Image Captioning Model Implementation

Authors

Hussein Mohamed - hussain.mansour2019@gmail.com

Marwan Essam - marwanesam243@gmail.com

Mariam Hossam - m.hossam2551@gmail.com

Introduction

Image captioning is the process of generating textual descriptions for images automatically. It combines computer vision and natural language processing techniques to understand the content of an image and generate a coherent and relevant description.

In this notebook, we apply Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) to image captioning for Flickr8k dataset.

Import Modules

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

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

```
!pip install datasets
!pip install keras-tuner
```

```
Collecting datasets
  Downloading datasets-2.19.1-py3-none-any.whl (542 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 542.0/542.0 kB 5.5 MB/s eta
0:00:00
ent already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from datasets) (3.14.0)
Requirement already satisfied: numpy>=1.17 in
/usr/local/lib/python3.10/dist-packages (from datasets) (1.25.2)
Requirement already satisfied: pyarrow>=12.0.0 in
/usr/local/lib/python3.10/dist-packages (from datasets) (14.0.2)
Requirement already satisfied: pyarrow-hotfix in
/usr/local/lib/python3.10/dist-packages (from datasets) (0.6)
Collecting dill<0.3.9,>=0.3.0 (from datasets)
  Downloading dill-0.3.8-py3-none-any.whl (116 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 116.3/116.3 kB 5.3 MB/s eta
0:00:00
ent already satisfied: pandas in /usr/local/lib/python3.10/dist-
packages (from datasets) (2.0.3)
```

```
Requirement already satisfied: requests>=2.19.0 in
/usr/local/lib/python3.10/dist-packages (from datasets) (2.31.0)
Requirement already satisfied: tqdm>=4.62.1 in
/usr/local/lib/python3.10/dist-packages (from datasets) (4.66.4)
Collecting xxhash (from datasets)
  Downloading xxhash-3.4.1-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (194 kB)
 194.1/194.1 kB 4.3 MB/s eta
0:00:00
ultiprocess (from datasets)
  Downloading multiprocess-0.70.16-py310-none-any.whl (134 kB)
 134.8/134.8 kB 6.0 MB/s eta
0:00:00
ent already satisfied: fsspec[http]<=2024.3.1,>=2023.1.0 in
/usr/local/lib/python3.10/dist-packages (from datasets) (2023.6.0)
Requirement already satisfied: aiohttp in
/usr/local/lib/python3.10/dist-packages (from datasets) (3.9.5)
Collecting huggingface-hub>=0.21.2 (from datasets)
  Downloading huggingface_hub-0.23.0-py3-none-any.whl (401 kB)
 401.2/401.2 kB 6.0 MB/s eta
0:00:00
ent already satisfied: packaging in /usr/local/lib/python3.10/dist-
packages (from datasets) (24.0)
Requirement already satisfied: pyyaml>=5.1 in
/usr/local/lib/python3.10/dist-packages (from datasets) (6.0.1)
Requirement already satisfied: aiosignal>=1.1.2 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(1.3.1)
Requirement already satisfied: attrs>=17.3.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(23.2.0)
Requirement already satisfied: frozenlist>=1.1.1 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(1.4.1)
Requirement already satisfied: multidict<7.0,>=4.5 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(6.0.5)
Requirement already satisfied: yarl<2.0,>=1.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(1.9.4)
Requirement already satisfied: async-timeout<5.0,>=4.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(4.0.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.21.2-
>datasets) (4.11.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.19.0-
>datasets) (3.3.2)
```

```
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.19.0-
>datasets) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.19.0-
>datasets) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.19.0-
>datasets) (2024.2.2)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas->datasets)
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas->datasets)
(2023.4)
Requirement already satisfied: tzdata>=2022.1 in
/usr/local/lib/python3.10/dist-packages (from pandas->datasets)
(2024.1)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2-
>pandas->datasets) (1.16.0)
Installing collected packages: xxhash, dill, multiprocess,
huggingface-hub, datasets
  Attempting uninstall: huggingface-hub
    Found existing installation: huggingface-hub 0.20.3
    Uninstalling huggingface-hub-0.20.3:
      Successfully uninstalled huggingface-hub-0.20.3
Successfully installed datasets-2.19.1 dill-0.3.8 huggingface-hub-
0.23.0 multiprocess-0.70.16 xxhash-3.4.1
```

```python
from PIL import Image
import os
import pickle
import re
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from datasets import load_dataset
from collections import Counter
from tensorflow.keras.preprocessing.image import img_to_array,
load_img
import tensorflow as tf
from tensorflow.keras.applications.resnet50 import ResNet50,
preprocess_input
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
from transformers import PreTrainedTokenizerFast
from keras import Sequential, layers
from keras.layers import *
from keras.models import Model
```

```
from tensorflow.keras.utils import to_categorical, plot_model
from tensorflow.keras.optimizers import RMSprop, Adam
from nltk.translate.bleu_score import corpus_bleu
from tensorflow.keras.utils import Sequence
from textwrap import wrap
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping,
ReduceLROnPlateau
from keras_tuner import HyperModel
from keras_tuner import RandomSearch
from keras_tuner import Hyperband
from keras_tuner import GridSearch

WORKING_DIR = '/content/drive/MyDrive/PR_3'
```

Dataset Preparation

Loading and Preprocessing Images

Loading, resizing and normalizing the images to a suitable format that can be efficiently processed by the CNN model.

1. Load dataset

For this project, we will be using the Flickr8k dataset, which consists of 8000 images. Each image in the dataset is associated with five different captions, providing diverse descriptions for the same image. The dataset is divided into three subsets:

- Training Set: 6000 images
- Validation Set: 1000 images
- Test Set: 1000 images

```
dataset = load_dataset("jxie/flickr8k")
print(dataset)
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/
_token.py:89: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your
settings tab (https://huggingface.co/settings/tokens), set it as
secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to
access public models or datasets.
  warnings.warn(
```


{"model_id":"97bd50b5758b448884a54761b96d6832","version_major":2,"version_minor":0}

{"model_id":"0e4118359043423fa015e92a612e7be0","version_major":2,"version_minor":0}

{"model_id":"f3994fb4e9fc457eadf72e8e231a82e6","version_major":2,"version_minor":0}

{"model_id":"1b21af2565214ab694e9b62d0c1357c3","version_major":2,"version_minor":0}

{"model_id":"8432e3ff82444c3397820ae52c05e3b9","version_major":2,"version_minor":0}

{"model_id":"35553b2b95884344879e9182c0be8cdc","version_major":2,"version_minor":0}

{"model_id":"6eadca39009b40f1947bbd35a36489d6","version_major":2,"version_minor":0}

{"model_id":"8df03b5105244e4798354d988edacae1","version_major":2,"version_minor":0}

```
DatasetDict({
    train: Dataset({
        features: ['image', 'caption_0', 'caption_1', 'caption_2',
'caption_3', 'caption_4'],
        num_rows: 6000
    })
    validation: Dataset({
        features: ['image', 'caption_0', 'caption_1', 'caption_2',
'caption_3', 'caption_4'],
        num_rows: 1000
    })
    test: Dataset({
        features: ['image', 'caption_0', 'caption_1', 'caption_2',
'caption_3', 'caption_4'],
        num_rows: 1000
    })
})
```

```
# Access specific fields in the first example
first_example = dataset['train'][0]
image = first_example['image']
plt.imshow(image)
plt.axis('off')  # Turn off axis
plt.show()
for caption in first_example:
  print(first_example[caption])
```

```
<PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=500x399 at
0x79A04D660EB0>
A black dog is running after a white dog in the snow .
Black dog chasing brown dog through snow
Two dogs chase each other across the snowy ground .
Two dogs play together in the snow .
Two dogs running through a low lying body of water .
```

1. Preprocess images by resizing then normalizing them

We use `resnet50.preprocess_input` which adequates images to the format the resnet50 model requires. Resnet50 preprocessing uses "caffe" style in which images are centred and not normalized.

```
# This function resize the given image to target size then normalize
it
def preprocess(image):
  image = image.resize((224, 224)) # (224, 224, 3)
  image = img_to_array(image) # Convert image to pixel array
  image = np.expand_dims(image, axis = 0) # (1, 224, 224, 3)
  image = preprocess_input(image) # Preprocess image based on resnet50
values
  return image
```

Features Extraction with CNN

1. Extract features from images using pre-trained ResNet model on the ImageNet dataset

- We utilize a pre-trained ResNet model on the ImageNet dataset and keep its layers frozen.
- Image features are extracted just before the last layer of classification.
- Global Average Pooling layer is selected as the final layer of the Resnet50 model for our feature extraction, so image embeddings will be a vector of size 2048.

To avoid long running time, upload this file to the *WORKING_DIR* then run only the last cell which loads features from the file.

```
base_model = ResNet50(include_top = False, weights = 'imagenet',
input_shape = (224, 224, 3), pooling = 'avg')
base_model.trainable = False # Freeze model
base_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/resnet/
resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 1s 0us/step
Model: "resnet50"
_________________________________________________________________
____________________________
 Layer (type)                Output Shape                 Param #
Connected to
=================================================================
============================
 input_3 (InputLayer)        [(None, 224, 224, 3)]        0          []


 conv1_pad (ZeroPadding2D)   (None, 230, 230, 3)          0
['input_3[0][0]']


 conv1_conv (Conv2D)         (None, 112, 112, 64)         9472
['conv1_pad[0][0]']


 conv1_bn (BatchNormalizati  (None, 112, 112, 64)         256
['conv1_conv[0][0]']
 on)


 conv1_relu (Activation)     (None, 112, 112, 64)         0
['conv1_bn[0][0]']


 pool1_pad (ZeroPadding2D)   (None, 114, 114, 64)         0
['conv1_relu[0][0]']
```

```
 pool1_pool (MaxPooling2D)   (None, 56, 56, 64)           0
['pool1_pad[0][0]']


 conv2_block1_1_conv (Conv2  (None, 56, 56, 64)           4160
['pool1_pool[0][0]']
 D)



 conv2_block1_1_bn (BatchNo  (None, 56, 56, 64)           256
['conv2_block1_1_conv[0][0]']
 rmalization)



 conv2_block1_1_relu (Activ  (None, 56, 56, 64)           0
['conv2_block1_1_bn[0][0]']
 ation)



 conv2_block1_2_conv (Conv2  (None, 56, 56, 64)           36928
['conv2_block1_1_relu[0][0]']
 D)



 conv2_block1_2_bn (BatchNo  (None, 56, 56, 64)           256
['conv2_block1_2_conv[0][0]']
 rmalization)



 conv2_block1_2_relu (Activ  (None, 56, 56, 64)           0
['conv2_block1_2_bn[0][0]']
 ation)



 conv2_block1_0_conv (Conv2  (None, 56, 56, 256)          16640
['pool1_pool[0][0]']
 D)



 conv2_block1_3_conv (Conv2  (None, 56, 56, 256)          16640
['conv2_block1_2_relu[0][0]']
 D)
```

```
 conv2_block1_0_bn (BatchNo  (None, 56, 56, 256)          1024
['conv2_block1_0_conv[0][0]']
 rmalization)


 conv2_block1_3_bn (BatchNo  (None, 56, 56, 256)          1024
['conv2_block1_3_conv[0][0]']
 rmalization)


 conv2_block1_add (Add)      (None, 56, 56, 256)          0
['conv2_block1_0_bn[0][0]',

'conv2_block1_3_bn[0][0]']

 conv2_block1_out (Activati  (None, 56, 56, 256)          0
['conv2_block1_add[0][0]']
 on)


 conv2_block2_1_conv (Conv2  (None, 56, 56, 64)           16448
['conv2_block1_out[0][0]']
 D)


 conv2_block2_1_bn (BatchNo  (None, 56, 56, 64)           256
['conv2_block2_1_conv[0][0]']
 rmalization)


 conv2_block2_1_relu (Activ  (None, 56, 56, 64)           0
['conv2_block2_1_bn[0][0]']
 ation)


 conv2_block2_2_conv (Conv2  (None, 56, 56, 64)           36928
['conv2_block2_1_relu[0][0]']
 D)


 conv2_block2_2_bn (BatchNo  (None, 56, 56, 64)           256
```

```
['conv2_block2_2_conv[0][0]']
 rmalization)



 conv2_block2_2_relu (Activ  (None, 56, 56, 64)           0
['conv2_block2_2_bn[0][0]']
 ation)



 conv2_block2_3_conv (Conv2  (None, 56, 56, 256)          16640
['conv2_block2_2_relu[0][0]']
 D)



 conv2_block2_3_bn (BatchNo  (None, 56, 56, 256)          1024
['conv2_block2_3_conv[0][0]']
 rmalization)



 conv2_block2_add (Add)      (None, 56, 56, 256)          0
['conv2_block1_out[0][0]',

'conv2_block2_3_bn[0][0]']


 conv2_block2_out (Activati  (None, 56, 56, 256)          0
['conv2_block2_add[0][0]']
 on)



 conv2_block3_1_conv (Conv2  (None, 56, 56, 64)           16448
['conv2_block2_out[0][0]']
 D)



 conv2_block3_1_bn (BatchNo  (None, 56, 56, 64)           256
['conv2_block3_1_conv[0][0]']
 rmalization)



 conv2_block3_1_relu (Activ  (None, 56, 56, 64)           0
['conv2_block3_1_bn[0][0]']
 ation)
```

```
 conv2_block3_2_conv (Conv2  (None, 56, 56, 64)          36928
['conv2_block3_1_relu[0][0]']
 D)

 conv2_block3_2_bn (BatchNo  (None, 56, 56, 64)          256
['conv2_block3_2_conv[0][0]']
 rmalization)

 conv2_block3_2_relu (Activ  (None, 56, 56, 64)          0
['conv2_block3_2_bn[0][0]']
 ation)

 conv2_block3_3_conv (Conv2  (None, 56, 56, 256)         16640
['conv2_block3_2_relu[0][0]']
 D)

 conv2_block3_3_bn (BatchNo  (None, 56, 56, 256)         1024
['conv2_block3_3_conv[0][0]']
 rmalization)

 conv2_block3_add (Add)      (None, 56, 56, 256)         0
['conv2_block2_out[0][0]',

'conv2_block3_3_bn[0][0]']

 conv2_block3_out (Activati  (None, 56, 56, 256)         0
['conv2_block3_add[0][0]']
 on)

 conv3_block1_1_conv (Conv2  (None, 28, 28, 128)         32896
['conv2_block3_out[0][0]']
 D)

 conv3_block1_1_bn (BatchNo  (None, 28, 28, 128)         512
```

```
['conv3_block1_1_conv[0][0]']
 rmalization)



 conv3_block1_1_relu (Activ  (None, 28, 28, 128)          0
['conv3_block1_1_bn[0][0]']
 ation)



 conv3_block1_2_conv (Conv2  (None, 28, 28, 128)          147584
['conv3_block1_1_relu[0][0]']
 D)



 conv3_block1_2_bn (BatchNo  (None, 28, 28, 128)          512
['conv3_block1_2_conv[0][0]']
 rmalization)



 conv3_block1_2_relu (Activ  (None, 28, 28, 128)          0
['conv3_block1_2_bn[0][0]']
 ation)



 conv3_block1_0_conv (Conv2  (None, 28, 28, 512)          131584
['conv2_block3_out[0][0]']
 D)



 conv3_block1_3_conv (Conv2  (None, 28, 28, 512)          66048
['conv3_block1_2_relu[0][0]']
 D)



 conv3_block1_0_bn (BatchNo  (None, 28, 28, 512)          2048
['conv3_block1_0_conv[0][0]']
 rmalization)



 conv3_block1_3_bn (BatchNo  (None, 28, 28, 512)          2048
['conv3_block1_3_conv[0][0]']
 rmalization)
```

```
 conv3_block1_add (Add)       (None, 28, 28, 512)          0
['conv3_block1_0_bn[0][0]',

'conv3_block1_3_bn[0][0]']


 conv3_block1_out (Activati  (None, 28, 28, 512)          0
['conv3_block1_add[0][0]']
 on)



 conv3_block2_1_conv (Conv2  (None, 28, 28, 128)          65664
['conv3_block1_out[0][0]']
 D)



 conv3_block2_1_bn (BatchNo  (None, 28, 28, 128)          512
['conv3_block2_1_conv[0][0]']
 rmalization)



 conv3_block2_1_relu (Activ  (None, 28, 28, 128)          0
['conv3_block2_1_bn[0][0]']
 ation)



 conv3_block2_2_conv (Conv2  (None, 28, 28, 128)          147584
['conv3_block2_1_relu[0][0]']
 D)



 conv3_block2_2_bn (BatchNo  (None, 28, 28, 128)          512
['conv3_block2_2_conv[0][0]']
 rmalization)



 conv3_block2_2_relu (Activ  (None, 28, 28, 128)          0
['conv3_block2_2_bn[0][0]']
 ation)



 conv3_block2_3_conv (Conv2  (None, 28, 28, 512)          66048
```

```
['conv3_block2_2_relu[0][0]']
 D)



 conv3_block2_3_bn (BatchNo  (None, 28, 28, 512)          2048
['conv3_block2_3_conv[0][0]']
 rmalization)



 conv3_block2_add (Add)      (None, 28, 28, 512)          0
['conv3_block1_out[0][0]',

'conv3_block2_3_bn[0][0]']


 conv3_block2_out (Activati  (None, 28, 28, 512)          0
['conv3_block2_add[0][0]']
 on)



 conv3_block3_1_conv (Conv2  (None, 28, 28, 128)          65664
['conv3_block2_out[0][0]']
 D)



 conv3_block3_1_bn (BatchNo  (None, 28, 28, 128)          512
['conv3_block3_1_conv[0][0]']
 rmalization)



 conv3_block3_1_relu (Activ  (None, 28, 28, 128)          0
['conv3_block3_1_bn[0][0]']
 ation)



 conv3_block3_2_conv (Conv2  (None, 28, 28, 128)          147584
['conv3_block3_1_relu[0][0]']
 D)



 conv3_block3_2_bn (BatchNo  (None, 28, 28, 128)          512
['conv3_block3_2_conv[0][0]']
 rmalization)
```

```
 conv3_block3_2_relu (Activ  (None, 28, 28, 128)          0
['conv3_block3_2_bn[0][0]']
 ation)


 conv3_block3_3_conv (Conv2  (None, 28, 28, 512)          66048
['conv3_block3_2_relu[0][0]']
 D)


 conv3_block3_3_bn (BatchNo  (None, 28, 28, 512)          2048
['conv3_block3_3_conv[0][0]']
 rmalization)


 conv3_block3_add (Add)      (None, 28, 28, 512)          0
['conv3_block2_out[0][0]',

'conv3_block3_3_bn[0][0]']


 conv3_block3_out (Activati  (None, 28, 28, 512)          0
['conv3_block3_add[0][0]']
 on)


 conv3_block4_1_conv (Conv2  (None, 28, 28, 128)          65664
['conv3_block3_out[0][0]']
 D)


 conv3_block4_1_bn (BatchNo  (None, 28, 28, 128)          512
['conv3_block4_1_conv[0][0]']
 rmalization)


 conv3_block4_1_relu (Activ  (None, 28, 28, 128)          0
['conv3_block4_1_bn[0][0]']
 ation)


 conv3_block4_2_conv (Conv2  (None, 28, 28, 128)          147584
```

```
['conv3_block4_1_relu[0][0]']
 D)



 conv3_block4_2_bn (BatchNo  (None, 28, 28, 128)          512
['conv3_block4_2_conv[0][0]']
 rmalization)



 conv3_block4_2_relu (Activ  (None, 28, 28, 128)          0
['conv3_block4_2_bn[0][0]']
 ation)



 conv3_block4_3_conv (Conv2  (None, 28, 28, 512)          66048
['conv3_block4_2_relu[0][0]']
 D)



 conv3_block4_3_bn (BatchNo  (None, 28, 28, 512)          2048
['conv3_block4_3_conv[0][0]']
 rmalization)



 conv3_block4_add (Add)      (None, 28, 28, 512)          0
['conv3_block3_out[0][0]',

'conv3_block4_3_bn[0][0]']


 conv3_block4_out (Activati  (None, 28, 28, 512)          0
['conv3_block4_add[0][0]']
 on)



 conv4_block1_1_conv (Conv2  (None, 14, 14, 256)          131328
['conv3_block4_out[0][0]']
 D)



 conv4_block1_1_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block1_1_conv[0][0]']
 rmalization)
```

```
 conv4_block1_1_relu (Activ  (None, 14, 14, 256)         0
['conv4_block1_1_bn[0][0]']
 ation)

 conv4_block1_2_conv (Conv2  (None, 14, 14, 256)         590080
['conv4_block1_1_relu[0][0]']
 D)

 conv4_block1_2_bn (BatchNo  (None, 14, 14, 256)         1024
['conv4_block1_2_conv[0][0]']
 rmalization)

 conv4_block1_2_relu (Activ  (None, 14, 14, 256)         0
['conv4_block1_2_bn[0][0]']
 ation)

 conv4_block1_0_conv (Conv2  (None, 14, 14, 1024)        525312
['conv3_block4_out[0][0]']
 D)

 conv4_block1_3_conv (Conv2  (None, 14, 14, 1024)        263168
['conv4_block1_2_relu[0][0]']
 D)

 conv4_block1_0_bn (BatchNo  (None, 14, 14, 1024)        4096
['conv4_block1_0_conv[0][0]']
 rmalization)

 conv4_block1_3_bn (BatchNo  (None, 14, 14, 1024)        4096
