Authors: Zhenguo Mo and Efe Saatci
EECS 349 Machine Learning Final Project Report

Task

Our task is to determine whether there is a fracture when given a study (anywhere from *1-n* x-ray images of a body part). Our simple and naïve approach to this task is to create a convolutional neural net classifier that determines whether an image has a fracture and set the answer to be if the classifier determined if there was any image with a fracture.

To minimize the complexity (and memory requirements) of the task, for this final project we have limited the scope to be determining whether an x-ray of a shoulder has a fracture.

Input → A shoulder x-ray image (resized down to 32 x 32 x 1)
Output → a binary classification for fracture (1) or no fracture (0).

We have taken our image data from the MURA-v1.1 dataset provided by Stanford earlier this year (2018), which is one of the largest public health datasets available. Of this subset, we extracted the XR_SHOULDER folder from training and validation sets and used them for the respective tasks.

Data

The data is not uniformly distributed and can be very different in orientation, exposure, and content. This is a very challenging vision problem, and we do not think CNNs are effective enough in taking the 3D spatial map into account. However, it seems like an effective first approach in an area not covered in depth during the course and will provide the best results compared to other models covered in the course.

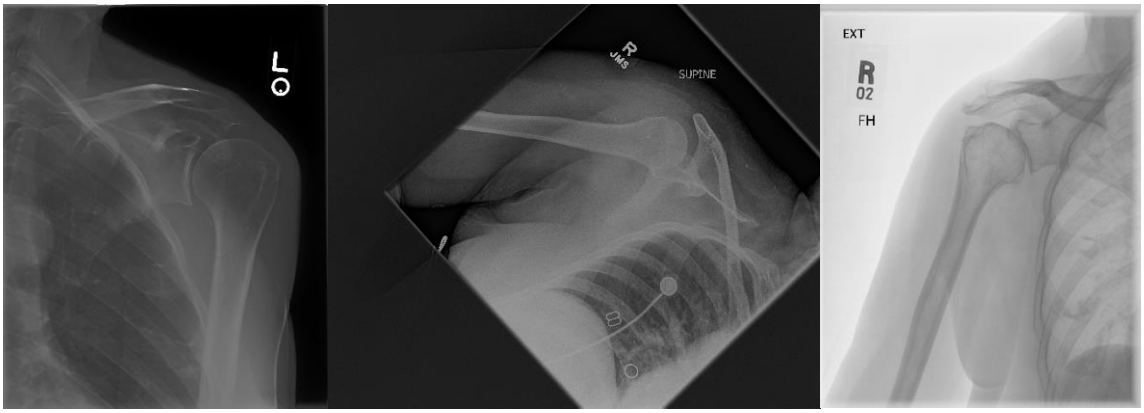


Figure 1: Examples of the wide range of images in this dataset.

One of the important points to note is that, while MURA is one of the largest public datasets out there, when taking the subset of shoulder images, there are only 3015 images total (2821 training, 194 validation images). This is not enough data to train a robust CNN. An attempt to combat this is to use data augmentation (rotations, flips, shear) to artificially create more data. A drawback to this is that shears will change the spatial relationship of the bones, but it did not seem to affect results.

Process

In this project, we used Keras with a TensorFlow backend. Keras was easier to mockup a design for convolutional neural nets than TensorFlow. A convenient data preprocessing method was

“flow_from_directory,” which can resize images (down to 32x32 for memory), shuffle the data, and feed the images as training inputs to the model. To do so, the data had to be formatted such that the hierarchy would be the following:

- Training Set
  - Class A
    - Image_1.png
    - Image_2.png
  - Class B
    - Image_3.png
    - Image_4.png
- Validation
  - Class A
    - Image_5.png
    - Image_6.png
  - Class B
    - Image_7.png
    - Image_8.png

A helpful terminal command to read the number of files in a directory is: var=$(ls -training_set | wc -l)

Then it was time to experiment with the layers and the parameters. Keras’s Sequential() method is very useful and makes it easy to set up a CNN. It was a bit difficult to train a CNN as AWS does not have very good support for deep learning on the free tier, so many constraints had to be made on the design of the layers out of concern for memory or time. The full code is provided in Appendix A.

