importing lab

In [1]:

```
'''image data generator tries to generate multiple data from
single image like by shrinking,zooming/croping etc.'''
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential #we are using sequential model for classification.
'''Conv is used to exract featurs from images
max pooling reduce the size of image without loosing it's feature  '''
from keras.layers import Conv2D, MaxPooling2D
'''Activation function used to tell the network when to activate neuron
by using Dropout our network doesn't overfit
Flatten convert 2d image  into 1d vector
Dense is used to create hidden and output layer'''
from keras.layers import Activation, Dropout,Flatten,Dense
''' backend tells us which channel is come first '''
from keras import backend as k
import numpy as np
from keras_preprocessing import image
```

```
Using TensorFlow backend.
```

Diamensions of the image

In [2]:

```
img_width, img_height =500,500
train_data_dir ='train/Train2'
test_data_dir ='test1/Test1'
nb_train_samples= 1000
nb_test_saples =100
epochs= 50
batch_size =20
```

In [3]:

```
%pwd
```

Out[3]:

```
'C:\\Users\\hp\\cats vs dogs'
```

In [4]:

```
#chech images are in right format or not
if k.image_data_format() == 'channels_first':
    input_shape = (3,img_width,img_height)
else:
    input_shape = (img_width,img_height,3)

train_datagen=ImageDataGenerator(rescale=1.0/255.0,
                                shear_range=0.2,
                                zoom_range=0.2,
                                horizontal_flip=True)
test_datagen=ImageDataGenerator(rescale =1.0/255.0)
```

In [5]:

```
train_generator = train_datagen.flow_from_directory(train_data_dir,
                                                   target_size=(img_width,img_height),
                                                   batch_size=batch_size,
                                                   class_mode='binary')

test_generator = test_datagen.flow_from_directory(train_data_dir,
                                                   target_size=(img_width,img_height),
                                                   batch_size=batch_size,
                                                   class_mode='binary')
```

```
Found 25000 images belonging to 2 classes.
Found 25000 images belonging to 2 classes.
```

In [6]:

```
model =Sequential()
model.add(Conv2D(32,(3,3),input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.summary()


model.add(Conv2D(32,(3,3),input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(32,(3,3),input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.summary()

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 498, 498, 32)      896       
_________________________________________________________________
activation_1 (Activation)    (None, 498, 498, 32)      0         
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 249, 249, 32)      0         
=================================================================
Total params: 896
Trainable params: 896
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 498, 498, 32)      896       
_________________________________________________________________
activation_1 (Activation)    (None, 498, 498, 32)      0         
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 249, 249, 32)      0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 247, 247, 32)      9248      
_________________________________________________________________
activation_2 (Activation)    (None, 247, 247, 32)      0         
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 123, 123, 32)      0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 121, 121, 32)      9248      
_________________________________________________________________
activation_3 (Activation)    (None, 121, 121, 32)      0         
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 60, 60, 32)        0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 115200)            0         
_________________________________________________________________
dense_1 (Dense)              (None, 64)                7372864   
_________________________________________________________________
activation_4 (Activation)    (None, 64)                0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65        
_________________________________________________________________
activation_5 (Activation)    (None, 1)                 0         
=================================================================
Total params: 7,392,321
Trainable params: 7,392,321
Non-trainable params: 0
_________________________________________________________________
```

In [7]:

```
model.fit_generator(train_generator,
                    steps_per_epoch=nb_train_samples // batch_size,
                    epochs=epochs,
                    validation_data=test_generator,
                    validation_steps=nb_test_saples// batch_size)
model.save_weights('first_try.h5')
```

```
Epoch 1/50
50/50 [==============================] - 219s 4s/step - loss: 1.0565 - acc
uracy: 0.5340 - val_loss: 0.6713 - val_accuracy: 0.6300
Epoch 2/50
50/50 [==============================] - 228s 5s/step - loss: 0.7415 - acc
uracy: 0.5750 - val_loss: 0.6874 - val_accuracy: 0.5300
Epoch 3/50
50/50 [==============================] - 216s 4s/step - loss: 0.7083 - acc
uracy: 0.5810 - val_loss: 0.6948 - val_accuracy: 0.5600
Epoch 4/50
50/50 [==============================] - 216s 4s/step - loss: 0.7004 - acc
uracy: 0.5800 - val_loss: 0.6521 - val_accuracy: 0.6500
Epoch 5/50
50/50 [==============================] - 216s 4s/step - loss: 0.7008 - acc
uracy: 0.6260 - val_loss: 0.5464 - val_accuracy: 0.6200
Epoch 6/50
50/50 [==============================] - 219s 4s/step - loss: 0.6710 - acc
uracy: 0.6150 - val_loss: 0.6181 - val_accuracy: 0.6500
Epoch 7/50
50/50 [==============================] - 216s 4s/step - loss: 0.6356 - acc
uracy: 0.6380 - val_loss: 0.6432 - val_accuracy: 0.5900
Epoch 8/50
50/50 [==============================] - 215s 4s/step - loss: 0.6597 - acc
uracy: 0.6510 - val_loss: 0.5824 - val_accuracy: 0.6900
Epoch 9/50
50/50 [==============================] - 216s 4s/step - loss: 0.6400 - acc
uracy: 0.6440 - val_loss: 0.6862 - val_accuracy: 0.6700
Epoch 10/50
50/50 [==============================] - 215s 4s/step - loss: 0.6608 - acc
uracy: 0.6560 - val_loss: 0.6116 - val_accuracy: 0.7200
Epoch 11/50
50/50 [==============================] - 847s 17s/step - loss: 0.6781 - ac
curacy: 0.6830 - val_loss: 0.7193 - val_accuracy: 0.5800
Epoch 12/50
50/50 [==============================] - 229s 5s/step - loss: 0.6257 - acc
uracy: 0.6510 - val_loss: 0.5982 - val_accuracy: 0.7000
Epoch 13/50
50/50 [==============================] - 218s 4s/step - loss: 0.6341 - acc
uracy: 0.6720 - val_loss: 0.5521 - val_accuracy: 0.7500
Epoch 14/50
50/50 [==============================] - 216s 4s/step - loss: 0.6287 - acc
uracy: 0.6660 - val_loss: 0.6645 - val_accuracy: 0.6600
Epoch 15/50
50/50 [==============================] - 216s 4s/step - loss: 0.6242 - acc
uracy: 0.6670 - val_loss: 0.6635 - val_accuracy: 0.7200
Epoch 16/50
50/50 [==============================] - 217s 4s/step - loss: 0.6005 - acc
uracy: 0.6750 - val_loss: 0.5201 - val_accuracy: 0.7400
Epoch 17/50
50/50 [==============================] - 219s 4s/step - loss: 0.6390 - acc
uracy: 0.6640 - val_loss: 0.4989 - val_accuracy: 0.7600
Epoch 18/50
50/50 [==============================] - 218s 4s/step - loss: 0.6342 - acc
uracy: 0.6840 - val_loss: 0.5466 - val_accuracy: 0.6800
Epoch 19/50
50/50 [==============================] - 217s 4s/step - loss: 0.6042 - acc
uracy: 0.7020 - val_loss: 0.6448 - val_accuracy: 0.6800
Epoch 20/50
50/50 [==============================] - 217s 4s/step - loss: 0.6288 - acc
uracy: 0.6750 - val_loss: 0.5446 - val_accuracy: 0.7900
Epoch 21/50
```

```
50/50 [==============================] - 218s 4s/step - loss: 0.6137 - acc
uracy: 0.6990 - val_loss: 0.6850 - val_accuracy: 0.7000
Epoch 22/50
50/50 [==============================] - 239s 5s/step - loss: 0.5894 - acc
uracy: 0.7030 - val_loss: 0.5231 - val_accuracy: 0.7200
Epoch 23/50
50/50 [==============================] - 229s 5s/step - loss: 0.5951 - acc
uracy: 0.6900 - val_loss: 0.5737 - val_accuracy: 0.6400
Epoch 24/50
50/50 [==============================] - 218s 4s/step - loss: 0.6110 - acc
uracy: 0.6880 - val_loss: 0.6281 - val_accuracy: 0.7300
Epoch 25/50
50/50 [==============================] - 215s 4s/step - loss: 0.5795 - acc
uracy: 0.6910 - val_loss: 0.4895 - val_accuracy: 0.6900
Epoch 26/50
50/50 [==============================] - 216s 4s/step - loss: 0.6180 - acc
uracy: 0.6860 - val_loss: 0.5593 - val_accuracy: 0.7200
Epoch 27/50
50/50 [==============================] - 217s 4s/step - loss: 0.5755 - acc
uracy: 0.7220 - val_loss: 0.6234 - val_accuracy: 0.7700
Epoch 28/50
50/50 [==============================] - 218s 4s/step - loss: 0.5967 - acc
uracy: 0.7240 - val_loss: 0.6101 - val_accuracy: 0.7700
Epoch 29/50
50/50 [==============================] - 217s 4s/step - loss: 0.6000 - acc
uracy: 0.7020 - val_loss: 0.5319 - val_accuracy: 0.7400
Epoch 30/50
50/50 [==============================] - 217s 4s/step - loss: 0.5657 - acc
uracy: 0.7140 - val_loss: 0.3651 - val_accuracy: 0.7900
Epoch 31/50
50/50 [==============================] - 218s 4s/step - loss: 0.6017 - acc
uracy: 0.7010 - val_loss: 0.4159 - val_accuracy: 0.7300
Epoch 32/50
50/50 [==============================] - 217s 4s/step - loss: 0.5774 - acc
uracy: 0.7160 - val_loss: 0.5290 - val_accuracy: 0.7100
Epoch 33/50
50/50 [==============================] - 217s 4s/step - loss: 0.5462 - acc
uracy: 0.7240 - val_loss: 0.5857 - val_accuracy: 0.6500
Epoch 34/50
50/50 [==============================] - 217s 4s/step - loss: 0.5602 - acc
uracy: 0.7380 - val_loss: 0.4789 - val_accuracy: 0.6900
Epoch 35/50
50/50 [==============================] - 217s 4s/step - loss: 0.5714 - acc
uracy: 0.7250 - val_loss: 0.5693 - val_accuracy: 0.7000
Epoch 36/50
50/50 [==============================] - 216s 4s/step - loss: 0.5841 - acc
uracy: 0.7080 - val_loss: 0.4507 - val_accuracy: 0.7500
Epoch 37/50
50/50 [==============================] - 217s 4s/step - loss: 0.5548 - acc
uracy: 0.7330 - val_loss: 0.4504 - val_accuracy: 0.7500
Epoch 38/50
50/50 [==============================] - 217s 4s/step - loss: 0.5929 - acc
uracy: 0.7290 - val_loss: 0.5524 - val_accuracy: 0.6600
Epoch 39/50
50/50 [==============================] - 220s 4s/step - loss: 0.5556 - acc
uracy: 0.7410 - val_loss: 0.5314 - val_accuracy: 0.7300
Epoch 40/50
50/50 [==============================] - 219s 4s/step - loss: 0.5583 - acc
uracy: 0.7230 - val_loss: 1.1708 - val_accuracy: 0.6500
Epoch 41/50
50/50 [==============================] - 218s 4s/step - loss: 0.5923 - acc
```

```
uracy: 0.7090 - val_loss: 0.5305 - val_accuracy: 0.7400
Epoch 42/50
50/50 [==============================] - 217s 4s/step - loss: 0.5668 - acc
uracy: 0.7350 - val_loss: 0.3616 - val_accuracy: 0.8000
Epoch 43/50
50/50 [==============================] - 217s 4s/step - loss: 0.5449 - acc
uracy: 0.7220 - val_loss: 0.5648 - val_accuracy: 0.7800
Epoch 44/50
50/50 [==============================] - 217s 4s/step - loss: 0.5615 - acc
uracy: 0.7280 - val_loss: 0.3880 - val_accuracy: 0.7800
Epoch 45/50
50/50 [==============================] - 217s 4s/step - loss: 0.5316 - acc
uracy: 0.7380 - val_loss: 0.4999 - val_accuracy: 0.7200
Epoch 46/50
50/50 [==============================] - 219s 4s/step - loss: 0.5387 - acc
uracy: 0.7650 - val_loss: 0.5359 - val_accuracy: 0.7500
Epoch 47/50
50/50 [==============================] - 217s 4s/step - loss: 0.5357 - acc
uracy: 0.7240 - val_loss: 0.5853 - val_accuracy: 0.6400
Epoch 48/50
50/50 [==============================] - 216s 4s/step - loss: 0.6072 - acc
uracy: 0.7170 - val_loss: 0.4887 - val_accuracy: 0.7300
Epoch 49/50
50/50 [==============================] - 217s 4s/step - loss: 0.5746 - acc
uracy: 0.7010 - val_loss: 0.6177 - val_accuracy: 0.7200
Epoch 50/50
50/50 [==============================] - 215s 4s/step - loss: 0.5862 - acc
uracy: 0.6900 - val_loss: 0.7180 - val_accuracy: 0.8200
```

In [12]:

```
import cv2
vidcap = cv2.VideoCapture('dogscats.mp4')
def getFrame(sec):
    vidcap.set(cv2.CAP_PROP_POS_MSEC,sec*1000)
    hasFrames,image = vidcap.read()
    if hasFrames:
        cv2.imwrite("image"+str(count)+".jpg", image)     # save frame as JPG file
    return hasFrames
sec = 0
frameRate = 0.2 #//it will capture image in each 0.2 second
count=1
success = getFrame(sec)
while success:
    count = count + 1
    sec = sec + frameRate
    sec = round(sec, 2)
    success = getFrame(sec)
```

In [13]:

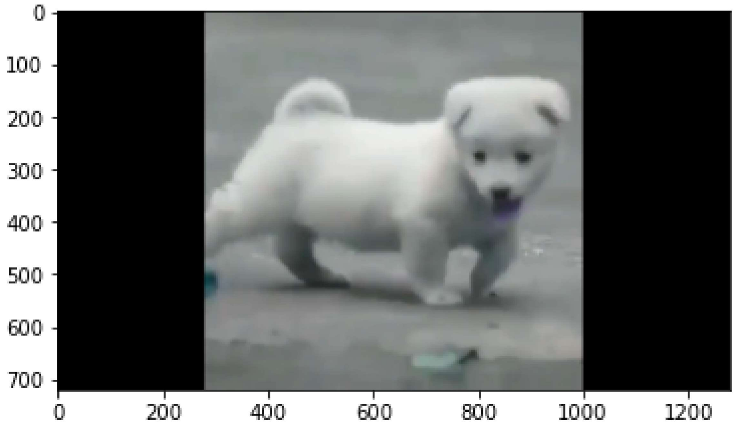
```
img_pred =image.load_img('vid1/image102.jpg',target_size= (500,500))
img_pred = image.img_to_array(img_pred)
img_pred = np.expand_dims(img_pred,axis=0)
```

In [14]:

```
rslt =model.predict(img_pred)
print(rslt)
if rslt[0][0]==1:
    prediction ='dog'
else:
    prediction ='cat'

print(prediction)
import matplotlib.pyplot as plt
img = cv2.imread('vid1/image102.jpg')
plt.imshow(img)
plt.show()
```

```
[[1.]]
dog
```



In [15]:

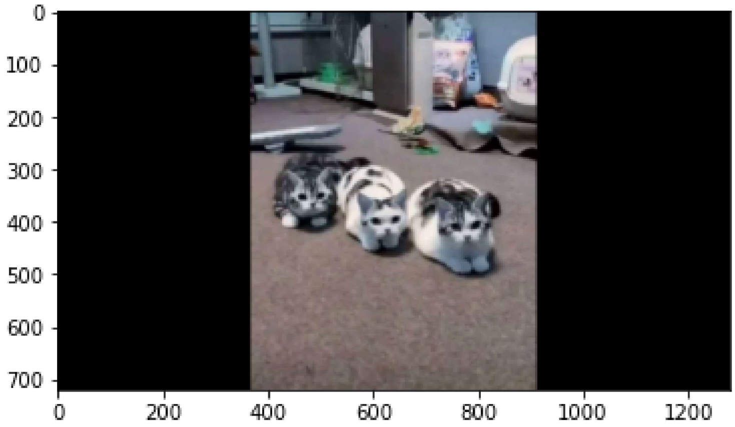
```
img_pred =image.load_img('vid1/image276.jpg',target_size= (500,500))
img_pred = image.img_to_array(img_pred)
img_pred = np.expand_dims(img_pred,axis=0)
```

In [16]:

```
rslt =model.predict(img_pred)
print(rslt)
if rslt[0][0]==1:
    prediction ='dog'
else:
    prediction ='cat'

print(prediction)
img = cv2.imread('vid1/image276.jpg')
plt.imshow(img)
plt.show()
```

```
[[2.6062375e-17]]
cat
```



In [10]:

```
img_pred =image.load_img('test1/Test1/dogs/2.jpg',target_size= (500,500))
img_pred = image.img_to_array(img_pred)
img_pred = np.expand_dims(img_pred,axis=0)
```

In [11]:

```
rslt =model.predict(img_pred)
print(rslt)
if rslt[0][0]==1:
    prediction ='dog'
else:
    prediction ='cat'

print(prediction)
```

```
[[1.]]
dog
```