2017-07-03

- Initial set up of neural net, based on leNet-5, no signal out, most likely due to using unprocessed data.
- The data is very padded, this has been reduced with the function `cropHeart(inp)`, but I will need to make sure all the files are the same size before they get fed into the CNN.
- I could try normalising the data to get a signal, but will need to get the unpadded data working first.

2017-07-04

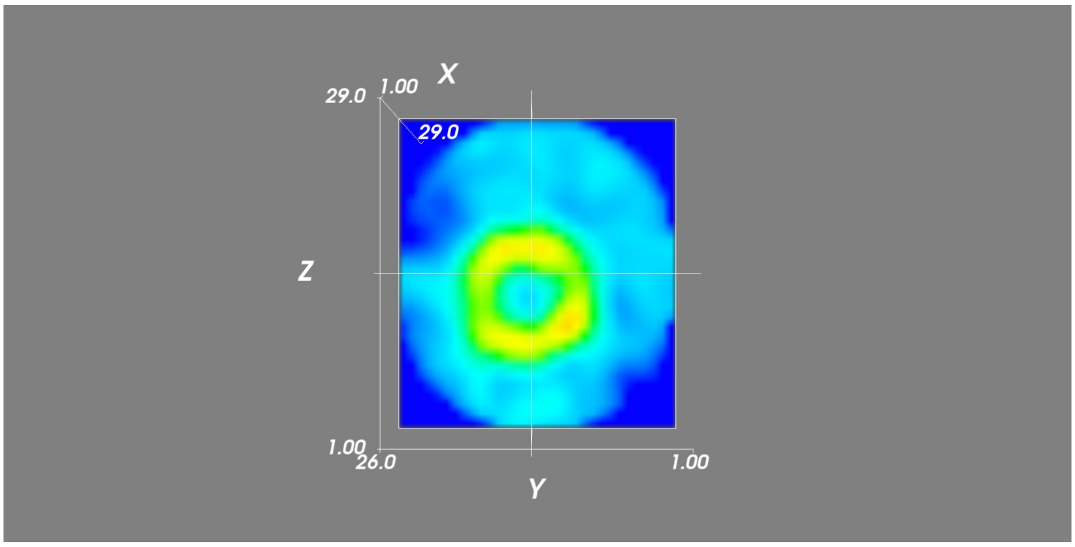
- Wrote a visualisation of the data (`visualisation.py`).
- Still working on repadding the cropped data (It's a bit of a pain).

2017-07-05

- Repadded the cropped data, it is now of size [68,34,34].
- Retrying the CNN with the new data doesn't get a signal. Maybe there isn't enough data to make it work?
- I will fiddle with the hyperparams to see if I can pick something up.
- Maybe normalising the data will help.

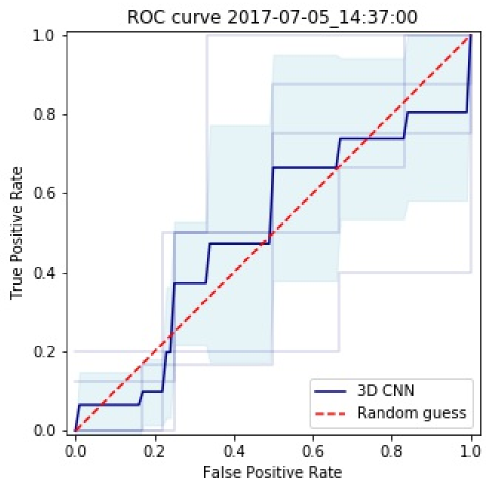
Got some results using 2D slices:

- The 2D slices I used look like:

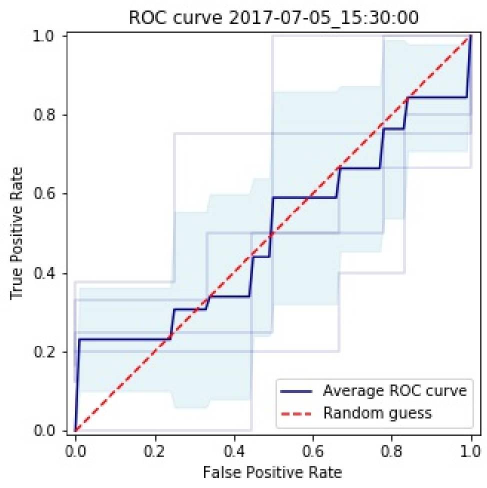


- I have used 2d slices of the data and it works well (halfway through the z-axis). It uses:
    - Slice of rest and slice of stress on z axis. Spacial x and spacial y on x and y axes.
    - LeNet-5 CNN with 3D convolution and subsampling.
    - [2,5,5] filters, pooling 2 with step 2.
    - learning rate of 0.0001, with ADAM optimiser, and batch size of 10.
    - After 50 epochs of 58 images it learns to ~95%.
- I will now apply a k-fold x-validation to it to see if it's not just picking up noise.