['conv4_block1_3_conv[0][0]']
 rmalization)

 conv4_block1_add (Add)      (None, 14, 14, 1024)        0
```

```
['conv4_block1_0_bn[0][0]',

'conv4_block1_3_bn[0][0]']


 conv4_block1_out (Activati  (None, 14, 14, 1024)         0
['conv4_block1_add[0][0]']
 on)



 conv4_block2_1_conv (Conv2  (None, 14, 14, 256)          262400
['conv4_block1_out[0][0]']
 D)



 conv4_block2_1_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block2_1_conv[0][0]']
 rmalization)



 conv4_block2_1_relu (Activ  (None, 14, 14, 256)          0
['conv4_block2_1_bn[0][0]']
 ation)



 conv4_block2_2_conv (Conv2  (None, 14, 14, 256)          590080
['conv4_block2_1_relu[0][0]']
 D)



 conv4_block2_2_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block2_2_conv[0][0]']
 rmalization)



 conv4_block2_2_relu (Activ  (None, 14, 14, 256)          0
['conv4_block2_2_bn[0][0]']
 ation)



 conv4_block2_3_conv (Conv2  (None, 14, 14, 1024)         263168
['conv4_block2_2_relu[0][0]']
 D)
```

```
 conv4_block2_3_bn (BatchNo  (None, 14, 14, 1024)         4096
['conv4_block2_3_conv[0][0]']
 rmalization)


 conv4_block2_add (Add)      (None, 14, 14, 1024)         0
['conv4_block1_out[0][0]',

'conv4_block2_3_bn[0][0]']


 conv4_block2_out (Activati  (None, 14, 14, 1024)         0
['conv4_block2_add[0][0]']
 on)


 conv4_block3_1_conv (Conv2  (None, 14, 14, 256)          262400
['conv4_block2_out[0][0]']
 D)


 conv4_block3_1_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block3_1_conv[0][0]']
 rmalization)


 conv4_block3_1_relu (Activ  (None, 14, 14, 256)          0
['conv4_block3_1_bn[0][0]']
 ation)


 conv4_block3_2_conv (Conv2  (None, 14, 14, 256)          590080
['conv4_block3_1_relu[0][0]']
 D)


 conv4_block3_2_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block3_2_conv[0][0]']
 rmalization)


 conv4_block3_2_relu (Activ  (None, 14, 14, 256)          0
```

```
['conv4_block3_2_bn[0][0]']
 ation)



 conv4_block3_3_conv (Conv2  (None, 14, 14, 1024)         263168
['conv4_block3_2_relu[0][0]']
 D)



 conv4_block3_3_bn (BatchNo  (None, 14, 14, 1024)         4096
['conv4_block3_3_conv[0][0]']
 rmalization)



 conv4_block3_add (Add)      (None, 14, 14, 1024)         0
['conv4_block2_out[0][0]',

'conv4_block3_3_bn[0][0]']


 conv4_block3_out (Activati  (None, 14, 14, 1024)         0
['conv4_block3_add[0][0]']
 on)



 conv4_block4_1_conv (Conv2  (None, 14, 14, 256)          262400
['conv4_block3_out[0][0]']
 D)



 conv4_block4_1_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block4_1_conv[0][0]']
 rmalization)



 conv4_block4_1_relu (Activ  (None, 14, 14, 256)          0
['conv4_block4_1_bn[0][0]']
 ation)



 conv4_block4_2_conv (Conv2  (None, 14, 14, 256)          590080
['conv4_block4_1_relu[0][0]']
 D)
```

```
 conv4_block4_2_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block4_2_conv[0][0]']
 rmalization)



 conv4_block4_2_relu (Activ  (None, 14, 14, 256)          0
['conv4_block4_2_bn[0][0]']
 ation)



 conv4_block4_3_conv (Conv2  (None, 14, 14, 1024)         263168
['conv4_block4_2_relu[0][0]']
 D)



 conv4_block4_3_bn (BatchNo  (None, 14, 14, 1024)         4096
['conv4_block4_3_conv[0][0]']
 rmalization)



 conv4_block4_add (Add)      (None, 14, 14, 1024)         0
['conv4_block3_out[0][0]',

'conv4_block4_3_bn[0][0]']


 conv4_block4_out (Activati  (None, 14, 14, 1024)         0
['conv4_block4_add[0][0]']
 on)



 conv4_block5_1_conv (Conv2  (None, 14, 14, 256)          262400
['conv4_block4_out[0][0]']
 D)



 conv4_block5_1_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block5_1_conv[0][0]']
 rmalization)



 conv4_block5_1_relu (Activ  (None, 14, 14, 256)          0
```

```
['conv4_block5_1_bn[0][0]']
 ation)



 conv4_block5_2_conv (Conv2  (None, 14, 14, 256)          590080
['conv4_block5_1_relu[0][0]']
 D)



 conv4_block5_2_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block5_2_conv[0][0]']
 rmalization)



 conv4_block5_2_relu (Activ  (None, 14, 14, 256)          0
['conv4_block5_2_bn[0][0]']
 ation)



 conv4_block5_3_conv (Conv2  (None, 14, 14, 1024)         263168
['conv4_block5_2_relu[0][0]']
 D)



 conv4_block5_3_bn (BatchNo  (None, 14, 14, 1024)         4096
['conv4_block5_3_conv[0][0]']
 rmalization)



 conv4_block5_add (Add)      (None, 14, 14, 1024)         0
['conv4_block4_out[0][0]',

'conv4_block5_3_bn[0][0]']


 conv4_block5_out (Activati  (None, 14, 14, 1024)         0
['conv4_block5_add[0][0]']
 on)



 conv4_block6_1_conv (Conv2  (None, 14, 14, 256)          262400
['conv4_block5_out[0][0]']
 D)
```

```
 conv4_block6_1_bn (BatchNo  (None, 14, 14, 256)         1024
['conv4_block6_1_conv[0][0]']
 rmalization)


 conv4_block6_1_relu (Activ  (None, 14, 14, 256)         0
['conv4_block6_1_bn[0][0]']
 ation)


 conv4_block6_2_conv (Conv2  (None, 14, 14, 256)         590080
['conv4_block6_1_relu[0][0]']
 D)


 conv4_block6_2_bn (BatchNo  (None, 14, 14, 256)         1024
['conv4_block6_2_conv[0][0]']
 rmalization)


 conv4_block6_2_relu (Activ  (None, 14, 14, 256)         0
['conv4_block6_2_bn[0][0]']
 ation)


 conv4_block6_3_conv (Conv2  (None, 14, 14, 1024)        263168
['conv4_block6_2_relu[0][0]']
 D)


 conv4_block6_3_bn (BatchNo  (None, 14, 14, 1024)        4096
['conv4_block6_3_conv[0][0]']
 rmalization)


 conv4_block6_add (Add)      (None, 14, 14, 1024)        0
['conv4_block5_out[0][0]',

'conv4_block6_3_bn[0][0]']


 conv4_block6_out (Activati  (None, 14, 14, 1024)        0
```

```
['conv4_block6_add[0][0]']
 on)



 conv5_block1_1_conv (Conv2  (None, 7, 7, 512)            524800
['conv4_block6_out[0][0]']
 D)



 conv5_block1_1_bn (BatchNo  (None, 7, 7, 512)            2048
['conv5_block1_1_conv[0][0]']
 rmalization)



 conv5_block1_1_relu (Activ  (None, 7, 7, 512)            0
['conv5_block1_1_bn[0][0]']
 ation)



 conv5_block1_2_conv (Conv2  (None, 7, 7, 512)            2359808
['conv5_block1_1_relu[0][0]']
 D)



 conv5_block1_2_bn (BatchNo  (None, 7, 7, 512)            2048
['conv5_block1_2_conv[0][0]']
 rmalization)



 conv5_block1_2_relu (Activ  (None, 7, 7, 512)            0
['conv5_block1_2_bn[0][0]']
 ation)



 conv5_block1_0_conv (Conv2  (None, 7, 7, 2048)           2099200
['conv4_block6_out[0][0]']
 D)



 conv5_block1_3_conv (Conv2  (None, 7, 7, 2048)           1050624
['conv5_block1_2_relu[0][0]']
 D)
```

```
 conv5_block1_0_bn (BatchNo  (None, 7, 7, 2048)           8192
['conv5_block1_0_conv[0][0]']
 rmalization)



 conv5_block1_3_bn (BatchNo  (None, 7, 7, 2048)           8192
['conv5_block1_3_conv[0][0]']
 rmalization)



 conv5_block1_add (Add)      (None, 7, 7, 2048)           0
['conv5_block1_0_bn[0][0]',

'conv5_block1_3_bn[0][0]']


 conv5_block1_out (Activati  (None, 7, 7, 2048)           0
['conv5_block1_add[0][0]']
 on)



 conv5_block2_1_conv (Conv2  (None, 7, 7, 512)            1049088
['conv5_block1_out[0][0]']
 D)



 conv5_block2_1_bn (BatchNo  (None, 7, 7, 512)            2048
['conv5_block2_1_conv[0][0]']
 rmalization)



 conv5_block2_1_relu (Activ  (None, 7, 7, 512)            0
['conv5_block2_1_bn[0][0]']
 ation)



 conv5_block2_2_conv (Conv2  (None, 7, 7, 512)            2359808
['conv5_block2_1_relu[0][0]']
 D)



 conv5_block2_2_bn (BatchNo  (None, 7, 7, 512)            2048
```

```
['conv5_block2_2_conv[0][0]']
 rmalization)



 conv5_block2_2_relu (Activ  (None, 7, 7, 512)            0
['conv5_block2_2_bn[0][0]']
 ation)



 conv5_block2_3_conv (Conv2  (None, 7, 7, 2048)           1050624
['conv5_block2_2_relu[0][0]']
 D)



 conv5_block2_3_bn (BatchNo  (None, 7, 7, 2048)           8192
['conv5_block2_3_conv[0][0]']
 rmalization)



 conv5_block2_add (Add)      (None, 7, 7, 2048)           0
['conv5_block1_out[0][0]',

'conv5_block2_3_bn[0][0]']


 conv5_block2_out (Activati  (None, 7, 7, 2048)           0
['conv5_block2_add[0][0]']
 on)



 conv5_block3_1_conv (Conv2  (None, 7, 7, 512)            1049088
['conv5_block2_out[0][0]']
 D)



 conv5_block3_1_bn (BatchNo  (None, 7, 7, 512)            2048
['conv5_block3_1_conv[0][0]']
 rmalization)



 conv5_block3_1_relu (Activ  (None, 7, 7, 512)            0
['conv5_block3_1_bn[0][0]']
 ation)
```

```
 conv5_block3_2_conv (Conv2  (None, 7, 7, 512)            2359808
['conv5_block3_1_relu[0][0]']
 D)



 conv5_block3_2_bn (BatchNo  (None, 7, 7, 512)            2048
['conv5_block3_2_conv[0][0]']
 rmalization)



 conv5_block3_2_relu (Activ  (None, 7, 7, 512)            0
['conv5_block3_2_bn[0][0]']
 ation)



 conv5_block3_3_conv (Conv2  (None, 7, 7, 2048)           1050624
['conv5_block3_2_relu[0][0]']
 D)



 conv5_block3_3_bn (BatchNo  (None, 7, 7, 2048)           8192
['conv5_block3_3_conv[0][0]']
 rmalization)



 conv5_block3_add (Add)      (None, 7, 7, 2048)           0
['conv5_block2_out[0][0]',

'conv5_block3_3_bn[0][0]']


 conv5_block3_out (Activati  (None, 7, 7, 2048)           0
['conv5_block3_add[0][0]']
 on)



 avg_pool (GlobalAveragePoo  (None, 2048)                 0
['conv5_block3_out[0][0]']
 ling2D)



======================================================================
```

```
============================
Total params: 23587712 (89.98 MB)
Trainable params: 0 (0.00 Byte)
Non-trainable params: 23587712 (89.98 MB)
______________________________________________________________________
____________________________

def extract_features(data):
  features = []
  for image in data:
    preprocessed_image = preprocess(image)
    feature = base_model.predict(preprocessed_image,
verbose=0).reshape(2048)
    features.append(feature)
  return features

features = {}
for subset in ['train', 'validation', 'test']:
  data = dataset[subset]
  images = [row['image'] for row in data]
  features[subset] = extract_features(images)

print(features['train'][0].shape) # Shape of feature vector

(2048,)

# store features in pickle
pickle.dump(features, open(os.path.join(WORKING_DIR, 'features.pkl'),
'wb'))

# load features from pickle
with open(os.path.join(WORKING_DIR, 'features.pkl'), 'rb') as f:
    features = pickle.load(f)
```

