Image Generation from Text Using COCO Images and Texts

Ata Berk Çakır, Berkay Taşkın, Buğra Yiğit Bulat, Taha Mermer
Electrical and Electronics Engineering Department, Bilkent University, Turkey
EEE 443 Neural Networks Spring 2021 Final Project

You can use this link for our work in collab an you can run in it to see results. (https://drive.google.com/drive/folders/16tlZfLdxiPAPKKc20si9v5BPcAfoXkXC?usp=sharing)

Abstract

The aim of this work is to generate images from the textual description of some images, for example, generating an image of a zebra on a grass field when the caption "a zebra standing on a grass field." The method for doing such a task can be summarized as encoding a text and feeding it as an input to a conditional GAN (Generative Adversarial Network). The term conditional indicates that we want distinct outputs i.e., images as the inputs change. In this paper, we will mainly explain our research about this topic since we couldn't get desirable results in our implementations.

1. Introduction

Image generation from the text has an enormous impact on various machine learning issues. From Deep Learning perception, text to image transformation is a crucial concept. By business context, this method is used in chatbots to create images and into search engines for generating images while the search is not adequate. The texts which are assigned in our dataset contain some descriptions about the images that we will generate. To transform the text to pixels, we used GAN structure which is formulated with two neural patterns. These two designs detect, grasp, repeat, and reproduce the diversities in a dataset [1]. The forms of GAN are applied to get better outcomes. Organization of characters is also effective with RNN (Recurrent Neural Networks). Discriminative vector approach from text is to be learned. The GAN method that we operate comprises generator and discriminator. To differentiate real data and the generated data, the discriminator is used. For training data, there are real and fake bases as data [2]. The generator is utilized to generate the aforementioned data. The issue is caused by grasping the idea of the text so that the visualization becomes an important issue. For these inferences, some words can be misunderstood. Therefore, accepting text as images is much simpler due to the attraction of images. Understanding Deep Learning is the key concept of the project. Deep learning concept is applied in various fields such as translating, object detection, and converting text to images. Artificial neural networks are the key of deep learning for the text to image conversion and generation because of aiding to process data [3]. In our project, we are required to implement a direct conversion method for text to images. To seize and grasp the text and its context, we can implement RNN. To get images, GAN is used and for the relation between input and images to be generated,

conditional-GAN is used. Conditional GAN enhances operation of GAN. Along with the generator and discriminator part of conditional GAN, we also have a text encoder to understand the text.

2. Methods

To implement the conditional GAN, we have used the COCO dataset, and preprocessing of this data includes several methods. In addition to that, there are a lot of possible implementations of conditional GANs [4], such as “Imagen”, “Lafite”, “DALL-E” etc. However, we have tried to implement the conditional GAN as simply as possible since the high performance conditional GANs required a lot of computational power and deeper understanding of the concept.

2.1 Data Processing

The dataset we use is the dataset provided to us, which is the COCO dataset. In this dataset there are multiple feature datasets, which are, “train_url”, “train_cap”, “train_imid” for train dataset, and “test_url”, “test_cap”, “test_imid” for test dataset, and finally “word_count” including words and their indexes. There were also two other datasets including pre-trained feature vectors, “train_ims” and “test_ims” which we didn’t use.

The dataset “train_url” includes 82783 links that represent an image, however around 10% of it is useless since the images in those links are corrupted or vanished. These Flickr images contain a variety of visuals, which makes the task harder since the GAN needs to train for so many different visuals. The set url set for the test set is similar to “train_url” ; it has some missing images and it has a variety of visuals. Since this dataset includes missing images, we formed our data according to the indexes that have an image. However, there were also grayscale images which was a problem because the network we will build will take RGB images as input so that we eliminated them as well. To eliminate URLs that do not have an image we used a “try-except” method according to error, according to “PIL.UnidentifiedImageError”. To eliminate the non RGB images we simply looked at the dimensions of the image array.

The dataset “train_cap” includes the captions for images represented in the “train_url” dataset. The dataset for testing is similar. The “train_cap” dataset includes the word indexes stored in the “word_count” array and with those indexes every row forms a caption. Every caption is formed by 17 words. The sentences which have less than 17 words are filled with zeros until it has a length of 17, and the first word and last word are “x_START_” and “x_END_” which are parity words. We took the values for this dataset directly (sentence formed by indexes) when we wanted to encode it with a pre-trained RNN model or we took it as a sentence formed by words when we wanted to encode it with GloVe, since the RNN network will not accept a string input such as words we used indexes, and GloVe

can give us a 50-dimensional vector from a pre-created txt file just by importing the vector from the corresponding word.

The dataset "train_imid" includes the indexes of "train_url" where the purpose is to show which sentence leads to which image. The index in the "train_imid" is the corresponding image of the sentence which can be accessed by the URL available on that index. We have found the indexes of images in this list, and returned every sentence that has the desired image's index.

Finally, there is the dataset "word_count" which is simply a dictionary for the words that are present in the given sentences. However not all of the words are meaningful, in the dataset there were words "x_START_, x_NULL_, x_END, x_UNK_" we have eliminated these words in the code. There were also words "xWhile" and "xFor", which were eliminated as well since the words do not represent a figure by themselves.