- The k=10-fold x-validation shows that the CNN is learning the noise in the data, although this could be due to the small amount of images in each k-fold (only 6!):



- I tried normalising the arrays, with no luck. It stopped overfitting the data, but still hasn't learnt significantly:

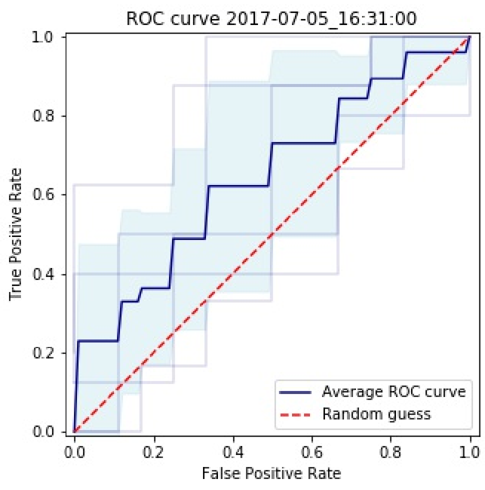


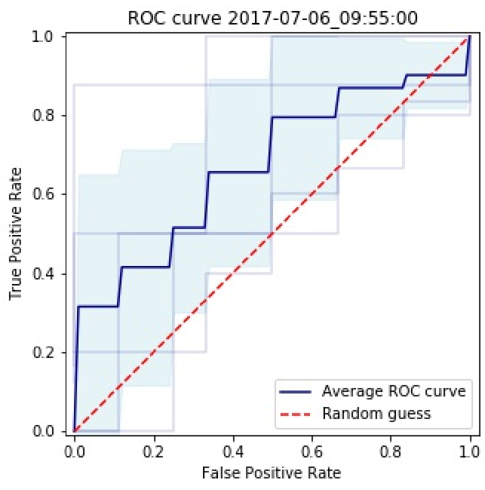
- I think the issue is still the massive amount of blankspace. I should try and scale the arrays so that they are the same size.

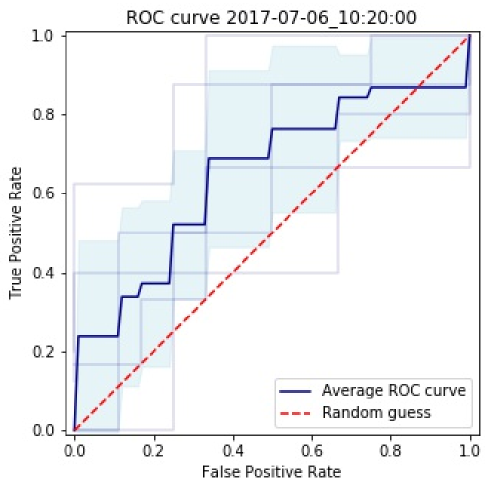
Have a signal!

- I have got a signal with the following CNN:
    - Slice of rest and slice of stress on z axis. Spacial x and spacial y on x and y axes.
    - LeNet-5 CNN with 3D convolution and subsampling.
    - [2,10,10] filters, pooling 2 with step 2.
    - learning rate of 0.0001, with ADAM optimiser, and batch size of 10.
    - 5 k-folds.
    - After 50 epochs of 47 images it learns training data to ~95%.
    - Over three repeats:
        - Avg Spec: 0.583, 0.623, 0.663
        - Avg Sens: 0.633, 0.683, 0.700

- ROC curves:







2017-07-06

- I redid the 2D slice data with three slices along the x, y, and z axes. It will take ~100 mins to finish learning. It's probably time to use some better hardware.

- Found a function (https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.ndimage.interpolation.zoom.html) which should work well for resizing the images.

- Maybe the reason that the slicing works, and the 3D data doesn't is because the CNN filter only sees one 3D image at a time, and sees both the rest and stress images at the same time in the 2D slice data. I could write a 4D CNN to fix this.

    - mhuen seems to have written a 4D convolution by stacking 3D CNN outputs (https://github.com/mhuen/TFScripts/blob/master/py/tfScripts.py). This might work for what I want to do, and stacking can be used for pooling too.

- I wrote a scaling function hat eliminates most of the whitespace. After training the CNN did not learn significantly.

- Added a ROC AUC calculator to the outputs.

- I'm going to try artificially expanding the data.

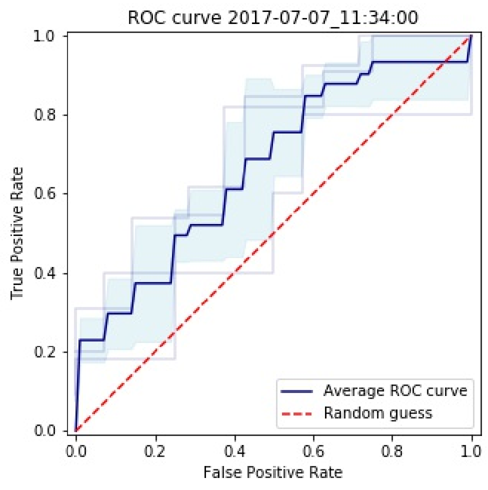
2017-07-07

- Because of the overfitting going on when running the CNN, I increased the L2 regularisers' weight decay from 0.001 to 0.01, and added an extra dropout layer between the two FC dense layers.

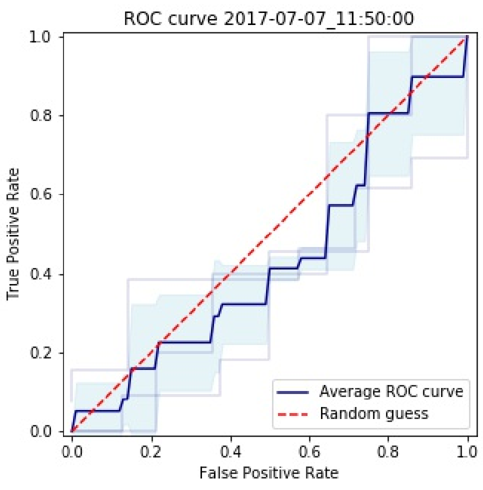
- Can't seem to get any results with a spec/sens over 60%, probably due to the way I'm orgainising the data.

- The CNN appears to train better when using non-scaled data. I can't figure out why. Maybe it's using the image sizes as an aid?

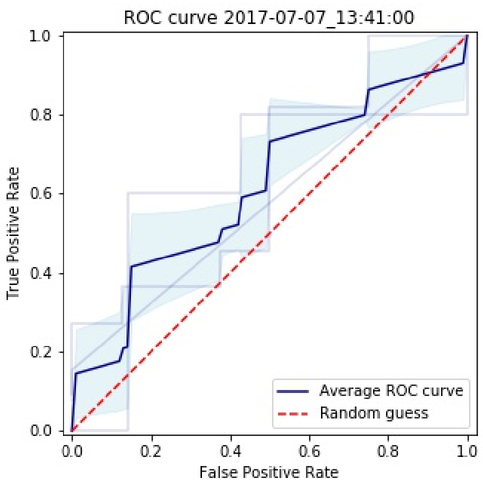
    - Conv filter: [2,15,15]; pool filter: [2,2,2]; 2 FC 1024 neurons, L2 regulisation at weight decay = 0.001, dropout at 0.5 after each FC layer; ADAM optimiser, learning rate = 0.0001, categorical x-entropy loss; batch size = 10 at 38 datum; k = 3 folds.

        - non-scaled: 

- scaled:



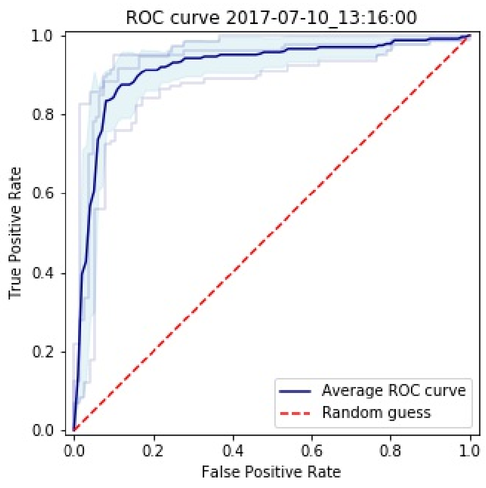
- scaled, not renormalised:

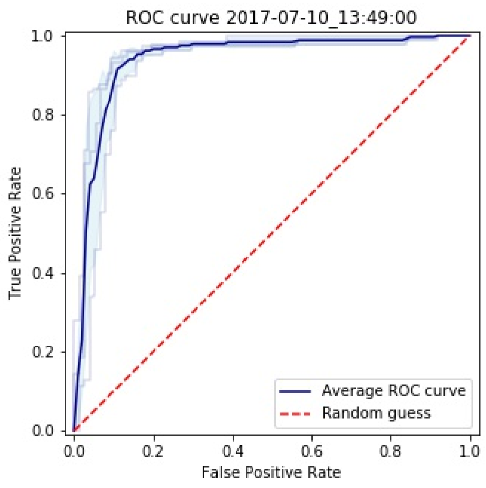


- As shown in the ROC curves, the only data that is causing consistent learning is the non-scaled one. I don't know why.
- Rewrote heart_data.ipynb so that it can resize the input data.

2017-07-10

- It might be better to use a Siamese CNN instead of a 4D CNN to compare two 3D images, as training will be faster.
  - I have written CNNs using two-channel, and Siamese architectures, along with the OG 3D convolution architecture. The two-channel and Siamese architectures are described here: (https://arxiv.org/pdf/1504.03641.pdf).
- The use of a very deep NN architecture would reduce linearity, and may be useful.
- Artificially expanding the data seems to have worked. I am getting after k = 3 folds (100 epochs) at 619 datum (two runs):
  - Spec: 0.864, 0.917
  - Sens: 0.888, 0.883
  - ROC AUC: 0.918, 0.940
  - This is with the two-channel architecture. ROC curves:

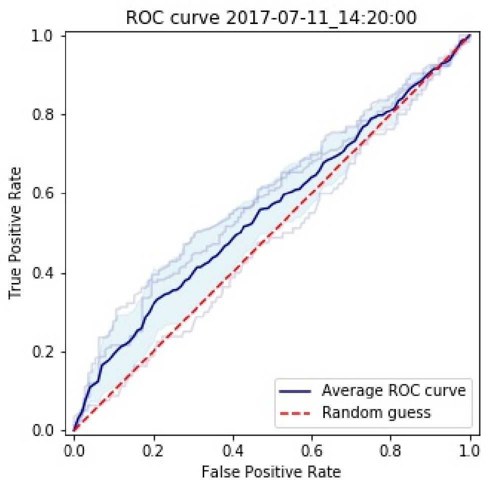
- ▪ 

- ▪ 

- Haven't got any significant results from the Siamese CNN, but have only trained it to ~30 epochs. It will probably need more training than the two-channel as there are nearly twice as many weights in the Siamese CNN.
- I should try validating the CNN on ppts that it hasn't seen before (like take 10 ppts from the pool before artificial expansion and use these to validate).

2017-07-11

- I have separated ppts into different k-folds before expansion, so each k-fold has unique ppts in it now, even after artificial expansion. We'll see how it performs now... (This is in the 2channel ipynb)
    - It doesn't work very well. Getting ~50% accuracy.

- 

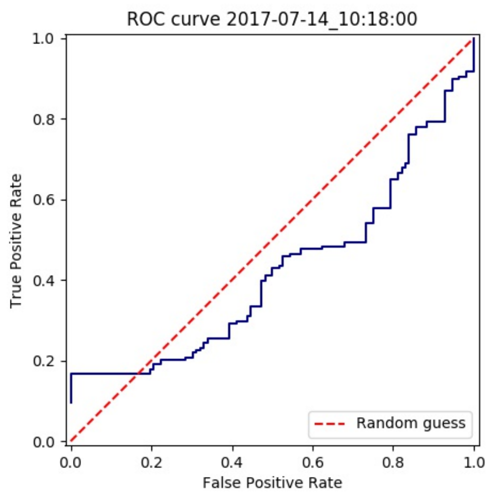
- More data would be helpful to reduce overfitting, but using all three dimensions of the heart data may be enough to get "good enough" results.
- I have written a 2 channel CNN for the 3D data. It should be ready to try on the supercomputer.

