In [1]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

In [3]:
```
train_data_generator = ImageDataGenerator(rescale = 1./255,
                                          shear_range = 0.2,
                                          zoom_range = 0.2,
                                          horizontal_flip = True)
```

In [4]:
```
train_set = train_data_generator.flow_from_directory('train',
                                                     target_size = (32, 32),
                                                     batch_size = 16,
                                                     class_mode = 'binary')
```

```
Found 40 images belonging to 2 classes.
```

In [5]:
```
train_set.filenames[0]
```

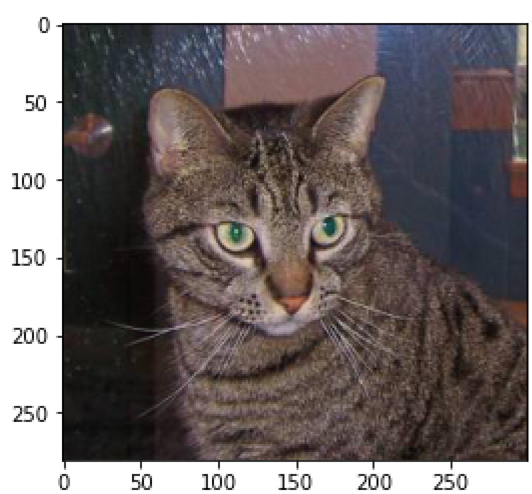
Out[5]: 'cats\\1.jpg'

In [6]:
```
img = tf.keras.preprocessing.image.load_img('train\\cats\\1.jpg')
```

In [7]:
```
import matplotlib.pyplot as plt
```

In [8]:
```
plt.imshow(img)
```

Out[8]: <matplotlib.image.AxesImage at 0x1ce95868c40>



In [9]:
```
train_set.class_indices
```

Out[9]:
```
{'cats': 0, 'dogs': 1}
```

In [10]:
```
test_data_generator = ImageDataGenerator(rescale = 1./255)
```

In [11]:
```
test_set = test_data_generator.flow_from_directory('test',
                                                   target_size = (32,32),
                                                   batch_size = 16,
                                                   class_mode = 'binary')
```

```
Found 20 images belonging to 2 classes.
```

In [57]:
```
 from tensorflow.keras.models import Sequential
```

In [58]:
```
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Dropou
```

In [59]:
```
model = Sequential()
```

Convolutional layer 1 with 32 filters of kernel size[5,5]

In [60]:
```
model.add(Conv2D(filters = 32,
                kernel_size = [5,5],
                activation = 'relu',
                padding = 'valid',
                input_shape = [32,32,3]))
```

Pooling layer 1 with pool size[2,2] and stride 2

In [61]:
```
model.add(MaxPool2D(pool_size = [2,2],
                   strides = 2))
```

Convolutional layer 2 with 64 filters of kernel size[5,5]

In [62]:
```
model.add(Conv2D(filters = 64,
                kernel_size = [5,5],
                activation = 'relu',
                padding = 'valid'))
```

Pooling layer 2 with pool size[2,2] and stride 2

In [63]:
```
model.add(MaxPool2D(pool_size = [2,2],
                   strides = 2))
```

In [64]:
```
model.add(Flatten())
```

Dense layer whose output size is fixed in the hyper parameter: fc_size=32

In [65]:
```
model.add(Dense(units = 32,
               activation = 'relu'))
```

Dropout layer with dropout probability 0.4

In [66]:
```
model.add(Dropout(0.40))
```

Predict the class by doing a sigmoid on the output of the dropout layers.

In [67]:
```
model.add(Dense(units = 1,
               activation = 'sigmoid'))
```

In [68]:
```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 28, 28, 32)        2432

 max_pooling2d_4 (MaxPooling  (None, 14, 14, 32)       0
 2D)

 conv2d_5 (Conv2D)           (None, 10, 10, 64)        51264

 max_pooling2d_5 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 flatten_2 (Flatten)         (None, 1600)              0

 dense_4 (Dense)             (None, 32)                51232

 dropout_2 (Dropout)         (None, 32)                0

 dense_5 (Dense)             (None, 1)                 33

=================================================================
Total params: 104,961
Trainable params: 104,961
Non-trainable params: 0
_________________________________________________________________
```

For the training step, define the loss function and minimize it

In [69]:
```
model.compile(optimizer = 'adam',
             loss = 'binary_crossentropy',
             metrics = ['accuracy'])
```

In [25]:
```
model.fit(train_set, validation_data = test_set, epochs = 100)
```

```
3/3 [==============================] - 0s 160ms/step - loss: 0.0557 - ac
curacy: 1.0000 - val_loss: 1.0487 - val_accuracy: 0.7000
Epoch 82/100
3/3 [==============================] - 0s 161ms/step - loss: 0.0364 - ac
curacy: 1.0000 - val_loss: 1.0387 - val_accuracy: 0.6500
Epoch 83/100
3/3 [==============================] - 0s 142ms/step - loss: 0.1005 - ac
curacy: 0.9500 - val_loss: 0.9699 - val_accuracy: 0.6500
Epoch 84/100
3/3 [==============================] - 0s 163ms/step - loss: 0.0535 - ac
curacy: 1.0000 - val_loss: 0.9231 - val_accuracy: 0.6500
Epoch 85/100
3/3 [==============================] - 0s 161ms/step - loss: 0.1307 - ac
curacy: 0.9750 - val_loss: 0.9241 - val_accuracy: 0.6000
Epoch 86/100
3/3 [==============================] - 0s 140ms/step - loss: 0.1090 - ac
curacy: 0.9500 - val_loss: 0.8887 - val_accuracy: 0.6000
Epoch 87/100
3/3 [==============================] - 0s 173ms/step - loss: 0.0579 - ac
curacy: 1.0000 - val_loss: 0.8754 - val_accuracy: 0.6500
```

In [48]:
```
model.fit(train_set, validation_data = test_set, epochs = 200)
```

```
Epoch 195/200
3/3 [==============================] - 0s 116ms/step - loss: 0.0544 - ac
curacy: 0.9750 - val_loss: 1.2127 - val_accuracy: 0.7500
Epoch 196/200
3/3 [==============================] - 0s 112ms/step - loss: 0.0332 - ac
curacy: 1.0000 - val_loss: 1.1159 - val_accuracy: 0.7000
Epoch 197/200
3/3 [==============================] - 0s 109ms/step - loss: 0.0254 - ac
curacy: 1.0000 - val_loss: 1.1362 - val_accuracy: 0.6500
Epoch 198/200
3/3 [==============================] - 0s 95ms/step - loss: 0.0081 - acc
uracy: 1.0000 - val_loss: 1.1631 - val_accuracy: 0.6500
Epoch 199/200
3/3 [==============================] - 0s 141ms/step - loss: 0.0309 - ac
curacy: 0.9750 - val_loss: 1.1101 - val_accuracy: 0.7500
Epoch 200/200
3/3 [==============================] - 0s 133ms/step - loss: 0.0077 - ac
curacy: 1.0000 - val_loss: 1.1017 - val_accuracy: 0.7000
```

Out[48]:
```
<keras.callbacks.History at 0x1ce98dbfc40>
```

In [70]:
```
model.fit(train_set, validation_data = test_set, epochs = 300)
```

```
3/3 [==============================] - 0s 119ms/step - loss: 0.0021 - ac
curacy: 1.0000 - val_loss: 1.0800 - val_accuracy: 0.8000
Epoch 281/300
3/3 [==============================] - 0s 86ms/step - loss: 0.0214 - acc
uracy: 1.0000 - val_loss: 1.0913 - val_accuracy: 0.8000
Epoch 282/300
3/3 [==============================] - 0s 99ms/step - loss: 0.0020 - acc
uracy: 1.0000 - val_loss: 1.0980 - val_accuracy: 0.8000
Epoch 283/300
3/3 [==============================] - 0s 77ms/step - loss: 0.0044 - acc
uracy: 1.0000 - val_loss: 1.0947 - val_accuracy: 0.8000
Epoch 284/300
3/3 [==============================] - 0s 78ms/step - loss: 0.0187 - acc
uracy: 0.9750 - val_loss: 1.0835 - val_accuracy: 0.8000
Epoch 285/300
3/3 [==============================] - 0s 80ms/step - loss: 0.0099 - acc
uracy: 1.0000 - val_loss: 1.0744 - val_accuracy: 0.8000
Epoch 286/300
3/3 [==============================] - 0s 85ms/step - loss: 0.0166 - acc
uracv: 1.0000 - val loss: 1.0358 - val accuracv: 0.8500
```

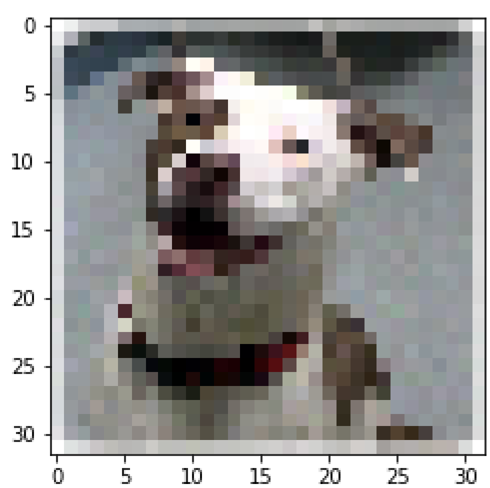
In [71]:
```
import numpy as np
```

In [72]:
```
from tensorflow.keras.preprocessing import image
```

In [73]:
```
test_img = image.load_img('test/dogs/101.jpg',target_size = (32,32,3))
```

In [74]:
```
plt.imshow(test_img)
```

Out[74]: <matplotlib.image.AxesImage at 0x1ce9c2c7790>



In [75]:
```
test_img = image.img_to_array(test_img)
test_img = np.expand_dims(test_img, axis=0)
test_img = test_img/255
```

In [76]:
```
result = model.predict(test_img)
```

In [77]:
```
result
```

Out[77]:
```
array([[1.]], dtype=float32)
```

In [78]:
```
if np.round(result[0][0]) == 1:
    prediction = 'dog'
else:
    prediction = 'cat'
print(prediction)
```

```
dog
```