Results and Analysis

Of the following tests, batch normalization was added after each convolutional layer, as is standard for most applications. From what we have learned, it seems to make training time faster while having a small regularizing effect on the data. The following tests were trained with 75 epochs. (We trained on smaller epochs to get a feel for the trends, but we will not report them here)

We first started off with a CNN design of the following, where x in Conv2D(x) is the number of filters and y in Dense(y) is the number of units:

Conv2D(32) -> MaxPooling2D -> Flatten -> Dense(128) -> Dense(1)

A single layer in most cases only encode direction and color. It doesn’t have very much complexity. Generally, the more convolutional layers, the more specific patterns can be found in the feature maps. Since this is the first net we tested, we expected the learning curve to be a bit poor.

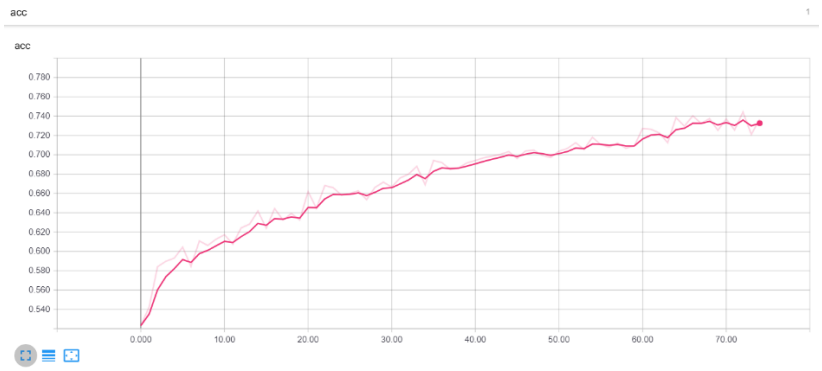


Figure 2: The training accuracy of the basic CNN

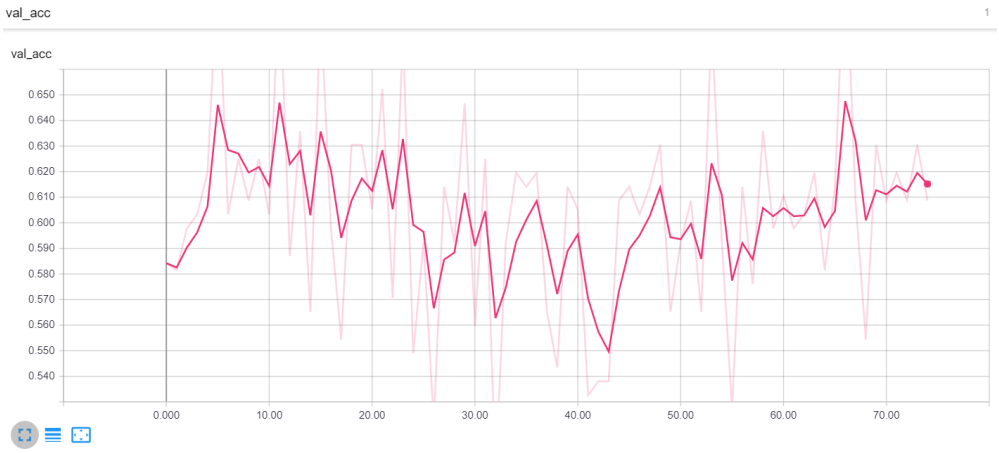


Figure 3: The validation accuracy of the basic CNN

The general structure for the three layers is in the form of the following layers, with *k* being the changes made to the model:
Conv2D(32) -> Conv2D(32) -> MaxPooling2D -> Conv2D(32) -> MaxPooling2D -> Flatten -> Dense(128) -> *k* -> Dense(1)

For some tests, we tried adding more dense layers and dropout to see the effects on the model.

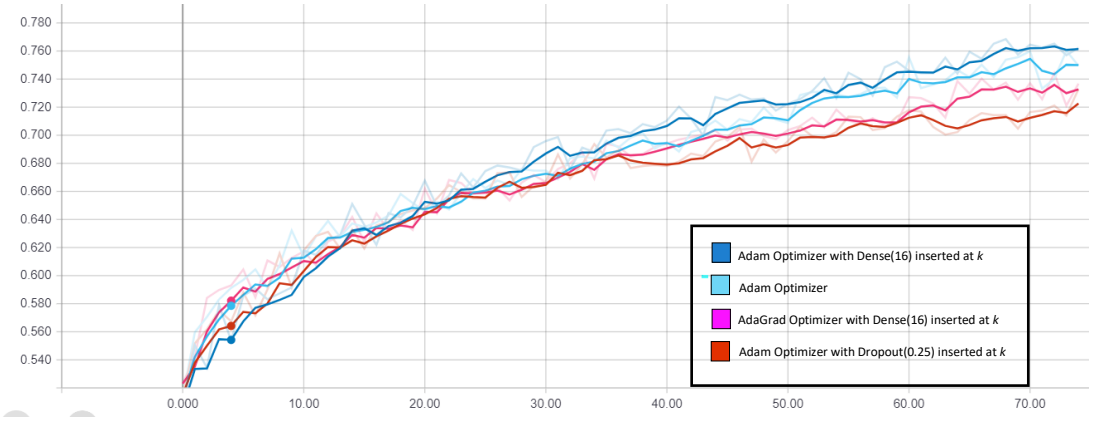


Figure 4: Three Layers Training Accuracy

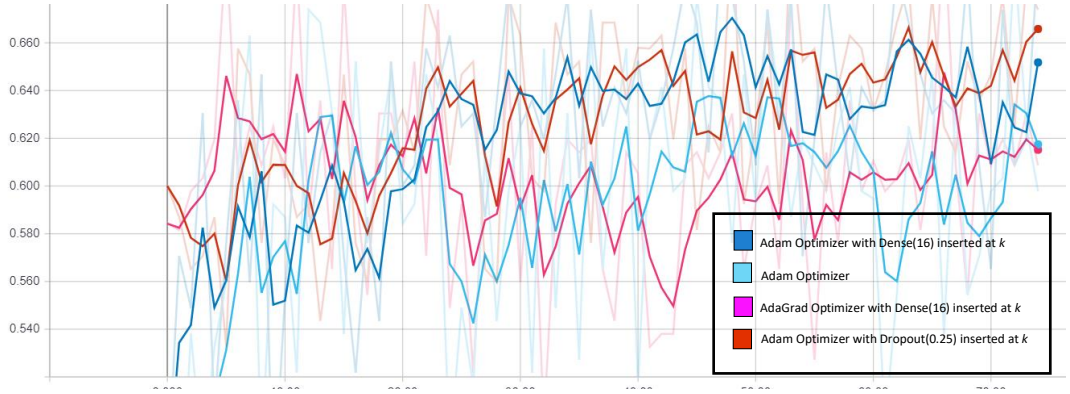


Figure 5: Three Layers Validation Accuracy

The large fluctuations of validation accuracy in each test are an unfortunate indicator that the model is not properly regularized, despite dropout, normalization, and data augmentation. This is likely because there is too much capacity in the data (not enough data points). We tested a smaller learning rate with not much improvement in validation accuracy.

When we set the random seed of our shuffle, it seems that the Adam Optimizer outperformed AdaGrad. Dropout seemed ineffective in getting the training accuracy to be higher, but in the following graph, its effects can be seen.

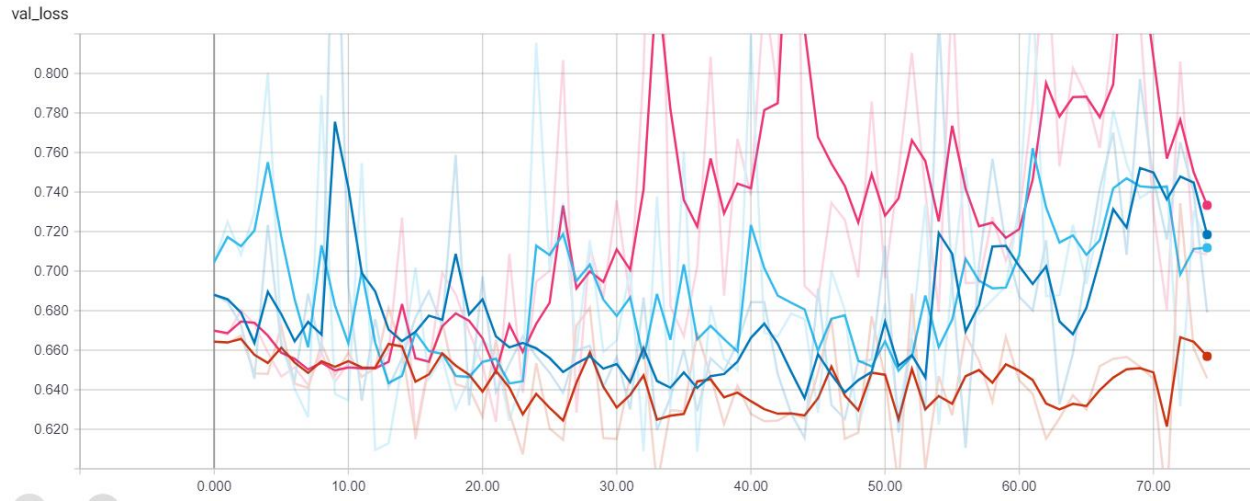


Figure 6: Three Convolutional Layers Validation Loss.

Using this information as a benchmark, we designed a final five-layer CNN to be our classifier in the following format (as reflected in the code):

Conv2D(32) -> Conv2D(32) -> MaxPooling2D -> Conv2D(64) -> Conv2D(64) -> MaxPooling2D -> Conv2D(32) -> MaxPooling2D -> Flatten -> Dense(128) -> Dropout(.25) -> Dense(1)

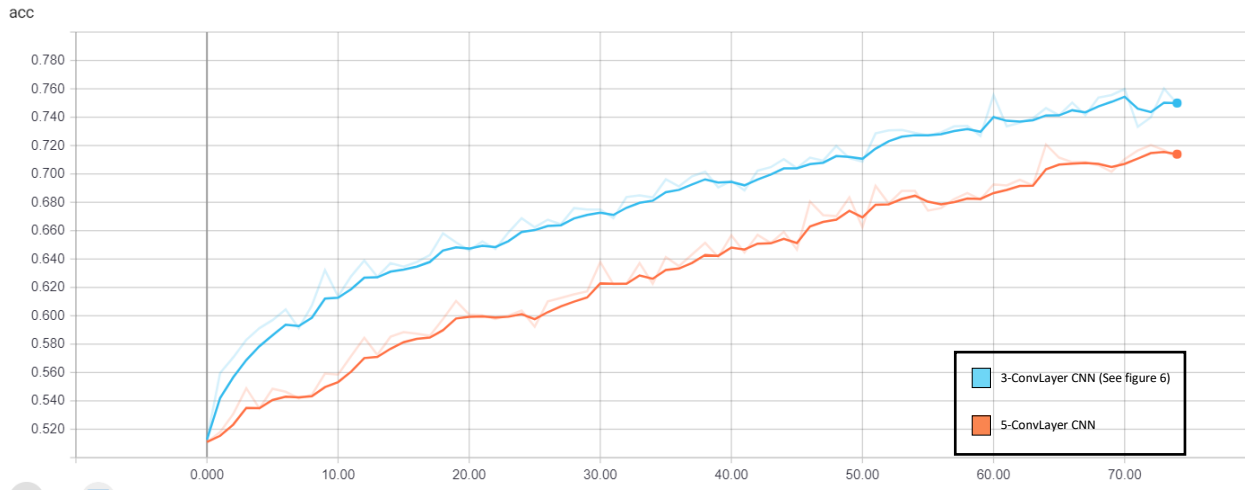


Figure 7: 3-ConvLayer CNN vs 5-ConvLayer CNN training accuracy

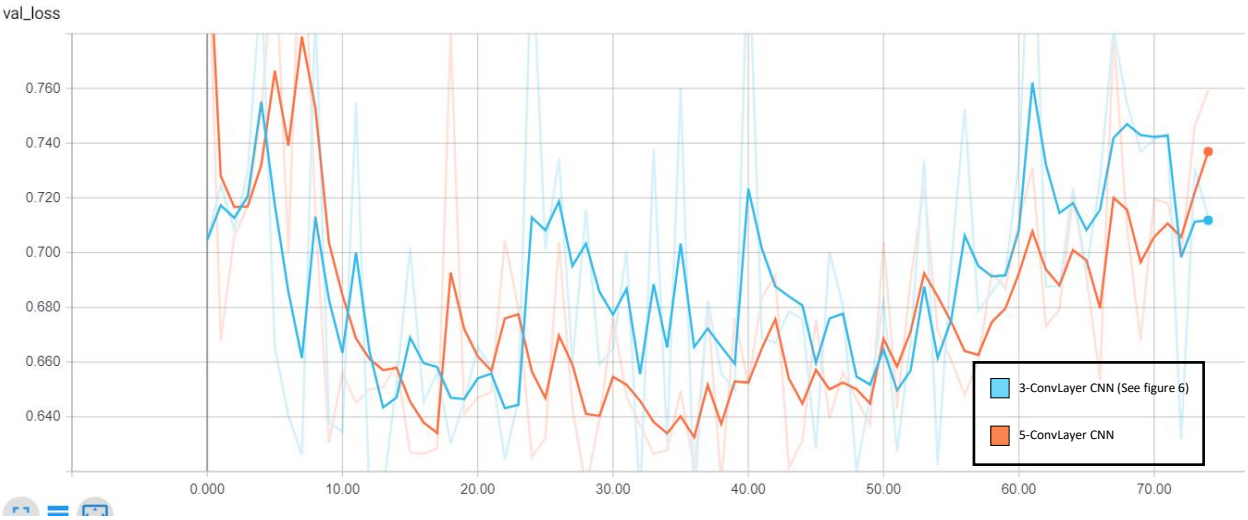
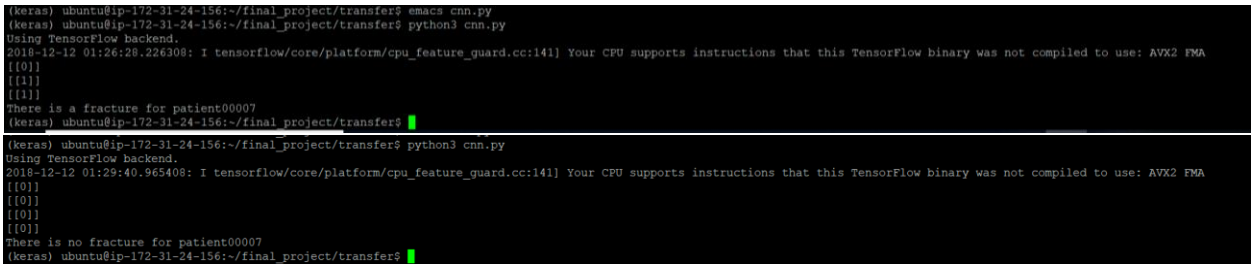


Figure 8: 3-ConvLayer CNN vs 5-ConvLayer CNN validation loss

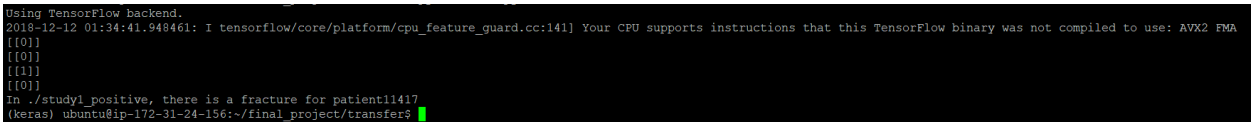
Unexpectedly, the 5-layer CNN seems to perform worse within these 75 epochs. Our best analysis is that a larger network is harder to train, and 75 epochs is not enough to train the weights accordingly. We had saved the weights, so we trained them both for approximately 25 epochs more, and the 5-layer CNN still grew linearly while the 3-layer CNN started to slow down around .77 training accuracy. This seems to indicate that the 5-layer CNN has more capacity to overfit, though we are not sure if that's true.

