence october 22 , 1970 this is to summariz
e our meeting on october 21 and to review what was agreed upon .
1 . starting january 1 , all consumer correspondence will be sent
```

In [ ]:
```python
val_questions, val_answers, val_ocr = create_dataset(val,'val')
print("\n",val_questions[-1])
print(val_answers[-1])
print(val_ocr[-1])
```

```
100%|██████████| 5352/5352 [00:10<00:00, 487.49it/s]


 <start> what is the amount of deposit ? <end>
<start> 100 . 00 <end>
<start> form 164 deposited by please see that all checks are endo
rsed dollars cents w . j . darby , m . d . route 2 , box 218 curr
ency thompson station , tenn . 37179 coins check august 9 , 1976
19 100 00 first american national bank of nashville receives all
items , whether for credit or collect tion , as depositor's agent
with authority to forward items for collection direct to the draw
ee or payor bank or through any other bank or clearing house at i
ts discretion , and to receive payment in drafts drawn by any of
the said banks . this bank shall not be liable for loss of items
in transit , or for misconduct , negligence , or any other defaul
ts of sub - agents , all of whom shall be deemed agents of deposi
tor . all credits are conditional subject to charge back to depos
itor's account if not collected , whether drawn on this bank or a
nother . firstamerican national bank nashville , tennessee total
hillsboro office 100 00 credit ( if additional spaces are needed
see reverse side ) 1 : 8888 0000 . : 321 285 10 <end>
```

In [ ]:
```python
test_questions, test_ocr = create_dataset(test, 'test')
print("\n",test_questions[-1])
print(test_ocr[-1])
```

```
100%|██████████| 5188/5188 [00:08<00:00, 600.46it/s]


 <start> what is the p . o . box number ? <end>
<start> additional copies available at no charge by writing to :
a proposal for restructuring children's the quaker oats company p
. o . box 3493 television merchandise mart plaza chicago , illino
is 60654 4 . . quaker <end>
```

In [ ]:
```python
def tokenize(tr_inp_q, tr_inp_a, tr_inp_o, v_inp_q, v_inp_a, v_inp_
  '''
    This function takes 2 parameters :
      data = list of the texts corresponding to the dataset.
      texts = list of all the texts with unsupervised tokens.
    It returns :
      seq = non-padded tokenized sequences of the 'data'.
      tensor = padded tokenized sequences of the 'data'.
      tokenizer = tokenizer which has been fit on all the tokens ir
  '''
  tr_q = np.array(tr_inp_q)
  tr_a = np.array(tr_inp_a)
  tr_o = np.array(tr_inp_o)
  v_q = np.array(v_inp_q)
  v_a = np.array(v_inp_a)
  v_o = np.array(v_inp_o)

  tr_inp_q.extend(tr_inp_a)
  del tr_inp_a
  tr_inp_q.extend(tr_inp_o)
  del tr_inp_o
  tr_inp_q.extend(v_inp_q)
  del v_inp_q
  tr_inp_q.extend(v_inp_a)
  del v_inp_a
  tr_inp_q.extend(v_inp_o)
  del v_inp_o
  tr_inp_q.extend(te_inp_q)
  del te_inp_q
  tr_inp_q.extend(te_inp_o)
  del te_inp_o

  #Fitting the tokenizer on texts with all the unsupervised tokens
  tokenizer = tf.keras.preprocessing.text.Tokenizer(
      filters='')
  tokenizer.fit_on_texts(tr_inp_q)

  #=============== Train ===============
  #Replacing text with tokens in the data
  tr_q_tensor = tokenizer.texts_to_sequences(tr_q)
  #Padding the tokenized sequences
  tr_q_tensor = tf.keras.preprocessing.sequence.pad_sequences(tr_q_
                                                                paddi

  #Replacing text with tokens in the data
  tr_a_tensor = tokenizer.texts_to_sequences(tr_a)
  #Padding the tokenized sequences
  tr_a_tensor = tf.keras.preprocessing.sequence.pad_sequences(tr_a_
                                                                paddi

  #Replacing text with tokens in the data
  tr_o_tensor = tokenizer.texts_to_sequences(tr_o)
  #Padding the tokenized sequences
  tr_o_tensor = tf.keras.preprocessing.sequence.pad_sequences(tr_o_
                                                                paddi

  #=============== Validation ===============
  #Replacing text with tokens in the data
  v_q_tensor = tokenizer.texts_to_sequences(v_q)
  #Padding the tokenized sequences
  v_q_tensor = tf.keras.preprocessing.sequence.pad_sequences(v_q_te
                                                               maxler
                                                               paddir

  #Replacing text with tokens in the data
  v_a_tensor = tokenizer.texts_to_sequences(v_a)
  #Padding the tokenized sequences
```

```
  v_a_tensor = tf.keras.preprocessing.sequence.pad_sequences(v_a_te
                                                             maxler
                                                         padding='p

  #Replacing text with tokens in the data
  v_o_tensor = tokenizer.texts_to_sequences(v_o)
  #Padding the tokenized sequences
  v_o_tensor = tf.keras.preprocessing.sequence.pad_sequences(v_o_te
                                                             maxler
                                                             paddir

  return tr_q_tensor, tr_a_tensor, tr_o_tensor, v_q_tensor, v_a_ter
```

In [ ]:

```
# Tokenizing the Train Questions, Answers and OCR
train_input_tensor_q, train_target_tensor, train_input_tensor_o, va









del train_questions, train_answers, train_ocr, val_questions, val_a
```

In [ ]:

```
# These results are inclusive of the unsupervised tokens.
#92285
print("Number of unique words in questions, answers and ocr :",len(
```

```
Number of unique words in questions, answers and ocr : 74507
```

In [ ]:

```
def convert(lang, tensor):
  for t in tensor:
    if t!=0:
      print ("%d ----> %s" % (t, lang.index_word[t]))
```

In [ ]:

```
print ("Input Text Question ; index to word mapping")
convert(text, train_input_tensor_q[0])
print ()
print ("Input Text OCR ; index to word mapping")
convert(text, train_input_tensor_o[0])
print ()
print ("Target Text Answer; index to word mapping")
convert(text, train_target_tensor[0])
```

```
Input Text Question : index to word mapping
```

The .bin which was downloaded earlier from the FastText website (https://fasttext.cc/) contains the model weigths of the fasttext model.

This model creates embedding for any type of text in the universe.

So, in this way I am creating an embedding matrix of the embeddings of all the tokens where each index represents the embedding array of length 300.

In [ ]:
```python
#importing fasttext model
model = fasttext.load_model("crawl-300d-2M-subword.bin")
```

```
Warning : `load_model` does not return WordVectorModel or Supervi
sedModel any more, but a `FastText` object which is very similar.
```

In [ ]:
```python
#Initiating the embedding matrix with respect to the tokens in the
embedding_matrix = np.zeros((len(text.word_index)+1, 300))
for word, i in text.word_index.items():
    embedding_vector = model[word]
    embedding_matrix[i] = embedding_vector
```

In [ ]:
```python
# Deleting the model to save RAM
del model
```

4.2.1.3. Images.

I have used InceptionV3 (which is pretrained on Imagenet) to classify each document image. I have extracted features from the last convolutional layer.

At First, I converted the images into InceptionV3's expected format by:

- Resizing the image to 299px by 299px.
- Preprocessing the images using the preprocess_input method to normalize the image so that it contains pixels in the range of -1 to 1, which matches the format of the images used to train InceptionV3.

In [ ]:
```python
def load_image(image_path):
    img = tf.io.read_file(image_path)
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.image.resize(img, (299, 299))
    img = tf.keras.applications.inception_v3.preprocess_input(img)
    return img, image_path
```

Now I created a tf.keras model where the output layer is the last convolutional layer in the InceptionV3 architecture. The shape of the output of this layer is 8x8x2048. I used the last convolutional layer because I was using attention in the baseline architecture.

I forwarded each image through the network and stored the resulting vector in a dictionary (image_name --> feature_vector).

In [ ]:
```python
image_model = tf.keras.applications.InceptionV3(include_top=False,
                                                weights='imagenet')
new_input = image_model.input
hidden_layer = image_model.layers[-1].output
```

```
image_features_extract_model = tf.keras.Model(new_input, hidden_lay
```

```
Downloading data from https://storage.googleapis.com/tensorflow/k
eras-applications/inception_v3/inception_v3_weights_tf_dim_orderi
ng_tf_kernels_notop.h5 (https://storage.googleapis.com/tensorflow
/keras-applications/inception_v3/inception_v3_weights_tf_dim_orde
ring_tf_kernels_notop.h5)
87916544/87910968 [==============================] - 2s 0us/step
```

I pre-processed each image with InceptionV3 and cached the output to the disk. Caching the output in RAM would be faster but also memory intensive, requiring 8 * 8 * 2048 floats per image.

In [ ]:

```
# Get unique images of Train and Validation
encode_train = list(train['Image'])
encode_val = list(val['Image'])

train_image_dataset = tf.data.Dataset.from_tensor_slices(encode_tra
del encode_train
train_image_dataset = train_image_dataset.map(
  load_image, num_parallel_calls=tf.data.experimental.AUTOTUNE).bat

val_image_dataset = tf.data.Dataset.from_tensor_slices(encode_val)
del encode_val
val_image_dataset = val_image_dataset.map(
  load_image, num_parallel_calls=tf.data.experimental.AUTOTUNE).bat

for img, path in tqdm(train_image_dataset):
  batch_features = image_features_extract_model(img)
  batch_features = tf.reshape(batch_features,
                              (batch_features.shape[0], -1, batch_f

  for bf, p in zip(batch_features, path):
    path_of_feature = p.numpy().decode("utf-8")
    np.save(path_of_feature, bf.numpy())

for img, path in tqdm(val_image_dataset):
  batch_features = image_features_extract_model(img)
  batch_features = tf.reshape(batch_features,
                              (batch_features.shape[0], -1, batch_f

  for bf, p in zip(batch_features, path):
    path_of_feature = p.numpy().decode("utf-8")
    np.save(path_of_feature, bf.numpy())

del train_image_dataset
del val_image_dataset
```

```
100%|██████████| 2467/2467 [13:20<00:00,  3.08it/s]
100%|██████████| 335/335 [01:31<00:00,  3.67it/s]
```

In [ ]:

```
train_img_to_cap_vector_inp_q = collections.defaultdict(list)
train_img_to_cap_vector_inp_o = collections.defaultdict(list)
train_img_to_cap_vector_tar = collections.defaultdict(list)
val_img_to_cap_vector_inp_q = collections.defaultdict(list)
val_img_to_cap_vector_inp_o = collections.defaultdict(list)
val_img_to_cap_vector_tar = collections.defaultdict(list)


for img, inp_q, inp_o, tar in zip(train['Image'], train_input_tens
  train_img_to_cap_vector_inp_q[img].append(inp_q)
  train_img_to_cap_vector_inp_o[img].append(inp_o)
  train_img_to_cap_vector_tar[img].append(tar)
for img, inp_q, inp_o, tar in zip(val['Image'], val_input_tensor_q,
  val_img_to_cap_vector_inp_q[img].append(inp_q)
  val_img_to_cap_vector_inp_o[img].append(inp_o)
  val_img_to_cap_vector_tar[img].append(tar)


train_img_keys = list(train_img_to_cap_vector_inp_q.keys())
random.shuffle(train_img_keys)
val_img_keys = list(val_img_to_cap_vector_inp_q.keys())
random.shuffle(val_img_keys)

img_name_train = []
input_train_q = []
input_train_o = []
target_train = []
for imgt in train_img_keys:
  inp_len = len(train_img_to_cap_vector_inp_q[imgt])
  img_name_train.extend([imgt] * inp_len)
  input_train_q.extend(train_img_to_cap_vector_inp_q[imgt])
  input_train_o.extend(train_img_to_cap_vector_inp_o[imgt])
  target_train.extend(train_img_to_cap_vector_tar[imgt])

img_name_val = []
input_val_q = []
input_val_o = []
target_val = []
for imgt in val_img_keys:
  inp_len = len(val_img_to_cap_vector_inp_q[imgt])
  img_name_val.extend([imgt] * inp_len)
  input_val_q.extend(val_img_to_cap_vector_inp_q[imgt])
  input_val_o.extend(val_img_to_cap_vector_inp_o[imgt])
  target_val.extend(val_img_to_cap_vector_tar[imgt])


del train_img_to_cap_vector_inp_q
del train_img_to_cap_vector_inp_o
del train_img_to_cap_vector_tar
del train_img_keys
del val_img_to_cap_vector_inp_q
del val_img_to_cap_vector_inp_o
del val_img_to_cap_vector_tar
del val_img_keys
```

```
In [ ]: len(img_name_train), len(input_train_q), len(input_train_o), len(ta
```

```
Out[27]: (39464, 39464, 39464, 39464)
```

```
In [ ]: len(img_name_val), len(input_val_q), len(input_val_o), len(target_v
```

```
Out[28]: (5352, 5352, 5352, 5352)
```

4.2.1.4. Saving data

In [ ]:
```
#Saving all the pickle files
pickle.dump((img_name_train, input_train_q, input_train_o, target_t
pickle.dump((img_name_val, input_val_q, input_val_o, target_val),op
```

4.2.1.5. Loading data

In [6]:
```
#Loading all the pickle files
img_name_train, input_train_q, input_train_o, target_train, text, e
img_name_val, input_val_q, input_val_o, target_val = pickle.load(op
```

4.2.1.4. Creating tf.data Dataset for training.

In [7]:
```
TRAIN_BUFFER_SIZE = 39464
VAL_BUFFER_SIZE = 5352
max_length_inp_q = 41
max_length_inp_o = 220
max_length_targ = 39
BATCH_SIZE = 2
```

In [ ]:
```
# Load the numpy files
def map_func(img_name, inp_q, inp_o, targ):
  img_tensor = np.load(img_name.decode('utf-8')+'.npy')
  return img_tensor, inp_q, inp_o, targ
```

In [ ]:
```
train_dataset = tf.data.Dataset.from_tensor_slices((img_name_train,
val_dataset = tf.data.Dataset.from_tensor_slices((img_name_val, inp
```

In [ ]:
```
# Creating tf.data Dataset

#========== Train ==========
# map to load the numpy files in parallel
train_dataset = train_dataset.map(lambda item1, item2, item3, item4


                                  num_parallel_calls=tf.data.experi

# Shuffle and batch
train_dataset = train_dataset.shuffle(TRAIN_BUFFER_SIZE).batch(BATC
train_dataset = train_dataset.prefetch(buffer_size=tf.data.experime

#========== Validation ==========
# map to load the numpy files in parallel
val_dataset = val_dataset.map(lambda item1, item2, item3, item4: tf
                                               [item1, item2, item3
                                               [tf.float32, tf.int3
                      num_parallel_calls=tf.data.experimental.AUTOT

# Shuffle and batch
val_dataset = val_dataset.shuffle(VAL_BUFFER_SIZE).batch(BATCH_SIZE
val_dataset = val_dataset.prefetch(buffer_size=tf.data.experimental
```

In [ ]:
```
#Printing the shapes of the train inputs
example_input_img, example_input_batch_q, example_input_batch_o, ex
example_input_img.shape,  example_input_batch_q.shape, example_inpu
```

Out[9]:

```
(TensorShape([2, 64, 2048]),
 TensorShape([2, 41]),
```

In [ ]:

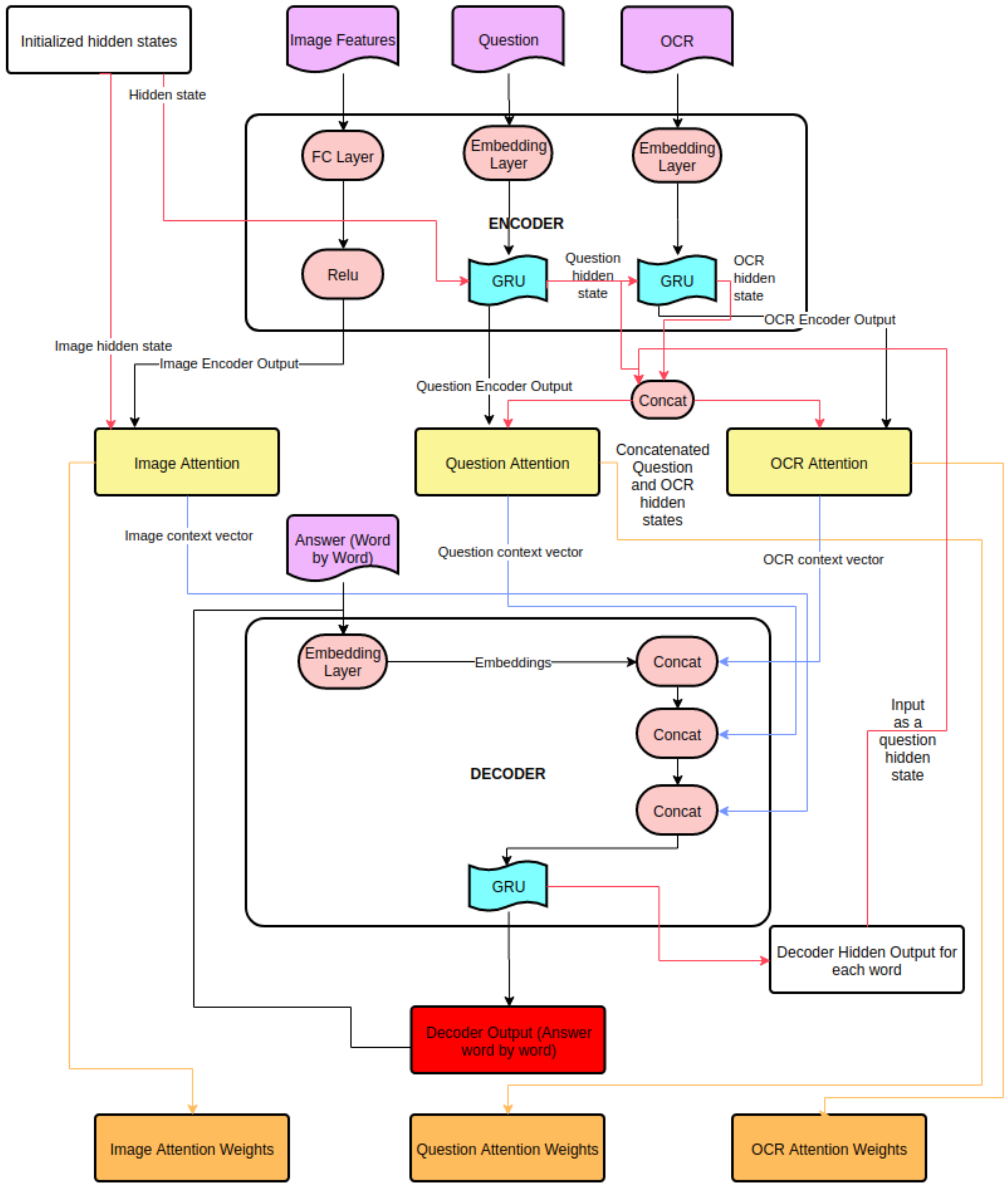
```
#Printing the shapes of the validation inputs
example_input_img, example_input_batch_q, example_input_batch_o, ex
example_input_img.shape,  example_input_batch_q.shape, example_inpu
```

Out[10]:

```
(TensorShape([2, 64, 2048]),
 TensorShape([2, 41]),
 TensorShape([2, 220]),
 TensorShape([2, 39]))
```

4.2.2. Model.

4.2.2.1. Model Architecture.



4.2.2.2. Encoder.

In [8]:
```
class Encoder(tf.keras.Model):
  '''
    Encoder model -- That takes a input sequence and returns encode
  '''
  def __init__(self, vocab_size, embedding_dim, enc_units, batch_sz
    super(Encoder, self).__init__()
    self.batch_sz = batch_sz
    self.enc_units = enc_units
    self.embedding = tf.keras.layers.Embedding(vocab_size,
                                               embedding_dim,
                                               weights=[inp_embed_m
                                               trainable=False)
    self.ocr_gru = tf.keras.layers.GRU(self.enc_units,
                                   return_sequences=True,
                                   return_state=True,
                                   recurrent_initializer='glorot_un
    self.question_gru = tf.keras.layers.GRU(self.enc_units,
                                   return_sequences=True,
                                   return_state=True,
                                   recurrent_initializer='glorot_un
    self.fc = tf.keras.layers.Dense(embedding_dim)

  def call(self, image, ocr, question, hidden):
    '''
      This function takes a sequence input of question, ocr and ima
      Pass the input sequences of question and ocr input to the Emb
      returns -- All encoder_outputs, last time steps hidden states
    '''

    #==================== Image Encoding ====================
    #Squashing the output shape of the InceptionV3 model to (batch_
    image = self.fc(image)
    #features shape after passing through the Dense Layer == (batch
    image = tf.nn.relu(image)

    #==================== Question Encoding ====================
    # question shape after passing through embedding == (batch_size
    question = self.embedding(question)
    # output shape == (batch_size, input_length(41), gru_size(enc_u
    # state shape == (batch_size, gru_size(enc_units))
    question_output, question_state = self.question_gru(question, i

    #==================== OCR Encoding ====================
    # question shape after passing through embedding == (batch_size
    ocr = self.embedding(ocr)
    # output shape == (batch_size, input_length(220), gru_size(enc_
    # state shape == (batch_size, gru_size(enc_units))
    ocr_output, ocr_state = self.ocr_gru(ocr, initial_state = quest

    return image, ocr_output, ocr_state, question_output, question_

  def initialize_hidden_state(self):
    return tf.zeros((self.batch_sz, self.enc_units))
```

In [9]:
```
def grader_check_encoder():
    '''
      This function is used to simulate the scenario for one traini
      and also to prove that all the shapes of the inputs and outpu
    '''
    vocab_size=len(text.word_index)+1
    embedding_size=300
    lstm_size=128
    input_length_q=41
    input_length_o=220
    batch_size=BATCH_SIZE
    encoder=Encoder(vocab_size,
                    embedding_size,
                    lstm_size,
                    batch_size,
                    embedding_matrix)
    input_question=tf.random.uniform(shape=[batch_size,input_length
                                     maxval=vocab_size,minval=0,
                                     dtype=tf.int32)
    input_ocr=tf.random.uniform(shape=[batch_size,input_length_o],
                                     maxval=vocab_size,minval=0,
                                     dtype=tf.int32)
    input_image=tf.random.uniform(shape=[batch_size,
                                        64,
                                        2048],
                                 maxval=vocab_size,minval=0,dtype=t
    initial_state=encoder.initialize_hidden_state()

    encoder_output_img,encoder_output_o,state_o,encoder_output_q,st




    assert(encoder_output_q.shape==(batch_size,input_length_q,lstm_
           encoder_output_o.shape==(batch_size,input_length_o,lstm_
           encoder_output_img.shape==(batch_size,64,embedding_size)
           state_q.shape==(batch_size,lstm_size) and
           state_o.shape==(batch_size,lstm_size))
    return True

print(grader_check_encoder())
```

```
True
```

4.2.2.3. Attention.

In [10]:

```
class Image_Attention(tf.keras.Model):
  '''
    Class that calculates Image attention score based on the scorir
  '''
  def __init__(self, units):
    super(Image_Attention, self).__init__()
    self.W1 = tf.keras.layers.Dense(units)
    self.W2 = tf.keras.layers.Dense(units)
    self.V = tf.keras.layers.Dense(1)

  def call(self, features, hidden_i):
    '''
      Attention mechanism takes two inputs current step -- image_hi
      * Based on the scoring function we will find the score or sin
        Multiply the score function with your encoder_outputs to ge
        Function returns context vector and attention weights(softm
    '''
    # features(CNN_encoder output) shape == (batch_size, 64, embedd
    # hidden shape == (batch_size, hidden_size)
    # hidden_with_time_axis shape == (batch_size, 1, hidden_size)
    hidden_with_time_axis_q = tf.expand_dims(hidden_i, 1)

    # attention_hidden_layer shape == (batch_size, 64, units)
    attention_hidden_layer = (tf.nn.relu(self.W1(hidden_with_time_a
                                         self.W2(features)))

    # score shape == (batch_size, 64, 1)
    # This gives you an unnormalized score for each image feature.
    score = self.V(attention_hidden_layer)

    # attention_weights shape == (batch_size, 64, 1)
    attention_weights = tf.nn.softmax(score, axis=1)

    # context_vector shape after sum == (batch_size, hidden_size)
    context_vector = attention_weights * features
    context_vector = tf.reduce_sum(context_vector, axis=1)

    return context_vector, attention_weights
```

In [11]:

```
class OCR_Attention(tf.keras.layers.Layer):
  '''
    Class that calculates OCR attention score based on the scoring_
  '''
  def __init__(self, units):
    super(OCR_Attention, self).__init__()
    self.W1 = tf.keras.layers.Dense(units)
    self.W2 = tf.keras.layers.Dense(units)
    self.V = tf.keras.layers.Dense(1)

  def call(self, query_o, values):
    '''
      Attention mechanism takes two inputs current step -- Concaten
      * Based on the scoring function we will find the score or sim
        Multiply the score function with your ocr_encoder_output to
        Function returns context vector and attention weights(softm
    '''
    # query hidden state shape == (batch_size, ocr hidden size)
    # query_with_time_axis shape == (batch_size, 1, ocr hidden size
    # values shape == (batch_size, max_len, ocr hidden size)
    # we are doing this to broadcast addition along the time axis t
    query_with_time_axis_o = tf.expand_dims(query_o, 1)

    # score shape == (batch_size, max_length(220), 1)
    # we get 1 at the last axis because we are applying score to se
    # the shape of the tensor before applying self.V is (batch_size
    score = self.V(tf.nn.relu(self.W1(query_with_time_axis_o) +
                              self.W2(values)))

    # attention_weights shape == (batch_size, max_length(220), 1)
    attention_weights = tf.nn.softmax(score, axis=1)

    # context_vector shape after sum == (batch_size, ocr_hidden_siz
    context_vector = attention_weights * values
    context_vector = tf.reduce_sum(context_vector, axis=1)

    return context_vector, attention_weights
```

In [12]:
```
class Question_Attention(tf.keras.layers.Layer):
  '''
    Class that calculates Question attention score based on the sco
  '''
  def __init__(self, units):
    super(Question_Attention, self).__init__()
    self.W1 = tf.keras.layers.Dense(units)
    self.W2 = tf.keras.layers.Dense(units)
    self.V = tf.keras.layers.Dense(1)

  def call(self, query_q, values):
    '''
      Attention mechanism takes two inputs current step -- Concaten
      * Based on the scoring function we will find the score or sim
        Multiply the score function with your question_encoder_outp
        Function returns context vector and attention weights(softm
    '''
    # query hidden state shape == (batch_size, question hidden size
    # query_with_time_axis shape == (batch_size, 1, question hidden
    # values shape == (batch_size, max_len, question hidden size)
    # we are doing this to broadcast addition along the time axis
    query_with_time_axis_q = tf.expand_dims(query_q, 1)

    # score shape == (batch_size, max_length(41), 1)
    # we get 1 at the last axis because we are applying score to se
    # the shape of the tensor before applying self.V is (batch_size
    score = self.V(tf.nn.relu(self.W1(query_with_time_axis_q) +
                              self.W2(values)))

    # attention_weights shape == (batch_size, max_length(41), 1)
    attention_weights = tf.nn.softmax(score, axis=1)

    # context_vector shape after sum == (batch_size, question_hidde
    context_vector = attention_weights * values
    context_vector = tf.reduce_sum(context_vector, axis=1)

    return context_vector, attention_weights
```

4.2.2.4. Decoder.

In [13]:
```
class Decoder(tf.keras.Model):
  '''
      Decoder model -- That takes a encoder outputs and hidden stat
  '''
  def __init__(self, vocab_size, embedding_dim, dec_units, batch_sz
    super(Decoder, self).__init__()

    #Initializing the variables
    self.batch_sz = batch_sz
    self.dec_units = dec_units

    # Initializing the layers
    self.embedding = tf.keras.layers.Embedding(vocab_size,
                                               embedding_dim,
                                               weights=[targ_embed_
                                               trainable=False)
    self.gru = tf.keras.layers.GRU(self.dec_units,
                                   return_sequences=True,
                                   return_state=True,
                                   recurrent_initializer='glorot_un
                                   dropout=0.1)
    self.fc = tf.keras.layers.Dense(vocab_size,
                                    activation='sigmoid',
                                    kernel_regularizer='l1')

    # Initializing the attention layers
    self.question_attention = Question_Attention(self.dec_units)
    self.ocr_attention = OCR_Attention(self.dec_units)
    self.image_attention = Image_Attention(self.dec_units)

  def call(self, x, hidden_q, hidden_o, hidden_i, enc_output_i, enc
    '''
        This function takes "input to decoder(x)", "question, ocr a
        "encoder hidden states (hidden_q, hidden_o, hidden_i)" and 
        from the hidden_q while predicting" for carrying out the op
        weigths for every input in the input_to_decoder(x).
    '''

    #=*=*=*=*=*=*=*=*=*=*= Concatenating the hidden states for the 


    #Concatenating the ocr and question hidden states for ocr and q
    hidden_qo = tf.concat([hidden_o,hidden_q],axis=-1)


    #=*=*=*=*=*=*=*=*=*=*= Calculating the context vectors and atte


    #==================== Question Attention ====================
    # defining question attention as a separate model
    # enc_output shape == (batch_size, input_length(41), hidden_siz
    context_vector_q, attention_weights_q = self.question_attention


    #==================== OCR Attention ====================
    # defining question attention as a separate model
    # enc_output shape == (batch_size, input_length(220), hidden_si
    context_vector_o, attention_weights_o = self.ocr_attention(hidd
                                                               enc_

    #==================== Image Attention ====================
    # defining image attention as a separate model
    context_vector_i, attention_weights_i = self.image_attention(en
                                                                 hi


    #=*=*=*=*=*=*=*=*=*=*= Concatenating the Context Vectors =*=*=*


    #==================== Concat OCR Context Vector ===============
    # Here x represents Decoder Input
    # x shape after passing through embedding == (batch_size, 1, em
    x = self.embedding(x)
    # x shape after concatenation == (batch_size, 1, embedding_dim 
    x = tf.concat([tf.expand_dims(context_vector_o, 1), x], axis=-1

    #==================== Concat Question Context Vector ==========
    # x shape after concatenation == (batch_size, 1, embedding_dim 
    x = tf.concat([tf.expand_dims(context_vector_q, 1), x], axis=-1

    #==================== Concat Image Context Vector ============
    # x shape after concatenation == (batch_size, 1, embedding_dim
    x = tf.concat([tf.expand_dims(context_vector_i, 1), x], axis=-1


    #=*=*=*=*=*=*=*=*=*=*= Decoding Answer =*=*=*=*=*=*=*=*=*=*=


    # passing the concatenated vector to the GRU
    output, state = self.gru(x, initial_state = hidden_q)

    # output shape == (batch_size * 1, hidden_size)
    output = tf.reshape(output, (-1, output.shape[2]))

    # output shape == (batch_size, vocab)
    x = self.fc(output)

    return x, state, attention_weights_q, attention_weights_o, atte

  def reset_state(self, batch_size):
    return tf.zeros((batch_size, self.dec_units))
```

In [14]:

```
def grader_check_decoder():
    '''
      This function is used to simulate the scenario for one traini
      and also to prove that all the shapes of the inputs and outpu
    '''
    vocab_size=len(text.word_index)+1
    embedding_size=300
    lstm_size=32
    input_length_q=41
    input_length_o=220
    targ_length=39
    batch_size=BATCH_SIZE

    encoder=Encoder(vocab_size,
                    embedding_size,
                    lstm_size,
                    batch_size,
                    embedding_matrix)

    input_question=tf.random.uniform(shape=[batch_size,input_length
                                     maxval=vocab_size,
                                     minval=0,
                                     dtype=tf.int32)
    input_ocr=tf.random.uniform(shape=[batch_size,input_length_o],
                                     maxval=vocab_size,minval=0,
                                     dtype=tf.int32)
    input_image=tf.random.uniform(shape=[batch_size,
                                         64,
                                         2048],
                                  maxval=vocab_size,
                                  minval=0,
                                  dtype=tf.float32)
    initial_state=encoder.initialize_hidden_state()

    encoder_output_img,encoder_output_o,state_o,encoder_output_q,st
    decoder = Decoder(vocab_size,
                      embedding_size,
                      lstm_size,
                      batch_size,
                      embedding_matrix)

    image_hidden = decoder.reset_state(batch_size)

    sample_decoder_output, state, attention_weights_q, attention_we
                                                              state
                                                              state
                                                              image
                                                              encod
                                                              encod
                                                              encod

    assert(sample_decoder_output.shape==(batch_size,vocab_size) and
           state.shape==(batch_size,lstm_size) and
           attention_weights_q.shape==(batch_size,41,1) and
           attention_weights_o.shape==(batch_size,400,1) and
           attention_weights_i.shape==(batch_size,64,1))
    return True

print(grader_check_decoder())
```

```
True
```

4.2.2.5. Optimizer and loss function.

In [15]:

```
vocab_size=len(text.word_index)+1
steps_per_epoch_train=39464//BATCH_SIZE
steps_per_epoch_val=5352//BATCH_SIZE
embedding_dim=300
units=128
batch_size=BATCH_SIZE
attention_features_shape = 64
```

In [16]:

```
#Initializing the objects of the model classes
encoder = Encoder(vocab_size,
                  embedding_dim,
                  units,
                  batch_size,
                  embedding_matrix)
decoder = Decoder(vocab_size,
                  embedding_dim,
                  units,
                  batch_size,
                  embedding_matrix)
```

In [17]:

```
#Initializing the optimizer
optimizer = tf.keras.optimizers.Adam()
#Initializing the loss function
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=False, reduction='none')

def loss_function(real, pred):
  mask = tf.math.logical_not(tf.math.equal(real, 0))
  loss_ = loss_object(real, pred)

  mask = tf.cast(mask, dtype=loss_.dtype)
  loss_ *= mask

  return tf.reduce_mean(loss_)
```

4.1.2.6. Training.

Steps involved while training :

1. Pass the input(Inceptionv3 image features and tokens of question) through the encoder. The image features are then concatenated with the question embeddings. The encoder then returns encoder output and the encoder hidden state.
2. The image features, encoder output, encoder hidden state and the decoder input (which is the start token) is passed to the decoder.
3. The decoder returns the predictions and the decoder hidden state.
4. The decoder hidden state is then passed back into the model and the predictions are used to calculate the loss.
5. teacher forcing method is used to decide the next input to the decoder.
6. Teacher forcing is the technique where the target word is passed as the next input to the decoder.
7. The final step is to calculate the gradients and apply it to the optimizer and backpropagate.

In [18]:

```
#Initializing the model checkpoint directory
checkpoint_dir = '/content/drive/MyDrive/Data/training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
checkpoint = tf.train.Checkpoint(optimizer=optimizer,
                                 encoder=encoder,
                                 decoder=decoder)
```

In [19]:

```
#Initializing the model log file for tensorboard
train_log_dir = '/content/drive/MyDrive/Data/tf_board_logs/gradient
train_summary_writer = tf.summary.create_file_writer(train_log_dir)
```

In [ ]:

```
@tf.function
def train_step(train_inp_image, train_inp_quest, train_inp_ocr, tra
  '''
    This function depicts the training step of the model.
    It takes 4 parameters -
      train_inp_image : Batched Image tensor of the train data.
      train_inp_quest : Batched Question tensor of the train data.
      train_inp_ocr : Batched OCR text tensor of the train data.
      train_targ_ans : Batched Answer tensor(Target tensor) of the
    This function updates the gradients of all the trainable parame
    It also returns the batch loss of the predictions of the batche
  '''
  train_loss = 0

  # initializing the hidden state for each batch
  # because the ANSWERS as well as QUESTIONS are not related from i
  train_img_hidden = decoder.reset_state(batch_size=BATCH_SIZE)
  train_enc_hidden = encoder.initialize_hidden_state()

  with tf.GradientTape() as tape:
    train_enc_output_i, train_enc_output_o, train_enc_hidden_o, tra



    train_dec_hidden_q = train_enc_hidden_q
    train_dec_hidden_o = train_enc_hidden_o

    train_dec_input = tf.expand_dims([text.word_index['<start>']] *

    # Teacher forcing - feeding the target as the next input
    for t in range(1, train_targ_ans.shape[1]):
      # passing encoder_output(enc_output) and image_features(image
      train_predictions, train_dec_hidden, _, _, _ = decoder(train_
                                                       train_
                                                       train_
                                                       train_
                                                       train_
                                                       train_
                                                       train_

      train_loss += loss_function(train_targ_ans[:, t], train_predi

      # using teacher forcing
      train_dec_input = tf.expand_dims(train_targ_ans[:, t], 1)

      train_dec_hidden_q = train_dec_hidden
      #train_dec_hidden_o = train_dec_hidden

  train_batch_loss = (train_loss / int(train_targ_ans.shape[1]))

  train_variables = encoder.trainable_variables + decoder.trainable

  train_gradients = tape.gradient(train_loss, train_variables)

  optimizer.apply_gradients(zip(train_gradients, train_variables))

  return train_batch_loss
```

In [ ]:

```
@tf.function
def cv_step(inp_image, inp_quest, inp_ocr, targ_ans):
  '''
    This function depicts the validation step of the model.
    It takes 4 parameters -
      inp_image : Batched Image tensor of the validation data.
      inp_quest : Batched Question tensor of the validation data.
      inp_ocr : Batched OCR text tensor of the validation data.
      targ_ans : Batched Answer tensor(Target tensor) of the valida
    This function only returns the batch loss of the prediction of
  '''
  loss = 0

  # initializing the hidden state for each batch
  # because the ANSWERS as well as QUESTIONS are not related from i
  img_hidden = decoder.reset_state(batch_size=inp_image.shape[0])
  enc_hidden = encoder.initialize_hidden_state()

  enc_output_i, enc_output_o, enc_hidden_o, enc_output_q, enc_hidde
  dec_hidden_q = enc_hidden_q
  dec_hidden_o = enc_hidden_o

  dec_input = tf.expand_dims([text.word_index['<start>']] * BATCH_S

  # Teacher forcing - feeding the target as the next input
  for t in range(1, targ_ans.shape[1]):
    # passing encoder_output(enc_output) and image_features(image_i
    predictions, dec_hidden, _, _, _ = decoder(dec_input,
                                                 dec_hidden_q,
                                                 dec_hidden_o,
                                                 img_hidden,
                                                 enc_output_i,
                                                 enc_output_o,
                                                 enc_output_q)

    loss += loss_function(targ_ans[:, t], predictions)