Preprocessing Captions for RNN

```
# Load dataset into training, validation, and test sets
train_set = dataset["train"]
val_set = dataset["validation"]
test_set = dataset["test"]
```

Steps for text preprocessing:

- Convert sentences into lowercase.
- Remove all special characters and numbers present in the text.
- Remove extra spaces (multiple consecutive spaces).
- Remove single character words.
- Add a starting tag (e.g. startseq) and an ending tag (e.g. endseq) to each sentence to indicate the beginning and the ending of a sentence.

```
def preprocess_captions(captions):
    # Convert to lowercase
    captions = [caption.lower() for caption in captions]

    # Remove special characcters and numbers in text
    captions = [caption.replace("[^A-Za-z]","") for caption in
captions]

    # Remove extra spaces
    captions = [caption.replace("\s+"," ") for caption in captions]

    # Remove single character words
    captions = [' '.join(word for word in caption.split() if len(word)
> 1) for caption in captions]

    # Add start and end tokens
    captions_with_start_end = ['startseq ' + caption + ' endseq' for
caption in captions]

    # Fit tokenizer on text
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(captions_with_start_end)

    vocab_size = len(tokenizer.word_index) + 1

    max_len = max(len(caption.split()) for caption in
captions_with_start_end)

    return captions_with_start_end, tokenizer, vocab_size, max_len
```

Combine images and captions into a single dataframe with 2 columns: **"image"** which stores image index in the dataset, and **"caption"** which contains one of the captions of the corresponding image after cleaning.

```
def read_data_df(dataset):
  lst = []
  for i in range(len(dataset)):
      for caption_key in ['caption_0', 'caption_1', 'caption_2',
'caption_3', 'caption_4']:
        lst.append([i, dataset[i][caption_key]])

  df = pd.DataFrame(lst, columns=['image', 'caption'])
  return df

train_data = read_data_df(train_set)
train_data.head(10)
```


```
{"summary":"{\n  \"name\": \"train_data\",\n  \"rows\": 30000,\n
\"fields\": [\n    {\n      \"column\": \"image\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
```

1732,\n   \"min\": 0,\n   \"max\": 5999,\n
\"num_unique_values\": 6000,\n   \"samples\": [\n   1782,\
n   3917,\n   221\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"caption\",\n   \"properties\":
{\n   \"dtype\": \"string\",\n   \"num_unique_values\":
29837,\n   \"samples\": [\n   \"A white dog with long hair
wades through a pond surrounded by green grass .\",\n   \"A man
sitting outside with bags and a camera .\",\n   \"Two people
dance in costume .\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"train_data"}

Preprocessing captions of the training data and building the vocabulary of the model.

```
cleaned_captions, tokenizer, vocab_size, max_len =
preprocess_captions(train_data['caption'])
print('Vocabulary size:', vocab_size)
print('Maximum sequence length:', max_len)
train_data['caption'] = cleaned_captions
train_data.head(10)
```

```
Vocabulary size: 7368
Maximum sequence length: 34
```


{"summary":"{\n \"name\": \"train_data\",\n \"rows\": 30000,\n
\"fields\": [\n   {\n   \"column\": \"image\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
1732,\n   \"min\": 0,\n   \"max\": 5999,\n
\"num_unique_values\": 6000,\n   \"samples\": [\n   1782,\
n   3917,\n   221\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"caption\",\n   \"properties\":
{\n   \"dtype\": \"string\",\n   \"num_unique_values\":
29768,\n   \"samples\": [\n   \"startseq two men hanging
from ropes over waterfall endseq\",\n   \"startseq kids play in
the water endseq\",\n   \"startseq hiker waves to the camera as
he standing in front of snowcapped mountains endseq\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe","variable_name":"train_data"}

Preparing Output Labels

Since Image Caption model training like any other neural network training is a highly resource utillizing process we cannot load the data into the main memory all at once, and hence we need to generate the data in the required format batch wise.

The inputs will be the image embeddings and their corresonding caption text embeddings for the training process.

The output labels represent the next word in the sequence at each sub-iteration.

```
class CustomDataGenerator(Sequence):
  """
  Prepares output labels for a sequence prediction model.

  Returns:
      tuple: A tuple containing:
          - X1: Image features.
          - X2: Padded sequences.
          - y: Output labels.
  """
  def __init__(self, df, X_col, y_col, batch_size, tokenizer,
                vocab_size, max_length, features, shuffle=True):

      self.df = df
      self.X_col = X_col
      self.y_col = y_col
      self.batch_size = batch_size
      self.tokenizer = tokenizer
      self.vocab_size = vocab_size
      self.max_length = max_length
      self.features = features
      self.shuffle = shuffle
      self.indexes = np.arange(len(df))

  def on_epoch_end(self):
      if self.shuffle:
          np.random.shuffle(self.indexes)

  def __len__(self):
      return len(self.df) // self.batch_size

  def __getitem__(self, index):
      indexes = self.indexes[index * self.batch_size:(index + 1) *
self.batch_size]
      batch = self.df.iloc[indexes]
      X1, X2, y = self.__get_data(batch)
      return (X1, X2), y

  def __get_data(self, batch):
      X1, X2, y = [], [], []

      for _, row in batch.iterrows():
          image = row[self.X_col]
          feature = self.features[image]

          caption = row[self.y_col]
          seq = self.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=self.max_length,
padding='post')[0]
            out_seq = to_categorical([out_seq],
num_classes=self.vocab_size)[0]
            X1.append(feature)
            X2.append(in_seq)
            y.append(out_seq)

    return np.array(X1), np.array(X2), np.array(y)
```

Model Building

According to the survey paper "Where to put the Image in an Image Caption Generator", different image captioning models are as follows:

Our model combines both pre-inject and merge structures. The pre-inject part is mainly based on the paper **Show and Tell: A Neural Image Caption Generator**. Then we merge the image feautures with the LSTM output before passing the output through fully connected layers.

Model Breakdown:

1. **Input Layers:**
- For the image features, a dense layer is used to reduce the dimensionality to `embed_size` with 'ReLU' activation.
- For the input captions, an embedding layer is used to convert each word index into a dense representation of size `embed_size`.
1. **Concatenation:**
- The processed image features are concatenated with the embedded captions. This concatenated vector serves as the input to the LSTM layer. The image features then act as the first word in the input sequence.
- We apply teacher forcing during training, where the ground-truth words are fed as inputs to the LSTM at each time step during training, instead of using the model's own predictions.
1. **LSTM Layer:**
- We use an LSTM layer with 256 units to process the concatenated input sequence.
- The LSTM layer processes the concatenated input sequence and learns to generate the caption.
1. **Merging Layer:**
- The output of the LSTM is added to the processed image features. This operation combines the information from the image and the caption sequence, which was proven to improve the performance of the model.
1. **Output Layers:**
- After the LSTM layer, a dense layer is introduced with ReLU activation to further process the LSTM output.

- Finally, a dense layer with a softmax activation function to predict the next word in the caption vocabulary.

1. **Dropout Layers:**

- Dropout is applied to the LSTM output to prevent overfitting.
- Another Dropout layer is applied to the output of the first dense layer for regularization.

1. **Model Compilation:**

- We utilize cross entropy loss as the loss function while training the network.
- We set 'Adam' as the model's optimization technique.

Let's now define our `build_model` function, which is responsible for constructing the neural network, and returning it:

```
class ImageCaptioningHyperModel(HyperModel):
    def __init__(self, vocab_size, max_len):
        self.vocab_size = vocab_size
        self.max_len = max_len

    def build(self, hp):
        embed_size = hp.Choice('embed_size', values=[128, 256, 512])
        lstm_units = embed_size
        dropout_rate = 0.5
        dense_units = 128
        optimizer_choice = hp.Choice('optimizer', values=['adam',
'rmsprop'])

        if optimizer_choice == 'adam':
            optimizer = Adam()
        elif optimizer_choice == 'rmsprop':
            optimizer = RMSprop()
        else:
            optimizer = SGD()

        input1 = Input(shape=(2048,))
        input2 = Input(shape=(self.max_len,))

        img_features = Dense(embed_size, activation='relu')(input1)
        img_features_reshaped = Reshape((1, embed_size),
input_shape=(embed_size,))(img_features)

        sentence_features = Embedding(self.vocab_size, embed_size,
mask_zero=False)(input2)
        merged = concatenate([img_features_reshaped,
sentence_features], axis=1)
        sentence_features = LSTM(lstm_units)(merged)

        x = Dropout(dropout_rate)(sentence_features)
        x = add([x, img_features])
        x = Dense(dense_units, activation='relu')(x)
```

```
        x = Dropout(dropout_rate)(x)
        output = Dense(self.vocab_size, activation='softmax')(x)

        model = Model(inputs=[input1, input2], outputs=output)
        model.compile(loss='categorical_crossentropy',
optimizer=optimizer, metrics=['accuracy'])

        return model

hypermodel = ImageCaptioningHyperModel(vocab_size=vocab_size,
max_len=max_len)

tuner = Hyperband(
    hypermodel,
    objective='val_loss',
    max_epochs=10, # The maximum number of epochs to train one model.
This is the upper limit of the epoch that Hyperband will search
    factor=5,     # The reduction factor for the number of
configurations to train in each round of Hyperband
    directory='hyperparam_tuning',
    project_name='image_captioning'
)
```

```
Reloading Tuner from hyperparam_tuning/image_captioning/tuner0.json
```

```
tuner = RandomSearch(
    hypermodel,
    objective='val_loss',
    max_trials=10,
    executions_per_trial=1,
    directory='hyperparam_tuning',
    project_name='image_captioning'
)
```

```
/opt/conda/lib/python3.10/site-packages/keras/src/layers/reshaping/
reshape.py:39: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

```
tuner = GridSearch(
    hypermodel,
    objective='val_loss',
    max_trials=None,  # Grid Search will try all combinations
    executions_per_trial=1,
    directory='hyperparam_tuning',
    project_name='image_captioning'
)
```

```
/opt/conda/lib/python3.10/site-packages/keras/src/layers/reshaping/
reshape.py:39: UserWarning: Do not pass an `input_shape`/`input_dim`
```

```
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

Training

Load and preprocess validation data:

```
val_data = read_data_df(val_set)
val_cleaned_captions, _, _, _ =
preprocess_captions(val_data['caption'])
val_data['caption'] = val_cleaned_captions
val_data.head(10)
```


```
{"summary":"{\n  \"name\": \"val_data\",\n  \"rows\": 5000,\n
\"fields\": [\n    {\n      \"column\": \"image\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
288,\n        \"min\": 0,\n        \"max\": 999,\n
\"num_unique_values\": 1000,\n        \"samples\": [\n          521,\n
737,\n          740\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"caption\",\n      \"properties\": {\n        \"dtype\": \"string\",\
n        \"num_unique_values\": 4994,\n        \"samples\": [\n
\"startseq two teenage girls hugging one wearing bicycle helmet with
cyclists in the background endseq\",\n          \"startseq man jumps
in the air while sky surfing endseq\",\n          \"startseq black and
white bird with orange beak in water with rocks around endseq\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe","variable_name":"val_data"}
```


Initialize the train and validation generators with `batch size = 64`:

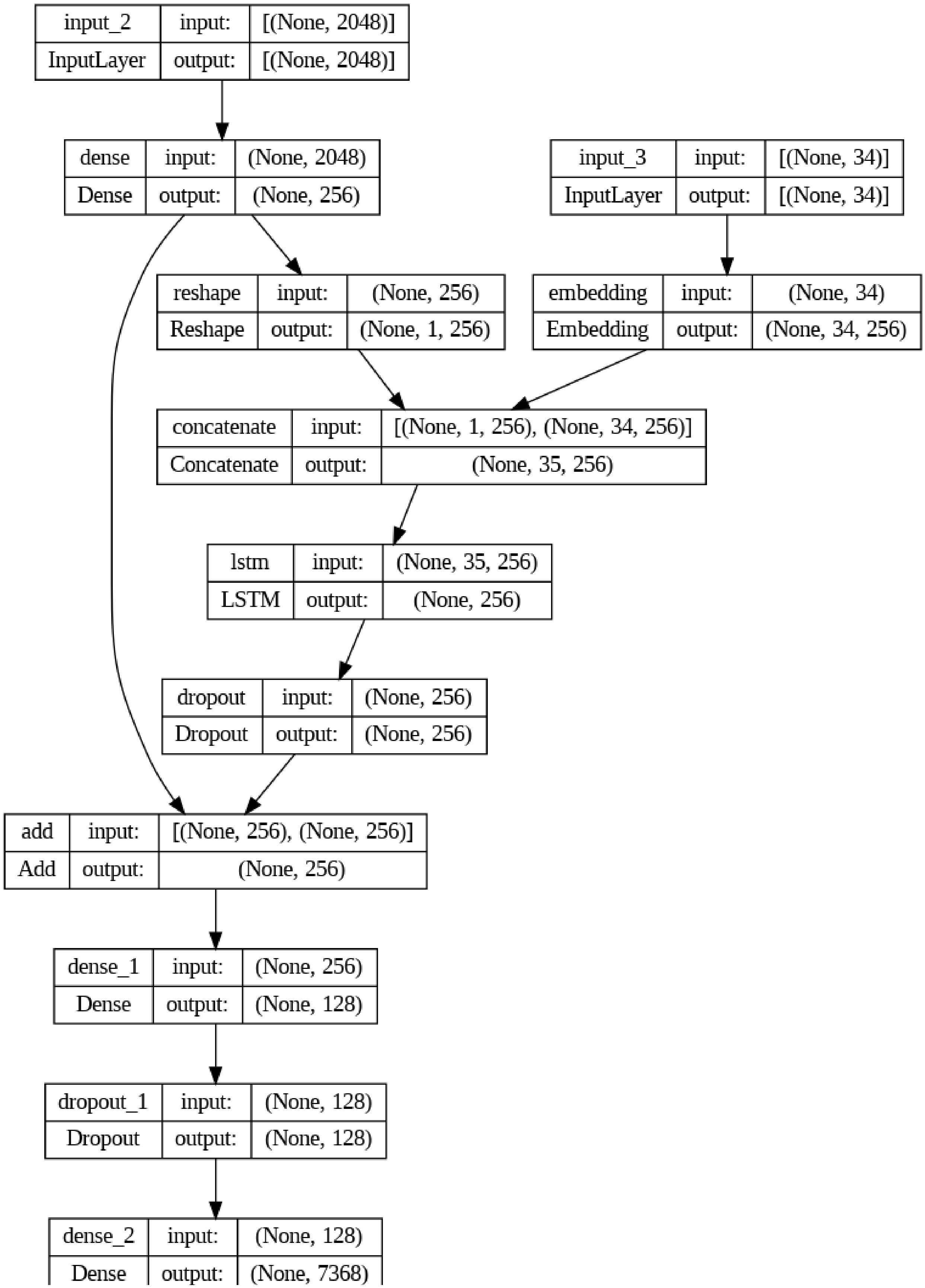
```
train_generator = CustomDataGenerator(df=train_data, X_col='image',
y_col='caption', batch_size=64,
                                      tokenizer=tokenizer,
vocab_size=vocab_size, max_length=max_len,
                                      features=features['train'])

validation_generator = CustomDataGenerator(df=val_data, X_col='image',
y_col='caption', batch_size=64,
                                           tokenizer=tokenizer,
vocab_size=vocab_size, max_length=max_len,

features=features['validation'])
```

We define our checkpoints and early stopping mechanism to stop training when overfitting is detected:

```
model_name = "model.keras"
checkpoint = ModelCheckpoint(model_name,
                            monitor="val_loss",
                            mode="min",
                            save_best_only = True,
                            verbose=1)

earlystopping = EarlyStopping(monitor='val_loss',min_delta = 0,
patience = 5, verbose = 1, restore_best_weights=True)

learning_rate_reduction = ReduceLROnPlateau(monitor='val_loss',
                                            patience=3,
                                            verbose=1,
                                            factor=0.2,
                                            min_lr=0.00000001)
```

Adjust the hyperparameters to determine the optimal values for building the model. Then, use the best model for fitting the data.

```
tuner.search(train_generator,
             validation_data=validation_generator,
             epochs=10,
             callbacks=[checkpoint, earlystopping])

best_hyperparameters = tuner.get_best_hyperparameters(num_trials=1)[0]

best_model = tuner.hypermodel.build(best_hyperparameters)

Trial 6 Complete [00h 10m 47s]
val_loss: 4.374006748199463

Best val_loss So Far: 3.7370548248291016
Total elapsed time: 01h 00m 52s

print("Best Hyperparameters:")
for param in best_hyperparameters.values:
    print(f"{param}: {best_hyperparameters.get(param)}")

Best Hyperparameters:
embed_size: 256
optimizer: adam

plot_model(best_model, show_shapes=True)
```


input_2
InputLayer
input:
output:
[(None, 2048)]
[(None, 2048)]
dense
Dense
input:
output:
(None, 2048)
(None, 256)
input_3
InputLayer
input:
output:
[(None, 34)]
[(None, 34)]
reshape
Reshape
input:
output:
(None, 256)
(None, 1, 256)
embedding
Embedding
input:
output:
(None, 34)
(None, 34, 256)
concatenate
Concatenate
input:
output:
[(None, 1, 256), (None, 34, 256)]
(None, 35, 256)
lstm
LSTM
input:
output:
(None, 35, 256)
(None, 256)
dropout
Dropout
input:
output:
(None, 256)
(None, 256)
add
Add
input:
output:
[(None, 256), (None, 256)]
(None, 256)
dense_1
Dense
input:
output:
(None, 256)
(None, 128)
dropout_1
Dropout
input:
output:
(None, 128)
(None, 128)
dense_2
Dense
input:
output:
(None, 128)
(None, 7368)

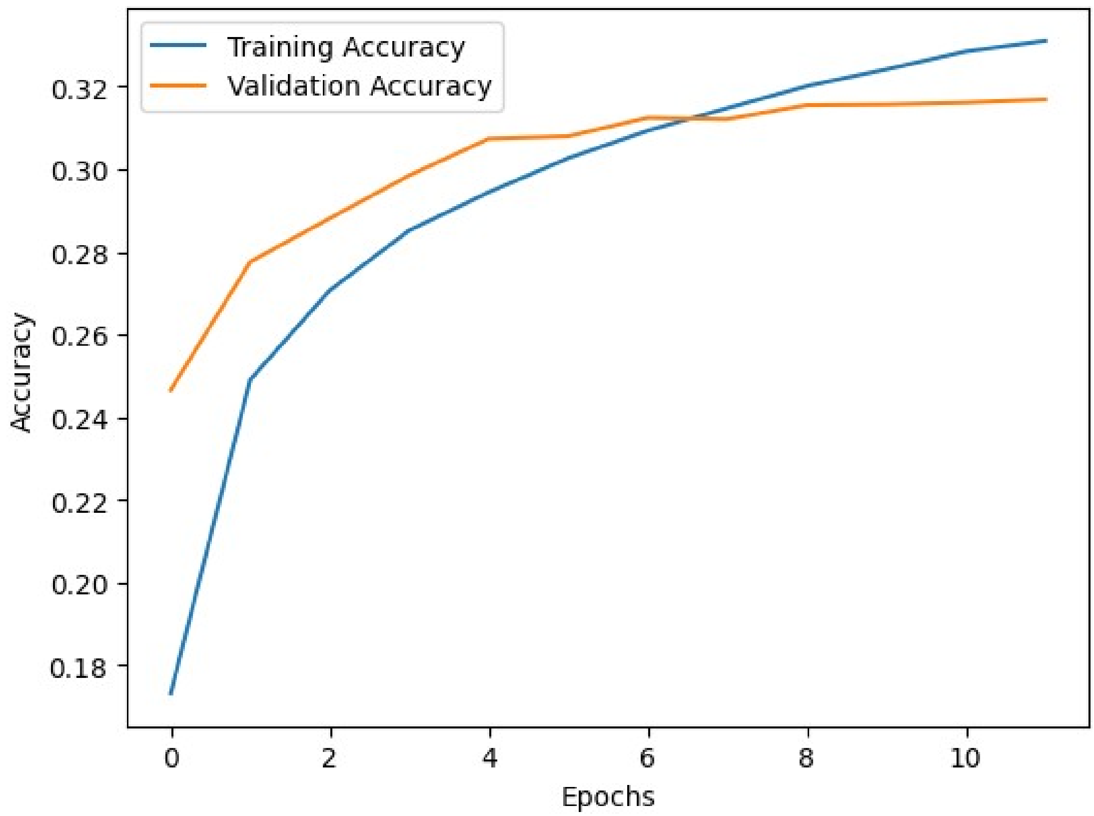
After building the model, the model is fit using the training dataset. The model is made to run for 50 epochs and the best model is chosen among the 50 epochs by computing loss function on validation dataset. The model with the lowest loss value is chosen for generating captions.

```
history = best_model.fit(
        train_generator,
        validation_data = validation_generator,
        callbacks=[checkpoint, earlystopping],
        epochs=50)
```

