Import Libraries

In [1]:

```
import os
import pickle
import numpy as np
from tqdm.notebook import tqdm

#for preprocessing of images.
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
#for preprocessing of text.
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Model
from tensorflow.keras.utils import to_categorical, plot_model
#Importing layers
from tensorflow.keras.layers import Input, Dense, LSTM, Embedding, Dropout, add
```

Importing Dataset

In [2]:

```
BASE_DIR = r'C:\Users\Hp\Desktop\All Documents\Data'
```

Extract Image Features

In [3]:

```
# load vgg16 model
model = VGG16()
# Restructure the model
model = Model(inputs=model.inputs, outputs=model.layers[-2].output)
# summarize
print(model.summary())
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0         
                                                                 
 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792      
                                                                 
 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928     
                                                                 
 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0         
                                                                 
 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856     
                                                                 
 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584    
                                                                 
 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0         
                                                                 
 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168    
                                                                 
 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080    
                                                                 
 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080    
                                                                 
 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0         
                                                                 
 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160   
                                                                 
 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808   
                                                                 
 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808   
                                                                 
 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0         
                                                                 
 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808   
                                                                 
 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808   
                                                                 
 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808   
                                                                 
 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0         
                                                                 
 flatten (Flatten)           (None, 25088)             0         
                                                                 
 fc1 (Dense)                 (None, 4096)              102764544 
                                                                 
 fc2 (Dense)                 (None, 4096)              16781312  
                                                                 
=================================================================
Total params: 134,260,544
Trainable params: 134,260,544
Non-trainable params: 0
_________________________________________________________________
None
```

In [13]:

```python
# extract features from image
features = {}
#To get the directory for Images
directory = os.path.join(BASE_DIR, 'Images')

for img_name in tqdm(os.listdir(directory)):
    # load the image from file for getting each image name.
    img_path = directory + '/' + img_name
    image = load_img(img_path, target_size=(224, 224))
```

Loading [MathJax]/extensions/Safe.js

```
    # convert image pixels to numpy array
    image = img_to_array(image)
    # reshape data for model
    image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
    # preprocess image for vgg
    image = preprocess_input(image)
    # extract features
    feature = model.predict(image, verbose=0)
    # get image ID
    image_id = img_name.split('.')[0]
    # store feature
    features[image_id] = feature
```

```
  0%|          | 0/8091 [00:01<?, ?it/s]
```

Store and Load Features

In [ ]:

```
#Store features in pickle.
pickle.dump(features, open(os.path.join(BASE_DIR, 'features.pkl'), 'wb'))
```

In [4]:

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

In [5]:

```
model.summary()
```

Loading [MathJax]/extensions/Safe.js

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0         
                                                                 
 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792      
                                                                 
 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928     
                                                                 
 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0         
                                                                 
 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856     
                                                                 
 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584    
                                                                 
 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0         
                                                                 
 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168    
                                                                 
 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080    
                                                                 
 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080    
                                                                 
 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0         
                                                                 
 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160   
                                                                 
 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808   
                                                                 
 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808   
                                                                 
 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0         
                                                                 
 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808   
                                                                 
 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808   
                                                                 
 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808   
                                                                 
 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0         
                                                                 
 flatten (Flatten)           (None, 25088)             0         
                                                                 
 fc1 (Dense)                 (None, 4096)              102764544 
                                                                 
 fc2 (Dense)                 (None, 4096)              16781312  
                                                                 
=================================================================
Total params: 134,260,544
Trainable params: 134,260,544
Non-trainable params: 0
_________________________________________________________________
```

Load the Data of Caption

In [6]:

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

In [8]:

```
# create mapping of image to captions
```

Loading [MathJax]/extensions/Safe.js

```
# process lines
for line in tqdm(captions_doc.split('\n')):
    # split the line by comma(,)
    tokens = line.split(',')
    if len(line) < 2:
        continue
    image_id, caption = tokens[0], tokens[1:]
    # remove extension from image ID
    image_id = image_id.split('.')[0]
    # convert caption list to string
    caption = " ".join(caption)
    # create list if needed
    if image_id not in mapping:
        mapping[image_id] = []
    # store the caption
    mapping[image_id].append(caption)
```

