In [1]:
```
from google.colab import drive
drive.mount("/content/drive/")
```

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

In [4]:
```
!wget --header="Host: storage.googleapis.com" --header="User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36" --header="Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9" --header="Accept-Language: en-US,en;q=0.9" --header="Referer: https://www.kaggle.com/" "https://storage.googleapis.com/kaggle-data-sets/623289/1111676/bundle/archive.zip?X-Goog-Algorithm=GOOG4-RSA-SHA256&X-Goog-Credential=gcp-kaggle-com%40kaggle-161607.iam.gserviceaccount.com%2F20210523%2Fauto%2Fstorage%2Fgoog4_request&X-Goog-Date=20210523T071250Z&X-Goog-Expires=259199&X-Goog-SignedHeaders=host&X-Goog-Signature=4f37286c429b274dd0fca35758b9f0bad4225fa4e8fcff1e366fdcea89b2e79b7587caf3859a1edbc71f7ff10d5a2df38d4eba9e8d77e8c34404d2576f0e2855e65ea1028e9b137bc331ec02b8626160930fdc4c3f0c393ae167ad58e7dc583d3b4d9e96d1f3de0e6700efbb27b16046459567790067b7361df6d4ef1e7daf7152290f09279ac23496514120309e10854ef13ac67978045c454adb58d34d20e5a17352d22f2281d2e5a7e11f3ec433d340a5a85049e7839d0b078c64c77af96983b9fb2fde5a4577770895bee02704f67abc05757f7f21371b06efe19e449e6924f9140bad0d25863b4be820ad37ac1271b0c50bcd6c971c0480bf562ba064d9" -c -O 'archive.zip'
```

```
--2021-05-23 07:13:15--  https://storage.googleapis.com/kaggle-data-sets/623289/1111676/bundle/archive.zip?X-Goog-Algorithm=GOOG4-RSA-SHA256&X-Goog-Credential=gcp-kaggle-com%40kaggle-161607.iam.gserviceaccount.com%2F20210523%2Fauto%2Fstorage%2Fgoog4_request&X-Goog-Date=20210523T071250Z&X-Goog-Expires=259199&X-Goog-SignedHeaders=host&X-Goog-Signature=4f37286c429b274dd0fca35758b9f0bad4225fa4e8fcff1e366fdcea89b2e79b7587caf3859a1edbc71f7ff10d5a2df38d4eba9e8d77e8c34404d2576f0e2855e65ea1028e9b137bc331ec02b8626160930fdc4c3f0c393ae167ad58e7dc583d3b4d9e96d1f3de0e6700efbb27b16046459567790067b7361df6d4ef1e7daf7152290f09279ac23496514120309e10854ef13ac67978045c454adb58d34d20e5a17352d22f2281d2e5a7e11f3ec433d340a5a85049e7839d0b078c64c77af96983b9fb2fde5a4577770895bee02704f67abc05757f7f21371b06efe19e449e6924f9140bad0d25863b4be820ad37ac1271b0c50bcd6c971c0480bf562ba064d9
Resolving storage.googleapis.com (storage.googleapis.com)... 108.177.119.128, 108.177.126.128, 108.177.127.128, ...
Connecting to storage.googleapis.com (storage.googleapis.com)|108.177.119.128|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1112971163 (1.0G) [application/zip]
Saving to: ‘archive.zip’

archive.zip         100%[===================>]   1.04G  65.6MB/s    in 18s

2021-05-23 07:13:34 (57.8 MB/s) - ‘archive.zip’ saved [1112971163/1112971163]
```

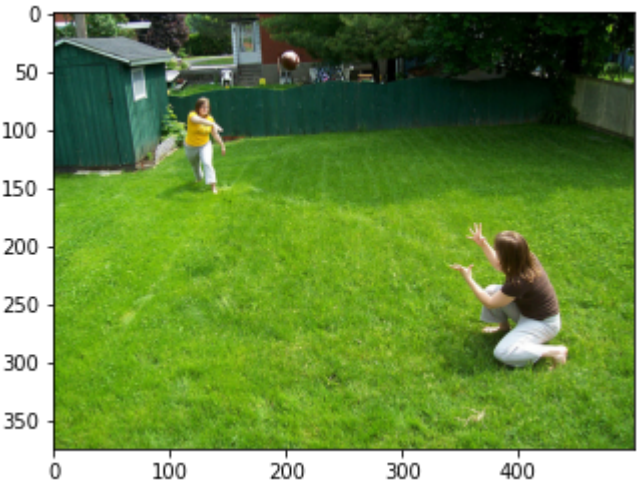
In [5]:
```
#Extracting the
from zipfile import ZipFile
with ZipFile("archive.zip","r") as zip:
  zip.extractall()
  print("done")
```

```
done
```

Processing The Image

In [6]:
```
import matplotlib.pyplot as plt
import cv2
import os
k=os.listdir("Images")
image_list=k[:2000]
image_name=k[:2000]
for i in range(len(image_list)):
  image_list[i]="Images/"+image_list[i]

img=cv2.imread("Images/534200447_b0f3ff02be.jpg")
img=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
plt.imshow(img)
```

Out[6]:
```
<matplotlib.image.AxesImage at 0x7f20662ffa10>
```



In [6]:
```

```

In [8]:
```
# Importing VGG16 for feature extraction from the images
from keras.applications import VGG16 ,ResNet50 ,InceptionV3
vgg=ResNet50(weights="imagenet")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applica
tions/resnet/resnet50_weights_tf_dim_ordering_tf_kernels.h5
102973440/102967424 [==============================] - 1s 0us/step
```

In [ ]:
```
from keras.models import Model
op = vgg.layers[-2].output
image_model = Model(inputs = vgg.input,outputs = op)
image_model.summary()
```

In [11]:
```
# we have to convert all the images to vector of (2048,1)
#for that I m writing a function which takes image and coverts it to vector

def img_2_vec(x):
  img=cv2.imread(x)
  img=cv2.resize(img, (224,224))
  img=img.reshape(1,224,224,3)
  img=image_model.predict(img)
  img=img.reshape(2048,)
  return img

img_vector=[]
for i in image_list:
  img_vector.append(img_2_vec(i))
```

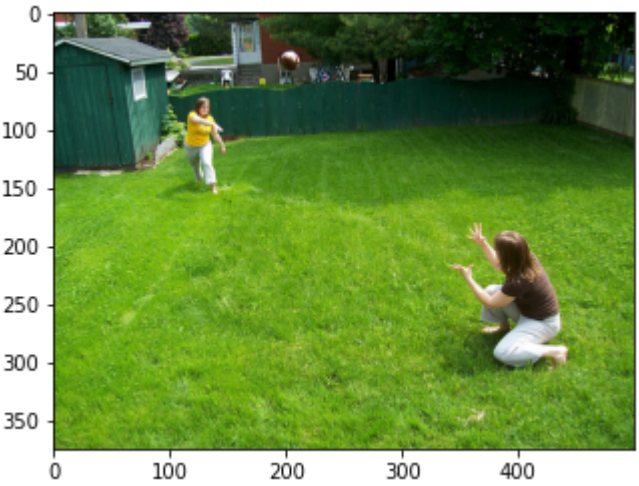
In [12]:
```
#saving image_name and its vector in dictionary
dict_vector={}
for i in range(len(image_list)):
    dict_vector[image_list[i].split("/")[1]]=img_vector[i]
```

In [14]:
```
#selecting only 2000 images
dict_vector=dict([(key ,dict_vector[key]) for key in image_name])
len(dict_vector)
```

Out[14]: 2000

In [15]:
```
# image and its feature vector
img=cv2.imread("Images/534200447_b0f3ff02be.jpg")
img=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
plt.imshow(img)
print("vector=",dict_vector[image_list[5].split("/")[1]])
```

```
vector= [0.582067   0.71971595 0.04135217 ... 0.776526   0.18743591 0.3457577
]
```



Processing Captions

In [16]:
```
# reading the Captions file
with open("captions.txt",'r') as file:
  text=file.readlines()
```

In [17]:
```
dict_captions={}
for i in text:
  name=i.split(",")[0]
  caption=i.split(",")[1]
  if name not in dict_captions:
    dict_captions[name]=[caption]
  else:
    dict_captions[name].append(caption)
dict_captions["534200447_b0f3ff02be.jpg"]

cap={}
for i in text:
  name=i.split(",")[0]
  caption=i.split(",")[1]
  if name not in cap:
    cap[name]=[caption]
  else:
    cap[name].append(caption)
```

In [18]:
```
dict_captions=dict([(key ,dict_captions[key]) for key in image_name])
len(dict_captions)
cap=dict([(key ,cap[key]) for key in image_name])
```

In [19]:
```
#Preprocessing the Captions
import re

def preprocess(d):
  for k,v in d.items():
    for i in range(len(v)):
      v[i]=re.sub('[^A-Za-z]+',' ',v[i])
      v[i]=v[i].lower()
      v[i]=v[i].strip()
      v[i]="start "+v[i]+" end"
  return d
dict_captions=preprocess(dict_captions)
cap=preprocess(cap)
dict_captions['489773343_a8aecf7db3.jpg']
```

Out[19]:
```
['start a small girl in pink and blue end',
 'start a young girl in a pink and blue wetsuit and oversized red flip flops
presents a little red ball to a big black dog while another dog looks on en
d',
 'start a young girl putting a red ball in a black dogs mouth on the beach en
d',
 'start the little girl in the sand is handing a ball to one of the two black
dogs end',
 'start two black dogs wait for a little girl to throw a ball at the beach en
d']
```

In [20]:
```
#Plotting the image and its captions
import cv2
import matplotlib.pyplot as plt
img=cv2.imread('Images/489773343_a8aecf7db3.jpg')
img=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
plt.imshow(img)
dict_captions['489773343_a8aecf7db3.jpg']
```

Out[20]:
```
['start a small girl in pink and blue end',
 'start a young girl in a pink and blue wetsuit and oversized red flip flops
presents a little red ball to a big black dog while another dog looks on en
d',
 'start a young girl putting a red ball in a black dogs mouth on the beach en
d',
 'start the little girl in the sand is handing a ball to one of the two black
dogs end',
 'start two black dogs wait for a little girl to throw a ball at the beach en
d']
```

In [21]:
```
#Creating Vocabulary for the words and giving  them the index/integer value
vocab=set()
for k,v in dict_captions.items():
  for i in range(len(v)):
    for j in v[i].split():
      vocab.add(j)
vocab=sorted(vocab)
vocabulary={}
for i,j in enumerate(vocab,1):
  vocabulary[j]=i

print("The Unique Number of words are ",len(vocab))


#trasforming these index to captions
for k,v in dict_captions.items():
  for i in v:
    t=[]
    for j in i.split():
      t.append(vocabulary[j])

    dict_captions[k][v.index(i)]=t

for i in dict_captions['489773343_a8aecf7db3.jpg']:
  print(i)
```

```
The Unique Number of words are  4300
[3569, 1, 3402, 1526, 1846, 2704, 80, 374, 1201]
[3569, 1, 4291, 1526, 1846, 1, 2704, 80, 374, 4178, 80, 2519, 2959, 1374, 139
4, 2815, 1, 2120, 2959, 207, 3862, 1, 333, 354, 1078, 4192, 85, 1078, 2146, 2
462, 1201]
[3569, 1, 4291, 1526, 2886, 1, 2959, 207, 1846, 1, 354, 1079, 2330, 2462, 381
5, 272, 1201]
[3569, 3815, 2120, 1526, 1846, 3815, 3113, 1900, 1659, 1, 207, 3862, 2464, 24
46, 3815, 3997, 354, 1079, 1201]
[3569, 3997, 354, 1079, 4100, 1429, 1, 2120, 1526, 3862, 3836, 1, 207, 146, 3
815, 272, 1201]
```

In [23]:
```
#finding the maxlenght of captions from train captions
max_len=[]
for k,v in dict_captions.items():
  for i in v:
    max_len.append(len(i))
max_length=max(max_len)
print("The maximum lenght of caption is",max(max_len))
```

```
The maximum lenght of caption is 33
```

Modelling

In [24]:
```
#splitting the according to images names
from sklearn.model_selection import train_test_split
train,test=train_test_split(image_name,test_size=0.2)
tr,cv=train_test_split(train,test_size=0.2)
```

In [25]:
```
# this function will be use
from keras.utils import to_categorical
from keras.preprocessing.sequence import pad_sequences
import numpy as np

VOCAB_SIZE = len(vocabulary)

def generator(image, caption):
  X = []
  y_in = []
  y_out = []
  for k, v in caption.items():
    for j in v:
      for i in range(1, len(j)):
        X.append(image[k])
        in_seq= [j[:i]]
        out_seq = [j[i]]
        in_seq = pad_sequences(in_seq, maxlen=max_length, padding='post', trun
cating='post')[0]
        out_seq = to_categorical([out_seq], num_classes=VOCAB_SIZE+1)[0]
        y_in.append(in_seq)
        y_out.append(out_seq)

  X=np.array(X)
  y_in=np.array(y_in)
  y_out=np.array(y_out)

  return X,y_in,y_out
```

In [26]:
```
#dict_captions.pop('image')
train_vector=dict([(key, dict_vector[key]) for key in tr])
train_captions=dict([(key, dict_captions[key]) for key in tr])

cv_vector=dict([(key, dict_vector[key]) for key in cv])
cv_captions=dict([(key, dict_captions[key]) for key in cv])

test_vector=dict([(key, dict_vector[key]) for key in test])
test_captions=dict([(key, dict_captions[key]) for key in test])
```

In [27]:
```
#dict_captions.pop('image')
train_x,train_in,train_out=generator(train_vector,train_captions)
cv_x,cv_in,cv_out=generator(cv_vector,cv_captions)
test_x,test_in,test_out=generator(test_vector,test_captions)
```

In [28]:
```
import pickle
embeddings_index = dict()
with open('/content/drive/My Drive/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())

essay_vocab = len(vocabulary) + 1
essay_matrix = np.zeros((essay_vocab, 300))
for word, i in vocabulary.items():
  embedding_vector = model.get(word)
  if embedding_vector is not None:
    essay_matrix[i] = embedding_vector
```

In [29]:
```
essay_matrix.shape
```

Out[29]:
```
(4301, 300)
```

In [ ]:
```
# model buliding
from keras.models import Model,Sequential
from keras.layers import Input,Dense,Embedding,LSTM,Dropout,add,Concatenate,Ba
tchNormalization ,Flatten ,RepeatVector ,TimeDistributed, Activation

from keras import regularizers


inputs1 = Input(shape=(2048,))
dense = Dense(512, activation='relu',kernel_initializer="HeUniform")(inputs1)
D1 = Dropout(0.5)(dense)
B1=BatchNormalization()(D1)
dense = Dense(256, activation='relu',kernel_initializer="HeUniform")(B1)
image_out=RepeatVector(max_length)(dense)


inputs2 = Input(shape=(max_length,))
E1 = Embedding(len(vocabulary)+1, 300, weights=[essay_matrix],trainable=False)
(inputs2)
D2 = Dropout(0.5)(E1)
lstm = LSTM(256 , return_sequences=True)(D2)
caption_out=TimeDistributed(Dense(256))(lstm)


concat = Concatenate()([image_out, caption_out])
x = LSTM(128, return_sequences=True)(concat)
x = LSTM(512, return_sequences=False)(x)
dense = Dense(256, activation='relu',kernel_initializer="HeUniform")(x)
D3=Dropout(0.5)(dense)
B3=BatchNormalization()(D3)
dense = Dense(256, activation='relu',kernel_initializer="HeUniform")(B3)
outputs = Dense(len(vocabulary)+1, activation='softmax')(dense)
model = Model(inputs=[inputs1, inputs2], outputs=outputs)
model.summary()
```

```
Model: "model_8"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_20 (InputLayer)           [(None, 2048)]       0
__________________________________________________________________________________________________
dense_36 (Dense)                (None, 512)          1049088     input_20[0][0]
__________________________________________________________________________________________________
input_21 (InputLayer)           [(None, 33)]         0
__________________________________________________________________________________________________
dropout_25 (Dropout)            (None, 512)          0           dense_36[0][0]
__________________________________________________________________________________________________
embedding_9 (Embedding)         (None, 33, 300)      1290300     input_21[0][0]
__________________________________________________________________________________________________
batch_normalization_16 (BatchNo (None, 512)          2048        dropout_25[0][0]
__________________________________________________________________________________________________
dropout_26 (Dropout)            (None, 33, 300)      0           embedding_9[0][0]
__________________________________________________________________________________________________
dense_37 (Dense)                (None, 256)          131328      batch_normalization_16[0][0]
__________________________________________________________________________________________________
lstm_12 (LSTM)                  (None, 33, 256)      570368      dropout_26[0][0]
__________________________________________________________________________________________________
repeat_vector_3 (RepeatVector)  (None, 33, 256)      0           dense_37[0][0]
__________________________________________________________________________________________________
time_distributed_3 (TimeDistrib (None, 33, 256)      65792       lstm_12[0][0]
__________________________________________________________________________________________________
concatenate_8 (Concatenate)     (None, 33, 512)      0           repeat_vector_3[0][0]
                                                                 time_distributed_3[0][0]
__________________________________________________________________________________________________
lstm_13 (LSTM)                  (None, 33, 128)      328192      concatenate_8[0][0]
__________________________________________________________________________________________________
```

```
_________________________
lstm_14 (LSTM)                (None, 512)          1312768     lstm_13[0]
[0]
_____________________________________________________________________________
_________________________
dense_39 (Dense)              (None, 256)          131328      lstm_14[0]
[0]
_____________________________________________________________________________
_________________________
dropout_27 (Dropout)          (None, 256)          0           dense_39[0]
[0]
_____________________________________________________________________________
_________________________
batch_normalization_17 (BatchNo (None, 256)          1024        dropout_27
[0][0]
_____________________________________________________________________________
_________________________
dense_40 (Dense)              (None, 256)          65792       batch_normal
ization_17[0][0]
_____________________________________________________________________________
_________________________
dense_41 (Dense)              (None, 4301)         1105357     dense_40[0]
[0]
=============================================================================
======================
Total params: 6,053,385
Trainable params: 4,761,549
Non-trainable params: 1,291,836
_____________________________________________________________________________
_________________________
```

