CAP 6619 Deep Learning

2024 Summer

Homework 3 [18 Pts, Due: June 10 2024. Late Penalty: -2/day]

[If two homework submissions are found to be similar to each other, both submissions will receive 0 grade]

[Homework solutions must be submitted through Canvas. No email submission is accepted. If you have multiple files, please include all files as one zip file, and submit zip file online (only zip, pdf, or word files are allowed). You can always update your submissions. Only the latest version will be graded.]

Question 1 [2 pts]: In Figure 1, the upper panel shows a convolutional filter being applied to an image with 6x6 pixels to generate output, and the lower panel shows a fully connected dense network to process the same image.

- For the dense network, how many neurons are needed to produce the results which are the same as the convolutional filter?

  The number of neurons depends on the dimension of the convolutional layer. It is equal to to N * M, where N and M are the length and width of the convolutional layer. Here, N=6 and M = 6, so, the number of neurons is 36.

- How many parameters are needed for the dense network?

  There are two parameters in the dense network, weights and biases.

- How many weight values does the convolutional filter have?

  The convolutional filter has nine weight values.

- Explain why the dense network is designed to find global patterns, whereas the convolutional filter is designed to find local patterns?

  The convolutional filters are designed to find the important features like color, shape, and edge within an image. They are immune to distortions like shifting and scaling. Dense neural networks fail to do so. For this reason, CNNs are used to find the most important features i.e. local patterns within the image. These patterns are then passed through a neural network to perform classification.

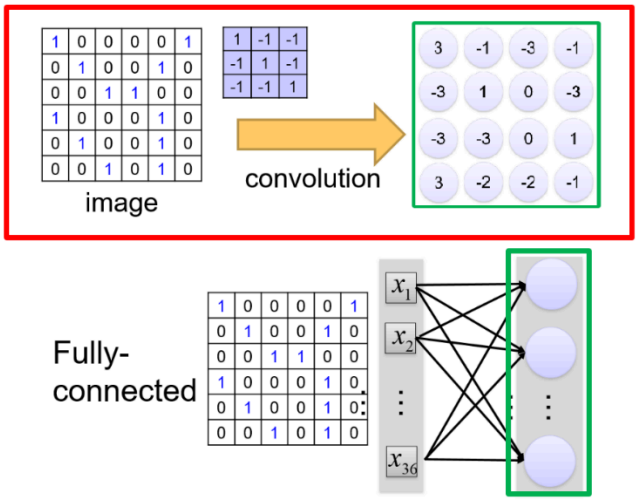


Figure 1

Question 2 [2 pts]: Figure 2 shows a 6x6 image, and a 3x3 convolutional filter, and the weight symbols specified in the matrix (w1, w2, …),

- please show convolutional filter output after the filter being applied to the red colored circle (using weight values w1,…, w9. w0 denotes bias) [0.5 pt]

  w1 - w2 - w3 - w4 + w5 - w6 -w7 - w8 + w9

- what is the purpose of the weight values (w1, w2, …) of the 3x3 filter? [0.5 pt]

  The filter is the heart of CNN, they are responsible for the extraction of the most important features. The weights within these filters assign the importance of each pixel within an image and assists in detection of features like edge, shape, and color of the image. The weights are updated via backward propagation to get the best outcome for the task at hand.

- When moving the filter across the image (using stride 1), what is the percentage of input shared been two consecutive points (horizontally or vertically) [0.5 pt]

  Two successive points (horizontally or vertically) share 50% of the input percentage.

- What is the size of the feature map after applying the filter to the whole image (using stride 1)? What are the factors determining the feature map size [0.5 pt]

  The factors are image size, filter size, stride, and padding. The size of the feature map is is going to be 4 * 4.

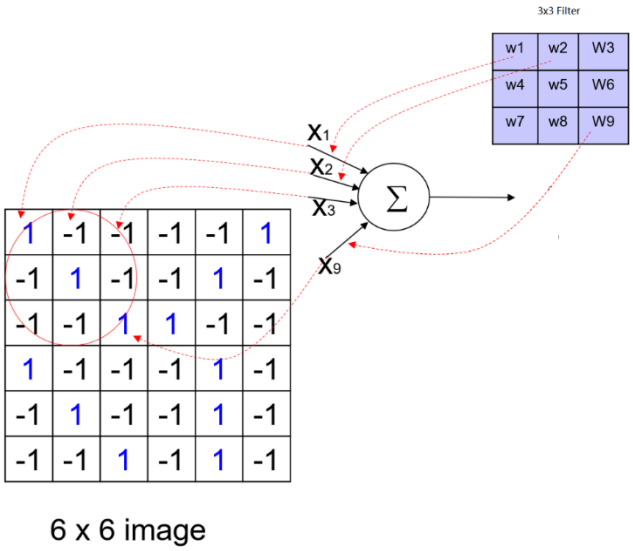


Figure 2

Question 3 [**2 pts**]: Table 1 shows a 3x6 synthetic image, and Table 2 shows a filter.

- Please apply the filter in Table 2 to the image in Table 1 (using convolutional filter and stride=1), and show the feature map output. [0.5 pt]

3

4 6 4 0 4 5

4 -5 5 1 6 3 -1 -1 -1

1 0 1 1 0 1 1 1 1

Pass-1

$$\begin{bmatrix} 1 & 6 & 4 \\ 4 & -5 & 5 \end{bmatrix} * \begin{bmatrix} -1 & -1 & -1 \\ 1 & 1 & 1 \end{bmatrix}$$

$-4-6-4+4$
$= -5+5$
$= -10$

Pass-2

$$\begin{bmatrix} 6 & 4 & 0 \\ -5 & 5 & 1 \end{bmatrix} * \begin{bmatrix} -1 & -1 & -1 \\ 1 & 1 & 1 \end{bmatrix}$$

$-6-4-5+5+1$
$= -9$

Pass-3

$$\begin{bmatrix} 4 & 0 & 4 \\ 5 & 1 & 6 \end{bmatrix} * \begin{bmatrix} -1 & -1 & -1 \\ 1 & 1 & 1 \end{bmatrix}$$

$-4-4+5+1+6$
$= 4$

Pass - 4

$$\begin{bmatrix} 0 & 4 & 5 \\ 1 & 6 & 3 \end{bmatrix} * \begin{bmatrix} -1 & -1 & -1 \\ 1 & 1 & 1 \end{bmatrix} = -4-5+1+6 = +3 = 1$$

Pass-5

$$\begin{bmatrix} 4 & -5 & 5 \\ 1 & 0 & 1 \end{bmatrix} * \begin{bmatrix} -1 & -1 & -1 \\ 1 & 1 & 1 \end{bmatrix} = -4+5-5+1+1 = -2$$

Pass - 6

$$\begin{bmatrix} -5 & 5 & 1 \\ 0 & 1 & 1 \end{bmatrix} * \begin{bmatrix} -1 & -1 & -1 \\ 1 & 1 & 1 \end{bmatrix} = 5-5-1+1+1 = 1$$

Pass - 7

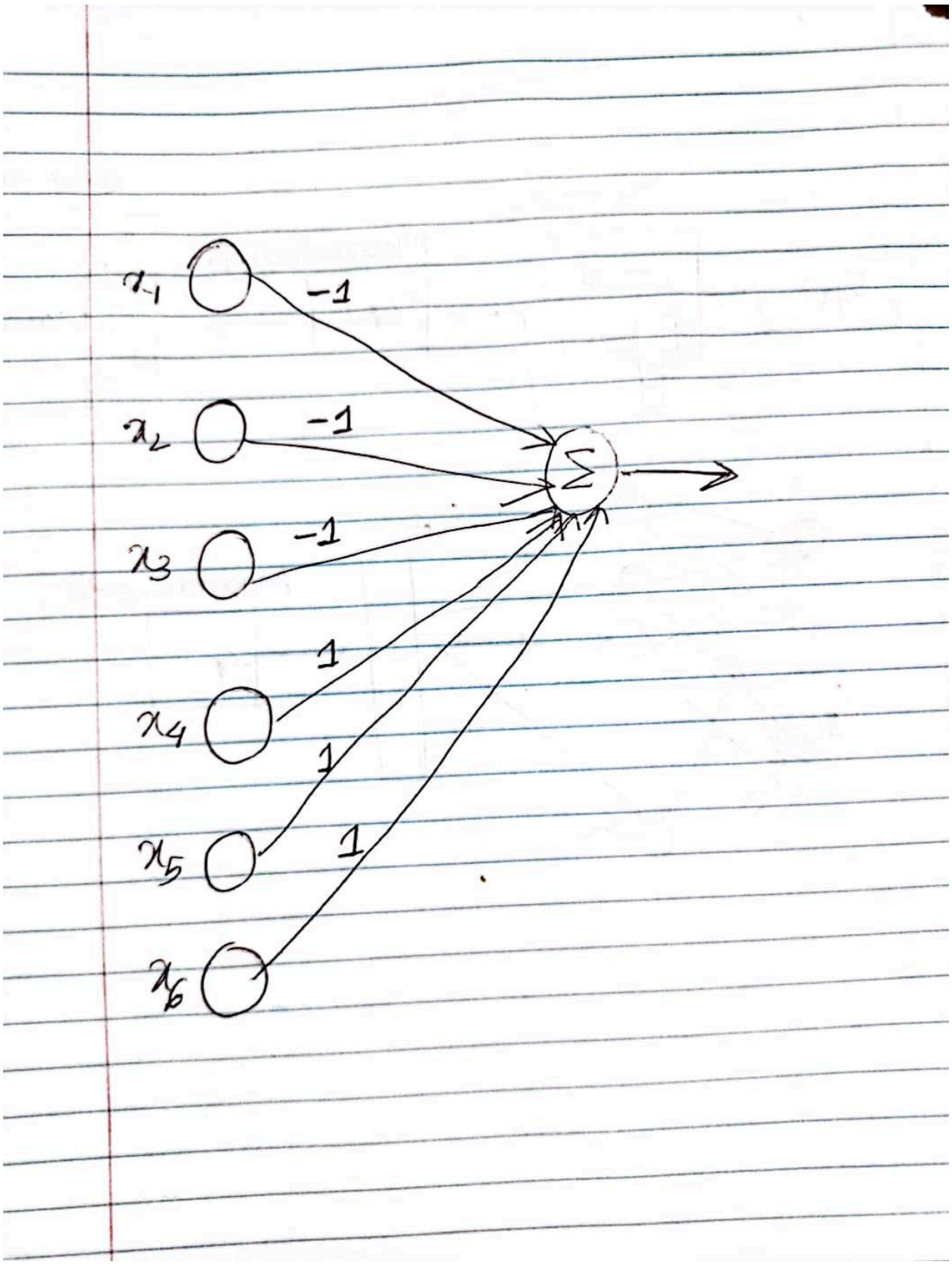
$$\begin{bmatrix} 5 & 1 & 6 \\ 1 & 1 & 0 \end{bmatrix} * \begin{bmatrix} -1 & -1 & -1 \\ 1 & 1 & 1 \end{bmatrix} = -5-1-6+1 = +1 = -10$$

$$\begin{bmatrix} 1 & 6 & 3 \\ 1 & 0 & 1 \end{bmatrix} * \begin{bmatrix} -1 & -1 & -1 \\ 1 & 1 & 1 \end{bmatrix}$$

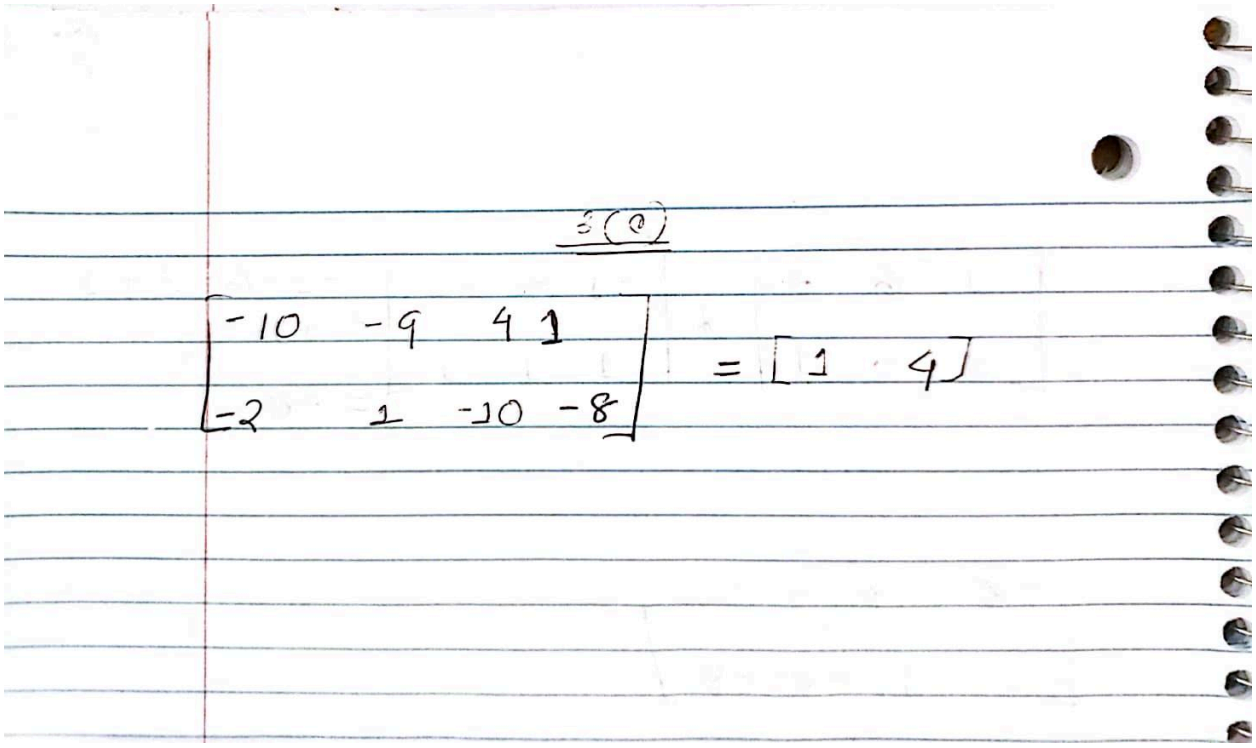
$-1-6-3+1$
$= +1$
$= -8$

$$\begin{bmatrix} -10 & -9 & 4 & 1 \\ -2 & 1 & -10 & -8 \end{bmatrix}$$

- Show perceptron architecture to implement the filter (ignore the bias) [0.5 pt]


x_1
x_2
x_3
x_4
x_5
x_6
-1
-1
-1
1
1
1
Σ

- Apply 2x2 max pooling to the above result, and report the resulting image. [0.5 pt]



- Explain the role of the filter in Table 2, and how does the filter achieve the goal through the convolution process. [0.5 pt]

  CNNs extract the most important features within an image and then pass it through a neural network to perform the task at hand. A filter is passed onto the image to extract the feature map from within the image. Although the feature map is irrelevant to the human eye, they are capable of encoding the specific features within the image. After sliding across the input data, a filter multiplies each element of the input by itself with the portion of the input it is currently on, adding up all of the results into a single output pixel. The weights within the filter are updated via backpropagation to get best outcome.

Table 1

4	6	4	0	4	5
4	-5	5	1	6	3
1	0	1	1	0	1

-1	-1	-1
1	1	1

Table 2

Question 4 [2 pts]: Figure 3 shows the structure of the LeNet5 convolutional neural network.

- What is the name of the C1 layer? What is the size of the filters of the C1 layer? How many weight values C1 layer has? [0.5 pt]

  C1 is the first convolutional layer known as the input layer. The C1 layer has 6 filters each having a size of 28*28. There are 3920 weight values in the C1 layer.

- What is the size of the filters in the C3 layer? How many weight values C3 layer has? [0.5 pt]

  The C3 layer has 16 filters each having a size of 10*10. There are 1600 weight values in C3 layer.

- How many weight values C5 and F6 layers each has, respectively [0.5 pt]

  The number of weights in C5 is 16 * 5 * 5 * 120 = 48000 and that of F6 is 84 * 48000 = 4032000.

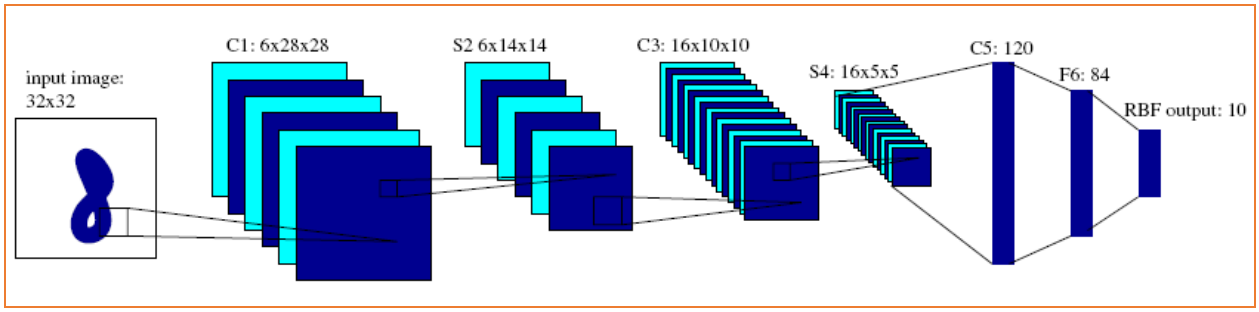
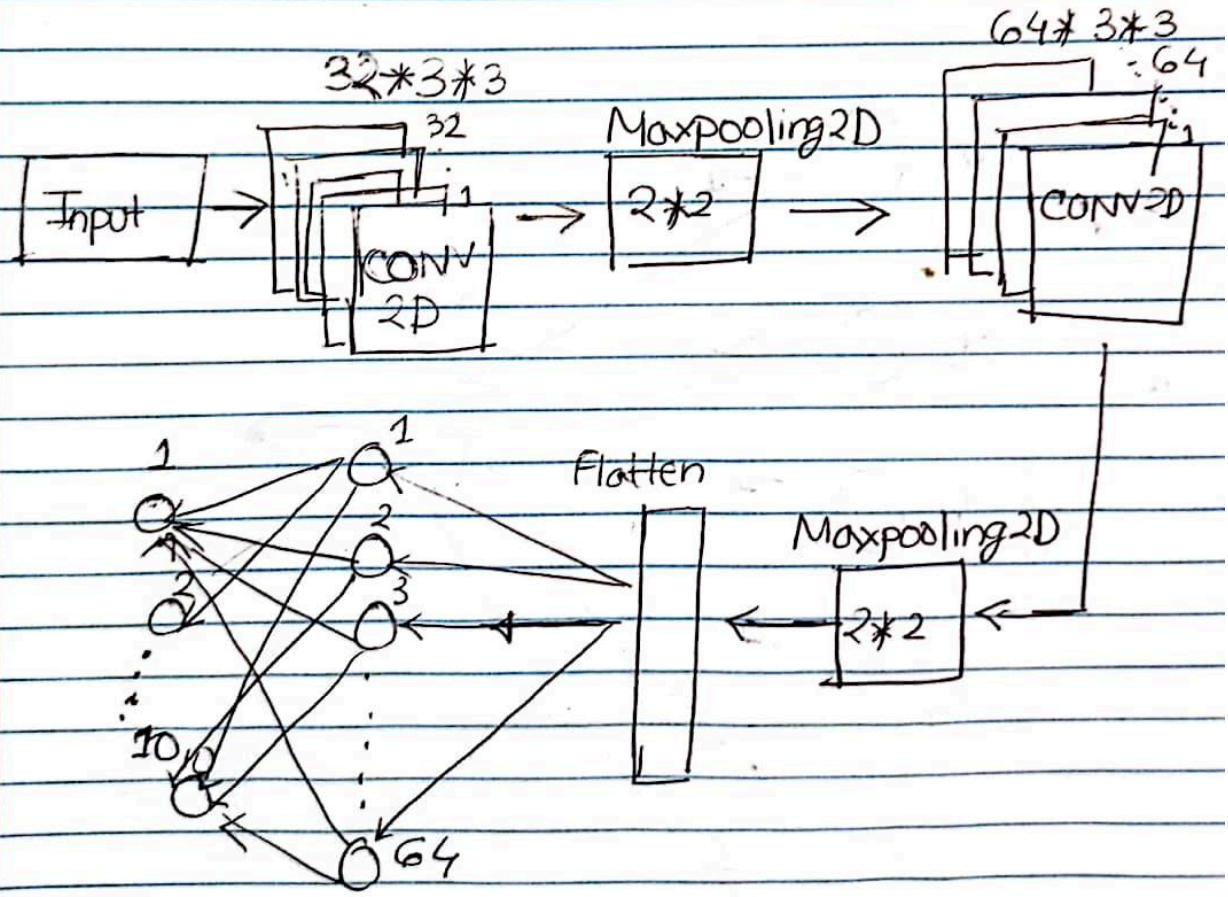


Figure 3

Question 5 [2 pts]: The following Keras codes show a deep learning network. Please draw the network structure from input to the output to explicitly show network components and parameters:

1. Please draw diagram of the designed network [0.5 pt]

3(b)


32*3*3
32
Input
CONV 2D
Maxpooling2D
2*2
64* 3*3
64
CONV2D
Flatten
Maxpooling2D
2*2
1
2
3
10
64

2. Show input, output sizes, and number of weight values of each convolution layer [0.5 pt].

Layer	Input size	Output size	Weights
1st convolutional layer	28, 28,3	26, 26, 32	896
2nd convolutional layer	13, 13, 32	11, 11, 64	18496

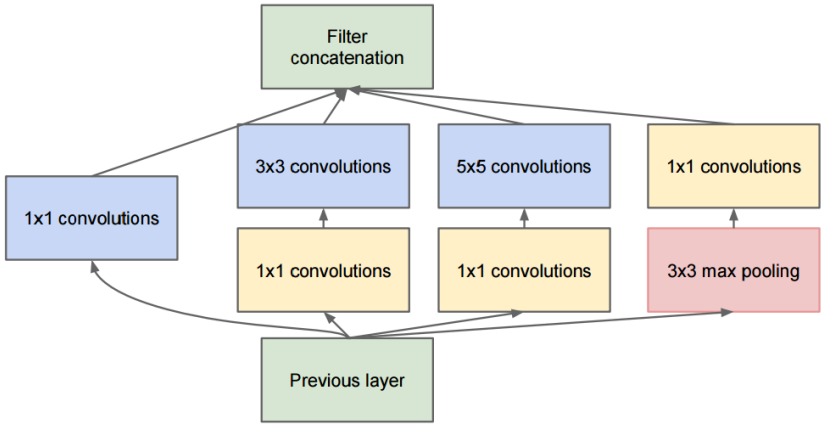
3. Show input and output size of each pooling layer [0.5 pt].

Layer	Input size	Output size
1st Maxpooling Layer	26, 26, 32	13,13, 32
2nd Maxpooling Layer	11, 11, 64	5, 5, 64

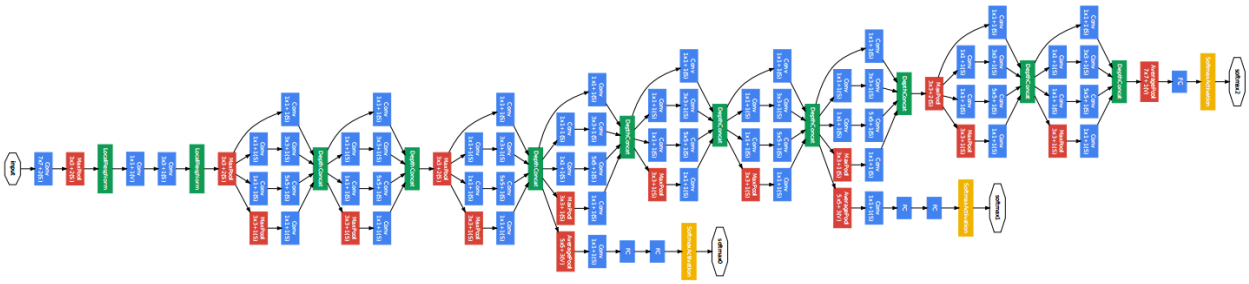
4. Show input, output sizes, and number of weight values of each dense layer [0.5 pt].

Layer	Input size	Output size	Weights
1st Densel layer	1600	64	102464
2nd Dense layer	64	1	65

```
network = Sequential()
model.add(Conv2D(32,(3,3),activation="relu",input_shape=c(28,28,3))
model.add(MaxPooling2D((2,2))
model.add(Conv2D(64,(3,3),activation="relu"))
model.add(MaxPooling2D((2,2))
model.add(Flatten())
model.add(Dense(64,activation='relu'))
model.add(Dense(10,activation='softmax'))
```



(a)



(b)

Figure 4: Inception module (https://arxiv.org/pdf/1409.4842.pdf), and GoogLeNet structure

Question 6 [4 pts]: Figure 4(a) shows structure of an inception module, and Figure 4(b) shows GoogLeNet structure.

- How many layers GoogLeNet have? What are the main differences between GoogleNet vs. VGG-16 in the context of convolutional neural network? [0.5 pt]

  GoogleNet has 22 layers.
  GoogleNet uses inception modules while VGG-16 uses convolutional layers. Due to inception modules, the number of parameters within GoogleNet is lower than in VGG-16.

- Explain motivation of the inception module [0.25 pt].

  A typical convolutional network is very large consisting of a large number of convolution layers. This increases the complexity of the network making it more prone to overfitting. Also, the network takes a long time to train. To solve these problems, inception module was created.

- Explain purpose and functionality of the 1x1 convolutions of the inception module [0.25 pt].

  It is used for dimension reduction.

- The input image sizes to 224x224x3 (three color challenges). The first convolution layer of the network has 64 filters (each of which is 7x7 in size), the second volution layer has 64 1x1 filters, followed by 192 convolution filters (each with size 3x3). What are the number of tunable parameters at second and third convolution layer, respectively? [0.5 pt]

  For 2nd convolutional layer, (3 * 3 * 192 * 64) = 37,184
  For 2nd convolutional layer, (3 * 3 * 64 * 192) + 192 (bias terms) = 110,784

- Figure 4(a) shows inception module layer 3(a), the number of feature maps from previously layer is 28x28x192 (i.e., 192 feature maps), calculate number tunable parameters for each path of the inception module [0.5x4=2pts]

1*1	(1 * 1 * 192 * 64) + 64 = 12352
3*3	(3 * 3 * 192 * 96) + 96 = 165984
5*5	(5 * 5 * 192 * 16) + 16 = 76816
Pooling layer	(1 * 1 * 192) = 192

- Explain purpose and functionality of the 1x1 convolutions of the inception module [0.25 pt].

  It is used for dimension reduction.

- The network has three softmax (and output) at different stage of the network. What are the purposes of these outputs [0.5 pt].

  In training, the vanishing gradient problem is solved with the help of the first softmax. The second and third softmax contribute to classification, with the third making the final decision.

Questions 7 is a programming task

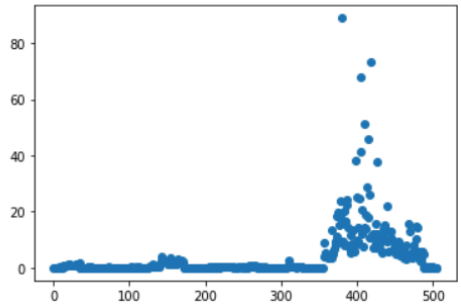
For all programming tasks, solutions must be submitted as notebook (html or pdf) files for grading (your submission must include scrips/code and the running results of the script).

If you are not familiar with Python programming (and want to use Python for the coding tasks), please check Python Plotting notebook and Python Simple Analysis notebook posted in the Canvas, before working the coding tasks.

For each subtask, please use task description (requirement) as comments, and report your coding and results in following format:

```
# Report all samples with respect to the Crim index on a plot (the x-axis shows the index of the sample, and the y-axis
# shows the crim index of the sample).

y = boston['Crim']
x=np.arange(y.shape[0]) # generate x index

plt.scatter(x, y, marker='o')
```

```
: <matplotlib.collections.PathCollection at 0x1c0d3ceebc8>
```



Question 7 [4 pts]: Please follow "**CNN Image Classification [html, Notebook]** " to create a convolutional neural network (CNN) image classifiers and validate its performances.

- Download the cat vs. dog images (556MB zip file) from the following URL https://www.cse.fau.edu/~xqzhu/courses/cap6619/dataset/train.zip
  You can also download images from the Kaggle site (Dogs vs. Cats | Kaggle)
- Unzip the downloaded (zip) file. There are 25,000 (dog and cat images, 12,500 for each category) in the "train" folder.
- Create a training dataset with least 2000 images (1000 for each category), a validation set with at least 1000 images (500 for each category), and a test set with at least 1000 images (500 for each category) [1 pt]
- Create a CNN classifier with at least three convolution layers, two pooling layers, and two dense layers. Train the network on the training set, and report the performance of the classifier on the test set. [1 pt]
- For the same network structure created above, please add a dropout layer (with a selected dropout rate) and add Batch Normalization. Train the network on the training set, and report the performance of the classifier on the test set. [1 pt]
- For the above network (including a mixed use of dropout layer and Batch Normalization), please use image_data_generator with rotation, width_shift, height_shift, shear, and zoom to create

image distortions for training. Train the network on the training set, and report the performance of the classifier on the test set. [1 pt]

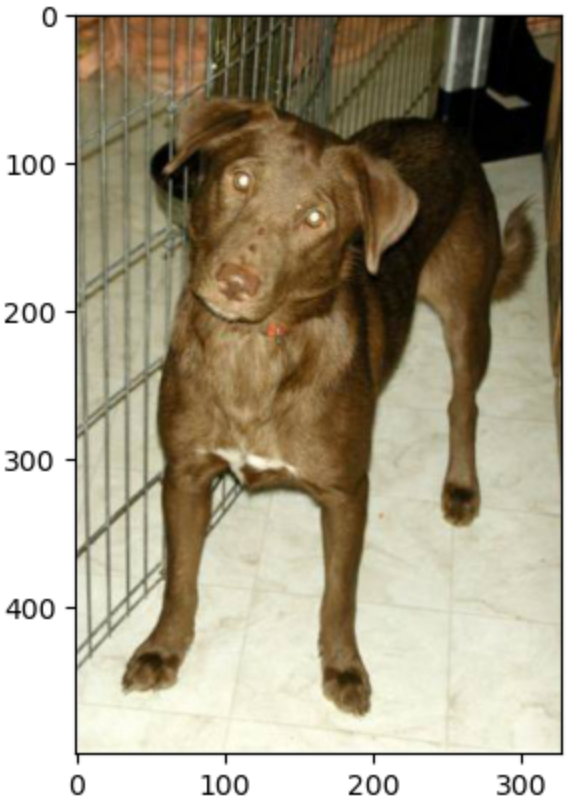
In [36]:
```
from matplotlib import pyplot
from matplotlib import pyplot as plt
from matplotlib.image import imread

import tensorflow as tf

from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Dropout, BatchNormalization

from keras.layers import Flatten
from keras.optimizers import SGD
from keras.preprocessing.image import ImageDataGenerator

from os import listdir
from numpy import asarray
from numpy import save
from matplotlib.image import imread

import shutil
from sklearn.model_selection import train_test_split
```

Question 7 [4 pts]: Please follow “CNN Image Classification [html, Notebook] ” to create a convolutional neural network (CNN) image classifiers and validate its performances. Download the cat vs. dog images (556MB zip file) from the following URL https://www.cse.fau.edu/~xqzhu/courses/cap6619/dataset/train.zip You can also download images from the Kaggle site (https://www.kaggle.com/c/dogs-vs-cats/data) Unzip the downloaded (zip) file. There are 25,000 (dog and cat images, 12,500 for each category) in the “train” folder.

Create a training dataset with least 2000 images (1000 for each category), a validation set with at least 1000 images (500 for each category), and a test set with at least 1000 images (500 for each category) [1 pt]

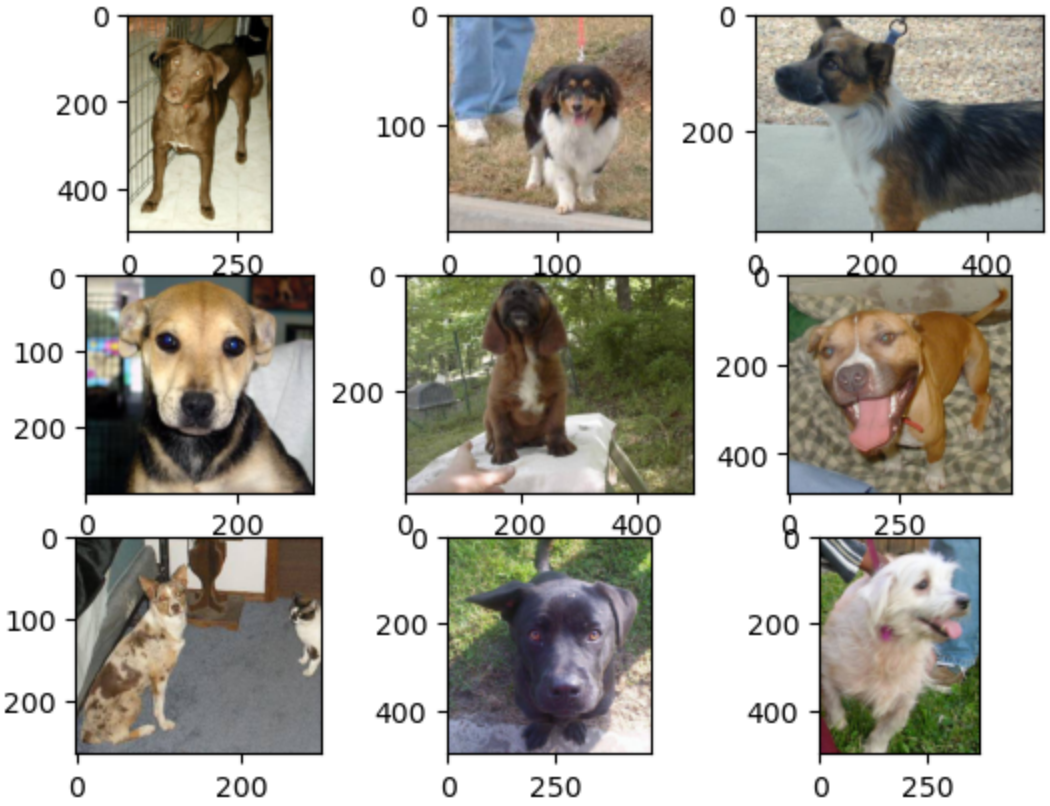
In [37]:
```
folder='train_2/'
# show one image
filename = folder + 'dog.' + '1' + '.jpg'
# load image pixels
image = imread(filename)
# plot raw pixel data
plt.imshow(image)
```

Out[37]:
```
<matplotlib.image.AxesImage at 0x76ea88159fa0>
```

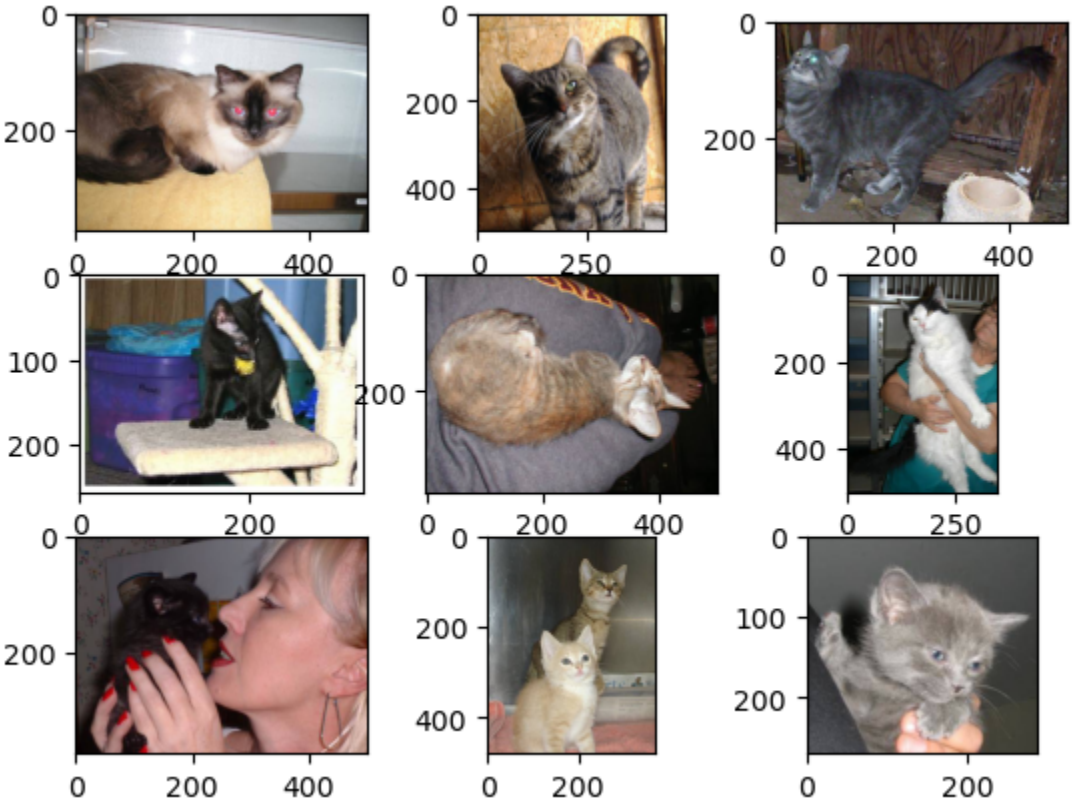


In [38]:
```
def displayImages(foldername,dogorcat,startID):
    # plot first few images
    for i in range(9):
        #define subplot 3x3
        plt.subplot(330 + 1 + i)
        # define filename
        filename = foldername + dogorcat +'.' + str(i+startID) + '.jpg'
        # load image pixels
        image = imread(filename)
        # plot raw pixel data
        plt.imshow(image)
        # show the figure
plt.show()
```

In [39]:
```
displayImages(folder,"dog",1)
```



In [40]:
```
displayImages(folder,"cat",20)
```



In [41]:
```
import os
```

In [42]:
```
base_dir = '/home/manishakarim/Class/Deep Learning/train_2/'
train_dir = os.path.join(base_dir, 'train')
test_dir = os.path.join(base_dir, 'test')
validation_dir = os.path.join(base_dir, 'validation')
os.makedirs(train_dir, exist_ok=True)
os.makedirs(validation_dir, exist_ok=True)
```

In [43]:
```
train_cats_dir = os.path.join(train_dir, 'cats')
train_dogs_dir = os.path.join(train_dir, 'dogs')
validation_cats_dir = os.path.join(validation_dir, 'cats')
validation_dogs_dir = os.path.join(validation_dir, 'dogs')
os.makedirs(train_cats_dir, exist_ok=True)
os.makedirs(train_dogs_dir, exist_ok=True)
os.makedirs(validation_cats_dir, exist_ok=True)
os.makedirs(validation_dogs_dir, exist_ok=True)
```

In [44]:
```
test_cats_dir = os.path.join(test_dir, 'cats')
test_dogs_dir = os.path.join(test_dir, 'dogs')
os.makedirs(test_cats_dir, exist_ok=True)
os.makedirs(test_dogs_dir, exist_ok=True)
```

In [45]:
```
new_train_dir = '/home/manishakarim/Class/Deep Learning/train_2/'
all_cats = [os.path.join(new_train_dir, f) for f in os.listdir(new_train_dir
all_dogs = [os.path.join(new_train_dir, f) for f in os.listdir(new_train_dir
```

In [46]:
```
train_cats, val_cats = train_test_split(all_cats, test_size=0.92, random_sta
train_dogs, val_dogs = train_test_split(all_dogs, test_size=0.92, random_sta

val_cats,test_cat = train_test_split(val_cats, test_size=0.9565, random_stat
val_dogs, test_dog = train_test_split(val_dogs, test_size=0.9565, random_sta
```

In [47]:
```
import random
test_cats = []
test_cats.extend(random.sample(test_cat, 500))

test_dogs = []
test_dogs.extend(random.sample(test_dog, 500))
```

In [48]:
```
for file in train_cats:
    shutil.copy(file, train_cats_dir)
for file in val_cats:
    shutil.copy(file, validation_cats_dir)
for file in test_cats:
    shutil.copy(file, test_cats_dir)

for file in train_dogs:
    shutil.copy(file, train_dogs_dir)
for file in val_dogs:
    shutil.copy(file, validation_dogs_dir)
for file in test_dogs:
    shutil.copy(file, test_dogs_dir)
```

Create a CNN classifier with at least three convolution layers, two pooling layers, and two dense layers. Train the network on the training set, and report the performance of the

classifier on the test set. [1 pt]

In [49]:
```
train_datagen = ImageDataGenerator(rescale=1./255)
validation_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=20,
    class_mode='binary'
)

validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
    target_size=(150, 150),
    batch_size=20,
    class_mode='binary'
)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(150, 150),
    batch_size=20,
    class_mode='binary'
)
```

```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
Found 1956 images belonging to 2 classes.
```

In [50]:
```
model = Sequential()

model.add(Conv2D(32, (3, 3),activation="relu",input_shape=(150,150,3)))
model.add(MaxPooling2D(pool_size=(3, 3)))

model.add(Conv2D(64, (3, 3),activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (3, 3),activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(100,activation="relu"))
model.add(Dense(50,activation="relu"))
model.add(Dense(1,activation="sigmoid"))

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

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_12 (Conv2D)          (None, 148, 148, 32)      896

 max_pooling2d_12 (MaxPoolin  (None, 49, 49, 32)       0
 g2D)

 conv2d_13 (Conv2D)          (None, 47, 47, 64)        18496

 max_pooling2d_13 (MaxPoolin  (None, 23, 23, 64)       0
 g2D)

 conv2d_14 (Conv2D)          (None, 21, 21, 128)       73856

 max_pooling2d_14 (MaxPoolin  (None, 10, 10, 128)      0
 g2D)

 flatten_4 (Flatten)         (None, 12800)             0

 dense_12 (Dense)            (None, 100)               1280100

 dense_13 (Dense)            (None, 50)                5050

 dense_14 (Dense)            (None, 1)                 51

=================================================================
Total params: 1,378,449
Trainable params: 1,378,449
Non-trainable params: 0
_________________________________________________________________
```

In [51]:
```
history = model.fit( train_generator, batch_size =25, epochs=30, validation_
```

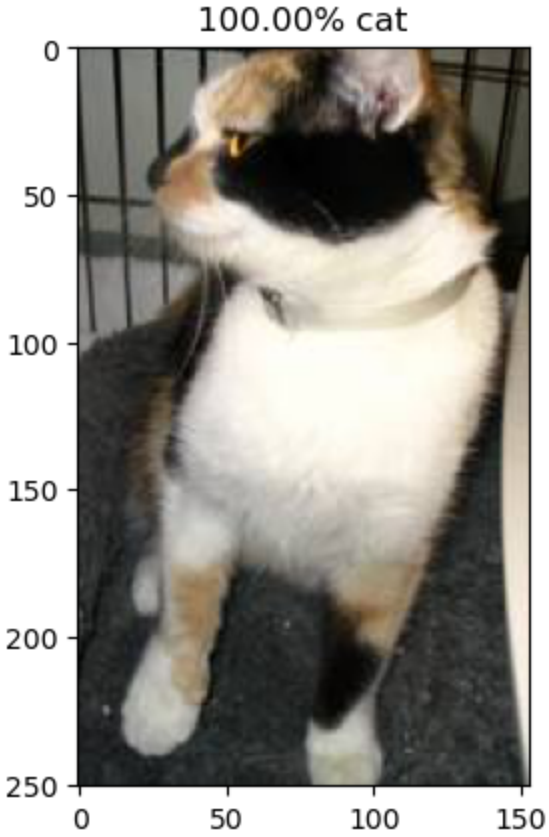
```
Epoch 1/30
100/100 [==============================] - 4s 37ms/step - loss: 0.6998 - acc
uracy: 0.4985 - val_loss: 0.6899 - val_accuracy: 0.5010
Epoch 2/30
100/100 [==============================] - 4s 36ms/step - loss: 0.6866 - acc
uracy: 0.5690 - val_loss: 0.6507 - val_accuracy: 0.5980
Epoch 3/30
100/100 [==============================] - 4s 38ms/step - loss: 0.6422 - acc
uracy: 0.6285 - val_loss: 0.5972 - val_accuracy: 0.6810
Epoch 4/30
100/100 [==============================] - 4s 36ms/step - loss: 0.5840 - acc
uracy: 0.6940 - val_loss: 0.6377 - val_accuracy: 0.6460
Epoch 5/30
100/100 [==============================] - 4s 36ms/step - loss: 0.5399 - acc
uracy: 0.7285 - val_loss: 0.5396 - val_accuracy: 0.7180
Epoch 6/30
100/100 [==============================] - 4s 36ms/step - loss: 0.4936 - acc
uracy: 0.7660 - val_loss: 0.7435 - val_accuracy: 0.6140
Epoch 7/30
100/100 [==============================] - 4s 36ms/step - loss: 0.4357 - acc
uracy: 0.7985 - val_loss: 0.5562 - val_accuracy: 0.7500
Epoch 8/30
100/100 [==============================] - 4s 36ms/step - loss: 0.3872 - acc
uracy: 0.8310 - val_loss: 0.5785 - val_accuracy: 0.7330
Epoch 9/30
100/100 [==============================] - 4s 36ms/step - loss: 0.3198 - acc
uracy: 0.8635 - val_loss: 0.5998 - val_accuracy: 0.7540
Epoch 10/30
100/100 [==============================] - 4s 36ms/step - loss: 0.2435 - acc
uracy: 0.9015 - val_loss: 0.7108 - val_accuracy: 0.7230
Epoch 11/30
100/100 [==============================] - 4s 36ms/step - loss: 0.1963 - acc
uracy: 0.9260 - val_loss: 0.7153 - val_accuracy: 0.7280
Epoch 12/30
100/100 [==============================] - 4s 36ms/step - loss: 0.1379 - acc
uracy: 0.9485 - val_loss: 0.7640 - val_accuracy: 0.7280
Epoch 13/30
100/100 [==============================] - 4s 36ms/step - loss: 0.1016 - acc
uracy: 0.9625 - val_loss: 1.0353 - val_accuracy: 0.7240
Epoch 14/30
100/100 [==============================] - 4s 37ms/step - loss: 0.0815 - acc
uracy: 0.9740 - val_loss: 1.2291 - val_accuracy: 0.7300
Epoch 15/30
100/100 [==============================] - 4s 37ms/step - loss: 0.0588 - acc
uracy: 0.9805 - val_loss: 1.5990 - val_accuracy: 0.7180
Epoch 16/30
100/100 [==============================] - 4s 37ms/step - loss: 0.0652 - acc
uracy: 0.9795 - val_loss: 1.3421 - val_accuracy: 0.7250
Epoch 17/30
100/100 [==============================] - 4s 37ms/step - loss: 0.0637 - acc
uracy: 0.9770 - val_loss: 1.2324 - val_accuracy: 0.7380
Epoch 18/30
100/100 [==============================] - 4s 37ms/step - loss: 0.0261 - acc
uracy: 0.9920 - val_loss: 1.8273 - val_accuracy: 0.7250
Epoch 19/30
100/100 [==============================] - 4s 36ms/step - loss: 0.0474 - acc
```

```
uracy: 0.9865 - val_loss: 1.4817 - val_accuracy: 0.7400
Epoch 20/30
100/100 [==============================] - 4s 37ms/step - loss: 0.0335 - acc
uracy: 0.9900 - val_loss: 1.5264 - val_accuracy: 0.7310
Epoch 21/30
100/100 [==============================] - 4s 36ms/step - loss: 0.0224 - acc
uracy: 0.9930 - val_loss: 1.8792 - val_accuracy: 0.7300
Epoch 22/30
100/100 [==============================] - 4s 36ms/step - loss: 0.0225 - acc
uracy: 0.9925 - val_loss: 1.8152 - val_accuracy: 0.7280
Epoch 23/30
100/100 [==============================] - 4s 37ms/step - loss: 0.0192 - acc
uracy: 0.9930 - val_loss: 2.2776 - val_accuracy: 0.7100
Epoch 24/30
100/100 [==============================] - 4s 36ms/step - loss: 0.0394 - acc
uracy: 0.9880 - val_loss: 1.7679 - val_accuracy: 0.7310
Epoch 25/30
100/100 [==============================] - 4s 37ms/step - loss: 0.0081 - acc
uracy: 0.9970 - val_loss: 2.2687 - val_accuracy: 0.7310
Epoch 26/30
100/100 [==============================] - 4s 36ms/step - loss: 0.0305 - acc
uracy: 0.9905 - val_loss: 2.3614 - val_accuracy: 0.7240
Epoch 27/30
100/100 [==============================] - 4s 36ms/step - loss: 0.0163 - acc
uracy: 0.9950 - val_loss: 2.2392 - val_accuracy: 0.7280
Epoch 28/30
100/100 [==============================] - 4s 36ms/step - loss: 0.0224 - acc
uracy: 0.9955 - val_loss: 2.7775 - val_accuracy: 0.7090
Epoch 29/30
100/100 [==============================] - 4s 37ms/step - loss: 0.0294 - acc
uracy: 0.9930 - val_loss: 2.3840 - val_accuracy: 0.7320
Epoch 30/30
100/100 [==============================] - 4s 36ms/step - loss: 0.0156 - acc
uracy: 0.9950 - val_loss: 2.4839 - val_accuracy: 0.7110
```

In [52]:

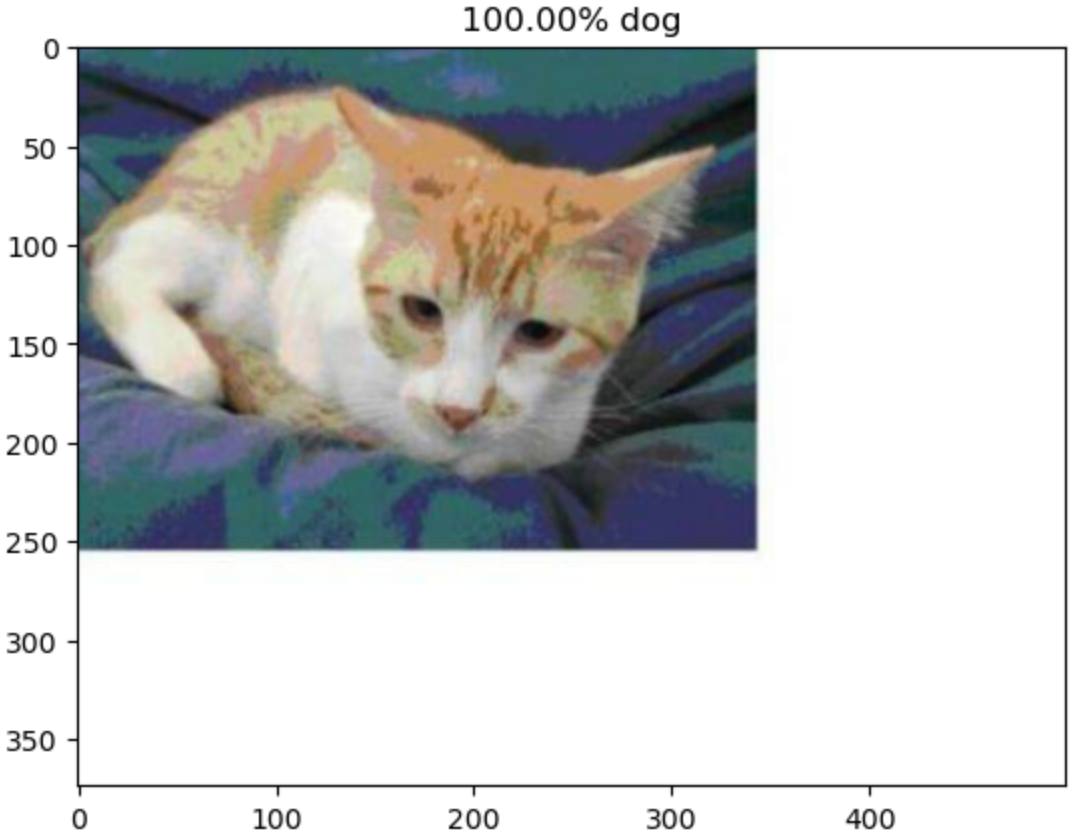
```
outputfile='output.txt'
numtestimages=5
open(outputfile,"w")
probabilities = model.predict(test_generator, numtestimages)
probabilities = probabilities[0:9]

for index, probability in enumerate(probabilities):
    image_path = test_dir + "/" +test_generator.filenames[index]
    img = imread(image_path)
    with open(outputfile,"a") as fh:
        fh.write(str(probability[0]) + " for: " + image_path + "\n")
    pyplot.imshow(img)
    if probability > 0.5:
        pyplot.title("%.2f" % (probability[0]*100) + "% dog")
    else:
        pyplot.title("%.2f" % ((1-probability[0])*100) + "% cat")
    plt.figure(figsize = [3, 3])
    pyplot.show()
```

```
98/98 [==============================] - 3s 26ms/step
```

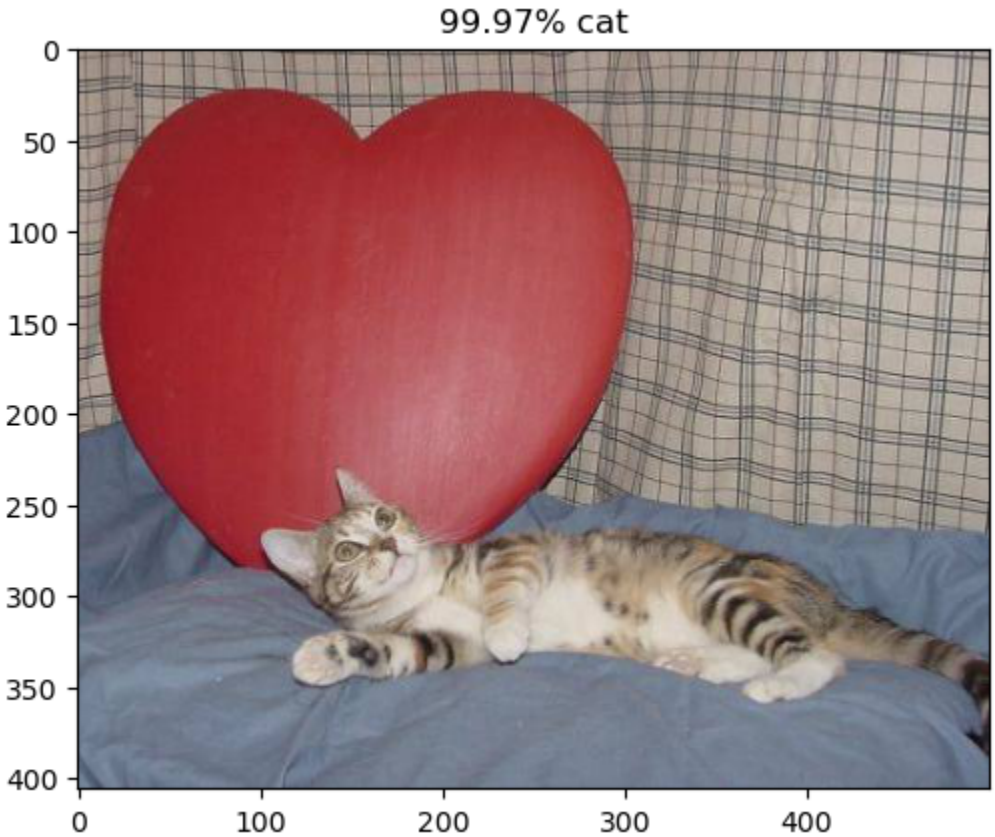



```
<Figure size 300x300 with 0 Axes>
```

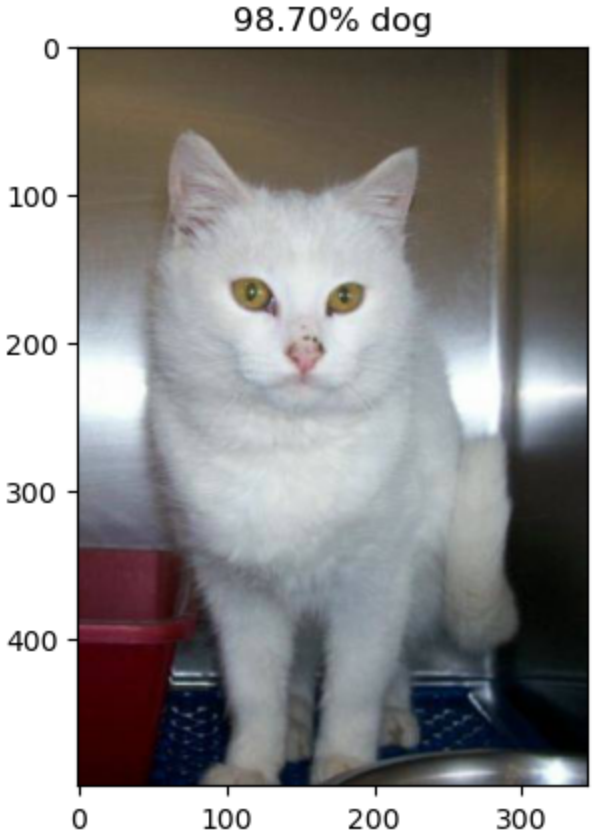

```
<Figure size 300x300 with 0 Axes>
```



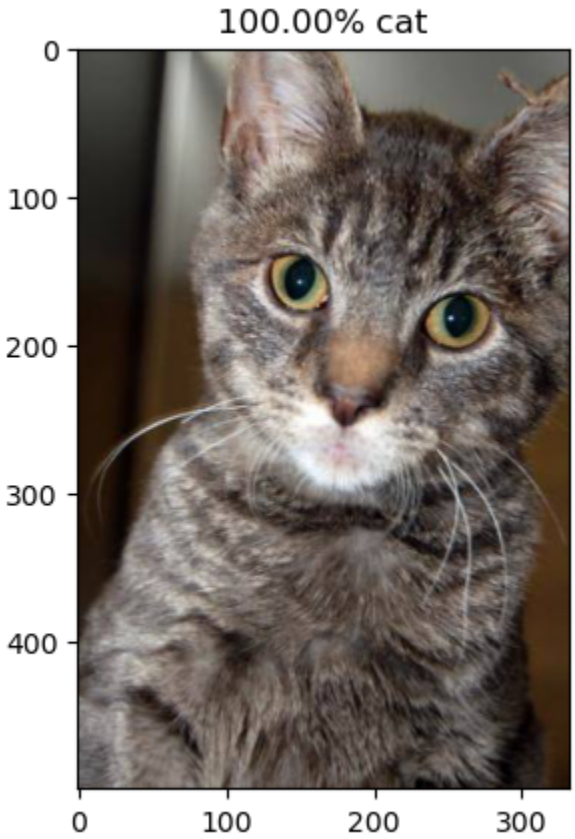
<Figure size 300x300 with 0 Axes>



<Figure size 300x300 with 0 Axes>

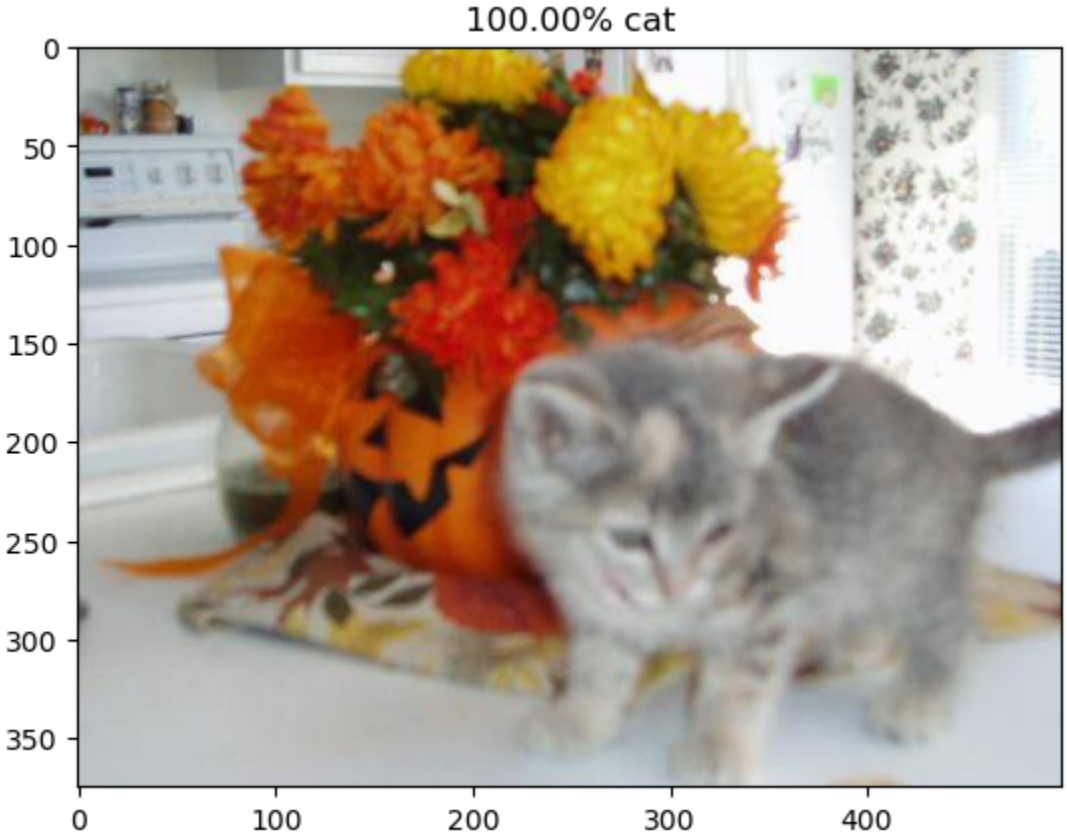


<Figure size 300x300 with 0 Axes>



<Figure size 300x300 with 0 Axes>

<Figure size 300x300 with 0 Axes>



<Figure size 300x300 with 0 Axes>

<Figure size 300x300 with 0 Axes>

For the same network structure created above, please add a dropout layer (with a selected dropout rate) and add Batch Normalization. Train the network on the training set, and report the performance of the classifier on the test set. [1 pt]

In [53]:
```
model = Sequential()

model.add(Conv2D(32, (3, 3),activation="relu",input_shape=(150,150,3)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(3, 3)))

model.add(Conv2D(64, (3, 3),activation="relu"))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (3, 3),activation="relu"))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(100,activation="relu"))
model.add(Dense(50,activation="relu"))
model.add(Dropout(0.3))
model.add(Dense(1,activation="sigmoid"))

model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
```

```
              metrics=["accuracy"])
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_15 (Conv2D)          (None, 148, 148, 32)      896

 batch_normalization_3 (Batc  (None, 148, 148, 32)     128
 hNormalization)

 max_pooling2d_15 (MaxPoolin  (None, 49, 49, 32)       0
 g2D)

 conv2d_16 (Conv2D)          (None, 47, 47, 64)        18496

 batch_normalization_4 (Batc  (None, 47, 47, 64)       256
 hNormalization)

 max_pooling2d_16 (MaxPoolin  (None, 23, 23, 64)       0
 g2D)

 conv2d_17 (Conv2D)          (None, 21, 21, 128)       73856

 batch_normalization_5 (Batc  (None, 21, 21, 128)      512
 hNormalization)

 max_pooling2d_17 (MaxPoolin  (None, 10, 10, 128)      0
 g2D)

 flatten_5 (Flatten)         (None, 12800)             0

 dense_15 (Dense)            (None, 100)               1280100

 dense_16 (Dense)            (None, 50)                5050

 dropout_1 (Dropout)         (None, 50)                0

 dense_17 (Dense)            (None, 1)                 51

=================================================================
Total params: 1,379,345
Trainable params: 1,378,897
Non-trainable params: 448
_________________________________________________________________
```

In [54]:
```
history = model.fit( train_generator, batch_size =25, epochs=30, validation_
```

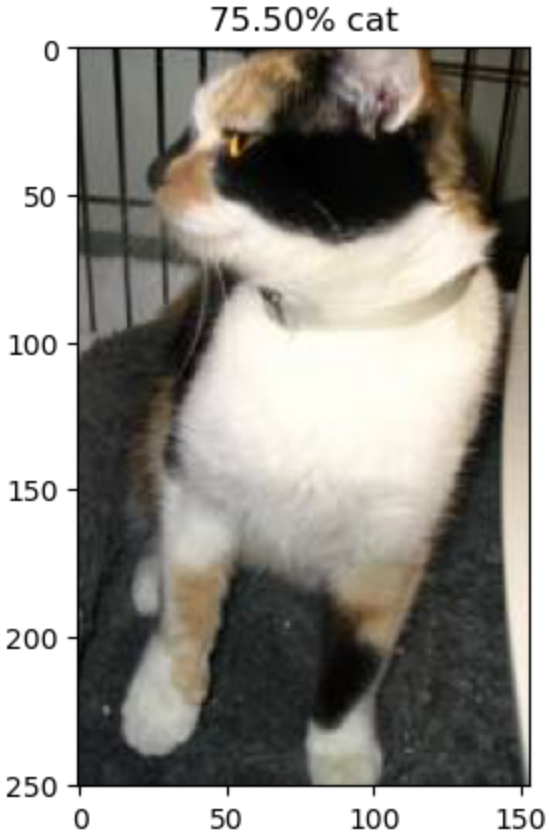
```
Epoch 1/30
100/100 [==============================] - 5s 38ms/step - loss: 1.0414 - acc
uracy: 0.5465 - val_loss: 0.7388 - val_accuracy: 0.5510
Epoch 2/30
100/100 [==============================] - 4s 38ms/step - loss: 0.7284 - acc
uracy: 0.6545 - val_loss: 0.8506 - val_accuracy: 0.5120
Epoch 3/30
100/100 [==============================] - 4s 37ms/step - loss: 0.5995 - acc
uracy: 0.7040 - val_loss: 1.2251 - val_accuracy: 0.5000
Epoch 4/30
100/100 [==============================] - 4s 38ms/step - loss: 0.5420 - acc
uracy: 0.7420 - val_loss: 0.8140 - val_accuracy: 0.5520
Epoch 5/30
100/100 [==============================] - 4s 37ms/step - loss: 0.4652 - acc
uracy: 0.7930 - val_loss: 0.7233 - val_accuracy: 0.6380
Epoch 6/30
100/100 [==============================] - 4s 36ms/step - loss: 0.3763 - acc
uracy: 0.8430 - val_loss: 0.6804 - val_accuracy: 0.7240
Epoch 7/30
100/100 [==============================] - 4s 37ms/step - loss: 0.2997 - acc
uracy: 0.8770 - val_loss: 0.5617 - val_accuracy: 0.7560
Epoch 8/30
100/100 [==============================] - 4s 36ms/step - loss: 0.2245 - acc
uracy: 0.9175 - val_loss: 0.9753 - val_accuracy: 0.7160
Epoch 9/30
100/100 [==============================] - 4s 38ms/step - loss: 0.1689 - acc
uracy: 0.9460 - val_loss: 1.1034 - val_accuracy: 0.6550
Epoch 10/30
100/100 [==============================] - 4s 37ms/step - loss: 0.1205 - acc
uracy: 0.9610 - val_loss: 1.5378 - val_accuracy: 0.7100
Epoch 11/30
100/100 [==============================] - 4s 36ms/step - loss: 0.0926 - acc
uracy: 0.9735 - val_loss: 1.3599 - val_accuracy: 0.7230
Epoch 12/30
100/100 [==============================] - 4s 37ms/step - loss: 0.1160 - acc
uracy: 0.9665 - val_loss: 2.3472 - val_accuracy: 0.6920
Epoch 13/30
100/100 [==============================] - 4s 37ms/step - loss: 0.0873 - acc
uracy: 0.9730 - val_loss: 2.0738 - val_accuracy: 0.7270
Epoch 14/30
100/100 [==============================] - 4s 37ms/step - loss: 0.0519 - acc
uracy: 0.9810 - val_loss: 1.6019 - val_accuracy: 0.7030
Epoch 15/30
100/100 [==============================] - 4s 37ms/step - loss: 0.0562 - acc
uracy: 0.9815 - val_loss: 1.5518 - val_accuracy: 0.7030
Epoch 16/30
100/100 [==============================] - 4s 36ms/step - loss: 0.0634 - acc
uracy: 0.9785 - val_loss: 2.1642 - val_accuracy: 0.7440
Epoch 17/30
100/100 [==============================] - 4s 37ms/step - loss: 0.0446 - acc
uracy: 0.9885 - val_loss: 1.7111 - val_accuracy: 0.7190
Epoch 18/30
100/100 [==============================] - 4s 37ms/step - loss: 0.0587 - acc
uracy: 0.9815 - val_loss: 4.3331 - val_accuracy: 0.6800
Epoch 19/30
100/100 [==============================] - 4s 37ms/step - loss: 0.0332 - acc
```

```
uracy: 0.9885 - val_loss: 2.2786 - val_accuracy: 0.7310
Epoch 20/30
100/100 [==============================] - 4s 37ms/step - loss: 0.0644 - acc
uracy: 0.9850 - val_loss: 2.4906 - val_accuracy: 0.7090
Epoch 21/30
100/100 [==============================] - 4s 37ms/step - loss: 0.0330 - acc
uracy: 0.9895 - val_loss: 1.8049 - val_accuracy: 0.7030
Epoch 22/30
100/100 [==============================] - 4s 37ms/step - loss: 0.0644 - acc
uracy: 0.9840 - val_loss: 3.7905 - val_accuracy: 0.6810
Epoch 23/30
100/100 [==============================] - 4s 37ms/step - loss: 0.0399 - acc
uracy: 0.9885 - val_loss: 3.3974 - val_accuracy: 0.7080
Epoch 24/30
100/100 [==============================] - 4s 38ms/step - loss: 0.0425 - acc
uracy: 0.9875 - val_loss: 3.3487 - val_accuracy: 0.7250
Epoch 25/30
100/100 [==============================] - 4s 36ms/step - loss: 0.0410 - acc
uracy: 0.9905 - val_loss: 2.1171 - val_accuracy: 0.7340
Epoch 26/30
100/100 [==============================] - 4s 37ms/step - loss: 0.0558 - acc
uracy: 0.9845 - val_loss: 2.0491 - val_accuracy: 0.7300
Epoch 27/30
100/100 [==============================] - 4s 36ms/step - loss: 0.0440 - acc
uracy: 0.9885 - val_loss: 2.8395 - val_accuracy: 0.7400
Epoch 28/30
100/100 [==============================] - 4s 37ms/step - loss: 0.0259 - acc
uracy: 0.9915 - val_loss: 4.8777 - val_accuracy: 0.7260
Epoch 29/30
100/100 [==============================] - 4s 36ms/step - loss: 0.0377 - acc
uracy: 0.9890 - val_loss: 3.1552 - val_accuracy: 0.7510
Epoch 30/30
100/100 [==============================] - 4s 37ms/step - loss: 0.0379 - acc
uracy: 0.9905 - val_loss: 3.9866 - val_accuracy: 0.6900
```

In [56]:

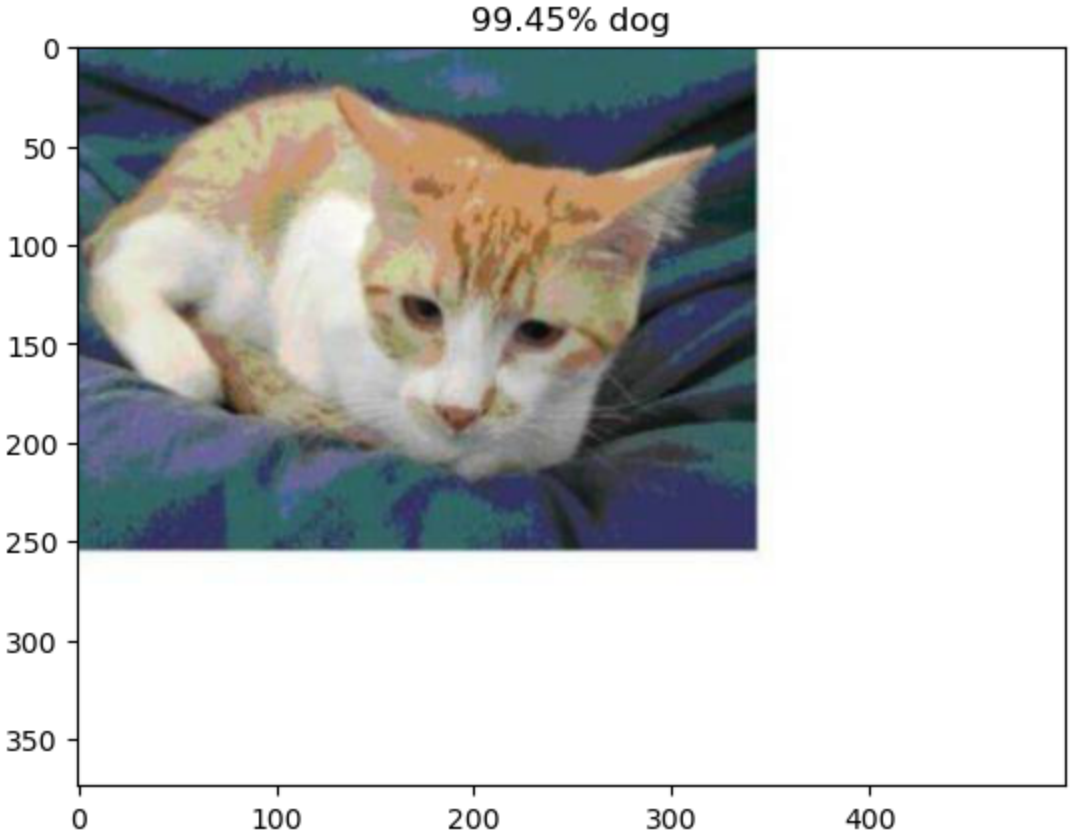
```
outputfile='output.txt'
numtestimages=5
open(outputfile,"w")
probabilities = model.predict(test_generator, numtestimages)
probabilities = probabilities[0:9]
for index, probability in enumerate(probabilities):
    image_path = test_dir + "/" +test_generator.filenames[index]
    img = imread(image_path)
    with open(outputfile,"a") as fh:
        fh.write(str(probability[0]) + " for: " + image_path + "\n")
    pyplot.imshow(img)
    if probability > 0.5:
        pyplot.title("%.2f" % (probability[0]*100) + "% dog")
    else:
        pyplot.title("%.2f" % ((1-probability[0])*100) + "% cat")
    plt.figure(figsize = [3, 3])
    pyplot.show()
```

```
98/98 [==============================] - 2s 24ms/step
```

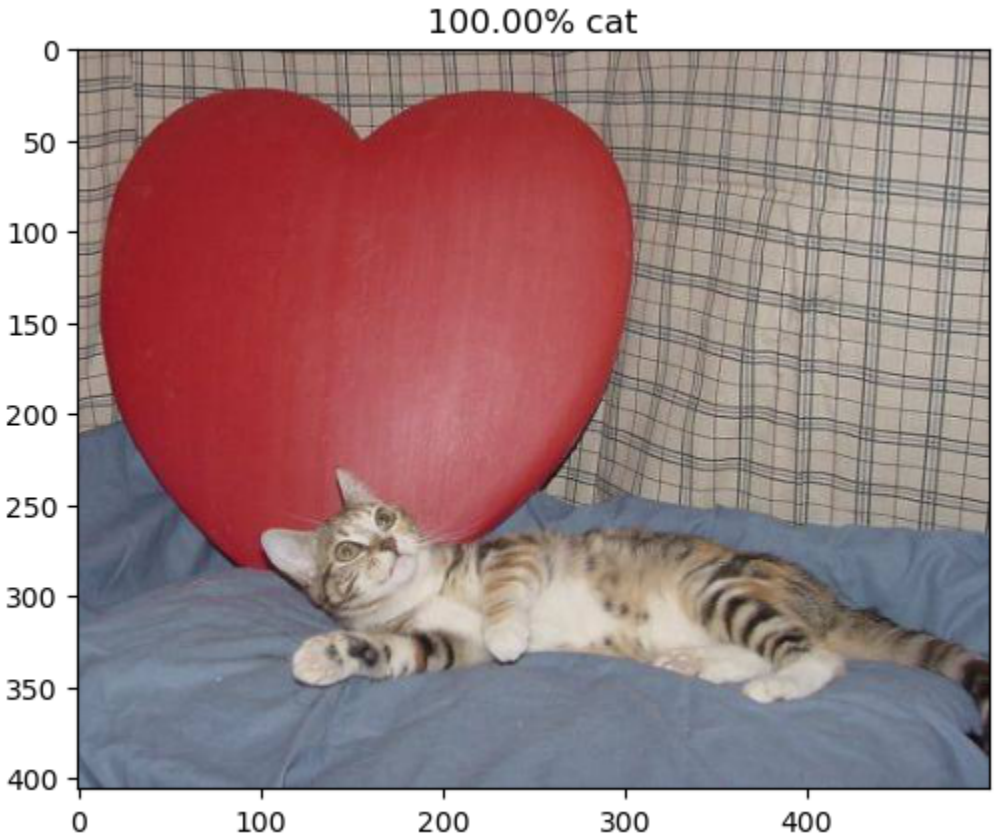



```
<Figure size 300x300 with 0 Axes>
```

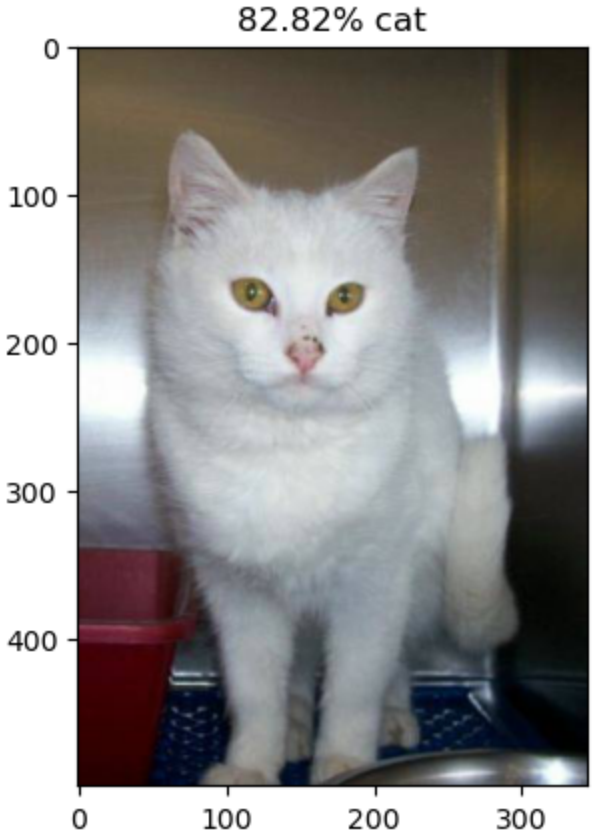

```
<Figure size 300x300 with 0 Axes>
```



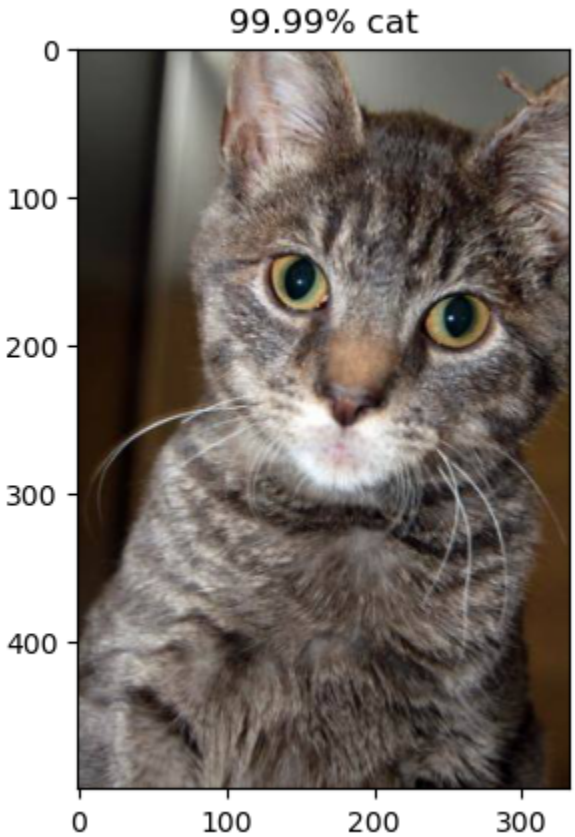
<Figure size 300x300 with 0 Axes>



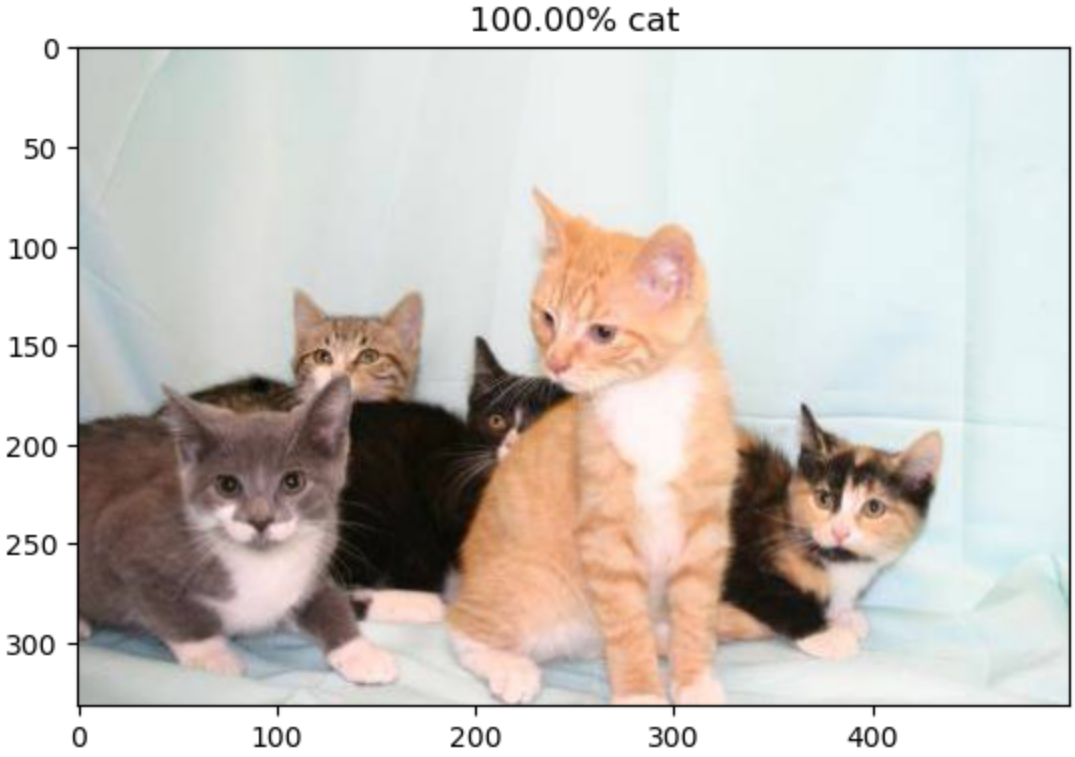
<Figure size 300x300 with 0 Axes>



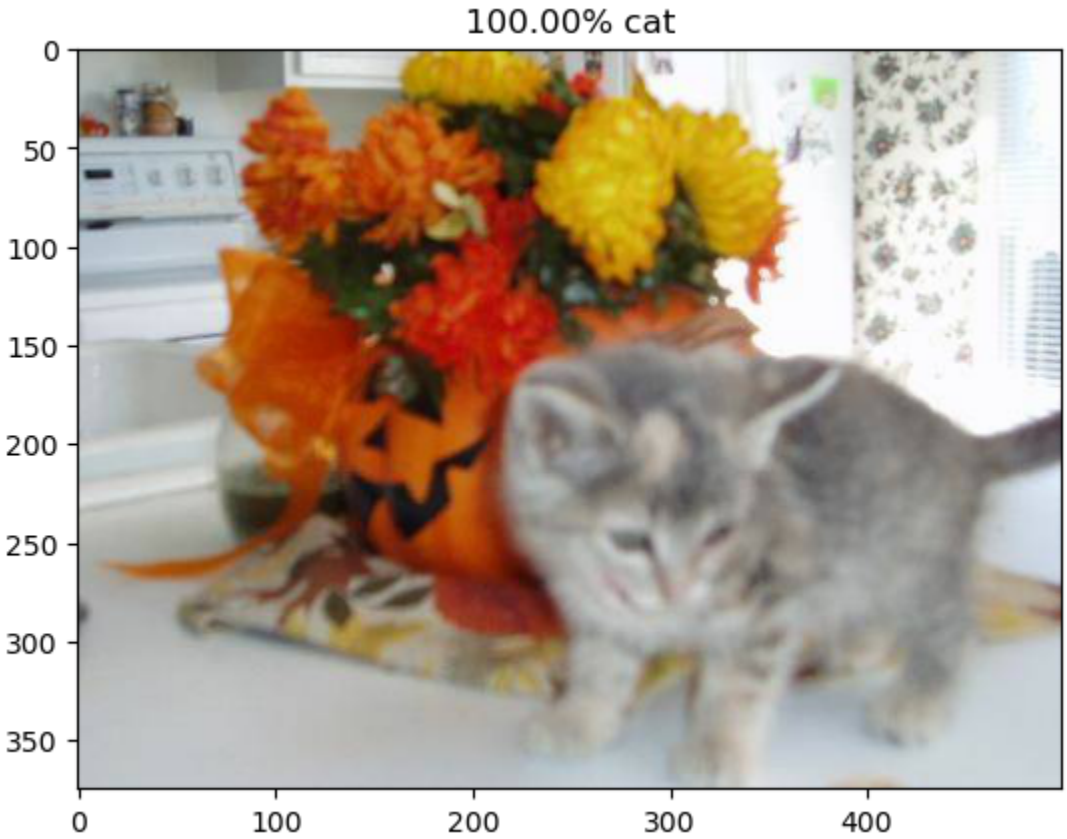
<Figure size 300x300 with 0 Axes>



<Figure size 300x300 with 0 Axes>



<Figure size 300x300 with 0 Axes>



<Figure size 300x300 with 0 Axes>

<Figure size 300x300 with 0 Axes>

For the above network (including a mixed use of dropout layer and Batch Normalization), please use image_data_generator with rotation, width_shift, height_shift, shear, and zoom to create image distortions for training. Train the network on the training set, and report the performance of the classifier on the test set. [1 pt]

In [57]:
```
train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=40,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True,
                                   fill_mode='nearest')
validation_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=20,
    class_mode='binary'
)

validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
    target_size=(150, 150),
    batch_size=20,
```

```
    class_mode='binary'
)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(150, 150),
    batch_size=20,
    class_mode='binary'
)
```

```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
Found 1956 images belonging to 2 classes.
```

In [58]:
```
model = Sequential()

model.add(Conv2D(32, (3, 3),activation="relu",input_shape=(150,150,3)))
model.add(MaxPooling2D(pool_size=(3, 3)))

model.add(Conv2D(64, (3, 3),activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (3, 3),activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(100,activation="relu"))
model.add(Dense(50,activation="relu"))
model.add(Dense(1,activation="sigmoid"))

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

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_18 (Conv2D)          (None, 148, 148, 32)      896

 max_pooling2d_18 (MaxPoolin  (None, 49, 49, 32)       0
 g2D)

 conv2d_19 (Conv2D)          (None, 47, 47, 64)        18496

 max_pooling2d_19 (MaxPoolin  (None, 23, 23, 64)       0
 g2D)

 conv2d_20 (Conv2D)          (None, 21, 21, 128)       73856

 max_pooling2d_20 (MaxPoolin  (None, 10, 10, 128)      0
 g2D)

 flatten_6 (Flatten)         (None, 12800)             0

 dense_18 (Dense)            (None, 100)               1280100

 dense_19 (Dense)            (None, 50)                5050

 dense_20 (Dense)            (None, 1)                 51

=================================================================
Total params: 1,378,449
Trainable params: 1,378,449
Non-trainable params: 0
_________________________________________________________________
```

In [59]:
```
history = model.fit( train_generator, batch_size =25, epochs=30, validation_
```

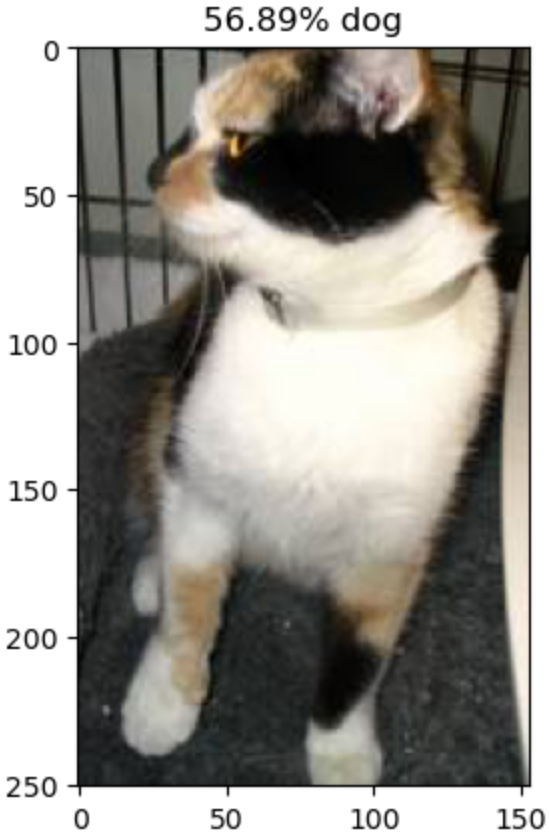
```
Epoch 1/30
100/100 [==============================] - 8s 70ms/step - loss: 0.7115 - acc
uracy: 0.5170 - val_loss: 0.6935 - val_accuracy: 0.5000
Epoch 2/30
100/100 [==============================] - 7s 69ms/step - loss: 0.6878 - acc
uracy: 0.5390 - val_loss: 0.6800 - val_accuracy: 0.5380
Epoch 3/30
100/100 [==============================] - 7s 70ms/step - loss: 0.6709 - acc
uracy: 0.5740 - val_loss: 0.9059 - val_accuracy: 0.5080
Epoch 4/30
100/100 [==============================] - 7s 69ms/step - loss: 0.6605 - acc
uracy: 0.6050 - val_loss: 0.6330 - val_accuracy: 0.6480
Epoch 5/30
100/100 [==============================] - 7s 70ms/step - loss: 0.6510 - acc
uracy: 0.6175 - val_loss: 0.6819 - val_accuracy: 0.5880
Epoch 6/30
100/100 [==============================] - 7s 70ms/step - loss: 0.6390 - acc
uracy: 0.6305 - val_loss: 0.5829 - val_accuracy: 0.6920
Epoch 7/30
100/100 [==============================] - 7s 70ms/step - loss: 0.6195 - acc
uracy: 0.6515 - val_loss: 0.9822 - val_accuracy: 0.5470
Epoch 8/30
100/100 [==============================] - 7s 69ms/step - loss: 0.6147 - acc
uracy: 0.6640 - val_loss: 0.5779 - val_accuracy: 0.7230
Epoch 9/30
100/100 [==============================] - 7s 71ms/step - loss: 0.6015 - acc
uracy: 0.6635 - val_loss: 0.5857 - val_accuracy: 0.6690
Epoch 10/30
100/100 [==============================] - 7s 71ms/step - loss: 0.6020 - acc
uracy: 0.6740 - val_loss: 0.5689 - val_accuracy: 0.7030
Epoch 11/30
100/100 [==============================] - 7s 71ms/step - loss: 0.6090 - acc
uracy: 0.6545 - val_loss: 0.5896 - val_accuracy: 0.6820
Epoch 12/30
100/100 [==============================] - 7s 70ms/step - loss: 0.5895 - acc
uracy: 0.6780 - val_loss: 0.5292 - val_accuracy: 0.7300
Epoch 13/30
100/100 [==============================] - 7s 70ms/step - loss: 0.5801 - acc
uracy: 0.6960 - val_loss: 0.5681 - val_accuracy: 0.7230
Epoch 14/30
100/100 [==============================] - 7s 71ms/step - loss: 0.5778 - acc
uracy: 0.6905 - val_loss: 0.5299 - val_accuracy: 0.7370
Epoch 15/30
100/100 [==============================] - 7s 73ms/step - loss: 0.5792 - acc
uracy: 0.6920 - val_loss: 0.5251 - val_accuracy: 0.7360
Epoch 16/30
100/100 [==============================] - 7s 73ms/step - loss: 0.5650 - acc
uracy: 0.7045 - val_loss: 0.5339 - val_accuracy: 0.7340
Epoch 17/30
100/100 [==============================] - 7s 72ms/step - loss: 0.5621 - acc
uracy: 0.7125 - val_loss: 0.5303 - val_accuracy: 0.7260
Epoch 18/30
100/100 [==============================] - 7s 73ms/step - loss: 0.5568 - acc
uracy: 0.7165 - val_loss: 0.5396 - val_accuracy: 0.7350
Epoch 19/30
100/100 [==============================] - 7s 73ms/step - loss: 0.5559 - acc
```

```
uracy: 0.7145 - val_loss: 0.5649 - val_accuracy: 0.7160
Epoch 20/30
100/100 [==============================] - 7s 72ms/step - loss: 0.5497 - acc
uracy: 0.7155 - val_loss: 0.5246 - val_accuracy: 0.7440
Epoch 21/30
100/100 [==============================] - 7s 72ms/step - loss: 0.5383 - acc
uracy: 0.7240 - val_loss: 0.5416 - val_accuracy: 0.7460
Epoch 22/30
100/100 [==============================] - 7s 73ms/step - loss: 0.5481 - acc
uracy: 0.7195 - val_loss: 0.4852 - val_accuracy: 0.7690
Epoch 23/30
100/100 [==============================] - 7s 75ms/step - loss: 0.5385 - acc
uracy: 0.7135 - val_loss: 0.5321 - val_accuracy: 0.7400
Epoch 24/30
100/100 [==============================] - 7s 74ms/step - loss: 0.5391 - acc
uracy: 0.7335 - val_loss: 0.5046 - val_accuracy: 0.7610
Epoch 25/30
100/100 [==============================] - 7s 73ms/step - loss: 0.5244 - acc
uracy: 0.7360 - val_loss: 0.5439 - val_accuracy: 0.7470
Epoch 26/30
100/100 [==============================] - 7s 74ms/step - loss: 0.5205 - acc
uracy: 0.7345 - val_loss: 0.6428 - val_accuracy: 0.6880
Epoch 27/30
100/100 [==============================] - 7s 72ms/step - loss: 0.5257 - acc
uracy: 0.7230 - val_loss: 0.5273 - val_accuracy: 0.7290
Epoch 28/30
100/100 [==============================] - 7s 73ms/step - loss: 0.5202 - acc
uracy: 0.7345 - val_loss: 0.4520 - val_accuracy: 0.7980
Epoch 29/30
100/100 [==============================] - 7s 73ms/step - loss: 0.5130 - acc
uracy: 0.7415 - val_loss: 0.4505 - val_accuracy: 0.7860
Epoch 30/30
100/100 [==============================] - 7s 75ms/step - loss: 0.5133 - acc
uracy: 0.7410 - val_loss: 0.5196 - val_accuracy: 0.7490
```

