In [1]:
```
import numpy as np
import pandas as pd
import cv2
import tensorflow as tf
import keras
import matplotlib.pyplot as plt
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing import image
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, AverageP
ooling2D, Dropout
```

In [2]:
```
test_data = 'test'
train_data = 'train'
```

In [3]:
```
train_datagen = ImageDataGenerator()
test_datagen = ImageDataGenerator()
```

In [4]:
```
train_set = train_datagen.flow_from_directory(train_data,
                    target_size = (224,224),
                    batch_size = 32,
                    color_mode = 'rgb',
                    shuffle = True,
                    class_mode = 'categorical')
```

```
Found 14407 images belonging to 10 classes.
```

In [5]:
```
test_set = test_datagen.flow_from_directory(test_data,
                    target_size = (224,224),
                    batch_size = 32,
                    color_mode = 'rgb',
                    shuffle = True,
                    class_mode = 'categorical')
```

```
Found 3602 images belonging to 10 classes.
```

In [6]:
```
model = Sequential()


# 1st Convolutional Layer
model.add(Conv2D(filters=96, input_shape=(224,224,3), kernel_size=(11,1
1), strides=(4,4), padding='valid', activation='relu' ))
# Max Pooling
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'
))

# 2nd Convolutional Layer
model.add(Conv2D(filters=256, kernel_size=(11,11), strides=(1,1), paddi
ng='valid', activation='relu'))
# Max Pooling
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'
,))

# 3rd Convolutional Layer
model.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding
='valid', activation='relu'))

# 4th Convolutional Layer
model.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding
='valid', activation='relu'))


# 5th Convolutional Layer
model.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), padding
='valid', activation='relu'))

# Max Pooling
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'
))

# Passing it to a Fully Connected layer
model.add(Flatten())
# 1st Fully Connected Layer
model.add(Dense(units = 4096, activation='relu'))
```

```
# Add Dropout to prevent overfitting
model.add(Dropout(0.4))

# 2nd Fully Connected Layer
model.add(Dense(units = 4096, activation='relu'))

# Add Dropout
model.add(Dropout(0.4))

# 3rd Fully Connected Layer
model.add(Dense(units = 1000, activation='relu'))

# Add Dropout
#model.add(Dropout(0.4))

# Output Layer
model.add(Dense(units = 10, activation='softmax'))
```

In [7]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 54, 54, 96)        34944
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 27, 27, 96)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 17, 17, 256)       2973952
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 256)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 6, 6, 384)         885120
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 4, 4, 384)         1327488
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 2, 2, 256)         884992
_________________________________________________________________
```

```
max_pooling2d_2 (MaxPooling2 (None, 1, 1, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 4096)              1052672
_________________________________________________________________
dropout (Dropout)            (None, 4096)              0
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dropout_1 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_2 (Dense)              (None, 1000)              4097000
_________________________________________________________________
dense_3 (Dense)              (None, 10)                10010
=================================================================
Total params: 28,047,490
Trainable params: 28,047,490
Non-trainable params: 0
_________________________________________________________________
```

In [8]:
```
# Compile the model
model.compile(loss='categorical_crossentropy',
              optimizer='adam', metrics=['accuracy'])
```

In [9]:
```
hist = model.fit(train_set, steps_per_epoch=200, epochs=10, validation_
data=test_set, validation_steps=100)
```

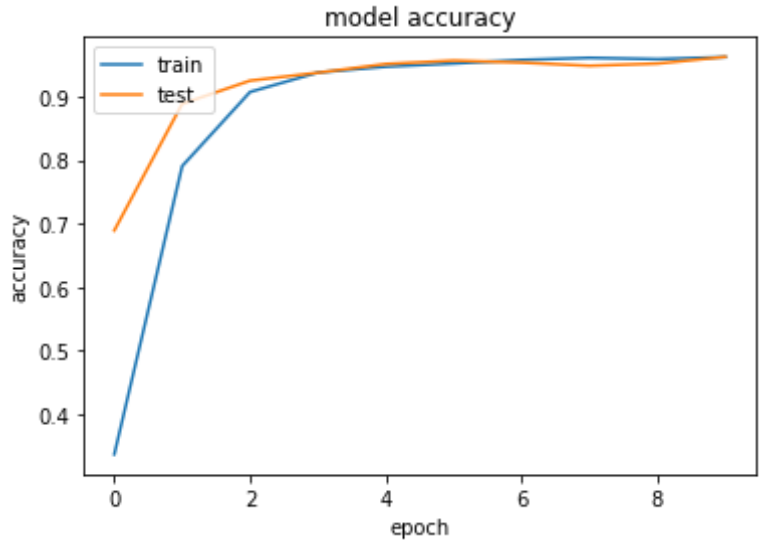
```
Epoch 1/10
200/200 [==============================] - 35s 177ms/step - loss: 2.722
9 - accuracy: 0.3366 - val_loss: 0.8666 - val_accuracy: 0.6891
Epoch 2/10
200/200 [==============================] - 32s 158ms/step - loss: 0.633
1 - accuracy: 0.7904 - val_loss: 0.4045 - val_accuracy: 0.8884
Epoch 3/10
200/200 [==============================] - 29s 145ms/step - loss: 0.343
8 - accuracy: 0.9072 - val_loss: 0.3040 - val_accuracy: 0.9250
Epoch 4/10
200/200 [==============================] - 28s 140ms/step - loss: 0.241
```