    # using teacher forcing
    dec_input = tf.expand_dims(targ_ans[:, t], 1)

    dec_hidden_q = dec_hidden
    #dec_hidden_o = dec_hidden

  batch_loss = (loss / int(targ_ans.shape[1]))

  return batch_loss
```

In [ ]:

```
#Reference - https://keras.io/guides/writing_a_training_loop_from_s
def training_func(epoch_number,epochs):
  '''
  The function "training_func" contains the actual training loop.
  It takes 2 parameters -
    epoch_number : The epoch number from where the training is to b
    epochs : Total number of actual epochs to be looped from the st
  This function saves the checkpoints of the model on every even nu
  It also writes the logs of the training and validation losses for
  PLEASE RUN THE "%load_ext tensorboard" CELL FIRST BEFORE THE TRAI
  The function also prints the time taken for every epoch.
  '''

  EPOCHS = epochs
  avg_batch_loss = 0

  for epoch in range(EPOCHS):
    start = time.time()

    train_dataset = tf.data.Dataset.from_tensor_slices((img_name_tr
    # map to load the numpy files in parallel
    train_dataset = train_dataset.map(lambda item1, item2, item3, i


                                    num_parallel_calls=tf.data.expe

    # Shuffle and batch
    train_dataset = train_dataset.shuffle(TRAIN_BUFFER_SIZE).batch(
    train_dataset = train_dataset.prefetch(buffer_size=tf.data.expe

    train_total_loss = 0
    val_total_loss = 0
    avg_batch_loss = 0

    for train_image_inp, train_inp_q, train_inp_o, train_targ_a in

      train_batch_loss = train_step(train_image_inp, train_inp_q, t
      train_total_loss += train_batch_loss

    for val_image_inp, val_inp_q, val_inp_o, val_targ_a in val_data
      val_batch_loss = cv_step(val_image_inp, val_inp_q, val_inp_o,
      val_total_loss += val_batch_loss

    # saving (checkpoint) the model every even numbered epochs
    if (epoch + 1) % 2 == 0:
      checkpoint.save(file_prefix = checkpoint_prefix)

    #Savings logs for tensorboard
    with train_summary_writer.as_default():
      tf.summary.scalar('Train loss', train_total_loss/steps_per_ep
      tf.summary.scalar('Validation loss', val_total_loss/steps_per

    if (epoch + 1) % 2 == 0:
      print('Epoch {} ====> Time taken : {:.4f} seconds (Checkpoint

    else:
      print('Epoch {} ====> Time taken : {:.4f} seconds'.format(epo
                                                                 ti
```

In [ ]:

```
def training_checkpoint_func(epoch_number, epochs, load_checkpoint=
  '''
    The function "training_checkpoint_func" is used to start or res
    It takes 3(2) parameters-
      epoch_number : The epoch number from where the training is to
      epochs : Total number of actual epochs to be looped from the
      load_checkpoint : The Boolean value for whether the model che
  '''
  if load_checkpoint:
    #restoring the latest checkpoint in checkpoint_dir
    checkpoint.restore(tf.train.latest_checkpoint(checkpoint_dir))
    training_func(epoch_number, epochs)
  else:
    training_func(epoch_number, epochs)
```

In [ ]:

```
# Epoch 1 to Epoch 4
training_checkpoint_func(0,4)
```

```
Epoch 1 ====>  Time taken : 5543.6917 seconds
Epoch 2 ====> Time taken : 5348.7856 seconds (Checkpoint saved)
Epoch 3 ====>  Time taken : 5314.9873 seconds
Epoch 4 ====> Time taken : 5314.1928 seconds (Checkpoint saved)
```

In [ ]:

```
# Epoch 5 to Epoch 6
training_checkpoint_func(4,2,True)
```

```
Epoch 5 ====>  Time taken : 5298.8619 seconds
Epoch 6 ====> Time taken : 5321.0967 seconds (Checkpoint saved)
```

In [ ]:

```
# Epoch 7 to Epoch 14
training_checkpoint_func(6,8,True)
```

```
Epoch 7 ====> Time taken : 5453.1431 seconds
Epoch 8 ====> Time taken : 5353.2369 seconds (Checkpoint saved)
Epoch 9 ====> Time taken : 5319.3721 seconds
Epoch 10 ====> Time taken : 5360.6223 seconds (Checkpoint saved)
Epoch 11 ====> Time taken : 5373.5034 seconds
```

In [ ]:

```
# Epoch 15 to Epoch 16
training_checkpoint_func(14,2,True)
```

```
Epoch 15 ====> Time taken : 5323.0677 seconds
Epoch 16 ====> Time taken : 5290.2460 seconds (Checkpoint saved)
```

In [ ]:

```
# Epoch 17 to Epoch 24
training_checkpoint_func(16,8,True)
```

```
Epoch 17 ====> Time taken : 5227.2561 seconds
Epoch 18 ====> Time taken : 5242.8972 seconds (Checkpoint saved)
Epoch 19 ====> Time taken : 5249.5653 seconds
Epoch 20 ====> Time taken : 5366.5269 seconds (Checkpoint saved)
Epoch 21 ====> Time taken : 5330.4849 seconds
Epoch 22 ====> Time taken : 5161.9509 seconds (Checkpoint saved)
Epoch 23 ====> Time taken : 5122.8673 seconds
Epoch 24 ====> Time taken : 5007.8092 seconds (Checkpoint saved)
```

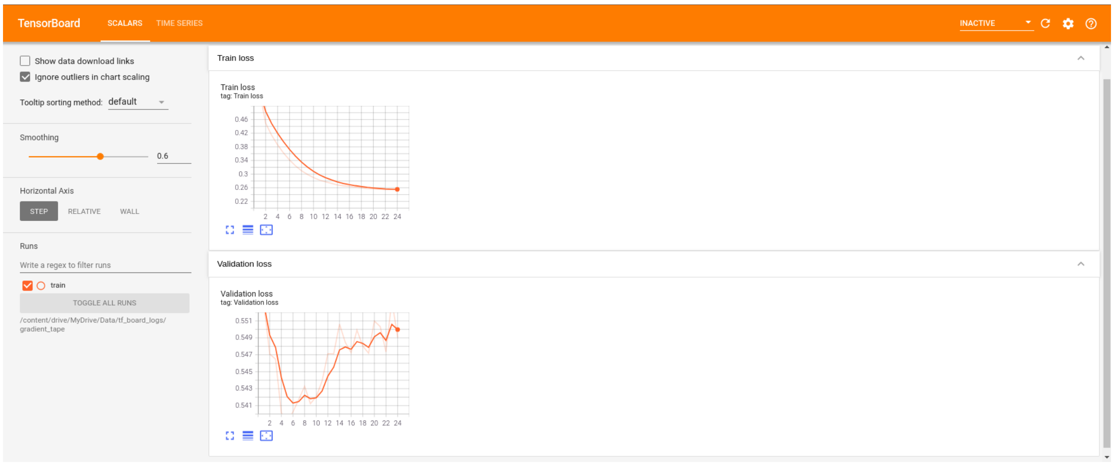
In [ ]:

```
# Epoch 24 to Epoch 26
#training_checkpoint_func(24,2,True)
```

In [ ]:

```
%load_ext tensorboard
%tensorboard --logdir /content/drive/MyDrive/Data/tf_board_logs/gra
```

```
<IPython.core.display.Javascript object>
```



□ Show data download links
☑ Ignore outliers in chart scaling
Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: STEP RELATIVE WALL

Runs

Write a regex to filter runs

☑ ○ train

TOGGLE ALL RUNS

/content/drive/MyDrive/Data/tf_board_logs/gradient_tape

Train loss
tag: Train loss

Validation loss
tag: Validation loss

4.2.3. Attention Plot.

In [20]:

```
checkpoint.restore(checkpoint_dir+'/ckpt-3')
```

Out[20]:

```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at
0x7f4da499d898>
```

In [21]:

```
def preprocess_qa(w):
    '''
      The function 'preprocess_qa' preprocesses both the questions as
    '''
    # creating a space between a word and the punctuation following i
```

```
  # eg: "he is a boy." => "he is a boy ."
  # Reference:- https://stackoverflow.com/questions/3645931/python-
  w = re.sub('([!"#$%&()*+.,-/:;=?@[\]<>?^_`{|}~])', r' \1 ', w)
  w = re.sub('\s{2,}', ' ', w)

  # replacing everything with space except (a-z, A-Z, ".", "?", "!
  w = re.sub('(?<=[A-Za-z])(?=[0-9])|(?<=[0-9])(?=[A-Za-z])',' ', w

  w = ' '.join(e.lower() for e in w.split())

  w = w.strip()

  # adding a start and an end token to the sentence
  # so that the model know when to start and stop predicting.
  w = '<start> ' + w + ' <end>'
  return w

def preprocess_ocr(w):
  with open(w, 'r') as f:
    annotations = json.load(f)
  w = ''
  for i in range(len(annotations['recognitionResults'][0]['lines'])
    w+= ' '+annotations['recognitionResults'][0]['lines'][i]['text'
  w = w.split(' Source: ', 1)[0]
  w = preprocess_qa(w)
  return w
```

```
In [22]: def load_image(image_path):
             img = tf.io.read_file(image_path)
             img = tf.image.decode_jpeg(img, channels=3)
             img = tf.image.resize(img, (299, 299))
             img = tf.keras.applications.inception_v3.preprocess_input(img)
             return img, image_path
```

```
In [23]: image_model = tf.keras.applications.InceptionV3(include_top=False,
                                                         weights='imagenet')
         new_input = image_model.input
         hidden_layer = image_model.layers[-1].output

         image_features_extract_model = tf.keras.Model(new_input, hidden_lay
```

```
In [24]: def evaluate(question, ocr, image):
           '''
             The function 'evaluate' takes 2 parameters:
               sentence = preproccessed Question tokens with <start> and <en
               image = the image features from the InceptionV3 model.

             It returns
               result=the predicted answer is in a sentence format for the Q
               img_result=the predicted answer is in the form of list for th
               sentence=passing the same sentence which has been passed as a
               attention_plot=these are the attention weights for the QUESTI
               image_attention_plot=these are the attention weights for the
           '''
           question_attention_plot = np.zeros((max_length_targ, max_length_i
           image_attention_plot = np.zeros((max_length_targ, attention_featu
           result = ''
           img_result = []
           hidden = tf.zeros((1, units))
```

```
#===== Question Input Preprocess =====
question = preprocess_qa(question)
inputs_q = [text.word_index[i] for i in question.split(' ')]
inputs_q = tf.keras.preprocessing.sequence.pad_sequences([inputs_
                                                         maxlen=m
                                                         padding=
inputs_q = tf.convert_to_tensor(inputs_q)


#===== OCR Input Process =====
temp_ocr_result = preprocess_ocr(ocr)
if len(temp_ocr_result.split())<221:
  ocr = temp_ocr_result
else:
  ocr_words = temp_ocr_result.split()
  temp_final_ocr = ' '.join(ocr_words[:219])
  if '<end>' not in temp_final_ocr:
    temp_final_ocr = temp_final_ocr + ' <end>'
  ocr = temp_final_ocr

inputs_o = ''
if len(ocr.split())==2:
  inputs_o = [text.word_index[i] for i in ['<start>','<end>']]
  inputs_o = tf.keras.preprocessing.sequence.pad_sequences([input
                                                          maxlen=
                                                          padding
  inputs_o = tf.convert_to_tensor(inputs_o)

else:
  inputs_o = [text.word_index[i] for i in ocr.split(' ')]
  inputs_o = tf.keras.preprocessing.sequence.pad_sequences([input
                                                          maxlen=
                                                          padding
  inputs_o = tf.convert_to_tensor(inputs_o)


#=====Image Input Preprocess=====
temp_input = tf.expand_dims(load_image(image)[0], 0)
img_tensor_val = image_features_extract_model(temp_input)
img_tensor_val = tf.reshape(img_tensor_val, (img_tensor_val.shape


#=====Encoder Phase=====
enc_out_i, enc_out_o, enc_hidden_o, enc_out_q, enc_hidden_q = enc




dec_hidden_q = enc_hidden_q
dec_hidden_o = enc_hidden_o

dec_input = tf.expand_dims([text.word_index['<start>']], 0)

#=====Decoder Phase=====
img_hidden = decoder.reset_state(batch_size=1)
for t in range(max_length_targ):
  predictions, dec_hidden, question_attention_weights, _, image_a
```

```
    # storing the attention weights to plot later on
    question_attention_weights = tf.reshape(question_attention_weig
    question_attention_plot[t] = question_attention_weights.numpy()
    image_attention_plot[t] = tf.reshape(image_attention_weights, (

    predicted_id = tf.argmax(predictions[0]).numpy()

    if predicted_id==0:
      result += text.index_word[predicted_id+1] + ' '
      img_result.append(text.index_word[predicted_id+1] + ' ')
    else:
      result += text.index_word[predicted_id] + ' '
      img_result.append(text.index_word[predicted_id] + ' ')
      if text.index_word[predicted_id] == '<end>':
        return result, img_result, question, ocr, question_attentio

    # the predicted ID is fed back into the model
    dec_input = tf.expand_dims([predicted_id], 0)

    dec_hidden_q = dec_hidden
    dec_hidden_o = dec_hidden

  image_attention_plot = image_attention_plot[:len(result), :]
  return result, img_result, question, ocr, question_attention_plot
```

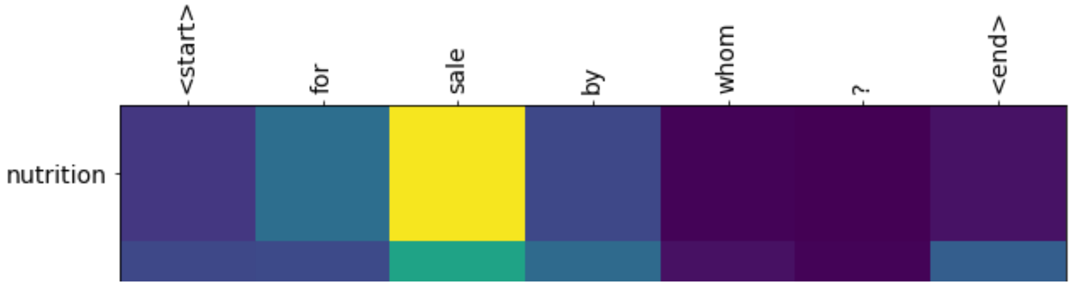
In [25]:
```
# function for plotting the attention weights
def plot_attention(attention, sentence, predicted_sentence):
  fig = plt.figure(figsize=(10,10))
  ax = fig.add_subplot(1, 1, 1)
  ax.matshow(attention, cmap='viridis')

  fontdict = {'fontsize': 14}

  ax.set_xticklabels([''] + sentence, fontdict=fontdict, rotation=9
  ax.set_yticklabels([''] + predicted_sentence, fontdict=fontdict)

  ax.xaxis.set_major_locator(ticker.MultipleLocator(1))
  ax.yaxis.set_major_locator(ticker.MultipleLocator(1))

  plt.show()
```

In [26]:
```
def image_plot_attention(image, result, attention_plot):
    temp_image = np.array(Image.open(image))

    fig = plt.figure(figsize=(10, 10))

    len_result = len(result)
    for l in range(len_result):
        temp_att = np.resize(attention_plot[l], (8, 8))
        ax = fig.add_subplot(len_result, len_result, l+1)
        ax.set_title(result[l])
        img = ax.imshow(temp_image)
        ax.imshow(temp_att, cmap='gray', alpha=0.6, extent=img.get_

    plt.tight_layout()
    plt.show()
```

In [27]:
```
def predict_answer(question, ocr, image_input):
  result, img_result, question, ocr, attention_plot, image_attentic

  print('\n\nQuestion : %s' % (question))
  print('Image OCR Result : ',ocr)
  print('Predicted Answer : {}'.format(result))

  attention_plot = attention_plot[:len(result.split(' ')), :len(que
  plot_attention(attention_plot, question.split(' '), result.split(
  image_plot_attention(image_input,img_result,image_attention_plot)
```

In [28]:
```
indices = random.sample(range(1, train.shape[0]), 20)
for idx in indices:
  predict_answer(list(train['Question'])[idx],list(train['OCR'])[ic
  print("\nActual Answer : ",list(train['Answer'])[idx])
  print("===========================================================
```

```


Question : <start> for sale by whom ? <end>
Image OCR Result :  <start> report of the secretary's commission
on pesticides and their relationship to environmental health heal
th . education ment by 473 m ony noiz u . s . department of healt
h , education , and welfare december 1969 for sale by the superin
tendent of documents , u . s . government printing office washing
ton , d . c . 20402 - price $ 3 . 00 <end>
Predicted Answer : nutrition foundation <end>
```



In [29]:
```
indices = random.sample(range(1, val.shape[0]), 20)
for idx in indices:
  predict_answer(list(val['Question'])[idx],list(val['OCR'])[idx],l
  print("\nActual Answer : ",list(val['Answer'])[idx])
  print("===========================================================
```

4.2.4. ANLS Score.

In [28]:
```
def create_dataset(df, type_of_dataset):
  '''
    The function 'create_dataset' takes 2 paramaeters:
      df = The dataframe of the dataset type.
      type_of_dataset = The type of dataset

    It returns list/s of preprocessed questions and/or answers base
  '''
  if type_of_dataset=='test':
    questions = []
    ocr = []
    for i in tqdm(range(len(df['Question']))):
      questions.append(preprocess_qa(list(df['Question'])[i]))
      temp_ocr_result = preprocess_ocr(list(df['OCR'])[i])
      if len(temp_ocr_result.split())<221:
        ocr.append(temp_ocr_result)
      else:
        ocr_words = temp_ocr_result.split()
        temp_final_ocr = ' '.join(ocr_words[:219])
        if '<end>' not in temp_final_ocr:
          temp_final_ocr = temp_final_ocr + ' <end>'
        ocr.append(temp_final_ocr)
    return questions, ocr

  else:
    questions = []
    answers = []
    ocr = []
    for i in tqdm(range(len(df['Question']))):
      questions.append(preprocess_qa(list(df['Question'])[i]))
      answers.append(preprocess_qa(list(df['Answer'])[i]))
      temp_ocr_result = preprocess_ocr(list(df['OCR'])[i])
      if len(temp_ocr_result.split())<221:
        ocr.append(temp_ocr_result)
      else:
        ocr_words = temp_ocr_result.split()
        temp_final_ocr = ' '.join(ocr_words[:219])
        if '<end>' not in temp_final_ocr:
          temp_final_ocr = temp_final_ocr + ' <end>'
        ocr.append(temp_final_ocr)
    return questions,answers,ocr
```

In [29]:
```
def anls_score(df, df_type):
  images_list = list(df['Image'])
  questions_list = list(df['Question'])
  ocr_list = list(df['OCR'])
  _, answers_list, _ = create_dataset(df,df_type)
  rho=0.5

  nl = NormalizedLevenshtein()
  nl_scores = []
  index_of_incorrect_pred = []
  for image_idx in tqdm(range(len(images_list))):
    result,_,_,_,_,_ =  evaluate(questions_list[image_idx],ocr_list
    actual_answer = answers_list[image_idx]
    actual_answer = actual_answer.replace('<end>','')
    actual_answer = actual_answer.replace('<start>','')
    actual_answer = actual_answer.strip()

    result = result
    result = result.replace('<end>','')
    result = result.strip()

    if round(nl.distance(result,actual_answer),1)<rho:
      nls = nl.similarity(result,actual_answer)
      nl_scores.append(nls)
    else:
      nls = 0.0
      nl_scores.append(nls)
      index_of_incorrect_pred.append(image_idx)
  anl_score = (sum(nl_scores))/(len(nl_scores))
  return anl_score, index_of_incorrect_pred
```

In [33]:

```
# Printing the Train ANLS score
train_score,_ = anls_score(train,'train')
print("\nTrain ANLS Score :",train_score)
```

```
100%|██████████| 39464/39464 [05:33<00:00, 118.43it/s]
100%|██████████| 39464/39464 [2:07:53<00:00,  5.14it/s]


Train ANLS Score : 0.27274390833899553
```

In [30]:

```
# Printing the Validation ANLS score
val_score, indices = anls_score(val,'val')
print("\nValidation ANLS Score :",val_score)
```

```
100%|██████████| 5352/5352 [00:09<00:00, 546.91it/s]
100%|██████████| 5352/5352 [17:21<00:00,  5.14it/s]


Validation ANLS Score : 0.12928868642647526
```

In [33]:

```
print("Number of datapoints with Normalized Levenshtein Similarity
```

```
Number of datapoints with Normalized Levenshtein Similarity as 0
are :  4481
```

In [67]:

```
def show_images(df, number_of_points, idxs):
  images_list = list(df['Image'])
  questions_list = list(df['Question'])
  ocr_list = list(df['OCR'])
  answer_list = list(df['Answer'])
  #random_idxs = [ for i in range(number_of_points)]
  random_idxs = random.choices(population=idxs, k=20)
  for i in random_idxs:
    #i = random.choice(idxs)
    idxs.remove(i)
    print("Question : ",questions_list[i])
    ocr = ''
    temp_ocr_result = preprocess_ocr(ocr_list[i])
    if len(temp_ocr_result.split())<221:
      ocr = temp_ocr_result
    else:
      ocr_words = temp_ocr_result.split()
      temp_final_ocr = ' '.join(ocr_words[:219])
      if '<end>' not in temp_final_ocr:
        temp_final_ocr = temp_final_ocr + ' <end>'
      ocr = temp_final_ocr
    ocr = ocr.replace('<start> ','')
    ocr = ocr.replace(' <end>','')
    print("OCR : ",ocr)
    img = mpimg.imread(images_list[random_idxs[0]])
    imgplot = plt.imshow(img)
    plt.show()
    result,_,_,_,_ =  evaluate(questions_list[i],ocr_list[i],imag
    result = result.replace(' <end>','')
    print("Actual Answer : ", answer_list[i])
    print("Predicted Answer : ", result)
    print("\n\n")
    print('=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*
    print("\n\n")
```

This analysis is done to figure out where the model is going wrong in predicting the right answers. This case has been derived from where the predictions have achieved a Normalised Levenshtein Similarity of 0.0.

In [72]:

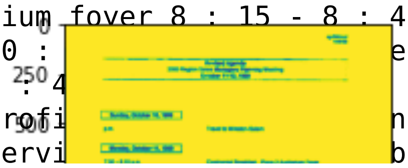
```
#Images with difficult diagrams
show_images(val_10.indices)
```

```
Question :  name of the company mentioned at the top right of the
page?
OCR :  the national sugar refining company comparison of basic ho
urly wage rates for male workers and cost of living in various su
gar refining centers 1941 - 2 basic cost hourly of rate living bo
ston $ 1 , 472 . new york 7730 $ 1 . 553 . philadelphia 684 $ 1 ,
383 . baltimore $ 1 . 384 . san francisco 794 $ 1 , 514 . savanna
h 39 d - 42 # $ 1 . 369 . 1942 new orleans 1941 new orleans 320 -
40 # $ 1 . 323 . 1942 galveston 1941 galveston 47 2 # $ 1 . 339 .
* 4 cost of living data from u . s . bureau of labor statistics ,
as of june 15 , 1941 : basic hourly wage rates from various union
contracts . actually , jacksonville , florida 45 actually , houst
on , texas march 26 , 1942
```

In [73]:

```
#Images with somewhat tabular information
show_images(val_10.indices)
```

```
Question :  when is the 'break' after lunch, on october 11?
OCR :  sgr 044 / ocd 10 / 8 / 99 revised agenda 2000 region sales
managers planning meeting october 11 - 12 , 1999 sunday , october
10 , 1999 p . m . travel to winston - salem monday , october 11 ,
1999 7 : 30 - 8 : 15 a . m continental breakfast - plaza 2 audito
rium foyer 8 : 15 - 8 : 45 a . m . opening - jim maguire 8 : 45 -
10 :                     e / marketing strategies - lynn beasley 1
0 : 4                     break 11 : 00 - 11 : 30 a . m . channel
profi                     nnell 11 : 30 - 11 : 45 a . m strategic o
vervi                     bryan stockdale , richard cross 11 : 45 a
```

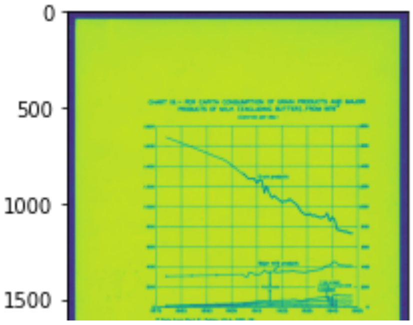


In [71]:
```
# Images with some handwritten texts
show images(val.10.indices)
```

```
Question :  from which year onwards american meat institute servi
ng the meat industry?
OCR :  american rec'd apr 1 5 1982 meat institute serving the mea
t industry since 1906 march 31 , 1982 memorandum to : fmi worksho
p speakers from : mem re : " odds and ends " fmi staff reminds me
to urge you to return your permission to record forms . if you ha
ve not sent it in , ( it is in the fmi speakers manual ) i have e
nclosed another copy for your convenience . for those of you who
need equipment and have not let me know , fmi also needs a listin
g of it soon . just call me ! p . o . box 3556 , washington , d .
c . 20007 . 1700 north moore street , arlington , va . 22209 . 70
3 / 841 - 2400
```

In [74]:
```
# Images with diagrams
show images(val.10.indices)
```

```
Question :  the per capita consumption of which product has decre
ased?
OCR :  chart 19 . - per capita consumption of grain products and
major products of milk ( excluding butter ) , from 1879 ( calorie
s per day ) 1 , 800 1 , 800 1 , 6 00 1 , 600 1 , 400 1 , 400 grai
n products 1 , 200 1 . 200 1 , 000 1 , 000 800 800 600 600 400 ma
jor milk products 400 i . ice cream 200 - fluid milk 2 . evap . +
cond . milk cheese 200 1875 1885 1895 1905 1915 1925 1935 1945 19
55 * data from part e , tables vi - a , viii , ix .
```



In [75]:
```
# Images with pictures and huge texts
show images(val.10.indices)
```

```
Question :  what is the position of william w. moore ?
OCR :  executive vice president's message officers board of direc
tors richard d . dotts walter h . abelmann , m . d . as a meeting
place and resource center for chairman of the board boston , mass
. the dedicated workers of the american john t . shepherd , m . d
. , d . sc . william h . ames , m . d . heart association , our n
ew national center president st . joseph , mo . now enables close
r relationships and harriet p . dustan , m . d . john s . andrews
communications with 55 affiliates and 1 , 196 president - elect y
oungstown , ohio chapters and units . it is also national elliot
rapaport , m . d . philip p . ardery headquarters of the associat
ion . immediate past president louisville , ky . planned and exec
uted with long - term ross reid adm . philip f . ashler savings i
n mind , this relocation is but a part immediate past chairman of
the board tallahassee , fla . w . gerald austen , m . d . of the
association's strategy to refine vice presidents boston , mass .
management practices and to implement owen beard , m . d economie
```

In [78]:

```
show_images(val,10,indices)
```

```
Question :  what is the title of the second table (bottom one) in
this page?
OCR :  progress report 10 totelle img client : wyeth august 16 ,
2002 project classification target status / action publication da
te / journal poster a 6 endometrium / safety wmc a comparative 2
- year study of two berlin , june 2002 poster presented sequentia
l regimens of img estradiol and trimegestone with 1 mg estradiol
and norethisterone upon profiles of endometrial bleeding and safe
ty in postmenopausal women p . koninckx , d . spielmann and the t
rimegestone 302 study group poster a 7 metabolic wmc blood lipid
profiles in postmenopausal women on either a impact / hemostasis
berlin , june 2002 poster presented sequential regimen of img est
radiol and trimegestone or 1 mg estradiol and norethisterone over
a 1 - year period p . koninckx , d . spielmann and the trimegesto
ne 302 study group poster a 8 hemostasis wmc berlin , june 2002 p
oster presented a 1 - year comparative assessment of the hemostat
ic profile of postmenopausal women following a sequential regimen
of img estradiol combined with either trimegestone or norethister
one c . kluft , d . spielmann and the trimegestone 302 study grou
```

Thus, the model retains the format of how the answers are supposed to be surprisingly based on the question and the OCR text even if the predicted answers differ. Usually the answers which are to be predicted from tables, diagrams, handwritten texts and images/photos(eg: piechart or a picture of an individual) which are present in the document image are difficult to figure out.

4.2.5. Predict Test answers

In [ ]:
```
def predict(df):
  images_list = list(df['Image'])
  questions_list = list(df['Question'])
  ocr_list = list(df['OCR'])
  results = []
  for image_idx in tqdm(range(len(images_list))):
    result,_,_,_,_,_ =  evaluate(questions_list[image_idx],ocr_list
    results.append(result)
  return results
```

In [ ]:
```
lst = predict(test)
```

