In [3]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image
from tensorflow.keras.optimizers import RMSprop
import tensorflow as tf
from tensorflow.keras import layers
import matplotlib.pyplot as plt
import os
import cv2
import numpy as np
```

I have downloaded fish images from https://swfscdata.nmfs.noaa.gov/labeled-fishes-in-the-wild/ which the NOAA provide.

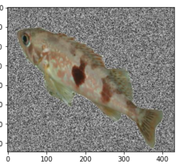
This is just an early idea and model, but currently to manage the 66 deliniated Large Marine Ecosystems (LME, https://www.lmehub.net/) they have to get scientific data and one of the means to get estimations of populations, which is used to make management decisions, is to trawl the bottom of the sea floor and count up all they pull up. A better idea would be to have an AI object classifier with multi label and object detection to count up unique individuals and get estimates that are not destructive.

This early model only takes a small data set and finds the features of fish and can tell you if it is or is not a fish through a sigmoid binary output.

This model will be a small 7 layer Neural network. I say 7 because I don't count the tf.keras.layers.Flatten() as a layer since there is no window/filter/kernal extracting more information and it is not being run through a function like relu.

In [4]:
```
img = image.load_img("Documents\Computer Vision\Fish_data\Train\Fish\DSCN0682_C.jpg")
plt.imshow(img)
```

Out[4]:
```
<matplotlib.image.AxesImage at 0x2c9f595cf40>
```



In [5]:
```
cv2.imread("Documents\Computer Vision\Fish_data\Train\Fish\DSCN0682_C.jpg").shape
```

Out[5]:
```
(372, 431, 3)
```

The cell above denotes the size of the images (height, width, number of channels) there are 3 channels since this is an RBG image.

In [6]:
```
train = ImageDataGenerator(rescale = 1/255)
```

The photos have 3 channels, red, green or blue. Each pixel is a number 0-255, so by dividing them by 255 the numbers are rescaled to be between 0-1.

From what I understand this will help the model distinguish between features. Scaling/ Normalizing in this manner should help reduce shadows and lighting issues in photos.

In [7]:
```
train_dataset = train.flow_from_directory('Documents\Computer Vision\Fish_data\Train',
                                          target_size = (200, 200),
                                          batch_size = 32,
                                          class_mode = 'binary')
```

```
Found 1306 images belonging to 2 classes.
```

The cell above pulls the images from the files and labels them as train_dataset. I did not set up a validation data set. If I did set up a validation set it would be a significant portion of the train dataset though, around 20% since there is so little data.

The test set is a series of videos from the deep sea rovers that the NOAA employed. I am still trying to find out how to run this model on the individual frames in the video.

In [8]:
```
train_dataset.class_indices
```

Out[8]:
```
{'Background': 0, 'Fish': 1}
```

In [9]:
```
model = tf.keras.models.Sequential([tf.keras.layers.Conv2D(16, (3,3),activation='relu', input_shape=(200,200,3)),
                                    tf.keras.layers.MaxPool2D(2,2),
                                    tf.keras.layers.Conv2D(32,(3,3),activation='relu'),
                                    tf.keras.layers.MaxPool2D(2,2),
                                    tf.keras.layers.Conv2D(64,(3,3),activation='relu'),
                                    tf.keras.layers.Flatten(),
                                    tf.keras.layers.Dense(512,activation='relu'),
                                    tf.keras.layers.Dense(1,activation='sigmoid')])
```

The above model will take the (200,200,3) images and feed it through a small CNN made on the tensorflow and keras libraries.

n= size of the image before going through the layer (200).

p = padding, not using any padding in this example.

f = filter/window/kernal size.

s = only using strides in the max pooling layers.

Following the formula ((n+2p-f)/s)+1 for the size after it passes through the window.

After 1st CNN layer: (198,198,16). Since (200 -3)+1 = 198 and 16 for the # of filters.

max pool effectivily divides the number of features by the size of it, here being 2. After 1st MaxPool Layer: (99,99,16) it divides (198/2) because the max pool layer has a stride of 2 in both directions.

After 2nd CNN layer: (97,97,32)

After 2nd Maxpool Layer: (48,48,32) it appears it rounds the number of features down from 48.5.

After 3rd CNN layer(46,46,64).

After flattening them all together it becomes (135424) because now we are combining all the features 46 *46* 64.

The Dense layer, honestly I don't really understand. I have read the documentation, I just know it can reduce and compact the features using elementwise multiplication and I can use it to reduce the features to the number of outputs I want. In this case 1 or 0.

At the end of this notebook, I have run model.summary() to also show the details.

In [10]:
```
model.compile(loss='binary_crossentropy',
              optimizer = RMSprop(lr=0.001),
              metrics =['accuracy'])
```

So, thankfully the keras library can call loss functions and I do not need to code them out. Keras also does foward and backward propagation so that I do not have to code the derivative functions.

This loss function 'binary_crossentropy' is a logistic loss function that will be the measure for how well the classification is doing

RMSprop or root mean square propagation is a way to increase gradient descent and will be used to update the weights and bias.

So imagine we have our label = y, original input = x, weights = w, bias = b, and z which is our inputs modified by weights and bias, z = wx+b. the t means that our wieghts have been transposed. We also have another term, a which will define z after it has gone through an activation function of g.

and our derivatives dw = derivative of w which is gotten from back prop and db = derivative of b which is essential the derivative of z. Which is also gotten from backprop.

so Z1 = W1x+b1 -> A1 = g(Z1) -> Z2 = W2A1+b2 -> A2 = g(Z2) and from this we can make a computation graph to get the derivatives. Which keras does for us.

but essentially dZ1 = A2 - y, dW1 = dZ1 * x[t], db1 = dZ1, dZ2 = A2 - y.... etc

With RMSprop, it will compute dw and db on the current mini-batch and will take the exponentially weighted averages and squares of dw and db, Sdw and Sdb to update the parameters w and b like so

w:= w - ((learning_rate)dw/Sdw)

b:= b - ((learning_rate)db/Sdb)

In [11]:
```
model_fit = model.fit(train_dataset,
                      steps_per_epoch = 3,
                      epochs = 15)
```

```
Epoch 1/15
3/3 [==============================] - 4s 1s/step - loss: 3.4694 - accuracy: 0.7083
Epoch 2/15
3/3 [==============================] - 3s 1s/step - loss: 1.1746 - accuracy: 0.6250
Epoch 3/15
3/3 [==============================] - 3s 1s/step - loss: 0.2663 - accuracy: 0.8958
Epoch 4/15
3/3 [==============================] - 4s 1s/step - loss: 1.1282 - accuracy: 0.9062
Epoch 5/15
3/3 [==============================] - 3s 1s/step - loss: 0.3417 - accuracy: 0.9375
Epoch 6/15
3/3 [==============================] - 4s 1s/step - loss: 0.2447 - accuracy: 0.9167
Epoch 7/15
3/3 [==============================] - 3s 1s/step - loss: 0.3829 - accuracy: 0.8438
Epoch 8/15
3/3 [==============================] - 4s 1s/step - loss: 0.3430 - accuracy: 0.8854
Epoch 9/15
3/3 [==============================] - 4s 1s/step - loss: 0.2502 - accuracy: 0.9375
Epoch 10/15
3/3 [==============================] - 3s 892ms/step - loss: 0.1269 - accuracy: 0.9556
Epoch 11/15
3/3 [==============================] - 4s 1s/step - loss: 0.1496 - accuracy: 0.9271
Epoch 12/15
3/3 [==============================] - 4s 1s/step - loss: 0.3345 - accuracy: 0.9062
Epoch 13/15
3/3 [==============================] - 4s 1s/step - loss: 0.2226 - accuracy: 0.9479
Epoch 14/15
3/3 [==============================] - 3s 1s/step - loss: 0.1720 - accuracy: 0.9167
Epoch 15/15
3/3 [==============================] - 3s 1s/step - loss: 0.2564 - accuracy: 0.8750
```

In [12]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 198, 198, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 99, 99, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 97, 97, 32)        4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 48, 48, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 46, 46, 64)        18496
_________________________________________________________________
flatten (Flatten)            (None, 135424)            0
_________________________________________________________________
dense (Dense)                (None, 512)               69337600
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 69,361,697
Trainable params: 69,361,697
Non-trainable params: 0
_________________________________________________________________
```

This is a very simple early model. Given more time the things I would do would be:

Gather more data. A LOT more data.

Label the data into individual species and output it to a softmax output.

Make the model a multi label, not a mult class model.

Then I would train the model (I do not have the computational power for it) and then test it on a test set and begin the analysis of improving the model by looking at the variance and bias.

I would also probably not use RMSprop, I would probably use Adam, which is RMSprop with momentum, and from my understanding, momentum takes into account the step of gradiant descent before it, helping it diverge faster.

Finally,

I would love to join this program because I see an amazing future for AI. I wish to work with indivduals who also have the skills and understanding of this type of work and work with them to approach issues in the real world and come up with solutions to them with the powerful technology.

In [ ]:
```
```