```
7 - accuracy: 0.9378 - val_loss: 0.2457 - val_accuracy: 0.9375
Epoch 5/10
200/200 [==============================] - 29s 143ms/step - loss: 0.216
1 - accuracy: 0.9470 - val_loss: 0.1791 - val_accuracy: 0.9513
Epoch 6/10
200/200 [==============================] - 28s 139ms/step - loss: 0.195
0 - accuracy: 0.9520 - val_loss: 0.1714 - val_accuracy: 0.9566
Epoch 7/10
200/200 [==============================] - 28s 139ms/step - loss: 0.183
4 - accuracy: 0.9577 - val_loss: 0.2094 - val_accuracy: 0.9534
Epoch 8/10
200/200 [==============================] - 28s 139ms/step - loss: 0.171
8 - accuracy: 0.9609 - val_loss: 0.2059 - val_accuracy: 0.9484
Epoch 9/10
200/200 [==============================] - 28s 139ms/step - loss: 0.192
2 - accuracy: 0.9589 - val_loss: 0.2552 - val_accuracy: 0.9519
Epoch 10/10
200/200 [==============================] - 27s 137ms/step - loss: 0.191
3 - accuracy: 0.9622 - val_loss: 0.2151 - val_accuracy: 0.9625
```

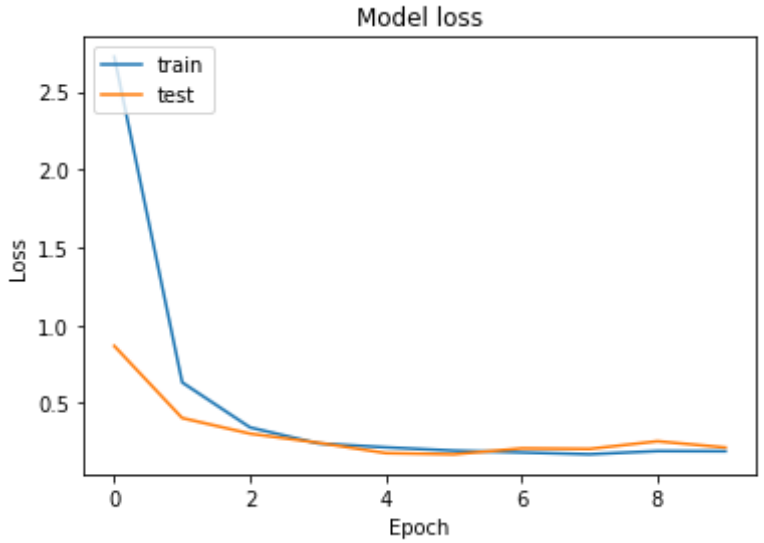
In [10]:
```
print(hist.history.keys())
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [11]:
```
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



In [12]:
```
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc = 'upper left')
plt.show()
```



In [15]:
```
 img = keras.preprocessing.image.load_img("30213.jpg",color_mode = 'rg
b', target_size=(224,224,3))  ##image for 3

img_array = keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0)                    # Create bat
ch axis

predictions = model.predict(img_array)
score1 = predictions[0]
print(score1)
```

```
[2.0905298e-18 4.9732890e-20 2.7932376e-10 1.0000000e+00 9.4291403e-15
 1.0687422e-15 3.3380085e-10 1.4069465e-10 8.1146917e-25 1.0871872e-18]
```

In [16]:
```
print(score1)                ### probability = 1 for 3
```

```
[2.0905298e-18 4.9732890e-20 2.7932376e-10 1.0000000e+00 9.4291403e-15
 1.0687422e-15 3.3380085e-10 1.4069465e-10 8.1146917e-25 1.0871872e-18]
```

In [18]:
```
img = keras.preprocessing.image.load_img("70405.jpg", color_mode = 'rg
b', target_size=(224,224,3))  ##image for 7
```

```
img_array = keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0)                    # Create bat
ch axis

predictions = model.predict(img_array)
score = predictions[0]
print(score)                                                        ###0.9
8 prob for 7
```

```
[2.7742510e-04 2.9676366e-07 2.1809428e-06 1.3913432e-04 7.5546843e-08
 4.3681299e-05 1.2821646e-02 9.8671561e-01 6.8866757e-09 1.8670679e-08]
```

In [ ]: