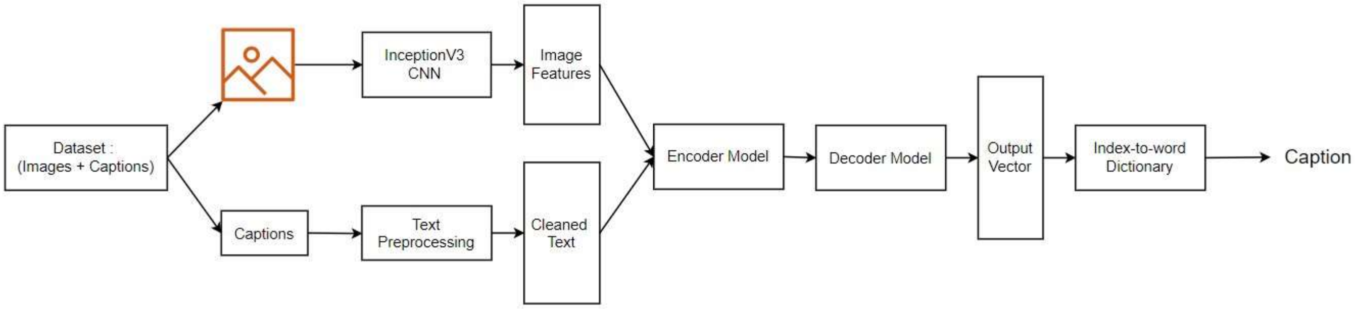
IMAGE CAPTIONER

Name: Appasamudram Naga Mohith

Register No: 122003032

Logic:



Dataset used:

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

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

Environment: Google Colab

Python code:

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

!unzip "/content/drive/MyDrive/Image Captioning/Flickr8k_Dataset.zip"
!unzip "/content/drive/MyDrive/Image Captioning/Flickr8k_text.zip"

# Importing libraries

import numpy as np
import pandas as pd
from time import time
import matplotlib.pyplot as plt
import tensorflow as tf
```

```
from tensorflow.keras.layers import (Input,Flatten,TimeDistributed,RepeatVector,Reshape,
                                     Dense,LSTM,Activation,Dropout,BatchNormalization,
                                     concatenate,Embedding,add)
from tensorflow.keras.models import Model,Sequential
from tensorflow.keras.preprocessing.image import load_img,img_to_array
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.optimizers import Adam,RMSprop
from tensorflow.keras.applications import MobileNet,InceptionV3,VGG16
from tensorflow.keras.applications.mobilenet import preprocess_input as preMobile
from tensorflow.keras.applications.inception_v3 import preprocess_input as preInception
from tensorflow.keras.applications.vgg16 import preprocess_input as preVgg
from tensorflow.keras.utils import to_categorical

import os
import string
import glob
import pickle
from PIL import Image
from tqdm import tqdm

START = "startseq"
STOP = "stopseq"
EPOCHS = 20

def hms_string(sec_elapsed):
    h = int(sec_elapsed / (60 * 60))
    m = int((sec_elapsed % (60 * 60)) / 60)
    s = sec_elapsed % 60
    return f"{h}:{m:>02}:{s:>05.2f}"

# Data Preprocessing
```

```
null_punct = str.maketrans(",",string.punctuation)
look_up = dict()
file = open("/content/Flickr8k.token.txt",'r')
maxlength = 0
for line in file.read().split('\n'):
  if len(line)<2:
    continue
  words = line.split()
  photoid,caption=words[0],words[1:]
  photoid = photoid.split('#')[0]

  # Cleaning the captions
  caption = [word.lower() for word in caption]
  caption = [word.translate(null_punct) for word in caption]
  caption = [word for word in caption if len(word)>1]
  caption = [word for word in caption if word.isalpha()]
  maxlength = max(len(caption),maxlength)
  caption = ' '.join(caption)
  if photoid not in look_up:
    look_up[photoid] = list()
  look_up[photoid].append(caption)

print(f'LookUp Table i.e photo-to-caption mapping size : {len(look_up)}')
vocabulary = set()     # a new vocabulary
for key in look_up:
  [vocabulary.update(d.split()) for d in look_up[key]]
print(f'Vocabulary size : {len(vocabulary)}')
print(f'Max Caption Length : {maxlength}')

data = pd.DataFrame(look_up.items())
data.iloc[0,1]
```

```
img = glob.glob(os.path.join('/content/Flicker8k_Dataset','*.jpg'))
len(img)

# Preparing Training Dataset

train_images = []
file = open('/content/Flickr_8k.trainImages.txt')
for line in file.read().strip().split('\n'):
  if len(line)<2:
    continue
  train_images.append(line)
file.close()

test_images = []
file = open('/content/Flickr_8k.testImages.txt')
for line in file.read().strip().split('\n'):
  if len(line)<2:
    continue
  test_images.append(line)
file.close()

dev_images = []
file = open('/content/Flickr_8k.devImages.txt')
for line in file.read().strip().split('\n'):
  if len(line)<2:
    continue
  dev_images.append(line)
file.close()

print(f'#Train Images = {len(train_images)}')
print(f'#Test Images = {len(test_images)}')
print(f'#Dev Images = {len(dev_images)}')
```

```
train_dataset = dict()
for photoid in train_images:
  if len(photoid)<2:
    continue
  train_dataset[photoid] = list()
  captions = look_up[photoid]
  for caption in captions:
    caption = START + " " + caption +" "+STOP
    train_dataset[photoid].append(caption)
print(f'Train Dataset Ready !! {len(train_dataset)} images')

len(train_dataset)

for k,v in train_dataset.items():
  print(k)
  for val in v:
    print(val)
  break

traindata = pd.DataFrame(train_dataset.items())
traindata.head()
```

# Encoding train images

```
#encode_model = MobileNet(weights='imagenet',include_top=False)
#WIDTH,HEIGHT,OUTPUT_DIM = 224,224,50176
encode_model = InceptionV3(weights='imagenet')
encode_model = Model(encode_model.input,encode_model.layers[-2].output)
WIDTH = 299
HEIGHT = 299
OUTPUT_DIM = 2048
```

```
encode_model.summary()

def encodeimage(img):
  img = img.resize((WIDTH,HEIGHT),Image.ANTIALIAS)
  X = img_to_array(img)
  X = np.expand_dims(img,axis=0)
  #X = preMobile(X)
  X = preInception(X)
  X = encode_model.predict(X)
  X = np.reshape(X,OUTPUT_DIM)
  return X

start = time()
train_encodings = dict()
for photoid in train_images:
  if len(photoid)<2:
    continue
  path = '/content/Flicker8k_Dataset/' + photoid
  train_image = load_img(path,target_size=(HEIGHT,WIDTH))
  train_encodings[photoid] = encodeimage(train_image)

with open('Train_Images_encoding_inception.pkl','wb') as fp:
  pickle.dump(train_encodings,fp)
print(f'\nGenerating training set took: {hms_string(time()-start)}')

with open('/content/drive/MyDrive/Image Captioning/Train_Images_encoding_inception.pkl','wb') as fp:
  pickle.dump(train_encodings,fp)

len(train_encodings)
```

```
train_enc = pd.DataFrame(train_encodings.items())
train_enc.iloc[0,1].shape

start = time()
test_encodings = dict()
for photoid in test_images:
  if len(photoid)<2:
    continue
  path = '/content/Flicker8k_Dataset/' + photoid
  test_image = load_img(path,target_size=(HEIGHT,WIDTH))
  test_encodings[photoid] = encodeimage(test_image)

with open('Test_Images_encoding_inception.pkl','wb') as fp:
  pickle.dump(test_encodings,fp)
print(f'\nGenerating test set took: {hms_string(time()-start)}')

with open('/content/drive/MyDrive/Image Captioning/Test_Images_encoding_inception.pkl','wb') as fp:
  pickle.dump(test_encodings,fp)

all_train_captions = list()
for photoid in train_dataset:
  for cap in train_dataset[photoid]:
    all_train_captions.append(cap)
len(all_train_captions)

threshold = 10
word_count = dict()
for cap in all_train_captions:
  for w in cap.split(' '):
    word_count[w] = word_count.get(w,0) + 1
vocab = [w for w,count in word_count.items() if count>threshold]
```

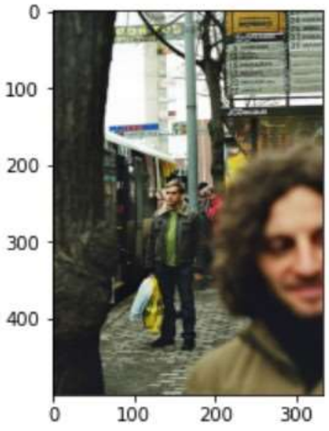
```
print('preprocessed words %d ==> %d' % (len(word_count), len(vocab)))

idxtoword = {}
wordtoidx = {}
ix = 1
for w in vocab:
  wordtoidx[w] = ix
  idxtoword[ix] = w
  ix += 1
vocabsize = len(idxtoword) + 1
vocabsize

maxlength = 34  #(32+2)

"""# Data Generator"""

def data_generator(descriptions, photos, wordtoidx,max_length, num_photos_per_batch):
  # x1 - Training data for photos
  # x2 - The caption that goes with each photo
  # y - The predicted rest of the caption
  x1, x2, y = [], [], []
  n=0
  while True:
    for key, desc_list in descriptions.items():
      n+=1
      photo = photos[key]
      # Each photo has 5 descriptions
      for desc in desc_list:
        # Convert each word into a list of sequences.
        seq = [wordtoidx[word] for word in desc.split(' ') \
               if word in wordtoidx]
        # Generate a training case for every possible sequence and outcome
        for i in range(1, len(seq)):
          in_seq, out_seq = seq[:i], seq[i]
          in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
          out_seq = to_categorical([out_seq], num_classes=vocabsize)[0]
          x1.append(photo)
          x2.append(in_seq)
          y.append(out_seq)
      if n==num_photos_per_batch:
        yield ([np.array(x1), np.array(x2)], np.array(y))
        x1, x2, y = [], [], []
        n=0

"""# Loading Glove Vectors"""

f = open('/content/glove.6B.200d.txt',encoding="utf-8")
embeddings_index = {}
for line in tqdm(f):
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs

f.close()
print(f'Found {len(embeddings_index)} word vectors.')

"""# Building the Neural Network

"""

embedding_dim = 200
# Get 200-dim dense vector for each of the 10000 words in out vocabulary
embedding_matrix = np.zeros((vocabsize,embedding_dim))
```

```
for word,i in wordtoidx.items():
  embedding_vector = embeddings_index.get(word)
  if embedding_vector is not None:
    embedding_matrix[i] = embedding_vector

embedding_matrix.shape

inputs1 = Input(shape=(OUTPUT_DIM,))
fe1 = Dropout(0.5)(inputs1)
fe2 = Dense(256,activation='relu')(fe1)
inputs2 = Input(shape=(maxlength,))
se1 = Embedding(vocabsize,embedding_dim,mask_zero=True)(inputs2)
se2 = Dropout(0.5)(se1)
se3 = LSTM(256)(se2)
decoder1 = add([fe2,se3])
decoder2 = Dense(256,activation='relu')(decoder1)
outputs = Dense(vocabsize,activation='softmax')(decoder2)

caption_model = Model(inputs=[inputs1,inputs2],outputs=outputs)

caption_model.summary()

caption_model.layers[2]

caption_model.layers[2].set_weights([embedding_matrix])
caption_model.layers[2].trainable=False
caption_model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=[('accuracy')])

caption_model.summary()

num_pics_per_batch = 3
```

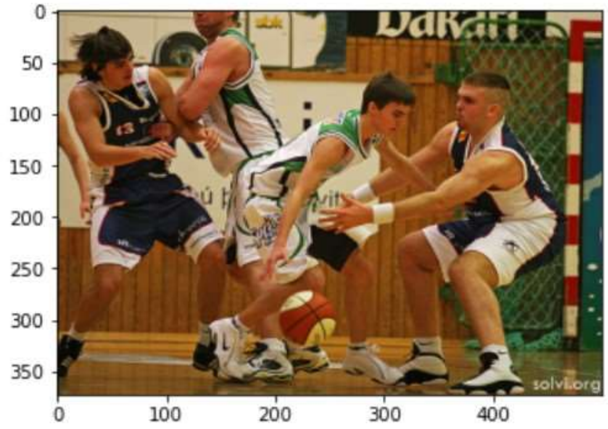
```
steps = len(train_dataset)//num_pics_per_batch

start = time()

for i in tqdm(range(EPOCHS*2)):
  generator = data_generator(train_dataset,train_encodings,wordtoidx,maxlength,num_pics_per_batch)
  caption_model.fit_generator(generator,epochs=1,steps_per_epoch=steps,verbose=1)
caption_model.optimizer.lr = 1e-4
num_pics_per_batch = 6
steps = len(train_dataset)//num_pics_per_batch

for i in range(EPOCHS):
  generator = data_generator(train_dataset,train_encodings,wordtoidx,maxlength,num_pics_per_batch)
  caption_model.fit_generator(generator,epochs=1,steps_per_epoch=steps,verbose=1)
caption_model.save('cpmodel.h5')
caption_model.save_weights('cpmodelweights.h5')
print(f"\Training took: {hms_string(time()-start)}")

def generateCaption(photo):
    in_text = START
    for i in range(maxlength):
        sequence = [wordtoidx[w] for w in in_text.split() if w in wordtoidx]
        sequence = pad_sequences([sequence], maxlen=maxlength)
        yhat = caption_model.predict([photo,sequence], verbose=0)
        yhat = np.argmax(yhat)
        word = idxtoword[yhat]
        in_text += ' ' + word
        if word == STOP:
            break
    final = in_text.split()
    final = final[1:-1]
    final = ' '.join(final)
    return final


caption_model.save('/content/drive/MyDrive/Image Captioning/cpmodel.h5')


caption_model.save_weights('/content/drive/MyDrive/Image Captioning/cpmodelweights.h5')


for i in range(20):
  pic = list(test_encodings.keys())[100+i]
  image = test_encodings[pic].reshape((1,OUTPUT_DIM))
  path = '/content/Flicker8k_Dataset/' + pic
  print(path)
  X = plt.imread(path)
  plt.imshow(X)
  plt.show()
  print("Caption: ",generateCaption(image))
  print("________________________________")
```

OUTPUT

```
_____________________________
/content/Flicker8k_Dataset/3344233740_c010378da7.jpg
```

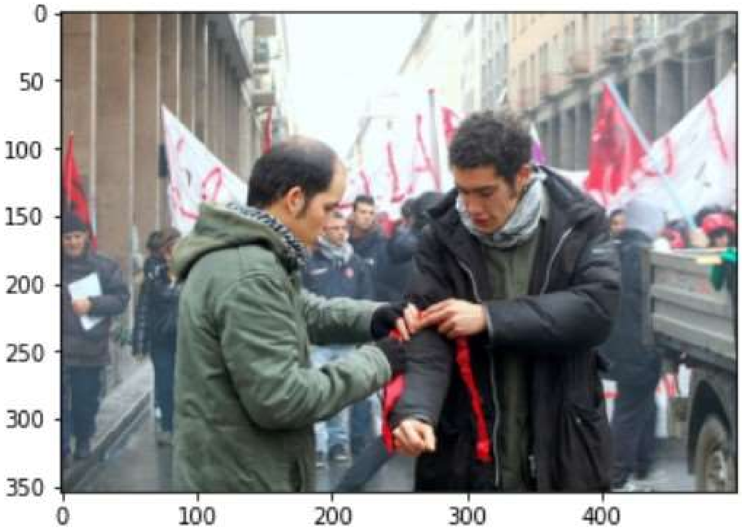


```
Caption:  man in black shirt and tie is standing in front of crowd of people
_____________________________
/content/Flicker8k_Dataset/3072172967_630e9c69d0.jpg
```



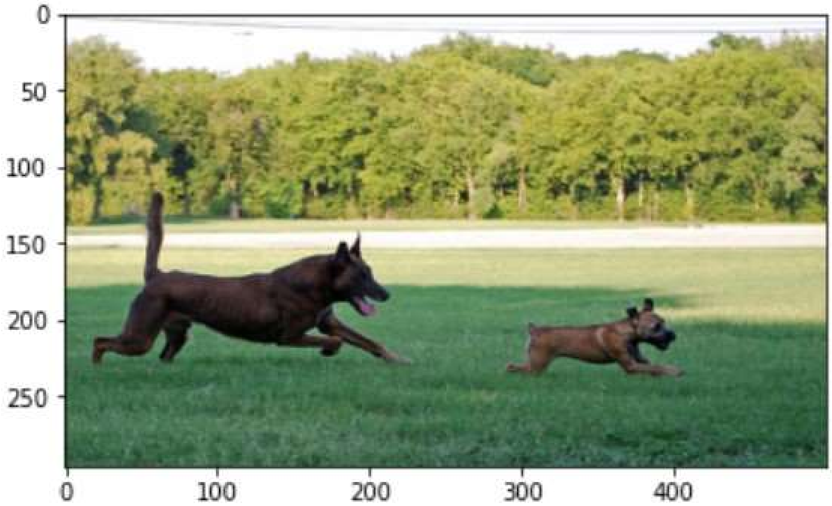
```
Caption:  basketball player in white uniform is trying to block player in white
```

```
_____________________________
/content/Flicker8k_Dataset/3110649716_c17e14670e.jpg
```



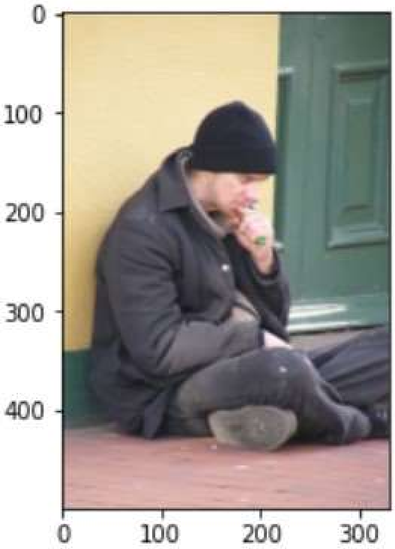
```
Caption:  man in black coat and cap talks to woman in black coat
```

```
/content/Flicker8k_Dataset/2723477522_d89f5ac62b.jpg
```

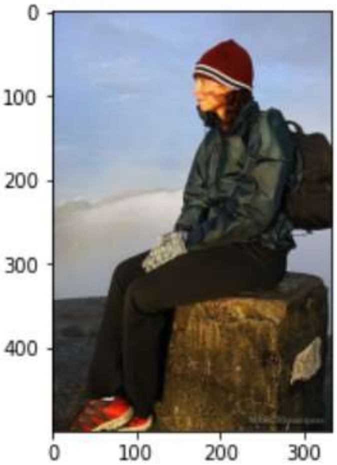


```
Caption:  two dogs are running in the grass
```

```
/content/Flicker8k_Dataset/2218609886_892dcd6915.jpg
```

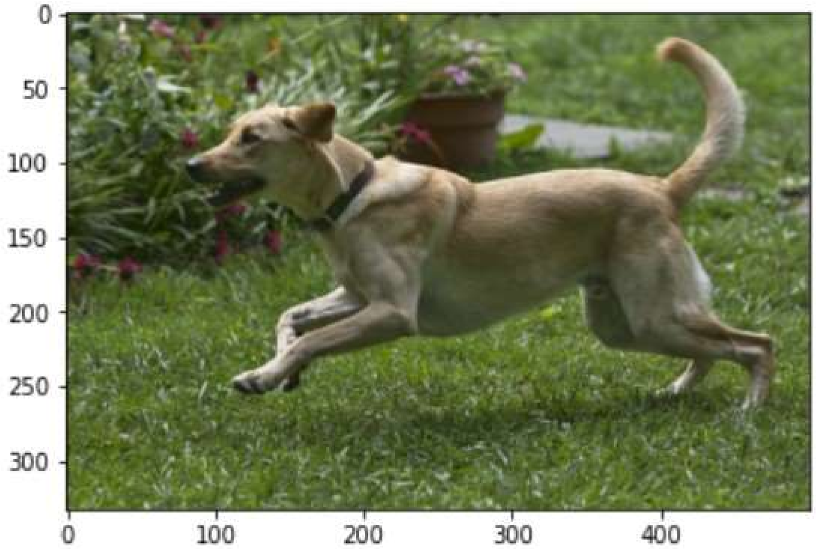


```
Caption:  man in black shirt and cast smokes cigarette
_________________________
/content/Flicker8k_Dataset/2435685480_a79d42e564.jpg
```

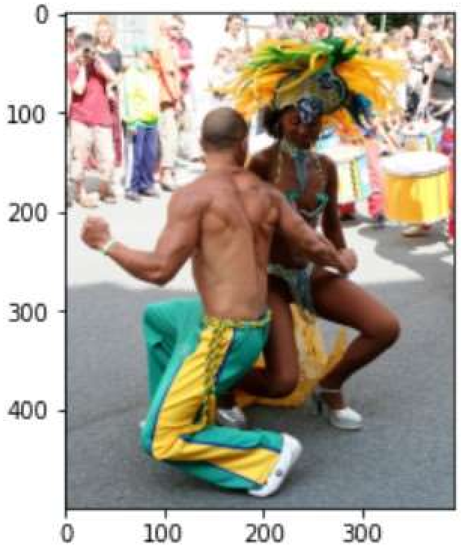


```
Caption:  man in red shirt and jeans is standing on the side of rock face
____________________________
```

```
_____________________________
/content/Flicker8k_Dataset/2654514044_a70a6e2c21.jpg
```

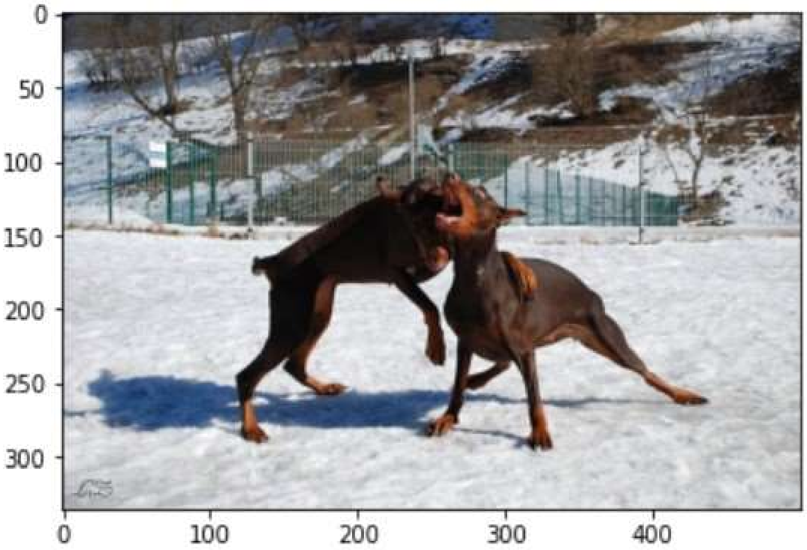


```
Caption:  brown dog is running through the grass
_____________________________
/content/Flicker8k_Dataset/311146855_0b65fdb169.jpg
```



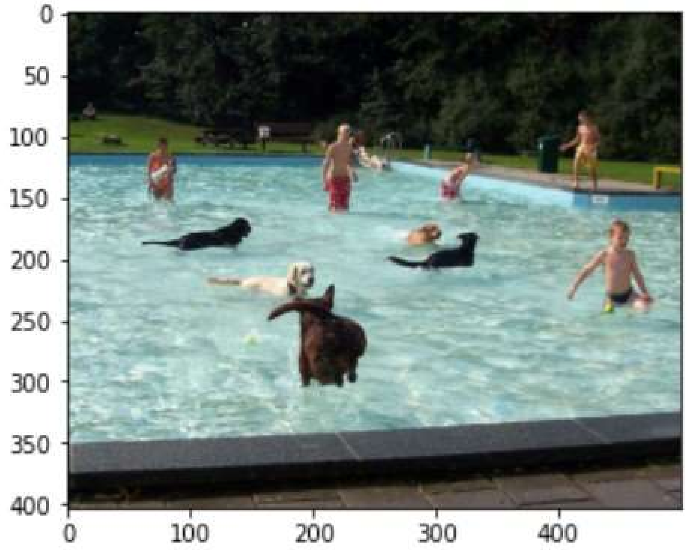
```
Caption:  man in red shirt and jeans is dancing
```

```
/content/Flicker8k_Dataset/3385593926_d3e9c21170.jpg
```

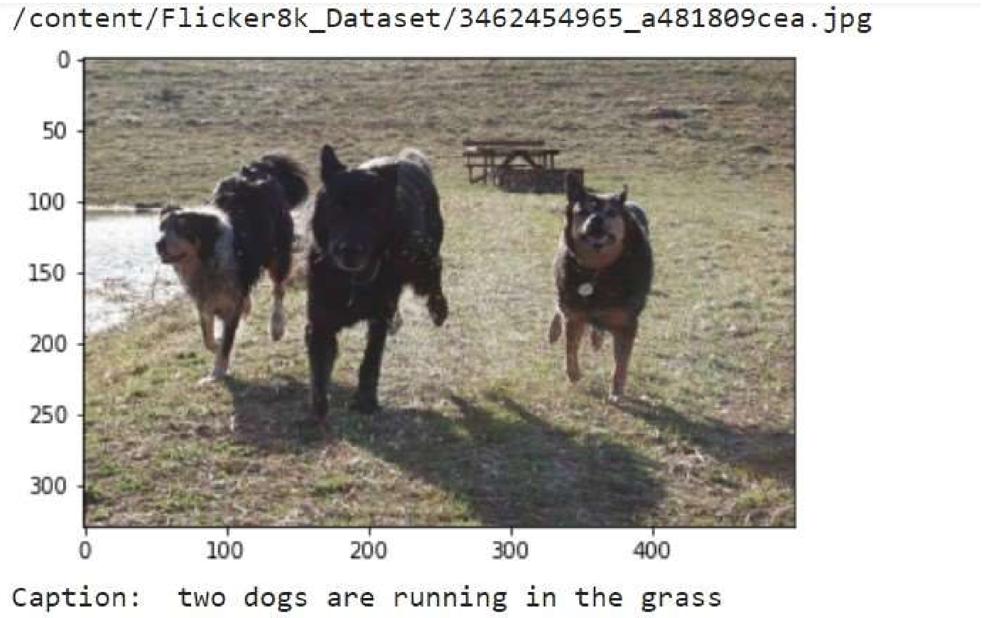


```
Caption:  brown dog is running through the snow
________________________________
```

```
/content/Flicker8k_Dataset/244571201_0339d8e8d1.jpg
```

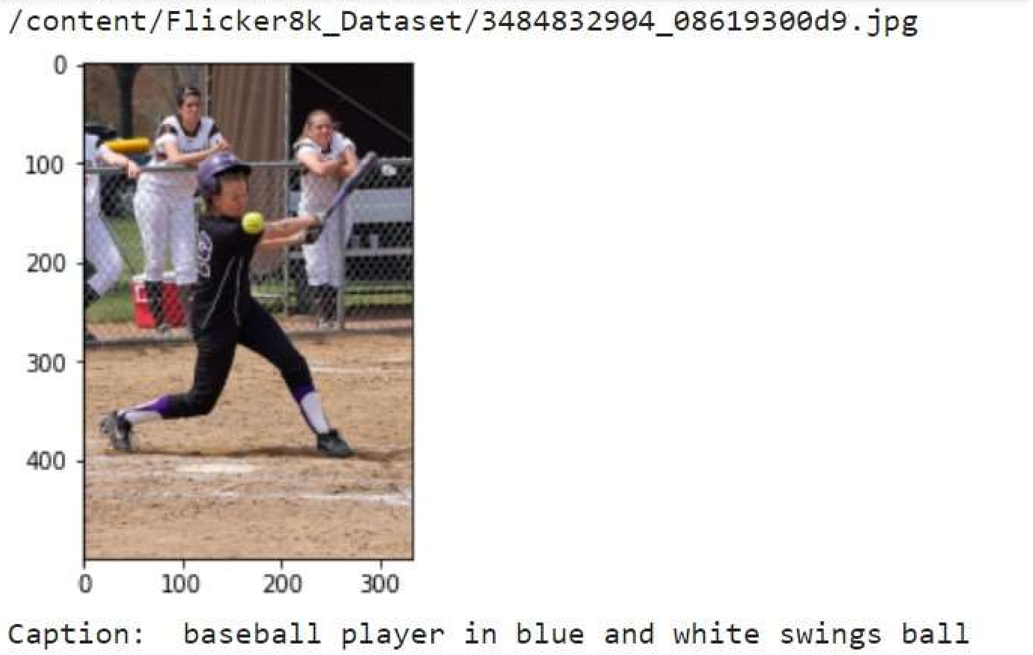


```
Caption:  two dogs are playing in the water
________________________________
```



```
/content/Flicker8k_Dataset/3462454965_a481809cea.jpg
```

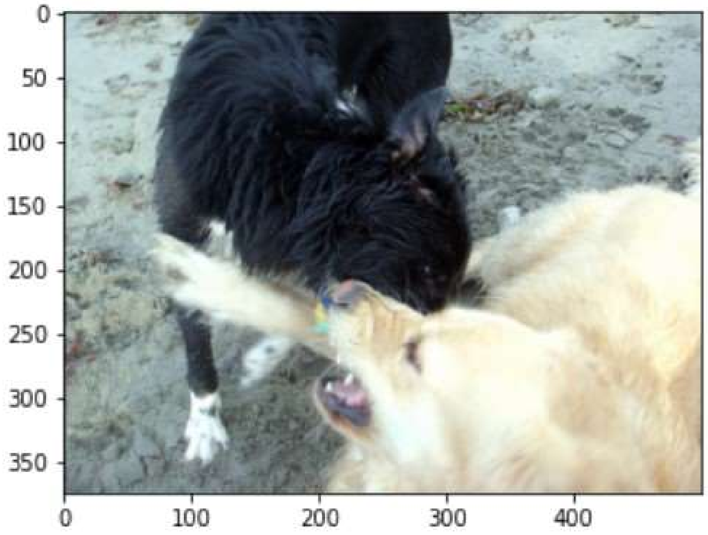
```
Caption:  two dogs are running in the grass
```



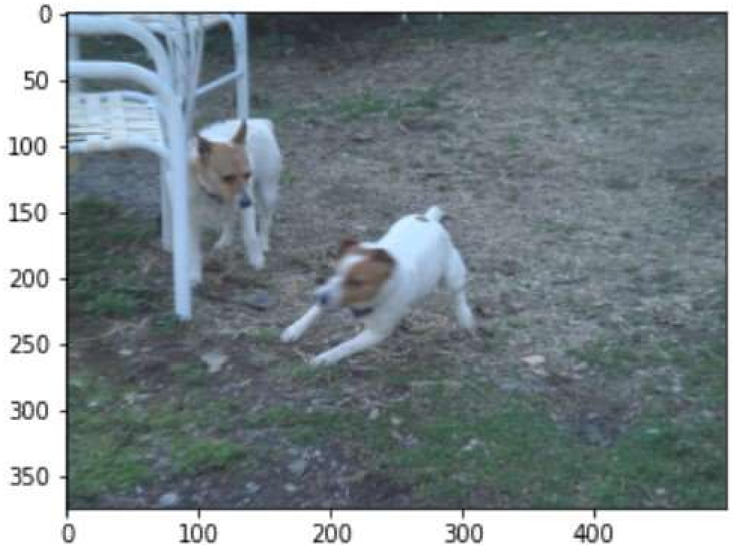
```
/content/Flicker8k_Dataset/3484832904_08619300d9.jpg
```

```
Caption:  baseball player in blue and white swings ball
```

```
/content/Flicker8k_Dataset/1897025969_0c41688fa6.jpg
```

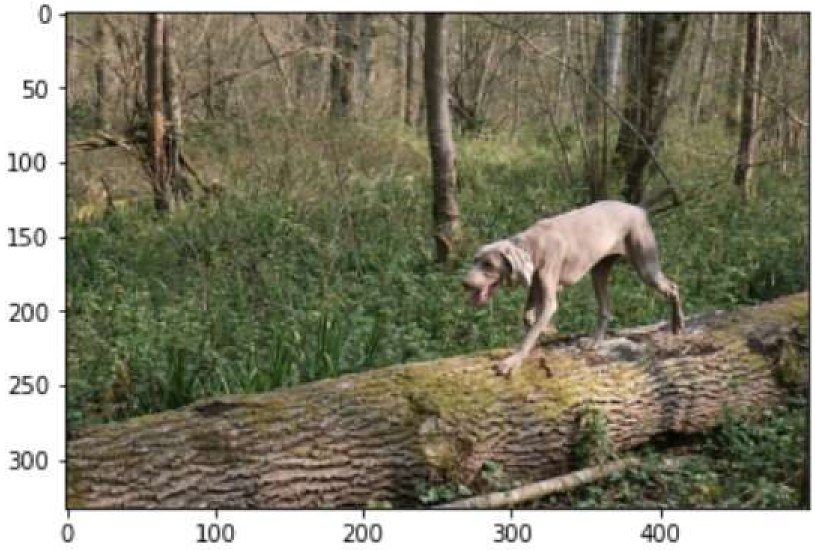


```
Caption:  black dog is playing with multicolored stuffed animal
______________________________
/content/Flicker8k_Dataset/2419221084_01a14176b4.jpg
```

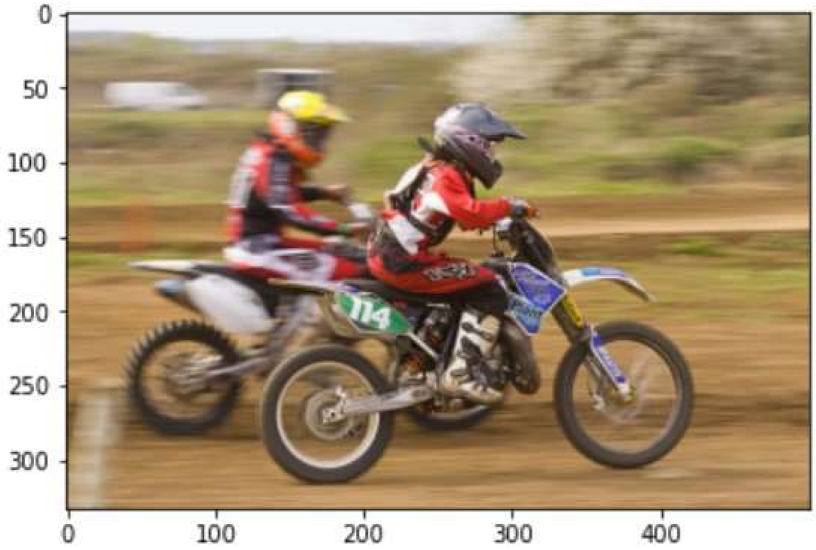


```
Caption:  two dogs are playing with each other in the grass
```

```
 /content/Flicker8k_Dataset/1554713437_61b64527dd.jpg
```



```
 Caption:  brown dog is running through the woods
_____________________________
/content/Flicker8k_Dataset/3437147889_4cf26dd525.jpg
```



```
Caption:  man on motorcycle is riding on track
```