In [141]:

```
import cv2
import numpy as np
import os
import matplotlib.pyplot as plt
import random
from sklearn .metrics import confusion_matrix,accuracy_score
```

In [142]:

```
train_path="Dog and Cat .png"
class_names=os.listdir(train_path)
```

In [143]:

```
print(class_names)
```

```
['Cat', 'Dog']
```

In [144]:

```
image_paths=[]
image_classes=[]
```

In [145]:

```
def img_list(path):
    return (os.path.join(path,f) for f in os.listdir(path))
```

In [146]:

```
for training_name in class_names:
    dir_=os.path.join(train_path,training_name)
    class_path=img_list(dir_)
    image_paths+=class_path
```

In [147]:

```
len(image_paths)
```

Out[147]:

```
100
```

In [148]:

```
image_classes_0=[0]*(len(image_paths)//2)
```

In [149]:

```
image_classes_1=[1]*(len(image_paths)//2)
```

In [150]:

```
image_classes=image_classes_0+image_classes_1
```

In [151]:

```
D=[]
```

In [152]:

```
for i in range(len(image_paths)):
    D.append((image_paths[i],image_classes[i]))
```

In [409]:

```
dataset = D
random.shuffle(dataset)
train = dataset[:30]
test = dataset[10:]

image_paths, y_train = zip(*train)
image_paths_test, y_test = zip(*test)
```

In [410]:

```
des_list=[]
```

In [411]:

```
orb=cv2.ORB_create()
```

In [412]:

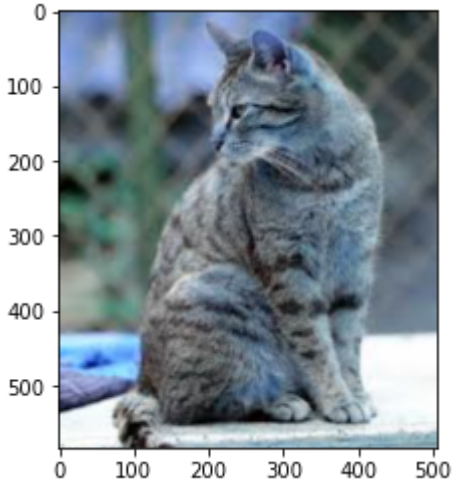
```
im=cv2.imread(image_paths[1])
```

In [413]:

```
plt.imshow(im)
```

Out[413]:

```
<matplotlib.image.AxesImage at 0x1934b63c250>
```

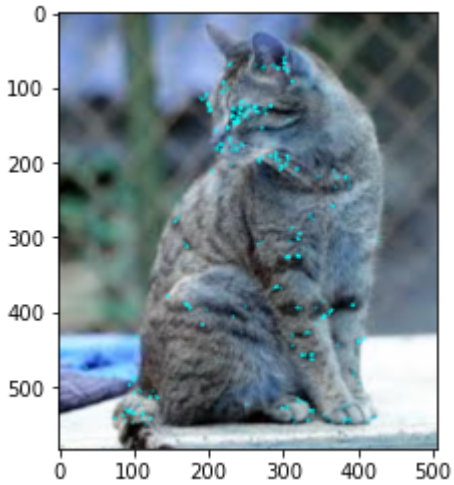


In [414]:

```
def draw_keypoints(vis, keypoints, color = (0, 255, 255)):
    for kp in keypoints:
            x, y = kp.pt
            plt.imshow(cv2.circle(vis, (int(x), int(y)), 2, color))
```

In [415]:

```
kp = orb.detect(im,None)
kp, des = orb.compute(im, kp)
img=draw_keypoints(im,kp)
```



In [416]:

```
for image_pat in image_paths:
    im=cv2.imread(image_pat)
    kp=orb.detect(im,None)
    keypoints,descriptor= orb.compute(im, kp)
    des_list.append((image_pat,descriptor))
```

In [417]:

```
descriptors=des_list[0][1]
for image_path,descriptor in des_list[1:]:
    descriptors=np.vstack((descriptors,descriptor))
```

In [418]:

```
descriptors.shape
```

Out[418]:

```
(13524, 32)
```

In [419]:

```
descriptors_float=descriptors.astype(float)
```

In [420]:

```
from scipy.cluster.vq import kmeans,vq
```

In [421]:

```
k=200
voc,variance=kmeans(descriptors_float,k,1)
```

In [422]:

```
im_features=np.zeros((len(image_paths),k),"float32")
for i in range(len(image_paths)):
    words,distance=vq(des_list[i][1],voc)
    for w in words:
        im_features[i][w]+=1
```

In [423]:

```
from sklearn.preprocessing import StandardScaler
stdslr=StandardScaler().fit(im_features)
im_features=stdslr.transform(im_features)
```

In [424]:

```
from sklearn.svm import LinearSVC
clf=LinearSVC(max_iter=100000)
clf.fit(im_features,np.array(y_train))
```

Out[424]:

```
LinearSVC(max_iter=100000)
```

In [425]:

```
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=100000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)
```

Out[425]:

```
LinearSVC(max_iter=100000)
```

In [426]:

```
des_list_test=[]
```

In [427]:

```
for image_pat in image_paths_test:
    image=cv2.imread(image_pat)
    kp=orb.detect(image,None)
    keypoints_test,descriptor_test= orb.compute(image, kp)
    des_list_test.append((image_pat,descriptor_test))
```

In [428]:

```
len(image_paths_test)
```

Out[428]:

```
90
```

In [429]:

```
from scipy.cluster.vq import vq
test_features=np.zeros((len(image_paths_test),k),"float32")
for i in range(len(image_paths_test)):
    words,distance=vq(des_list_test[i][1],voc)
    for w in words:
        test_features[i][w]+=1
```

In [430]:

```
test_features
```

Out[430]:

```
array([[1., 1., 3., ..., 0., 0., 6.],
       [0., 0., 0., ..., 0., 1., 0.],
       [0., 2., 1., ..., 5., 1., 4.],
       ...,
       [0., 6., 2., ..., 2., 4., 1.],
       [2., 2., 3., ..., 0., 3., 1.],
       [1., 0., 0., ..., 1., 2., 4.]], dtype=float32)
```

In [431]:

```
test_features=stdslr.transform(test_features)
```

In [432]:

```
true_classes=[]
for i in y_test:
    if i==1:
        true_classes.append("Cat")
    else:
        true_classes.append("Dog")
```

In [433]:

```
print(true_classes)
```

```
['Cat', 'Dog', 'Cat', 'Dog', 'Cat', 'Cat', 'Dog', 'Dog', 'Dog', 'Dog', 'Do
g', 'Cat', 'Dog', 'Dog', 'Dog', 'Dog', 'Cat', 'Cat', 'Cat', 'Cat', 'Dog', 'C
at', 'Cat', 'Cat', 'Cat', 'Dog', 'Dog', 'Dog', 'Cat', 'Dog', 'Dog', 'Dog',
'Cat', 'Dog', 'Dog', 'Cat', 'Dog', 'Cat', 'Cat', 'Dog', 'Cat', 'Dog', 'Cat',
'Dog', 'Dog', 'Cat', 'Dog', 'Cat', 'Dog', 'Cat', 'Dog', 'Cat', 'Cat', 'Dog',
'Cat', 'Dog', 'Cat', 'Cat', 'Dog', 'Dog', 'Cat', 'Dog', 'Cat', 'Cat', 'Cat',
'Cat', 'Cat', 'Dog', 'Cat', 'Dog', 'Dog', 'Dog', 'Cat', 'Dog', 'Cat', 'Dog',
'Dog', 'Dog', 'Cat', 'Dog', 'Dog', 'Cat', 'Cat', 'Dog', 'Cat', 'Cat', 'Cat',
'Dog', 'Cat', 'Dog']
```

In [434]:

```
predict_classes=[]
for i in clf.predict(test_features):
    if i==1:
        predict_classes.append("Cat")
    else:
        predict_classes.append("Dog")
```

In [435]:

```
print(predict_classes)
```

```
['Cat', 'Dog', 'Cat', 'Dog', 'Cat', 'Cat', 'Dog', 'Dog', 'Dog', 'Dog', 'Do
g', 'Cat', 'Dog', 'Dog', 'Dog', 'Dog', 'Cat', 'Cat', 'Cat', 'Cat', 'Cat', 'C
at', 'Dog', 'Dog', 'Dog', 'Cat', 'Dog', 'Cat', 'Cat', 'Dog', 'Cat', 'Dog',
'Cat', 'Dog', 'Dog', 'Cat', 'Dog', 'Dog', 'Cat', 'Dog', 'Dog', 'Cat', 'Dog',
'Dog', 'Dog', 'Cat', 'Cat', 'Cat', 'Dog', 'Cat', 'Cat', 'Dog', 'Dog', 'Dog',
'Cat', 'Dog', 'Cat', 'Cat', 'Cat', 'Dog', 'Cat', 'Dog', 'Cat', 'Dog', 'Dog',
'Dog', 'Cat', 'Cat', 'Cat', 'Dog', 'Dog', 'Cat', 'Cat', 'Dog', 'Cat', 'Dog',
'Dog', 'Cat', 'Dog', 'Cat', 'Cat', 'Dog', 'Dog', 'Dog', 'Dog', 'Cat', 'Dog',
'Cat', 'Cat', 'Dog']
```

In [436]:

```
clf.predict(test_features)
```

Out[436]:

```
array([1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1,
       1, 0])
```

In [437]:

```
accuracy=accuracy_score(true_classes,predict_classes)
print(accuracy)
```

```
0.6666666666666666
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: