Image captioning

SUBJECT : ADVANCE ML

GUIDE: PROF. RACHIT CHAYA

ARJUN |202211036

NIKUNJ | 202211014

DIPEN | 202211054

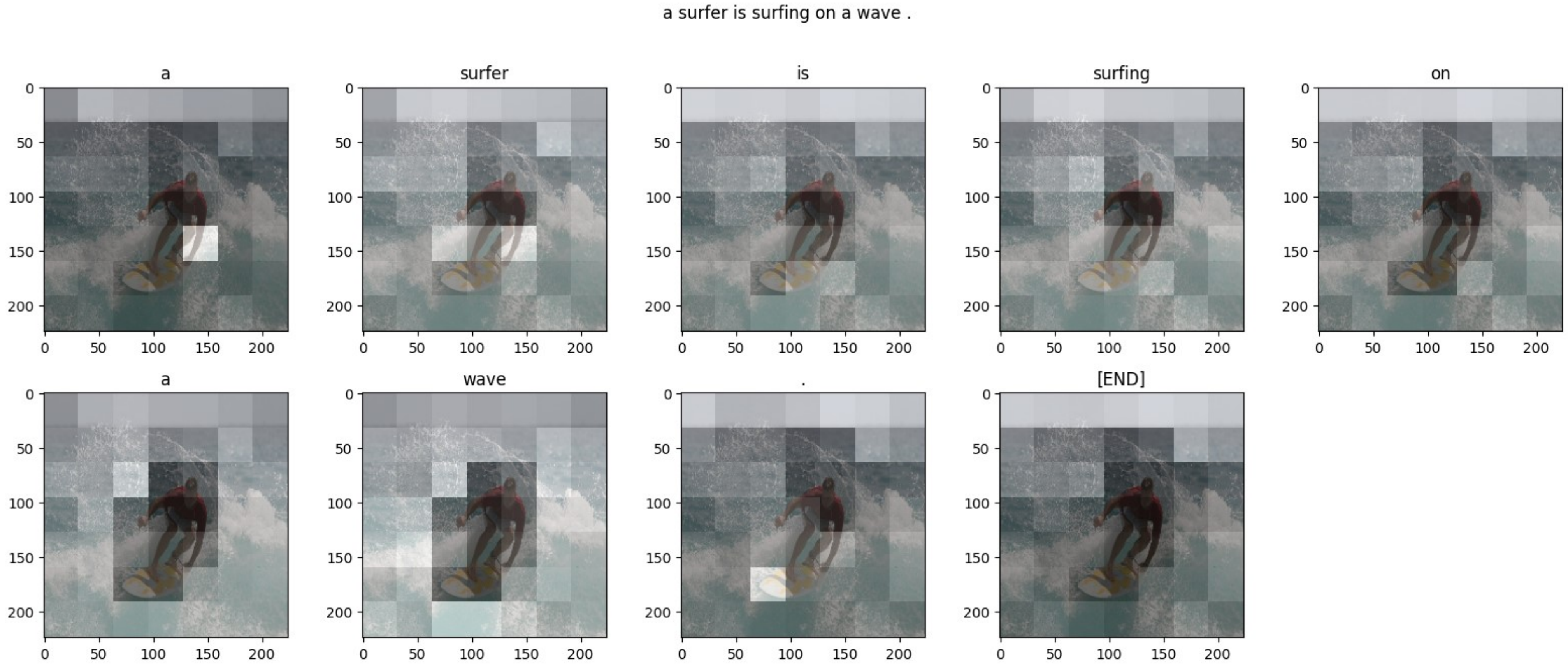
VIVEK | 202211069

Image Captioning (CV+NLP)

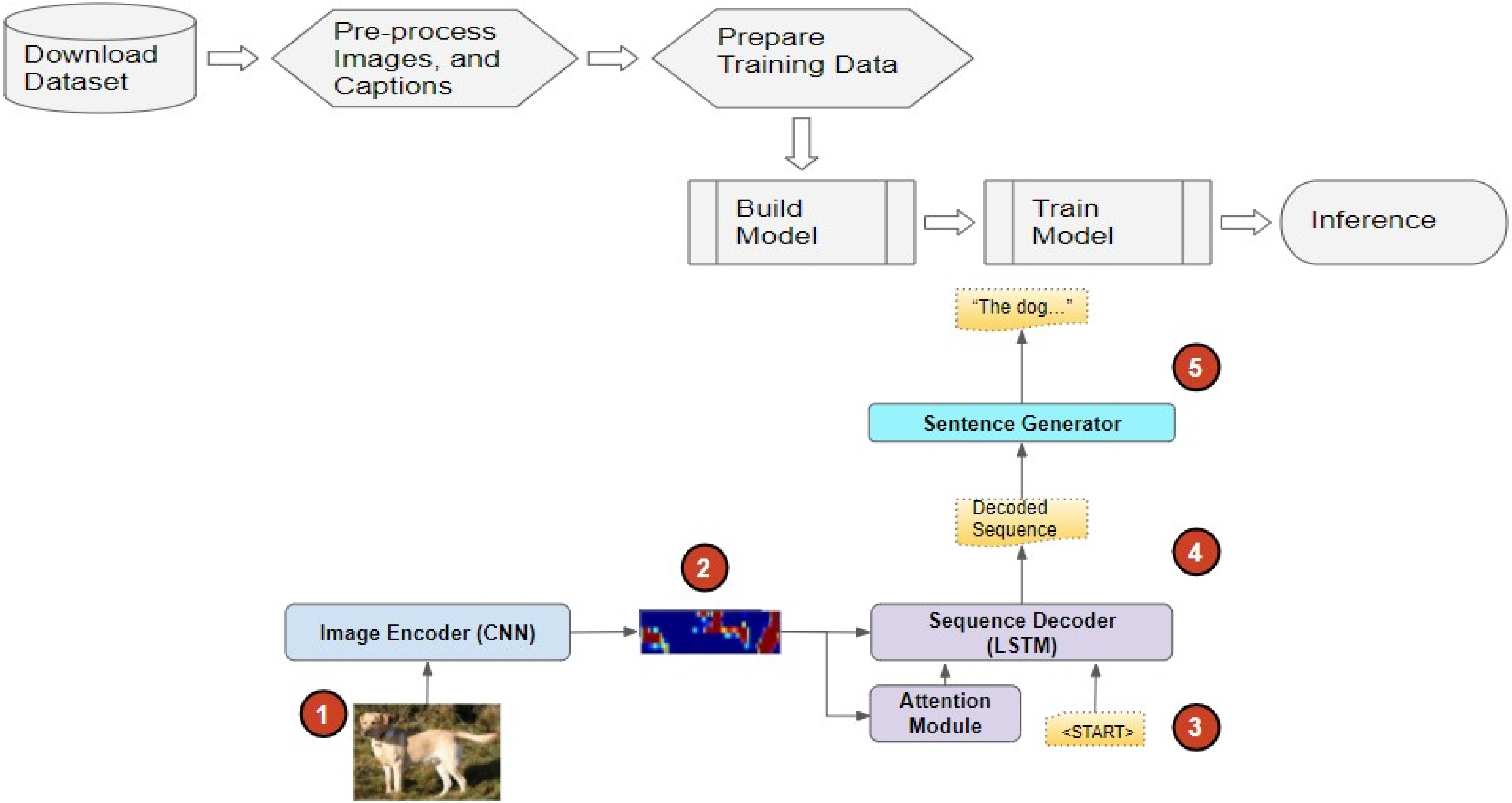
- Image captioning is the process of generating a textual description of an image using machine learning techniques. It is a combination of computer vision and natural language processing that involves analyzing an image to identify its contents and then generating a human-like textual description of what is happening in the image.

- In image captioning, a deep neural network is trained on a large dataset of images and corresponding captions. The network is typically an encoder-decoder architecture, where the encoder extracts features from the image and the decoder generates the caption. The features extracted by the encoder are then used by the decoder to generate a sequence of words that form the final caption.

Attention helped the model focus on the most relevant portion of the image as it generated each word of the caption.



[[1] Image captioning with visual attention | TensorFlow Core]

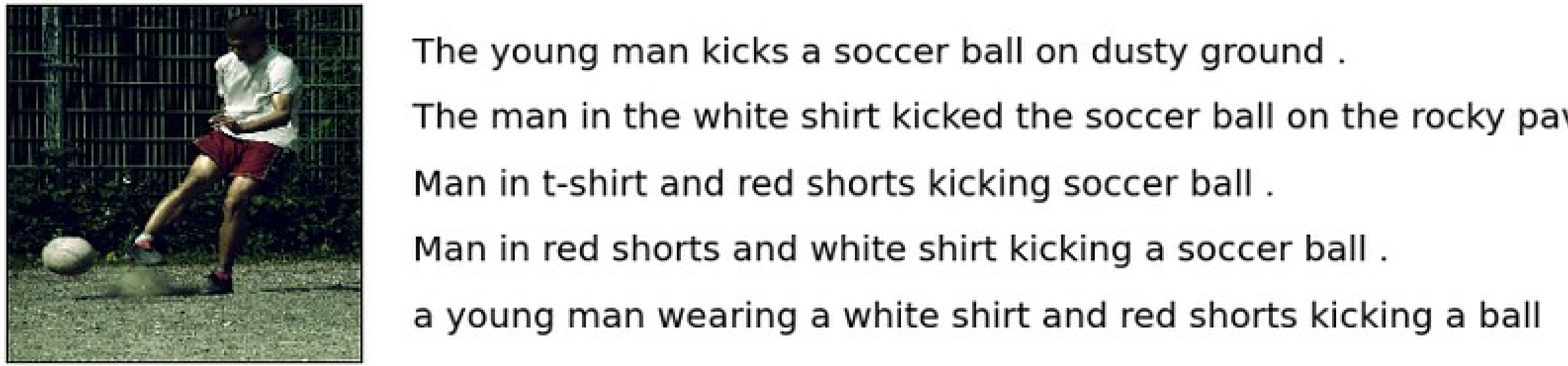


[2] Image Captions with Attention in Tensorflow, Step-by-step | by Ketan Doshi | Towards Data Science

Flicker8k_Dataset

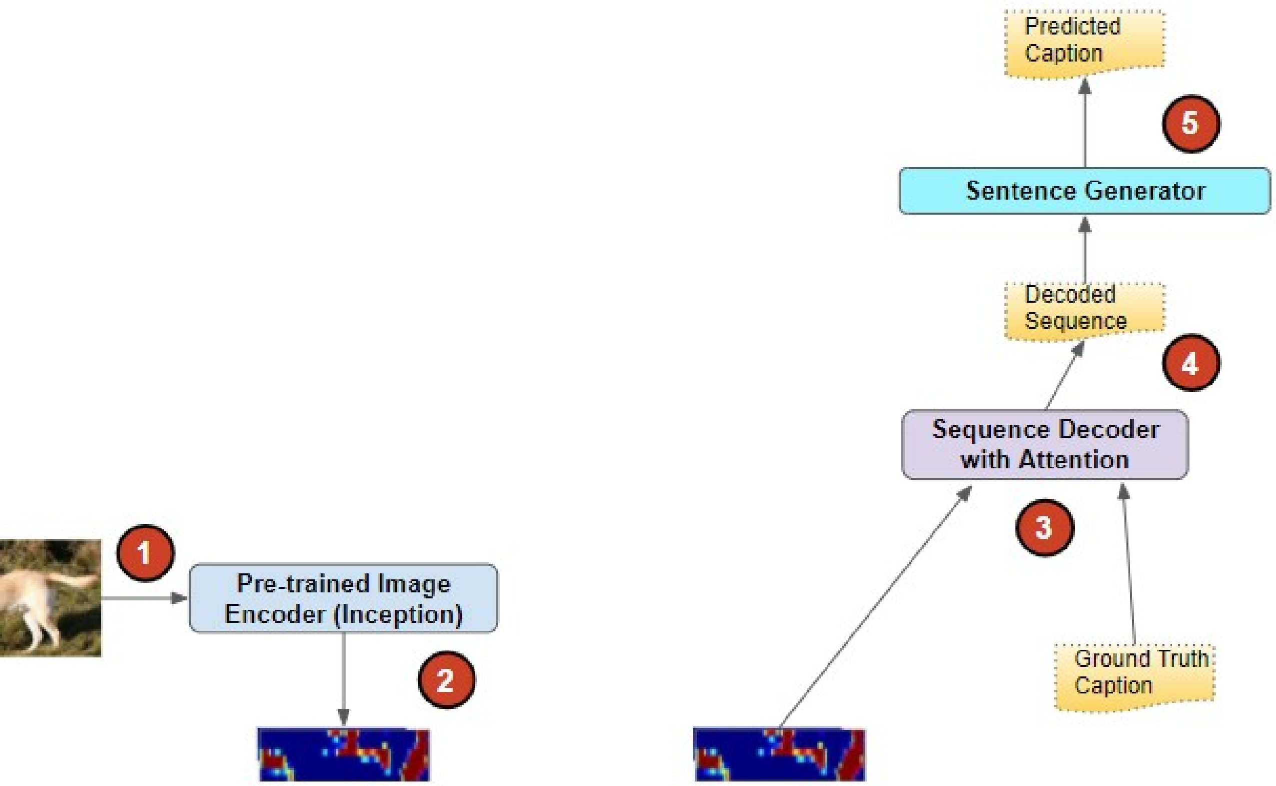
- **Image files** in the ‘*Flicker8k_Dataset*’ folder: This folder contains roughly 8000 .jpg files eg. ‘*1000268201_693b08cb0e.jpg*’
- **Captions** in the ‘*Flickr8k.token.txt*’ file in the main folder: It contains captions for all the images. Because the same image can be described in many different ways, there are 5 captions per image.
- **List of Training, Validation, and Test Images** in a set of .txt files in the main folder: ‘*Flickr_8k.trainImages.txt*’ contains the list of image file names to be used for training. Similarly, there are files for validatior

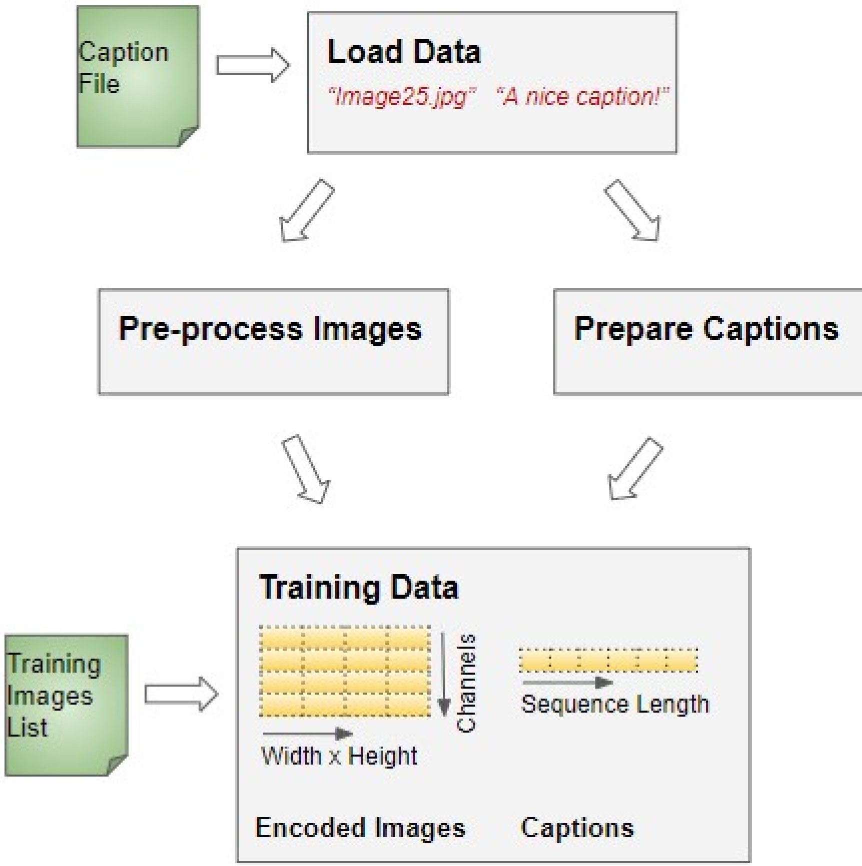
[Flickr 8k Dataset | Kaggle](#)



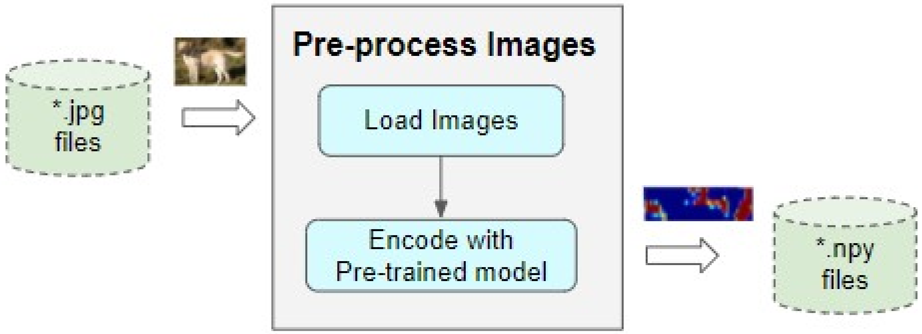
[3] Image Captions with Attention in Tensorflow, Step-by-step | by Ketan Doshi | Towards Data Science

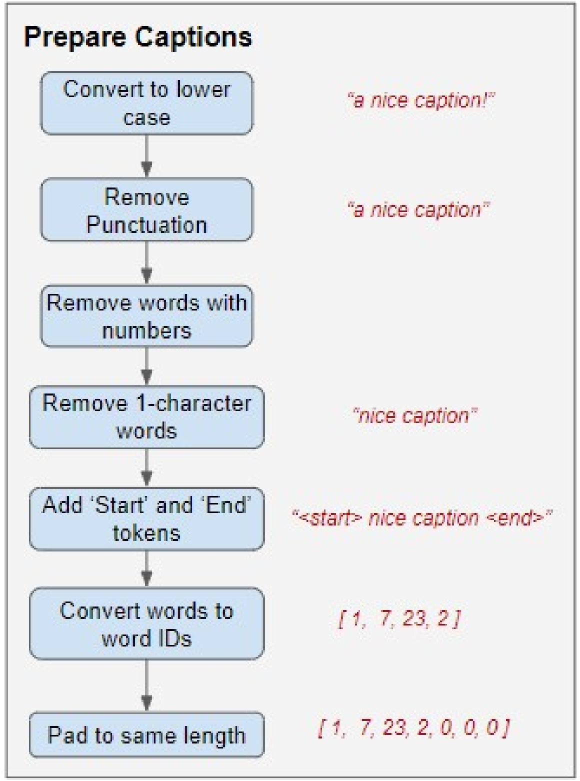
Training data pipeline





Prepare Captions





(1)flickr8k-image-captioning-using-cnns-lstms.ipynb

View Insert Runtime Tools Help All changes saved

+ Code + Text

_data
s.txt
.zip
json
png

```
#https://www.kaggle.com/datasets/adityajn105/flickr8k
```

[1]
```
import tensorflow as tf
tf.__version__
```

'2.12.0'

[2]
```
import keras
keras.__version__
```

'2.12.0'

[3]
```
!pip install -q kaggle
```

[4]
```
!ls -lha kaggle.json
```

```
-rw-r--r-- 1 root root 67 Apr 24 21:26 kaggle.json
```

[5]
```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
```

[6]
```
!chmod 600 /root/.kaggle/kaggle.json
```

[7]
```
!pwd
```

```
/content
```

[8]
```
!kaggle datasets download -d adityajn105/flickr8k
```

```
Downloading flickr8k.zip to /content
100% 1.03G/1.04G [00:29<00:00, 42.2MB/s]
100% 1.04G/1.04G [00:29<00:00, 37.8MB/s]
```

[9]
```
!unzip flickr8k.zip
```

```
 inflating: Images/856985136_649c0a3881.jpg
 inflating: Images/857914283_270d7d1c87.jpg
 inflating: Images/859620561_de417cac1e.jpg
 inflating: Images/860928274_744d14f198.jpg
 inflating: Images/861608773_bdafd5c996.jpg
 inflating: Images/861661418_8a37024ace.jpg
```

AML(1)flickr8k-image-captioning-using-cnns-lstms.ipynb

File Edit View Insert Runtime Tools Help Saving...

Files

Images
- 1000268201_693b08cb0e.j...
- 1001773457_577c3a7d70.j...
- 1002674143_1b742ab4b8.j...
- 1003163366_44323f5815.j...
- 1007129816_e794419615.j...
- 1007320043_627395c3d8.j...
- 1009434119_febe49276a.j...
- 1012212859_01547e3f17.j...
- 1015118661_980735411b.j...
- 1015584366_dfcec3c85a.j...
- 101654506_8eb26cfb60.jpg
- 101669240_b2d3e7f17b.jpg
- 1016887272_03199f49c4.j...
- 1019077836_6fc9b15408.j...
- 1019604187_d087bf9a5f.j...
- 1020651753_06077ec457.j...
- 1022454332_6af2c1449a.j...
- 1022454428_b6b660a67b.j...
- 1022975728_75515238d8.j...
- 102351840_323e3de834.jpg
- 1024138940_f1fefbdce1.jpg
- 102455176_5f8ead62d5.jpg
- 1026685415_0431cbf574.j...
- 1028205764_7e8df9a2ea.j...
- 1030985833_b0902ea560.j...
- 103106960_e8a41d64f8.jpg
- 103195344_5d2dc613a3.jpg
- 103205630_682ca7285b.jpg
- 1032122270_ea6f0beedb.j...
- 1032460886_4a598ed535.j...
- 1034276567_49bb87c51c.j...
- 104136873_5b5d41be75.jpg
- 1042020065_fb3d3ba5ba.j...
- 1042590306_95dea0916c.j...

+ Code + Text

```python
from tqdm.notebook import tqdm
tqdm.pandas()
import cv2, warnings
warnings.filterwarnings('ignore')
import tensorflow as tf
from tensorflow.keras.layers import Dense, Flatten, Input, Add, Dropout, LSTM, TimeDistributed, Embedding, RepeatVector, Concatenate, Bidirectional, Convolution2D
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.utils import to_categorical, plot_model
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import ModelCheckpoint
```

Loading the images

```python
img_path = '/content/Images/'
images = glob(img_path+'*.jpg')
images[:5]
```

```
['/content/Images/2259336826_0cb294e1f7.jpg',
 '/content/Images/3258395783_2de3a4ba27.jpg',
 '/content/Images/2429212017_77fc107699.jpg',
 '/content/Images/2592019072_a6c0090da4.jpg',
 '/content/Images/3251646144_d9f4ccca3f.jpg']
```

[12]
```python
len(images)
```

```
8091
```

Loading the captions

[13]
```python
captions = open('/content/captions.txt','rb').read().decode('utf-8').split('\n')
captions[:5]
```

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

[14]
```python
len(captions)
```

```
40457
```

izing images along with their captions

```
 in range(5):
t.figure(figsize=(5,5))
ng = cv2.imread(images[i])
ng = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
t.imshow(img);
```

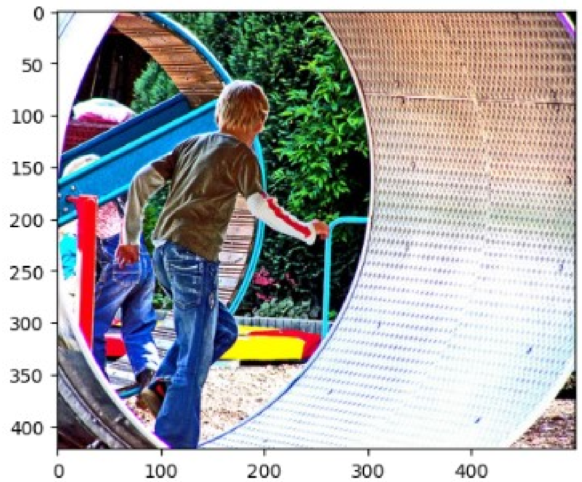


Image-Captioning-using-cnns-lstms

- Image captioning using CNNs (Convolutional Neural Networks) and LSTMs (Long Short-Term Memory networks) is a popular approach for generating captions for images.
- CNNs are used to extract relevant features from the input image, while LSTMs are used to generate a sequence of words to describe the image. The CNN is used as an encoder to encode the image into a fixed-length feature vector. This feature vector is then used as the initial hidden state of the LSTM decoder. The LSTM decoder generates a sequence of words by predicting the next word in the sequence given the previous words and the encoded image features.
- During training, the model is trained to minimize the difference between the predicted caption and the ground truth caption. This is done using a loss function such as cross-entropy loss.
- The main advantage of this approach is that it can generate accurate and descriptive captions for a wide range of images. However, it requires a large amount of training data and computing resources to train the model effectively.

```
nizer = Tokenizer()
nizer.fit_on_texts(captions)
ab_size = len(tokenizer.word_index) + 1
_length = max(len(caption.split()) for caption in captions)

ges = data['image'].unique().tolist()
ages = len(images)

_index = round(0.85*nimages)
_images = images[:split_index]
images = images[split_index:]

 = data[data['image'].isin(train_images)]
= data[data['image'].isin(val_images)]

.reset_index(inplace=True,drop=True)
reset_index(inplace=True,drop=True)

nizer.texts_to_sequences([captions[1]])[0]
```

```
8, 315, 63, 195, 116, 2]
```

```
lel = DenseNet201()
Model(inputs=model.input, outputs=model.layers[-2].output)

_size = 224
ures = {}
mage in tqdm(data['image'].unique().tolist()):
g = load_img(os.path.join(image_path,image),target_size=(img_size,img_size))
g = img_to_array(img)
g = img/255.
g = np.expand_dims(img,axis=0)
ature = fe.predict(img, verbose=0)
atures[image] = feature
```

```
%|██████████████| 8091/8091 [15:04<00:00,  8.95it/s]
```

```
class CustomDataGenerator(Sequence):

    def __init__(self, df, X_col, y_col, batch_size, directory, tokenizer,
                 vocab_size, max_length, features,shuffle=True):

        self.df = df.copy()
        self.X_col = X_col
        self.y_col = y_col
        self.directory = directory
        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.n = len(self.df)

    def on_epoch_end(self):
        if self.shuffle:
            self.df = self.df.sample(frac=1).reset_index(drop=True)

    def __len__(self):
        return self.n // self.batch_size

    def __getitem__(self,index):

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

    def __get_data(self,batch):

        X1, X2, y = list(), list(), list()

        images = batch[self.X_col].tolist()

        for image in images:
            feature = self.features[image][0]
```

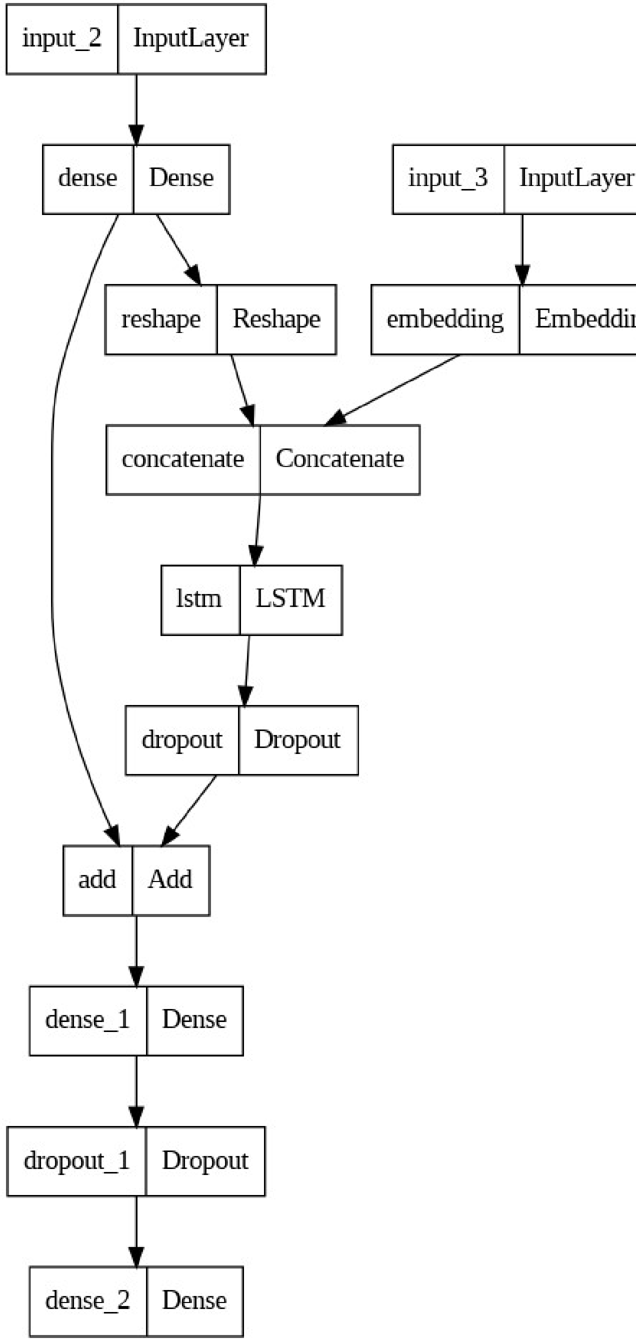
elling

he image embedding representations are concatenated with the first word of sentence ie. starseq and passed to the LSTM network

he LSTM network starts generating words after each input thus forming a sentence at the end

```
put1 = Input(shape=(1920,))
put2 = Input(shape=(max_length,))

g_features = Dense(256, activation='relu')(input1)

g_features_reshaped = Reshape((1, 256), input_shape=(256,))(img_features)

ntence_features = Embedding(vocab_size, 256, mask_zero=False)(input2)
erged = concatenate([img_features_reshaped,sentence_features],axis=1)
ntence_features = LSTM(256)(merged)
 Dropout(0.5)(sentence_features)
 add([x, img_features])
 Dense(128, activation='relu')(x)
 Dropout(0.5)(x)
tput = Dense(vocab_size, activation='softmax')(x)

ption_model = Model(inputs=[input1,input2], outputs=output)
ption_model.compile(loss='categorical_crossentropy',optimizer='adam')
```

```
om tensorflow.keras.utils import plot_model
```



```
n_generator = CustomDataGenerator(df=train,X_col='image',y_col='caption',batch_size=8,directory=image_path,
                    tokenizer=tokenizer,vocab_size=vocab_size,max_length=max_length,features=features)

dation_generator = CustomDataGenerator(df=test,X_col='image',y_col='caption',batch_size=8,directory=image_path,
                    tokenizer=tokenizer,vocab_size=vocab_size,max_length=max_length,features=features)
```

```
del_name = "model.h5"
ckpoint = ModelCheckpoint(model_name,
              monitor="val_loss",
              mode="min",
              save_best_only = True,
              verbose=1)

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

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

```
ory = caption_model.fit(
  train_generator,
  epochs=5,
  validation_data=validation_generator,
  callbacks=[checkpoint,earlystopping,learning_rate_reduction])
```

```
ch 1/5
8/4298 [==============================] - ETA: 0s - loss: 4.7734
ch 1: val_loss improved from inf to 4.10007, saving model to model.h5
8/4298 [==============================] - 197s 45ms/step - loss: 4.7734 - val_loss: 4.1001 - lr: 0.0010
ch 2/5
5/4298 [============================>.] - ETA: 0s - loss: 4.0492
ch 2: val_loss improved from 4.10007 to 3.86395, saving model to model.h5
8/4298 [==============================] - 82s 19ms/step - loss: 4.0490 - val_loss: 3.8640 - lr: 0.0010
```

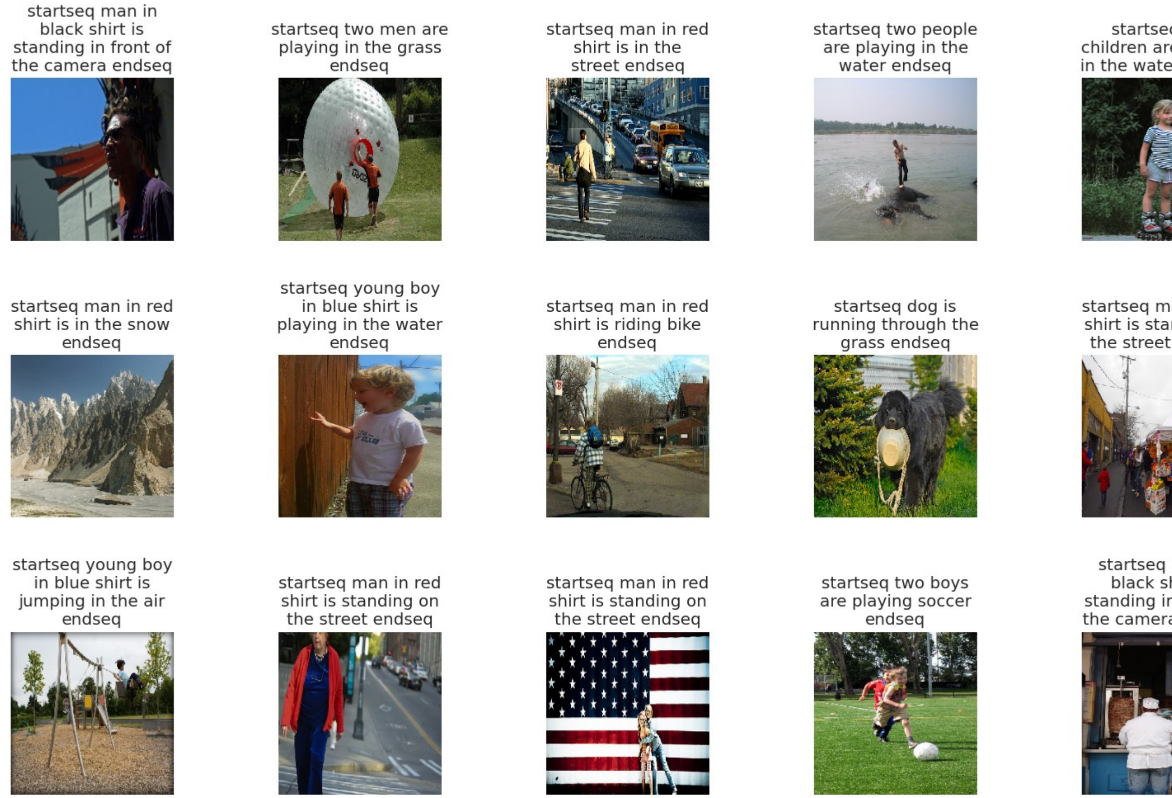


Image Caption Generation Using Resnet & LSTM

- Image caption generation using ResNet and LSTMs is another approach for generating captions for images.
- ResNet (Residual Network) is a type of CNN architecture that has been shown to be very effective in image recognition tasks. In this approach, ResNet is used to extract features from the input image, and the resulting feature vector is used as input to an LSTM network.
- The LSTM network is used to generate a sequence of words to describe the image. The LSTM takes the encoded image features as input and generates a sequence of words by predicting the next word in the sequence given the previous words and the encoded image features.
- Similar to the CNN-LSTM approach, during training, the model is trained to minimize the difference between the predicted caption and the ground truth caption. This is done using a loss function such as cross-entropy loss.
- This approach has been shown to be effective in generating high-quality captions for images. It is also relatively efficient compared to other approaches that use attention mechanisms, making it a good option for real-time applications.

oading the ResNet50 inception model

```
ption_model = ResNet50(include_top=True)
ption_model.summary()
```

```
nloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels.h5
67424/102967424 [==============================] - 3s 0us/step
el: "resnet50"
__________________________________________________________________________________________________
r (type)                Output Shape         Param #     Connected to
==================================================================================================
t_1 (InputLayer)        [(None, 224, 224, 3  0           []
                        )]

v1_pad (ZeroPadding2D)      (None, 230, 230, 3)  0        ['input_1[0][0]']

v1_conv (Conv2D)        (None, 112, 112, 64  9472        ['conv1_pad[0][0]']
                        )

v1_bn (BatchNormalization)  (None, 112, 112, 64  256     ['conv1_conv[0][0]']
                        )

v1_relu (Activation)    (None, 112, 112, 64  0           ['conv1_bn[0][0]']
                        )

l1_pad (ZeroPadding2D)      (None, 114, 114, 64  0       ['conv1_relu[0][0]']
                        )

l1_pool (MaxPooling2D)      (None, 56, 56, 64)   0       ['pool1_pad[0][0]']

v2_block1_1_conv (Conv2D)   (None, 56, 56, 64)   4160    ['pool1_pool[0][0]']

v2_block1_1_bn (BatchNormal  (None, 56, 56, 64)  256     ['conv2_block1_1_conv[0][0]']
on)

v2_block1_1_relu (Activatio  (None, 56, 56, 64)  0       ['conv2_block1_1_bn[0][0]']

v2_block1_2_conv (Conv2D)   (None, 56, 56, 64)   36928   ['conv2_block1_1_relu[0][0]']

v2_block1_2_bn (BatchNormal  (None, 56, 56, 64)  256     ['conv2_block1_2_conv[0][0]']
on)

v2_block1_2_relu (Activatio  (None, 56, 56, 64)  0       ['conv2_block1_2_bn[0][0]']

v2_block1_0_conv (Conv2D)   (None, 56, 56, 256)  16640   ['pool1_pool[0][0]']

v2_block1_3_conv (Conv2D)   (None, 56, 56, 256)  16640   ['conv2_block1_2_relu[0][0]']

v2_block1_0_bn (BatchNormal  (None, 56, 56, 256)  1024   ['conv2_block1_0_conv[0][0]']
on)

v2_block1_3_bn (BatchNormal  (None, 56, 56, 256)  1024   ['conv2_block1_3_conv[0][0]']
on)

v2_block1_add (Add)         (None, 56, 56, 256)  0       ['conv2_block1_0_bn[0][0]',
                                                          'conv2_block1_3_bn[0][0]']

v2_block1_out (Activation)  (None, 56, 56, 256)  0       ['conv2_block1_add[0][0]']

v2_block2_1_conv (Conv2D)   (None, 56, 56, 64)   16448   ['conv2_block1_out[0][0]']
```

```
= inception_model.layers[-2].output # Output of the penultimate layer of ResNet model
el = Model(inputs=inception_model.input,outputs=last)
el.summary()
```

```
4_block6_1_bn (BatchNormal  (None, 14, 14, 256)  1024    ['conv4_block6_1_conv[0][0]']
```

```
last = inception_model.layers[-2].output # Output of the penultimate layer of ResNet model
model = Model(inputs=inception_model.input,outputs=last)
model.summary()
```

```
conv5_block2_1_bn (BatchNormal  (None, 7, 7, 512)  2048    ['conv5_block2_1_conv[0][0]']
ization)

conv5_block2_1_relu (Activatio  (None, 7, 7, 512)  0       ['conv5_block2_1_bn[0][0]']
n)

conv5_block2_2_conv (Conv2D)   (None, 7, 7, 512)  2359808  ['conv5_block2_1_relu[0][0]']

conv5_block2_2_bn (BatchNormal  (None, 7, 7, 512)  2048    ['conv5_block2_2_conv[0][0]']
ization)

conv5_block2_2_relu (Activatio  (None, 7, 7, 512)  0       ['conv5_block2_2_bn[0][0]']
n)

conv5_block2_3_conv (Conv2D)   (None, 7, 7, 2048)  1050624  ['conv5_block2_2_relu[0][0]']

conv5_block2_3_bn (BatchNormal  (None, 7, 7, 2048)  8192   ['conv5_block2_3_conv[0][0]']
ization)

conv5_block2_add (Add)         (None, 7, 7, 2048)  0       ['conv5_block1_out[0][0]',
                                                            'conv5_block2_3_bn[0][0]']

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

conv5_block3_1_conv (Conv2D)   (None, 7, 7, 512)  1049088  ['conv5_block2_out[0][0]']

conv5_block3_1_bn (BatchNormal  (None, 7, 7, 512)  2048    ['conv5_block3_1_conv[0][0]']
ization)

conv5_block3_1_relu (Activatio  (None, 7, 7, 512)  0       ['conv5_block3_1_bn[0][0]']
n)

conv5_block3_2_conv (Conv2D)   (None, 7, 7, 512)  2359808  ['conv5_block3_1_relu[0][0]']

conv5_block3_2_bn (BatchNormal  (None, 7, 7, 512)  2048    ['conv5_block3_2_conv[0][0]']
ization)

conv5_block3_2_relu (Activatio  (None, 7, 7, 512)  0       ['conv5_block3_2_bn[0][0]']
n)

conv5_block3_3_conv (Conv2D)   (None, 7, 7, 2048)  1050624  ['conv5_block3_2_relu[0][0]']

conv5_block3_3_bn (BatchNormal  (None, 7, 7, 2048)  8192   ['conv5_block3_3_conv[0][0]']
ization)

conv5_block3_add (Add)         (None, 7, 7, 2048)  0       ['conv5_block2_out[0][0]',
                                                            'conv5_block3_3_bn[0][0]']

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

avg_pool (GlobalAveragePooling  (None, 2048)       0       ['conv5_block3_out[0][0]']
2D)

==================================================================================================
Total params: 23,587,712
Trainable params: 23,534,592
Non-trainable params: 53,120
```

tures from images

```
dm(images):
d(img_path)
lor(img,cv2.COLOR_BGR2RGB)
(img,(224,224)) # ResNet model requires images of dimensions (224,224,3)
pe(1,224,224,3) # Reshaping image to the dimensions of a single image
el.predict(img).reshape(2048,) # Feature extraction from images
g_path.split('/')[-1] # Extracting image name
ng_name] = features

eatures of only 1500 images as using more than 1500 images leads to overloading memory issues
):

= 0:
```

g the captions text

```
s[1:]
```

```
8b08cb0e.jpg,A child in a pink dress is climbing up a set of stairs in an entry way .',
b08cb0e.jpg,A girl going into a wooden building .',
b08cb0e.jpg,A little girl climbing into a wooden playhouse .',
b08cb0e.jpg,A little girl climbing the stairs to her playhouse .',
b08cb0e.jpg,A little girl in a pink dress going into a wooden cabin .']
```

```
')[1]
```

```
ent breeds looking at each other on the road .'
```

```
:

cap.split(',')[0]
.split(',')[1]
has 5 captions
n img_features:
e not in captions_dict:
_dict[img_name] = [caption] # Storing the first caption

_dict[img_name].append(caption) # Adding the remaining captions
```

Creating vocabulary of the entire text corpus

```
count_words = dict()
cnt = 1

for key, val in captions_dict.items(): # Iterating through all images with keys as images and their values as 5 captions
    for item in val: # Iterating through all captions for each image
        for word in item.split(): # Iterating through all words in each caption
            if word not in count_words:
                count_words[word] = cnt
                cnt += 1
```

```
len(count_words) # Vocab size
```

```
3919
```

```
# Encoding the text by assigning each word to its corresponding index in the vocabulary i.e. count_words dictionary
for key, val in captions_dict.items():
    for caption in val:
        encoded = []
        for word in caption.split():
            encoded.append(count_words[word])
        captions_dict[key][val.index(caption)] = encoded
```

Building a custom generator function to generate input image features, previously generated text and the text to be generated as output

```
def generator(img,caption):
    n_samples = 0
    X = []
    y_input = []
    y_output = []

    for key, val in caption.items():
        for item in val:
            for i in range(1,len(item)):
                X.append(img[key]) # Appending the input image features
                input_seq = [item[:i]] # Previously generated text to be used as input to predict the next word
                output_seq = item[i] # The next word to be predicted as output
                # Padding encoded text sequences to the maximum length
                input_seq = pad_sequences(input_seq,maxlen=max_len,padding='post',truncating='post')[0]
                # One Hot encoding the output sequence with vocabulary size as the total no. of classes
                output_seq = to_categorical([output_seq],num_classes=vocab_size+1)[0]
                y_input.append(input_seq)
                y_output.append(output_seq)

    return X, y_input, y_output
```

```
X, y_in, y_out = generator(img_features,captions_dict)
```

```
len(X), len(y_in), len(y_out)
```

```
(91827, 91827, 91827)
```

```
# Converting input and output into Numpy arrays for faster processing
X = np.array(X)
y_in = np.array(y_in,dtype='float64')
y_out = np.array(y_out,dtype='float64')
```

```
X.shape, y_in.shape, y_out.shape
```

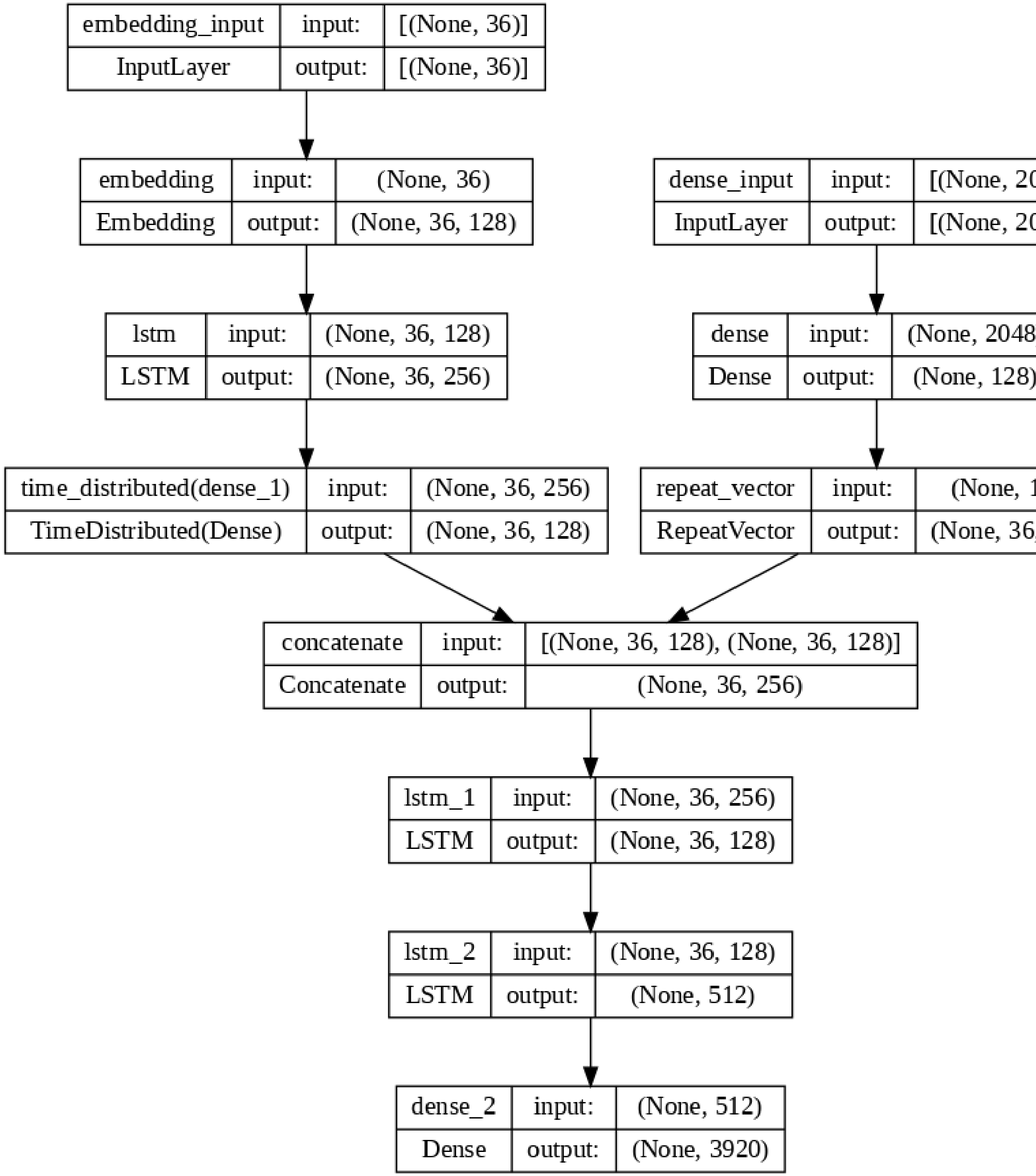
```
((91827, 2048), (91827, 36), (91827, 3920))
```

Model architecture

Establishing the model architecture

```
[ ] embedding_len = 128
    MAX_LEN = max_len
    vocab_size = len(count_words)

    # Model for image feature extraction
    img_model = Sequential()
    img_model.add(Dense(embedding_len,input_shape=(2048,),activation='relu'))
    img_model.add(RepeatVector(MAX_LEN))

    img_model.summary()

    # Model for generating captions from image features
    captions_model = Sequential()
    captions_model.add(Embedding(input_dim=vocab_size+1,output_dim=embedding_len,input_length=MAX_LEN))
    captions_model.add(LSTM(256,return_sequences=True))
    captions_model.add(TimeDistributed(Dense(embedding_len)))

    captions_model.summary()

    # Concatenating the outputs of image and caption models
    concat_output = Concatenate()([img_model.output,captions_model.output])
    # First LSTM Layer
    output = LSTM(units=128,return_sequences=True)(concat_output)
    # Second LSTM Layer
    output = LSTM(units=512,return_sequences=False)(output)
    # Output Layer
    output = Dense(units=vocab_size+1,activation='softmax')(output)
    # Creating the final model
    final_model = Model(inputs=[img_model.input,captions_model.input],outputs=output)
    final_model.compile(loss='categorical_crossentropy',optimizer='RMSprop',metrics='accuracy')
    final_model.summary()
```

```
 repeat_vector (RepeatVector  (None, 36, 128)       0
 )

=================================================================
Total params: 262,272
Trainable params: 262,272
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 36, 128)           501760

 lstm (LSTM)                 (None, 36, 256)           394240

 time_distributed (TimeDistr  (None, 36, 128)          32896
 ibuted)

=================================================================
Total params: 928,896
Trainable params: 928,896
Non-trainable params: 0
_________________________________________________________________
Model: "model_1"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 embedding_input (InputLayer)   [(None, 36)]         0           []

 dense_input (InputLayer)       [(None, 2048)]       0           []
```



Prediction

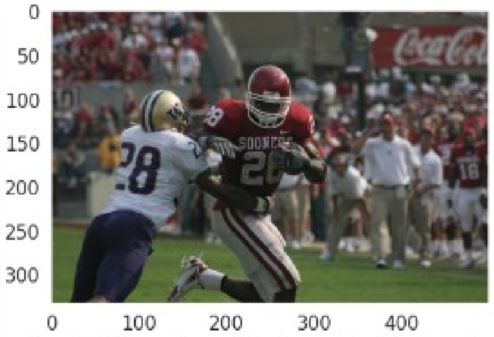
Generating sample predictions

```python
# Custom function for extracting an image and transforming it into an appropriate format
def getImage(idx):
    test_img_path = images[idx]
    test_img = cv2.imread(test_img_path)
    test_img = cv2.cvtColor(test_img,cv2.COLOR_BGR2RGB)
    test_img = cv2.resize(test_img,(224,224))
    test_img = np.reshape(test_img,(1,224,224,3))
    return test_img
```

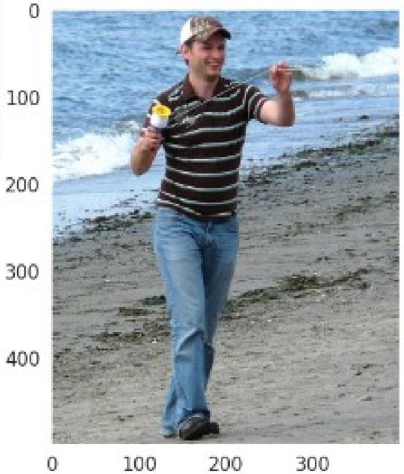
```python
for i in range(10):
    random_no = np.random.randint(0,1501,(1,1))[0,0]
    test_feature = model.predict(getImage(random_no)).reshape(1,2048)
    test_img_path = images[random_no]
    test_img = cv2.imread(test_img_path)
    test_img = cv2.cvtColor(test_img,cv2.COLOR_BGR2RGB)
    pred_text = ['startofseq']
    count = 0
    caption = '' # Stores the predicted captions text

    while count < 25:
      count += 1
      # Encoding the captions text with numbers
      encoded = []

      for i in pred_text:
        encoded.append(count_words[i])

      encoded = [encoded]
      # Padding the encoded text sequences to maximum length
      encoded = pad_sequences(encoded,maxlen=MAX_LEN,padding='post',truncating='post')
      pred_idx = np.argmax(final_model.predict([test_feature,encoded])) # Fetching the predicted word index having the maximum probability of occurence
      sampled_word = inverse_dict[pred_idx] # Extracting the predicted word by its respective index
      # Checking for ending of the sequence
      if sampled_word == 'endofseq':
        break
      caption = caption + ' ' + sampled_word
      #print(sampled_word)
      pred_text.append(sampled_word)


    plt.figure(figsize=(5,5))
    plt.imshow(test_img)
    plt.xlabel(caption)
```

a woman in a guitar in a guitar .

a man traveling through a snowy path .



a football player in a white jersey is tackle the play



ıan in a kite shirt and ieans is walkina in the

a man sits in a dirt road and pointing on the road

a black and white dog jumps up to catch a frisbee

Image Captioning Text Generation Attention

- Image captioning is a task in computer vision and natural language processing that involves generating a textual description of an image. It requires the system to analyze the visual content of the image and generate a coherent and accurate textual description of it.

- Text generation is a broader term that refers to the task of generating textual content, such as sentences, paragraphs, or entire documents, based on some input or context.

- Attention is a mechanism used in neural networks that allows the model to focus on specific parts of the input when generating output. In the context of image captioning, attention can be used to focus on different regions of the image when generating each word of the caption.

- Therefore, image captioning text generation attention refers to the process of generating a textual description of an image using neural networks, with the added use of attention mechanisms to improve the accuracy and coherence of the generated captions.

```
 spacy for the better text tokenization
_eng = spacy.load("en_core_web_sm")

ple
"This is a good place to find a city"
.text.lower() for token in spacy_eng.tokenizer(text)]
```

```
'is', 'a', 'good', 'place', 'to', 'find', 'a', 'city']
```

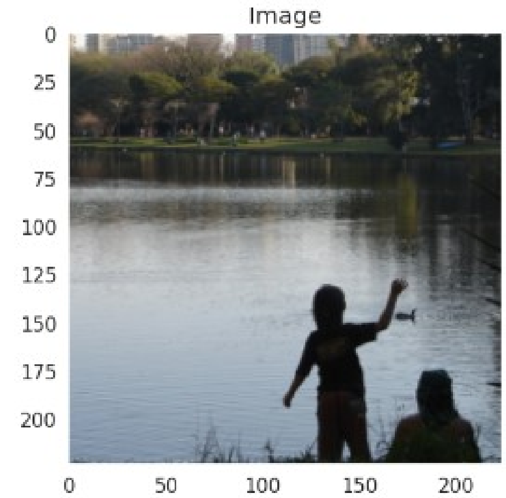
```
ocabulary:
__init__(self,freq_threshold):
etting the pre-reserved tokens int to string tokens
lf.itos = {0:"<PAD>",1:"<SOS>",2:"<EOS>",3:"<UNK>"}

tring to int tokens
ts reverse dict self.itos
lf.stoi = {v:k for k,v in self.itos.items()}

lf.freq_threshold = freq_threshold

__len__(self): return len(self.itos)

aticmethod
tokenize(text):
turn [token.text.lower() for token in spacy_eng.tokenizer(text)]

build_vocab(self, sentence_list):
equencies = Counter()
x = 4

r sentence in sentence_list:
 for word in self.tokenize(sentence):
     frequencies[word] += 1

     #add the word to the vocab if it reaches minum frequecy threshold
     if frequencies[word] == self.freq_threshold:
        self.stoi[word] = idx
        self.itos[idx] = word
        idx += 1

numericalize(self,text):
" For each word in the text corresponding index token for that word form the vocab built as list """
kenized_text = self.tokenize(text)
turn [ self.stoi[token] if token in self.stoi else self.stoi["<UNK>"] for token in tokenized_text ]
```

```
g the vicab class
cabulary(freq_threshold=1)

_vocab(["This is a good place to find a city"])
.stoi)
.numericalize("This is a good place to find a city here!!"))
```

```
>': 0, '<SOS>': 1, '<EOS>': 2, '<UNK>': 3, 'this': 4, 'is': 5, 'a': 6, 'good': 7, 'place': 8, 'to': 9, 'find': 10, 'city': 11}
, 7, 8, 9, 10, 6, 11, 3, 3, 3]
```

```
[79] class FlickrDataset(Dataset):
         """
         FlickrDataset
         """
         def __init__(self,root_dir,captions_file,transform=None,freq_threshold=5):
             self.root_dir = root_dir
             self.df = pd.read_csv("/content/captions.txt")
             self.transform = transform

             #Get image and caption colum from the dataframe
             self.imgs = self.df["image"]
             self.captions = self.df["caption"]

             #Initialize vocabulary and build vocab
             self.vocab = Vocabulary(freq_threshold)
             self.vocab.build_vocab(self.captions.tolist())


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

         def __getitem__(self,idx):
             caption = self.captions[idx]
             img_name = self.imgs[idx]
             img_location = os.path.join(self.root_dir,img_name)
             img = Image.open(img_location).convert("RGB")

             #apply the transfromation to the image
             if self.transform is not None:
                 img = self.transform(img)

             #numericalize the caption text
             caption_vec = []
             caption_vec += [self.vocab.stoi["<SOS>"]]
             caption_vec += self.vocab.numericalize(caption)
             caption_vec += [self.vocab.stoi["<EOS>"]]

             return img, torch.tensor(caption_vec)
```

```
#defining the transform to be applied
transforms = T.Compose([
    T.Resize(226),
    T.RandomCrop(224),
    T.ToTensor(),
    T.Normalize((0.485, 0.456, 0.406),(0.229, 0.224, 0.225))
])

#testing the dataset class
dataset = FlickrDataset(
    root_dir = "/content/Images",
    captions_file = "/content/captions.txt",
    transform=transforms
)



img, caps = dataset[80]
show_image(img,"Image")
print("Token:",caps)
print("Sentence:")
print([dataset.vocab.itos[token] for token in caps.tolist()])
```




```
Token: tensor([  1,   4,  28,  16,   4,  80,  34,  19, 766, 163,   8,   4, 250, 215,
          5,   2])
Sentence:
['<SOS>', 'a', 'child', 'and', 'a', 'woman', 'are', 'at', 'waters', 'edge', 'in', 'a', 'big', 'city', '.', '<EO
```

```
map representation from ResNet model will be used as input to our Attention model
derCNN(nn.Module):
it__(self):
(EncoderCNN, self).__init__()
t = models.resnet50(pretrained=True)
aram in resnet.parameters():
am.requires_grad_(False)

les = list(resnet.children())[:-2] # everythin except last Conv block
esnet = nn.Sequential(*modules)


ard(self, images):
res = self.resnet(images)                    #(batch_size,2048,7,7)
res = features.permute(0, 2, 3, 1)                    #(batch_size,7,7,2048)
res = features.view(features.size(0), -1, features.size(-1)) #(batch_size,49,2048)
n features
```

```
u Attention - allows our Decoder to pay attention to certain features (important parts)
tion(nn.Module):
it__(self, encoder_dim,decoder_dim,attention_dim):
(Attention, self).__init__()

ttention_dim = attention_dim

pass encoder/decoder hidden states to weights layer
/ = nn.Linear(decoder_dim,attention_dim)
 = nn.Linear(encoder_dim,attention_dim)

erate alignment scores
 = nn.Linear(attention_dim,1)

ut we accept feature map representation  and prev decoder hidden state
ard(self, features, hidden_state):
= self.U(features)     #(batch_size,num_layers,attention_dim)
 = self.W(hidden_state) #(batch_size,attention_dim)

ing one mode dimension for combined states
bined_states = torch.tanh(u_hs + w_ah.unsqueeze(1)) #(batch_size,num_layers,attemtion_dim)

tion_scores = self.A(combined_states)        #(batch_size,num_layers,1)
tion_scores = attention_scores.squeeze(2)    #(batch_size,num_layers)

version to probabilities using softmax
 = F.softmax(attention_scores,dim=1)        #(batch_size,num_layers)

erating the context vector
tion_weights = features * alpha.unsqueeze(2) #(batch_size,num_layers,features_dim)
tion_weights = attention_weights.sum(dim=1)  #(batch_size,num_layers)

n alpha,attention_weights
```

```
[89] # Attention Decoder module to generate captions
     class DecoderRNN(nn.Module):
         def __init__(self,embed_size, vocab_size, attention_dim,encoder_dim,decoder_dim,drop_prob=0.3):
             super().__init__()

             #save the model param
             self.vocab_size = vocab_size
             self.attention_dim = attention_dim
             self.decoder_dim = decoder_dim

             # generate embeddings for words
             self.embedding = nn.Embedding(vocab_size,embed_size)
             self.attention = Attention(encoder_dim,decoder_dim,attention_dim)

             # features representation
             self.init_h = nn.Linear(encoder_dim, decoder_dim)
             self.init_c = nn.Linear(encoder_dim, decoder_dim)
             self.lstm_cell = nn.LSTMCell(embed_size+encoder_dim,decoder_dim,bias=True)
             self.f_beta = nn.Linear(decoder_dim, encoder_dim)

             # linear layer that outputs one-hot vector of predicted word
             self.fcn = nn.Linear(decoder_dim,vocab_size)
             self.drop = nn.Dropout(drop_prob)

         # inputs are feature representation and captions (vectors)
         def forward(self, features, captions):

             #vectorize the caption
             embeds = self.embedding(captions)

             # Initialize LSTM state
             h, c = self.init_hidden_state(features)  # (batch_size, decoder_dim)

             #get the seq length to iterate
             seq_length = len(captions[0])-1 #Exclude the last one
             batch_size = captions.size(0)
             num_features = features.size(1)

             # predicted captions in form of one-hot vectors
             preds = torch.zeros(batch_size, seq_length, self.vocab_size).to(device)
             alphas = torch.zeros(batch_size, seq_length,num_features).to(device)

             # feed in the input for each time instance along with context vectors
             for s in range(seq_length):
                 # first, pass the features and decoder hidden state
                 alpha,context = self.attention(features, h)

                 # lstm input are embeddings repr words and context vectors
                 lstm_input = torch.cat((embeds[:, s], context), dim=1)

                 # hidden state for next time instance
                 h, c = self.lstm_cell(lstm_input, (h, c))

                 # pass through dropout layer
                 output = self.fcn(self.drop(h))

                 # get the prediction and weights
                 preds[:,s] = output
                 alphas[:,s] = alpha

             return preds, alphas

         def generate_caption(self,features,max_len=50,vocab=None):
```

```
[90] # Seq2Seq model to generate image captions

     class EncoderDecoder(nn.Module):
         def __init__(self,embed_size, vocab_size, attention_dim,encoder_dim,decoder_dim,drop_prob=0.3):
             super().__init__()
             # encoder doesn't need any params to specify
             self.encoder = EncoderCNN()
             # decoder params need to be specified
             self.decoder = DecoderRNN(
                 embed_size=embed_size,
                 vocab_size = len(dataset.vocab),
                 attention_dim=attention_dim,
                 encoder_dim=encoder_dim,
                 decoder_dim=decoder_dim
             )

         def forward(self, images, captions):
             # pass the images through encoder to get feature representations
             features = self.encoder(images)
             # features and captions are passed to decoder
             outputs = self.decoder(features, captions)
             return outputs
```

Setting Hypperparameter and model initialization

```
[91] # Hyperparams to tweak

     embed_size=500
     vocab_size = len(dataset.vocab)
     attention_dim=256
     encoder_dim=2048
     decoder_dim=512
     learning_rate = 3e-4
```

```
[92] # Seq2Seq model initialization

     model = EncoderDecoder(
         embed_size=500,
         vocab_size = len(dataset.vocab),
         attention_dim=256,
         encoder_dim=2048,
         decoder_dim=512
     ).to(device)

     criterion = nn.CrossEntropyLoss(ignore_index=dataset.vocab.stoi["<PAD>"])
     optimizer = optim.Adam(model.parameters(), lr=learning_rate)

     Downloading: "https://download.pytorch.org/models/resnet50-0676ba61.pth" to /root/.cache/torch/hub/checkpoints/resnet50-0676ba61.pth
     100%|██████████| 97.8M/97.8M [00:00<00:00, 105MB/s]
```

```
[93] # helper function to save the model
     def save_model(model,num_epochs):
         model_state = {
             'num_epochs':num_epochs,
             'embed_size':embed_size,
             'vocab_size':len(dataset.vocab),
             'attention_dim':attention_dim,
             'encoder_dim':encoder_dim,
             'decoder_dim':decoder_dim,
             'state_dict':model.state_dict()
         }

         torch.save(model_state,'attention_model_state.pth')
```

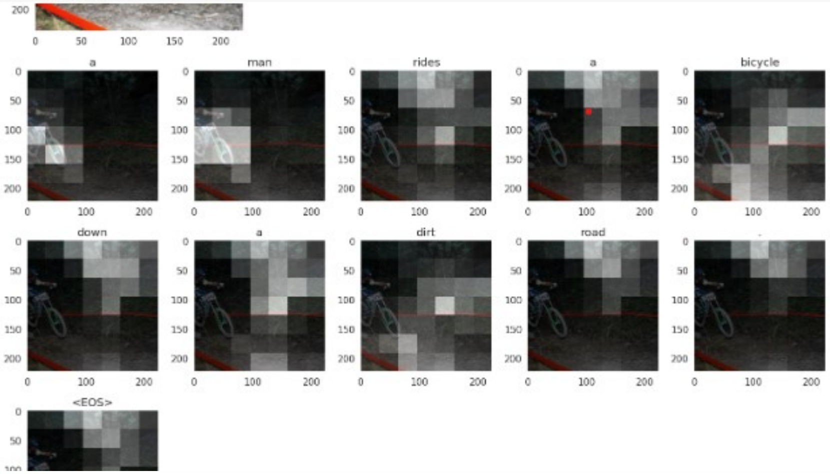
g Job from above configs

th every epoch the generated text becomes more and more meaningful !

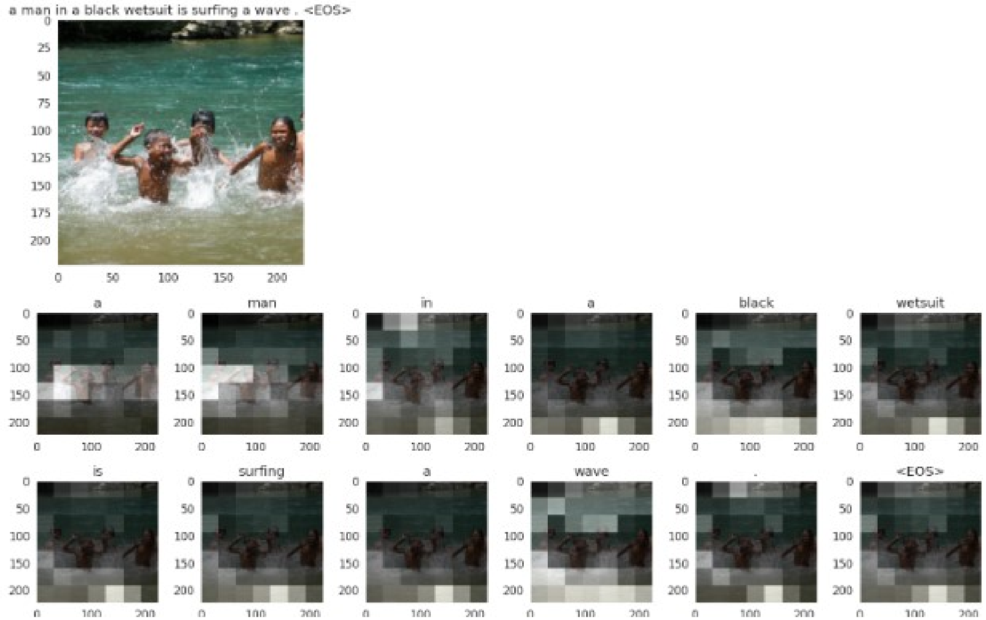
```
ochs = 3 #25
ery = 100 #1000

h in range(1,num_epochs+1):
, (image, captions) in enumerate(iter(data_loader)):
ge,captions = image.to(device),captions.to(device)

ro the gradients.
imizer.zero_grad()

ed forward
puts,attentions = model(image, captions)

lculate the batch loss.
ets = captions[:,1:]
 = criterion(outputs.view(-1, vocab_size), targets.reshape(-1))

ckward pass.
.backward()

pdate the parameters in the optimizer.
imizer.step()

dx+1)%print_every == 0:
rint("Epoch: {} Batch: {} loss: {:.5f}".format(epoch, idx+1, loss.item()))

rate the caption and display it
l.eval()
orch.no_grad():
iter = iter(data_loader)
,_ = next(dataiter)
ures = model.encoder(img[0:1].to(device))
s,alphas = model.decoder.generate_caption(features,vocab=dataset.vocab)
tion = ' '.join(caps)
w_image(img[0],title=caption)

l.train()

the latest model after every epoch
model(model,epoch)
```

```
 Batch: 100 loss: 4.77175
 Batch: 200 loss: 4.20422
 Batch: 300 loss: 4.45675
 Batch: 400 loss: 4.92297
 Batch: 500 loss: 4.67482
 Batch: 600 loss: 4.43538
 Batch: 700 loss: 4.02689
 Batch: 800 loss: 3.97568
```

```
dataiter = iter(data_loader)
images,_ = next(dataiter)

img = images[0].detach().clone()
img1 = images[0].detach().clone()
caps,alphas = get_caps_from(img.unsqueeze(0))

plot_attention(img1, caps, alphas)
```




```
#show any 1
dataiter = iter(data_loader)
images,_ = next(dataiter)

img = images[0].detach().clone()
img1 = images[0].detach().clone()
caps,alphas = get_caps_from(img.unsqueeze(0))

plot_attention(img1, caps, alphas)
```



Thank you so much