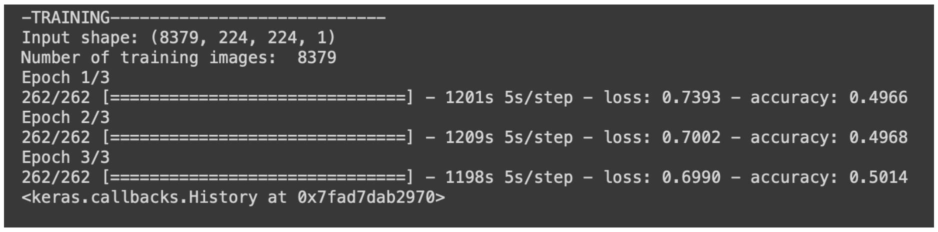
Part3: Preprocessing and feature extraction:

Since I am working with x rays, I decided to use a convolutional neural network to train my dataset. The main issue with the MURA dataset is that the x rays are not all formatted the same. Some are entire frame x rays and others you have half the image being the xray and the rest is blank space etc. I was not sure how to fix this with such a large dataset however here is what I did to set up my data for a CNN.

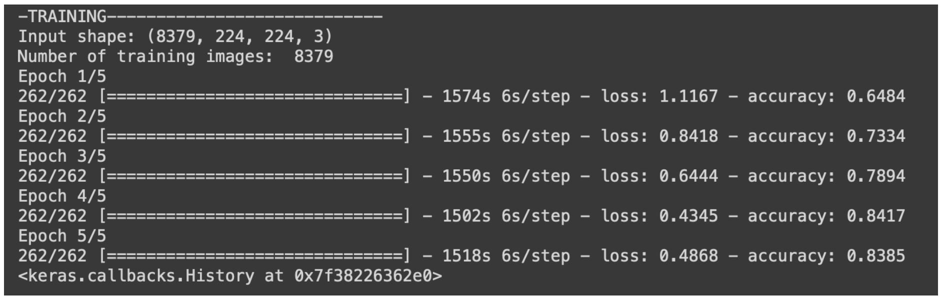
I used the class example from MNIST to format my data into x_train, y_train and x_test y_test. But first, since I decided to only use the shoulder x rays, I made a dataframe with the path to every image and then had a class for if the image was positive or negative for healthy and abnormal bones. Then, i was able to filter the dataframe to only include the shoulder x ray images. I made two data frames, one for the training data using the train folder of data from MURA, then another for the testing data from the valid folder in the MURA dataset. Once I did that, I could have used keras's generator flow from dataframe however, i chose to follow the MNIST example so I took the data frames and turned the information into ndarrays for the x_train and y_train and x_test and y_test. I also encoded the categorical labels for the training such that positive and negative correlated to 0 and 1 for classification.

For testing, I started out with manually creating the convolutional layers and dense layers for the feature extraction like used in the MNIST example, however this did not give me great accuracy no matter what combinations I chose and it takes a long time to do each epoch.The difference in the accuracy of the CNN models on the training data can also be found below, noticing that the manually made CNN model had ~0.50 accuracy while the DenseNet169 had ~ 0.83 accuracy.



```
-TRAINING---------------------------
Input shape: (8379, 224, 224, 1)
Number of training images:  8379
Epoch 1/3
262/262 [==============================] - 1201s 5s/step - loss: 0.7393 - accuracy: 0.4966
Epoch 2/3
262/262 [==============================] - 1209s 5s/step - loss: 0.7002 - accuracy: 0.4968
Epoch 3/3
262/262 [==============================] - 1198s 5s/step - loss: 0.6990 - accuracy: 0.5014
<keras.callbacks.History at 0x7fad7dab2970>
```