```
100%|██████████| 5188/5188 [18:30<00:00,  4.67it/s]
```

In [ ]:
```
pred = pd.DataFrame(lst)
pred.head()
```

Out[32]:

	0
0	$ 165 <end>
1	200 <end>
2	$ 165 <end>
3	8 <end>
4	the united corporation <end>

In [ ]:
```
pred.to_csv('/content/drive/MyDrive/Data/test_results_6_chckpnt.csv
```

4.2.6. Submitting results.

In [ ]:
```
answer_df = pd.read_csv('/content/drive/MyDrive/Data/test_results_6
test = pd.read_csv('/content/drive/MyDrive/Data/test.csv')
```

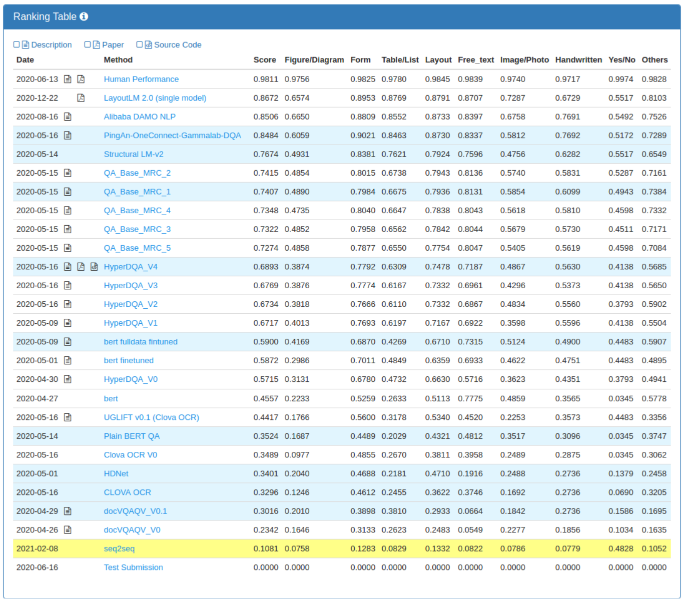
In [ ]:
```
results_list = []
for i in tqdm(range(len(test['Question_Id']))):
  temp_dict = {}
  temp_answer = list(answer_df['0'])[i]
  temp_answer = temp_answer.replace('<end>','')
  temp_answer = ' '.join(temp_answer.split())
  temp_dict['answer'] = temp_answer
  temp_dict['questionId'] = list(test['Question_Id'])[i]
  results_list.append(temp_dict)
```

```
100%|██████████| 5188/5188 [00:03<00:00, 1391.59it/s]
```

In [ ]:
```
with open("/content/drive/MyDrive/Data/model_6_chckpnt_result.json'
  json.dump(results_list, outfile)
```

4.2.7. Test Result.

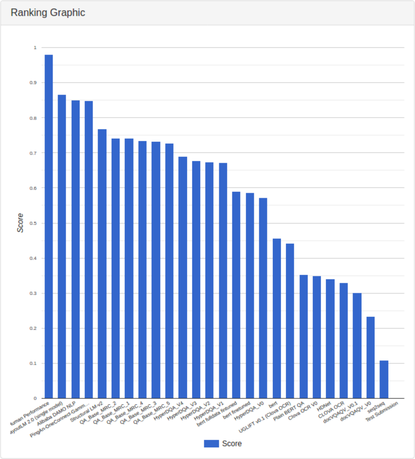
The second model achieved a Test ANLS score of 0.1081 after submitting the results on the RRC (https://rrc.cvc.uab.es/?ch=17&com=evaluation&task=1) website. Thereby you can see that the analysis which was done on the validation data is the same for the test set too as mentioned in the results table.



Ranking Table ℹ

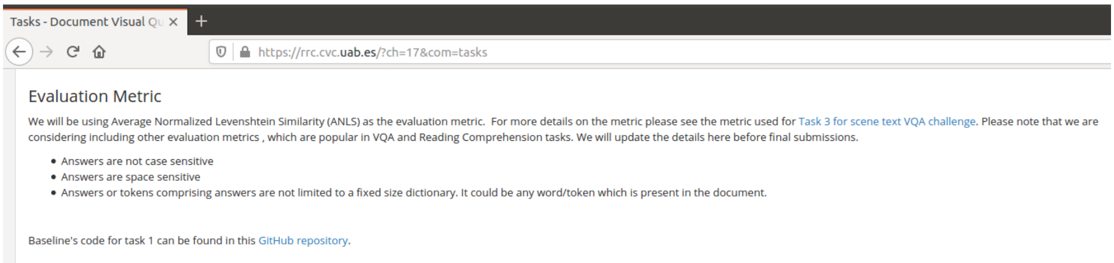
□ Description □ Paper □ Source Code

Date	Method	Score	Figure/Diagram	Form	Table/List	Layout	Free_text	Image/Photo	Handwritten	Yes/No	Others
2020-06-13	Human Performance	0.9811	0.9756	0.9825	0.9780	0.9845	0.9839	0.9740	0.9717	0.9974	0.9828
2020-12-22	LayoutLM 2.0 (single model)	0.8672	0.6574	0.8953	0.8769	0.8791	0.8707	0.7287	0.6729	0.5517	0.8103
2020-08-16	Alibaba DAMO NLP	0.8506	0.6650	0.8809	0.8552	0.8733	0.8397	0.6758	0.7691	0.5492	0.7526
2020-05-16	PingAn-OneConnect-Gammalab-DQA	0.8484	0.6059	0.9021	0.8463	0.8730	0.8337	0.5812	0.7692	0.5172	0.7289
2020-05-14	Structural LM-v2	0.7674	0.4931	0.8381	0.7621	0.7924	0.7596	0.4756	0.6282	0.5517	0.6549
2020-05-15	QA_Base_MRC_2	0.7415	0.4854	0.8015	0.6738	0.7943	0.8136	0.5740	0.5831	0.5287	0.7161
2020-05-15	QA_Base_MRC_1	0.7407	0.4890	0.7984	0.6675	0.7936	0.8131	0.5854	0.6099	0.4943	0.7384
2020-05-15	QA_Base_MRC_4	0.7348	0.4735	0.8040	0.6647	0.7838	0.8043	0.5618	0.5810	0.4598	0.7332
2020-05-15	QA_Base_MRC_3	0.7322	0.4852	0.7958	0.6562	0.7842	0.8044	0.5679	0.5730	0.4511	0.7171
2020-05-15	QA_Base_MRC_5	0.7274	0.4858	0.7877	0.6550	0.7754	0.8047	0.5405	0.5619	0.4598	0.7084
2020-05-16	HyperDQA_V4	0.6893	0.3874	0.7792	0.6309	0.7478	0.7187	0.4867	0.5630	0.4138	0.5685
2020-05-16	HyperDQA_V3	0.6769	0.3876	0.7774	0.6167	0.7332	0.6961	0.4296	0.5373	0.4138	0.5650
2020-05-16	HyperDQA_V2	0.6734	0.3818	0.7666	0.6110	0.7332	0.6867	0.4834	0.5560	0.3793	0.5902
2020-05-09	HyperDQA_V1	0.6717	0.4013	0.7693	0.6197	0.7167	0.6922	0.3598	0.5596	0.4138	0.5504
2020-05-09	bert fulldata fintuned	0.5900	0.4169	0.6870	0.4269	0.6710	0.7315	0.5124	0.4900	0.4483	0.5907
2020-05-01	bert finetuned	0.5872	0.2986	0.7011	0.4849	0.6359	0.6933	0.4622	0.4751	0.4483	0.4895
2020-04-30	HyperDQA_V0	0.5715	0.3131	0.6780	0.4732	0.6630	0.5716	0.3623	0.4351	0.3793	0.4941
2020-04-27	bert	0.4557	0.2233	0.5259	0.2633	0.5113	0.7775	0.4859	0.3565	0.0345	0.5778
2020-05-16	UGLIFT v0.1 (Clova OCR)	0.4417	0.1766	0.5600	0.3178	0.5340	0.4520	0.2253	0.3573	0.4483	0.3356
2020-05-14	Plain BERT QA	0.3524	0.1687	0.4489	0.2029	0.4321	0.4812	0.3517	0.3096	0.0345	0.3747
2020-05-16	Clova OCR V0	0.3489	0.0977	0.4855	0.2670	0.3811	0.3958	0.2489	0.2875	0.0345	0.3062
2020-05-01	HDNet	0.3401	0.2040	0.4688	0.2181	0.4710	0.1916	0.2488	0.2736	0.1379	0.2458
2020-05-16	CLOVA OCR	0.3296	0.1246	0.4612	0.2455	0.3622	0.3746	0.1692	0.2736	0.0690	0.3205
2020-04-29	docVQAQV_V0.1	0.3016	0.2010	0.3898	0.3810	0.2933	0.0664	0.1842	0.2736	0.1586	0.1695
2020-04-26	docVQAQV_V0	0.2342	0.1646	0.3133	0.2623	0.2483	0.0549	0.2277	0.1856	0.1034	0.1635
2021-02-08	seq2seq	0.1081	0.0758	0.1283	0.0829	0.1332	0.0822	0.0786	0.0779	0.4828	0.1052
2020-06-16	Test Submission	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000



The Test result is again based again on the basis of where the actual test data answers do not have any whitespaces between the punctuations and the predicted answers do have it.

This is because they have mentioned that the ANLS metric used during submission is ofcourse space sensitive.


Tasks - Document Visual Qu ×
+
https://rrc.cvc.uab.es/?ch=17&com=tasks

Evaluation Metric

We will be using Average Normalized Levenshtein Similarity (ANLS) as the evaluation metric. For more details on the metric please see the metric used for Task 3 for scene text VQA challenge. Please note that we are considering including other evaluation metrics , which are popular in VQA and Reading Comprehension tasks. We will update the details here before final submissions.

- Answers are not case sensitive
- Answers are space sensitive
- Answers or tokens comprising answers are not limited to a fixed size dictionary. It could be any word/token which is present in the document.

Baseline's code for task 1 can be found in this GitHub repository.