In [ ]:
```
model.compile(loss='categorical_crossentropy', optimizer='adam',metrics=["accu
racy"])
model.fit([train_x, train_in], train_out, batch_size=512, epochs=20,validation
_data=([cv_x, cv_in], cv_out))
```

```
Epoch 1/20
144/144 [==============================] - 19s 102ms/step - loss: 5.9207 - ac
curacy: 0.1296 - val_loss: 5.1523 - val_accuracy: 0.1933
Epoch 2/20
144/144 [==============================] - 14s 96ms/step - loss: 4.7055 - acc
uracy: 0.1926 - val_loss: 4.8256 - val_accuracy: 0.1945
Epoch 3/20
144/144 [==============================] - 14s 97ms/step - loss: 4.5215 - acc
uracy: 0.2036 - val_loss: 4.6425 - val_accuracy: 0.2322
Epoch 4/20
144/144 [==============================] - 14s 98ms/step - loss: 4.3334 - acc
uracy: 0.2349 - val_loss: 4.4816 - val_accuracy: 0.2603
Epoch 5/20
144/144 [==============================] - 14s 100ms/step - loss: 4.0407 - ac
curacy: 0.2634 - val_loss: 4.2341 - val_accuracy: 0.2817
Epoch 6/20
144/144 [==============================] - 14s 100ms/step - loss: 3.8424 - ac
curacy: 0.2825 - val_loss: 4.1155 - val_accuracy: 0.2968
Epoch 7/20
144/144 [==============================] - 14s 101ms/step - loss: 3.7120 - ac
curacy: 0.2956 - val_loss: 4.0413 - val_accuracy: 0.3034
Epoch 8/20
144/144 [==============================] - 15s 101ms/step - loss: 3.6170 - ac
curacy: 0.3002 - val_loss: 4.1324 - val_accuracy: 0.2975
Epoch 9/20
144/144 [==============================] - 15s 102ms/step - loss: 3.5162 - ac
curacy: 0.3063 - val_loss: 3.9819 - val_accuracy: 0.3057
Epoch 10/20
144/144 [==============================] - 15s 103ms/step - loss: 3.4322 - ac
curacy: 0.3136 - val_loss: 3.9601 - val_accuracy: 0.3135
Epoch 11/20
144/144 [==============================] - 15s 103ms/step - loss: 3.3343 - ac
curacy: 0.3231 - val_loss: 3.9231 - val_accuracy: 0.3233
Epoch 12/20
144/144 [==============================] - 15s 102ms/step - loss: 3.2772 - ac
curacy: 0.3269 - val_loss: 3.8929 - val_accuracy: 0.3290
Epoch 13/20
144/144 [==============================] - 15s 102ms/step - loss: 3.2133 - ac
curacy: 0.3341 - val_loss: 3.8910 - val_accuracy: 0.3275
Epoch 14/20
144/144 [==============================] - 15s 102ms/step - loss: 3.1658 - ac
curacy: 0.3356 - val_loss: 3.9050 - val_accuracy: 0.3288
Epoch 15/20
144/144 [==============================] - 15s 102ms/step - loss: 3.1108 - ac
curacy: 0.3390 - val_loss: 3.8884 - val_accuracy: 0.3298
Epoch 16/20
144/144 [==============================] - 15s 102ms/step - loss: 3.0575 - ac
curacy: 0.3438 - val_loss: 3.9162 - val_accuracy: 0.3274
Epoch 17/20
144/144 [==============================] - 15s 103ms/step - loss: 2.9896 - ac
curacy: 0.3450 - val_loss: 3.8932 - val_accuracy: 0.3326
Epoch 18/20
144/144 [==============================] - 15s 103ms/step - loss: 2.9552 - ac
curacy: 0.3517 - val_loss: 3.9078 - val_accuracy: 0.3361
Epoch 19/20
144/144 [==============================] - 15s 103ms/step - loss: 2.9281 - ac
curacy: 0.3548 - val_loss: 3.8705 - val_accuracy: 0.3373
```

```
Epoch 20/20
144/144 [==============================] - 15s 103ms/step - loss: 2.8811 - ac
curacy: 0.3580 - val_loss: 3.9119 - val_accuracy: 0.3370
```

Out[ ]: <tensorflow.python.keras.callbacks.History at 0x7fd8414872d0>

In [ ]:
```
import tensorflow as tf
model.compile(loss='categorical_crossentropy', optimizer=tf.keras.optimizers.A
dam(learning_rate=0.0001),metrics=["accuracy"])
model.fit([train_x, train_in], train_out, batch_size=512, epochs=20,validation
_data=([test_x, test_in], test_out))
```

```
Epoch 1/20
144/144 [==============================] - 20s 111ms/step - loss: 2.7509 - ac
curacy: 0.3741 - val_loss: 3.6807 - val_accuracy: 0.3535
Epoch 2/20
144/144 [==============================] - 15s 105ms/step - loss: 2.7209 - ac
curacy: 0.3760 - val_loss: 3.6966 - val_accuracy: 0.3535
Epoch 3/20
144/144 [==============================] - 15s 106ms/step - loss: 2.6930 - ac
curacy: 0.3814 - val_loss: 3.7154 - val_accuracy: 0.3534
Epoch 4/20
144/144 [==============================] - 15s 106ms/step - loss: 2.6710 - ac
curacy: 0.3810 - val_loss: 3.7276 - val_accuracy: 0.3550
Epoch 5/20
144/144 [==============================] - 15s 105ms/step - loss: 2.6635 - ac
curacy: 0.3816 - val_loss: 3.7452 - val_accuracy: 0.3562
Epoch 6/20
144/144 [==============================] - 15s 105ms/step - loss: 2.6334 - ac
curacy: 0.3850 - val_loss: 3.7660 - val_accuracy: 0.3550
Epoch 7/20
144/144 [==============================] - 15s 105ms/step - loss: 2.6357 - ac
curacy: 0.3856 - val_loss: 3.7781 - val_accuracy: 0.3545
Epoch 8/20
144/144 [==============================] - 15s 105ms/step - loss: 2.6239 - ac
curacy: 0.3896 - val_loss: 3.7909 - val_accuracy: 0.3567
Epoch 9/20
144/144 [==============================] - 15s 105ms/step - loss: 2.6122 - ac
curacy: 0.3859 - val_loss: 3.8144 - val_accuracy: 0.3544
Epoch 10/20
144/144 [==============================] - 15s 105ms/step - loss: 2.6023 - ac
curacy: 0.3890 - val_loss: 3.8230 - val_accuracy: 0.3576
Epoch 11/20
144/144 [==============================] - 15s 105ms/step - loss: 2.5888 - ac
curacy: 0.3898 - val_loss: 3.8370 - val_accuracy: 0.3558
Epoch 12/20
144/144 [==============================] - 15s 105ms/step - loss: 2.5789 - ac
curacy: 0.3946 - val_loss: 3.8482 - val_accuracy: 0.3543
Epoch 13/20
144/144 [==============================] - 15s 105ms/step - loss: 2.5729 - ac
curacy: 0.3913 - val_loss: 3.8602 - val_accuracy: 0.3568
Epoch 14/20
144/144 [==============================] - 15s 105ms/step - loss: 2.5631 - ac
curacy: 0.3934 - val_loss: 3.8824 - val_accuracy: 0.3535
Epoch 15/20
144/144 [==============================] - 15s 105ms/step - loss: 2.5485 - ac
curacy: 0.3930 - val_loss: 3.8839 - val_accuracy: 0.3559
Epoch 16/20
144/144 [==============================] - 15s 105ms/step - loss: 2.5436 - ac
curacy: 0.3975 - val_loss: 3.9005 - val_accuracy: 0.3557
Epoch 17/20
144/144 [==============================] - 15s 105ms/step - loss: 2.5487 - ac
curacy: 0.3959 - val_loss: 3.9064 - val_accuracy: 0.3556
Epoch 18/20
144/144 [==============================] - 15s 105ms/step - loss: 2.5279 - ac
curacy: 0.3966 - val_loss: 3.9186 - val_accuracy: 0.3555
Epoch 19/20
144/144 [==============================] - 15s 106ms/step - loss: 2.5143 - ac
curacy: 0.3996 - val_loss: 3.9353 - val_accuracy: 0.3549
```

```
Epoch 20/20
144/144 [==============================] - 15s 105ms/step - loss: 2.5146 - ac
curacy: 0.3997 - val_loss: 3.9462 - val_accuracy: 0.3547
```

Out[ ]: <tensorflow.python.keras.callbacks.History at 0x7fd83fb03e50>

In [33]:
```
from keras.models import load_model
#model.save('/content/drive/My Drive/CASE Studies/Image captioning/model.h5')
model=load_model("/content/drive/My Drive/CASE Studies/Image captioning/model.
h5")
```

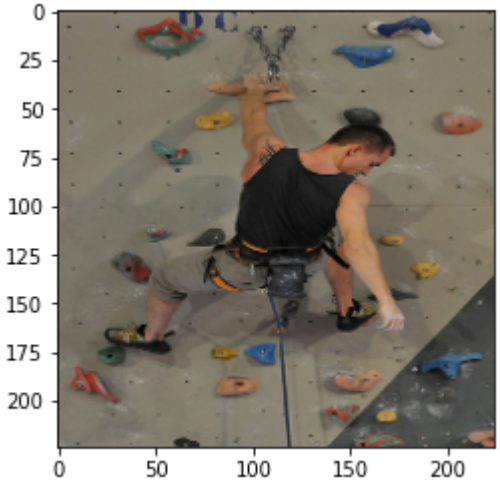
Prediction

In [34]:
```
# creating a dictionary whic is inverse of vocabulary
invert_vocab={v:k for k ,v in vocabulary.items()}
```

In [35]:
```
#funtion for making single prediction
def prediction(img):
  img = cv2.imread(img)
  img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
  img = cv2.resize(img, (224,224))
  plt.imshow(img)
  img = np.reshape(img, (1,224,224,3))


  x=image_model.predict(img).reshape(1,2048)

  y_in=["start"]
  caption=''

  for i in range(max_length):
    t=[]
    for j in y_in:

      t.append(vocabulary[j])

    input=pad_sequences([t], maxlen=max_length, padding='post', truncating='po
st')
    pred=np.argmax(model.predict([x,input]))

    caption=caption +" " +invert_vocab[pred]

    if invert_vocab[pred]=="end":
      break

    y_in.append(invert_vocab[pred])
  return caption
```

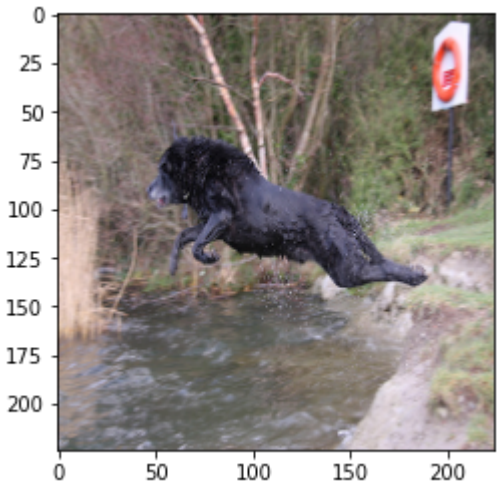
In [46]:
```
import nltk.translate.bleu_score as bleu
import warnings
warnings.filterwarnings('ignore')

img="Images/"+test[71]
caption=prediction(img)
caption=caption.replace("end","")
print("\n The caption generated is==>",caption)
print("\n The original captions are:" )
score=[]
for i in cap[test[71]]:
  print("\t\t",i)
  reference=i
  translation = caption
  BLEU_score= bleu.sentence_bleu(reference, translation)
  score.append(BLEU_score)

print("\nThe Best  Blue Score is ",max(score))
```

```
 The caption generated is==>  a boy is climbing a rock

 The original captions are:
                start a climber is climbing on an artificial climbing wall e
nd
                start a man attemping to climb a fake rock wall end
                start a man climbs a rock wall end
                start a man climbs up an artificial climbing wall end
                start a rock climber climbs a rock wall end

The Best  Blue Score is  0.8408964152537145
```



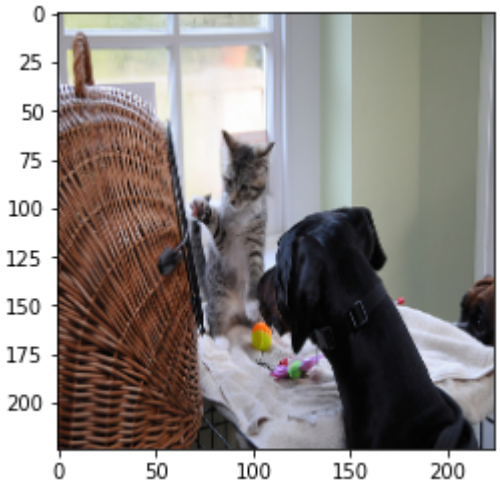
In [45]:
```
import nltk.translate.bleu_score as bleu
import warnings
warnings.filterwarnings('ignore')

img="Images/"+test[48]
caption=prediction(img)
print("\n The caption generated is==>",caption)
print("\n The original captions are:" )
score=[]
for i in cap[test[48]]:
  print("\t\t",i)
  reference=i
  translation = caption
  BLEU_score= bleu.sentence_bleu(reference, translation)
  score.append(BLEU_score)

print("\nThe Best  Blue Score is ",max(score))
```

```
 The caption generated is==>  a dog is jumping over a stick end

 The original captions are:
                start a black dog is leaping into a creek end
                start a black dog is leaping over a stream of running water
end
                start a black dog jumping off a river bank near a wooded are
a end
                start a wet black dog jumping into a lake end
                start a wet black dog leaps across a stream end

The Best  Blue Score is  0.8529987544592307
```



In [38]:
```
import nltk.translate.bleu_score as bleu
import warnings
warnings.filterwarnings('ignore')

img="Images/"+test[187]
caption=prediction(img)
print("\n The caption generated is==>",caption)
print("\n The original captions are:" )
score=[]
for i in cap[test[187]]:
  print("\t\t",i)
  reference=i
  translation = caption
  BLEU_score= bleu.sentence_bleu(reference, translation)
  score.append(BLEU_score)

print("\nThe Best  Blue Score is ",max(score))
```

```
 The caption generated is==>  a dog is laying on a bed end

 The original captions are:
                start a black dog is looking at the tabby cat that is sittin
g by the window end
                start a dog and cat sit by a table with toys on it end
                start a tawny cat is playing with a basket while a black dog
watches end
                start big black dog and kitten curiously looking at the wood
en weave basket end
                start the pets play in the apartment end

The Best  Blue Score is  0.8020396005825877
```



In [ ]:
```
import pickle
with open("/content/drive/My Drive/CASE Studies/Image captioning/vocabulary.pk
l","wb") as f:
  pickle.dump(vocabulary,f)
```

In [ ]:
```
import pickle
with open("/content/drive/My Drive/CASE Studies/Image captioning/invert_vocabu
lary.pkl","wb") as f:
  pickle.dump(invert_vocab,f)
```

In [ ]:
```
import nltk.translate.bleu_score as bleu
import warnings
warnings.filterwarnings('ignore')
s=[]
for j in test:
  img="Images/"+j
  caption=prediction(img)
  score=[]
  for i in cap[j]:
    reference=i
    translation = caption
    BLEU_score= bleu.sentence_bleu(reference, translation)
    score.append(BLEU_score)


  s.append(max(score))
```

In [40]:
```
s=np.array(s)
print("The Median BleuScore is ",np.median(s))
print("The Mean BleuScore is ",s.mean())
print("The Max BleuScore is ",max(s))
print("The Min BleuScore is ",min(s))
```

```
The Median BleuScore is  0.7561559858236034
The Mean BleuScore is  0.7328405376797648
The Max BleuScore is  0.8907817059279295
The Min BleuScore is  0.3932836841548814
```

In [ ]:
```

```