**Dependencies**

import os

import math

import warnings

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import joblib

import cv2

import imgaug.augmenters as iaa

import tensorflow as tf

from tensorflow.keras.layers import Input, Dense, Embedding, LSTMDot, Reshape, Concatenate, BatchNormalization, GlobalAveragePooling2D, GlobalMaxPooling2D, Dropout, Add, MaxPooling2D, GRU, AveragePooling2D

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from nltk.translate.bleu\_score import sentence\_bleu

tf.compat.v1.enable\_eager\_execution()

warnings.filterwarnings("ignore")

sns.set(palette='muted',style='white')

**Tokenizer Setup**

tokenizer = Tokenizer(filters = '',oov\_token = '<unk>')

tokenizer.fit\_on\_texts(train.impression\_final.values)

train\_captions = tokenizer.texts\_to\_sequences(train.impression\_final)

test\_captions = tokenizer.texts\_to\_sequences(test.impression\_final)

vocab\_size = len(tokenizer.word\_index)

caption\_len = np.array([len(i) for i in train\_captions])

start\_index = tokenizer.word\_index['<cls>'

end\_index = tokenizer.word\_index['<end>']

**Input Data Pipeline**

class Dataset():

  def \_\_init\_\_(self,df,input\_size,tokenizer = tokenizer, augmentation = True, max\_pad = max\_pad):

    self.image1 = df.image\_1

    self.image2 = df.image\_2

    self.caption = df.impression\_ip

    self.caption1 = df.impression\_op

    self.input\_size = input\_size

    self.tokenizer = tokenizer

    self.augmentation = augmentation

    self.max\_pad = max\_pad

    self.aug1 = iaa.Fliplr(1) # horizontal flip images

    self.aug2 = iaa.Flipud(1) # vertical flip images

  def \_\_len\_\_(self):

    return len(self.image1)

  def \_\_getitem\_\_(self,i):

    image1 = cv2.imread(self.image1[i],cv2.IMREAD\_UNCHANGED)/255

    image2 = cv2.imread(self.image2[i],cv2.IMREAD\_UNCHANGED)/255

    if image1.any()==None:

      print("%i , %s image sent null value"%(i,self.image1[i]))

    if image2.any()==None:

      print("%i , %s image sent null value"%(i,self.image2[i]))

    caption = self.tokenizer.texts\_to\_sequences(self.caption[i:i+1])

    caption = pad\_sequences(caption,maxlen = self.max\_pad,padding = 'post')

    caption = tf.squeeze(caption,axis=0)

    if self.augmentation:

          a = np.random.uniform()

          if a < 0.333:

              image1 = self.aug1.augment\_image(image1)

              image2 = self.aug1.augment\_image(image2)

          elif a < 0.667:

              image1 = self.aug2.augment\_image(image1)

              image2 = self.aug2.augment\_image(image2)

    return image1,image2,caption,caption1

**ChexNet (Transfer Learning Model)**

def create\_chexnet(chexnet\_weights = chexnet\_weights,input\_size = input\_size):

  model = tf.keras.applications.DenseNet121(include\_top=False, input\_shape = input\_size)

  x = model.output

  x = GlobalAveragePooling2D()(x)

  x = Dense(14, activation="sigmoid", name="chexnet\_output")(x)

  chexnet = tf.keras.Model(inputs = model.input,outputs = x)

  chexnet.load\_weights(chexnet\_weights)

  chexnet = tf.keras.Model(inputs = model.input,outputs = chexnet.layers[-3].output)  #3rd last layer

  return chexnet

**Image Encoder Layer**

class Image\_encoder(tf.keras.layers.Layer):

  def \_\_init\_\_(self, name = "image\_encoder\_block"):

    super().\_\_init\_\_()

    self.chexnet = create\_chexnet()

    self.chexnet.trainable = False

    self.avgpool = AveragePooling2D()

    #   self.chexnet.layers[-i].trainable = True

  def call(self,data):

    op = self.chexnet(data) # (None,7,7,1024)

    op = self.avgpool(op) # (None,3,3,1024)

    return op

**Image Encoder Block**

def encoder(image1,image2,dense\_dim = dense\_dim,dropout\_rate = dropout\_rate):

  im\_encoder = Image\_encoder()

  bkfeat1 = im\_encoder(image1) # (None,9,1024)

  bk\_dense= Dense(dense\_dim,name = 'bkdense',activation = 'relu')

  bkfeat1 = bk\_dense(bkfeat1)

  bkfeat2 = im\_encoder(image2) # (None,9,1024)

  bkfeat2 = bk\_dense(bkfeat2) # (None,9,512)

  concat = Concatenate(axis=1)([bkfeat1,bkfeat2]) #(None,18,1024)

  bn = BatchNormalization(name = "encoder\_batch\_norm")(concat)

  dropout = Dropout(dropout\_rate,name = "encoder\_dropout")(bn)

  return dropout

**Global Attention Layer**

class global\_attention(tf.keras.layers.Layer):

  def \_\_init\_\_(self,dense\_dim = dense\_dim):

    super().\_\_init\_\_()

    self.W1 = Dense(units = dense\_dim)

    self.W2 = Dense(units = dense\_dim)

    self.V = Dense(units = 1)

  def call(self,encoder\_output,decoder\_h):

    decoder\_h = tf.expand\_dims(decoder\_h,axis=1)

    tanh\_input = self.W1(encoder\_output) + self.W2(decoder\_h)

    tanh\_output =  tf.nn.tanh(tanh\_input)

    attention\_weights = tf.nn.softmax(self.V(tanh\_output),axis=1)

    op = attention\_weights\*encoder\_output

    context\_vector = tf.reduce\_sum(op,axis=1)

    return context\_vector,attention\_weights