NAME: BHOOMIKA NANJARAJA
GWID: G41609848
# NLP FINAL PROJECT

## 1. Introduction

With the rapid evolution of multimedia technologies, the ability to generate human-like captions for images is becoming increasingly significant. This capability is essential for improving user interactions and accessibility across various digital platforms. The goal of this project was to build an image captioning model capable of generating meaningful and descriptive captions for visual content. By leveraging Convolutional Neural Networks (CNNs) and Gated Recurrent Units (GRUs), the model combines image feature extraction with natural language processing to create contextually accurate descriptions.

### Project Overview

This project utilizes an advanced encoder-decoder framework with attention mechanisms to tackle the challenges of image captioning. The encoder, based on the pretrained Inception-V3 CNN model, extracts key visual features from input images, transforming them into dense feature representations. The decoder, powered by GRUs, processes these features to generate fluent and contextually relevant captions. Additionally, attention mechanisms are integrated into the architecture, allowing the model to focus on specific parts of an image during the captioning process, thereby improving the quality and accuracy of the output.

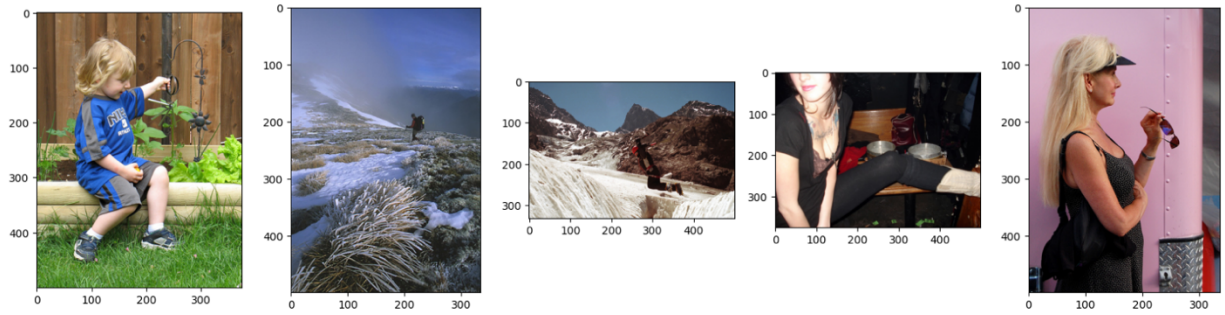
## 2. Dataset Description

This project employs the Flickr8k dataset, a widely used benchmark for image captioning tasks, known for its diversity and manageable size. The dataset provides a robust foundation for training and evaluating the image captioning model.

### Dataset Overview

- **Images**: The dataset consists of 8,091 images sourced from the Flickr platform, showcasing a variety of everyday scenes and activities.
- **Captions**: Each image is annotated with five unique captions, resulting in a total of 40,455 image-caption pairs. These annotations, stored in the file `captions.txt`, provide detailed descriptions of the visual content.
- **Dataset Size**: With multiple captions per image, the dataset offers a rich pool of 40,455 samples, making it an excellent resource for training the model effectively.

## Sample Images



## Sample Captions

```
image,caption
1000268201_693b08cb0e.jpg,A child in a pink dress is climbing up a set of stairs in an entry way .
1000268201_693b08cb0e.jpg,A girl going into a wooden building .
1000268201_693b08cb0e.jpg,A little girl climbing into a wooden playhouse .
1000268201_693b08cb0e.jpg,A little girl climbing
```

Here are the essential packages that I have used in the project:

1. **Core Libraries**:
    - numpy, pandas: For numerical computations and data manipulation.
    - matplotlib.pyplot, seaborn: For visualizations.
    - collections.Counter: For counting elements (e.g., word frequency).

2. **TensorFlow and Keras**:
    - tensorflow, keras: For building and training the deep learning model.
    - tensorflow.keras.applications.inception_v3: For leveraging the InceptionV3 model.
    - tensorflow.keras.layers: For constructing custom layers.
    - tensorflow.keras.preprocessing: For preprocessing images and text.

3. **Image and File Handling**:
    - PIL.Image, imageio: For loading and processing images.
    - glob, os: For file path manipulations.

4. **Progress Visualization**:
    - tqdm: For creating progress bars.

5. **Word Clouds and Audio**:
    - wordcloud.WordCloud: For generating word clouds.

## Key Contributions

- Loaded and visualized images from the dataset using imageio and PIL.Image.
- Extracted captions from a text file and linked them to respective images.
- Cleaned and preprocessed text captions:
    - Lowercased text.
    - Removed punctuation and numeric values.
    - Added <start> and <end> tokens to captions.
- Tokenized captions using keras.preprocessing.text.Tokenizer and created padded sequences for consistent input length.
- Visualized word frequency distributions using bar plots and word clouds.
- Resized and preprocessed images to the InceptionV3 input format (299x299) using TensorFlow utilities.
- Built a feature extraction pipeline with a pretrained InceptionV3 model, storing image features in a dictionary for quick access.
- Constructed a TensorFlow dataset pipeline for efficient image and caption loading, batching, and prefetching.
- Developed the Encoder class using TensorFlow's Model subclassing API, projecting image features into an embedding space with a Dense layer and ReLU activation.
- Implemented visualization functions to display images alongside their captions for qualitative analysis.

The following is a detailed explanation of the sections I contributed to:

1. Visualising first 5 images :

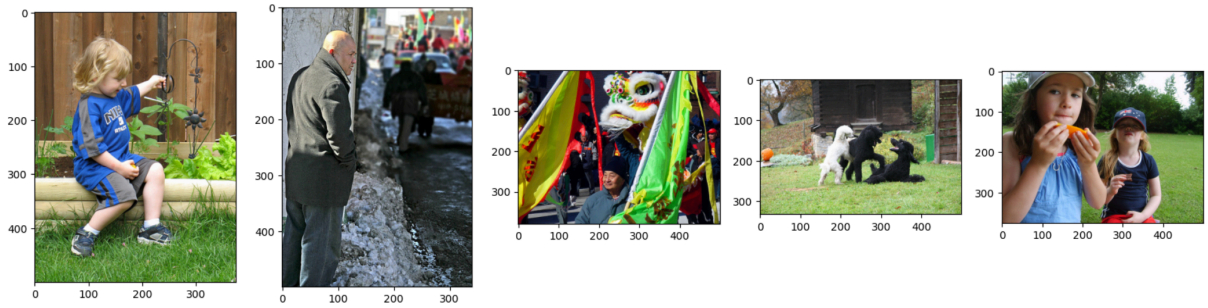
```
import imageio


Display_Images = all_imgs[0:5]
figure, axes = plt.subplots(1,5)
figure.set_figwidth(20)

for ax, image in zip(axes, Display_Images):
  ax.imshow(imageio.imread(image), cmap=None)
```

This function visualizes the first 5 images from the all_imgs list. It creates a horizontal row of 5 subplots using Matplotlib, sets the figure width to 20 for better visibility, and displays each image within its corresponding subplot. The imageio.imread() function is used to read and load each image into the visualization.

```
<ipython-input-10-86938efd3ca9>:11: DeprecationWarning: Starting with ImageIO v3 the behavior of this function will switch t
  ax.imshow(imageio.imread(image), cmap=None)
```



## 2. Loading and Displaying Text Data

```
text_file = '/content/drive/MyDrive/captions.txt'

def load_doc(filename):
        open_file = open(text_file, 'r', encoding='latin-1' )
    text = open_file.read()
    open_file.close()
    return text

doc = load_doc(text_file)
print(doc[:300])
```

The load_doc function reads a text file from the specified filename path (in this case, captions.txt) using the latin-1 encoding. It reads the file's content into a string, closes the file, and returns the string. The first 300 characters of the file content are printed to verify the data.

```
image,caption
1000268201_693b08cb0e.jpg,A child in a pink dress is climbing up a set of stairs in an entry way .
1000268201_693b08cb0e.jpg,A girl going into a wooden building .
1000268201_693b08cb0e.jpg,A little girl climbing into a wooden playhouse .
1000268201_693b08cb0e.jpg,A little girl climbing
```

## 3. Extracting Image IDs, Paths, and Captions

```
img_path = '/content/drive/MyDrive/Images/'
```

```
all_img_id = [] #store all the image id here
all_img_vector = [] #store all the image path here
annotations = [] #store all the captions here

with open('/content/drive/MyDrive/captions.txt' , 'r') as fo:
  next(fo)
  for line in fo :
    split_arr = line.split(',')
    all_img_id.append(split_arr[0])
    annotations.append(split_arr[1].rstrip('\n.'))
    all_img_vector.append(img_path+split_arr[0])

df = pd.DataFrame(list(zip(all_img_id,
all_img_vector,annotations)),columns =['ID','Path', 'Captions'])

df.head(10)
```

- This code extracts image IDs, paths, and captions from a dataset and stores them in separate lists.
- The open function reads the captions file line by line, skipping the first line (header).
- Each line is split into an image ID and caption, with IDs appended to all_img_id, captions to annotations, and complete image paths to all_img_vector.
- A DataFrame df is created to organize the data with columns: ID, Path, and Captions.
- Finally, the first 10 rows of the DataFrame are displayed to inspect the extracted data.

| | ID | Path | Captions |
|---|---|---|---|
| 0 | 1000268201_693b08cb0e.jpg | /content/drive/MyDrive/Images/1000268201_693b0... | A child in a pink dress is climbing up a set o... |
| 1 | 1000268201_693b08cb0e.jpg | /content/drive/MyDrive/Images/1000268201_693b0... | A girl going into a wooden building |
| 2 | 1000268201_693b08cb0e.jpg | /content/drive/MyDrive/Images/1000268201_693b0... | A little girl climbing into a wooden playhouse |
| 3 | 1000268201_693b08cb0e.jpg | /content/drive/MyDrive/Images/1000268201_693b0... | A little girl climbing the stairs to her playh... |
| 4 | 1000268201_693b08cb0e.jpg | /content/drive/MyDrive/Images/1000268201_693b0... | A little girl in a pink dress going into a woo... |
| 5 | 1001773457_577c3a7d70.jpg | /content/drive/MyDrive/Images/1001773457_577c3... | A black dog and a spotted dog are fighting |
| 6 | 1001773457_577c3a7d70.jpg | /content/drive/MyDrive/Images/1001773457_577c3... | A black dog and a tri-colored dog playing with... |
| 7 | 1001773457_577c3a7d70.jpg | /content/drive/MyDrive/Images/1001773457_577c3... | A black dog and a white dog with brown spots a... |
| 8 | 1001773457_577c3a7d70.jpg | /content/drive/MyDrive/Images/1001773457_577c3... | Two dogs of different breeds looking at each o... |
| 9 | 1001773457_577c3a7d70.jpg | /content/drive/MyDrive/Images/1001773457_577c3... | Two dogs on pavement moving toward each other |

```
#check total captions and images present in dataset
print("Total captions present in the dataset: "+ str(len(annotations)))
print("Total images present in the dataset: " + str(len(all_imgs)))
```

```
Total captions present in the dataset: 40455
Total images present in the dataset: 8091
```

## 4. Counting Word Frequencies in Captions

```
vocabulary = [word.lower() for line in annotations for word in
line.split()]

val_count = Counter(vocabulary)
val_count
```

This snippet computes the frequency of each word in the captions:

- Creates a list of lowercase words from all captions.
- Uses collections.Counter to count the occurrences of each word.
- Stores the frequency count in val_count, which is a dictionary-like object mapping words to their counts.


```
Counter({'a': 60196,
         'child': 1507,
         'in': 18174,
         'pink': 702,
         'dress': 332,
         'is': 9069,
         'climbing': 490,
         'up': 1215,
         'set': 105,
         'of': 6495,
         'stairs': 109,
         'an': 2325,
         'entry': 1,
         'way': 48,
         'girl': 3277,
         'going': 145,
         'into': 1046,
         'wooden': 278,
         'building': 485,
         'little': 1736,
         'playhouse': 6,
         'the': 17507,
         'to': 3005,
         'her': 1102,
         'cabin': 4,
         'black': 3620,
         'dog': 7948,
         'and': 8057,
         'spotted': 36,
         'are': 3365,
         'fighting': 130,
         'tri-colored': 12,
         'playing': 1954,
         'with': 7304,
```

## 5. Visualise the top 30 occuring words in the captions

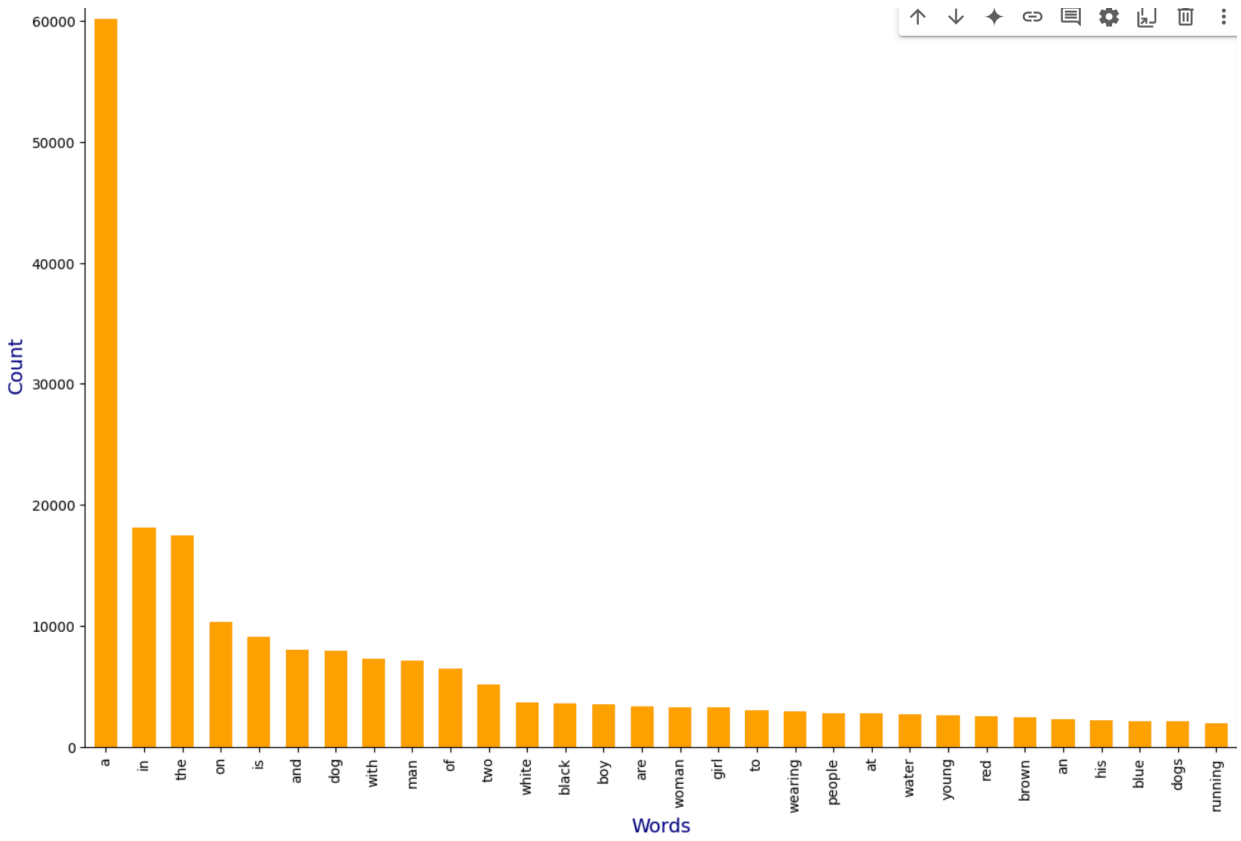
```
for word, count in val_count.most_common(30):
  print(word, ": ", count)

lst = val_count.most_common(30)
most_common_words_df = pd.DataFrame(lst, columns = ['Word', 'Count'])
most_common_words_df.plot.bar(x='Word', y='Count', width=0.6,
color='orange', figsize=(15, 10))
plt.title("Top 30 maximum frequency words", fontsize = 18, color=
'navy')
plt.xlabel("Words", fontsize = 14, color= 'navy')
plt.ylabel("Count", fontsize = 14, color= 'navy')
```

