Image-to-Text Generation Project

2023 Fall Statistical Deep Learning

Team 3 - Haelee Kim & Nayaeun Kwon

What is Image-to-Text Generation?

1. **Concept**:

   A concept in artificial intelligence where computers learn to describe the content of images using human-like textual language

2. **Objective**:

   To enable machines to understand visual content and express it in natural language, bridging the gap between visual and textual modalities

3. **Workflow**:

   Involves training deep learning models on datasets containing images paired with human-generated captions to learn the correlation between visual features and textual descriptions

4. **Model Component**:

   Utilized Convolutional Neural Networks (CNNs) for extracting image features and Recurrent Neural Networks (RNNs) for generating descriptive captions

About Dataset and Environment

1. Dataset Source

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

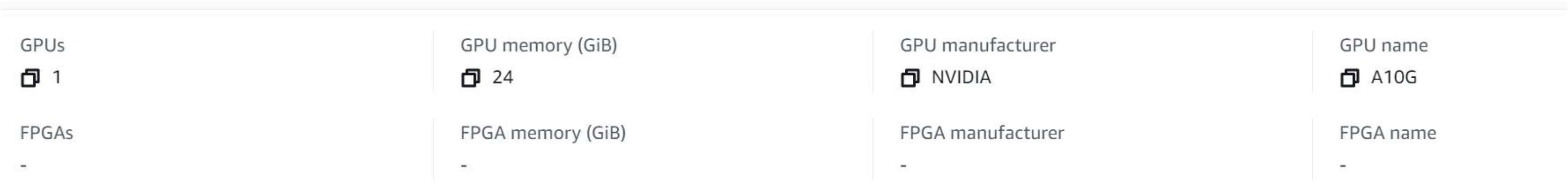
The Grainger College of Engineering
Computer Science

Flickr8k_Dataset.zip
https://github.com/jbrownlee/Datasets/releases/download/Flickr8k/Flickr8k_Dataset.zip

Flickr8k_text.zip
https://github.com/jbrownlee/Datasets/releases/download/Flickr8k/Flickr8k_text.zip

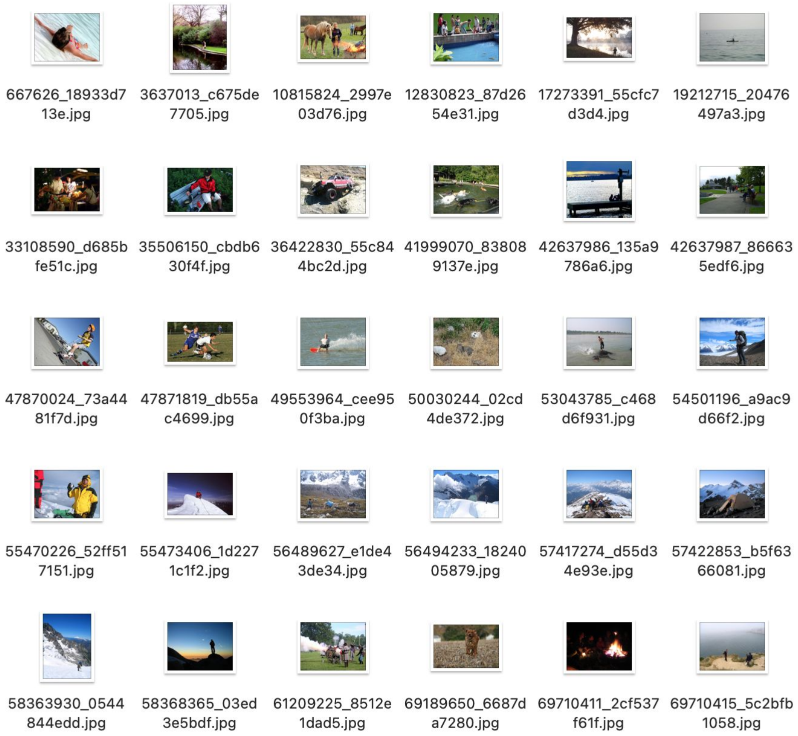
2. Amazon Web Service (AWS)

GPU Environment: EC2 Instance - Instance Type: g5.2xlarge

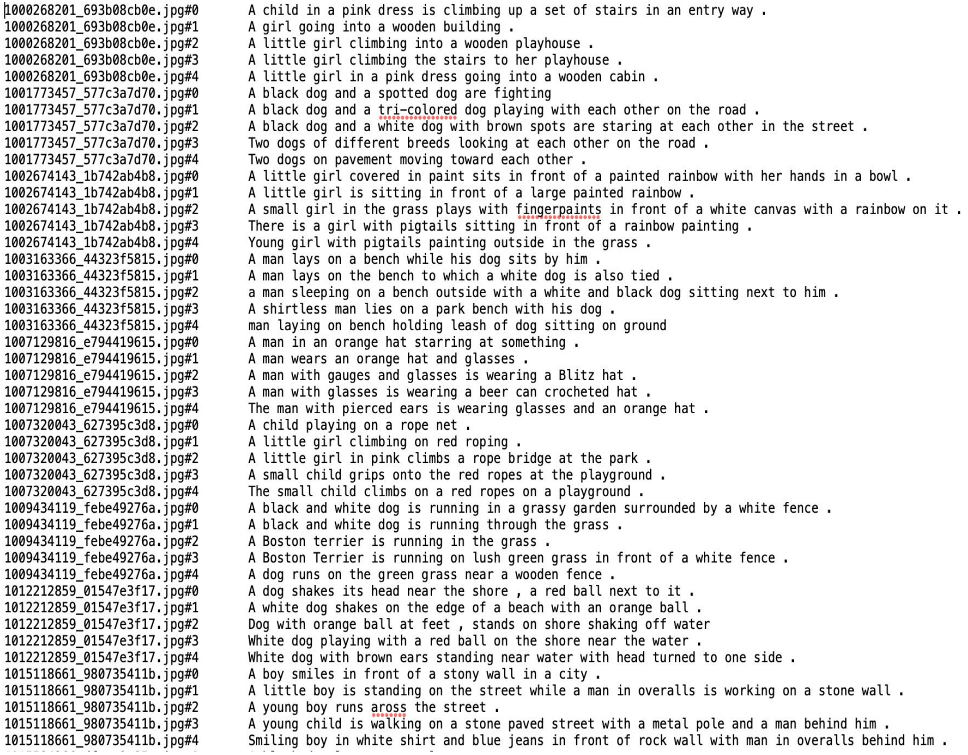


GPUs	GPU memory (GiB)	GPU manufacturer	GPU name
1	24	NVIDIA	A10G
FPGAs	**FPGA memory (GiB)**	**FPGA manufacturer**	**FPGA name**
-	-	-	-

Image Dataset (n = 8091)



Text Dataset (n = 8091 * 5)



```
1000268201_693b08cb0e.jpg#0	A child in a pink dress is climbing up a set of stairs in an entry way .
1000268201_693b08cb0e.jpg#1	A girl going into a wooden building .
1000268201_693b08cb0e.jpg#2	A little girl climbing into a wooden playhouse .
1000268201_693b08cb0e.jpg#3	A little girl climbing the stairs to her playhouse .
1000268201_693b08cb0e.jpg#4	A little girl in a pink dress going into a wooden cabin .
1001773457_577c3a7d70.jpg#0	A black dog and a spotted dog are fighting
1001773457_577c3a7d70.jpg#1	A black dog and a tri-colored dog playing with each other on the road .
1001773457_577c3a7d70.jpg#2	A black dog and a white dog with brown spots are staring at each other in the street .
1001773457_577c3a7d70.jpg#3	Two dogs of different breeds looking at each other on the road .
1001773457_577c3a7d70.jpg#4	Two dogs on pavement moving toward each other .
1002674143_1b742ab4b8.jpg#0	A little girl covered in paint sits in front of a painted rainbow with her hands in a bowl .
1002674143_1b742ab4b8.jpg#1	A little girl is sitting in front of a large painted rainbow .
1002674143_1b742ab4b8.jpg#2	A small girl in the grass plays with fingerpaints in front of a white canvas with a rainbow on it .
1002674143_1b742ab4b8.jpg#3	There is a girl with pigtails sitting in front of a rainbow painting .
1002674143_1b742ab4b8.jpg#4	Young girl with pigtails painting outside in the grass .
1003163366_44323f5815.jpg#0	A man lays on a bench while his dog sits by him .
1003163366_44323f5815.jpg#1	A man lays on the bench to which a white dog is also tied .
1003163366_44323f5815.jpg#2	a man sleeping on a bench outside with a white and black dog sitting next to him .
1003163366_44323f5815.jpg#3	A shirtless man lies on a park bench with his dog .
1003163366_44323f5815.jpg#4	man laying on bench holding leash of dog sitting on ground
1007129816_e794419615.jpg#0	A man in an orange hat starring at something .
1007129816_e794419615.jpg#1	A man wears an orange hat and glasses .
1007129816_e794419615.jpg#2	A man with gauges and glasses is wearing a Blitz hat .
1007129816_e794419615.jpg#3	A man with glasses is wearing a beer can crocheted hat .
1007129816_e794419615.jpg#4	The man with pierced ears is wearing glasses and an orange hat .
1007320043_627395c3d8.jpg#0	A child playing on a rope net .
1007320043_627395c3d8.jpg#1	A little girl climbing on red roping .
1007320043_627395c3d8.jpg#2	A little girl in pink climbs a rope bridge at the park .
1007320043_627395c3d8.jpg#3	A small child grips onto the red ropes at the playground .
1007320043_627395c3d8.jpg#4	The small child climbs on a red ropes on a playground .
1009434119_febe49276a.jpg#0	A black and white dog is running in a grassy garden surrounded by a white fence .
1009434119_febe49276a.jpg#1	A black and white dog is running through the grass .
1009434119_febe49276a.jpg#2	A Boston terrier is running in the grass .
1009434119_febe49276a.jpg#3	A Boston Terrier is running on lush green grass in front of a white fence .
1009434119_febe49276a.jpg#4	A dog runs on the green grass near a wooden fence .
1012212859_01547e3f17.jpg#0	A dog shakes its head near the shore , a red ball next to it .
1012212859_01547e3f17.jpg#1	A white dog shakes on the edge of a beach with an orange ball .
1012212859_01547e3f17.jpg#2	Dog with orange ball at feet , stands on shore shaking off water
1012212859_01547e3f17.jpg#3	White dog playing with a red ball on the shore near the water .
1012212859_01547e3f17.jpg#4	White dog with brown ears standing near water with head turned to one side .
1015118661_980735411b.jpg#0	A boy smiles in front of a stony wall in a city .
1015118661_980735411b.jpg#1	A little boy is standing on the street while a man in overalls is working on a stone wall .
1015118661_980735411b.jpg#2	A young boy runs aross the street .
1015118661_980735411b.jpg#3	A young child is walking on a stone paved street with a metal pole and a man behind him .
1015118661_980735411b.jpg#4	Smiling boy in white shirt and blue jeans in front of rock wall with man in overalls behind him .
```

Workflow

1. Image Feature Extraction
2. Structure textual information with images
3. Text Preprocessing and Tokenization
4. Dataset Splitting for Training and Testing
5. Data Generation
6. Model Generation and Train Model
7. Text Generation

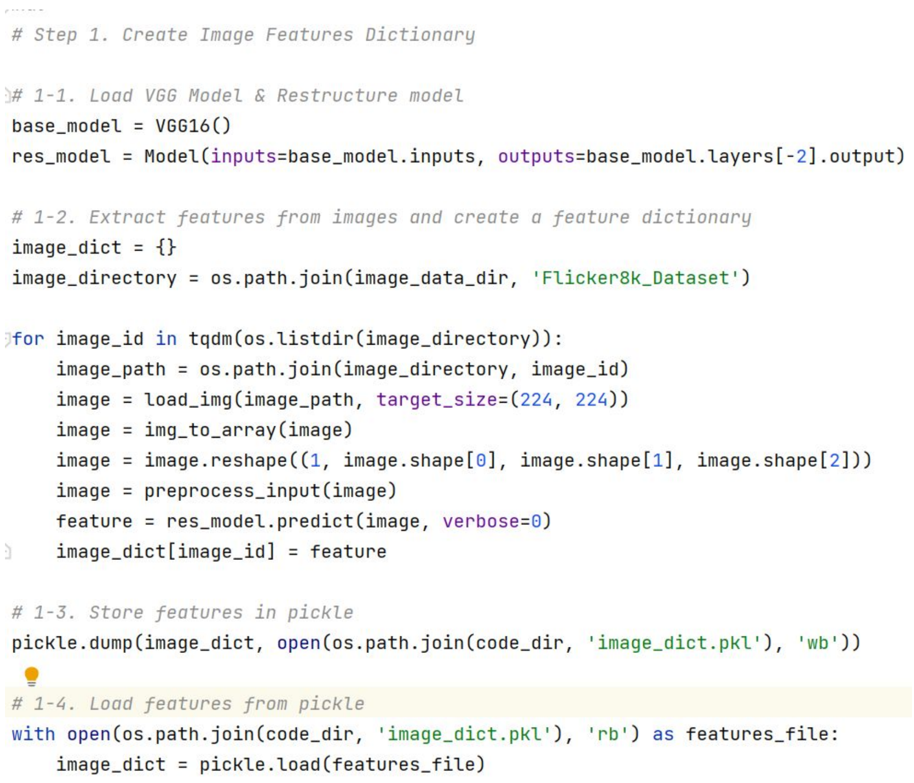
Step 1. Image Feature Extraction - VGG Model

- **What is Visual Geometry Group (VGG) model?**
  - Pre-trained deep convolutional neural network (CNN) for image classification
  - known for its effectiveness in image recognition tasks

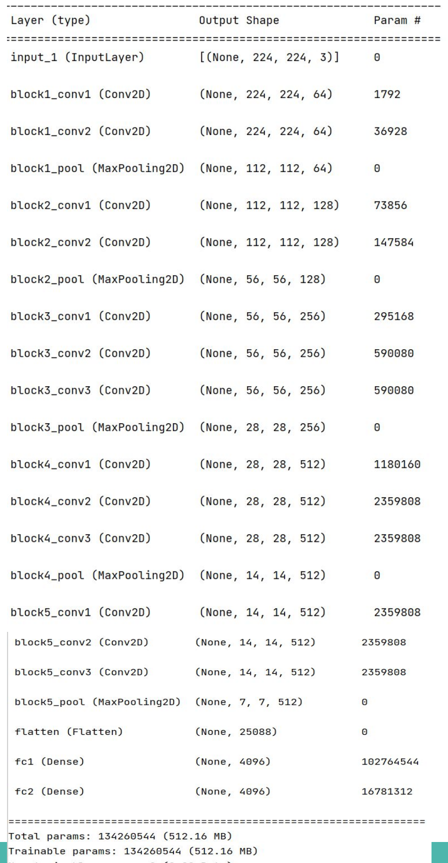
- **What we did?**
  - Extract Features from Images:
    - Process each image in the dataset through the VGG16 model.
    - Extract the high-level features from the images using the pre-trained weights of the VGG16.
  - Create and Store the Image Feature Dictionary:
    - Create a dictionary to associate each image ID with its corresponding feature vector.
    - The feature vector captures the essential characteristics of the image.
    - Store the generated image feature dictionary for later use in the project.