Figure 1: Manually made CNN model training accuracy (~0.50)



```
-TRAINING---------------------------
Input shape: (8379, 224, 224, 3)
Number of training images:  8379
Epoch 1/5
262/262 [==============================] - 1574s 6s/step - loss: 1.1167 - accuracy: 0.6484
Epoch 2/5
262/262 [==============================] - 1555s 6s/step - loss: 0.8418 - accuracy: 0.7334
Epoch 3/5
262/262 [==============================] - 1550s 6s/step - loss: 0.6444 - accuracy: 0.7894
Epoch 4/5
262/262 [==============================] - 1502s 6s/step - loss: 0.4345 - accuracy: 0.8417
Epoch 5/5
262/262 [==============================] - 1518s 6s/step - loss: 0.4868 - accuracy: 0.8385
<keras.callbacks.History at 0x7f38226362e0>
```

Figure 2: DenseNet169 CNN model accuracy (~0.83)

So, I read more about what others used when training on the MURA dataset and many used the DenseNet169 CNN structure. This uses a batch normalization layer as well instead of just relu convolutional layers and uses a singular sigmoid dense layer instead of the softmax dense layers used in the MNIST example. From what I read, this also eliminates the problem of the vanishing gradient which I would assume woul be a major issue for xrays especially when the dataset images are not all the same. The way the MNIST example stepped through the gradient at 0.1 and with a 0.9 “weight of importance” basically was not doing my dataset justice as well. For the densenet 169 I couldn't keep the singular css for the 4 dimensional arrays either, the last digit needed to be 3 so the grayscale also was different and not used as it was when I prepared the images for the manual cnn like the MNIST example. The 3 channels I think being kept helps the X Rays as otherwise the tiny details get more washed out and the coloring helps the image. Also, the Images, when converted to grayscale for the manually made CNN end up all being very different again still. Some are washed out while others seem much clearer. You can see this difference in the figure below.

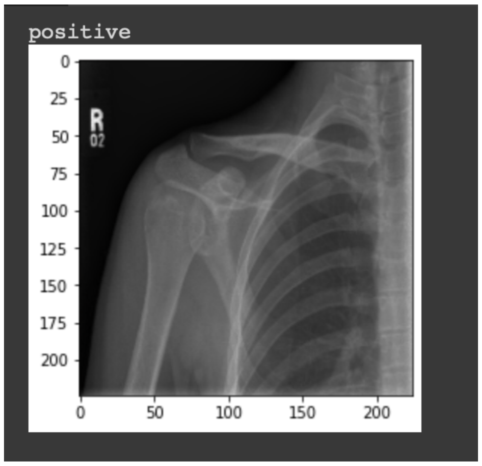
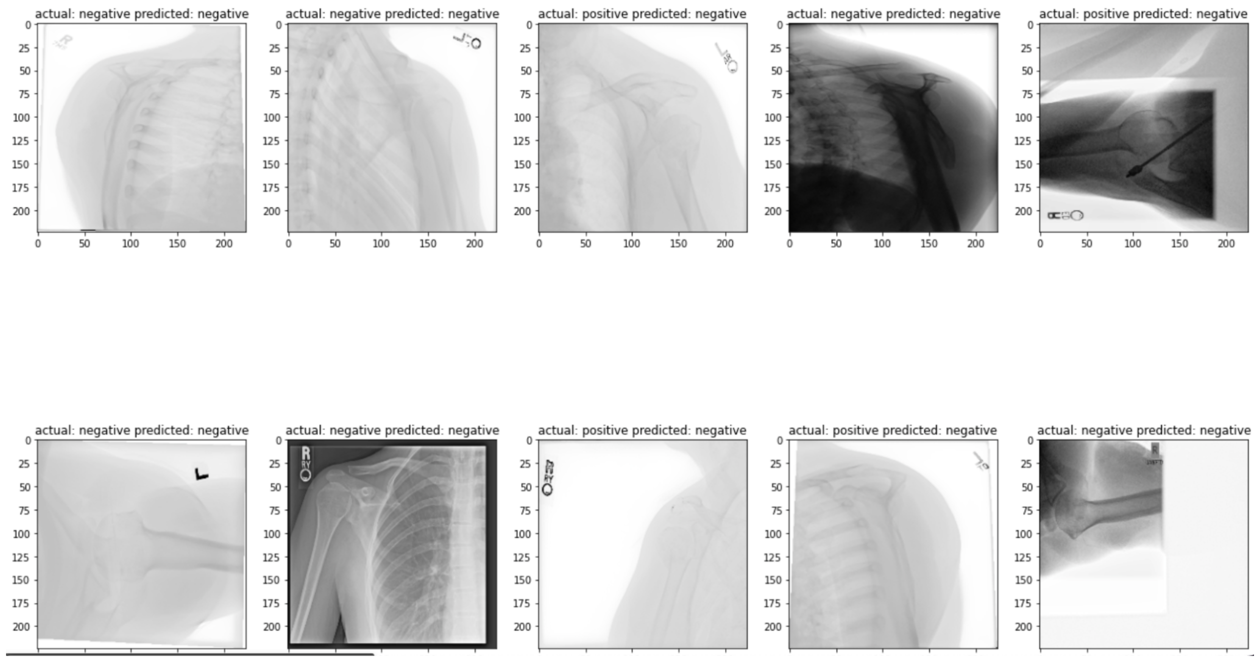
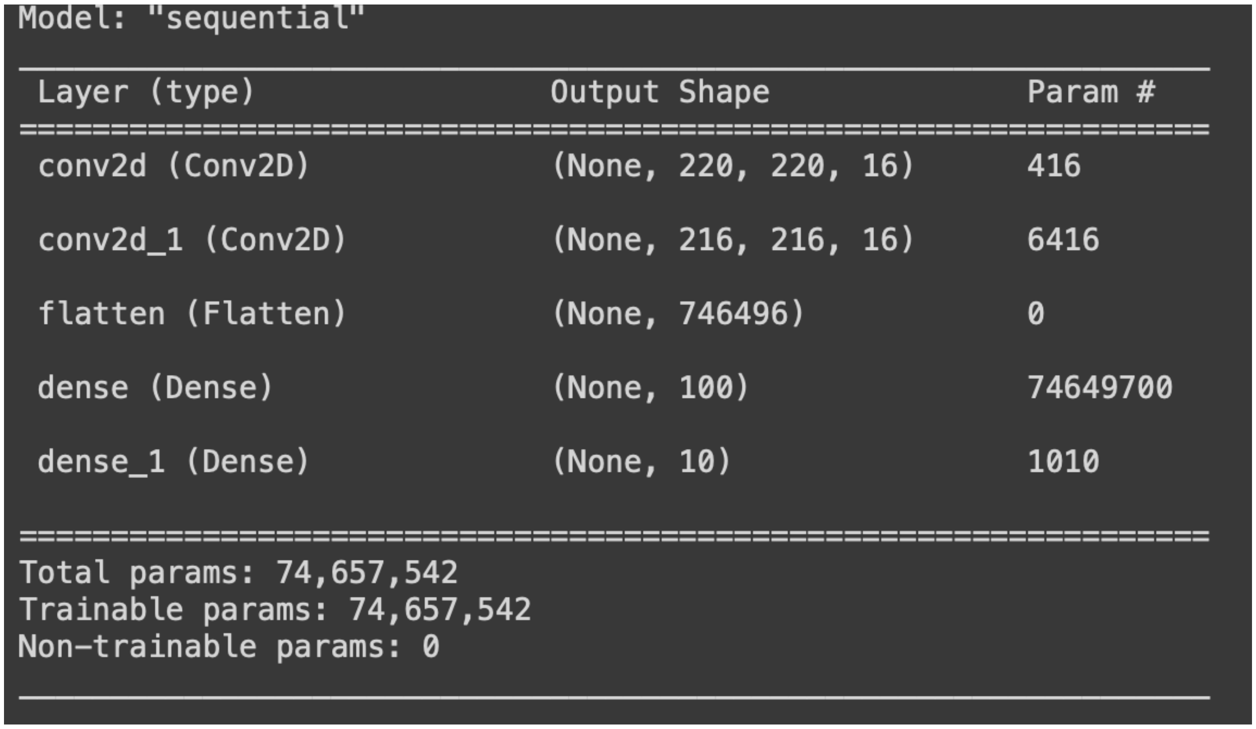


Figure 3: Image not in Grayscale (for denseNet169)

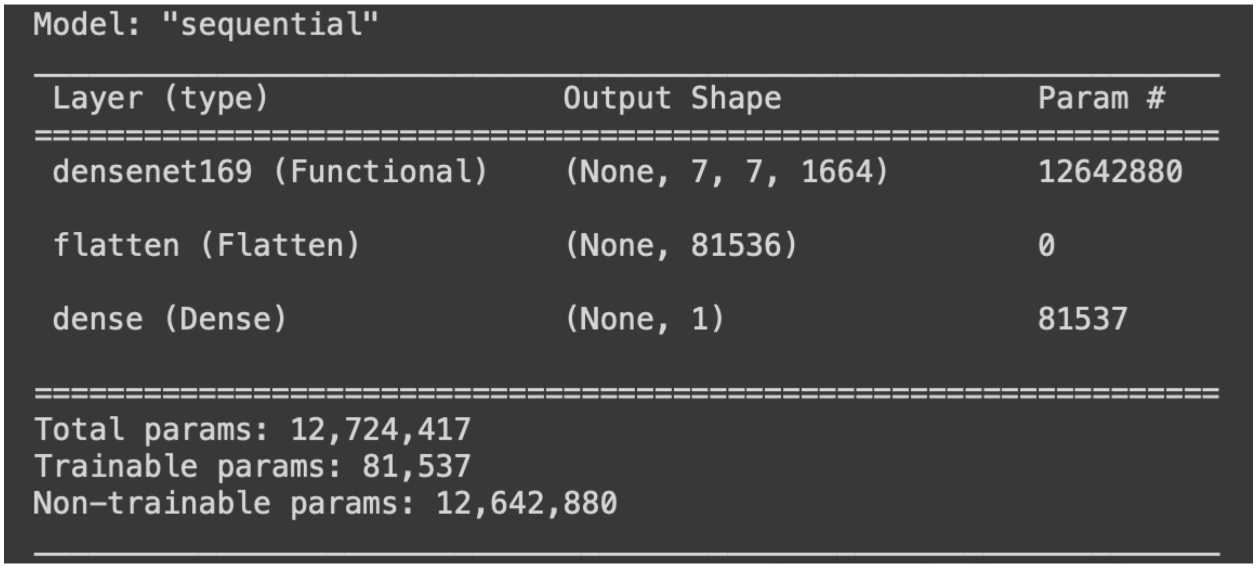

actual: negative predicted: negative
actual: negative predicted: negative
actual: positive predicted: negative
actual: negative predicted: negative
actual: positive predicted: negative
actual: negative predicted: negative
actual: negative predicted: negative
actual: positive predicted: negative
actual: positive predicted: negative
actual: negative predicted: negative
Figure 4: Manually made CNN model with grayscale images (these are test results)

One thing about the DenseNet169 is also that in the segmentation results, the non-trainable parameters are high while the manual CNN structure always had 0 nontrainable parameters. This is because it was suggested to freeze the DenseNet169 parameters since it was trained on imagenet already and thus all of those were not trainable. You can see the segmentation results below.



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 220, 220, 16)      416

 conv2d_1 (Conv2D)           (None, 216, 216, 16)      6416

 flatten (Flatten)           (None, 746496)            0

 dense (Dense)               (None, 100)               74649700

 dense_1 (Dense)             (None, 10)                1010

=================================================================
Total params: 74,657,542
Trainable params: 74,657,542
Non-trainable params: 0
_________________________________________________________________
```

Figure 5: Manually made CNN model with 0 non-trainable parameters



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 densenet169 (Functional)    (None, 7, 7, 1664)        12642880

 flatten (Flatten)           (None, 81536)             0

 dense (Dense)               (None, 1)                 81537

=================================================================
Total params: 12,724,417
Trainable params: 81,537
Non-trainable params: 12,642,880
_________________________________________________________________
```

Figure 6: DenseNet 169 with 12,642,880 non-trainable parameters (frozen because of Imagenet)