IMPLEMENTASI ARSITEKTUR MENGGUNAKAN METODE RECURRENT NEURAL NETWORK (IMAGE CAPTIONING)

Disusun oleh:

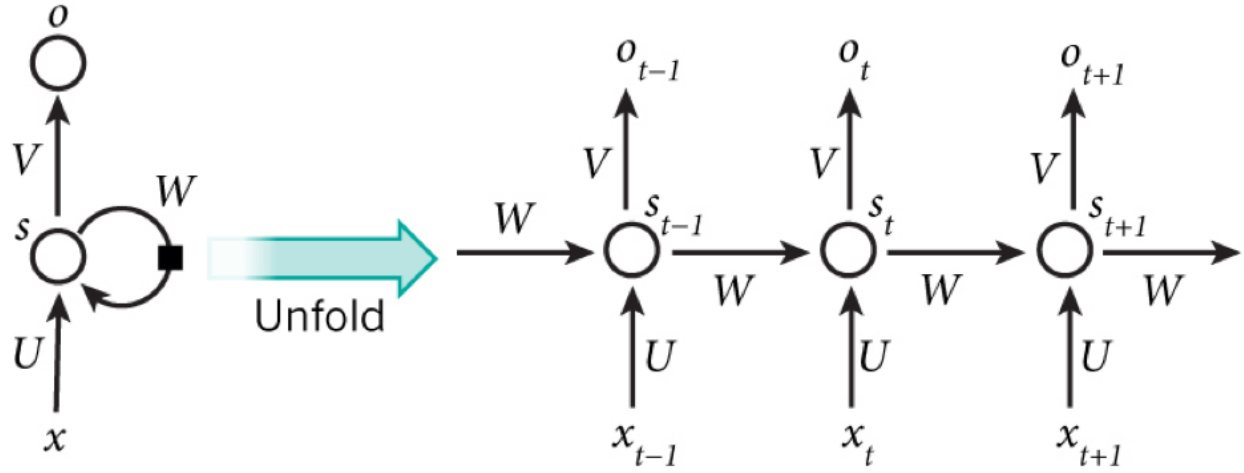
Dheaz Kelvin Harahap 1620122330
Daffa Pratama Yudhistira 162012233065

TEKNIK ROBOTIKA DAN KECERDASAN BUATAN
FAKULTAS TEKNOLOGI MAJU DAN MULTIDISIPLIN

2023

Recurrent Neural Network

Ide di balik arsitektur RNN ini adalah memanfaatkan struktur data berurutan. RNN merupakan singkatan dari "*Recurrent Neural Network*" yang menggambarkan fakta bahwa ia beroperasi secara berulang. Dalam RNN, operasi yang sama diulang untuk setiap elemen dalam urutan data, dengan outputnya bergantung pada input saat ini dan operasi sebelumnya. Intinya, RNN berfokus pada sifat data di mana informasi pada waktu sebelumnya (t) atau saat ini mempengaruhi informasi pada waktu berikutnya (t + 1).



Gambar: Proses RNN saat perhitungan waktu di depannya. Sumber: Denny Britz, 2015

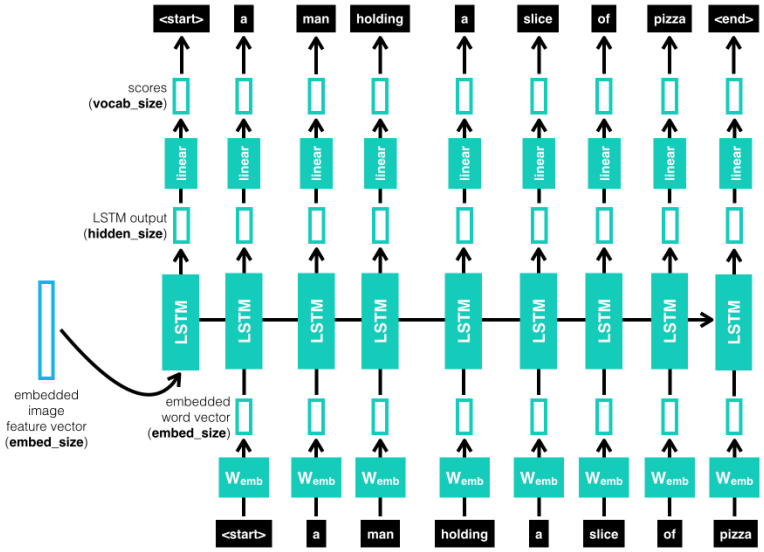
Pada gambar di atas, terdapat dua gambar yang menggambarkan arsitektur RNN. Gambar sebelah kiri adalah sebuah diagram sirkuit yang menunjukkan RNN dalam bentuk yang tidak dibuka (*unrolled*) ke jaringan penuh (*full network*). Dalam diagram tersebut, kotak hitam melambangkan penundaan waktu (*time delay*) dari satu langkah waktu (*time step*) ke langkah waktu berikutnya. Diagram tersebut memberikan gambaran bahwa RNN beroperasi dengan mengalirkan informasi dari satu langkah waktu ke langkah waktu selanjutnya. Sementara itu, gambar sebelah kanan menunjukkan RNN yang telah dibuka (*unrolled/unfolded*) menjadi jaringan penuh (*full network*), sehingga urutan sekuensinya menjadi lengkap. Misalnya, jika kita memiliki sebuah urutan sekuensial dalam bentuk satu kalimat dengan lima kata, maka jaringan RNN akan dibuka menjadi jaringan saraf dengan lima lapisan (*layer*), satu lapisan untuk setiap kata dalam urutan tersebut. Dengan demikian, gambar sebelah kanan mengilustrasikan bagaimana RNN dapat memproses urutan sekuensial secara bertahap dengan melibatkan lapisan-lapisan (*layers*) yang mewakili setiap elemen dalam urutan tersebut. Jadi, secara umum RNN adalah salah satu bagian dari keluarga *Neural Network* untuk memproses data yang bersambung (*sequential data*). Cara yang dilakukan RNN untuk dapat menyimpan informasi dari masa lalu adalah dengan melakukan *looping* di dalam arsitekturnya, yang secara otomatis membuat informasi dari masa lalu tetap tersimpan.

Image Captioning

Image Captioning bertujuan untuk menjelaskan isi konten dari sebuah gambar dengan menggunakan RNN (*Recurrent Neural Networks*) untuk membangun *Generator* Keterangan Gambar. Model ini akan didasarkan pada makalah *Show and Tell: A Neural Image Caption Generator* oleh Oriol Vinyals, dkk. dan akan diimplementasikan menggunakan Tensorflow dan Keras. *Dataset* yang digunakan adalah Flickr 8K, yang terdiri dari 8.000 gambar, masing-masing dipasangkan dengan lima keterangan yang berbeda untuk memberikan deskripsi yang jelas.

Arsitektur

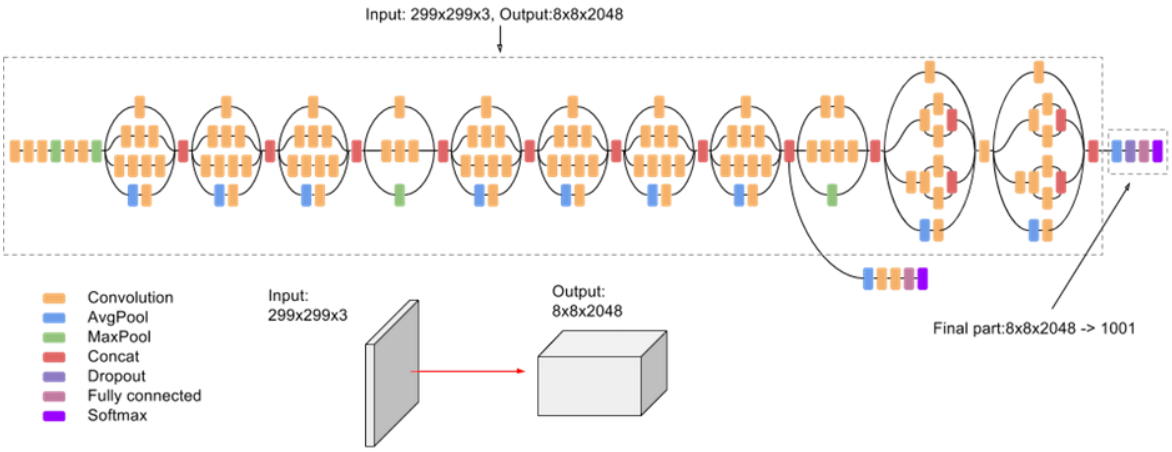
Arsitektur model ini didasari RNN (*Recurrent Neural Network*) berdasarkan lapisan *Long Short Term Memory* (LSTM). Perbedaan paling signifikan dengan model lainnya adalah bahwa *embedding* gambar diberikan sebagai input pertama ke jaringan RNN dan hanya sekali.



Gambar: Arsitektur *Image Captioning* (Sumber: Irjet, 2021)

Encoder Model

Untuk mengekstrak fitur dari gambar, digunakan Inception V3. Pada gambar di bawah ini, terdapat representasi arsitektur jaringan yang digunakan.



Gambar: *Encoder Model Inception V3.* (Sumber: paperswithcode.com)

Implementasi

Recurrent Neural Network (RNN) digunakan dalam Image Captioning untuk memodelkan konteks gambar dan kalimat dengan mempertimbangkan informasi dari waktu sebelumnya. Berikut adalah beberapa implementasi RNN yang umum digunakan dalam Image Captioning:

1. Encoder-Decoder RNN: Model ini terdiri dari dua bagian utama, yaitu Encoder dan Decoder. Encoder mengambil gambar sebagai input dan menghasilkan representasi fitur gambar. Decoder kemudian mengambil representasi fitur tersebut dan menghasilkan caption.
2. LSTM-based Encoder-Decoder: Model ini menggunakan Long Short-Term Memory (LSTM) sebagai unit dasar dari RNN. LSTM memungkinkan model untuk mengingat informasi penting dari input sebelumnya dan mengabaikan informasi yang tidak relevan.
3. Attention-based Encoder-Decoder: Model ini memperkenalkan mekanisme attention untuk memfokuskan pada bagian-bagian gambar yang paling relevan saat menghasilkan caption. Dalam model ini, encoder dan decoder dipasangkan dengan lapisan attention yang memilih representasi fitur gambar yang paling penting untuk menghasilkan kata-kata selanjutnya.
4. DenseCap: Model ini menggunakan Convolutional Neural Network (CNN) untuk menghasilkan representasi fitur gambar dan memperkenalkan lapisan Dense Captioning (DenseCap) yang memprediksi beberapa kata pada saat yang sama, sehingga menghasilkan caption yang lebih akurat dan padat.
5. Show and Tell: Model ini memperkenalkan mekanisme attention seperti pada Attention-based Encoder-Decoder dan menggunakan RNN untuk menghasilkan caption. Namun, model ini tidak memerlukan tahap pemrosesan pemisahan objek seperti pada DenseCap. Sebaliknya, model ini memperhitungkan semua bagian gambar secara bersamaan dan memfokuskan pada area yang paling penting menggunakan mekanisme attention.

Tahapan code yang biasanya digunakan dalam implementasi RNN untuk Image Captioning disebut sebagai pipeline. Pipeline mengacu pada urutan langkah-langkah yang dilakukan untuk memproses gambar dan menghasilkan caption dengan menggunakan model RNN. Tahapan dalam pipeline umumnya meliputi:

1. Pra-pemrosesan gambar, seperti resizing, cropping, dan normalisasi intensitas piksel.
2. Ekstraksi fitur gambar menggunakan arsitektur CNN.
3. Pra-pemrosesan teks, seperti tokenisasi dan pembentukan kamus kata.
4. Pembentukan dataset pelatihan, termasuk pengelompokan gambar dengan caption yang sesuai dan membagi dataset menjadi subset pelatihan, validasi, dan pengujian.
5. Pelatihan model RNN dengan dataset pelatihan yang telah dibentuk.
6. Evaluasi model menggunakan dataset validasi atau pengujian, dan tuning parameter model.
7. Pengujian model pada dataset yang belum pernah dilihat sebelumnya untuk mengevaluasi performa generalisasi model.

Setelah tahapan pipeline selesai dilakukan, model dapat digunakan untuk menghasilkan caption untuk gambar yang belum pernah dilihat sebelumnya. Tahapan ini melibatkan pengolahan gambar yang sama seperti pada tahap pra-pemrosesan, ekstraksi fitur gambar, dan pengolahan teks untuk menghasilkan caption.

Hasil

Impor Library

```
import re
import random

import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

from tensorflow import keras
from time import time

from tqdm import tqdm # progress bar
from sklearn.model_selection import train_test_split # Dividing train test
from nltk.translate.bleu_score import corpus_bleu # BLEU Score
```

Code di atas menggunakan beberapa modul untuk melakukan tugas-tugas tertentu terkait dengan pemrosesan bahasa alami (NLP) dan deep learning menggunakan TensorFlow. Berikut adalah penjelasan singkat untuk setiap bagian kode tersebut:

- import re: modul untuk melakukan operasi operasi regular expression atau ekspresi reguler
- import random: modul untuk menghasilkan bilangan acak
- import numpy as np: modul untuk melakukan operasi matematika, terutama pada array
- import tensorflow as tf: modul utama untuk melakukan deep learning pada TensorFlow
- import matplotlib.pyplot as plt: modul untuk membuat grafik atau plot
- from tensorflow import keras: modul dari TensorFlow untuk membangun model deep learning
- from time import time: modul untuk menghitung waktu eksekusi program
- from tqdm import tqdm: modul untuk membuat progress bar saat program berjalan
- from sklearn.model_selection import train_test_split: modul untuk membagi dataset menjadi subset pelatihan dan pengujian
- from nltk.translate.bleu_score import corpus_bleu: modul untuk menghitung skor BLEU (bilingual evaluation understudy) atau evaluasi dwibahasa pada data yang diterjemahkan

Data Set

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
Mounted at /content/drive
```

Kode di atas menghubungkan Google Colab dengan Google Drive pengguna. Google Colab adalah lingkungan pengembangan terpadu (IDE) berbasis cloud yang disediakan oleh Google, sedangkan Google Drive adalah layanan penyimpanan awan dari Google.

Pertama, kode mengimpor modul "drive" dari "google.colab", yang merupakan modul untuk menghubungkan ke Google Drive. Kemudian, fungsi "drive.mount()" dipanggil untuk meminta otorisasi pengguna untuk mengakses Google Drive.

```
!pip install kaggle
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheel
Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/di
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-pack
```

menginstal paket atau modul "kaggle" menggunakan pip, yaitu package installer untuk Python. "kaggle" adalah sebuah platform kompetisi data online yang menyediakan dataset publik untuk latihan dan penelitian di bidang data science dan machine learning.

```
#Make a directory named kaggle and copy the kaggle.json file there.
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
# change the permission of the file
!chmod 600 ~/.kaggle/kaggle.json
```

memindahkan file kaggle.json ke direktori ~/.kaggle/ dan mengubah izin akses file tersebut agar hanya pengguna yang memiliki izin akses tertentu yang dapat mengaksesnya.

```
!kaggle datasets download -d adityajn105/flickr8k
```

```
Downloading flickr8k.zip to /content
100% 1.04G/1.04G [00:49<00:00, 23.4MB/s]
100% 1.04G/1.04G [00:49<00:00, 22.3MB/s]
```

memerintah sel yang digunakan untuk mengunduh dataset Flickr8k dari Kaggle dengan menggunakan perintah "kaggle datasets download" dan menyertakan argumen "-d" yang menentukan ID dataset.

```
from zipfile import ZipFile
file_name = 'flickr8k.zip' #the file is your dataset exact name
with ZipFile(file_name, 'r') as zip:
  zip.extractall()
  print('Done')
```

```
Done
```

mengekstrak isi dari file zip yang telah diunduh sebelumnya menggunakan perintah **!kaggle datasets download**. Dalam hal ini, file zip yang diekstrak adalah **flickr8k.zip**.

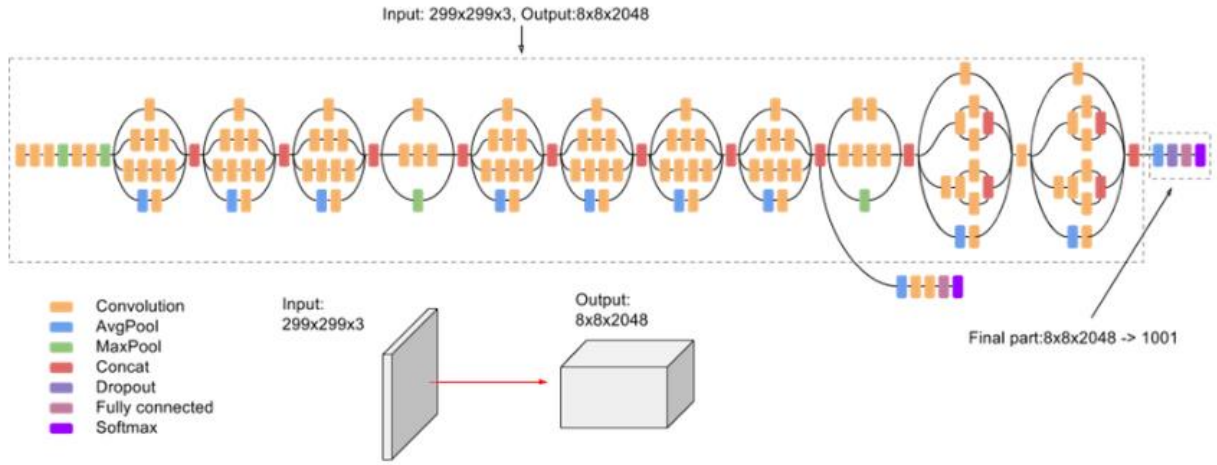
Image Configuration

```
img_height = 180
img_width = 180
validation_split = 0.2
```

mendefinisikan variabel yang digunakan dalam mempersiapkan data untuk pelatihan model. **img_height** dan **img_width** adalah dimensi gambar yang digunakan dalam model. Dalam hal ini, gambar akan diubah menjadi ukuran 180x180 piksel.

Encoder Model

Untuk mengekstraksi fitur dari gambar, model CNN pra-pelatihan, bernama Inception V3 digunakan. Pada gambar dibawah ini merupakan representasi dari arsitektur jaringan yang digunakan.




```
# Remove the last layer of the Inception V3 model
def get_encoder():
    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_layer)
    return image_features_extract_model
```

mendefinisikan fungsi **get_encoder()** yang digunakan untuk mengambil model Inception V3 yang telah dilatih Pada dataset ImageNet sebagai ekstraktor fitur gambar untuk tugas image captioning. pada baris pertama, model Inception V3 dimuat dengan menentukan argumen **include_top=False** yang mengindikasikan bahwa lapisan terakhir model, yaitu lapisan klasifikasi, akan dihilangkan.

Kemudian, pada baris kedua, input model disimpan dalam variabel **new_input**. Pada baris ketiga, lapisan terakhir model disimpan dalam variabel **hidden_layer**. pada baris keempat, model baru yang terdiri dari **new_input** dan **hidden_layer** dijadikan sebagai output dengan menggunakan **tf.keras.Model()**. Model ini kemudian digunakan untuk mengekstrak fitur dari setiap gambar masukan.

Membaca *Caption*

```
# Preprocess the caption, splitting the string and adding <start> and
<end> tokens
def get_preprocessed_caption(caption):
    caption = re.sub(r'\s+', ' ', caption)
    caption = caption.strip()
    caption = "<start> " + caption + " <end>"
    return caption
```

mendefinisikan fungsi **get_preprocessed_caption()** yang digunakan untuk memproses teks deskripsi dari setiap gambar pada dataset. Tujuannya adalah untuk memisahkan kata-kata pada deskripsi menjadi token-token yang terpisah dan menambahkan token **<start>** dan **<end>** pada awal dan akhir deskripsi.

```
images_captions_dict = {}

with open('/content/drive/MyDrive/tess' + "/captions.txt", "r") as
dataset_info:
    next(dataset_info) # Omit header: image, caption

    # Using a subset of 4,000 entries out of 40,000
    for info_raw in list(dataset_info)[:4000]:
        info = info_raw.split(",")
        image_filename = info[0]
        caption = get_preprocessed_caption(info[1])

        if image_filename not in images_captions_dict.keys():
            images_captions_dict[image_filename] = [caption]
        else:
            images_captions_dict[image_filename].append(caption)
```

membaca file **captions.txt** yang berisi informasi deskripsi dari setiap gambar pada dataset dan memprosesnya menjadi suatu dictionary yang memetakan setiap nama file gambar dengan deskripsi yang terkait.

Pada baris pertama, dictionary **images_captions_dict** yang akan diisi dengan informasi deskripsi dari setiap gambar pada dataset diinisialisasi sebagai dictionary kosong.

Pada baris ke-3, file **captions.txt** dibuka dengan mode read (**"r"**) dan pointer file ditempatkan pada baris kedua (baris pertama adalah header).

Pada baris ke-6, list sepanjang 4.000 entri (dari 40.000 entri total) dibuat dari file **captions.txt**. List ini dibuat untuk mengambil hanya subset dari seluruh dataset untuk keperluan pengujian yang lebih cepat.

Pada baris ke-7, setiap entri pada list **dataset_info** diproses. Entri ini berisi nama file gambar dan deskripsi terkait, yang dipisahkan dengan tanda koma.

Pada baris ke-8, nama file gambar dipisahkan dari deskripsi dengan menggunakan **info[0]**, sedangkan deskripsi diproses dan dipisahkan menjadi token-token dengan menggunakan fungsi **get_preprocessed_caption()** pada **info[1]**.

Pada baris ke-10, informasi deskripsi pada suatu gambar dimasukkan ke dalam dictionary **images_captions_dict**. Jika suatu nama file gambar belum ada di dictionary, suatu key baru dengan value berupa list yang berisi deskripsi pertama untuk gambar tersebut dibuat. Jika key dengan nama file gambar sudah ada di dictionary, deskripsi baru akan ditambahkan ke dalam list value yang terkait.

Setelah loop selesai dieksekusi, dictionary **images_captions_dict** akan berisi informasi deskripsi untuk setiap gambar pada dataset yang ada pada subset data sebanyak 4.000 entri.

Membaca Gambar

- **Membuat kamus dengan nama file gambar sebagai kunci dan fitur gambar diekstraksi menggunakan model yang telah dilatih sebelumnya sebagai nilainya.**

```
def load_image(image_path):
    img = tf.io.read_file('/content/Images/' +
'1022975728_75515238d8.jpg')
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.image.resize(img, (img_height, img_width))
    img = tf.keras.applications.inception_v3.preprocess_input(img) #
preprocessing needed for pre-trained model
    return img, image_path
```

mendefinisikan sebuah fungsi **load_image()** yang menerima path ke sebuah file gambar dan akan memuat gambar tersebut. Pada saat pemanggilan fungsi, argumen **image_path** akan dimasukkan dan membaca file gambar dengan menggunakan **tf.io.read_file()**, kemudian mengubahnya ke dalam tensor dengan menggunakan **tf.image.decode_jpeg()**. Selanjutnya, gambar akan diubah ukurannya menggunakan **tf.image.resize()** menjadi ukuran **img_height** dan **img_width** yang telah didefinisikan sebelumnya. Terakhir, gambar akan diproses menggunakan **tf.keras.applications.inception_v3.preprocess_input()** yang diperlukan untuk model pre-trained. Fungsi ini akan mengembalikan tensor gambar dan path gambar.

```
image_captions_dict_keys = list(images_captions_dict.keys())
image_dataset =
tf.data.Dataset.from_tensor_slices(image_captions_dict_keys)
```

```
image_dataset = image_dataset.map(load_image,
num_parallel_calls=tf.data.experimental.AUTOTUNE).batch(64)
```

memuat dataset gambar dengan mengambil daftar kunci (keys) dari kamus yang berisi daftar nama file gambar dan caption terkait. Setelah itu, menggunakan fungsi **load_image** untuk memuat gambar, melakukan preprocessing pada gambar tersebut, dan mengembalikan gambar beserta path-nya. Kemudian dataset gambar tersebut di-map menggunakan fungsi **load_image** dan batch size sebesar 64. Penggunaan **tf.data.experimental.AUTOTUNE** memungkinkan TensorFlow untuk menentukan jumlah thread secara otomatis untuk memuat data tanpa harus menentukan secara manual.

```
images_dict = {}
encoder = get_encoder()
for img_tensor, path_tensor in tqdm(image_dataset):
    batch_features_tensor = encoder(img_tensor)

    # Loop over batch_features_tensor and enumerate the index and value
    for idx, batch_features in enumerate(batch_features_tensor):
        path = path_tensor[idx]
        decoded_path = path.numpy().decode("utf-8")
        images_dict[decoded_path] = batch_features.numpy()
```

```
100%|██████████| 13/13 [00:05<00:00,  2.54it/s]
```

melakukan ekstraksi fitur dari setiap gambar dalam dataset menggunakan model **InceptionV3** yang telah dihapus layer terakhirnya. Kemudian setiap fitur yang diekstraksi disimpan ke dalam suatu dictionary yang menggunakan nama file gambar sebagai key dan nilai fitur sebagai valuenya.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount,
call drive.mount("/content/drive", force_remount=True).
```

menghubungkan Colab dengan Google Drive dan meminta akses data di Google Drive. Setelah dijalankan, pengguna akan diminta untuk mengikuti beberapa langkah autentikasi untuk memperoleh kode otorisasi yang dapat dimasukkan ke dalam Colab. Setelah otorisasi berhasil, pengguna dapat mengakses dan membaca data di Google Drive dari Colab.

- **Ukuran gambar setelah mengekstraksi fitur dari model terlatih**

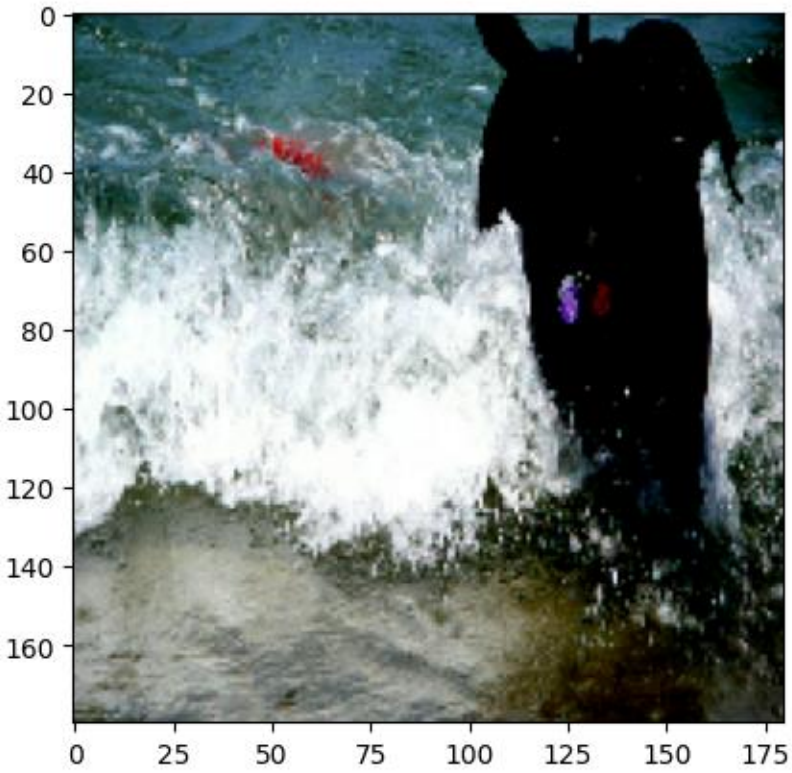
```
list(images_dict.items())[0][1].shape
```

```
(4, 4, 2048)
```

mengambil item pertama dari dictionary **images_dict** menggunakan **list(images_dict.items())[0]**. Kemudian, **[1]** digunakan untuk mengambil nilai atau value dari item tersebut yang berupa array. Terakhir, **shape** digunakan untuk mengembalikan tuple yang menunjukkan dimensi array tersebut. Jadi, code tersebut akan mengembalikan bentuk (shape) array pertama dalam **images_dict**.

```
plt.imshow(load_image('/content/Images/'                                    +
'1022975728_75515238d8.jpg')[0].numpy())
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
with RGB data ([0..1] for floats or [0..255] for integers).
<matplotlib.image.AxesImage at 0x7fcd78340070>
```



fungsi **load_image** untuk membaca dan menampilkan gambar dengan menggunakan library **matplotlib**. Gambar yang akan ditampilkan adalah gambar dengan path **'/content/Images/1022975728_75515238d8.jpg'**. Fungsi **load_image** akan mengembalikan image tensor dari gambar tersebut. Kemudian, **numpy()** digunakan untuk mengubah image tensor menjadi numpy array sehingga dapat ditampilkan menggunakan **plt.imshow()**.

Mengambil gambar dan label dari *filenames*

```
def get_images_labels(image_filenames):
    images = []
    labels = []

    for image_filename in image_filenames:
        image = images_dict[image_filename]
        captions = images_captions_dict[image_filename]

        # Add one instance per caption
        for caption in captions:
            images.append(image)
            labels.append(caption)

    return images, labels
```

Fungsi **get_images_labels** mengambil daftar nama file gambar dan mengembalikan daftar gambar dan label yang sesuai.

Pada bagian **for image_filename in image_filenames:** fungsi tersebut melakukan iterasi untuk setiap nama file gambar yang diberikan, kemudian mengambil nilai gambar dari kamus **images_dict** berdasarkan nama file gambar yang diberikan, dan mengambil caption terkait dari kamus **images_captions_dict**.

Kemudian, untuk setiap caption yang terkait dengan gambar, fungsi menambahkan gambar yang sesuai dan caption tersebut ke daftar **images** dan **labels**. Dengan demikian, fungsi ini akan mengembalikan dua daftar, yaitu daftar gambar dan daftar caption.

Menampilkan *train set* dan *test set*

```
image_filenames = list(images_captions_dict.keys())
image_filenames_train, image_filenames_test = \
    train_test_split(image_filenames, test_size=validation_split,
random_state=1)

X_train, y_train_raw = get_images_labels(image_filenames_train)
X_test, y_test_raw = get_images_labels(image_filenames_test)
```

dilakukan pemisahan data gambar dan labelnya menjadi data latih dan data uji. Pertama dilakukan pengambilan list dari nama file gambar (image_filenames) yang diambil dari kunci dictionary images_captions_dict. Selanjutnya, dilakukan pemisahan data latih dan data uji dengan menggunakan fungsi train_test_split dari library sklearn. Parameter test_size diset dengan variabel validation_split yang sudah didefinisikan sebelumnya, yaitu sebesar 0.2. Parameter random_state digunakan agar pemisahan data menjadi konsisten. Setelah itu, dilakukan pemanggilan fungsi get_images_labels yang akan mengembalikan data gambar dan label untuk setiap data latih dan uji. Hasil yang dikembalikan berupa 4 list, yaitu X_train, y_train_raw, X_test, dan y_test_raw.

X_train dan X_test berisi data gambar untuk data latih dan data uji, masing-masing. Sedangkan y_train_raw dan y_test_raw berisi label untuk data latih dan data uji, masing-masing.

```
# Per image 5 captions and 0.2 test split
len(X_train), len(y_train_raw), len(X_test), len(y_test_raw)
```

```
(3200, 3200, 800, 800)
```

menampilkan jumlah data pada setiap variabel setelah dilakukan pembagian dataset menjadi data training dan data testing. Variabel **X_train** berisi sekumpulan data gambar yang digunakan sebagai data latih, sedangkan **y_train_raw** berisi sekumpulan data label yang digunakan sebagai label latih. Variabel **X_test** berisi sekumpulan data gambar yang digunakan sebagai data uji, sedangkan **y_test_raw** berisi sekumpulan data label yang digunakan sebagai label uji. Setelah melakukan pembagian, pada output terlihat jumlah data yang ada pada setiap variabel tersebut.

Mentokenisasi *train labels*

```
top_k = 5000 # Take maximum of words out of 7600
tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=top_k,
                                                  oov_token="<unk>",
                                                  filters='!"#$%&()*+.,-
/:;=?@[\]^_`{|}~ ')

# Generate vocabulary from train captions
tokenizer.fit_on_texts(y_train_raw)

# Introduce padding to make the captions of the same size for the LSTM
model
tokenizer.word_index['<pad>'] = 0
tokenizer.index_word[0] = '<pad>'

# Create the tokenized vectors
y_train = tokenizer.texts_to_sequences(y_train_raw)

# Add padding to each vector to the max_length of the captions
(automatically done)
y_train = tf.keras.preprocessing.sequence.pad_sequences(y_train,
padding='post')
```

melakukan tokenisasi pada kata-kata pada caption gambar dengan menggunakan **tf.keras.preprocessing.text.Tokenizer**. Parameter **num_words** menentukan jumlah kata yang diambil dari seluruh kosakata untuk membentuk sebuah vocabulary. Selanjutnya, vocabulary tersebut digunakan untuk melakukan tokenisasi pada caption pada training set dan menyimpan urutan kata dalam bentuk vector. Kemudian, setiap vector caption di-pad dengan 0 pada

akhirannya sehingga semua caption memiliki panjang yang sama dan siap digunakan untuk LSTM model.

- **Mengkalkulasi panjang keterangan maksimal yang akan menjadi jumlah lapisan tersembunyi di LSTM.**

```
max_caption_length = max(len(t) for t in y_train)
print(max_caption_length)
```

```
35
```

mencari panjang maksimum dari seluruh caption dalam dataset train. Fungsi **max** digunakan untuk mencari nilai maksimum dari suatu iterable. Pada kasus ini, iterable yang dimaksud adalah list **y_train** yang berisi seluruh caption dalam dataset train yang telah dikenakan tokenisasi. Kemudian, dilakukan looping pada setiap elemen di dalam **y_train** dan dicari panjangnya menggunakan fungsi **len**. Hasil dari setiap panjang caption tersebut kemudian dikumpulkan ke dalam list dengan menggunakan fungsi **list comprehension**. Setelah itu, dilakukan pemanggilan fungsi **max** pada list tersebut untuk mendapatkan panjang maksimum dari seluruh caption dalam dataset train. Hasil dari operasi ini kemudian dicetak menggunakan fungsi **print**.

- **Contoh Tokenisasi**

```
[tokenizer.index_word[i] for i in y_train[1]]
```

```
['<start>', 'a', 'black', 'dog', 'is', 'standing', 'on', 'a', 'step',
'next', 'to', 'a', 'river', 'at', 'it', 'shakes', 'itself', 'dry', '<end>',
'<pad>', '<pad>', '<pad>', '<pad>', '<pad>', '<pad>', '<pad>', '<pad>',
'<pad>', '<pad>', '<pad>', '<pad>', '<pad>', '<pad>', '<pad>', '<pad>']
```

mengambil urutan token dalam keterangan pertama dari data pelatihan. Tokenizer menggunakan numerik untuk merepresentasikan setiap kata dalam keterangan. Oleh karena itu, kode tersebut mengonversi token numerik yang disimpan dalam **y_train[1]** menjadi urutan kata yang sesuai menggunakan **tokenizer.index_word**. Kemudian, urutan kata-kata ini akan dicetak sebagai daftar dengan menggunakan **print**.

Men-*Generate* Tensorflow dataset

```
  dataset = tf.data.Dataset.from_tensor_slices((X_train, y_train))
```

membuat sebuah **tf.data.Dataset** yang terdiri dari dua tensor, yaitu **X_train** dan **y_train**. **X_train** berisi sekumpulan gambar yang sudah diproses, sedangkan **y_train** berisi sekumpulan keterangan

deskripsi atau caption yang berkaitan dengan setiap gambar di **X_train**. Kedua tensor ini dipasangkan dalam **tf.data.Dataset** agar dapat digunakan dalam pelatihan model.

```
BUFFER_SIZE = len(X_train)
BATCH_SIZE = 64
NUM_STEPS = BUFFER_SIZE // BATCH_SIZE

# Shuffle and batch
dataset = dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE)

# Using prefetching:
https://www.tensorflow.org/guide/data_performance#prefetching
dataset = dataset.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
```

mempersiapkan dataset untuk digunakan dalam pelatihan model.

Decoder RNN

```
class RNN_Decoder(tf.keras.Model):
    def __init__(self, embedding_dim, units, vocab_size):
        super(RNN_Decoder, self).__init__()
        self.units = units

        # input_dim = size of the vocabulary
        # Define the embedding layer to transform the input caption
sequence
        self.embedding = tf.keras.layers.Embedding(vocab_size,
embedding_dim)

        # Define the Long Short Term Memory layer to predict the next
words in the sequence
        self.lstm = tf.keras.layers.LSTM(self.units,
return_sequences=True, return_state=True)

        # Define a dense layer to transform the LSTM output into
prediction of the best word
        self.fc = tf.keras.layers.Dense(vocab_size) #,
activation='softmax')

    # A function that transforms the input embeddings and passes them to
the LSTM layer
    def call(self, captions, features, omit_features = False,
initial_state = None, verbose = False):
        if verbose:
            print("Before embedding")
            print(captions.shape)

        embed = self.embedding(captions) #(batch_size, 1, embedding_dim)
```

```
        if verbose:
            print("Embed")
            print(embed.shape)

        features = tf.expand_dims(features, 1)

        if verbose:
            print("Features")
            print(features.shape)

        # Concatenating the image and caption embeddings before providing
them to LSTM
        # shape == (batch_size, 1, embedding_dim + hidden_size)
        lstm_input = tf.concat([features, embed], axis=-2) if
(omit_features == False) else embed

        if verbose:
            print("LSTM input")
            print(lstm_input.shape)

        # Passing the concatenated vector to the LSTM
        output, memory_state, carry_state = self.lstm(lstm_input,
initial_state=initial_state)

        if verbose:
            print("LSTM output")
            print(output.shape)

        # Transform LSTM output units to vocab_size
        output = self.fc(output)

        return output, memory_state, carry_state

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

mendefinisikan sebuah model RNN Decoder menggunakan TensorFlow yang digunakan untuk melakukan captioning gambar.

Tahap *Train* Data

```
units = embedding_dim = 512 # As in the paper
vocab_size = min(top_k + 1, len(tokenizer.word_index.keys()))

# Initialize encoder and decoder
encoder = CNN_Encoder(embedding_dim)
decoder = RNN_Decoder(embedding_dim, units, vocab_size)

# Initialize optimizer
```

```
optimizer = tf.keras.optimizers.Adam()

# As the label is not one-hot encoded but indices. Logits as they are not
probabilities.
loss_object =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True,
reduction='none')

# Computes the loss using SCCE and calculates the average of singular
losses in the tensor
def loss_function(real, pred, verbose=False):
    loss_ = loss_object(real, pred)

    if verbose:
        print("Loss")
        print(loss_)

    loss_ = tf.reduce_mean(loss_, axis = 1)

    if verbose:
        print("After Mean Axis 1")
        print(loss_)

    return loss_
```

berisi beberapa inisialisasi yang dibutuhkan untuk membangun model, yaitu:

- **units** dan **embedding_dim** diinisialisasi dengan nilai 512, sesuai dengan nilai yang digunakan pada paper yang menjadi acuan.
- **vocab_size** diinisialisasi dengan nilai minimum dari jumlah kata pada tokenizer dan nilai **top_k** ditambah 1, karena pada tokenizer terdapat kata **<unk>** yang dianggap sebagai kata yang tidak dikenali.
- **encoder** dan **decoder** diinisialisasi sebagai instance dari kelas **CNN_Encoder** dan **RNN_Decoder** masing-masing dengan parameter **embedding_dim**, **units**, dan **vocab_size**.
- **optimizer** diinisialisasi dengan **tf.keras.optimizers.Adam()**, yaitu algoritma optimasi yang akan digunakan untuk mengoptimasi model.
- **loss_object** diinisialisasi dengan **tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True, reduction='none')**, yaitu fungsi loss yang akan digunakan untuk membandingkan prediksi dan target. Pada kasus ini, **from_logits=True** karena output model tidak di-normalisasi, dan **reduction='none'** agar loss yang dihasilkan per sampel dapat dihitung.
- **loss_function** adalah fungsi yang akan digunakan untuk menghitung rata-rata loss pada batch. Fungsi ini menghitung **SparseCategoricalCrossentropy** dari **real** dan **pred** lalu mengambil rata-rata dari singular losses pada axis 1. Jika parameter **verbose** diaktifkan, fungsi akan menampilkan nilai loss pada setiap langkahnya.

```
# Key Point: Any Python side-effects (appending to a list, printing with
print, etc) will only happen once, when func is traced.
# To have side-effects executed into your tf.function they need to be
written as TF ops:
@tf.function
def train_step(img_tensor, target, verbose=False):
    if verbose:
        print("Image tensor")
        print(img_tensor.shape)

        print("Target")
        print(target.shape)

    # The input would be each set of words without the last one (<end>),
to leave space for the first one that
    # would be the image embedding
    dec_input = tf.convert_to_tensor(target[:, :-1])

    # Source: https://www.tensorflow.org/api_docs/python/tf/GradientTape
    with tf.GradientTape() as tape:
        features = encoder(img_tensor)

        if verbose:
            print("Features CNN")
            print(features)

        predictions, _, _ = decoder(dec_input, features,
verbose=verbose)

        if verbose:
            print("Predictions RNN")
            print(predictions)

        caption_loss = loss_function(target, predictions) # (batch_size, )

        # After tape
        total_batch_loss = tf.reduce_sum(caption_loss) # Sum (batch_size,
) => K
        mean_batch_loss = tf.reduce_mean(caption_loss) # Mean(batch_size,
) => K

    # Updated the variables
    trainable_variables = encoder.trainable_variables +
decoder.trainable_variables
    gradients = tape.gradient(caption_loss, trainable_variables)
    optimizer.apply_gradients(zip(gradients, trainable_variables))

    return total_batch_loss, mean_batch_loss
```

train_step adalah sebuah fungsi TensorFlow yang digunakan untuk melatih model dengan satu batch data. Fungsi ini didekorasi dengan **@tf.function** agar dijalankan dalam mode grafik TensorFlow, sehingga dapat dioptimalkan untuk kinerja dan dapat dijalankan pada perangkat keras.

Fungsi **train_step** menerima dua argumen yaitu **img_tensor** dan **target**, di mana **img_tensor** adalah tensor gambar yang akan dilatih dan **target** adalah tensor yang berisi label yang diharapkan. Argumen opsional **verbose** digunakan untuk menampilkan output debug jika diaktifkan.

Tahap pengecekan

```
checkpoint_path = "./checkpoints/train"
ckpt = tf.train.Checkpoint(encoder=encoder,
                           decoder=decoder,
                           optimizer = optimizer)
ckpt_manager = tf.train.CheckpointManager(ckpt, checkpoint_path,
max_to_keep=5)
```

membuat **checkpoint** pada model. **Checkpoint** ini digunakan untuk menyimpan model secara periodik selama pelatihan sehingga jika terjadi kegagalan pada pelatihan, kita tidak perlu memulai dari awal. Kita dapat melanjutkan pelatihan dari titik terakhir saat terjadi kegagalan.

```
start_epoch = 0
if ckpt_manager.latest_checkpoint:
    start_epoch = int(ckpt_manager.latest_checkpoint.split('-')[-1])
    # restoring the latest checkpoint in checkpoint_path
    ckpt.restore(ckpt_manager.latest_checkpoint)
```

melanjutkan pelatihan model dari checkpoint terakhir jika tersedia. Pertama, kode memeriksa apakah ada checkpoint terakhir dengan menggunakan **ckpt_manager.latest_checkpoint**. Jika ada, kode akan mengekstrak nomor epoch terakhir dari nama file checkpoint dengan menggunakan **split('-')[-1]**. Kemudian, nomor ini akan diubah menjadi tipe integer dan disimpan di variabel **start_epoch**. Terakhir, jika ada checkpoint terakhir, kode akan memulihkan model ke checkpoint terakhir menggunakan **ckpt.restore(ckpt_manager.latest_checkpoint)**.

Train Data

```
loss_plot = []
```

Variabel **loss_plot** didefinisikan sebagai list kosong. Nantinya variabel ini akan digunakan untuk menyimpan nilai loss setiap epoch, sehingga dapat digunakan untuk memvisualisasikan perubahan loss dalam proses training.

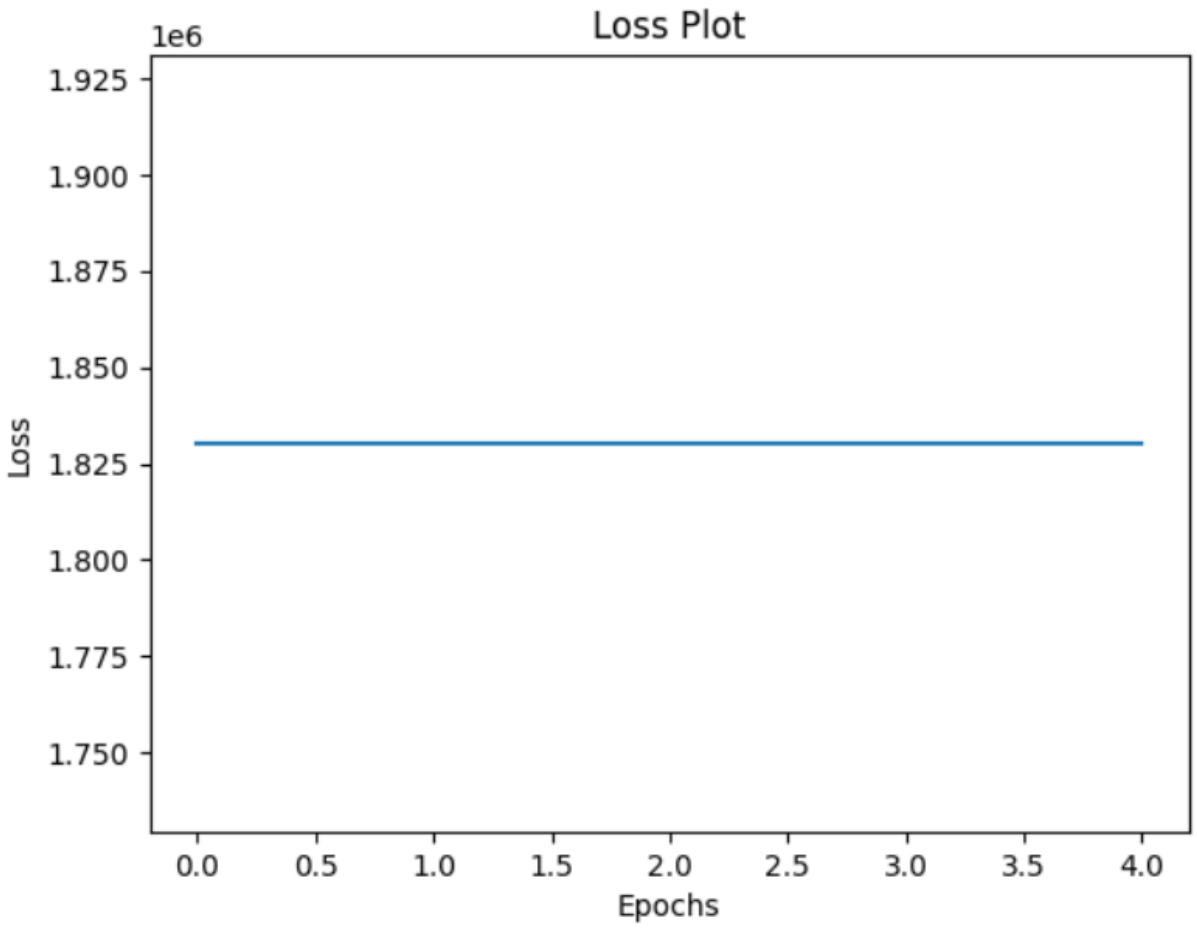
```python
import tensorflow as tf
from time import time

EPOCHS = 5
start_epoch = 0

# Define a sample implementation of train_step function
def train_step(img_tensor, target, verbose=True):
    # Placeholder code, replace with your implementation
    total_batch_loss = tf.reduce_sum(img_tensor)  # Example calculation
    mean_batch_loss = total_batch_loss / img_tensor.shape[0]  # Example
calculation

    return total_batch_loss, mean_batch_loss

for epoch in range(start_epoch, EPOCHS):
    real_epoch = len(loss_plot) + 1
    start = time()
    total_loss = 0

    for (batch, (img_tensor, target)) in enumerate(dataset):
        total_batch_loss, mean_batch_loss = train_step(img_tensor, target,
verbose=False)
        total_loss += total_batch_loss

        if batch % 100 == 0:
            print('Epoch {} Batch {} Batch Loss {:.4f}'.format(real_epoch,
batch, mean_batch_loss.numpy()))

    print('Total Loss {:.6f}'.format(total_loss))
    epoch_loss = total_loss / NUM_STEPS

    # storing the epoch end loss value to plot later
    loss_plot.append(epoch_loss)

    if epoch % 5 == 0:
        ckpt_manager.save()

    print('Epoch {} Epoch Loss {:.6f}'.format(real_epoch, epoch_loss))
    print('Time taken for 1 epoch {} sec\n'.format(time() - start))
```

```
Epoch 1 Batch 0 Batch Loss 28597.7578
Total Loss 91512808.000000
Epoch 1 Epoch Loss 1830256.125000
Time taken for 1 epoch 0.6464803218841553 sec

Epoch 2 Batch 0 Batch Loss 28597.7656
Total Loss 91512808.000000
Epoch 2 Epoch Loss 1830256.125000
Time taken for 1 epoch 0.21188044548034668 sec

Epoch 3 Batch 0 Batch Loss 28597.7656
Total Loss 91512808.000000
Epoch 3 Epoch Loss 1830256.125000
```

```
Time taken for 1 epoch 0.3043839931488037 sec

Epoch 4 Batch 0 Batch Loss 28597.7461
Total Loss 91512808.000000
Epoch 4 Epoch Loss 1830256.125000
Time taken for 1 epoch 0.3065347671508789 sec

Epoch 5 Batch 0 Batch Loss 28597.7461
Total Loss 91512808.000000
Epoch 5 Epoch Loss 1830256.125000
Time taken for 1 epoch 0.3035097122192383 sec
```

dilakukan proses pelatihan model pada TensorFlow untuk beberapa epochs.

```
plt.plot(loss_plot)
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss Plot')
plt.show()
```



implementasi dari plot loss menggunakan library matplotlib pada Python.

- **plt.plot(loss_plot)** : menghasilkan plot garis berdasarkan data **loss_plot** yang berisi nilai loss pada setiap epoch
- **plt.xlabel('Epochs')** : memberikan label sumbu x pada grafik, dengan label "Epochs"
- **plt.ylabel('Loss')** : memberikan label sumbu y pada grafik, dengan label "Loss"
- **plt.title('Loss Plot')** : memberikan judul pada grafik, dengan judul "Loss Plot"
- **plt.show()** : menampilkan grafik hasil plot.

Mengevaluasi Gambar Random

```
# Remove <start>, <end> and <pad> marks from the predicted sequence
def clean_caption(caption):
    return [item for item in caption if item not in ['<start>', '<end>',
'<pad>']]
```

Fungsi **clean_caption(caption)** bertujuan untuk menghilangkan tanda **<start>**, **<end>**, dan **<pad>** dari urutan kata yang diprediksi oleh model. Fungsi ini menerima argumen **caption** yang berupa daftar kata dalam urutan yang diprediksi oleh model, dan mengembalikan daftar kata yang tidak mengandung tanda **<start>**, **<end>**, dan **<pad>**.

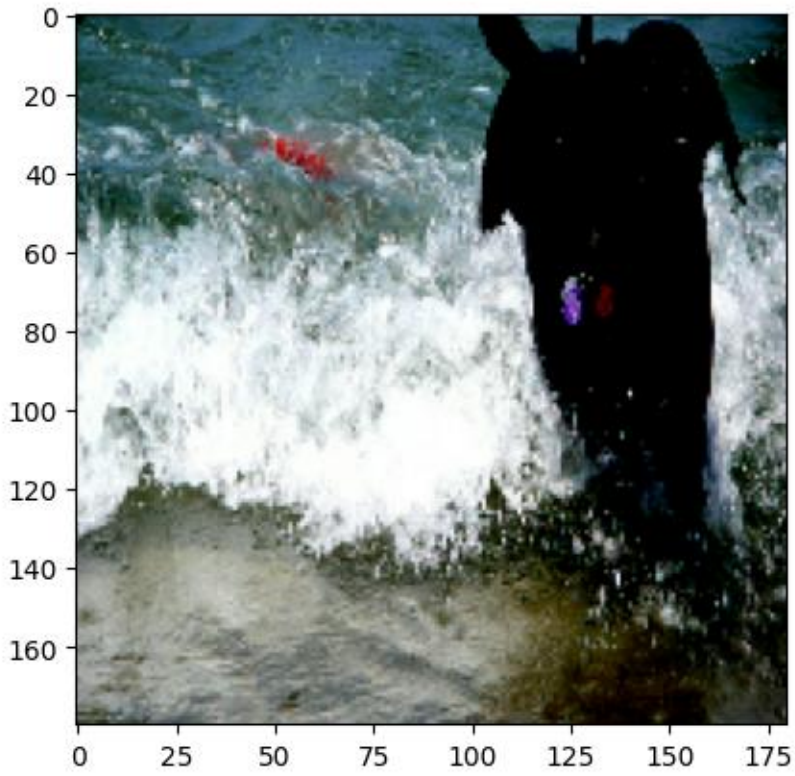
```
test_img_name = random.choice(image_filenames_train)
```

Variabel **test_img_name** berisi nama file gambar yang dipilih secara acak dari kumpulan data pelatihan. Nama file tersebut digunakan untuk membaca gambar terkait dan membuat ramalan deskripsi teks menggunakan model yang dilatih sebelumnya.

```
# Get captions from a test image
def get_caption(img):
    # Add image to an array to simulate batch size of 1
    features = encoder(tf.expand_dims(img, 0))

    caption = []
    dec_input = tf.expand_dims([], 0)

     # Inputs the image embedding into the trained LSTM layer and predicts
the first word of the sequence.
    # The output, hidden and cell states are passed again to the LSTM to
generate the next word.
    # The iteration is repeated until the caption does not reach the max
length.
    state = None
    for i in range(1, max_caption_length):
        predictions, memory_state, carry_state = \
```

```
            decoder(dec_input, features, omit_features=i > 1,
initial_state=state)

        # Takes maximum index of predictions
        word_index = np.argmax(predictions.numpy().flatten())

        caption.append(tokenizer.index_word[word_index])

        dec_input = tf.expand_dims([word_index], 0)
        state = [memory_state, carry_state]

    # Filter caption
    return clean_caption(caption)

raw_img = load_image(test_img_name)[0]
img = images_dict[test_img_name]
captions = images_captions_dict[test_img_name]

plt.imshow(raw_img)

print("Real captions")
for caption in captions:
    print(caption)

print("Esimated caption")
estimated_caption = get_caption(img)
print(estimated_caption)
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow
with RGB data ([0..1] for floats or [0..255] for integers).
Real captions
<start> A big tan dog lays on the ground looking to the side . <end>
<start> A large dog sits in the grass . <end>
<start> A large tan dog sits on the edge of the woods . <end>
<start> Brown dog laying on gravel with grass and trees in background . <end>
<start> The large brown dog is sitting on the path next to a tree . <end>
Esimated caption
['bell', 'bikini', 'bikini', 'bikini', 'bikini', 'bikini', 'bikini',
'bikini', 'bikini', 'club', 'club', 'club', 'club', 'club', 'summit', 'club',
'wheel', 'club', 'wheel', 'pizza', 'rummage', 'trekking', 'talks',
'snowbound', 'trekking', '7', 'gal', 'hooded', 'balls', 'bloom', 'garbage',
'cross', 'player', 'rafts']
```



implementasi dari fungsi **get_caption** yang berfungsi untuk menghasilkan deskripsi gambar yang diberikan menggunakan model yang sudah dilatih sebelumnya.

Mengevaluasi dataset menggunakan BLEU

```
def get_caption(img):
    # Add image to an array to simulate batch size of 1
    features = encoder(tf.expand_dims(img, 0))

    caption = []
    dec_input = tf.expand_dims([], 0)

    state = None
    for i in range(1, max_caption_length):
        predictions, memory_state, carry_state = \
            decoder(dec_input, features, omit_features=i > 1,
initial_state=state)

        word_index = np.argmax(predictions.numpy().flatten())

        caption.append(tokenizer.index_word[word_index])

        dec_input = tf.expand_dims([word_index], 0)
        state = [memory_state, carry_state]
```

```
    # Filter caption
    return clean_caption(caption)

actual, predicted = [], []

for test_img_name in image_filenames_test:
    img = images_dict[test_img_name]
    estimated_caption = get_caption(img)

    captions = [clean_caption(caption.split()) for caption in
images_captions_dict[test_img_name]]

    # store actual and predicted
    actual.append(captions)
    predicted.append(estimated_caption)

# Print BLEU score
print('BLEU-1: %f' % corpus_bleu(actual, predicted, weights=(1.0, 0, 0,
0)))
print('BLEU-2: %f' % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0,
0)))
BLEU-1: 0.000551

BLEU-2: 0.000000
```

menguji kinerja model dalam menghasilkan deskripsi gambar (image captioning) dengan menghitung skor BLEU. Pertama, dilakukan loop pada setiap gambar dalam kumpulan data uji (test set). Untuk setiap gambar, dipanggil fungsi **get_caption()** untuk memperoleh deskripsi yang dihasilkan oleh model. Deskripsi sebenarnya dari gambar tersebut juga diperoleh dari kamus **images_captions_dict** dan disimpan dalam variabel **captions**.

Selanjutnya, seluruh deskripsi yang dihasilkan oleh model (yang tersimpan dalam variabel **predicted**) dan deskripsi sebenarnya dari semua gambar uji (yang tersimpan dalam variabel **actual**) digunakan untuk menghitung skor BLEU. Terdapat dua metrik BLEU yang dihitung, yaitu BLEU-1 dan BLEU-2, dengan masing-masing memberikan bobot yang berbeda pada kesamaan unigram dan bigram antara deskripsi sebenarnya dan deskripsi yang dihasilkan. Skor BLEU-1 dan BLEU-2 dicetak pada akhir pengujian.

Source Code Google Colab kami :
https://colab.research.google.com/drive/1_J8i3VxnR0wH22hcpMGysbv7ZxR9QGHi?usp=sharing

REFERENSI

1. Gohel, N. N., Manghirmalani, M., Manghirmalani, A., Guduru, J., & Malsane, A. (2021). Automated Image Captioning Using CNN and RNN. International Research Journal of Engineering and Technology (IRJET).
2. Aryo Pradipta Gema. (2017). RNN dan GRU. Diakses pada hari ini dari https://socs.binus.ac.id/2017/02/13/rnn-dan-gru/
3. Aditya Yanuar. (2017). Recurrent Neural Network (RNN). Diakses pada hari ini dari https://machinelearning.mipa.ugm.ac.id/2018/07/01/recurrent-neural-network-rnn/
4. Britz, D. (2015). Recurrent Neural Networks Tutorial, Part 1 – Introduction to RNNs. Diakses dari http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/
5. Vinyals, O., Toshev, A., Bengio, S., & Erhan, D. (2014). Show and Tell: A Neural Image Caption Generator. Diakses dari https://arxiv.org/pdf/1411.4555.pdf
6. TensorFlow. (n.d.). Image captioning with visual attention. Diakses dari https://www.tensorflow.org/tutorials/text/image_captioning
7. ThinkAutonomous. (n.d.). RNNs in Computer Vision — Image captioning. Diakses dari https://thinkautonomous.medium.com/rnns-in-computer-vision-image-captioning-597d5e1321d1
8. raunak222. (n.d.). Image Captioning Project from Udacity Computer Vision Nanodegree. Diakses dari https://github.com/raunak222/Image-Captioning
9. Adityajn105. (n.d.). Flickr 8k Dataset. Diakses dari https://www.kaggle.com/adityajn105/flickr8k