We have resized the images as 64x64 for reducing the required computational cost. The following figures show some of the 64x64 figures we imported and resized from Flickr.

Figure 1. Some of the imported and resized images

To create a meaningful input to our network from the dataset we tried different embedding methods. First method is about embedding the sentences and images (as features) to create a joint embedding. The sentence embedding is done with GloVe, which is just importing the representations from the GloVe dataset and concatenating the word embeddings, and to embed the images (extract features) we have used the VGG-16 a CNN pre-trained model.[5] The idea of jointly embedding these encodings come from the need to having distinct inputs i.e., conditions to feed the conditional GAN we have tried to implement. Since, if the conditional GAN does not have distinct inputs that represent different images and sentences, it would be difficult for it to find proper weights to generate distinct images. The GloVe embedding we derived from the GloVe dataset is a 750x1 vector since every word is a 50x1 vector and our sentences have a length of 15 (excluding start and stop words from every sentence). In the image embedding, we obtained a 4096x1 vector for each image. We tried to do the joint embedding [5][6] and form a multi model space with the following algorithm.

$$(W_x^*, W_y^*) = argmax_{W_x, W_y} corr(W_x^T X, W_y^T Y) \quad (1)$$

Where the X and Y are embeddings, we have found from VGG-16 and GloVe and we try to find the W_x and W_y which are the encodings of image and text in the multi model space. The term multi model space can be illustrated as in the following figure.

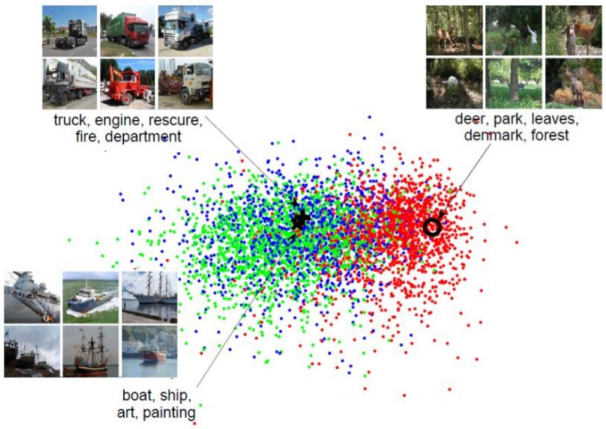

Figure 2. The multi model space [5]

As seen in the figure 2, the images and the captions are close and uncorrelated images and text are far from each other in the multi model space. This joint embedding is a proper input for a conditional GAN to have since it represents the sentence and image data very distinctively, but finding this embedding for all the available data requires a lot of computational power and we couldn't manage to do it properly. Therefore, we have used a simpler encoding model and we encoded only sentences.

To do the sentence encoding we have tried TensorFlow's embedding method [7], which has the sole purpose of converting a set of positive integers into different vectors of desired length. In the end we were able to achieve a 128x1 encoding for every image. The structure we used for this text embedding is a RNN [8] which is a popular NLP encoder model as we found on our research [9]. We have used TensorFlow, Keras libraries to implement it and we used a GRU (Gated Recurrent Units) layer which is very similar to LSTM, as it is stated that the GRU reduces the computational power [9].

2.2 Conditional Generative Adversarial Network

We tried to implement a deep convolutional generative adversarial network (DC-GAN) as proposed in the paper [6]. The algorithm behind a GAN is a two-sided neural network competing against each other. These networks are Generator network and Discriminator network. The algorithm is that Generator tries to generate an image according to the conditioning input (which is encoded sentences) and the Discriminator tries to determine whether the generated image is real or fake. The mathematical expression for this algorithm can be expressed as [6],

$$min_G max_D\, V(D,G) \;=\; E_{x\to p_{data}(x)}[logD(x)] + E_{x\to p_z(z)}[log(1-D(G(z)))] \qquad (2)$$

Since the minimax game [6] represented above is a two-sided optimization, the convergence of the GANs is not clear and it is hard to find, as the loss functions for the generator and discriminator are separate. The generator tries to maximize, $log(\,D(G(z))\,)$ and discriminator tries to maximize $log(\,D(x)\,)$. The structure of the network is as shown in figure 3.

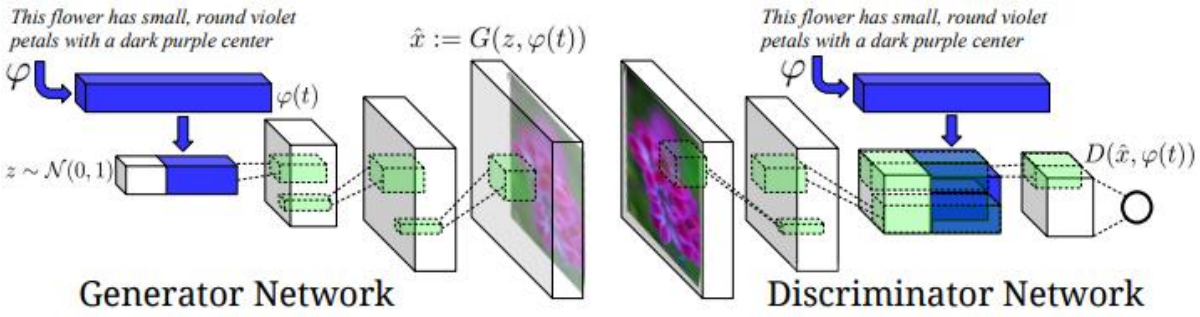

Figure 3. The structure of the conditional GAN [6]

After encoding the text, we concatenated the 128x1 input text vector with random noise and gave it as an input to the network [6]. The reason that we use random noise in inputs is to prevent overfitting issues. The overfitting can happen if there is no noise since similar images with similar meaning captions cannot mix with each other i.e., one of them will be generated clearly while other might be faulty, according to the number of encounters.

We have used the TensorFlow and Keras libraries to implement this GAN and through our implementation we have followed the methods that are mostly used [6][10] such as LeakyReLU activation functions between layers, the only difference between LeakyReLU and normal ReLU is that leaky ReLU has a small slope for negative values while normal ReLU has zero slope. If we use normal ReLU and if the network starts to produce nothing but zero outputs the learning stops as the network is stuck, so usage of leaky ReLU solves this problem. [6][10] The optimizers we use in this project are Adam optimizers, both in generator and discriminator. We decided to use Adam optimizer since we wanted to see the results generated from batches faster and we did not want to have a high computational power required optimizer. The batch size we used is 64 and the learning rate we determined was 0.001.

3. Results

Firstly, for the training setup we have created a “train_step” function in our code that returns generator and discriminator loss after each step and updates the model. Also, in order to visualize the progress in the created image by network we have implemented the “test_step” function. Train function iterates according to the epoch number after each 50 epoch it creates a checkpoint and saves the model to drive under corresponding directory for further use. Also, after every 100th epoch we wanted to display the progress by picking random 8 sentences from the test set and show the constructed images by using generator class. Parameters of the model can be seen in the following figure.

```
hparas = {
        'MAX_SEQ_LENGTH': 15,
        'EMBED_DIM': 256,
        'VOCAB_SIZE': 1004,
        'RNN_HIDDEN_SIZE': 128,
        'Z_DIM': 512,
        'DENSE_DIM': 128,
        'IMAGE_SIZE': [64, 64, 3],
        'BATCH_SIZE': 64,
        'LR': 1e-4,
        'LR_DECAY': 0.5,
        'BETA_1': 0.5,
        'N_EPOCH': 600,
        'N_SAMPLE': len(df),
        'PRINT_FREQ': 100,
        }
```

Figure 4. Model Parameters

Unfortunately, our implementation is not working properly right now. The constructed images after each 100th epoch can be seen below.

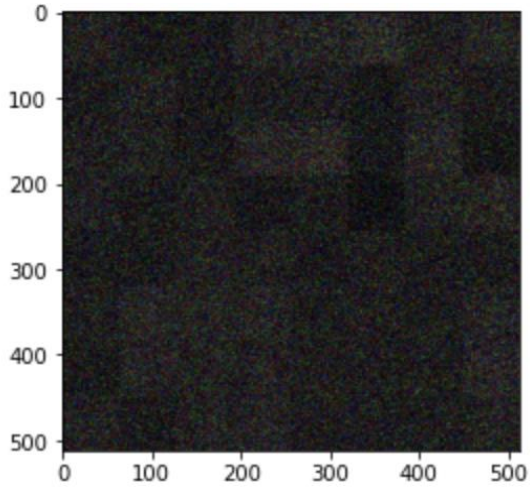

Figure 5. Constructed Images After Each 100th Epoch

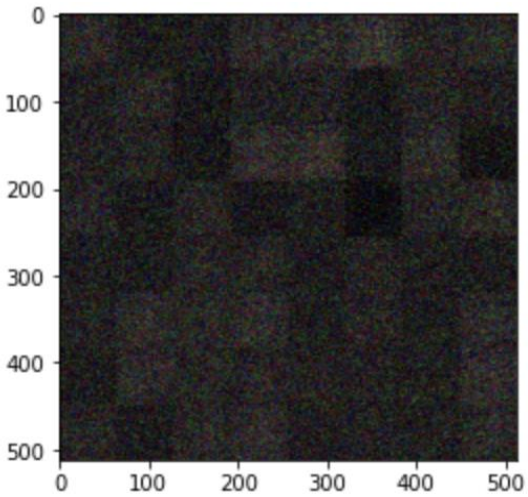

Figure 6. Constructed Images After Each 200th Epoch

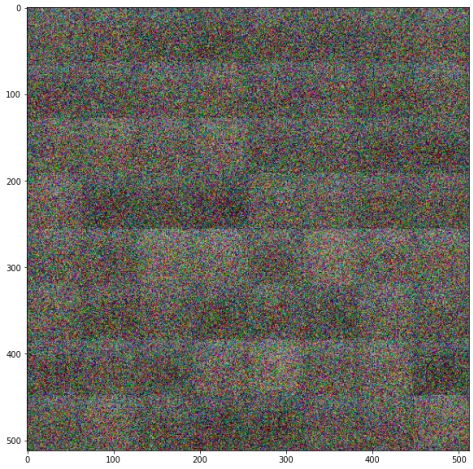

Figure 7. Constructed Images After Each 300th Epoch

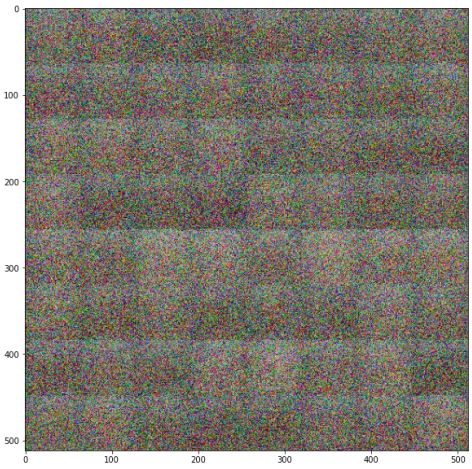

Figure 8. Constructed Images After Each 400th Epoch

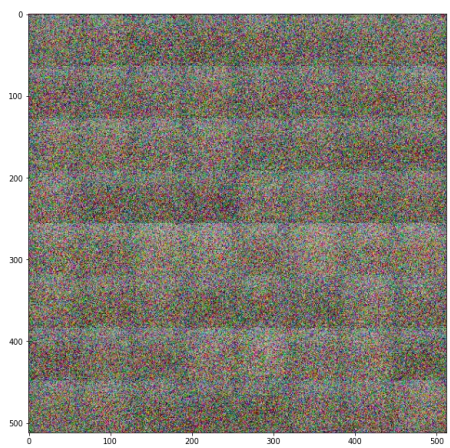

Figure 9. Constructed Images After Each 500th Epoch

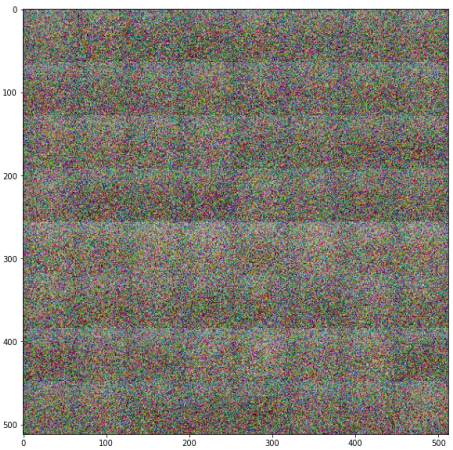

Figure 10. Constructed Images After Each 600th Epoch

Although we are able to add colour to generated images after each epoch, in the current situation we have failed to generate meaningful images that include the features learned from the test captions. Furthermore, some of the losses and elapsed time during epoch can be seen below.

```
Time for epoch 36 is 3.1548 sec
Epoch 37, generative loss: 87.6619, discriminator loss: 62.1214
Time for epoch 37 is 3.2436 sec
Epoch 38, generative loss: 90.0310, discriminator loss: 60.7153
Time for epoch 38 is 3.2397 sec
Epoch 39, generative loss: 92.7607, discriminator loss: 54.9573
```

Figure 11. Generative Losses and Discriminator Losses for Epoch 37-38-39

```
Time for epoch 253 is 2.6046 sec
Epoch 254, generative loss: 96.5058, discriminator loss: 37.8819
Time for epoch 254 is 2.9421 sec
Epoch 255, generative loss: 82.8056, discriminator loss: 44.3865
Time for epoch 255 is 3.0724 sec
Epoch 256, generative loss: 86.0450, discriminator loss: 44.3059
Time for epoch 256 is 3.1953 sec
Epoch 257, generative loss: 93.2783, discriminator loss: 36.5469
Time for epoch 257 is 3.1419 sec
```

Figure 12. Generative Losses and Discriminator Losses for Epoch 254-255-256-257

```
Epoch 597, generative loss: 109.3816, discriminator loss: 42.8354
Time for epoch 597 is 3.2034 sec
Epoch 598, generative loss: 94.2870, discriminator loss: 51.0906
Time for epoch 598 is 3.1462 sec
Epoch 599, generative loss: 125.9918, discriminator loss: 39.9072
```

Figure 13. Generative Losses and Discriminator Losses for Epoch 597-598-599

According to our train results and generated images are not promising although our many different trials to solve the issues. We could not implement any test setup for the rest of the dataset. During the implementation progress our main focus was to make the model work instead of creating a simulation environment for the test set. The possible reasons behind failure will be explained in the discussion section.

4. Discussion

As we know, there are a number of factors which are responsible for the performance of computer vision applications and computing power is one of the important factors. Training NN models on GPU instead of CPU giving better performance is a known fact. Since our computers are Macs, we couldn't import the CUDA library which is designed to train models using GPU instead of CPU, therefore we have decided to use GoogleColab for implementation. In our first trials we have realized that the default GPU accelerator is also not providing enough performance for our project again, thus we moved on with the pro version of the Colab. We hoped that this would solve the performance issues that we faced but again this was not the case. Since we are provided only the URL of the images in COCO dataset, before moving to the implementation first we had to pre-process download the images to drive. As explained in the pre-processing part, the dataset contains various different sizes of images and qualities and some images have problematic parts such as black corrupted areas. One of the samples for those types of images can be seen below.

Figure 10. Sample Image

When we tried to download those images to drive by default settings it took approximately 3 hours for 8000 images. Since the train dataset consists of 82000, saving all dataset took more than 30 hours. Collab limits the time for daily use with 12 hours therefore this is not applicable in our case. To solve this issue and also make all the images same size for feeding to network we have resized images to 64*64. By doing so the saving process becomes faster but only half the time it takes. Therefore, we move on with only 8000 images. Using a subclass of a dataset is a problematic issue for many different machine learning applications. The subclass that we take may not have image features for test data. We believe that not being able to use the whole dataset for training caused problems for the feature and caption embedding. Also, we are thinking that saving images as 64*64 caused another problem. Extracting features from such small images is harder compared to the normal sized images. To overcome this issue, we have tried to implement deeper networks; however, again we abandoned this idea because of the training time concerns. By using 8000 number of 64*64 images for training, 600 epochs took approximately 30 minutes. When we made images 128*128, time doubles and making images even bigger shape make training even harder. If we used a whole training set containing 80000 images, training 600 epochs would be finished in 10 hours. Since we don’t have enough time to train such a model in a Collab we have decided to move on with the current setup. Although we have done lots of background research and read papers on the topic, we weren't able to provide a working model with such a small dataset. We believe that using more powerful hardware with a whole training dataset, higher number of epochs and bigger pre-processed images, the model would perform better and we would at least be able to see some features in generated images that correspond to the relevant captions.

References

[1] Akanksha Singh, Sonam Anekar , Ritika Shenoy , Sainath Pati, T*ext to Image using Deep Learning*, International Journal of Engineering Research & Technology (IJERT), Volume 10, Issue, Apr. 24 2021.

[2] Shan-Hung Wu, *Text To Image*, DataLab Cup 3: Reverse Image Caption. [Online]. Available: https://nthu-datalab.github.io/ml/competitions/Comp_03_Reverse-Image-Caption/03_Reverse-Image-Caption.html

[3] Brownlee Jason, *How to Implement the Inception Score (IS) for Evaluating GANs.* Machine Learning Mastery. Aug.28,2019. [Online]. Available: https://machinelearningmastery.com/how-to-implement-the-inception-score-from-scratch-for-evaluating-generated-images/.

[4] Text to Image Generation on COCO. [Online]. Available: https://paperswithcode.com/sota/text-to-image-generation-on-coco

[5] Joint Representations for Images and Text. [Online]. Available: https://towardsdatascience.com/tag2image-and-image2tag-joint-representations-for-images-and-text-9ad4e5d0d99

[6] Generative Adversarial Text to Image Synthesis. Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, Honglak Lee. [Online]. Available: https://arxiv.org/pdf/1605.05396.pdf

[7] TensorFlow Keras Embedding. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Embedding

[8] Encoder-Decoder Recurrent Neural Network Models for Neural Machine Translation. [Online]. Available: https://machinelearningmastery.com/encoder-decoder-recurrent-neural-network-models-neural-machine-translation/

[9] Encoder-Decoder Models for Natural Language Processing. [Online]. Available: https://www.baeldung.com/cs/nlp-encoder-decoder-models

[10] Deep Convolutional Generative Adversarial Networks (DCGANs). [Online]. Available: https://medium.datadriveninvestor.com/deep-convolutional-generative-adversarial-networks-dcgans-3176238b5a3d#:~:text=Leaky%20ReLU%20prevent%20this%20dying,learning%20process%20won't%20happen.

Appendix

```
# -*- coding: utf-8 -*-
"""Tetx2ImageFinal.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/15i1wMzj6SzKinYI4oCOJ3Zs03nAxSfqz
"""

from typing import List
import string
import numpy as np
import h5py
import urllib3
import PIL
from io import BytesIO, StringIO
from PIL import Image
import matplotlib.pyplot as plt
import scipy.io
import os
import pickle
import random
import time
import pandas as pd
import tensorflow as tf
from keras import Input, Model
from keras import backend as K
from keras.callbacks import TensorBoard
from keras.layers import Dense, LeakyReLU, BatchNormalization, ReLU, Reshape, UpSampling2D,
Conv2D, Activation, Flatten, Lambda, Concatenate
from tensorflow.keras.optimizers import Adam
import keras,os
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPool2D , Flatten
from keras.preprocessing.image import ImageDataGenerator
```

```
from keras.applications.vgg16 import VGG16
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.vgg16 import preprocess_input,decode_predictions
from keras import backend as K

from __future__ import absolute_import, division, print_function, unicode_literals
import tensorflow as tf
from tensorflow.keras import layers
import os


from pathlib import Path

import re
from IPython import display

model = VGG16(weights='imagenet')

"""# Load COCO dataset from web"""

# Import the training and test data from HDF5 files.
with h5py.File('/content/drive/MyDrive/required/eee443_project_dataset_train.h5','r') as trainh5:
  traincaps = list(trainh5['train_cap'])
  trainimgind = list(trainh5['train_imid'])
  trainURLs = list(trainh5['train_url'])

with h5py.File('/content/drive/MyDrive/required/eee443_project_dataset_test.h5','r') as testh5:
  testcaps = list(testh5['test_caps'])
  testimgind = list(testh5['test_imid'])
  testURLs = list(testh5['test_url'])

# The dictionary is imported via MATLAB and converted to numpy.ndarray
word_ind = scipy.io.loadmat('/content/drive/MyDrive/required/word_inds.mat')
word_indexes = (word_ind['word_inds'])

with open('/content/drive/MyDrive/required/words.txt', 'r') as words:
    word_str = words.readlines()
```

```
word_names = []
for x in word_str:
    word_names.append(x.replace("\n", ""))
#word_names.append(" ")
word_names = np.reshape(word_names,(1004,1))

# From list tot np.array
traincaption = np.array(traincaps)
trainimid = np.array(trainimgind)

testcaption = np.array(testcaps)
testimid = np.array(testimgind)

# GloVe word embedding
glove = open("/content/drive/MyDrive/required/glove.6B.50d.txt",encoding='utf8')

embedding_index = {}
for line in glove :
    values = line.split()
    word = values[0]
    word_embedding = np.array(values[1:],dtype = 'float')
    embedding_index[word] = word_embedding

word_names

urllib3.disable_warnings()

def get_image(index):
  """
  Download image from internet by using URL
  """
  y = 0
  url = trainURLs[index].decode('UTF-8')
  http = urllib3.PoolManager()
  r = http.request('GET', url)
  while True:
```

```
    try:
        image = Image.open(BytesIO(r.data))
        if len(np.shape(image)) != 3:
         y = 1
         return y
        else:
         return image
        break
    except PIL.UnidentifiedImageError:
        y = 1
        return y
        break

def get_sentence(index):
  """
  Get sentences for corresponding images
  """
  endsentence = np.where(traincaption[index,:]==2)
  endsentence = endsentence[0]
  endsentence = endsentence[0]
  x = traincaption[index,1:endsentence]
  sentence = []
  for i in x:
    k = np.where(word_indexes == i)
    m = word_names[k]
    m = m[0]
    if m == 'x_UNK_':
      x = np.delete(x,3)
      pass
    else:
      sentence.append(m)
  return sentence,x

def sentence_encoding(sentence):
  """
  Encode sentecnces using GloVe
  """
```

```
  sentence_array = []
  for word in sentence:
    word_array = embedding_index["word"]
    sentence_array = np.concatenate((sentence_array, word_array), axis=None)
  if len(sentence_array)<15*50:
    num_missing = 15*50-len(sentence_array)
    zeros = np.zeros((num_missing,),dtype=int)
    sentence_array = np.concatenate((sentence_array, zeros), axis=None)
  return sentence_array

def get_encoded_image(index):
  """
  Use VGG-16 pretrained model for image encoding
  """
  img = get_image(index)
  img = img.resize((224,224))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)
  x = preprocess_input(x)
  get_feature = K.function([model.layers[0].input],
                           [model.layers[-2].output])
  layer_output = get_feature([x])[0][0]
  return np.array(layer_output)

def save_train_images(index):
  """
  Save images to the drive and retun saved images path and corresponding sentences
  """
  sentences = np.where(trainimid == index)[0]
  img = get_image(index-1)
  if img == 1:
    sentences = "pass"
    path = "pass"
    pass
  else:
    img = img.resize((64,64), Image.ANTIALIAS)
    try:
```

```
      img.save('/content/drive/MyDrive/real_train/'+ str(index) +'.png')
      path = "/content/drive/MyDrive/real_train/"+ str(index) +'.png'
    except OSError:
      img.convert('RGB').save('/content/drive/MyDrive/real_train/'+ str(index) +'.png', "PNG",
optimize=True)
  return sentences,path

def word_to_id (sentences):
  """
  Turns words in sentences into corresponding IDs in the word dicitonary
  """
  sentences_word_id = []
  for sentence in sentences:
    sent,word_id = get_sentence(sentence)
    while len(word_id)<15:
      word_id = np.append(word_id,1004)
    sentences_word_id.append(word_id)
  return np.array(sentences_word_id)

# Run this cell once: Image gathering from dataset
df = pd.DataFrame(columns = ['ID','Captions','ImagePath'])
sents = []
imgids = []
paths = []
for i in range(1,8000):
  sentences,path = save_train_images(i)
  if sentences == "pass":
    pass
  else:
    word_id = word_to_id (sentences)
    sents.append(word_id)
    imgids.append(i)
    paths.append(path)
    if i%100 == 0 :
      print(str(i)+ "th image is saved")
#create dataframe that stores converted sentences and corresponding image's path
df = pd.DataFrame({'ID': np.array(imgids), 'Captions': sents, 'ImagePath': paths}).set_index('ID')
```

```
#save df as pickle file for futher use
df.to_pickle("/content/drive/MyDrive/data_frame.pkl")

"""# Create Dataset by TensorFlow API"""

#read df from pickle
df = pd.read_pickle('/content/drive/MyDrive/data_frame.pkl')
df

def image_to_tensor(caption, image_path):
  """
  Load the image and caption according to image path into tensor using its own functions
  """
  image = tf.io.read_file(image_path)
  image = tf.image.decode_image(image, channels=3)
  image = tf.image.convert_image_dtype(image, tf.float32)
  image.set_shape([None, None, 3])
  image = tf.image.resize(image, size=[64, 64])
  image.set_shape([64, 64, 3])
  caption = tf.cast(caption, tf.int32)
  return image, caption

def train_dataset_generator(df, batch_size, image_to_tensor):
  """
  Using the image_to_tensor method crate appropiate dataset as input to network
  """
  captions = df['Captions'].values
  image_path = df['ImagePath'].values
  caption = []
  for i in range(len(captions)):
      caption.append(random.choice(captions[i]))
  caption = np.asarray(caption)
  caption = caption.astype(np.int)
  train_dataset = tf.data.Dataset.from_tensor_slices((caption, image_path))
  train_dataset = train_dataset.map(image_to_tensor,
num_parallel_calls=tf.data.experimental.AUTOTUNE)
  train_dataset = train_dataset.shuffle(len(caption)).batch(batch_size, drop_remainder=True)
```

```
  train_dataset = train_dataset.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)

  return train_dataset

#Since batch size chosen by us is 64 function called accordingly
train_dataset = train_dataset_generator(df, 64, image_to_tensor)

"""# Conditional GAN Model

## Text Encoder
"""

class TextEncoder(tf.keras.Model):
  """
  Encode text (a caption) into hidden representation
  input: text, which is a list of ids
  output: embedding, or hidden representation of input text in dimension of RNN_HIDDEN_SIZE
  """
  def __init__(self, hparas):
      super(TextEncoder, self).__init__()
      self.hparas = hparas
      self.batch_size = self.hparas['BATCH_SIZE']
      self.embedding = layers.Embedding(self.hparas['VOCAB_SIZE'], self.hparas['EMBED_DIM'])
      self.gru = layers.GRU(self.hparas['RNN_HIDDEN_SIZE'],return_sequences=True,return_state=
True,recurrent_initializer='glorot_uniform')

  def call(self, text, hidden):
      text = self.embedding(text)
      output, state = self.gru(text, initial_state = hidden)
      return output[:, -1, :], state

  def initialize_hidden_state(self):
      return tf.zeros((self.hparas['BATCH_SIZE'], self.hparas['RNN_HIDDEN_SIZE']))

"""## Generator Class"""

class Generator(tf.keras.Model):
```

```
  """
  Generate fake image based on given text(hidden representation) and noise z
  input: text and noise
  output: fake image with size 64*64*3
  """
  def __init__(self, hparas):
      super(Generator, self).__init__()
      self.hparas = hparas
      self.flatten = tf.keras.layers.Flatten()
      self.d1 = tf.keras.layers.Dense(self.hparas['DENSE_DIM'])
      self.d2 = tf.keras.layers.Dense(64*64*3)

  def call(self, text, noise_z):
      text = self.flatten(text)
      text = self.d1(text)
      text = tf.nn.leaky_relu(text)
      concatinated_text = tf.concat([noise_z, text], axis=1)
      concatinated_text = self.d2(concatinated_text)
      logits = tf.reshape(concatinated_text, [-1, 64, 64, 3])
      output = tf.nn.tanh(logits)

      return logits, output

"""## Discriminator"""

class Discriminator(tf.keras.Model):
  """
  Differentiate the real and fake image
  input: image and corresponding text
  output: labels, the real image should be 1, while the fake should be 0
  """
  def __init__(self, hparas):
      super(Discriminator, self).__init__()
      self.hparas = hparas
      self.flatten = tf.keras.layers.Flatten()
      self.d_text = tf.keras.layers.Dense(self.hparas['DENSE_DIM'])
      self.d_img = tf.keras.layers.Dense(self.hparas['DENSE_DIM'])
```

```
        self.d = tf.keras.layers.Dense(1)

    def call(self, img, text):
        text = self.flatten(text)
        text = self.d_text(text)
        text = tf.nn.leaky_relu(text)
        image = self.flatten(img)
        image = self.d_img(image)
        image = tf.nn.leaky_relu(image)
        img_text = tf.concat([text, image], axis=1)
        logits = self.d(img_text)
        output = tf.nn.sigmoid(logits)

        return logits, output

hparas = {
            'MAX_SEQ_LENGTH': 15,
            'EMBED_DIM': 256,
            'VOCAB_SIZE': 1004,
            'RNN_HIDDEN_SIZE': 128,
            'Z_DIM': 512,
            'DENSE_DIM': 128,
            'IMAGE_SIZE': [64, 64, 3],
            'BATCH_SIZE': 64,
            'LR': 1e-4,
            'LR_DECAY': 0.5,
            'BETA_1': 0.5,
            'N_EPOCH': 600,
            'N_SAMPLE': len(df),
            'PRINT_FREQ': 100,
            }

text_encoder = TextEncoder(hparas)
generator = Generator(hparas)
discriminator = Discriminator(hparas)

"""## Optimizer and Loss Function"""
```

```
def loss(real_logits, fake_logits):
  """
  Function returns discriminator and generator loss using cross entropy loss
  """
  cross_entropy_loss = tf.keras.losses.BinaryCrossentropy(from_logits=True)
  loss_real_image = cross_entropy_loss(tf.ones_like(real_logits), real_logits)
  loss_fake_image = cross_entropy_loss(tf.zeros_like(fake_logits), fake_logits)
  discriminator_loss = loss_real_image + loss_fake_image
  generator_loss = cross_entropy_loss(tf.ones_like(fake_logits), fake_logits)

  return discriminator_loss,generator_loss

# Using adam for optimizer
generator_optimizer = tf.keras.optimizers.Adam(hparas['LR'])
discriminator_optimizer = tf.keras.optimizers.Adam(hparas['LR'])

"""# Train and Test Steps"""

@tf.function
def train_step(real_image, caption, hidden):
    # random noise for generator
    noise = tf.random.normal(shape=[hparas['BATCH_SIZE'], hparas['Z_DIM']], mean=0.0, stddev=1.
0)

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        text_embed, hidden = text_encoder(caption, hidden)
        _, fake_image = generator(text_embed, noise)
        real_logits, real_output = discriminator(real_image, text_embed)
        fake_logits, fake_output = discriminator(fake_image, text_embed)

        d_loss,g_loss = loss(real_logits, fake_logits)

    grad_g = gen_tape.gradient(g_loss, generator.trainable_variables)
    grad_d = disc_tape.gradient(d_loss, discriminator.trainable_variables)

    generator_optimizer.apply_gradients(zip(grad_g, generator.trainable_variables))
```

```
    discriminator_optimizer.apply_gradients(zip(grad_d, discriminator.trainable_variables))

    return g_loss, d_loss

@tf.function
def test_step(caption, noise, hidden):
    text_embed, hidden = text_encoder(caption, hidden)
    _, fake_image = generator(text_embed, noise)
    return fake_image


checkpoint_prefix = "/content/drive/MyDrive/demo/checkpoints/ckpt"
checkpoint = tf.train.Checkpoint(generator_optimizer=generator_optimizer,
                                 discriminator_optimizer=discriminator_optimizer,
                                 text_encoder=text_encoder,
                                 generator=generator,
                                 discriminator=discriminator)

ni = 8
sample_size = 64
sample_seed = np.random.normal(loc=0.0, scale=1.0, size=(64, 512)).astype(np.float32)
sample_list = [[4, 341, 54, 592, 99, 506, 10, 287, 1005, 1005, 1005, 1005, 1005, 1005, 1005],
               [4, 12, 372, 4, 62, 135, 8, 3, 1005, 1005, 1005, 1005, 1005, 1005, 1005],
               [4, 168, 48, 26, 4, 12, 11, 17, 8, 1005, 1005, 1005, 1005, 1005, 1005],
               [4, 155, 10, 23, 20, 576, 8, 7, 62, 1005, 1005, 1005, 1005, 1005, 1005],
               [7, 683, 91, 5, 7, 137, 11, 38, 4, 99, 24, 1005, 1005, 1005, 1005],
               [94, 19, 20, 5, 7, 195, 124, 4, 64, 3, 50, 1005, 1005, 1005, 1005],
               [37, 22, 62, 9, 4, 226, 10, 764, 66, 8, 31, 1005, 1005, 1005, 1005],
               [16, 19, 8, 61, 125, 107, 72, 18, 15, 3, 1005, 1005, 1005, 1005, 1005]]

def sample_generator(caption, batch_size):
    caption = np.asarray(caption)
    caption = caption.astype(np.int)
    dataset = tf.data.Dataset.from_tensor_slices(caption)
    dataset = dataset.batch(batch_size)
    return dataset
```

```
def create_sample(sample_list):
  samples = []
  for sentence in sample_list:
    sentences = []
    for word in sentence:
      sentences.append(str(word))
    for i in range(8):
      samples.append(sentences)
  return samples

def merge(images, size):
    h, w = images.shape[1], images.shape[2]
    img = np.zeros((h * size[0], w * size[1], 3))
    for idx, image in enumerate(images):
        i = idx % size[1]
        j = idx // size[1]
        img[j*h:j*h+h, i*w:i*w+w, :] = image
    return img

def imsave(images, size, path):
    return plt.imsave(path, merge(images, size)*0.5 + 0.5)

def save_images(images, size, image_path):
    return imsave(images, size, image_path)

def train(dataset, epochs):
     samples = create_sample(sample_list)
     sample = sample_generator(samples, sample_size)
     hidden = text_encoder.initialize_hidden_state()
     steps_per_epoch = int(hparas['N_SAMPLE']/hparas['BATCH_SIZE'])
     for epoch in range(hparas['N_EPOCH']):
         g_total_loss = 0
         d_total_loss = 0
         start = time.time()
         for image, caption in dataset:
             g_loss, d_loss = train_step(image, caption, hidden)
             g_total_loss += g_loss
```

```
            d_total_loss += d_loss
        time_tuple = time.localtime()
        time_string = time.strftime("%m/%d/%Y, %H:%M:%S", time_tuple)
        print("Epoch {}, generative loss: {:.4f}, discriminator loss: {:.4f}".format(epoch+1,g_total_loss
/steps_per_epoch,d_total_loss/steps_per_epoch))
        print('Time for epoch {} is {:.4f} sec'.format(epoch+1, time.time()-start))
        # save the model at each 50 epoch
        if (epoch + 1) % 50 == 0:
            checkpoint.save(file_prefix = checkpoint_prefix)
        # show generated images from sample sentence
        if (epoch + 1) % hparas['PRINT_FREQ'] == 0:
          for caption in sample:
            fake_image = test_step(caption, sample_seed, hidden)
          im = merge(fake_image,[8,8])
          plt.figure(figsize = (10,10))
          plt.imshow(im)
          plt.show()
          save_images(fake_image, [8, 8], '/content/drive/MyDrive/demo/train_{:02d}.jpg'.format(epoch
)


train(train_dataset, hparas['N_EPOCH'])
```