Importing the dependencies

In [1]:
```
import numpy as np # linear algebra
import os # for file hierarchy
import matplotlib.pyplot as plt # plotting the images and data
import matplotlib.image as mpimg # reading the images
from  sklearn.model_selection import train_test_split # for preprocessing and cutting whole dataset into train and test subsets
from PIL import Image # for loading the images
import cv2 # for reading the images and converting the images into various forms
import glob # for getting the files from directory
```

Getting the dataset in list.

In [2]:
```
dirpath = '../input/kaggle-cat-vs-dog-dataset/kagglecatsanddogs_3367a/PetImages'
files = next(os.walk(dirpath))
dogs = next(os.walk(dirpath +'/Dog'))
cats = next(os.walk(dirpath +'/Cat'))
dogs = dogs[2]
cats = cats[2]
```

Counting the number of images in dataset

In [3]:
```
print(f'The number of dogs images is {len(dogs)}')
print(f'The number of cat images is {len(cats)}')
print(f'Total number of images in dataset is {len(cats) + len(dogs)}')
```

```
The number of dogs images is 12470
The number of cat images is 12491
Total number of images in dataset is 24961
```

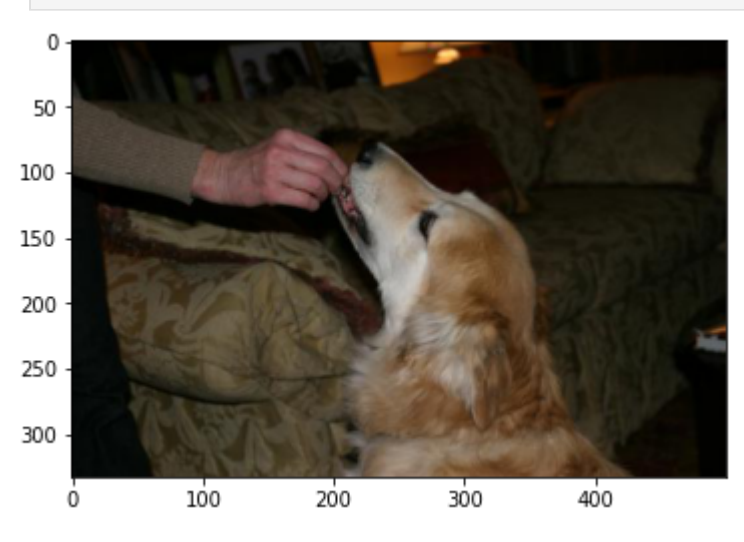
Cutting the dataset into smaller set

In [4]:
```
dogs = dogs[:3000]
cats = cats[:3000]
print(f'The number of dogs images after cutting is {len(dogs)}')
print(f'The number of cat images after cutting is {len(cats)}')
print(f'Total number of images in dataset is {len(cats) + len(dogs)}')
```

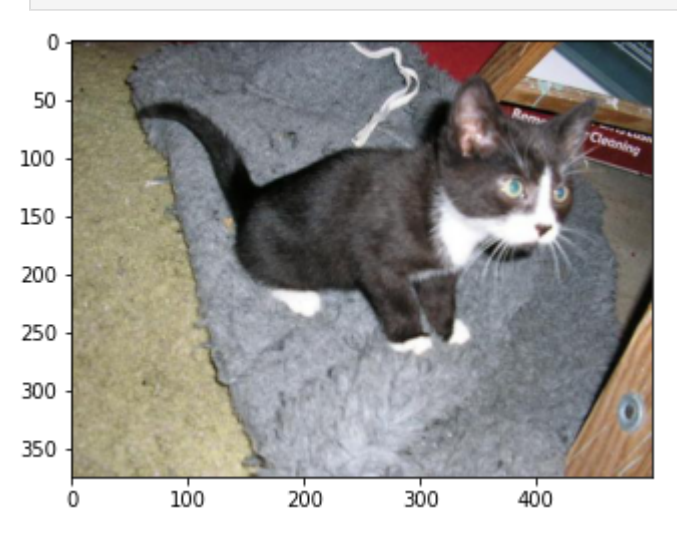
```
The number of dogs images after cutting is 3000
The number of cat images after cutting is 3000
Total number of images in dataset is 6000
```

Displaying the sample Images

In [5]:
```
%matplotlib inline
img = mpimg.imread(f'{dirpath}/Dog/{dogs[2]}')
imgplot = plt.imshow(img)
```



In [6]:
```
%matplotlib inline
img = mpimg.imread(f'{dirpath}/Cat/{cats[2]}')
imgplot = plt.imshow(img)
```



