Introduction to Image Captioning

Image Captioning is a fascinating way for computers to describe images using words. Similar to how we glance at a picture and understand what's happening, computers can learn to do the same!

Imagine showing a computer an image of an adorable cat. Image Captioning is like magic that makes the computer say something like, "A fluffy white cat is sitting on a windowsill." It's a beautiful blend of teaching computers to comprehend images and communicate using human-like language.

How It Works

Think of Image Captioning as a collaboration between two essential components of the computer's brain:

1. **The Eye (Convolutional Neural Networks - CNNs):** Just as we have eyes to see, computers have CNNs to analyze pictures. These networks help the computer identify important elements in the image, such as the cat's ears or tail. These key elements are translated into a special set of numbers that the computer understands. These special numbers are called "vector embeddings."

2. **The Mouth (Recurrent Neural Networks - RNNs):** The computer's "mouth" is the RNN. It takes those special numbers (vector embeddings) from the CNN and combines them with the power of words. It's as if we're teaching the computer to narrate a story about the image. The RNN takes one word at a time and starts forming a sentence. It begins with "Fluffy," followed by "white," and so on, until a complete description is created.

Why It's Fascinating

Image Captioning empowers computers to describe images just like humans do. This enhances computers' image understanding capabilities and enables them to communicate using descriptive language. If you're curious to see how this collaboration looks in visual representation, check out the Modelling section!

Import necessary modules

```
# Basic libraries
import os
import pickle
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import warnings
warnings.filterwarnings('ignore')
from math import ceil
from collections import defaultdict
from tqdm.notebook import tqdm        # Progress bar library for
Jupyter Notebook

# Deep learning framework for building and training models
import tensorflow as tf
## Pre-trained model for image feature extraction
from tensorflow.keras.applications.vgg16 import VGG16,
preprocess_input
from tensorflow.keras.preprocessing.image import load_img,
img_to_array

## Tokenizer class for captions tokenization
from tensorflow.keras.preprocessing.text import Tokenizer

## Function for padding sequences to a specific length
from tensorflow.keras.preprocessing.sequence import pad_sequences

## Class for defining Keras models
from tensorflow.keras.models import Model
from tensorflow.keras.utils import to_categorical, plot_model
from tensorflow.keras.layers import Input, Dense, LSTM, Embedding,
Dropout, concatenate, Bidirectional, Dot, Activation, RepeatVector,
```

```
Multiply, Lambda

# For checking score
from nltk.translate.bleu_score import corpus_bleu

# Setting the input and output directory
INPUT_DIR = '/kaggle/input/flickr8k'
OUTPUT_DIR = '/kaggle/working'
```

Image Features Extraction

When it comes to understanding images, we need a helping hand from specialized models. Here's where the pre-trained VGG16 model steps in. This model is like a superhero for extracting important details from images, helping us understand what's happening.

Why VGG16?

VGG16 is popular because it has a knack for extracting both simple and complex features from images. Think of it as an image interpreter. It can tell us about the cat's pointy ears, its fluffy fur, and even the windowsill it's sitting on.

With the image features extracted by VGG16, we'll be able to merge the world of images and words, creating meaningful captions that describe the pictures as if the computer were telling a story.

So, let's harness the power of VGG16 for our image feature extraction!

```
# We are going to use pretraind vgg model
# Load the vgg16 model
model = VGG16()
```

```
# Restructuring the model to remove the last classification layer,
this will give us access to the output features of the model
model = Model(inputs=model.inputs, outputs=model.layers[-2].output)

# Printing the model summary
print(model.summary())
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels.h5
553467096/553467096 [==============================] - 3s 0us/step
Model: "model"
_________________________________________________________________
 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: 134,260,544
Trainable params: 134,260,544
Non-trainable params: 0
_________________________________________________________________
None

# Initialize an empty dictionary to store image features
image_features = {}

# Define the directory path where images are located
img_dir = os.path.join(INPUT_DIR, 'Images')

# Loop through each image in the directory
for img_name in tqdm(os.listdir(img_dir)):
    # Load the image from file
    img_path = os.path.join(img_dir, img_name)
    image = load_img(img_path, target_size=(224, 224))
    # Convert image pixels to a numpy array
    image = img_to_array(image)
    # Reshape the data for the model
    image = image.reshape((1, image.shape[0], image.shape[1],
image.shape[2]))
    # Preprocess the image for ResNet50
    image = preprocess_input(image)
    # Extract features using the pre-trained ResNet50 model
    image_feature = model.predict(image, verbose=0)
    # Get the image ID by removing the file extension
    image_id = img_name.split('.')[0]
    # Store the extracted feature in the dictionary with the image ID
as the key
    image_features[image_id] = image_feature

{"model_id":"3f431ae60ec9425da8aed82d25d2ed81","version_major":2,"vers
ion_minor":0}

# Store the image features in pickle
pickle.dump(image_features, open(os.path.join(OUTPUT_DIR,
'img_features.pkl'), 'wb'))

# Load features from pickle file
pickle_file_path = os.path.join(OUTPUT_DIR, 'img_features.pkl')
```

```
with open(pickle_file_path, 'rb') as file:
    loaded_features = pickle.load(file)
```

Loading Caption Data

```
with open(os.path.join(INPUT_DIR, 'captions.txt'), 'r') as file:
    next(file)
    captions_doc = file.read()

# Create mapping of image to captions
image_to_captions_mapping = defaultdict(list)

# Process lines from captions_doc
for line in tqdm(captions_doc.split('\n')):
    # Split the line by comma(,)
    tokens = line.split(',')
    if len(tokens) < 2:
        continue
    image_id, *captions = tokens
    # Remove extension from image ID
    image_id = image_id.split('.')[0]
    # Convert captions list to string
    caption = " ".join(captions)
    # Store the caption using defaultdict
    image_to_captions_mapping[image_id].append(caption)

# Print the total number of captions
total_captions = sum(len(captions) for captions in
image_to_captions_mapping.values())
print("Total number of captions:", total_captions)
```

{"model_id":"37dbdb3c69f04a10b405704f3e3e609f","version_major":2,"version_minor":0}

```
Total number of captions: 40455
```

So there are total 40455 captions for 8091 images that's means there are 5 captions for each image

Preprocessing Captions: Getting Them Ready

Before we dive into the exciting world of creating captions for images, we need to prepare our captions so that our models can understand them. This process is known as preprocessing.

```
# Function for processing the captions
def clean(mapping):
```

```
    for key, captions in mapping.items():
        for i in range(len(captions)):
            # Take one caption at a time
            caption = captions[i]
            # Preprocessing steps
            # Convert to lowercase
            caption = caption.lower()
            # Remove non-alphabetical characters
            caption = ''.join(char for char in caption if
char.isalpha() or char.isspace())
            # Remove extra spaces
            caption = caption.replace('\s+', ' ')
            # Add unique start and end tokens to the caption
            caption = 'startseq ' + ' '.join([word for word in
caption.split() if len(word) > 1]) + ' endseq'
            captions[i] = caption

# before preprocess of text
image_to_captions_mapping['1026685415_0431cbf574']
```

```
['A black dog carries a green toy in his mouth as he walks through the
grass .',
 'A black dog carrying something through the grass .',
 'A black dog has a blue toy in its mouth .',
 'A dog in grass with a blue item in his mouth .',
 'A wet black dog is carrying a green toy through the grass .']
```

```
# preprocess the text
clean(image_to_captions_mapping)

# after preprocess of text
image_to_captions_mapping['1026685415_0431cbf574']
```

```
['startseq black dog carries green toy in his mouth as he walks
through the grass endseq',
 'startseq black dog carrying something through the grass endseq',
 'startseq black dog has blue toy in its mouth endseq',
 'startseq dog in grass with blue item in his mouth endseq',
 'startseq wet black dog is carrying green toy through the grass
endseq']
```

```
# Creating a List of All Captions
all_captions = [caption for captions in
image_to_captions_mapping.values() for caption in captions]

all_captions[:10]
```

```
['startseq child in pink dress is climbing up set of stairs in an
entry way endseq',
 'startseq girl going into wooden building endseq',
 'startseq little girl climbing into wooden playhouse endseq',
```

```
 'startseq little girl climbing the stairs to her playhouse endseq',
 'startseq little girl in pink dress going into wooden cabin endseq',
 'startseq black dog and spotted dog are fighting endseq',
 'startseq black dog and tricolored dog playing with each other on the
road endseq',
 'startseq black dog and white dog with brown spots are staring at
each other in the street endseq',
 'startseq two dogs of different breeds looking at each other on the
road endseq',
 'startseq two dogs on pavement moving toward each other endseq']

# Tokenizing the Text
tokenizer = Tokenizer()
tokenizer.fit_on_texts(all_captions)

# Save the tokenizer
with open('tokenizer.pkl', 'wb') as tokenizer_file:
    pickle.dump(tokenizer, tokenizer_file)

# Load the tokenizer
with open('tokenizer.pkl', 'rb') as tokenizer_file:
    tokenizer = pickle.load(tokenizer_file)

# Calculate maximum caption length
max_caption_length = max(len(tokenizer.texts_to_sequences([caption])
[0]) for caption in all_captions)
vocab_size = len(tokenizer.word_index) + 1

# Print the results
print("Vocabulary Size:", vocab_size)
print("Maximum Caption Length:", max_caption_length)
```

```
Vocabulary Size: 8768
Maximum Caption Length: 34
```

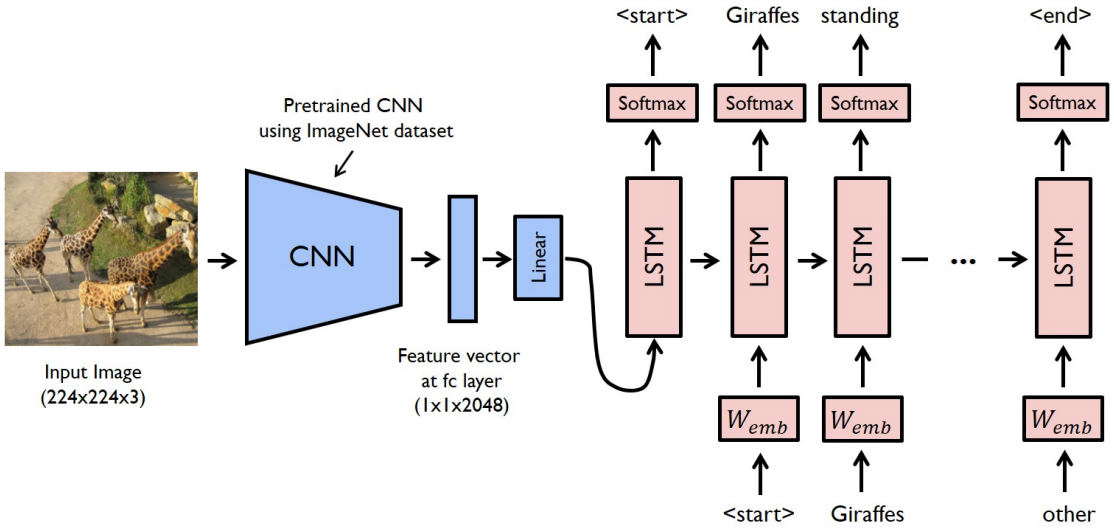
Train Test Split

```
# Creating a List of Image IDs
image_ids = list(image_to_captions_mapping.keys())
# Splitting into Training and Test Sets
split = int(len(image_ids) * 0.90)
train = image_ids[:split]
test = image_ids[split:]

# Data generator function
def data_generator(data_keys, image_to_captions_mapping, features,
tokenizer, max_caption_length, vocab_size, batch_size):
    # Lists to store batch data
    X1_batch, X2_batch, y_batch = [], [], []
```

```
    # Counter for the current batch size
    batch_count = 0

    while True:
        # Loop through each image in the current batch
        for image_id in data_keys:
            # Get the captions associated with the current image
            captions = image_to_captions_mapping[image_id]

            # Loop through each caption for the current image
            for caption in captions:
                # Convert the caption to a sequence of token IDs
                caption_seq = tokenizer.texts_to_sequences([caption])
[0]

                # Loop through the tokens in the caption sequence
                for i in range(1, len(caption_seq)):
                    # Split the sequence into input and output pairs
                    in_seq, out_seq = caption_seq[:i], caption_seq[i]

                    # Pad the input sequence to the specified maximum
caption length
                    in_seq = pad_sequences([in_seq],
maxlen=max_caption_length)[0]

                    # Convert the output sequence to one-hot encoded
format
                    out_seq = to_categorical([out_seq],
num_classes=vocab_size)[0]

                    # Append data to batch lists
                    X1_batch.append(features[image_id][0])  # Image
features
                    X2_batch.append(in_seq)  # Input sequence
                    y_batch.append(out_seq)  # Output sequence

                    # Increase the batch counter
                    batch_count += 1

                    # If the batch is complete, yield the batch and
reset lists and counter
                    if batch_count == batch_size:
                        X1_batch, X2_batch, y_batch =
np.array(X1_batch), np.array(X2_batch), np.array(y_batch)
                        yield [X1_batch, X2_batch], y_batch
                        X1_batch, X2_batch, y_batch = [], [], []
                        batch_count = 0
```

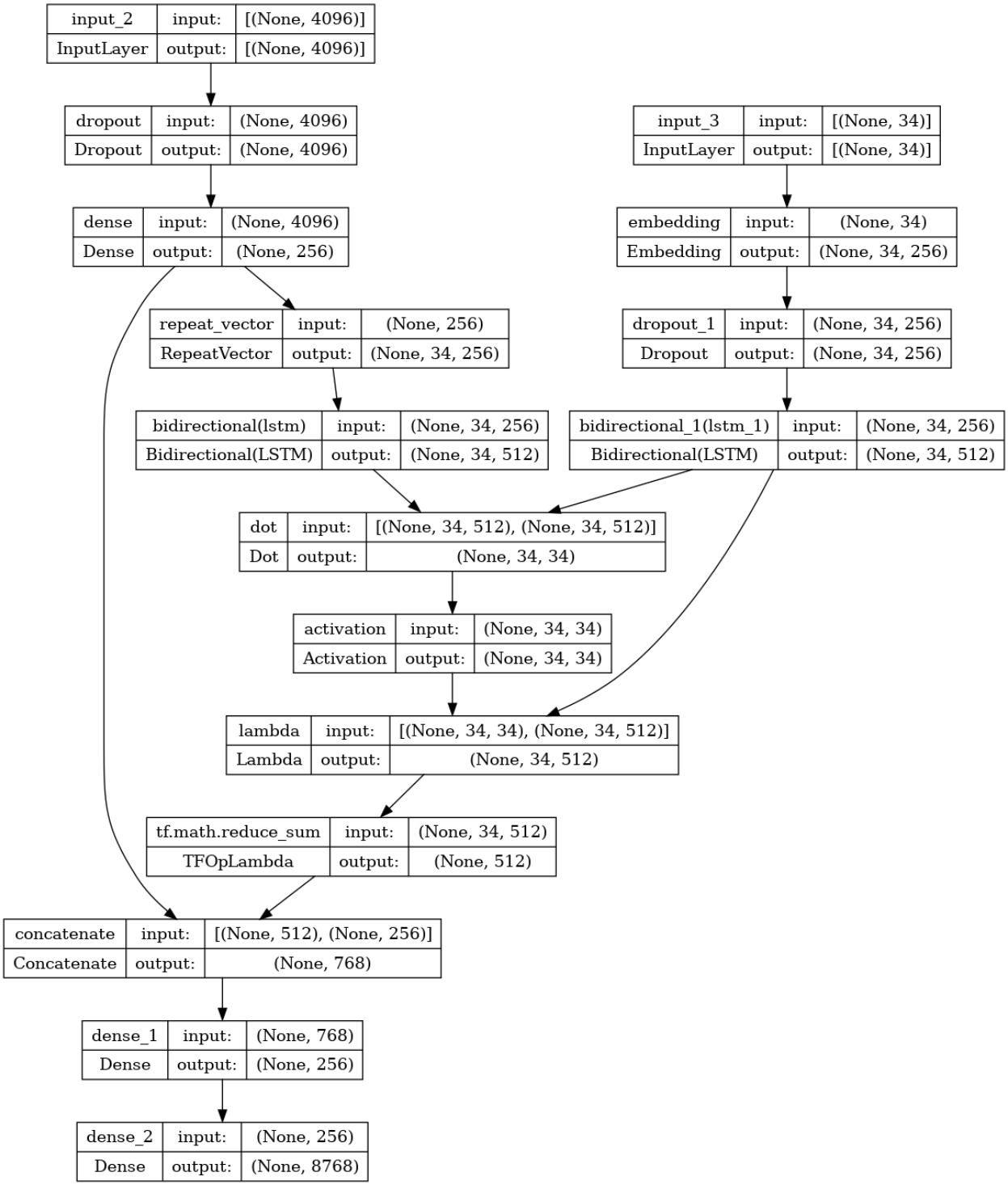
LSTM Model Training

We've got our image features, and we're ready to make our captions come to life. In this section, we'll be diving into the training of our LSTM model. This is where the real magic happens as we teach our model to generate descriptive captions for our images.




```
# Encoder model
inputs1 = Input(shape=(4096,))
fe1 = Dropout(0.5)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)
fe2_projected = RepeatVector(max_caption_length)(fe2)
fe2_projected = Bidirectional(LSTM(256, return_sequences=True))
(fe2_projected)

# Sequence feature layers
inputs2 = Input(shape=(max_caption_length,))
se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
se2 = Dropout(0.5)(se1)
se3 = Bidirectional(LSTM(256, return_sequences=True))(se2)

# Apply attention mechanism using Dot product
attention = Dot(axes=[2, 2])([fe2_projected, se3])  # Calculate
attention scores

# Softmax attention scores
attention_scores = Activation('softmax')(attention)

# Apply attention scores to sequence embeddings
attention_context = Lambda(lambda x: tf.einsum('ijk,ijl->ikl', x[0],
x[1]))([attention_scores, se3])
```

```
# Sum the attended sequence embeddings along the time axis
context_vector = tf.reduce_sum(attention_context, axis=1)

# Decoder model
decoder_input = concatenate([context_vector, fe2], axis=-1)
decoder1 = Dense(256, activation='relu')(decoder_input)
outputs = Dense(vocab_size, activation='softmax')(decoder1)

# Create the model
model = Model(inputs=[inputs1, inputs2], outputs=outputs)
model.compile(loss='categorical_crossentropy', optimizer='adam')

# Visualize the model
plot_model(model, show_shapes=True)
```




```
# Set the number of epochs, batch size
epochs = 50
batch_size = 32

# Calculate the steps_per_epoch based on the number of batches in one
epoch
```

```
steps_per_epoch = ceil(len(train) / batch_size)
validation_steps = ceil(len(test) / batch_size)  # Calculate the steps
for validation data

# Loop through the epochs for training
for epoch in range(epochs):
    print(f"Epoch {epoch+1}/{epochs}")

    # Set up data generators
    train_generator = data_generator(train, image_to_captions_mapping,
loaded_features, tokenizer, max_caption_length, vocab_size,
batch_size)
    test_generator = data_generator(test, image_to_captions_mapping,
loaded_features, tokenizer, max_caption_length, vocab_size,
batch_size)

    model.fit(train_generator, epochs=1,
steps_per_epoch=steps_per_epoch,
          validation_data=test_generator,
validation_steps=validation_steps,
          verbose=1)
```

```
Epoch 1/50
228/228 [==============================] - 73s 258ms/step - loss:
6.4520 - val_loss: 6.3741
Epoch 2/50
228/228 [==============================] - 52s 230ms/step - loss:
5.2670 - val_loss: 6.1908
Epoch 3/50
228/228 [==============================] - 51s 225ms/step - loss:
4.8472 - val_loss: 6.2954
Epoch 4/50
228/228 [==============================] - 52s 226ms/step - loss:
4.5103 - val_loss: 6.4987
Epoch 5/50
228/228 [==============================] - 51s 226ms/step - loss:
4.2174 - val_loss: 6.5653
Epoch 6/50
228/228 [==============================] - 52s 227ms/step - loss:
3.9107 - val_loss: 6.3477
Epoch 7/50
228/228 [==============================] - 52s 228ms/step - loss:
3.6526 - val_loss: 6.9551
Epoch 8/50
228/228 [==============================] - 52s 228ms/step - loss:
3.4303 - val_loss: 6.7037
Epoch 9/50
228/228 [==============================] - 53s 231ms/step - loss:
3.2574 - val_loss: 7.7882
Epoch 10/50
```

```
228/228 [==============================] - 52s 228ms/step - loss:
3.0598 - val_loss: 7.1565
Epoch 11/50
228/228 [==============================] - 52s 227ms/step - loss:
2.8572 - val_loss: 7.0395
Epoch 12/50
228/228 [==============================] - 52s 228ms/step - loss:
2.6469 - val_loss: 7.4882
Epoch 13/50
228/228 [==============================] - 52s 226ms/step - loss:
2.4832 - val_loss: 7.6258
Epoch 14/50
228/228 [==============================] - 52s 229ms/step - loss:
2.3544 - val_loss: 7.2070
Epoch 15/50
228/228 [==============================] - 53s 230ms/step - loss:
2.2270 - val_loss: 7.6325
Epoch 16/50
228/228 [==============================] - 52s 226ms/step - loss:
2.0946 - val_loss: 7.5636
Epoch 17/50
228/228 [==============================] - 52s 229ms/step - loss:
2.0050 - val_loss: 7.6034
Epoch 18/50
228/228 [==============================] - 53s 230ms/step - loss:
1.8646 - val_loss: 8.3715
Epoch 19/50
228/228 [==============================] - 51s 224ms/step - loss:
1.8057 - val_loss: 8.1286
Epoch 20/50
228/228 [==============================] - 52s 228ms/step - loss:
1.7117 - val_loss: 8.1025
Epoch 21/50
228/228 [==============================] - 52s 228ms/step - loss:
1.5914 - val_loss: 9.2000
Epoch 22/50
228/228 [==============================] - 52s 229ms/step - loss:
1.4990 - val_loss: 9.0162
Epoch 23/50
228/228 [==============================] - 52s 229ms/step - loss:
1.4568 - val_loss: 9.2996
Epoch 24/50
228/228 [==============================] - 52s 227ms/step - loss:
1.3984 - val_loss: 8.8099
Epoch 25/50
228/228 [==============================] - 53s 232ms/step - loss:
1.2844 - val_loss: 9.8439
Epoch 26/50
228/228 [==============================] - 52s 228ms/step - loss:
```

```
1.2764 - val_loss: 9.4101
Epoch 27/50
228/228 [==============================] - 51s 224ms/step - loss:
1.2762 - val_loss: 9.4165
Epoch 28/50
228/228 [==============================] - 52s 230ms/step - loss:
1.1724 - val_loss: 10.1617
Epoch 29/50
228/228 [==============================] - 52s 228ms/step - loss:
1.1457 - val_loss: 10.3351
Epoch 30/50
228/228 [==============================] - 51s 225ms/step - loss:
1.1080 - val_loss: 11.1505
Epoch 31/50
228/228 [==============================] - 52s 229ms/step - loss:
0.9995 - val_loss: 10.1985
Epoch 32/50
228/228 [==============================] - 52s 227ms/step - loss:
0.9462 - val_loss: 11.3429
Epoch 33/50
228/228 [==============================] - 52s 229ms/step - loss:
0.9364 - val_loss: 11.7128
Epoch 34/50
228/228 [==============================] - 53s 231ms/step - loss:
0.8476 - val_loss: 11.9435
Epoch 35/50
228/228 [==============================] - 52s 226ms/step - loss:
0.7734 - val_loss: 12.8019
Epoch 36/50
228/228 [==============================] - 52s 229ms/step - loss:
0.7803 - val_loss: 12.5563
Epoch 37/50
228/228 [==============================] - 51s 225ms/step - loss:
0.7313 - val_loss: 12.3470
Epoch 38/50
228/228 [==============================] - 52s 230ms/step - loss:
0.6992 - val_loss: 12.6281
Epoch 39/50
228/228 [==============================] - 53s 231ms/step - loss:
0.6898 - val_loss: 13.1547
Epoch 40/50
228/228 [==============================] - 52s 227ms/step - loss:
0.6790 - val_loss: 12.5031
Epoch 41/50
228/228 [==============================] - 52s 230ms/step - loss:
0.6323 - val_loss: 13.3413
Epoch 42/50
228/228 [==============================] - 52s 227ms/step - loss:
0.5857 - val_loss: 13.4368
```

```
Epoch 43/50
228/228 [==============================] - 52s 227ms/step - loss:
0.5501 - val_loss: 13.9978
Epoch 44/50
228/228 [==============================] - 53s 231ms/step - loss:
0.5530 - val_loss: 13.8764
Epoch 45/50
228/228 [==============================] - 52s 227ms/step - loss:
0.5414 - val_loss: 13.4980
Epoch 46/50
228/228 [==============================] - 52s 229ms/step - loss:
0.5516 - val_loss: 15.7645
Epoch 47/50
228/228 [==============================] - 52s 230ms/step - loss:
0.5572 - val_loss: 14.0722
Epoch 48/50
228/228 [==============================] - 52s 227ms/step - loss:
0.5342 - val_loss: 14.6159
Epoch 49/50
228/228 [==============================] - 52s 228ms/step - loss:
0.5112 - val_loss: 13.9990
Epoch 50/50
228/228 [==============================] - 52s 229ms/step - loss:
0.4954 - val_loss: 15.6088

# Save the model
model.save(OUTPUT_DIR+'/mymodel.h5')
```

Captions Generation

```
def get_word_from_index(index, tokenizer):
    return next((word for word, idx in tokenizer.word_index.items() if
idx == index), None)

def predict_caption(model, image_features, tokenizer,
max_caption_length):
    # Initialize the caption sequence
    caption = 'startseq'

    # Generate the caption
    for _ in range(max_caption_length):
        # Convert the current caption to a sequence of token indices
        sequence = tokenizer.texts_to_sequences([caption])[0]
        # Pad the sequence to match the maximum caption length
        sequence = pad_sequences([sequence],
maxlen=max_caption_length)
        # Predict the next word's probability distribution
        yhat = model.predict([image_features, sequence], verbose=0)
```

```
        # Get the index with the highest probability
        predicted_index = np.argmax(yhat)
        # Convert the index to a word
        predicted_word = get_word_from_index(predicted_index,
tokenizer)

        # Append the predicted word to the caption
        caption += " " + predicted_word

        # Stop if the word is None or if the end sequence tag is
encountered
        if predicted_word is None or predicted_word == 'endseq':
            break

    return caption

# Initialize lists to store actual and predicted captions
actual_captions_list = []
predicted_captions_list = []

# Loop through the test data
for key in tqdm(test):
    # Get actual captions for the current image
    actual_captions = image_to_captions_mapping[key]
    # Predict the caption for the image using the model
    predicted_caption = predict_caption(model, loaded_features[key],
tokenizer, max_caption_length)

    # Split actual captions into words
    actual_captions_words = [caption.split() for caption in
actual_captions]
    # Split predicted caption into words
    predicted_caption_words = predicted_caption.split()

    # Append to the lists
    actual_captions_list.append(actual_captions_words)
    predicted_captions_list.append(predicted_caption_words)

# Calculate BLEU score
print("BLEU-1: %f" % corpus_bleu(actual_captions_list,
predicted_captions_list, weights=(1.0, 0, 0, 0)))
print("BLEU-2: %f" % corpus_bleu(actual_captions_list,
predicted_captions_list, weights=(0.5, 0.5, 0, 0)))
```

{"model_id":"7a4d4562d0ec4a03ac75bf8956d4c841","version_major":2,"version_minor":0}

```
BLEU-1: 0.379877
BLEU-2: 0.167031
```

Predicting captions for Images

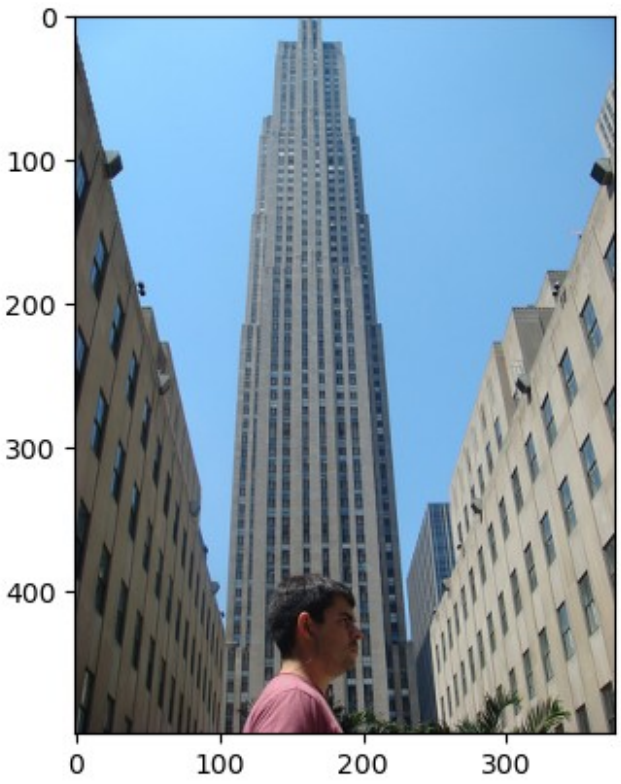
```
# Function for generating caption
def generate_caption(image_name):
    # load the image
    image_id = image_name.split('.')[0]
    img_path = os.path.join(INPUT_DIR, "Images", image_name)
    image = Image.open(img_path)
    captions = image_to_captions_mapping[image_id]
    print('---------------------Actual---------------------')
    for caption in captions:
        print(caption)
    # predict the caption
    y_pred = predict_caption(model, loaded_features[image_id],
tokenizer, max_caption_length)
    print('--------------------Predicted--------------------')
    print(y_pred)
    plt.imshow(image)

generate_caption("101669240_b2d3e7f17b.jpg")
```

```
---------------------Actual---------------------
startseq man in hat is displaying pictures next to skier in blue hat
endseq
startseq man skis past another man displaying paintings in the snow
endseq
startseq person wearing skis looking at framed pictures set up in the
snow endseq
startseq skier looks at framed pictures in the snow next to trees
endseq
startseq man on skis looking at artwork for sale in the snow endseq
--------------------Predicted--------------------
startseq person skis past another man displaying displaying pictures
in the snow endseq
```


```
generate_caption("1077546505_a4f6c4daa9.jpg")

----------------------Actual----------------------
startseq boy in blue shorts slides down slide into pool endseq
startseq boy in blue swimming trunks slides down yellow slide into
wading pool with inflatable toys floating in the water endseq
startseq boy rides down slide into small backyard pool endseq
startseq boy sliding down slide into pool with colorful tubes endseq
startseq child is falling off slide onto colored balloons floating on
pool of water endseq
--------------------Predicted--------------------
startseq child is falling down slide into pool with colorful tubes
endseq
```


```
generate_caption("1002674143_1b742ab4b8.jpg")

----------------------Actual----------------------
startseq little girl covered in paint sits in front of painted rainbow
with her hands in bowl endseq
startseq little girl is sitting in front of large painted rainbow
endseq
startseq small girl in the grass plays with fingerpaints in front of
white canvas with rainbow on it endseq
startseq there is girl with pigtails sitting in front of rainbow
painting endseq
startseq young girl with pigtails painting outside in the grass endseq
---------------------Predicted---------------------
startseq little girl in front of the camera in front of rainbow
painting endseq
```


```
generate_caption("1032460886_4a598ed535.jpg")

----------------------Actual----------------------
startseq man is standing in front of skyscraper endseq
startseq man stands in front of skyscraper endseq
startseq man stands in front of very tall building endseq
startseq behind the man in red shirt stands large skyscraper endseq
startseq there is skyscraper in the distance with man walking in front
of the camera endseq
--------------------Predicted--------------------
startseq man stands in front of very tall building endseq
```




```
generate_caption("1032122270_ea6f0beedb.jpg")

----------------------Actual----------------------
startseq woman crouches near three dogs in field endseq
startseq three dogs are playing on grassy hill with blue sky endseq
startseq three dogs are standing in the grass and person is sitting
next to them endseq
startseq three dogs on grassy hill endseq
startseq three dogs stand in grassy field while person kneels nearby
endseq
---------------------Predicted---------------------
startseq three dogs are standing in the grass and person is sitting
next to them endseq
```


0
50
100
150
200
250
300
350
0
100
200
300
400