```
  0%|          | 0/40456 [00:00<?, ?it/s]
```

In [9]:

```
#To check how many images.
len(mapping)
```

Out[9]:

```
8091
```

Preprocessing of Captions

In [10]:

```
#Create function of clean.
def clean(mapping):
    for key, captions in mapping.items():
        for i in range(len(captions)):
            # take one caption at a time
            caption = captions[i]
            # preprocessing steps
            # convert to lowercase
            caption = caption.lower()
            # delete digits, special chars, etc.
            caption = caption.replace('[^A-Za-z]', '')
            # delete additional spaces
            caption = caption.replace('\s+', ' ')
            # add start and end tags to the caption and remove single characters like a
            caption = 'startseq ' + " ".join([word for word in caption.split() if len(wo
            captions[i] = caption
```

In [11]:

```
# before preprocess of text
mapping['1000268201_693b08cb0e']
```

Out[11]:

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

In [12]:

```
# preprocess the text
clean(mapping)
```

In [13]:

```
# after preprocess of text
mapping['1000268201_693b08cb0e']
```

Out[13]:

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

Create Tokenizer and Vocab Size for Text Data

In [14]:

```
#Store all captions in single list.
all_captions = []
for key in mapping:
    for caption in mapping[key]:
        all_captions.append(caption)
```

In [15]:

```
#To check lenght of captions.
len(all_captions)
```

Out[15]:

```
40455
```

In [16]:

```
all_captions[:10]
```

Out[16]:

```
['startseq child in pink dress is climbing up set of stairs in an entry way endseq',
 'startseq girl going into wooden building endseq',
 'startseq little girl climbing into wooden playhouse endseq',
 'startseq little girl climbing the stairs to her playhouse endseq',
 'startseq little girl in pink dress going into wooden cabin endseq',
 'startseq black dog and spotted dog are fighting endseq',
 'startseq black dog and tri-colored dog playing with each other on the road endseq',
 'startseq black dog and white dog with brown spots are staring at each other in the str
eet endseq',
 'startseq two dogs of different breeds looking at each other on the road endseq',
 'startseq two dogs on pavement moving toward each other endseq']
```

In [17]:

```
# tokenize the text
#Initialize the tokenizer
tokenizer = Tokenizer()
tokenizer.fit_on_texts(all_captions)
vocab_size = len(tokenizer.word_index) + 1
```

In [18]:

```
vocab_size
```

Out[18]:

```
8485
```

In [19]:

```
# get maximum length of the caption available
max_length = max(len(caption.split()) for caption in all_captions)
max_length
```

Out[19]:

```
35
```

Train Test Split

In [20]:

```
#For getting image ids.
image_ids = list(mapping.keys())
split = int(len(image_ids) * 0.90)
split
train = image_ids[:split]
test = image_ids[split:]
```

Data Generator

Loading [MathJax]/extensions/Safe.js

In [21]:

```
# create data generator to get data in batch (avoids session crash)
def data_generator(data_keys, mapping, features, tokenizer, max_length, vocab_size, batc
    # loop over images
    X1, X2, y = list(), list(), list()
    #for determining wether we reach the batch size or not.
    n = 0
    while 1:
        for key in data_keys:
            n += 1
            captions = mapping[key]
            # process each caption
            for caption in captions:
                # encode the sequence, this will assign each word an index.
                seq = tokenizer.texts_to_sequences([caption])[0]
                # split the sequence into X (input), y (output) pairs.
                for i in range(1, len(seq)):
                    # split into input and output pairs
                    in_seq, out_seq = seq[:i], seq[i]
                    # pad input sequence
                    in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                    # encode output sequence
                    out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]

                    # store the sequences
                    X1.append(features[key][0])
                    X2.append(in_seq)
                    y.append(out_seq)
            if n == batch_size:
                X1, X2, y = np.array(X1), np.array(X2), np.array(y)
                #return the collected samples to generator.
                yield [X1, X2], y
                X1, X2, y = list(), list(), list()
                n = 0
```

In [89]:

```
# encoder model
# image feature layers
!pip install pydot
```

```
Collecting pydot
  Downloading pydot-1.4.2-py2.py3-none-any.whl (21 kB)
Requirement already satisfied: pyparsing>=2.1.4 in c:\users\hp\anaconda3\lib\site-packag
es (from pydot) (3.0.4)
Installing collected packages: pydot
Successfully installed pydot-1.4.2
('You must install pydot (`pip install pydot`) and install graphviz (see instructions at
https://graphviz.gitlab.io/download/) ', 'for plot_model/model_to_dot to work.')
```

Model Creation

In [22]:

```
import matplotlib.pyplot as plt
import graphviz
import pydot

#Encoder Model
#Image feature layers
#4096 is the shape of features.
inputs1 = Input(shape=(4096))
fe1 = Dropout(0.4)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)
# sequence feature layers
inputs2 = Input(shape=(max_length))
se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
```

Loading [MathJax]/extensions/Safe.js

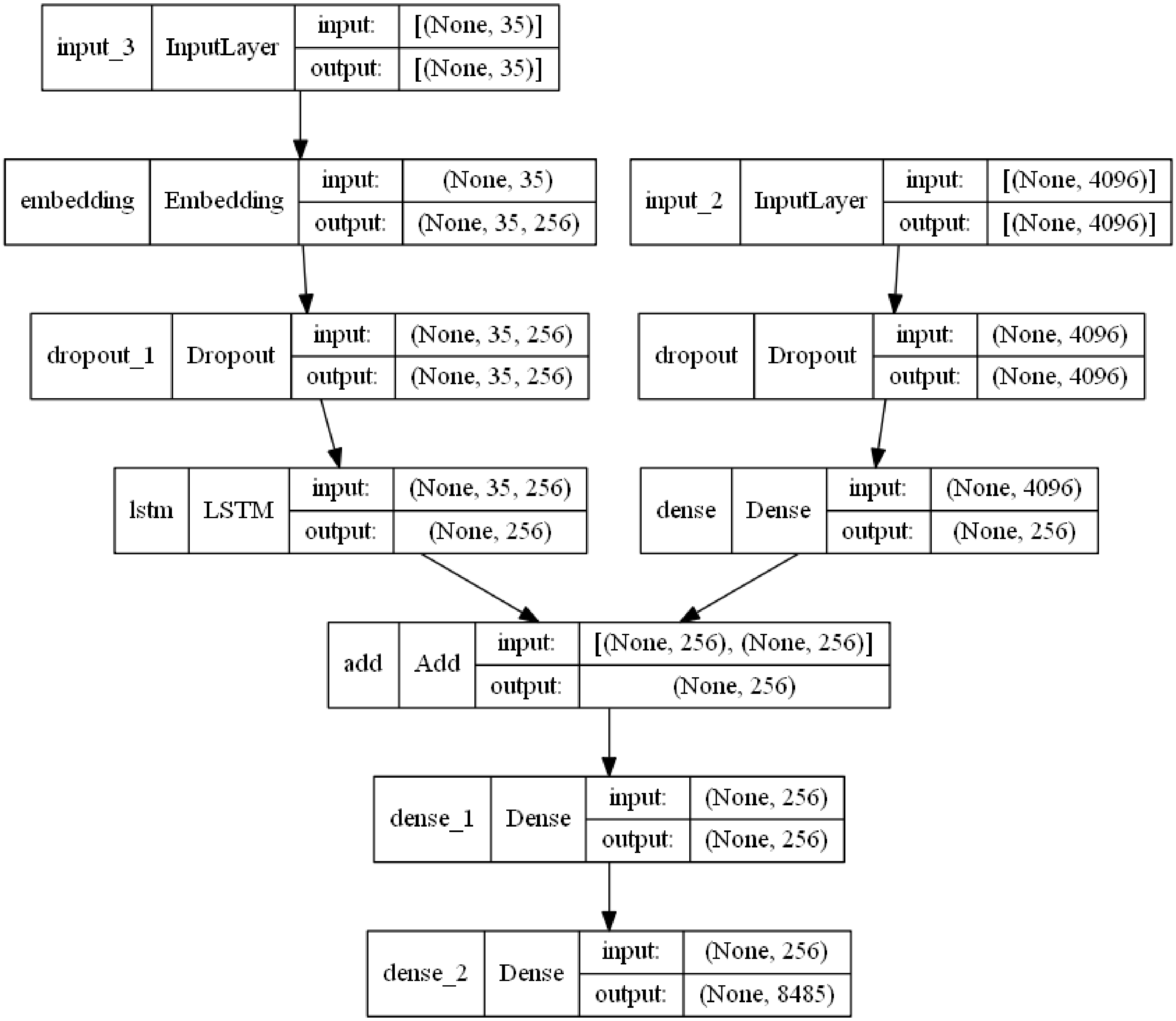
```python
#for avoiding overfitting we use 0.4 or 0.5.
se2 = Dropout(0.4)(se1)
se3 = LSTM(256)(se2)

# Decoder Model
#concatinating the image and text features.
decoder1 = add([fe2, se3])
decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(vocab_size, activation='softmax')(decoder2)

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

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

Out[22]:


input_3 | InputLayer | input: [(None, 35)] | output: [(None, 35)]
embedding | Embedding | input: (None, 35) | output: (None, 35, 256)
input_2 | InputLayer | input: [(None, 4096)] | output: [(None, 4096)]
dropout_1 | Dropout | input: (None, 35, 256) | output: (None, 35, 256)
dropout | Dropout | input: (None, 4096) | output: (None, 4096)
lstm | LSTM | input: (None, 35, 256) | output: (None, 256)
dense | Dense | input: (None, 4096) | output: (None, 256)
add | Add | input: [(None, 256), (None, 256)] | output: (None, 256)
dense_1 | Dense | input: (None, 256) | output: (None, 256)
dense_2 | Dense | input: (None, 256) | output: (None, 8485)

In [23]:

```python
model.summary()
```

Loading [MathJax]/extensions/Safe.js

```
Model: "model_1"
________________________________________________________________________________________
__________
 Layer (type)                   Output Shape         Param #     Connected to
========================================================================================
==========
 input_3 (InputLayer)           [(None, 35)]         0           []

 input_2 (InputLayer)           [(None, 4096)]       0           []

 embedding (Embedding)          (None, 35, 256)      2172160     ['input_3[0][0]']

 dropout (Dropout)              (None, 4096)         0           ['input_2[0][0]']

 dropout_1 (Dropout)            (None, 35, 256)      0           ['embedding[0][0]']

 dense (Dense)                  (None, 256)          1048832     ['dropout[0][0]']

 lstm (LSTM)                    (None, 256)          525312      ['dropout_1[0][0]']

 add (Add)                      (None, 256)          0           ['dense[0][0]',
                                                                  'lstm[0][0]']

 dense_1 (Dense)                (None, 256)          65792       ['add[0][0]']

 dense_2 (Dense)                (None, 8485)         2180645     ['dense_1[0][0]']

========================================================================================
==========
Total params: 5,992,741
Trainable params: 5,992,741
Non-trainable params: 0
________________________________________________________________________________________
__________
```

Train the Model

In [24]:

```
# train the model
epochs = 20
batch_size = 32
steps = len(train) // batch_size

for i in range(epochs):
    # create data generator
    generator = data_generator(train, mapping, features, tokenizer, max_length, vocab_si
    # fit for one epoch
    model.fit(generator, epochs=1, steps_per_epoch=steps, verbose=1)
```

```
227/227 [==============================] - 1816s 8s/step - loss: 5.2080
227/227 [==============================] - 1792s 8s/step - loss: 4.0111
227/227 [==============================] - 1776s 8s/step - loss: 3.5766
227/227 [==============================] - 1791s 8s/step - loss: 3.3080
227/227 [==============================] - 1798s 8s/step - loss: 3.1108
227/227 [==============================] - 1788s 8s/step - loss: 2.9623
227/227 [==============================] - 1784s 8s/step - loss: 2.8463
227/227 [==============================] - 1794s 8s/step - loss: 2.7546
227/227 [==============================] - 1803s 8s/step - loss: 2.6744
227/227 [==============================] - 1791s 8s/step - loss: 2.6035
227/227 [==============================] - 1803s 8s/step - loss: 2.5378
227/227 [==============================] - 1795s 8s/step - loss: 2.4787
227/227 [==============================] - 1787s 8s/step - loss: 2.4250
227/227 [==============================] - 1792s 8s/step - loss: 2.3776
227/227 [==============================] - 1788s 8s/step - loss: 2.3400
227/227 [==============================] - 1793s 8s/step - loss: 2.3008
227/227 [==============================] - 2785s 12s/step - loss: 2.2662
227/227 [==============================] - 1779s 8s/step - loss: 2.2419
227/227 [==============================] - 1785s 8s/step - loss: 2.2078
227/227 [==============================] - 1786s 8s/step - loss: 2.1818
```

Save the Model

In [70]:

```
import os.path
if os.path.isfile('BASE_DIR/save_model.h5') is False:
    model.save('BASE_DIR/save_model.h5')
```

Load the Model

In [24]:

```
from tensorflow.keras.models import load_model
new_model = load_model('BASE_DIR/save_model.h5')
```

In [25]:

```
new_model.summary()
```

```
Model: "model_1"
__________________________________________________________________________________________
__________
 Layer (type)                   Output Shape          Param #     Connected to
==========================================================================================
==========
 input_3 (InputLayer)           [(None, 35)]          0           []

 input_2 (InputLayer)           [(None, 4096)]        0           []

 embedding (Embedding)          (None, 35, 256)       2172160     ['input_3[0][0]']

 dropout (Dropout)              (None, 4096)          0           ['input_2[0][0]']

 dropout_1 (Dropout)            (None, 35, 256)       0           ['embedding[0][0]']

 dense (Dense)                  (None, 256)           1048832     ['dropout[0][0]']

 lstm (LSTM)                    (None, 256)           525312      ['dropout_1[0][0]']

 add (Add)                      (None, 256)           0           ['dense[0][0]',
                                                                   'lstm[0][0]']

 dense_1 (Dense)                (None, 256)           65792       ['add[0][0]']

 dense_2 (Dense)                (None, 8485)          2180645     ['dense_1[0][0]']

==========================================================================================
==========
Total params: 5,992,741
Trainable params: 5,992,741
Non-trainable params: 0
__________________________________________________________________________________________
__________
```

Generate Caption for Images

In [26]:

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

In [27]:

```
# generate caption for an image
def predict_caption(new_model, image, tokenizer, max_length):
    # add start tag for generation process
    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]
        # pad the sequence
        sequence = pad_sequences([sequence], max_length)
        # predict next word
        yhat = new_model.predict([image, sequence], verbose=0)
        # get index with high probability
        yhat = np.argmax(yhat)
        # convert index to word
        word = idx_to_word(yhat, tokenizer)
        # stop if word not found
Loading [MathJax]/extensions/Safe.js rd 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
```

Validation with Test data and Bleu Score

In [28]:

```
from nltk.translate.bleu_score import corpus_bleu
# validate with test data
actual, predicted = list(), list()

for key in tqdm(test):
    # get actual caption
    captions = mapping[key]
    # predict the caption for image
    y_pred = predict_caption(new_model, features[key], tokenizer, max_length)
    # split into words
    actual_captions = [caption.split() for caption in captions]
    y_pred = y_pred.split()
    # append to the list
    actual.append(actual_captions)
    predicted.append(y_pred)

# calcuate 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)))
```

```
  0%|          | 0/810 [00:00<?, ?it/s]
BLEU-1: 0.530205
BLEU-2: 0.308311
```

Visualize the Results

In [29]:

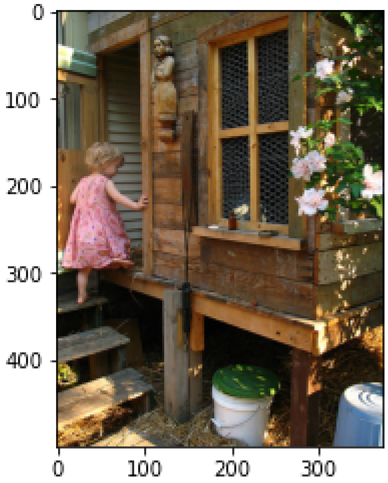
```
#Image for loading the image
from PIL import Image
import matplotlib.pyplot as plt
def generate_caption(image_name):
    # load the image
    # image_name = "1001773457_577c3a7d70.jpg"
    image_id = image_name.split('.')[0]
    img_path = os.path.join(BASE_DIR, "Images", image_name)
    image = Image.open(img_path)
    captions = mapping[image_id]
    print('---------------------Actual---------------------')
    for caption in captions:
        print(caption)
    # predict the caption
    y_pred = predict_caption(new_model, features[image_id], tokenizer, max_length)
    print('--------------------Predicted--------------------')
    print(y_pred)
    plt.imshow(image)
```

In [30]:

```
generate_caption("1000268201_693b08cb0e.jpg")
```

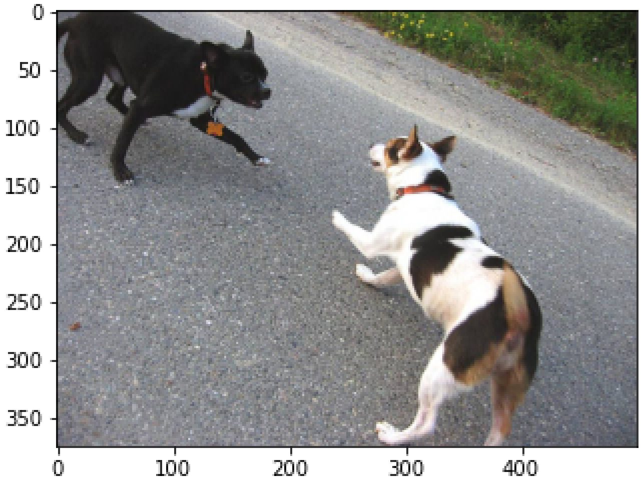
Loading [MathJax]/extensions/Safe.js

```
---------------------Actual---------------------
startseq child in pink dress is climbing up set of stairs in an entry way endseq
startseq girl going into wooden building endseq
startseq little girl climbing into wooden playhouse endseq
startseq little girl climbing the stairs to her playhouse endseq
startseq little girl in pink dress going into wooden cabin endseq
--------------------Predicted--------------------
startseq little girl in red dress is climbing the stairs endseq
```



In [31]:
```python
generate_caption("1001773457_577c3a7d70.jpg")
```

```
---------------------Actual---------------------
startseq black dog and spotted dog are fighting endseq
startseq black dog and tri-colored dog playing with each other on the road endseq
startseq black dog and white dog with brown spots are staring at each other in the stree
t endseq
startseq two dogs of different breeds looking at each other on the road endseq
startseq two dogs on pavement moving toward each other endseq
--------------------Predicted--------------------
startseq two dogs playing on the street endseq
```



In [32]:
```python
generate_caption("1007320043_627395c3d8.jpg")
```

```
---------------------Actual---------------------
startseq child playing on rope net endseq
startseq little girl climbing on red roping endseq
startseq little girl in pink climbs rope bridge at the park endseq
startseq small child grips onto the red ropes at the playground endseq
startseq the small child climbs on red ropes on playground endseq
--------------------Predicted--------------------
startseq little girl grips the equipment on the playground endseq
```

In [33]:
```
generate_caption("1009434119_febe49276a.jpg")
```

```
---------------------Actual---------------------
startseq black and white dog is running in grassy garden surrounded by white fence endse
q
startseq black and white dog is running through the grass endseq
startseq boston terrier is running in the grass endseq
startseq boston terrier is running on lush green grass in front of white fence endseq
startseq dog runs on the green grass near wooden fence endseq
--------------------Predicted--------------------
startseq dog jumps over an obstacle endseq
```