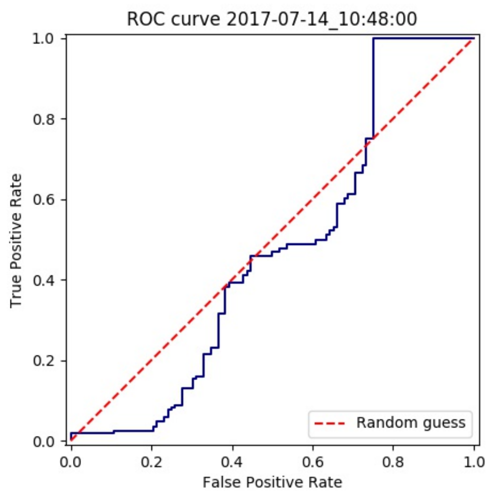
2017-07-12

- Testing the 2dSiameseCNN on the supercomputer:
  `$ qsub -q gpu -l nodes=1:ppn=16 -I -X -l walltime=24:00:00`
  It doesn't seem to work. How long is the queue?
- I have found a bug in the 2dsliceCNN that may be causing the lack of learning. The expansion doesn't relabel the expanded data correctly. I have hopefully fixed this.
- Running for 20 epochs @ k = 5 folds to see how it does.
  - Again, ~50% accuracy.
- I have increased the number of conv layers to 4.
  - No change.
- Running the 3D CNN on the hub. It looks like it takes ~20 epochs to train to 100% (I should use validation to see if/when it starts overfitting). It also takes ~12s to train an epoch. To contast it takes my computer ~16mins per epoch, an 80x speedup.
- Added 4(!) new convolution layers to 3D CNN. Since this reduces linearity, we may find something.
  - Getting some odd results. The CNN comes out with the opposite of what I was expecting (low ROC, accuracy).
  - Look at labelling, try on simpler data (MNIST 0s and 1s?), reintroduce k-folding?

2017-07-14

- Added overall average performance metric to 3dCNN-nokfold.

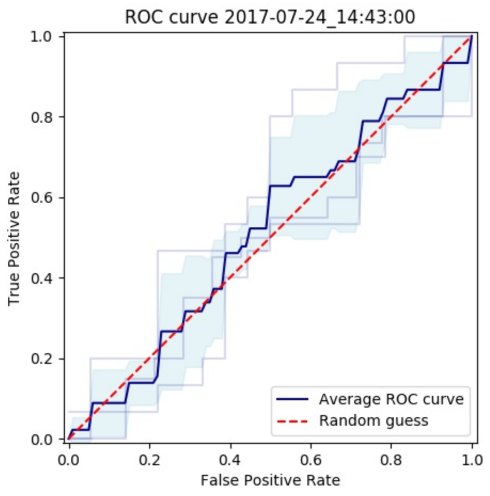
- I think I have found the cause of the low ROC/accuracy. The random state shuffle is set to 1. If I change it, it may get some more believable results.
- Looks like that was what the issue was. The CNN got lucky with the cubes taken out for testing:
  - 

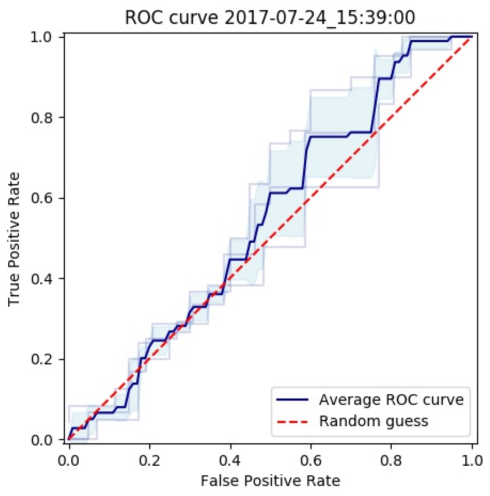
  - 

- Using fake data to train a CNN. It's found on `/data/jim/Heart/sims`.
- The CNNs aren't training. For normal/infarction data I have the loss decreasing but the accuracy is static. [(https://www.reddit.com/r/cs231n/comments/4p12oc/what_does_it_mean_when_the_loss](https://www.reddit.com/r/cs231n/comments/4p12oc/what_does_it_mean_when_the_loss) < It looks like it is due to the CNN training well on "easy" examples.

2017-07-24

- Trying a CNN with both heart cubes encoded as one (through matrix multiplication).
- The fake data doesn't seem to be working. I should look for ways to reduce noise in it.
- There was an error in my normalisation function. I'm going to go back and fix it and see if anything happens.
- It's finding something, but it looks like it's getting stuck in local minima. I'll fiddle about with the learning rate.
- Reducing the ppts to 50 healthy 50 unhealthy has got an accuracy of ~70%. This is promising. Maybe the CNN just needs a while to learn?

- Got this:
    - 

- Running again with 400 epochs. Taking ~20 min per k-fold of 100 ppts. I should try this with the full dataset. It will take a long time (~7 hours), so if this works I'll run overnight.
- Got this for 400 epochs:
    - 

- Set up a job for overnight. We'll see how it does tomorrow.