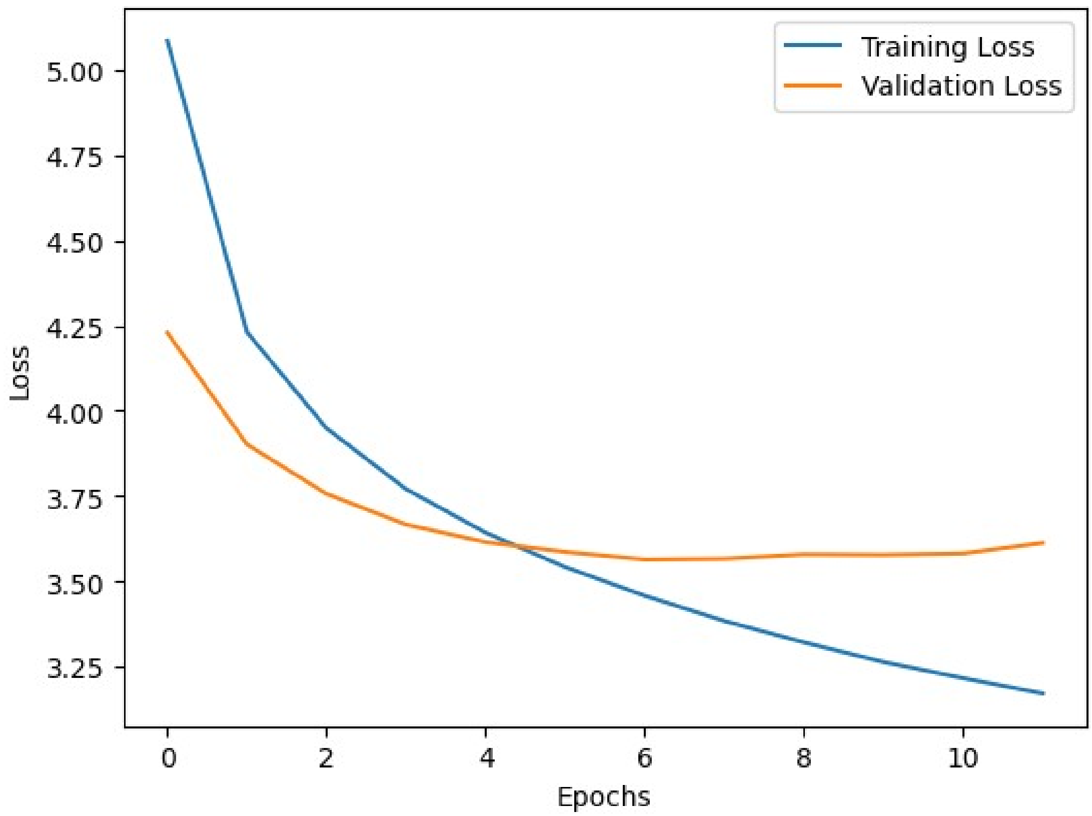
```
Epoch 1/50
468/468 ━━━━━━━━━━━━━━━━━━━━ 0s 107ms/step - accuracy: 0.1292 - loss:
5.6866
Epoch 1: val_loss improved from inf to 4.22917, saving model to
model.keras
468/468 ━━━━━━━━━━━━━━━━━━━━ 61s 126ms/step - accuracy: 0.1293 - loss:
5.6854 - val_accuracy: 0.2466 - val_loss: 4.2292
Epoch 2/50
468/468 ━━━━━━━━━━━━━━━━━━━━ 0s 107ms/step - accuracy: 0.2429 - loss:
4.2919
Epoch 2: val_loss improved from 4.22917 to 3.90175, saving model to
model.keras
468/468 ━━━━━━━━━━━━━━━━━━━━ 59s 125ms/step - accuracy: 0.2429 - loss:
4.2918 - val_accuracy: 0.2775 - val_loss: 3.9017
Epoch 3/50
468/468 ━━━━━━━━━━━━━━━━━━━━ 0s 107ms/step - accuracy: 0.2667 - loss:
3.9836
Epoch 3: val_loss improved from 3.90175 to 3.75650, saving model to
model.keras
468/468 ━━━━━━━━━━━━━━━━━━━━ 59s 125ms/step - accuracy: 0.2667 - loss:
3.9835 - val_accuracy: 0.2881 - val_loss: 3.7565
Epoch 4/50
468/468 ━━━━━━━━━━━━━━━━━━━━ 0s 108ms/step - accuracy: 0.2833 - loss:
3.7787
Epoch 4: val_loss improved from 3.75650 to 3.66641, saving model to
model.keras
468/468 ━━━━━━━━━━━━━━━━━━━━ 59s 124ms/step - accuracy: 0.2833 - loss:
3.7787 - val_accuracy: 0.2985 - val_loss: 3.6664
Epoch 5/50
468/468 ━━━━━━━━━━━━━━━━━━━━ 0s 104ms/step - accuracy: 0.2940 - loss:
3.6370
Epoch 5: val_loss improved from 3.66641 to 3.61489, saving model to
model.keras
468/468 ━━━━━━━━━━━━━━━━━━━━ 58s 122ms/step - accuracy: 0.2940 - loss:
3.6370 - val_accuracy: 0.3073 - val_loss: 3.6149
Epoch 6/50
467/468 ━━━━━━━━━━━━━━━━━━━━ 0s 108ms/step - accuracy: 0.3023 - loss:
3.5361
Epoch 6: val_loss improved from 3.61489 to 3.58622, saving model to
model.keras
```

```
468/468 ━━━━━━━━━━━━━━━━━━━━ 59s 125ms/step - accuracy: 0.3023 - loss:
3.5362 - val_accuracy: 0.3079 - val_loss: 3.5862
Epoch 7/50
468/468 ━━━━━━━━━━━━━━━━━━━━ 0s 108ms/step - accuracy: 0.3096 - loss:
3.4464
Epoch 7: val_loss improved from 3.58622 to 3.56405, saving model to
model.keras
468/468 ━━━━━━━━━━━━━━━━━━━━ 59s 125ms/step - accuracy: 0.3096 - loss:
3.4464 - val_accuracy: 0.3124 - val_loss: 3.5641
Epoch 8/50
467/468 ━━━━━━━━━━━━━━━━━━━━ 0s 108ms/step - accuracy: 0.3154 - loss:
3.3725
Epoch 8: val_loss did not improve from 3.56405
468/468 ━━━━━━━━━━━━━━━━━━━━ 59s 125ms/step - accuracy: 0.3154 - loss:
3.3725 - val_accuracy: 0.3121 - val_loss: 3.5662
Epoch 9/50
468/468 ━━━━━━━━━━━━━━━━━━━━ 0s 107ms/step - accuracy: 0.3210 - loss:
3.3077
Epoch 9: val_loss did not improve from 3.56405
468/468 ━━━━━━━━━━━━━━━━━━━━ 59s 125ms/step - accuracy: 0.3210 - loss:
3.3078 - val_accuracy: 0.3155 - val_loss: 3.5791
Epoch 10/50
467/468 ━━━━━━━━━━━━━━━━━━━━ 0s 106ms/step - accuracy: 0.3250 - loss:
3.2503
Epoch 10: val_loss did not improve from 3.56405
468/468 ━━━━━━━━━━━━━━━━━━━━ 59s 124ms/step - accuracy: 0.3250 - loss:
3.2504 - val_accuracy: 0.3156 - val_loss: 3.5777
Epoch 11/50
468/468 ━━━━━━━━━━━━━━━━━━━━ 0s 106ms/step - accuracy: 0.3288 - loss:
3.2017
Epoch 11: val_loss did not improve from 3.56405
468/468 ━━━━━━━━━━━━━━━━━━━━ 59s 123ms/step - accuracy: 0.3288 - loss:
3.2017 - val_accuracy: 0.3161 - val_loss: 3.5817
Epoch 12/50
467/468 ━━━━━━━━━━━━━━━━━━━━ 0s 109ms/step - accuracy: 0.3323 - loss:
3.1517
Epoch 12: val_loss did not improve from 3.56405
468/468 ━━━━━━━━━━━━━━━━━━━━ 60s 126ms/step - accuracy: 0.3322 - loss:
3.1518 - val_accuracy: 0.3169 - val_loss: 3.6125
Epoch 12: early stopping
Restoring model weights from the end of the best epoch: 7.
```

```
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```




```
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Inference

Here the image embeddings are passed along with the first word to the model. The predicted word is then added to the previous sequence and passed to generate the next word. The process continues till the reaching the end tag or the maximum sequence length.

```
def idx_to_word(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None

# generate caption for an image
def predict_caption(model, image, tokenizer, max_length):
    in_text = 'startseq'

    # iterate over the max length of sequence
    for i in range(max_length):
        # encode input sequence
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence], max_length)
```

```
        yhat = model.predict([image, sequence], verbose=0)
        # get index with high probability
        yhat = np.argmax(yhat)
        word = idx_to_word(yhat, tokenizer)

        # stop if word not found
        if word is None:
            break

        # append word as input for generating next word
        in_text += " " + word

        # stop if we reach end tag
        if word == 'endseq':
            break

    return in_text
```

Evaluation

Quantitative Results

Report loss, accuracy and bleu score on test data:

```
# Load model
best_model = tuner.hypermodel.build(best_hyperparameters)

# Load model weights
best_model.load_weights(os.path.join(WORKING_DIR,
'model_weights.weights.h5'))
```

```
WARNING:absl:Skipping variable loading for optimizer 'Adam', because
it has 1 variables whereas the saved optimizer has 22 variables.
```

```
# Load and preprocess test data
test_data = read_data_df(test_set)
test_cleaned_captions, _, _, _ =
preprocess_captions(test_data['caption'])
test_data['caption'] = test_cleaned_captions
test_data.head(10)
```


{"summary":"{\n \"name\": \"test_data\",\n \"rows\": 5000,\n
\"fields\": [\n   {\n   \"column\": \"image\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
288,\n   \"min\": 0,\n   \"max\": 999,\n
\"num_unique_values\": 1000,\n   \"samples\": [\n   521,\n
737,\n   740\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"caption\",\n   \"properties\": {\n   \"dtype\": \"string\",\

```
n        \"num_unique_values\": 4991,\n        \"samples\": [\n
\"startseq black and white dog is chasing ping frisbee endseq\",\n
\"startseq two lean dogs one brown and white and one black and white
run together endseq\",\n          \"startseq brown dog jumps through
three hoops endseq\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"test_data"}
```


```
test_generator = CustomDataGenerator(df=test_data, X_col='image',
y_col='caption', batch_size=64,
                                    tokenizer=tokenizer,
vocab_size=vocab_size, max_length=max_len,
                                    features=features['test'],
shuffle=False)

# Evaluate the model on the test data
results = best_model.evaluate(test_generator)
print(f"Test Loss: {results[0]}")
print(f"Test Accuracy: {results[1]*100}%")
```

```
78/78 ━━━━━━━━━━━━━━━━━━━━ 9s 115ms/step - accuracy: 0.3200 - loss:
3.4906
Test Loss: 3.481898546218872
Test Accuracy: 32.18182623386383%
```

```
def evaluate_model(model, data, tokenizer, max_length, features):
  actual, predicted = list(), list()

  # step over the whole set
  for idx in range(len(features)):
    feature = features[idx]
    captions = data.loc[data['image']==idx, 'caption'].tolist()

    # generate description
    yhat = predict_caption(model, np.array([feature]), tokenizer,
max_length)

    references = [caption.split() for caption in captions]
    actual.append(references)
    predicted.append(yhat.split())

  # calculate BLEU score
  print('BLEU-1: %f' % corpus_bleu(actual, predicted, weights=(1.0, 0,
0, 0)))
  print('BLEU-2: %f' % corpus_bleu(actual, predicted, weights=(0.5,
0.5, 0, 0)))
  print('BLEU-3: %f' % corpus_bleu(actual, predicted, weights=(0.3,
0.3, 0.3, 0)))
  print('BLEU-4: %f' % corpus_bleu(actual, predicted, weights=(0.25,
0.25, 0.25, 0.25)))
```

```
evaluate_model(best_model, test_data, tokenizer, max_len,
features['test'])
```

```
BLEU-1: 0.631338
BLEU-2: 0.396658
BLEU-3: 0.286683
BLEU-4: 0.151087
```

Qualitative Results

```
def display_images_with_captions(images, captions, cols=2):
    rows = (len(images) + cols - 1) // cols
    fig, axes = plt.subplots(rows, cols, figsize=(15, 5*rows))
    axes = axes.flatten()

    for img, caption, ax in zip(images, captions, axes):
        #img = Image.open(img_path)
        ax.imshow(img)
        ax.axis('off')
        ax.set_title(caption)

    for ax in axes[len(images):]:
        fig.delaxes(ax)

    plt.tight_layout()
    plt.show()

def display_images_from_dir(image_dir):
  # Get image paths
  images = [load_img(os.path.join(image_dir, img_name)) for img_name
in os.listdir(image_dir)]
  captions = []

  for img in images:
    processed_image = preprocess(img)
    image_features = base_model.predict(processed_image, verbose=0)
    caption = predict_caption(model, image_features, tokenizer,
max_len)
    captions.append(caption)

  display_images_with_captions(images, captions, cols=2)
```

Test on images from the test set:

```
samples = test_data.sample(16)
samples.reset_index(drop=True, inplace=True)

for index, record in samples.iterrows():
  id = record['image']
  image = features['test'][id]
```

```
  caption = predict_caption(model, np.array([image]), tokenizer,
max_len)
  samples.loc[index, 'image'] = test_set['image'][id]
  samples.loc[index,'caption'] = caption

display_images_with_captions(samples['image'], samples['caption'], 2)
```

startseq young boy is swinging on swing endseq

startseq man in blue shirt is standing on the edge of rock endseq

startseq two people are playing in the water endseq

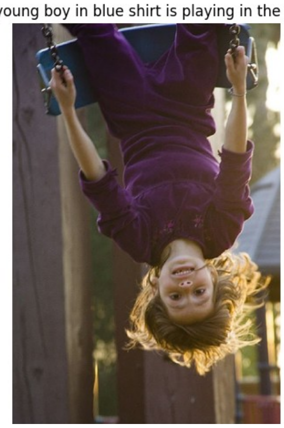
startseq dog is running through the water endseq

startseq man is jumping over the edge of the water endseq

startseq two dogs are running in the snow endseq

startseq dog is running through snow endseq

startseq young boy in blue shirt is playing in the air endseq



Test on images from ImageNet dataset on which the CNN model is pretrained on:

```
print('=========================== ImageNet Dataset ===========================\n')
image_dir = os.path.join(WORKING_DIR, 'test_images/ImageNet')
display_images_from_dir(image_dir)
```

```
=========================== ImageNet Dataset ===========================
```

startseq man is standing on the water endseq

startseq dog is licking its nose endseq

startseq dog is running through the grass endseq

startseq brown dog is running in the snow endseq

startseq man in red shirt is standing on the street endseq

startseq man in blue shirt is standing on the water endseq

startseq the man is standing on rock endseq

startseq man is sitting on rock endseq

Test on new images from COCO dataset:

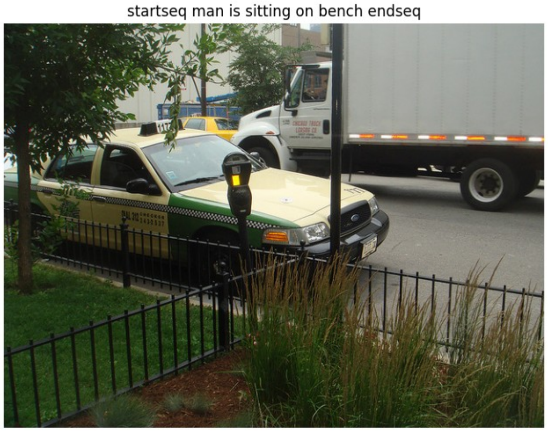
```
print('=========================== COCO Dataset
===========================\n')
image_dir = os.path.join(WORKING_DIR, 'test_images/COCO')
display_images_from_dir(image_dir)
```

```
===========================COCO Dataset===========================
```

startseq dog is standing on the water endseq

startseq two dogs are running through the grass endseq

startseq man is sitting on bench endseq



startseq boy in blue shirt is playing baseball endseq

startseq two girls are walking on the beach endseq

startseq group of people are standing on the snow endseq

startseq dog is running through the water endseq

startseq man in black shirt is sitting on the sidewalk endseq

Results:

- As we can clearly see there is some redundant caption generation e.g. Dog for any animal, man for unidentified objects, overusage of 'shirt' for any other type of cloth.
- The model confuses many colors, with the overusage of red and blue colors.
- The model also fails sometimes to clearly specify the number of people in the image e.g. two while only one person exits, one or two for a group of people.
- When there are multiple objects in an image, the model sometimes fails to recognize all of them especially those farther away from the camera.
- The model correctly identifies dogs, grass, snow, water, and types of sports in most cases.
- The model performance can be further improved by training on more data and using attention mechanism so that our model can focus on relevant areas during the text generation.

Comments

Performance:

- The model has clearly overfit, possibly due to less amount of data.
- We used dropout layers, regularization, learning rate scheduling, and decreasing model's complexity to overcome overftitting but none of these methods seemed to achieve any improvement.
- The model achieves high loss value (~3.6) and low accuracy (~31%) on validation data.
- Nevertheless, the model acheives a good bleu 1 score (~0.63).

Areas of Improvement:

- We can tackle the overfitting problem in two ways:
  - Train the model on a larger dataset e.g. Flickr30k
  - Attention Models
- Increase the training time (number of epochs).
- Use beam search to generate captions for a better performance.

Attention Mechanism

The attention mechanism is a technique used in deep learning that focuses on different parts of an input sequence when predicting the next part of the sequence. It is widely used in sequence-to-sequence models, such as machine translation and image captioning, to improve the model's performance.

Attention Model Architecture

Steps to Add Attention Mechanism:

Define the Attention Layer: This layer will calculate the attention scores and context vectors.

Compute Attention Weights: Using the attention layer to compute the weights.

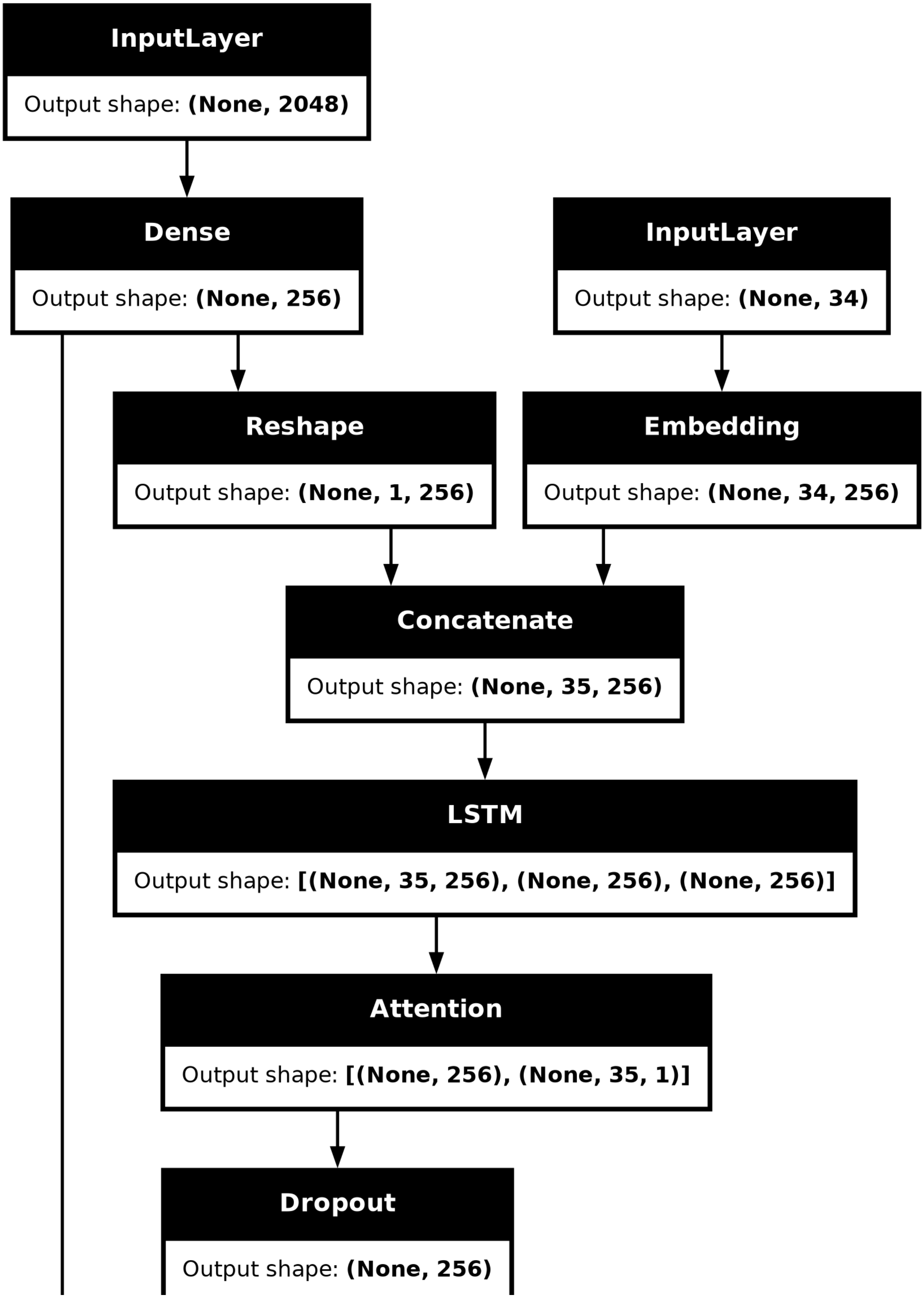
Apply the Attention Weights: Applying the attention weights to the encoder outputs to obtain a context vector.

Concatenate the Context Vector: Combine the context vector with the decoder input.

Proceed with the Decoder: Continue with the decoder as originally.

```
class Attention(Layer):
    def __init__(self, units):
        super(Attention, self).__init__()
        self.W1 = Dense(units)
        self.W2 = Dense(units)
        self.V = Dense(1)

    def call(self, features, hidden):
        # features shape: (batch_size, max_len, embedding_dim)
        # hidden shape: (batch_size, embedding_dim)
        hidden_with_time_axis = tf.expand_dims(hidden, 1)

        # Score shape: (batch_size, max_len, 1)
        score = tf.nn.tanh(self.W1(features) +
self.W2(hidden_with_time_axis))
        attention_weights = tf.nn.softmax(self.V(score), axis=1)

        # Context vector shape: (batch_size, embedding_dim)
        context_vector = attention_weights * features
        context_vector = tf.reduce_sum(context_vector, axis=1)

        return context_vector, attention_weights

def build_model(embed_size, max_len, vocab_size):
    input1 = Input(shape=(2048,))
    input2 = Input(shape=(max_len,))

    # Image feature branch
    img_features = Dense(embed_size, activation='relu')(input1)
    img_features_reshaped = Reshape((1, embed_size))(img_features)

    # Sentence feature branch
    sentence_features = Embedding(vocab_size, embed_size,
mask_zero=False)(input2)

    # Concatenate image and sentence features
    merged = concatenate([img_features_reshaped, sentence_features],
axis=1)

    # LSTM layer
    lstm_output, state_h, state_c = LSTM(256, return_sequences=True,
return_state=True)(merged)

    # Attention layer
```

```
    attention = Attention(256)
    context_vector, attention_weights = attention(lstm_output,
state_h)

    # Add attention context vector to LSTM output
    x = Dropout(0.5)(context_vector)
    x = add([x, img_features])
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.5)(x)
    output = Dense(vocab_size, activation='softmax')(x)

    model = Model(inputs=[input1, input2], outputs=output)
    model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

    return model

model = build_model(256, max_len, vocab_size)
plot_model(model, show_shapes=True)
```


InputLayer
Output shape: (None, 2048)
Dense
Output shape: (None, 256)
InputLayer
Output shape: (None, 34)
Reshape
Output shape: (None, 1, 256)
Embedding
Output shape: (None, 34, 256)
Concatenate
Output shape: (None, 35, 256)
LSTM
Output shape: [(None, 35, 256), (None, 256), (None, 256)]
Attention
Output shape: [(None, 256), (None, 35, 1)]
Dropout
Output shape: (None, 256)

```python
model_name = "attention.keras"
checkpoint = ModelCheckpoint(model_name,
                            monitor="val_loss",
                            mode="min",
                            save_best_only = True,
                            verbose=1)

 history = model.fit(
        train_generator,
        validation_data = validation_generator,
        callbacks=[checkpoint, earlystopping],
        epochs=50)
```

```
Epoch 1/50
```

```
/opt/conda/lib/python3.10/site-packages/keras/src/trainers/
data_adapters/py_dataset_adapter.py:120: UserWarning: Your `PyDataset`
class should call `super().__init__(**kwargs)` in its constructor.
`**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will
be ignored.
  self._warn_if_super_not_called()
```

```
468/468 ━━━━━━━━━━━━━━━━━━━━ 0s 107ms/step - accuracy: 0.1154 - loss:
5.8511
Epoch 1: val_loss improved from inf to 4.50086, saving model to
attention.keras
468/468 ━━━━━━━━━━━━━━━━━━━━ 64s 126ms/step - accuracy: 0.1155 - loss:
5.8499 - val_accuracy: 0.2054 - val_loss: 4.5009
Epoch 2/50
468/468 ━━━━━━━━━━━━━━━━━━━━ 0s 106ms/step - accuracy: 0.2121 - loss:
4.5137
Epoch 2: val_loss improved from 4.50086 to 4.00417, saving model to
attention.keras
468/468 ━━━━━━━━━━━━━━━━━━━━ 59s 124ms/step - accuracy: 0.2121 - loss:
4.5135 - val_accuracy: 0.2679 - val_loss: 4.0042
Epoch 3/50
468/468 ━━━━━━━━━━━━━━━━━━━━ 0s 104ms/step - accuracy: 0.2570 - loss:
4.0717
Epoch 3: val_loss improved from 4.00417 to 3.80573, saving model to
attention.keras
468/468 ━━━━━━━━━━━━━━━━━━━━ 58s 122ms/step - accuracy: 0.2570 - loss:
4.0716 - val_accuracy: 0.2837 - val_loss: 3.8057
Epoch 4/50
468/468 ━━━━━━━━━━━━━━━━━━━━ 0s 105ms/step - accuracy: 0.2745 - loss:
3.8616
Epoch 4: val_loss improved from 3.80573 to 3.71526, saving model to
attention.keras
468/468 ━━━━━━━━━━━━━━━━━━━━ 58s 123ms/step - accuracy: 0.2745 - loss:
3.8616 - val_accuracy: 0.2920 - val_loss: 3.7153
```

```
Epoch 5/50
468/468 ━━━━━━━━━━━━━━━━━━━━ 0s 106ms/step - accuracy: 0.2880 - loss:
3.7051
Epoch 5: val_loss improved from 3.71526 to 3.64269, saving model to
attention.keras
468/468 ━━━━━━━━━━━━━━━━━━━━ 59s 123ms/step - accuracy: 0.2880 - loss:
3.7051 - val_accuracy: 0.3000 - val_loss: 3.6427
Epoch 6/50
468/468 ━━━━━━━━━━━━━━━━━━━━ 0s 106ms/step - accuracy: 0.2984 - loss:
3.5792
Epoch 6: val_loss improved from 3.64269 to 3.62534, saving model to
attention.keras
468/468 ━━━━━━━━━━━━━━━━━━━━ 59s 124ms/step - accuracy: 0.2984 - loss:
3.5792 - val_accuracy: 0.3021 - val_loss: 3.6253
Epoch 7/50
468/468 ━━━━━━━━━━━━━━━━━━━━ 0s 105ms/step - accuracy: 0.3055 - loss:
3.4906
Epoch 7: val_loss improved from 3.62534 to 3.58753, saving model to
attention.keras
468/468 ━━━━━━━━━━━━━━━━━━━━ 58s 123ms/step - accuracy: 0.3055 - loss:
3.4906 - val_accuracy: 0.3080 - val_loss: 3.5875
Epoch 8/50
468/468 ━━━━━━━━━━━━━━━━━━━━ 0s 107ms/step - accuracy: 0.3136 - loss:
3.4082
Epoch 8: val_loss improved from 3.58753 to 3.57507, saving model to
attention.keras
468/468 ━━━━━━━━━━━━━━━━━━━━ 59s 125ms/step - accuracy: 0.3136 - loss:
3.4082 - val_accuracy: 0.3115 - val_loss: 3.5751
Epoch 9/50
468/468 ━━━━━━━━━━━━━━━━━━━━ 0s 107ms/step - accuracy: 0.3186 - loss:
3.3425
Epoch 9: val_loss did not improve from 3.57507
468/468 ━━━━━━━━━━━━━━━━━━━━ 59s 124ms/step - accuracy: 0.3186 - loss:
3.3425 - val_accuracy: 0.3136 - val_loss: 3.5792
Epoch 10/50
468/468 ━━━━━━━━━━━━━━━━━━━━ 0s 107ms/step - accuracy: 0.3231 - loss:
3.2713
Epoch 10: val_loss did not improve from 3.57507
468/468 ━━━━━━━━━━━━━━━━━━━━ 59s 124ms/step - accuracy: 0.3231 - loss:
3.2713 - val_accuracy: 0.3166 - val_loss: 3.5824
Epoch 11/50
468/468 ━━━━━━━━━━━━━━━━━━━━ 0s 109ms/step - accuracy: 0.3321 - loss:
3.2067
Epoch 11: val_loss did not improve from 3.57507
468/468 ━━━━━━━━━━━━━━━━━━━━ 60s 126ms/step - accuracy: 0.3321 - loss:
3.2068 - val_accuracy: 0.3155 - val_loss: 3.5828
Epoch 12/50
468/468 ━━━━━━━━━━━━━━━━━━━━ 0s 106ms/step - accuracy: 0.3346 - loss:
3.1589
```

```
Epoch 12: val_loss did not improve from 3.57507
468/468 ━━━━━━━━━━━━━━━━━━━━ 57s 120ms/step - accuracy: 0.3346 - loss:
3.1590 - val_accuracy: 0.3185 - val_loss: 3.6121
Epoch 13/50
468/468 ━━━━━━━━━━━━━━━━━━━━ 0s 107ms/step - accuracy: 0.3396 - loss:
3.1137
Epoch 13: val_loss did not improve from 3.57507
468/468 ━━━━━━━━━━━━━━━━━━━━ 59s 124ms/step - accuracy: 0.3396 - loss:
3.1137 - val_accuracy: 0.3161 - val_loss: 3.6071
Epoch 13: early stopping
Restoring model weights from the end of the best epoch: 8.
```

```python
print("start")
evaluate_model(model, test_data, tokenizer, max_len, features['test'])
print("finished")
```

```
start
BLEU-1: 0.343246
BLEU-2: 0.204782
BLEU-3: 0.134640
BLEU-4: 0.058412
finished
```

References

- https://huggingface.co/datasets/jxie/flickr8k
- https://rupamgoyal12.medium.com/image-caption-generator-using-resnet50-and-lstm-model-a5b11f60cd23
- https://www.geeksforgeeks.org/residual-networks-resnet-deep-learning/
- https://stackoverflow.com/questions/47555829/preprocess-input-method-in-keras
- https://machinelearningmastery.com/calculate-bleu-score-for-text-python/
- https://luckytoilet.wordpress.com/2018/03/22/i-trained-a-neural-network-to-describe-pictures-and-its-hilariously-bad/
- https://www.kaggle.com/code/ramoliyafenil/image-captioning#Modelling
- https://www.analyticsvidhya.com/blog/2019/11/comprehensive-guide-attention-mechanism-deep-learning/
- https://towardsdatascience.com/foundations-of-nlp-explained-bleu-score-and-wer-metrics-1a5ba06d812b