- **Result: A Robust 16-Layer Model**
  - The resulting model comprises 16 layers, encompassing 13 convolutional layers and 3 fully connected layers.
  - These layers collaboratively extract and encapsulate essential features from the input images.

Step 1. Image Feature Extraction - VGG Model



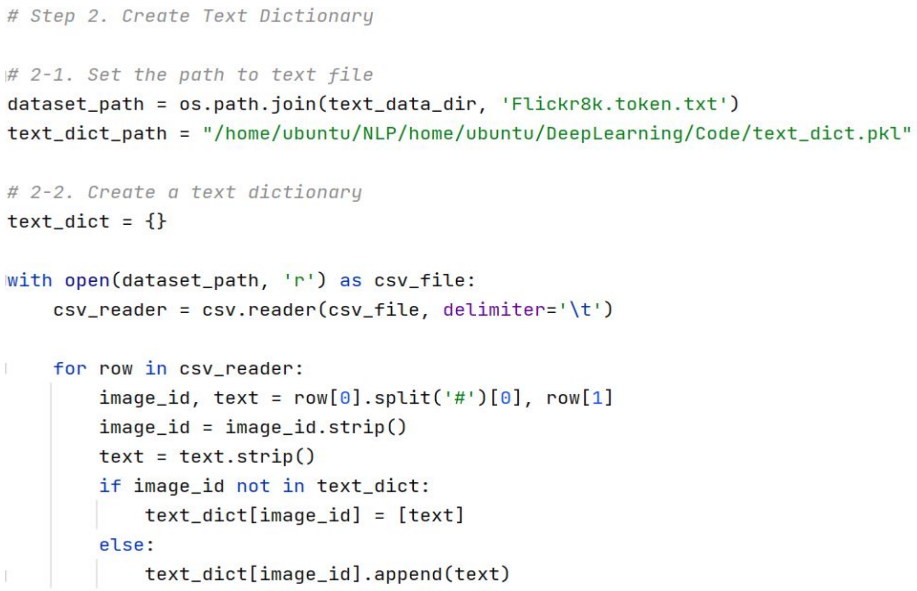
```python
# Step 1. Create Image Features Dictionary

# 1-1. Load VGG Model & Restructure model
base_model = VGG16()
res_model = Model(inputs=base_model.inputs, outputs=base_model.layers[-2].output)

# 1-2. Extract features from images and create a feature dictionary
image_dict = {}
image_directory = os.path.join(image_data_dir, 'Flicker8k_Dataset')

for image_id in tqdm(os.listdir(image_directory)):
    image_path = os.path.join(image_directory, image_id)
    image = load_img(image_path, target_size=(224, 224))
    image = img_to_array(image)
    image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
    image = preprocess_input(image)
    feature = res_model.predict(image, verbose=0)
    image_dict[image_id] = feature

# 1-3. Store features in pickle
pickle.dump(image_dict, open(os.path.join(code_dir, 'image_dict.pkl'), 'wb'))

# 1-4. Load features from pickle
with open(os.path.join(code_dir, 'image_dict.pkl'), 'rb') as features_file:
    image_dict = pickle.load(features_file)
```



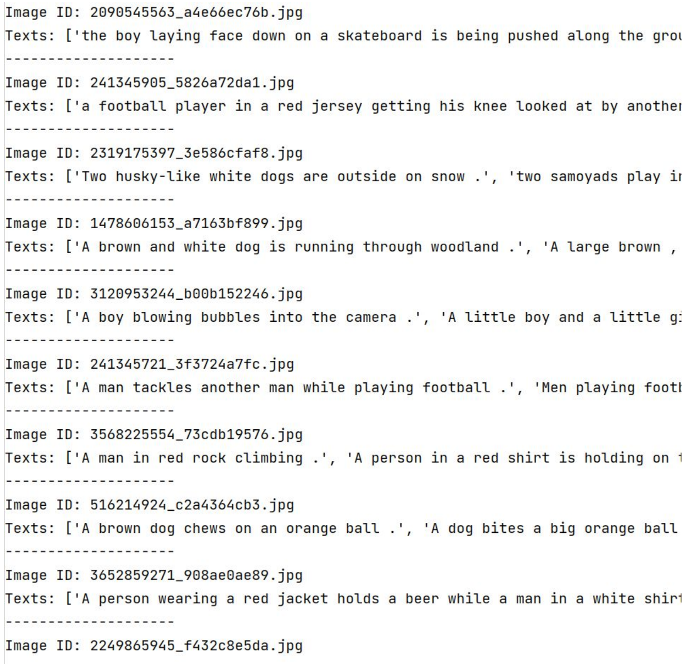
```
----------------------------------------------------------------
 Layer (type)                Output Shape              Param #
================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 fc1 (Dense)                 (None, 4096)              102764544

 fc2 (Dense)                 (None, 4096)              16781312

================================================================
Total params: 134260544 (512.16 MB)
Trainable params: 134260544 (512.16 MB)
```

Step 2. Structure textual information with images

- Crucial link between images and textual descriptions.
- Key dataset for training and evaluating the image-to-text generation model.



```python
# Step 2. Create Text Dictionary

# 2-1. Set the path to text file
dataset_path = os.path.join(text_data_dir, 'Flickr8k.token.txt')
text_dict_path = "/home/ubuntu/NLP/home/ubuntu/DeepLearning/Code/text_dict.pkl"

# 2-2. Create a text dictionary
text_dict = {}

with open(dataset_path, 'r') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter='\t')

    for row in csv_reader:
        image_id, text = row[0].split('#')[0], row[1]
        image_id = image_id.strip()
        text = text.strip()
        if image_id not in text_dict:
            text_dict[image_id] = [text]
        else:
            text_dict[image_id].append(text)
```



```
Image ID: 2090545563_a4e66ec76b.jpg
Texts: ['the boy laying face down on a skateboard is being pushed along the grou
--------------------
Image ID: 241345905_5826a72da1.jpg
Texts: ['a football player in a red jersey getting his knee looked at by another
--------------------
Image ID: 2319175397_3e586cfaf8.jpg
Texts: ['Two husky-like white dogs are outside on snow .', 'two samoyads play in
--------------------
Image ID: 1478606153_a7163bf899.jpg
Texts: ['A brown and white dog is running through woodland .', 'A large brown ,
--------------------
Image ID: 3120953244_b00b152246.jpg
Texts: ['A boy blowing bubbles into the camera .', 'A little boy and a little gi
--------------------
Image ID: 241345721_3f3724a7fc.jpg
Texts: ['A man tackles another man while playing football .', 'Men playing footb
--------------------
Image ID: 3568225554_73cdb19576.jpg
Texts: ['A man in red rock climbing .', 'A person in a red shirt is holding on t
--------------------
Image ID: 516214924_c2a4364cb3.jpg
Texts: ['A brown dog chews on an orange ball .', 'A dog bites a big orange ball
--------------------
Image ID: 3652859271_908ae0ae89.jpg
Texts: ['A person wearing a red jacket holds a beer while a man in a white shirt
--------------------
Image ID: 2249865945_f432c8e5da.jpg
```

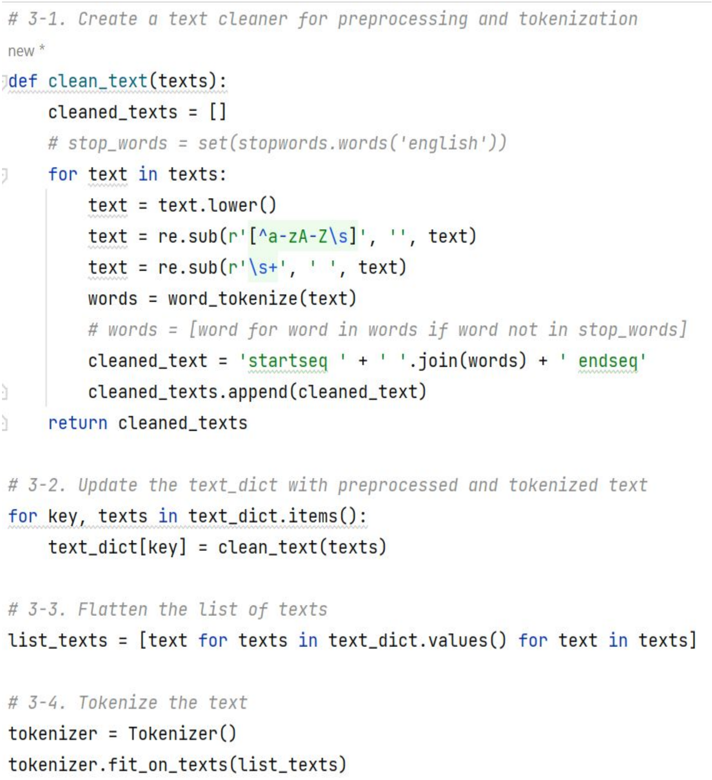
Step 3. Text Preprocessing and Tokenization

Text Preprocess for refined and standardized textual information

- NLTK library for nlp task
- Convert text to lowercase
- Remove non-alphabetic characters.
- Tokenize texts into individual words.
- Add special tokens "startseq" and "endseq"

Text Tokenization for mapping of words to numerical indices

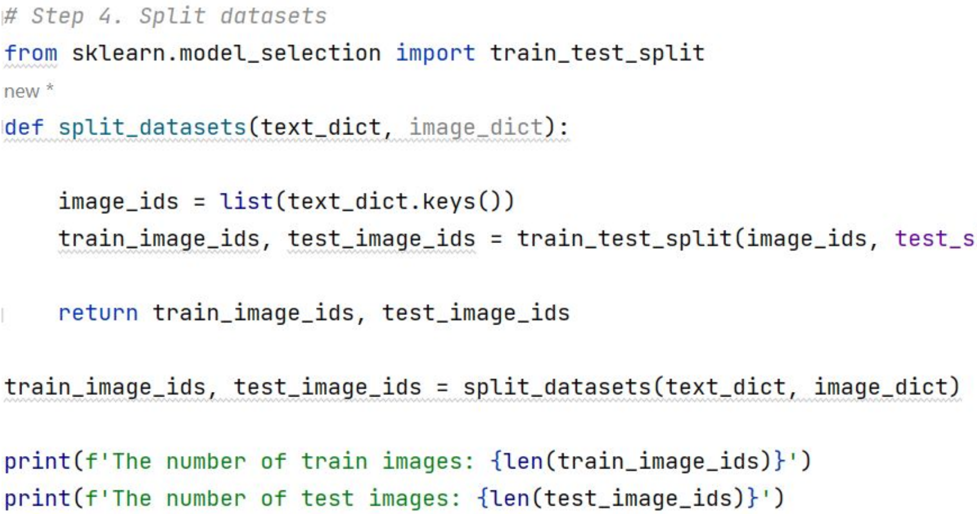
- Use the Keras Tokenizer to tokenize the cleaned captions.
- Construct a vocabulary of unique words present in the entire dataset.
- Determine the size of the vocabulary, a critical factor for model input.
- Prepare textual data for model input



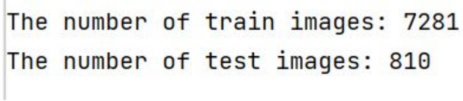
```
# 3-1. Create a text cleaner for preprocessing and tokenization
def clean_text(texts):
    cleaned_texts = []
    # stop_words = set(stopwords.words('english'))
    for text in texts:
        text = text.lower()
        text = re.sub(r'[^a-zA-Z\s]', '', text)
        text = re.sub(r'\s+', ' ', text)
        words = word_tokenize(text)
        # words = [word for word in words if word not in stop_words]
        cleaned_text = 'startseq ' + ' '.join(words) + ' endseq'
        cleaned_texts.append(cleaned_text)
    return cleaned_texts


# 3-2. Update the text_dict with preprocessed and tokenized text
for key, texts in text_dict.items():
    text_dict[key] = clean_text(texts)


# 3-3. Flatten the list of texts
list_texts = [text for texts in text_dict.values() for text in texts]


# 3-4. Tokenize the text
tokenizer = Tokenizer()
tokenizer.fit_on_texts(list_texts)
```

Step 4. Dataset Splitting for Training and Testing

- Utilized the train_test_split function from sklearn to divide the dataset into training and testing sets.
- 90% of the data was allocated for training, and 10% for testing.



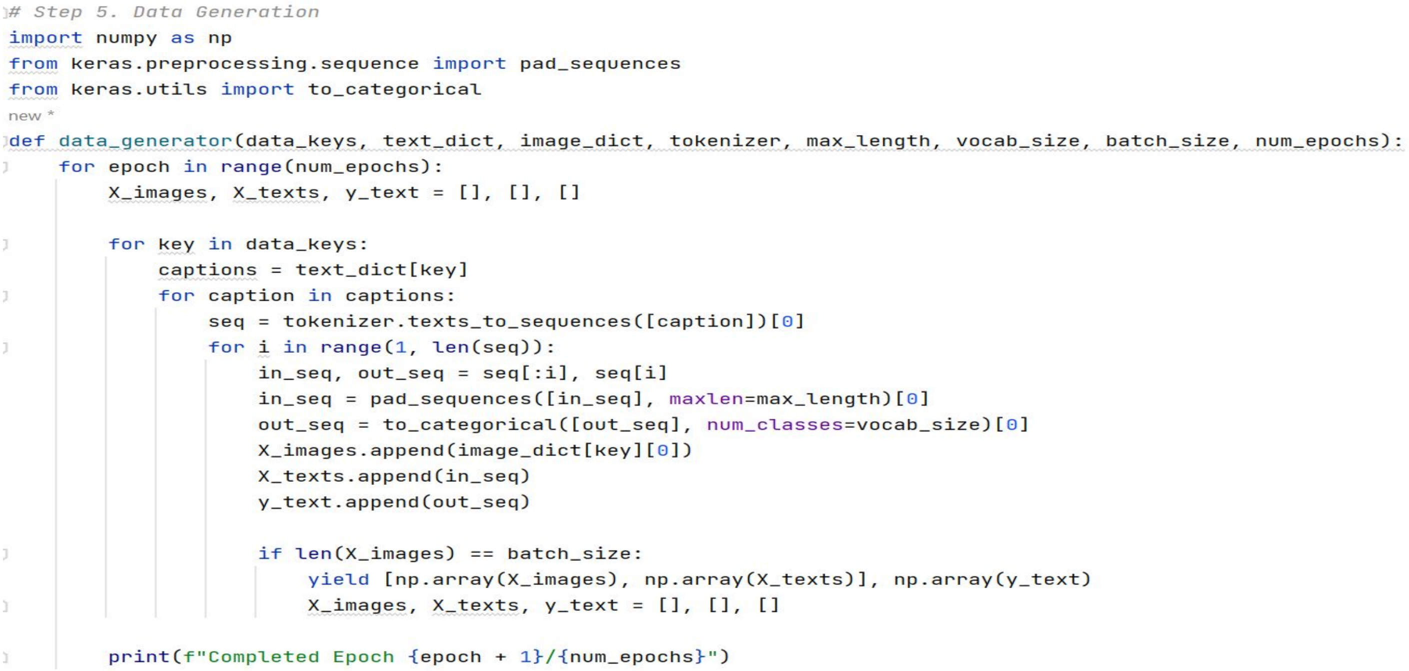
```
# Step 4. Split datasets
from sklearn.model_selection import train_test_split
new *
def split_datasets(text_dict, image_dict):

    image_ids = list(text_dict.keys())
    train_image_ids, test_image_ids = train_test_split(image_ids, test_size=0.10, random_state=42)

    return train_image_ids, test_image_ids

train_image_ids, test_image_ids = split_datasets(text_dict, image_dict)

print(f'The number of train images: {len(train_image_ids)}')
print(f'The number of test images: {len(test_image_ids)}')
```



```
The number of train images: 7281
The number of test images: 810
```

Step 5. Data Generation

This process enables the model to associate image features with textual descriptions by learning the sequential relationships within captions, allowing it to generate contextually relevant text.

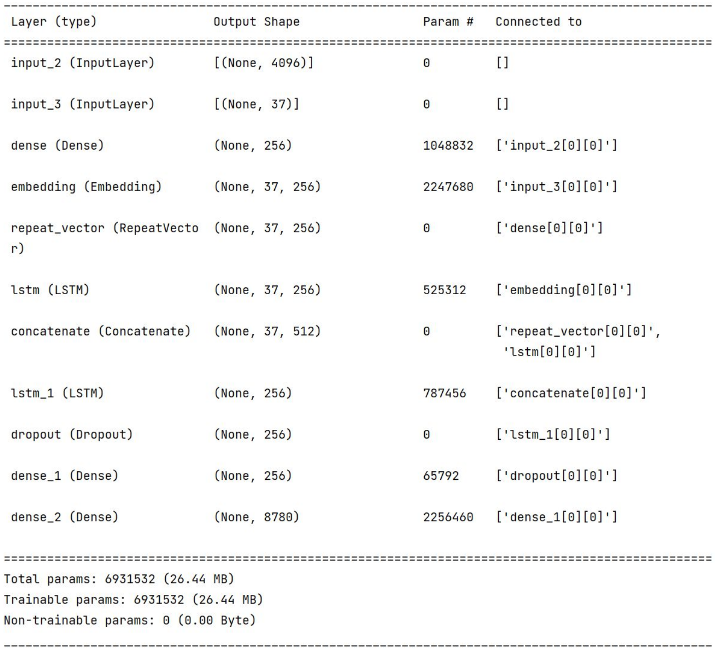


```
# Step 5. Data Generation
import numpy as np
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
new *
def data_generator(data_keys, text_dict, image_dict, tokenizer, max_length, vocab_size, batch_size, num_epochs):
    for epoch in range(num_epochs):
        X_images, X_texts, y_text = [], [], []

        for key in data_keys:
            captions = text_dict[key]
            for caption in captions:
                seq = tokenizer.texts_to_sequences([caption])[0]
                for i in range(1, len(seq)):
                    in_seq, out_seq = seq[:i], seq[i]
                    in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                    out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
                    X_images.append(image_dict[key][0])
                    X_texts.append(in_seq)
                    y_text.append(out_seq)

                    if len(X_images) == batch_size:
                        yield [np.array(X_images), np.array(X_texts)], np.array(y_text)
                        X_images, X_texts, y_text = [], [], []

        print(f"Completed Epoch {epoch + 1}/{num_epochs}")
```

Step 6. Model Generation and Train Model

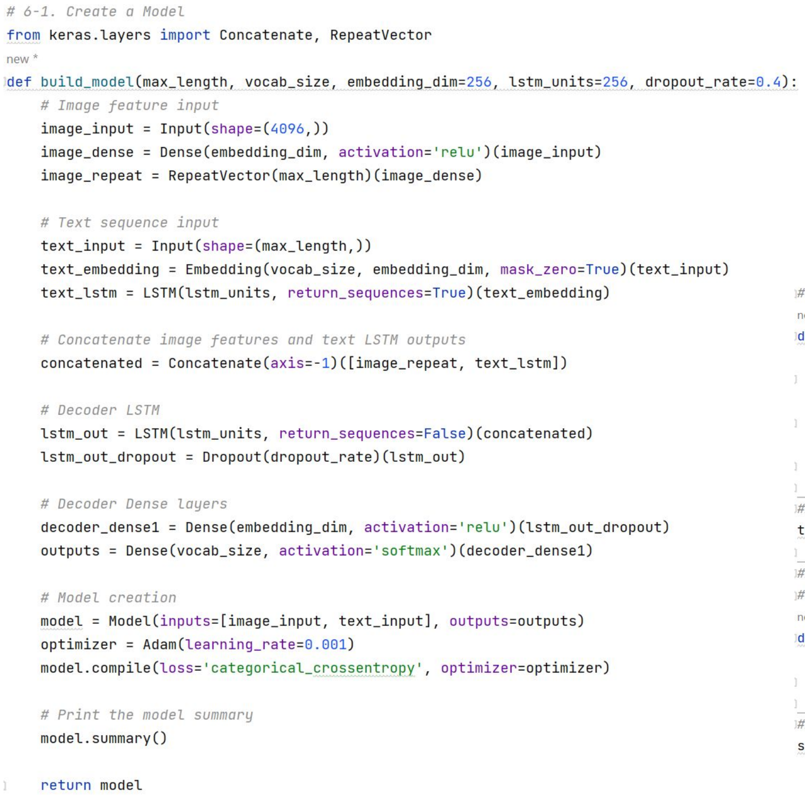
Model Architecture

- Input Layers:
  - Image feature input with a Dense layer.
  - Text sequence input with an Embedding layer.
- Encoder-Decoder Structure:
  - Utilizes an encoder-decoder structure for generating sequences.
- Image-Text Concatenation:
  - Concatenates image features and text LSTM outputs.
- LSTM Layers:
  - Employs Long Short-Term Memory (LSTM) layers for sequence learning.
- Dense Layers:
  - Includes Dense layers for output generation.
- Dropout Regularization:
  - Applies dropout regularization to prevent overfitting.
- Activation Functions - ReLU activation and softmax activation
- Optimizer: Adam optimizer with a learning rate of 0.001.
- Loss Function: Categorical crossentropy



```
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_2 (InputLayer)        [(None, 4096)]               0         []

 input_3 (InputLayer)        [(None, 37)]                 0         []

 dense (Dense)               (None, 256)                  1048832   ['input_2[0][0]']

 embedding (Embedding)       (None, 37, 256)              2247680   ['input_3[0][0]']

 repeat_vector (RepeatVecto  (None, 37, 256)              0         ['dense[0][0]']
 r)

 lstm (LSTM)                 (None, 37, 256)              525312    ['embedding[0][0]']

 concatenate (Concatenate)   (None, 37, 512)              0         ['repeat_vector[0][0]',
                                                                     'lstm[0][0]']

 lstm_1 (LSTM)               (None, 256)                  787456    ['concatenate[0][0]']

 dropout (Dropout)           (None, 256)                  0         ['lstm_1[0][0]']

 dense_1 (Dense)             (None, 256)                  65792     ['dropout[0][0]']

 dense_2 (Dense)             (None, 8780)                 2256460   ['dense_1[0][0]']

==================================================================================================
Total params: 6931532 (26.44 MB)
Trainable params: 6931532 (26.44 MB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
```

Step 6. Model Generation and Train Model

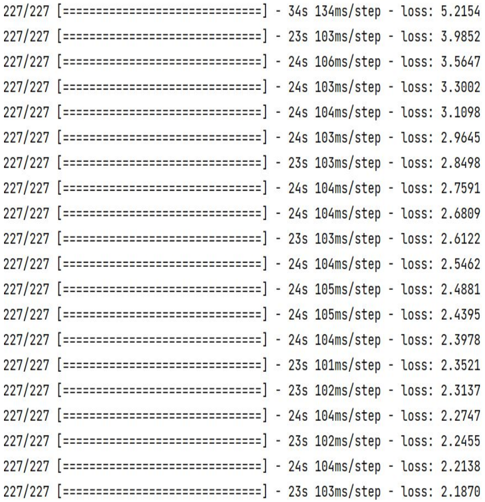


```
# 6-1. Create a Model
from keras.layers import Concatenate, RepeatVector
new *
def build_model(max_length, vocab_size, embedding_dim=256, lstm_units=256, dropout_rate=0.4):
    # Image feature input
    image_input = Input(shape=(4096,))
    image_dense = Dense(embedding_dim, activation='relu')(image_input)
    image_repeat = RepeatVector(max_length)(image_dense)

    # Text sequence input
    text_input = Input(shape=(max_length,))
    text_embedding = Embedding(vocab_size, embedding_dim, mask_zero=True)(text_input)
    text_lstm = LSTM(lstm_units, return_sequences=True)(text_embedding)

    # Concatenate image features and text LSTM outputs
    concatenated = Concatenate(axis=-1)([image_repeat, text_lstm])

    # Decoder LSTM
    lstm_out = LSTM(lstm_units, return_sequences=False)(concatenated)
    lstm_out_dropout = Dropout(dropout_rate)(lstm_out)

    # Decoder Dense layers
    decoder_dense1 = Dense(embedding_dim, activation='relu')(lstm_out_dropout)
    outputs = Dense(vocab_size, activation='softmax')(decoder_dense1)

    # Model creation
    model = Model(inputs=[image_input, text_input], outputs=outputs)
    optimizer = Adam(learning_rate=0.001)
    model.compile(loss='categorical_crossentropy', optimizer=optimizer)

    # Print the model summary
    model.summary()

    return model
```

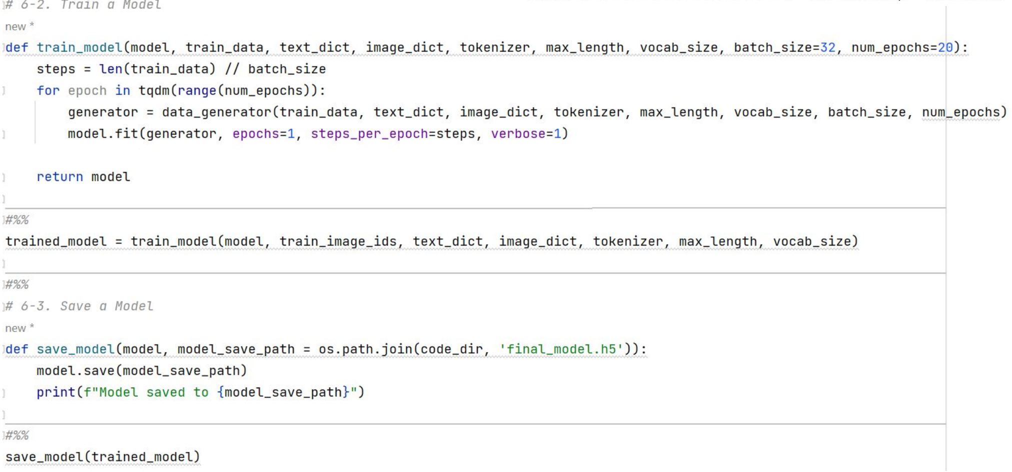
Step 1. Created a Model

Step 2. Trained a Model

Step 3. Saved a Model

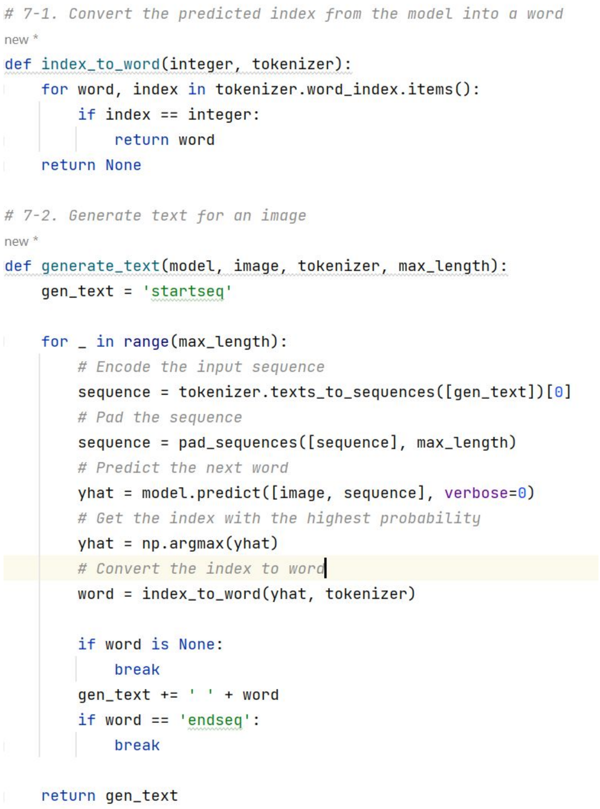


```
227/227 [==============================] - 34s 134ms/step - loss: 5.2154
227/227 [==============================] - 23s 103ms/step - loss: 3.9852
227/227 [==============================] - 24s 106ms/step - loss: 3.5647
227/227 [==============================] - 24s 103ms/step - loss: 3.3002
227/227 [==============================] - 24s 104ms/step - loss: 3.1098
227/227 [==============================] - 24s 103ms/step - loss: 2.9645
227/227 [==============================] - 23s 103ms/step - loss: 2.8498
227/227 [==============================] - 24s 104ms/step - loss: 2.7591
227/227 [==============================] - 24s 104ms/step - loss: 2.6809
227/227 [==============================] - 23s 103ms/step - loss: 2.6122
227/227 [==============================] - 24s 104ms/step - loss: 2.5462
227/227 [==============================] - 24s 105ms/step - loss: 2.4881
227/227 [==============================] - 24s 105ms/step - loss: 2.4395
227/227 [==============================] - 24s 104ms/step - loss: 2.3978
227/227 [==============================] - 23s 101ms/step - loss: 2.3521
227/227 [==============================] - 23s 102ms/step - loss: 2.3137
227/227 [==============================] - 24s 104ms/step - loss: 2.2747
227/227 [==============================] - 23s 102ms/step - loss: 2.2455
227/227 [==============================] - 24s 104ms/step - loss: 2.2138
227/227 [==============================] - 23s 103ms/step - loss: 2.1870
```



```
# 6-2. Train a Model
new *
def train_model(model, train_data, text_dict, image_dict, tokenizer, max_length, vocab_size, batch_size=32, num_epochs=20):
    steps = len(train_data) // batch_size
    for epoch in tqdm(range(num_epochs)):
        generator = data_generator(train_data, text_dict, image_dict, tokenizer, max_length, vocab_size, batch_size, num_epochs)
        model.fit(generator, epochs=1, steps_per_epoch=steps, verbose=1)

    return model

#%%
trained_model = train_model(model, train_image_ids, text_dict, image_dict, tokenizer, max_length, vocab_size)

#%%
# 6-3. Save a Model
new *
def save_model(model, model_save_path = os.path.join(code_dir, 'final_model.h5')):
    model.save(model_save_path)
    print(f"Model saved to {model_save_path}")

#%%
save_model(trained_model)
```

Step 7. Text Generation



```
# 7-1. Convert the predicted index from the model into a word
new *
def index_to_word(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None

# 7-2. Generate text for an image
new *
def generate_text(model, image, tokenizer, max_length):
    gen_text = 'startseq'

    for _ in range(max_length):
        # Encode the input sequence
        sequence = tokenizer.texts_to_sequences([gen_text])[0]
        # Pad the sequence
        sequence = pad_sequences([sequence], max_length)
        # Predict the next word
        yhat = model.predict([image, sequence], verbose=0)
        # Get the index with the highest probability
        yhat = np.argmax(yhat)
        # Convert the index to word
        word = index_to_word(yhat, tokenizer)

        if word is None:
            break
        gen_text += ' ' + word
        if word == 'endseq':
            break

    return gen_text
```

- 7-1. Convert Predicted Index to Word
- 7-2. Generate Text for an Image

The model iteratively predicts the next word in the sequence using the trained model

Initialization:

The input sequence is initialized with the start token 'startseq'.

Prediction Loop:

Iteratively predicts the next word using the trained model.

Involves encoding the current sequence, padding it, and predicting the next word based on image features and the sequence.

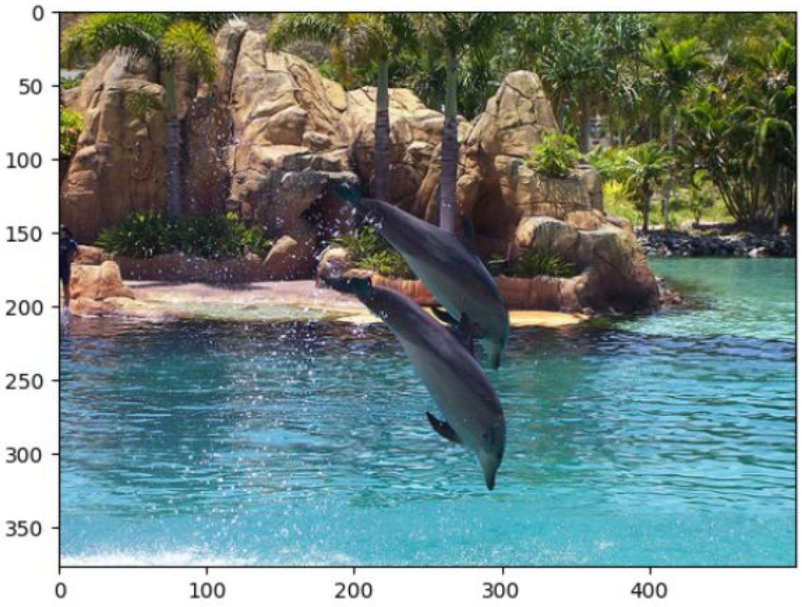
Termination:

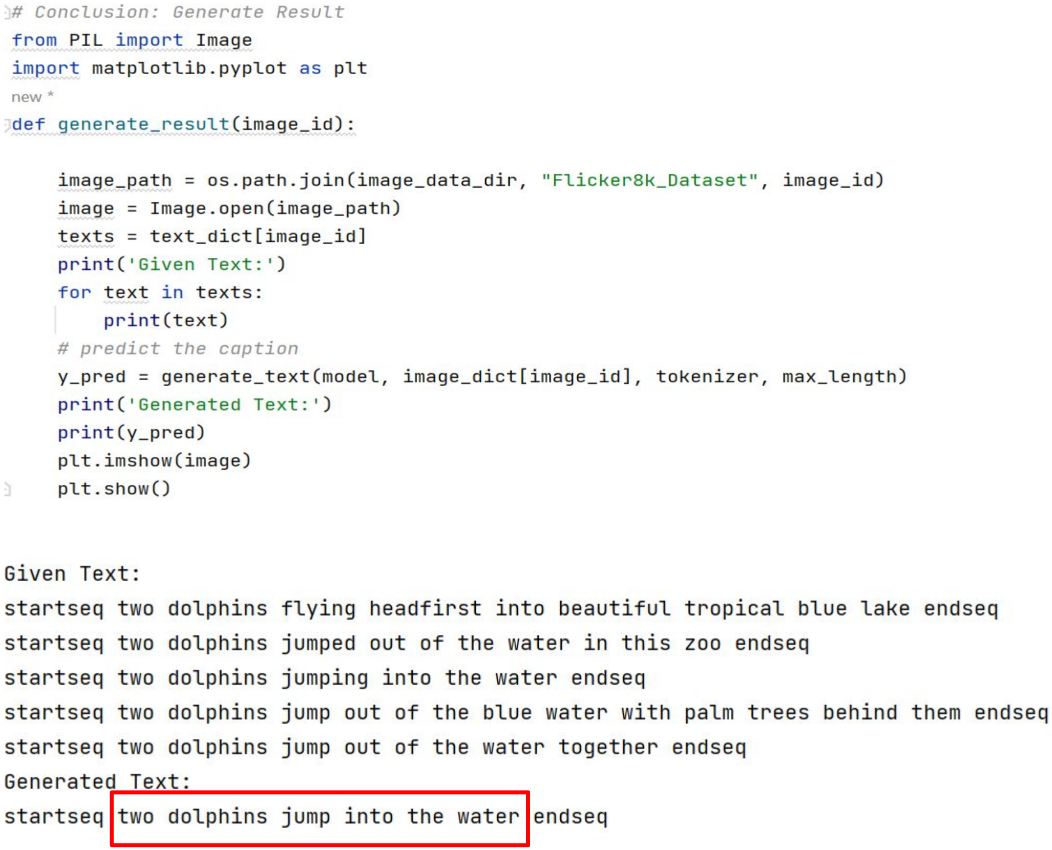
Continues until the maximum length or 'endseq' token is predicted, marking the completion of the caption.

Return:

Generates and returns the textual description for the given image.

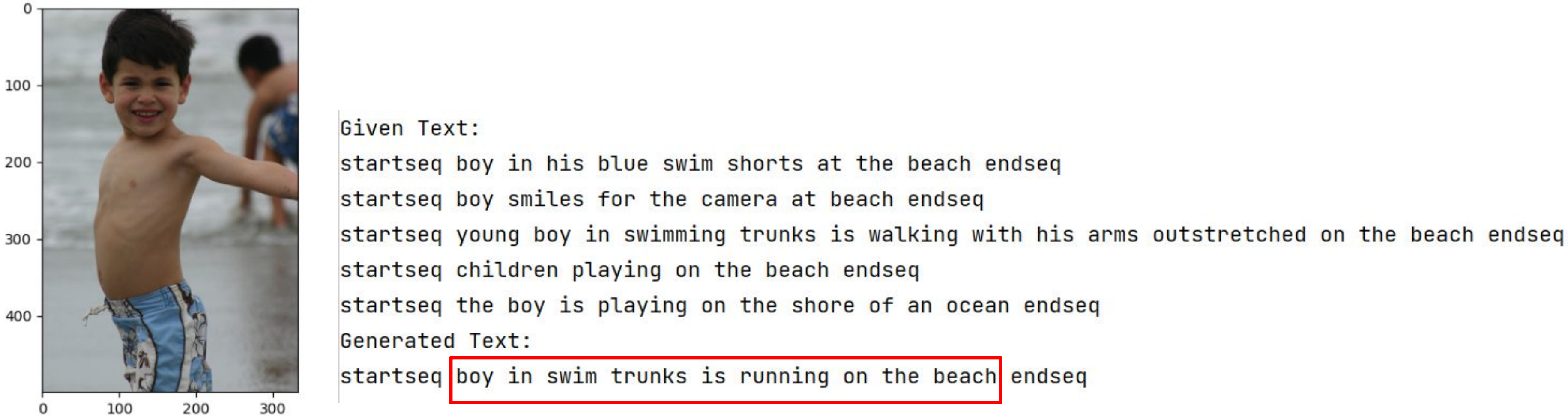
Result



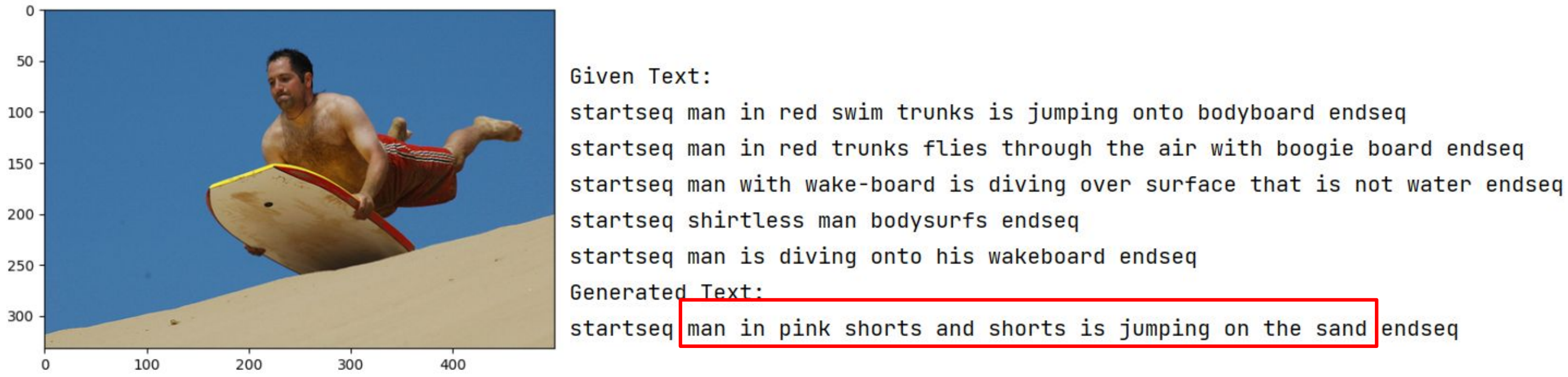


```
# Conclusion: Generate Result
from PIL import Image
import matplotlib.pyplot as plt
new *
def generate_result(image_id):

    image_path = os.path.join(image_data_dir, "Flicker8k_Dataset", image_id)
    image = Image.open(image_path)
    texts = text_dict[image_id]
    print('Given Text:')
    for text in texts:
        print(text)
    # predict the caption
    y_pred = generate_text(model, image_dict[image_id], tokenizer, max_length)
    print('Generated Text:')
    print(y_pred)
    plt.imshow(image)
    plt.show()
```

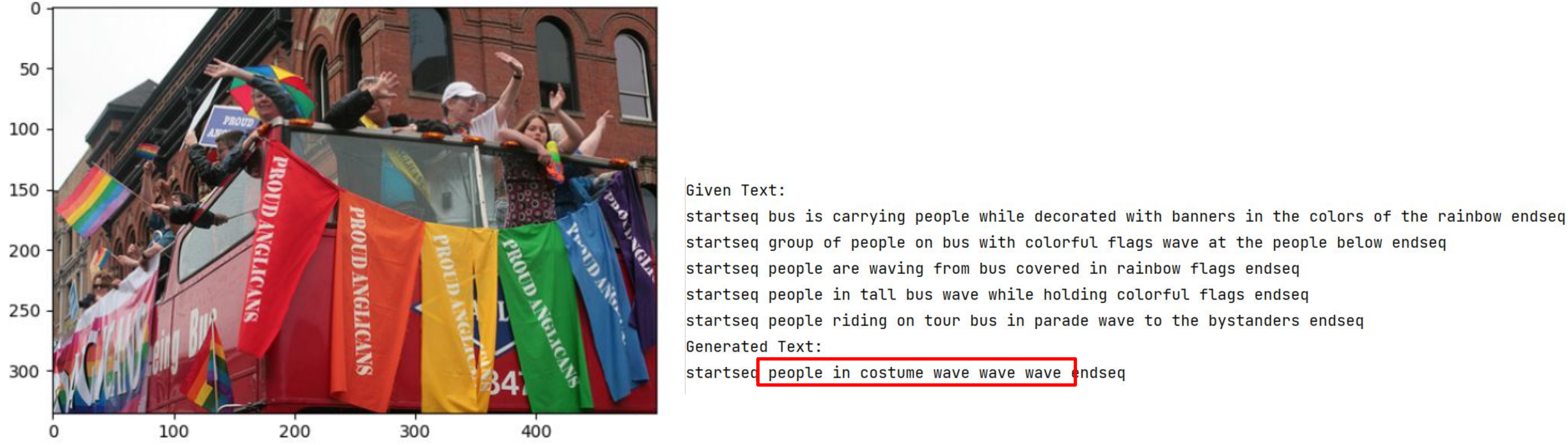
```
Given Text:
startseq two dolphins flying headfirst into beautiful tropical blue lake endseq
startseq two dolphins jumped out of the water in this zoo endseq
startseq two dolphins jumping into the water endseq
startseq two dolphins jump out of the blue water with palm trees behind them endseq
startseq two dolphins jump out of the water together endseq
Generated Text:
startseq two dolphins jump into the water endseq
```



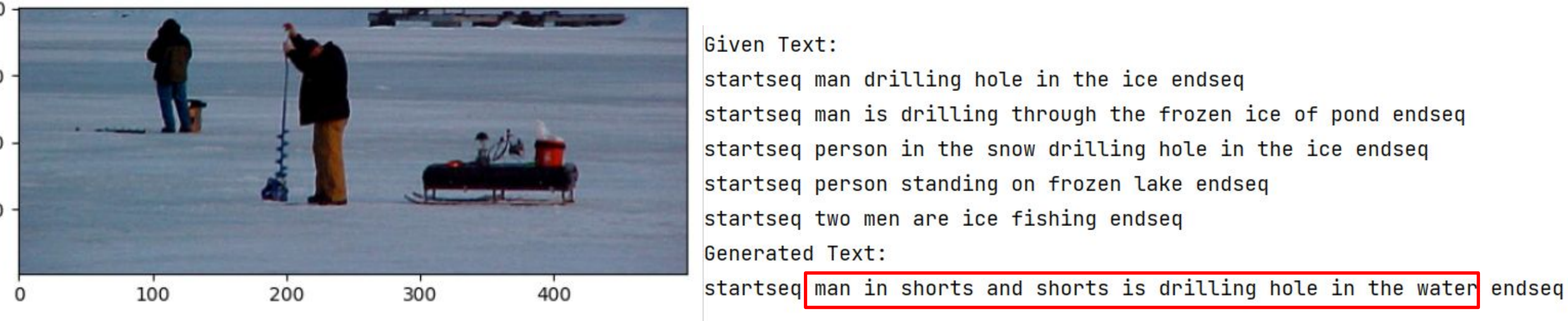
```
Given Text:
startseq boy in his blue swim shorts at the beach endseq
startseq boy smiles for the camera at beach endseq
startseq young boy in swimming trunks is walking with his arms outstretched on the beach endseq
startseq children playing on the beach endseq
startseq the boy is playing on the shore of an ocean endseq
Generated Text:
startseq boy in swim trunks is running on the beach endseq
```



```
Given Text:
startseq man in red swim trunks is jumping onto bodyboard endseq
startseq man in red trunks flies through the air with boogie board endseq
startseq man with wake-board is diving over surface that is not water endseq
startseq shirtless man bodysurfs endseq
startseq man is diving onto his wakeboard endseq
Generated Text:
startseq man in pink shorts and shorts is jumping on the sand endseq
```



```
Given Text:
startseq bus is carrying people while decorated with banners in the colors of the rainbow endseq
startseq group of people on bus with colorful flags wave at the people below endseq
startseq people are waving from bus covered in rainbow flags endseq
startseq people in tall bus wave while holding colorful flags endseq
startseq people riding on tour bus in parade wave to the bystanders endseq
Generated Text:
startseq people in costume wave wave wave endseq
```



```
Given Text:
startseq man drilling hole in the ice endseq
startseq man is drilling through the frozen ice of pond endseq
startseq person in the snow drilling hole in the ice endseq
startseq person standing on frozen lake endseq
startseq two men are ice fishing endseq
Generated Text:
startseq man in shorts and shorts is drilling hole in the water endseq
```

Conclusion

- Successful implementation of a model capable of generating textual descriptions for images.
- Limitation
    - Lack of Fine-tuning
    - Text Length Limitation
    - Handling Rare Words
    - Single GPU Limit

Thank you!

Code Link: github.com/HL-Kim/Image-to-Text-Generation