This snippet identifies and visualizes the 30 most frequent words in the captions:

1. **Identify the Most Common Words**:
   - Extracts the top 30 most common words and their counts from val_count using the most_common() method.
   - Prints each word and its count.
2. **Create a DataFrame**:
   - Constructs a DataFrame, most_common_words_df, with columns Word and Count, holding the top 30 words and their frequencies.
3. **Visualize Word Frequencies**:
   - Plots a bar chart of the top 30 words using matplotlib, with orange bars and labeled axes.
   - Sets a descriptive title and labels for the chart, enhancing interpretability.

```
a :  60196
in :  18174
the :  17507
on :  10357
is :  9069
and :  8057
dog :  7948
with :  7304
man :  7137
of :  6495
two :  5132
white :  3706
black :  3620
boy :  3514
are :  3365
woman :  3304
girl :  3277
to :  3005
wearing :  2916
people :  2811
at :  2810
water :  2676
young :  2587
red :  2553
brown :  2457
an :  2325
his :  2255
blue :  2125
dogs :  2095
running :  1996
Text(0, 0.5, 'Count')
```



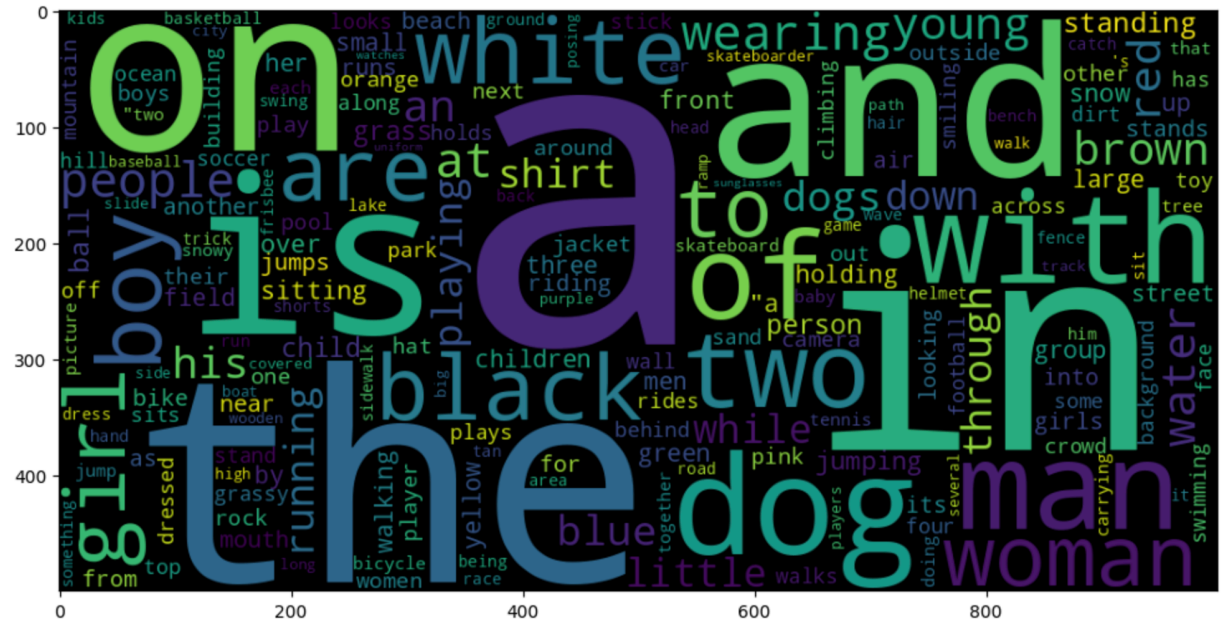
## 6. Generate and Visualize Word Cloud from Word Frequencies:

```
wordcloud = WordCloud(width = 1000, height =
500).generate_from_frequencies(val_count)
plt.figure(figsize = (12, 12))
plt.imshow(wordcloud)
```

This snippet generates and visualizes a word cloud to represent word frequencies in the captions:

1. **Generate Word Cloud**:
   - Creates a WordCloud object with specified width and height dimensions (1000x500).
   - Uses the word frequency dictionary val_count to generate the word cloud, where more frequent words appear larger.
2. **Visualize the Word Cloud**:
   - Displays the generated word cloud as an image using matplotlib (plt.imshow()).

```
<matplotlib.image.AxesImage at 0x79ac0c003010>
```




## 7. Display Multiple Images with Captions

```
def execute_img_capt(start, end, frame) :
  for r in range(start, end) :
    caption_with_img_plot(frame.ID.drop_duplicates().iloc[r], frame)

execute_img_capt(0, 5, df)
```

This function iteratively displays a range of images along with their captions:

1. **Iterate Through Image IDs**:
   - Loops through the range of rows specified by start and end in the DataFrame frame.
2. **Call Caption Display Function**:
   - For each row, retrieves the unique image_id using drop_duplicates() to avoid repetition.
   - Passes the image_id and the DataFrame frame to the caption_with_img_plot() function.
3. **Display Images**:
   - Uses caption_with_img_plot() to display each image and its associated captions one by one.

A child in a pink dress is climbing up a set of stairs in an entry way

A girl going into a wooden building

A little girl climbing into a wooden playhouse

A little girl climbing the stairs to her playhouse

A little girl in a pink dress going into a wooden cabin

## 8. Clean Captions by Removing Punctuation, Numbers, and Formatting

```
#data cleaning
rem_punct = str.maketrans('', '', string.punctuation)
for r in range(len(annotations)) :
  line = annotations[r]
  line = line.split()

  line = [word.lower() for word in line]
```

```
    line = [word.translate(rem_punct) for word in line]
     line = [word for word in line if len(word) > 1]

     line = [word for word in line if word.isalpha()]

     annotations[r] = ' '.join(line)
```

**Initialize Punctuation Removal**:

- Creates a translation table, rem_punct, to remove all punctuation characters.

**Iterate Through Captions**:

- Loops through each caption in the annotations list.

**Steps for Cleaning**:

- **Convert to Lowercase**:
  - Ensures all words are in lowercase for consistency.
- **Remove Punctuation**:
  - Strips punctuation from each word using the translation table.
- **Filter Short Words**:
  - Removes words with a single character to avoid noise.
- **Remove Numeric Values**:
  - Filters out words containing numeric characters.

**Rejoin Cleaned Captions**:

- Joins the cleaned list of words back into a single caption string for each entry.

## 9. Add Start and End Tokens to Captions:

```
annotations = ['<start>' + ' ' + line + ' ' + '<end>' for line in
annotations]
    all_img_path = all_img_vector
```

```
annotations[0:5]
```

```
['<start> child in pink dress is climbing up set of stairs in an entry way <end>',
 '<start> girl going into wooden building <end>',
 '<start> little girl climbing into wooden playhouse <end>',
 '<start> little girl climbing the stairs to her playhouse <end>',
 '<start> little girl in pink dress going into wooden cabin <end>']
```

- For each caption in annotations, it appends a <start> token at the beginning and an <end> token at the end. This formatting is essential for sequence models to identify where the caption begins and ends during training.

## 10.Tokenize Captions and Prepare for Sequence Transformation

```
    top_word_cnt = 5000
    tokenizer = Tokenizer(num_words = top_word_cnt+1, filters=
'!"#$%^&*()_+.,:;-?/~`{}[]|\=@ ',
     lower = True, char_level = False, oov_token = 'UNK')
     tokenizer.fit_on_texts(annotations)
```

```
    train_seqs = tokenizer.texts_to_sequences(annotations)
    tokenizer.word_index['PAD'] = 0
    tokenizer.index_word[0] = 'PAD'
```

**Initialize the Tokenizer**:

- Configures the tokenizer to:
  - Use a vocabulary size of top_word_cnt + 1 (to include a special token for unknown words).
  - Filter out specified special characters and punctuation.
  - Convert all words to lowercase for consistency.
  - Represent out-of-vocabulary (OOV) words with the token "UNK".

**Fit the Tokenizer**:

- Trains the tokenizer on the list of cleaned captions (annotations), building a vocabulary based on the most frequent words.

**Transform Captions to Sequences**:

- Converts each caption into a sequence of integer indices, where each integer corresponds to a word in the vocabulary.

**Add Special Tokens**:

- Assigns PAD (padding) token with index 0 for sequence padding.
- Updates the tokenizer's word_index and index_word mappings to include the PAD token.

```
print(tokenizer.oov_token)
print(tokenizer.index_word[0])
```

```
UNK
PAD
```

## 11.Tokenize Captions and Prepare for Sequence Transformation

```
    train_seqs_len = [len(seq) for seq in train_seqs]
    longest_word_length = max(train_seqs_len)
    cap_vector= tf.keras.preprocessing.sequence.pad_sequences(train_seqs,
padding= 'post', maxlen = longest_word_length,dtype='int32', value=0)
print("The shape of Caption vector is :" + str(cap_vector.shape))
```

This snippet tokenizes the captions, converts them to integer sequences, and prepares for padding.

**Initialize Tokenizer**:

- Creates a tokenizer to handle up to top_word_cnt + 1 words.
- Configures it to lowercase all text and remove specified punctuation and symbols.
- Sets UNK as the token for out-of-vocabulary words.

**Fit Tokenizer**:

- Trains the tokenizer on annotations to build a word-to-index mapping.

**Transform Captions**:

- Converts each caption into a sequence of integers using texts_to_sequences.

**Add Padding Token**:

- Adds a special PAD token with an index of 0 to the tokenizer for padding.

## 12. Image Preprocessing and Visualization

**Purpose**:
This snippet preprocesses the images from the dataset to make them compatible with the InceptionV3 model and visualizes the preprocessed images.

```
for img in all_imgs[0:5] :
    img = tf.io.read_file(img, name=None)
    img = tf.image.decode_jpeg(img, channels=0)
    img = tf.image.resize(img, (299, 299))
    img = tf.keras.applications.inception_v3.preprocess_input(img)

    preprocessed_image.append(img)

Display_Images = preprocessed_image[0:5]
figure, axes = plt.subplots(1,5)
figure.set_figwidth(25)

for ax, image in zip(axes, Display_Images) :
  print('Shape after resize : ', image.shape)
  ax.imshow(image)
  ax.grid('off')
```

- **Image Reading**:
  - Iterates through the first 5 images in all_imgs.
  - Reads each image file as a binary string using tf.io.read_file().

**Image Decoding**:

- Decodes the binary string into a JPEG image tensor using tf.image.decode_jpeg() with the default number of color channels.

**Image Resizing**:

- Resizes each image to (299, 299) dimensions to match the input size required by the InceptionV3 model.

**Preprocessing for Model Input**:

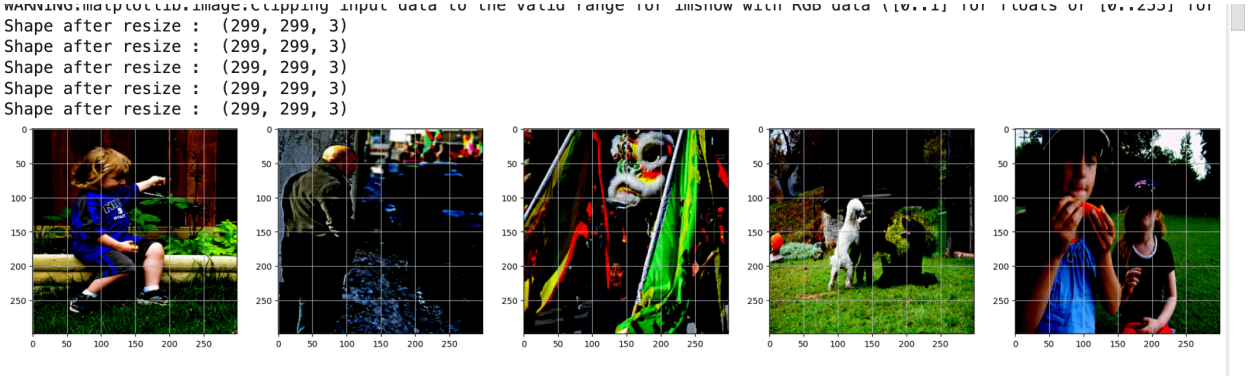
- Applies normalization and other preprocessing steps using tf.keras.applications.inception_v3.preprocess_input() to prepare the image for feature extraction.

**Store Preprocessed Images**:

- Appends the processed image tensors to the preprocessed_image list.

**Visualization**:

- Extracts the first 5 preprocessed images from preprocessed_image for visualization.
- Creates a Matplotlib figure with 5 subplots arranged in a single row.
- Displays each image in its respective subplot and prints the shape of each preprocessed image.




## 13.Image Dataset Creation and Splitting

```
def load_images(image_path) :
  img = tf.io.read_file(image_path, name = None)
  img = tf.image.decode_jpeg(img, channels=0)
  img = tf.image.resize(img, IMAGE_SHAPE)
  img = tf.keras.applications.inception_v3.preprocess_input(img)
  return img, image_path

training_list = sorted(set(all_img_vector))
New_Img = tf.data.Dataset.from_tensor_slices(training_list)
New_Img = New_Img.map(load_images, num_parallel_calls =
tf.data.experimental.AUTOTUNE)
New_Img = New_Img.batch(64, drop_remainder=False)
path_train, path_test, caption_train, caption_test =
train_test_split(all_img_vector, cap_vector, test_size = 0.2, random_state
= 42)
```

## 14.**Load and Preprocess Images**:

- **Function**: load_images(image_path)
  - Reads an image file from the given image_path using tf.io.read_file.
  - Decodes the image from a JPEG file into a tensor using tf.image.decode_jpeg.

- Resizes the image to the predefined dimensions (IMAGE_SHAPE) using tf.image.resize.
- Normalizes the image using tf.keras.applications.inception_v3.preprocess_input to make it compatible with the InceptionV3 model.
- Returns the preprocessed image tensor and its path.

**Create a TensorFlow Dataset**:

- Converts training_list (a sorted list of unique image paths) into a TensorFlow dataset using tf.data.Dataset.from_tensor_slices.
- Applies the load_images function to each image path in the dataset using .map(), enabling parallel preprocessing with tf.data.experimental.AUTOTUNE.

**Batch the Dataset**:

- Groups the dataset into batches of 64 using .batch(64, drop_remainder=False).
- Ensures efficient training by dynamically adjusting the number of parallel calls for preprocessing based on the available hardware (CPU/GPU).

**Split into Training and Testing Sets**:

- Splits the image paths (all_img_vector) and caption vectors (cap_vector) into training and testing datasets using an 80:20 ratio with train_test_split.
- Sets random_state=42 for reproducibility.

## 15. Image Feature Extraction Model

```
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.compat.v1.keras.Model(new_input,
hidden_layer)

image_features_extract_model.summary()
```

**Load Pre-trained InceptionV3**:

- Initializes the **InceptionV3** model from tf.keras.applications with:
    - include_top=False: Excludes the fully connected (classification) layers, retaining only the convolutional base for feature extraction.
    - weights='imagenet': Loads pre-trained weights trained on the ImageNet dataset.

**Retrieve Input and Output Layers**:

- **Input Layer (new_input)**: Extracts the input tensor of the InceptionV3 model to define the input for the new model.
- **Output Layer (hidden_layer)**: Extracts the output of the last convolutional layer (mixed10) of the InceptionV3 model, which provides high-level feature maps for the image.

**Build the Feature Extraction Model**:

- Constructs a new TensorFlow model, image_features_extract_model, using the extracted input and output layers.
- This model takes an image as input and outputs a 4D tensor of feature maps.

**Model Summary**:

- Prints a summary of the new feature extraction model, showing the structure, layers, and output shapes.

Inception V3 Architecture

the Inception V3 model is made of.

| TYPE | PATCH / STRIDE SIZE | INPUT SIZE |
|---|---|---|
| Conv | 3×3/2 | 299×299×3 |
| Conv | 3×3/1 | 149×149×32 |
| Conv padded | 3×3/1 | 147×147×32 |
| Pool | 3×3/2 | 147×147×64 |
| Conv | 3×3/1 | 73×73×64 |
| Conv | 3×3/2 | 71×71×80 |
| Conv | 3×3/1 | 35×35×192 |
| 3 × Inception | Module 1 | 35×35×288 |
| 5 × Inception | Module 2 | 17×17×768 |
| 2 × Inception | Module 3 | 8×8×1280 |
| Pool | 8 × 8 | 8 × 8 × 2048 |
| Linear | Logits | 1 × 1 × 2048 |
| Softmax | Classifier | 1 × 1 × 1000 |

The given code loads the **Inception V3** model with the following configuration:

- **include_top=False**: This excludes the fully connected layers (classification head) at the top of the model, retaining only the convolutional base.
- **weights='imagenet'**: The model is pre-trained on the ImageNet dataset, making it capable of extracting meaningful image features.

The **Inception V3** architecture includes several key components designed for efficient image feature extraction:

### Key Architectural Components:

1. **Input Layer**:
   - The input tensor accepts images of shape (None, None, 3) by default, allowing flexible input sizes (commonly resized to (299, 299, 3) for this model).
2. **Initial Layers**:
   - A series of convolution and pooling layers to downsample the image and extract low-level features such as edges and textures.
3. **Inception Modules**:
   - The hallmark of the Inception V3 architecture is the **Inception Module**, which applies multiple filters (1x1, 3x3, 5x5, etc.) in parallel and concatenates their outputs.
   - These modules capture multi-scale spatial features, improving the model's representational power without increasing computational cost significantly.
4. **Reduction Modules**:
   - Specialized layers for dimensionality reduction using pooling and convolution operations, reducing feature map sizes while preserving essential information.
5. **Auxiliary Classifier Heads** (Excluded Here):
   - Intermediate outputs for regularization during training (not included in this model since include_top=False).
6. **Global Average Pooling Layer** (Excluded Here):
   - Used to generate classification logits by pooling spatial dimensions into a single vector (also excluded here).
7. **Final Output Layer**:
   - The code extracts the output of the last convolutional layer (mixed10) with a tensor shape (None, 8, 8, 2048):
     - **8x8**: Spatial resolution of the feature map.
     - **2048**: Number of feature channels (depth).

### Summary of the Extracted Architecture:

- The loaded **Inception V3 model** contains **11,898,944 trainable parameters**.
- The convolutional base outputs a 4D tensor (batch size, 8, 8, 2048) of spatial feature maps.
- This model serves as the encoder in the image captioning pipeline, effectively encoding high-level visual features of input images.

**Description for Feature Extraction from Images**

```
img_features = {}
for image, image_path in tqdm(New_Img) :
```

```
  batch_features = image_features_extract_model(image)
  batch_features_flattened = tf.reshape(batch_features,
(batch_features.shape[0], -1, batch_features.shape[3]))

  for batch_feat, path in zip(batch_features_flattened, image_path) :
    feature_path = path.numpy().decode('utf-8')
    img_features[feature_path] = batch_feat.numpy()
```

**Purpose**:
This code extracts high-level feature representations from images using the **Inception V3** feature extraction model (image_features_extract_model) and stores the results in a dictionary (img_features), where each image's features are indexed by its file path.

---

## Steps Explained:

1. **Initialize Feature Storage**:
   - Creates an empty dictionary img_features to store extracted features for each image, indexed by its file path.
2. **Iterate Over Image Dataset**:
   - Loops through batches of images and their corresponding file paths in New_Img using tqdm for progress tracking.
   - New_Img contains preprocessed and batched images.
3. **Feature Extraction**:
   - Passes each batch of images to the pre-trained **Inception V3** model (image_features_extract_model) to extract feature maps.
   - The model outputs a 4D tensor of feature maps with dimensions (batch_size, 8, 8, 2048).
4. **Reshape Features**:
   - Flattens the spatial dimensions (8x8) into a single dimension using tf.reshape, resulting in a 3D tensor of shape (batch_size, 64, 2048):
     - 64 corresponds to the total spatial regions (8x8).
     - 2048 represents the feature channels.
5. **Store Features**:
   - Iterates through the flattened features (batch_feat) and corresponding image paths (path):
     - Converts the image path tensor to a string using numpy().decode('utf-8').
     - Stores the flattened feature vector (batch_feat.numpy()) in the img_features dictionary, indexed by the file path.

```
100%|██████████| 127/127 [28:21<00:00, 13.40s/it]
```

```
batch_features
```

```
          [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
           0.00000000e+00, 0.00000000e+00, 1.70470846e+00],
          ...,
          [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
           0.00000000e+00, 2.00714803e+00, 1.98217130e+00],
          [0.00000000e+00, 1.01070476e+00, 0.00000000e+00, ...,
           9.11544412e-02, 1.17547202e+00, 9.26892161e-01],
          [0.00000000e+00, 0.00000000e+00, 3.08890790e-01, ...,
           0.00000000e+00, 4.88666147e-01, 2.04599887e-01]],

         [[0.00000000e+00, 2.22088128e-01, 0.00000000e+00, ...,
           0.00000000e+00, 0.00000000e+00, 1.06316447e+00],
          [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
           0.00000000e+00, 0.00000000e+00, 1.12167799e+00],
          [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
           0.00000000e+00, 0.00000000e+00, 1.59463489e+00],
          ...,
          [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
           2.67917931e-01, 2.56294203e+00, 2.25518560e+00],
          [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
           1.27264276e-01, 1.82892597e+00, 1.24883318e+00],
          [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
           0.00000000e+00, 1.39449489e+00, 7.87283540e-01]],
```

## Description of Code for Mapping and Dataset Generation

```
def map(image_name, caption):

    img_tensor = img_features[image_name.decode('utf-8')]
    return img_tensor, caption

BUFFER_SIZE = 1000
BATCH_SIZE = 64

def gen_dataset(img, capt):
    data = tf.data.Dataset.from_tensor_slices((img, capt))
    data = data.map(lambda ele1, ele2 : tf.numpy_function(map, [ele1,
ele2], [tf.float32, tf.int32]),
                    num_parallel_calls = tf.data.experimental.AUTOTUNE)
```

```
    data = (data.shuffle(BUFFER_SIZE, reshuffle_each_iteration=
True).batch(BATCH_SIZE, drop_remainder =
False).prefetch(tf.data.experimental.AUTOTUNE))
    return data
```

This code defines functions to map pre-extracted image features to corresponding captions and generate an optimized TensorFlow dataset for training or evaluation.

1. map Function

**Purpose**:

The map function retrieves precomputed image features for a given image and pairs them with its corresponding caption.

**Steps**:

1. **Image Feature Lookup**:
    - Takes an image name (image_name) and decodes it to a string using .decode('utf-8').
    - Retrieves the corresponding feature vector from the precomputed img_features dictionary.
2. **Return Pair**:
    - Returns the image feature tensor (img_tensor) and its associated caption.

This function establishes a mapping between image features and captions.

2. gen_dataset Function

**Purpose**:

This function creates a TensorFlow dataset from images and captions, applying mapping, shuffling, batching, and prefetching for efficient data loading during training.

**Steps**:

1. **Create Dataset**:
    - Uses tf.data.Dataset.from_tensor_slices to create a dataset from img (image paths) and capt (captions).
2. **Apply Mapping**:
    - Maps the map function to each dataset element using tf.numpy_function, converting image paths to tensors of image features and pairing them with their captions.
    - Enables parallel processing with num_parallel_calls=tf.data.experimental.AUTOTUNE for improved performance.
3. **Shuffle Dataset**:
    - Randomly shuffles the dataset with a buffer size of BUFFER_SIZE=1000 for randomness in each epoch.
4. **Batch Dataset**:
    - Groups data into batches of size BATCH_SIZE=64, allowing efficient batch processing. The last batch is not dropped (drop_remainder=False).
5. **Prefetch Dataset**:
    - Prepares the next batch while the current one is being processed using .prefetch(tf.data.experimental.AUTOTUNE) to optimize data loading.

6. **Return Dataset**:
    - Returns the processed TensorFlow dataset for training or evaluation.

## Encoder Model

```
embedding_dim = 256
units = 512

vocab_size = 5001
train_num_steps = len(path_train) // BATCH_SIZE
test_num_steps = len(path_test) // BATCH_SIZE

max_length = 31
feature_shape = batch_feat.shape[1]
attention_feature_shape = batch_feat.shape[0]

class Encoder(Model):
    def __init__(self,embed_dim):
        super(Encoder, self).__init__()
        self.dense = tf.keras.layers.Dense(embed_dim)

    def call(self, features):
        features =  self.dense(features)
        features =  tf.keras.activations.relu(features, alpha=0.01,
max_value=None, threshold=0)
        return features

encoder=Encoder(embedding_dim)
```

This code defines and initializes an **Encoder** as part of an image captioning model pipeline, which converts image features into a dense representation suitable for further processing by the decoder.

### Key Components and Steps

#### *1. Parameter Initialization*

- **embedding_dim**:
    - Dimension of the dense embedding space (256).
- **units**:
    - Number of units in the subsequent recurrent layers of the model (512).
- **vocab_size**:
    - Vocabulary size for captions (5,001).

- **train_num_steps and test_num_steps**:
  - Number of steps for training and testing datasets, calculated by dividing the number of image paths by the batch size (BATCH_SIZE).
- **max_length**:
  - Maximum length of caption sequences (31 tokens).
- **feature_shape**:
  - Shape of extracted image features along the feature dimension (inferred from the batch features).
- **attention_feature_shape**:
  - Shape of extracted image features along the spatial dimensions for attention mechanisms.

### *2. Encoder Class*

The Encoder class processes the extracted image features and maps them into an embedding space.

- **Inheritance**:
  - Subclasses tf.keras.Model to build a custom trainable layer.
- **Initialization (__init__)**:
  - Defines a Dense layer with embed_dim=256 units and ReLU activation to transform the feature vectors.
- **Forward Pass (call)**:
  - Receives image features as input (features).
  - Applies the Dense layer to project the features into the embedding space.
  - Uses tf.keras.activations.relu to introduce non-linearity in the output.

### *3. Encoder Initialization*

- An instance of the Encoder class is created with the embedding dimension (embedding_dim=256), allowing it to process image features.

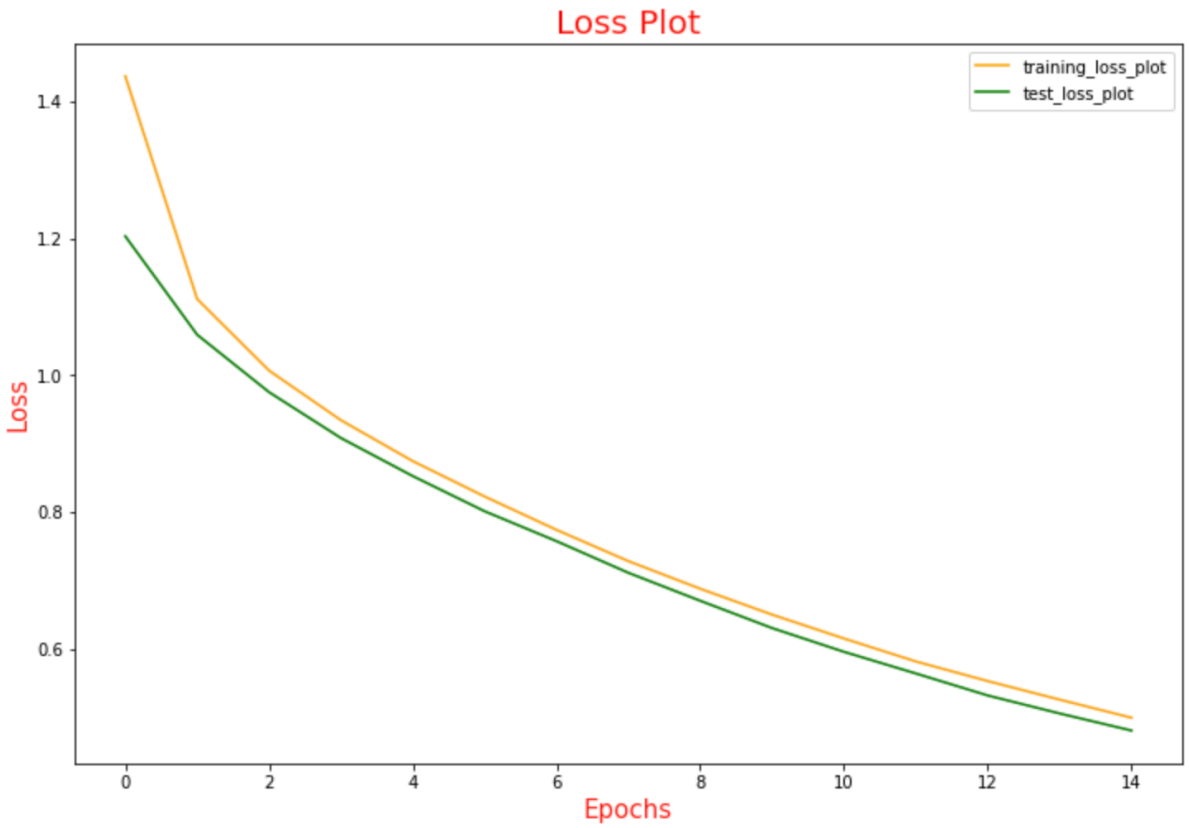
## Outcome

The **Encoder** maps image features from the InceptionV3 feature extractor (shape: (batch_size, 8*8, 2048)) into a lower-dimensional dense space (shape: (batch_size, 8*8, 256)). This output is then passed to the decoder for generating captions. My teammates implemented the decoder and the attention mechanism to generate captions dynamically. The decoder, built using GRU cells, processed the encoded features along with previously generated words to predict the next word in the sequence. The attention mechanism allowed the model to focus on specific regions of the image at each timestep, enhancing the contextual relevance of the generated captions. For training, the  teacher forcing mechanism was employed, where the true word from the dataset was fed as input for the next timestep during training, stabilizing the learning process. This approach ensured faster convergence and minimized compounding errors in sequence generation. The results demonstrated the model's capability to generate descriptive and

contextually accurate captions, supported by robust loss curves and qualitative observations that aligned well with reference captions.

**Loss Plot Description.**

This is the loss plot obtained after training an **Encoder-Decoder model with Attention Mechanism**, implemented collaboratively with my teammates. Below are the key observations:

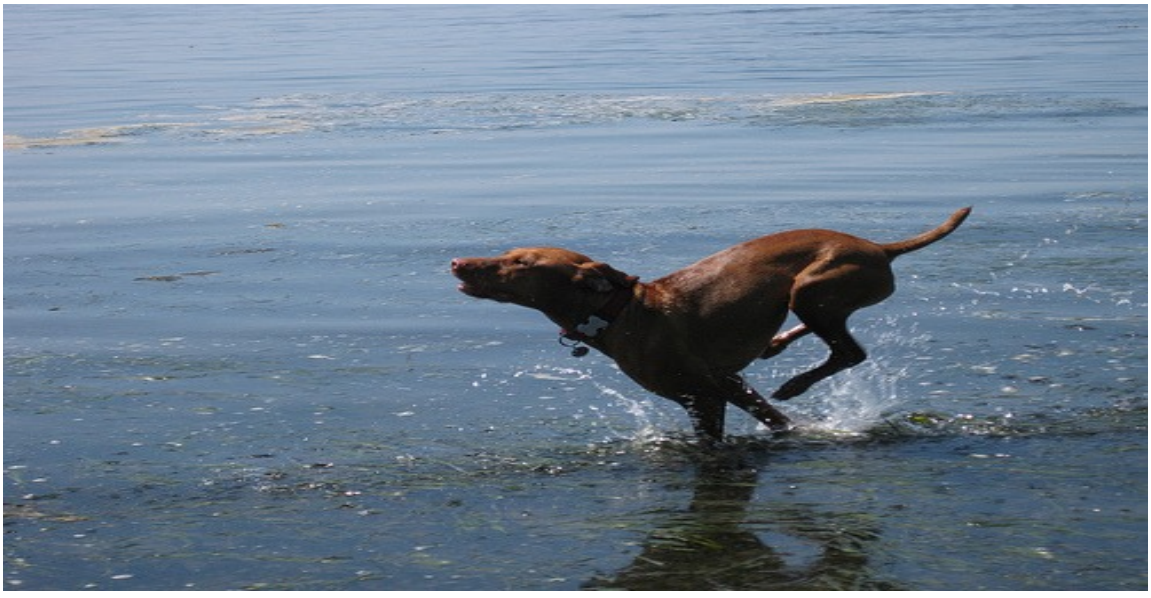



1. **Training and Test Loss Trends**:
   - Both training loss (orange) and test loss (green) decrease steadily over 14 epochs.
   - This indicates that the model is learning effectively without signs of overfitting.
2. **Narrowing Gap Between Loss Curves**:
   - The small gap between training and test loss demonstrates good generalization, as the model performs similarly on unseen data.
3. **Convergence Behavior**:
   - The gradual and smooth decline in both curves shows stable training and suggests that further training could further reduce loss.

## Evaluation Metric (BLEU Score):

**BLEU score** was used as the evaluation metric to assess the quality of the generated captions. The BLEU score measures the similarity between the predicted captions and the ground-truth captions by comparing n-grams, providing an effective way to evaluate the model's performance in image captioning tasks.

```
BELU score: 63.245553203367585
Real Caption: many birds sit on ledge
Prediction Caption: group of birds sit on ledge
```



```
BELU score: 65.14390575310556
Real Caption: brown dog is running in the water by the beach
Prediction Caption: brown dog is running in the water
```

```
BELU score: 55.04959249310236
Real Caption: man wearing backpack and helmet riding bicycle
Prediction Caption: woman wearing backpack and helmet and cycling gear
```

## Summary of Results

1. **Image Preprocessing and Feature Extraction**:
    - Successfully preprocessed the images using InceptionV3, extracting high-dimensional feature vectors.
    - These features were mapped to lower-dimensional embeddings using a custom Encoder, ensuring efficient input to the decoder.
2. **Caption Tokenization and Padding**:
    - Processed captions by cleaning, tokenizing, and padding them, creating a structured format suitable for model training.
    - Handled vocabulary limitations with out-of-vocabulary (OOV) tokens and padding for sequence consistency.
3. **Dataset Creation**:
    - Built an efficient TensorFlow pipeline for mapping image features to captions with dynamic batching, shuffling, and prefetching, ensuring smooth training workflows.
4. **Encoder Implementation**:
    - Designed and implemented a dense embedding layer for image feature transformation, enabling the use of attention mechanisms in the decoder.

## What I Have Learned

1. **Technical Insights**:
   - **Data Processing**: Gained hands-on experience in image preprocessing, feature extraction, and efficient dataset preparation using TensorFlow pipelines.
   - **NLP Techniques**: Improved understanding of text cleaning, tokenization, and sequence padding for natural language processing tasks.
   - **Deep Learning**: Learned how to implement and integrate custom encoder models into a larger architecture for image captioning.
2. **Modeling Concepts**:
   - **Feature Extraction**: Understood how pre-trained models (like InceptionV3) can be adapted to new tasks by repurposing their convolutional layers.
   - **Encoder-Decoder Framework**: Developed insights into encoder-decoder models and how dense embeddings and attention mechanisms facilitate learning.
3. **Workflow Optimization**:
   - Discovered the importance of batch processing, shuffling, and prefetching in optimizing training pipelines for deep learning.

## Suggestions for Future Improvements

1. **Model Enhancements**:
   - **Bidirectional Language Models**: Experiment with pre-trained language models (e.g., GPT, BERT) for enhanced natural language generation.
   - **Fine-Tuning**: Fine-tune the InceptionV3 encoder on a domain-specific dataset to improve feature extraction for specific types of images.
2. **Data Augmentation**:
   - Apply data augmentation techniques (e.g., rotation, flipping, cropping) to increase the diversity of the image dataset, potentially improving generalization.
3. **Larger Datasets**:
   - Train on larger datasets like Flickr30k or MS COCO to improve model robustness and the diversity of generated captions.

Appendix:

```
#Import all the required libraries


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter

import tensorflow as tf
```

```
import keras
from keras.preprocessing.image import load_img
import string
import time
from sklearn.model_selection import train_test_split
from keras_preprocessing.text import Tokenizer

from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.models import Model
from tensorflow.keras import layers
from tensorflow.keras import activations
from tensorflow.keras import Input
from PIL import Image

#used for creating Progress Meters or Progress Bars
from tqdm import tqdm
import glob
from gtts import gTTS
from playsound import playsound
from IPython import display

import collections
import wordcloud
from wordcloud import WordCloud, STOPWORDS

import numpy as np
import pandas as pd

import imageio

#Visualising first 5 images :
images='/content/drive/MyDrive/Images'

all_imgs = glob.glob(images + '/*.jpg',recursive=True)
print("The total images present in the dataset: {}".format(len(all_imgs)))

Display_Images = all_imgs[0:5]
figure, axes = plt.subplots(1,5)
figure.set_figwidth(20)

for ax, image in zip(axes, Display_Images):
  ax.imshow(imageio.imread(image), cmap=None)

text_file = '/content/drive/MyDrive/captions.txt'

def load_doc(filename):

    #your code here
```

```
    open_file = open(text_file, 'r', encoding='latin-1' ) #returns a file object
    text = open_file.read() #reads contents of the file
    open_file.close()
    #print(text)

    return text

doc = load_doc(text_file)
print(doc[:300])

img_path = '/content/drive/MyDrive/Images/'

all_img_id = [] #store all the image id here
all_img_vector = [] #store all the image path here
annotations = [] #store all the captions here

with open('/content/drive/MyDrive/captions.txt' , 'r') as fo:
  next(fo) #to skip the heading
  for line in fo :
    split_arr = line.split(',')
    all_img_id.append(split_arr[0])
    annotations.append(split_arr[1].rstrip('\n.')) #removing out the \n.
    all_img_vector.append(img_path+split_arr[0])

df = pd.DataFrame(list(zip(all_img_id, all_img_vector,annotations)),columns
=['ID','Path', 'Captions'])

df.head(10)

print("Total captions present in the dataset: "+ str(len(annotations)))
print("Total images present in the dataset: " + str(len(all_imgs)))

vocabulary = [word.lower() for line in annotations for word in line.split()]

val_count = Counter(vocabulary)

for word, count in val_count.most_common(30):
  print(word, ": ", count)

lst = val_count.most_common(30)
most_common_words_df = pd.DataFrame(lst, columns = ['Word', 'Count'])
most_common_words_df.plot.bar(x='Word', y='Count', width=0.6, color='orange',
figsize=(15, 10))
plt.title("Top 30 maximum frequency words", fontsize = 18, color= 'navy')
plt.xlabel("Words", fontsize = 14, color= 'navy')
plt.ylabel("Count", fontsize = 14, color= 'navy')

wordcloud = WordCloud(width = 1000, height = 500).generate_from_frequencies(val_count)
```

```
plt.figure(figsize = (12, 12))
plt.imshow(wordcloud)

def caption_with_img_plot(image_id, frame) :
  #get the captions
  capt = ("\n" *2).join(frame[frame['ID'] == image_id].Captions.to_list())
  fig, ax = plt.subplots()
  ax.set_axis_off()
  idx = df.ID.to_list().index(image_id)
  im =  Image.open(df.Path.iloc[idx])
  w, h = im.size[0], im.size[-1]
  ax.imshow(im)
  ax.text(w+50, h, capt, fontsize = 18, color = 'navy')
caption_with_img_plot(df.ID.iloc[8049], df)

def execute_img_capt(start, end, frame) :
  for r in range(start, end) :
    caption_with_img_plot(frame.ID.drop_duplicates().iloc[r], frame)

execute_img_capt(0, 5, df)

rem_punct = str.maketrans('', '', string.punctuation)
for r in range(len(annotations)) :
  line = annotations[r]
  line = line.split()

  # converting to lowercase
  line = [word.lower() for word in line]

  # remove punctuation from each caption and hanging letters
  line = [word.translate(rem_punct) for word in line]
  line = [word for word in line if len(word) > 1]

  # remove numeric values
  line = [word for word in line if word.isalpha()]

  annotations[r] = ' '.join(line)

  annotations = ['<start>' + ' ' + line + ' ' + '<end>' for line in annotations]

#Create a list which contains all the path to the images
all_img_path = all_img_vector


top_word_cnt = 5000
tokenizer = Tokenizer(num_words = top_word_cnt+1, filters= '!"#$%^&*()_+.,:;-
?/~`{}[]|\=@ ',
                      lower = True, char_level = False,
```

```
                      oov_token = 'UNK')

tokenizer.fit_on_texts(annotations)

#transform each text into a sequence of integers
train_seqs = tokenizer.texts_to_sequences(annotations)
tokenizer.word_index['PAD'] = 0
tokenizer.index_word[0] = 'PAD'

print(tokenizer.oov_token)
print(tokenizer.index_word[0])

tokenizer_top_words = [word for line in annotations for word in line.split() ]

#tokenizer_top_words_count
tokenizer_top_words_count = collections.Counter(tokenizer_top_words)
tokenizer_top_words_count

for word, count in tokenizer_top_words_count.most_common(30) :
  print(word, ": ", count)

tokens = tokenizer_top_words_count.most_common(30)
most_com_words_df = pd.DataFrame(tokens, columns = ['Word', 'Count'])

#plot 30 most common words
most_common_words_df.plot.bar(x = 'Word', y= 'Count', width=0.8, color = 'indigo',
figsize = (17, 10))
plt.title('Top 30 common words', fontsize =20, color= 'navy')
plt.xlabel('Words', fontsize =14, color= 'navy')
plt.ylabel('Counts', fontsize =14, color= 'navy')
plt.grid(True)

wordcloud_token = WordCloud(width = 1000, height =
500).generate_from_frequencies(tokenizer_top_words_count)
plt.figure(figsize = (12, 8))
plt.imshow(wordcloud_token)
plt.grid(True)

train_seqs_len = [len(seq) for seq in train_seqs]

#store elements from list with maximum value
longest_word_length = max(train_seqs_len)

#calculate longest word_length and pads all sequences to equal length as that of the
longest.
cap_vector= tf.keras.preprocessing.sequence.pad_sequences(train_seqs, padding= 'post',
maxlen = longest_word_length,
                                                          dtype='int32', value=0)
```

```
print("The shape of Caption vector is :" + str(cap_vector.shape))

preprocessed_image = []
IMAGE_SHAPE = (299, 299)

for img in all_imgs[0:5] :
    img = tf.io.read_file(img, name=None)

    # we need to decode jpeg encoded images (here by default channels = 0)
    img = tf.image.decode_jpeg(img, channels=0)
    img = tf.image.resize(img, (299, 299))
    img = tf.keras.applications.inception_v3.preprocess_input(img)

    #append preprocessed images to the list
    preprocessed_image.append(img)

Display_Images = preprocessed_image[0:5]
figure, axes = plt.subplots(1,5)
figure.set_figwidth(25)

for ax, image in zip(axes, Display_Images) :
  print('Shape after resize : ', image.shape)
  ax.imshow(image)
  ax.grid('off')

def load_images(image_path) :
  img = tf.io.read_file(image_path, name = None)
  img = tf.image.decode_jpeg(img, channels=0)
  img = tf.image.resize(img, IMAGE_SHAPE)
  img = tf.keras.applications.inception_v3.preprocess_input(img)
  return img, image_path

training_list = sorted(set(all_img_vector))
New_Img = tf.data.Dataset.from_tensor_slices(training_list)
New_Img = New_Img.map(load_images, num_parallel_calls = tf.data.experimental.AUTOTUNE)
New_Img = New_Img.batch(64, drop_remainder=False)

path_train, path_test, caption_train, caption_test = train_test_split(all_img_vector,
cap_vector, test_size = 0.2, random_state = 42)
print("Training data for images: " + str(len(path_train)))
print("Testing data for images: " + str(len(path_test)))
print("Training data for Captions: " + str(len(caption_train)))
print("Testing data for Captions: " + str(len(caption_test)))

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.compat.v1.keras.Model(new_input, hidden_layer)
image_features_extract_model.summary()

img_features = {}
for image, image_path in tqdm(New_Img) :
  batch_features = image_features_extract_model(image)
  batch_features_flattened = tf.reshape(batch_features, (batch_features.shape[0], -1,
batch_features.shape[3]))

  for batch_feat, path in zip(batch_features_flattened, image_path) :
    feature_path = path.numpy().decode('utf-8')
    img_features[feature_path] = batch_feat.numpy()

def map(image_name, caption):

    img_tensor = img_features[image_name.decode('utf-8')]
    return img_tensor, caption

BUFFER_SIZE = 1000
BATCH_SIZE = 64

def gen_dataset(img, capt):

    data = tf.data.Dataset.from_tensor_slices((img, capt))
    data = data.map(lambda ele1, ele2 : tf.numpy_function(map, [ele1, ele2],
[tf.float32, tf.int32]),
                    num_parallel_calls = tf.data.experimental.AUTOTUNE)


    data = (data.shuffle(BUFFER_SIZE, reshuffle_each_iteration=
True).batch(BATCH_SIZE, drop_remainder =
False).prefetch(tf.data.experimental.AUTOTUNE))
    return data

train_dataset = gen_dataset(path_train,caption_train)
test_dataset = gen_dataset(path_test,caption_test)

sample_img_batch, sample_cap_batch = next(iter(train_dataset))
print(sample_img_batch.shape)  #(batch_size, 8*8, 2048)
print(sample_cap_batch.shape) #(batch_size,max_len)

embedding_dim = 256
units = 512

#top 5,000 words +1
vocab_size = 5001
```

```
train_num_steps = len(path_train) // BATCH_SIZE #len(total train images) // BATCH_SIZE
test_num_steps = len(path_test) // BATCH_SIZE  #len(total test images) // BATCH_SIZE

max_length = 31
feature_shape = batch_feat.shape[1]
attention_feature_shape = batch_feat.shape[0]

class Encoder(Model):
    def __init__(self,embed_dim):
        super(Encoder, self).__init__()
        self.dense = tf.keras.layers.Dense(embed_dim) #build your Dense layer with
relu activation

    def call(self, features):
        features =  self.dense(features) # extract the features from the image shape:
(batch, 8*8, embed_dim)
        features =  tf.keras.activations.relu(features, alpha=0.01, max_value=None,
threshold=0)
        return features

encoder=Encoder(embedding_dim)
```

References:


https://journalofbigdata.springeropen.com/articles/10.1186/s40537-022-00571-w

https://ieeexplore.ieee.org/document/10127293

https://medium.com/@raman.shinde15/image-captioning-with-flickr8k-dataset-bleu-4bcba0b52926

https://www.youtube.com/watch?v=Uc5eERhIYzc

https://medium.com/@zeyneptufekci.etu/cnn-lstm-for-image-captioning-with-progressive-loading-52a740705b2c