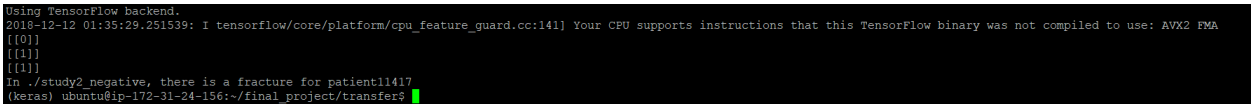
These neural nets took a very long time to train, and in the interest of time, we used the weights saved after 75 epochs from the 5-layer CNN to load all the images in a directory and predict them.

From the training data in patient 7's positive and negative studies, each shown respectively, our classifier classified them correctly.



And the results for patient11417 from the validation data is as follows:





Of the 4 tests we quickly did, 3 of them passed. Unfortunately, we did not get to implementing an accuracy logger for the studies in the entire validation set, but this naïve approach would fare poorly.

Design Choices and Analysis

Batch size was kept at 10 as it did not affect the results much. While doing research about CNN, we have read that changing batch size, in theory, only changes the speed of convergence.

$$BatchSize * StepsPerEpoch = \#\ of\ examples$$

The number of filters represent the number of features that the network can potentially learn. There isn't a standard way to find the number of filters; in general, trial and error is required. We kept the number of filters at 32 and 64 in the interest of memory.

The fully connected layers (the dense layers) were kept with relatively few units. Layers with many units are hard to train properly and given the lack of data could be detrimental to the network. We would still need to experiment more on this, however.

A suggestion would be to include the entire dataset into the training. This would take much more time to train, but it would be helpful in determining useful features of a fracture, as fractures are independent of location.

Limitations and Future Steps

Deep learning with images is a very expensive learning task, and the lack of a GPU and enough memory made it difficult to fully and freely experiment with all the parameters of the model. In the interest of time, we have limited the scope of the project extensively, causing us to not utilize much of the data provided by the MURA dataset. We would like to train a CNN on the entire dataset at some point.

We believe that, while a monitored and well-tuned CNN could yield good results, it will not be a truly accurate solution. We would like a converging validation accuracy when we utilize the entire dataset. Our future goals are to create a model for all parts of the body with the CNN and ensemble it with other models such as an all convolutional model (with a global average pooling).

Tasks done

Zhenguo Mo

- Developed, designed, and implemented the CNN model and preprocessed data for use
- Tested, logged, and collected data with Tensorboard while training
- Wrote the final report
- Finalized the website

Efe Saatci

- Created data subsets and folders from the data
- Designed the website and format
- Worked with CIFAR-10 dataset to develop a better intuitive understanding of CNNs

Appendix A: Five Layer CNN Code

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization,
Activation, AveragePooling2D
from keras.preprocessing.image import ImageDataGenerator
import numpy as np
from keras.preprocessing import image
from keras.backend import tf as ktf
from time import time
from keras.callbacks import TensorBoard
import os

def create_model():
    # initializing sequence of layers
    classifier = Sequential()
    # Convolutional and Pool Layers
    # output width of a convolution layer is (W - Kw + 2P)/Sw + 1, where W is width, Kw is kernel width, P
is padding, and Sw is stride width
    # output height of a convolution layer is (H -Kh + 2P)/Sh + 1 where H is height, Kw is kernel height, P is
padding, and Sh is stride height
    # Note, a size can be smaller than the expected input. The default is valid padding, where only the
valid part of the image is used.
    classifier.add(Conv2D(32, (3, 3), input_shape=(32,32,1))) # First Layer
    classifier.add(BatchNormalization()) # Adding batch normalization to speed up learning in layers.
    classifier.add(Activation('relu'))
    classifier.add(Conv2D(32, (3, 3))) #Second Layer
    classifier.add(BatchNormalization())
    classifier.add(Activation('relu'))
    classifier.add(MaxPooling2D(pool_size=(2,2)))
    classifier.add(Conv2D(64, (3, 3))) #Third Layer
    classifier.add(BatchNormalization())
    classifier.add(Activation('relu'))
    classifier.add(Conv2D(64, (3, 3))) #Fourth Layer
    classifier.add(BatchNormalization())
    classifier.add(Activation('relu'))
    classifier.add(MaxPooling2D(pool_size = (2, 2)))
    classifier.add(Conv2D(32, (3, 3), activation='relu')) #Fifth Layer
    classifier.add(BatchNormalization())
    classifier.add(MaxPooling2D(strides = (2, 2)))
    # FC layer
    classifier.add(Flatten())
    classifier.add(Dense(units=128, activation='relu'))
    classifier.add(BatchNormalization())
    classifier.add(Dropout(0.25))
    classifier.add(Dense(units=64, activation='relu'))
    # Initialize output layer
```

```
    classifier.add(Dense(units=1, activation= 'sigmoid'))
    return classifier

def cnn_classifier(loadmodel=None):
    classifier = create_model()
    #adam = Adam(lr = 0.00001, beta_1 = 0.95, beta_2 = 0.999)
    if loadmodel:
        classifier.load_weights(loadmodel)
    classifier.compile(optimizer= 'adam', loss= 'binary_crossentropy', metrics= ['accuracy'], )
    # Part 2
    tensorboard = TensorBoard(log_dir="logs/{}".format(time()))
    train_datagen = ImageDataGenerator(rescale = 1./255,
                                       shear_range = 0.2,
                                       zoom_range = 0.2,
                                       horizontal_flip = True)
    test_datagen = ImageDataGenerator(rescale = 1./255)
    training_set = train_datagen.flow_from_directory('./MURA-v1.1/training_set',
                                                     target_size = (32, 32),
                                                     color_mode = 'grayscale',
                                                     shuffle=True,
                                                     batch_size=10,
                                                     seed = 1,
                                                     class_mode = 'binary')
    test_set = test_datagen.flow_from_directory('./MURA-v1.1/valid',
                                                target_size = (32, 32),
                                                batch_size = 10,
                                                seed = 1,
                                                color_mode = 'grayscale',
                                                class_mode = 'binary')
    STEP_SIZE_TRAIN = 2821//10 #num_examples//batch size
    STEP_SIZE_VALID = 194//10
    classifier.fit_generator(training_set, steps_per_epoch = STEP_SIZE_TRAIN, validation_data = test_set,
                             validation_steps = STEP_SIZE_VALID, epochs=25, callbacks=[tensorboard])
    classifier.save_weights('Five_convolutions_layers.h5')
    classifier.summary()

# Given saved weights and an absolute path to a particular study instance, load and predict fracture or
not
def load_and_predict(model_weights, study_dir):
    model = create_model()
    model.load_weights(model_weights)
    imgs = os.listdir(study_dir)
    counter = 0
    for img in imgs:
        test_image = image.load_img(study_dir+'/'+img, target_size=(32, 32), color_mode="grayscale")
        test_image = image.img_to_array(test_image)
        test_image = np.expand_dims(test_image, axis=0)
        test_image /= 255 #Normalizes input
        answer = model.predict(test_image)
```

```
        answer = (answer > 0.5).astype(np.int)
        #print(answer)
        if(answer == 1):
            counter += 1
    return counter > 0 #If any output a 1, then it's a fracture

if __name__ == '__main__':
    load_weights = './Five_convolutions_hundred_epochs.h5'
    study_dir = './study1_positive'
    #study_dir = './study1_negative'
    if load_and_predict(load_weights, study_dir):
        print("There is a fracture")
    else:
        print("There is no fracture")
```