Resizing the images

In [7]:
```
# creating a saparate folder
import shutil

# delete the folder hiarchy if the resized is already runned
if os.path.isdir('./resized'):
    path = './resized'
    shutil.rmtree(path)
os.mkdir('./resized')
os.mkdir('./resized/dog')
os.mkdir('./resized/cat')
```

Resizing the dog images

In [8]:
```
resized_dog_path = './resized/dog'
resized_cat_path = './resized/cat'
c = 0

for i in range(3000):
    oImg = f'{dirpath}/Dog/{dogs[i]}'
    img = Image.open(oImg)
    img = img.resize((224,224))
    img = img.convert('RGB')
    img.save(f'{resized_dog_path}/{c}.jpg')
    c+=1
print('completed resizing for dogs')
```

```
/opt/conda/lib/python3.7/site-packages/PIL/TiffImagePlugin.py:845: UserWarning: Truncated File Read
  warnings.warn(str(msg))
completed resizing for dogs
```

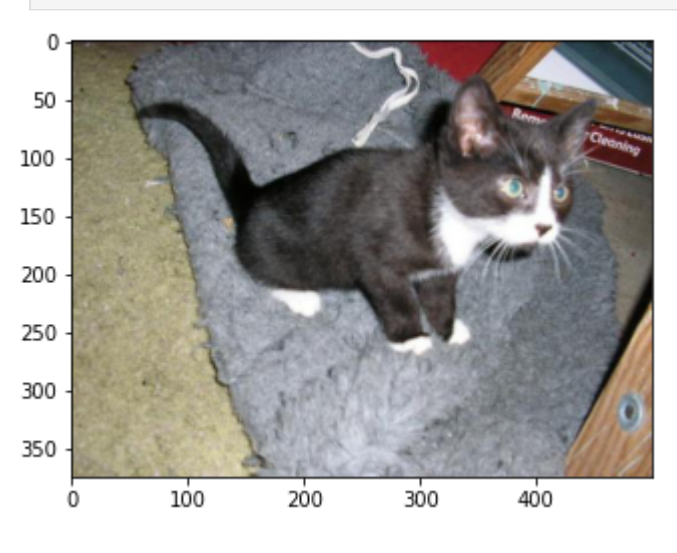
Resizing the cat images

In [9]:
```
c = 0
for i in range(3000):
    oImg = f'{dirpath}/Cat/{cats[i]}'
    img = Image.open(oImg)
    img = img.resize((224,224))
    img = img.convert('RGB')
    img.save(f'{resized_cat_path}/{c}.jpg')
    c+=1
print('completed resizing for cats')
```

```
completed resizing for cats
```

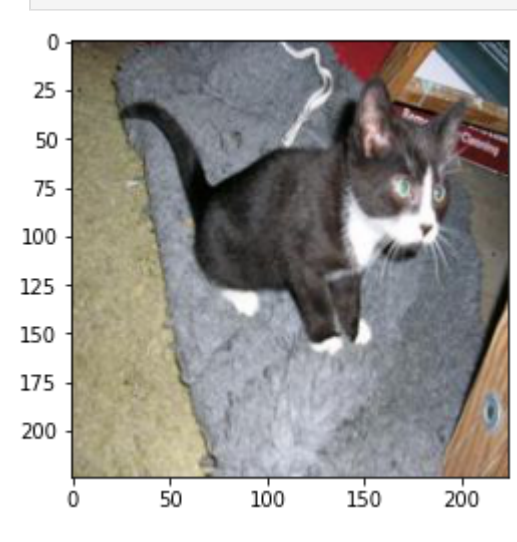
Original image

In [10]:
```
%matplotlib inline
img = mpimg.imread(f'{dirpath}/Cat/{cats[2]}')
imgplot = plt.imshow(img)
```



Resized Image

In [11]:
```
%matplotlib inline
img = mpimg.imread(f'{resized_cat_path}/2.jpg')
imgplot = plt.imshow(img)
```



Creating labels for dataset

In [12]:
```
# cats -> 0
# dogs -> 1
resized_dogs = os.listdir('./resized/dog')
labels=[]
for i in resized_dogs:
    labels.append(1)

resized_cats = os.listdir('./resized/cat')
for i in resized_cats:
    labels.append(0)
```

Finding the unique value and count of individual subsets

In [13]:
```
val , cnt = np.unique(labels,return_counts=True)
print(f'The value is {val[0]} and count of that is {cnt[0]}')
print(f'The value is {val[1]} and count of that is {cnt[1]}')
```

```
The value is 0 and count of that is 3000
The value is 1 and count of that is 3000
```

Creating a numpy array of all the images present

In [14]:
```
allImgPath = './resized/dog/'
img_extension = ['png','jpg']
files = []

[files.extend(glob.glob(allImgPath + '*.' + e)) for e in img_extension]


allImgPath = './resized/cat/'
[files.extend(glob.glob(allImgPath + '*.' + e)) for e in img_extension]

allImg = np.asarray([cv2.imread(file) for file in files])
```

Getting information from numpy array 'allImg' that consist all the images in matrix form

In [15]:
```
print(allImg.shape)
print(allImg.dtype)
print(type(allImg))
print(len(allImg))
```

```
(6000, 224, 224, 3)
uint8
<class 'numpy.ndarray'>
6000
```

Train test split

In [16]:
```
X = allImg
Y = np.asarray(labels)
```

In [17]:
```
x_train , x_test , y_train , y_test = train_test_split(X,Y,test_size=0.1,random_state=2)
```

Getting the insights of train test split

In [18]:
```
print (f'The X images are splitted into train set : {len(x_train)} test set : {len(x_test)}')
print (f'The Y labels are splitted into train set : {len(y_train)} test set : {len(y_test)}')
print(x_train[0])
```

```
The X images are splitted into train set : 5400 test set : 600
The Y labels are splitted into train set : 5400 test set : 600
[[[177 163 167]
  [177 163 167]
  [176 162 166]
  ...
  [176 163 165]
  [177 164 166]
  [178 165 167]]

 [[180 166 170]
  [180 166 170]
  [179 165 169]
  ...
  [177 164 166]
  [179 166 168]
  [180 167 169]]

 [[180 169 172]
  [179 168 171]
  [178 167 170]
  ...
  [178 165 167]
  [181 168 170]
  [183 170 172]]

 ...

 [[207 208 212]
  [206 207 211]
  [203 203 209]
  ...
  [209 207 213]
  [206 204 210]
  [201 199 205]]

 [[213 214 218]
  [210 211 215]
  [207 207 213]
  ...
  [202 200 206]
  [196 194 200]
  [191 189 195]]

 [[212 213 217]
  [209 210 214]
  [209 209 215]
  ...
  [203 201 207]
  [199 197 203]
  [196 194 200]]]
```

scaling the data by dividing with 255

In [19]:
```
x_train = x_train/255
x_test = x_test/255
```

In [20]:
```
print(x_train[0])
```

```
[[[0.69411765 0.63921569 0.65490196]
  [0.69411765 0.63921569 0.65490196]
  [0.69019608 0.63529412 0.65098039]
  ...
  [0.69019608 0.63921569 0.64705882]
  [0.69411765 0.64313725 0.65098039]
  [0.69803922 0.64705882 0.65490196]]

 [[0.70588235 0.65098039 0.66666667]
  [0.70588235 0.65098039 0.66666667]
  [0.70196078 0.64705882 0.6627451 ]
  ...
  [0.69411765 0.64313725 0.65098039]
  [0.70196078 0.65098039 0.65882353]
  [0.70588235 0.65490196 0.6627451 ]]

 [[0.70588235 0.6627451  0.6745098 ]
  [0.70196078 0.65882353 0.67058824]
  [0.69803922 0.65490196 0.66666667]
  ...
  [0.69803922 0.64705882 0.65490196]
  [0.70980392 0.65882353 0.66666667]
  [0.71764706 0.66666667 0.6745098 ]]

 ...

 [[0.81176471 0.81568627 0.83137255]
  [0.80784314 0.81176471 0.82745098]
  [0.79607843 0.79607843 0.81960784]
  ...
  [0.81960784 0.81176471 0.83529412]
  [0.80784314 0.8        0.82352941]
  [0.78823529 0.78039216 0.80392157]]

 [[0.83529412 0.83921569 0.85490196]
  [0.82352941 0.82745098 0.84313725]
  [0.81176471 0.81176471 0.83529412]
  ...
  [0.79215686 0.78431373 0.80784314]
  [0.76862745 0.76078431 0.78431373]
  [0.74901961 0.74117647 0.76470588]]

 [[0.83137255 0.83529412 0.85098039]
  [0.81960784 0.82352941 0.83921569]
  [0.81960784 0.81960784 0.84313725]
  ...
  [0.79607843 0.78823529 0.81176471]
  [0.78039216 0.77254902 0.79607843]
  [0.76862745 0.76078431 0.78431373]]]
```

building a neural network

In [21]:
```
import tensorflow as tf
import tensorflow_hub as hub
```

In [22]:
```
mobilenet_model = '../input/tensorflow/'

pretrained_model = hub.KerasLayer(mobilenet_model, input_shape=(224,224,3), trainable=False)
```

```
2022-11-16 13:27:17.668015: I tensorflow/core/common_runtime/process_util.cc:146] Creating new thread pool with default inter op setting: 2. Tune using inter_op_parallelism_threads for best performance.
```