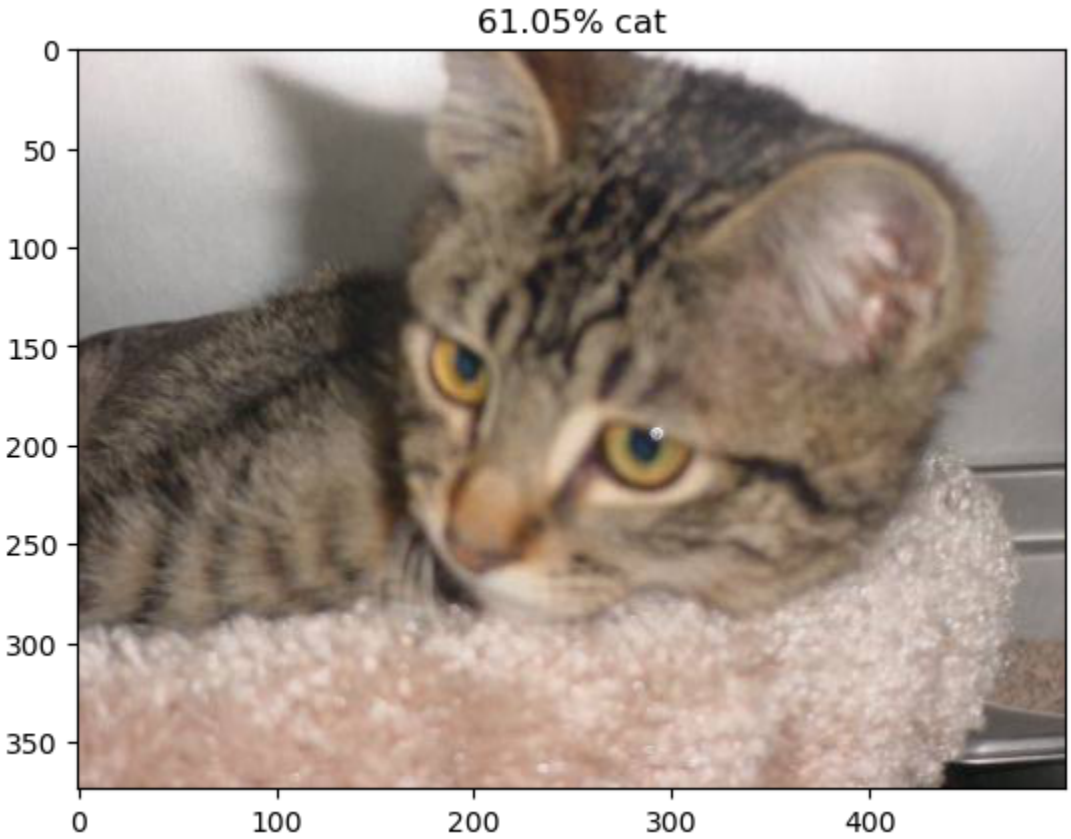
In [60]:

```
outputfile='output.txt'
numtestimages=5
open(outputfile,"w")
probabilities = model.predict(test_generator, numtestimages)

probabilities = probabilities[0:9]
for index, probability in enumerate(probabilities):
    image_path = test_dir + "/" +test_generator.filenames[index]
    img = imread(image_path)
    with open(outputfile,"a") as fh:
        fh.write(str(probability[0]) + " for: " + image_path + "\n")
    pyplot.imshow(img)
    if probability > 0.5:
        pyplot.title("%.2f" % (probability[0]*100) + "% dog")
    else:
        pyplot.title("%.2f" % ((1-probability[0])*100) + "% cat")
    plt.figure(figsize = [3, 3])
    pyplot.show()
```

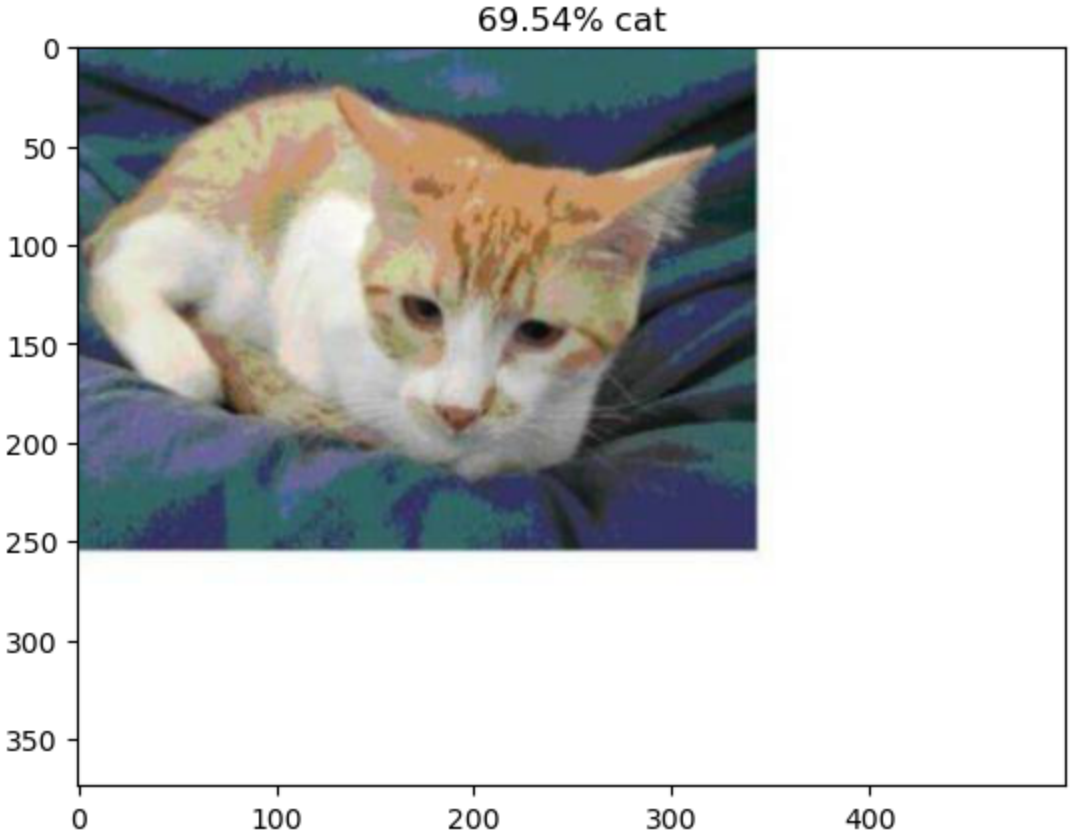
```
98/98 [==============================] - 3s 26ms/step
```



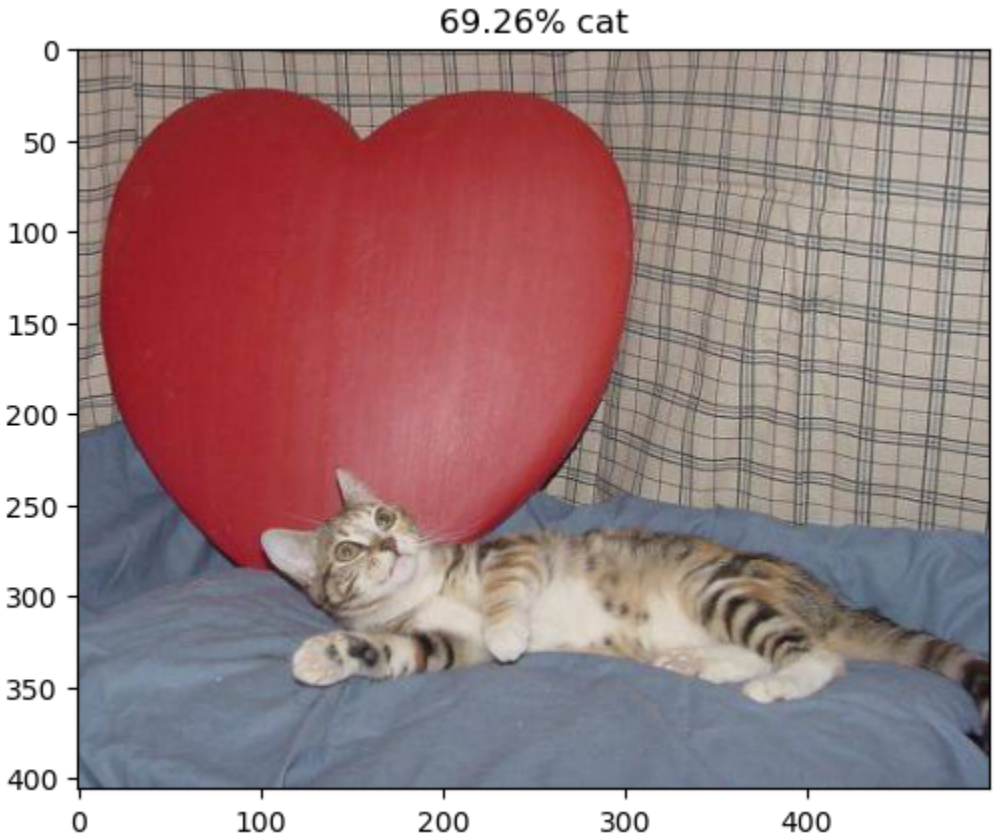
<Figure size 300x300 with 0 Axes>



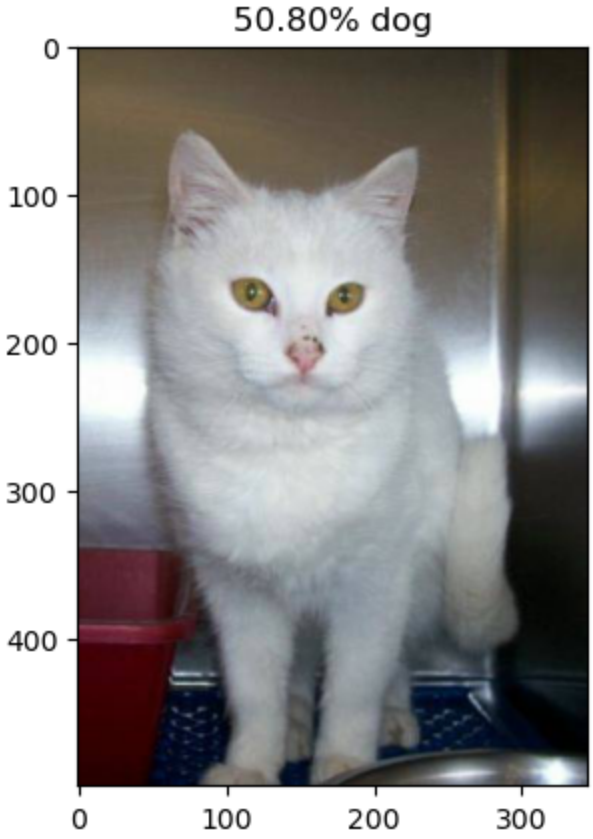
<Figure size 300x300 with 0 Axes>



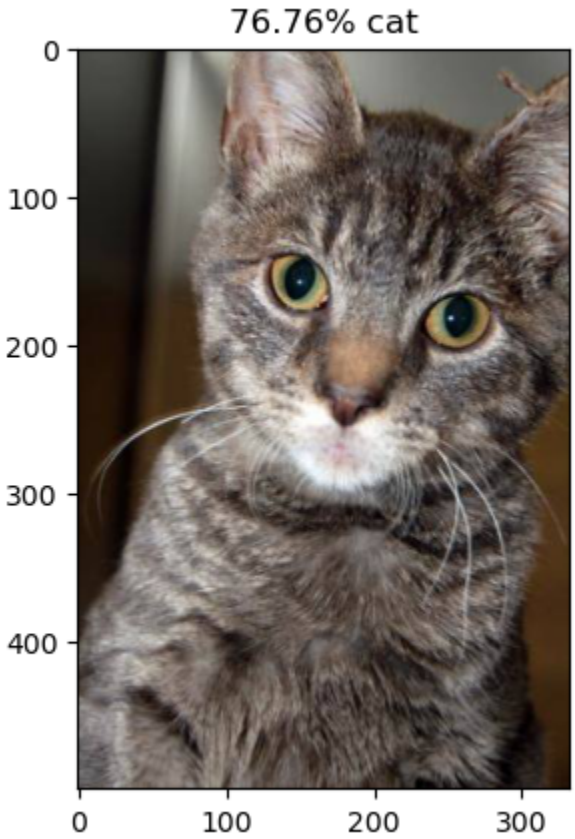
<Figure size 300x300 with 0 Axes>



<Figure size 300x300 with 0 Axes>

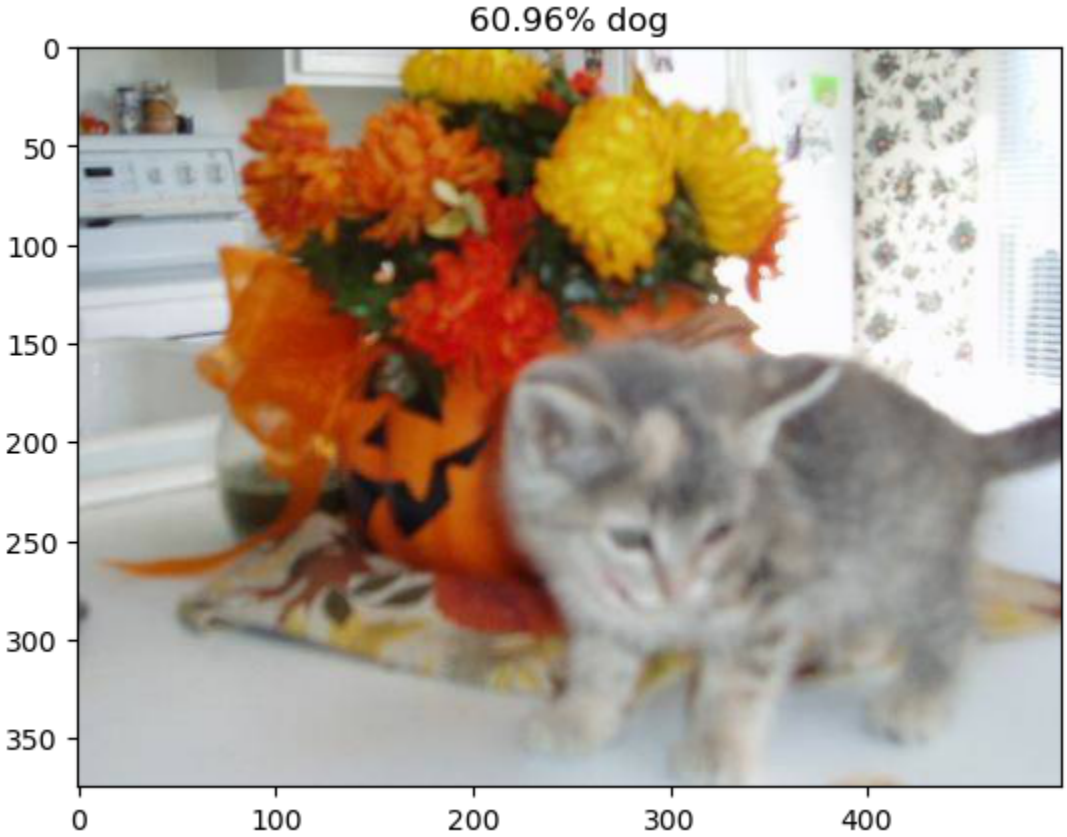



```
<Figure size 300x300 with 0 Axes>
```




```
<Figure size 300x300 with 0 Axes>
```

<Figure size 300x300 with 0 Axes>



<Figure size 300x300 with 0 Axes>


```
<Figure size 300x300 with 0 Axes>
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: