**Dependencies**

import os

import cv2

import joblib

import numpy as np

import pandas as pd

import tensorflow as tf

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

from tensorflow.keras.preprocessing.text import Tokenizer

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

from tensorflow.keras.layers import GlobalMaxPooling2D, Dropout, Add, MaxPooling2D, GRU, AveragePooling2D

from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Input, Embedding, LSTM, Dot, Reshape, Concatenate, BatchNormalization

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['<start>']

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.img1 = df.image\_1

    self.img\_2 = 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.img1)

  def \_\_getitem\_\_(self,i):

    img1 = cv2.imread(self.img1[i],cv2.IMREAD\_UNCHANGED)/255

    img\_2 = cv2.imread(self.img\_2[i],cv2.IMREAD\_UNCHANGED)/255

    if img1.any()==None:

      print("%i , %s image sent null value"%(i,self.img1[i]))

    if img\_2.any()==None:

      print("%i , %s image sent null value"%(i,self.img\_2[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:

              img1 = self.aug1.augment\_image(img1)

              img\_2 = self.aug1.augment\_image(img\_2)

          elif a < 0.667:

              img1 = self.aug2.augment\_image(img1)

              img\_2 = self.aug2.augment\_image(img\_2)

**ChexNet (Transfer Learning Model)**

def create\_chexnet(weights=chexnet\_weights, input\_size=(224, 224)):

    model = tf.keras.applications.DenseNet121(

        include\_top=False, input\_shape=input\_size+(3,))

    out = model.output

    out = GlobalAveragePooling2D()(out)

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

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

    chexnet.load\_weights(weights)

    chexnet = tf.keras.Model(

        inputs=model.input, outputs=chexnet.layers[-3].output)

    return chexnet

**Image Encoder Layer**

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

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

        super().\_\_init\_\_()

        self.chexnet = create\_chexnet(input\_size=(224, 224))

        self.chexnet.trainable = False

        self.avgpool = AveragePooling2D()

    def call(self, data):

        output = self.chexnet(data)

        output = self.avgpool(output)

        output = tf.reshape(

            output, shape=(-1, output.shape[1]\*output.shape[2], output.shape[3]))

        return output

**Image Encoder Block**

def encoder(img1, img2, dense\_dim, dropout\_rate):

    img\_enc = Image\_encoder()

    dense = Dense(dense\_dim, name='enc\_dense',

                  activation='relu')

    imf1 = img\_enc(img1)

    imf1 = dense(imf1)

    imf2 = img\_enc(img2)

    imf2 = dense(imf2)

    concat = Concatenate(axis=1)([imf1, imf2])

    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):

        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)

        output = attention\_weights\*encoder\_output

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

        return context\_vector, attention\_weights