In [23]:
```
num_of_classes = 2

model = tf.keras.Sequential([

    pretrained_model,
    tf.keras.layers.Dense(num_of_classes)

])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
keras_layer (KerasLayer)     (None, 1280)              2257984
_________________________________________________________________
dense (Dense)                (None, 2)                 2562
=================================================================
Total params: 2,260,546
Trainable params: 2,562
Non-trainable params: 2,257,984
_________________________________________________________________
```

In [24]:
```
model.compile(
    optimizer = 'adam',
    loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics = ['acc']
)
```

In [25]:
```
model.fit(x_train,y_train, epochs=5)
```

```
2022-11-16 13:27:32.481289: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR Optimization Passes are enabled (registered 2)
Epoch 1/5
169/169 [==============================] - 59s 327ms/step - loss: 0.1249 - acc: 0.9569
Epoch 2/5
169/169 [==============================] - 52s 307ms/step - loss: 0.0550 - acc: 0.9809
Epoch 3/5
169/169 [==============================] - 51s 301ms/step - loss: 0.0424 - acc: 0.9854
Epoch 4/5
169/169 [==============================] - 52s 307ms/step - loss: 0.0355 - acc: 0.9898
Epoch 5/5
169/169 [==============================] - 52s 305ms/step - loss: 0.0303 - acc: 0.9913
```

Out[25]:
```
<keras.callbacks.History at 0x7f61ec9a9790>
```

In [26]:
```
score , acc = model.evaluate(x_test,y_test)
```

```
19/19 [==============================] - 7s 302ms/step - loss: 0.0608 - acc: 0.9783
```

In [27]:
```
print(f'Accuracy of test dataset is : {int(acc*100)} %')
```

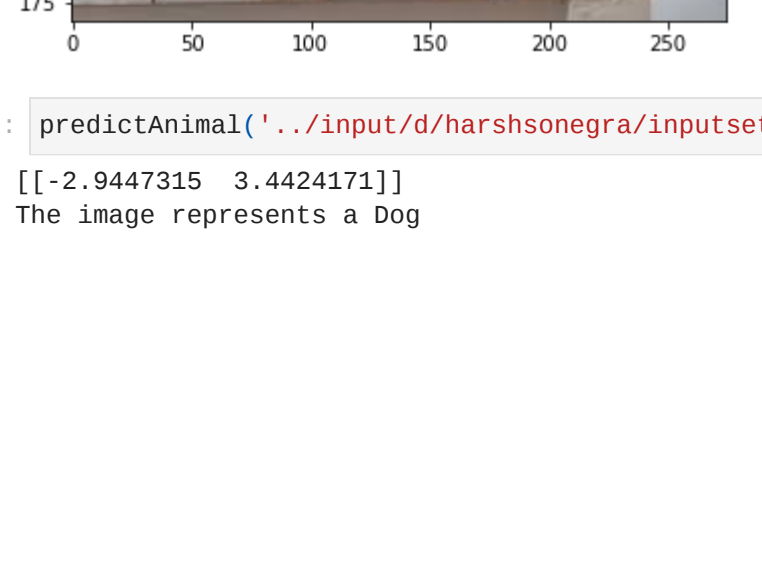
```
Accuracy of test dataset is : 97 %
```

prediction of unidentified image

In [28]:
```
def predictAnimal(str):
    input_image = mpimg.imread(str)
    imgplot = plt.imshow(input_image)
    input_image_resize = cv2.resize(input_image, (224,224))
    input_image_scaled = input_image_resize/255
    image_reshaped = np.reshape(input_image_scaled, [1,224,224,3])
    input_prediction = model.predict(image_reshaped)
    input_pred_label = np.argmax(input_prediction)
    print(input_prediction)
    if input_pred_label == 0:
      print('The image represents a Cat')
    else:
      print('The image represents a Dog')
```

In [29]:
```
predictAnimal('../input/d/harshsonegra/inputset/cat.jpg')
```

```
[[ 5.289031  -2.2416222]]
The image represents a Cat
```



In [30]:
```
predictAnimal('../input/d/harshsonegra/inputset/dog.jpg')
```

```
[[-2.9447315  3.4424171]]
The image represents a Dog
```


0
25
50
75
100
125
150
175
0
50
100
150
200
250