2017-07-25

- To run headless on a server I need the following at the top (matplotlib uses X by defualt).

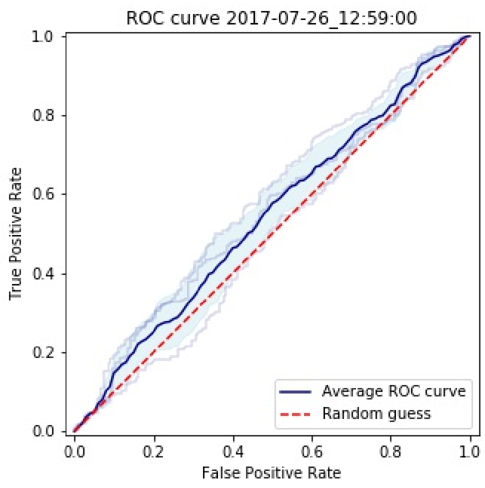
```
    import matplotlib
    matplotlib.use('Agg')
```

- Rerunning the 400 epoch 1500 ppt CNN...
- There may be an issue with the resizing of the arrays into the zeroArr. I think removing the centring code fixes it, and doesn't affect the CNN. Running a test on the 2D CNN.
    - I can safely remove the centring code.

- It takes a very long time to denoise the heart cubes. Will need to do this on the server.
- OOM error! I will need to rewrite the python script so that each k-fold is considered separately. Looks promising though: ~0.6 accuracy after 400 epochs.

2017-07-26

- The results from the latest run have training accuracy at 55%, with validation accuracy around the same (mean AUC = 0.54). I'll try again with less regularisation (it may be underfitting).

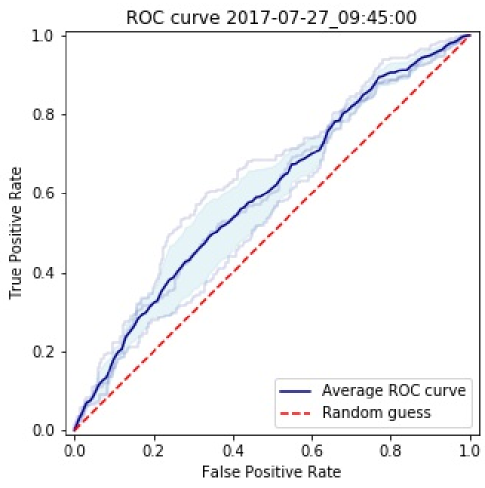
    - 

- Rewrote cnn.py so that the logging is more transparent (in plaintext after each k-fold).
- Rerunning the CNN with 500 epochs without dropout. Will be done tomorrow.
- I took out the resizing movement from cnn.py between the 60% and 55% runs. If there is no improvement in the current run I should put it back in:

```
    ##### There is probably a better way of doing this...
    if calm3d.shape[0] != 34:
        startInd = (34 - calm3d.shape[0])/2
        zeroArr0[startInd:calm3d.shape[0]+startInd,:calm3d.shape[1],\
            :calm3d.shape[2]] = calm3d
    if calm3d.shape[1] != 34:
        startInd = (34 - calm3d.shape[1])/2
        zeroArr0[:calm3d.shape[0],startInd:calm3d.shape[1]+startInd,\
            :calm3d.shape[2]] = calm3d
    if calm3d.shape[2] != 34:
        startInd = (34 - calm3d.shape[2])/2
        zeroArr0[:calm3d.shape[0],:calm3d.shape[1],\
            startInd:calm3d.shape[2]+startInd] = calm3d


    if stress3d.shape[0] != 34:
        startInd = (34 - stress3d.shape[0])/2
        zeroArr1[startInd:stress3d.shape[0]+startInd,:stress3d.shape[1],\
            :stress3d.shape[2]] = stress3d
    if stress3d.shape[1] != 34:
        startInd = (34 - stress3d.shape[1])/2
        zeroArr1[:stress3d.shape[0],startInd:stress3d.shape[1]+startInd,\
            :stress3d.shape[2]] = stress3d
    if stress3d.shape[2] != 34:
        startInd = (34 - stress3d.shape[2])/2
        zeroArr1[:stress3d.shape[0],:stress3d.shape[1],\
            startInd:stress3d.shape[2]+startInd] = stress3d
```

- I have updated cnn.py to start saving the trained CNN models.
- It might also be beneficial to start using the real data as a validation set.
- Processing the log files would be better done in an ipynb.

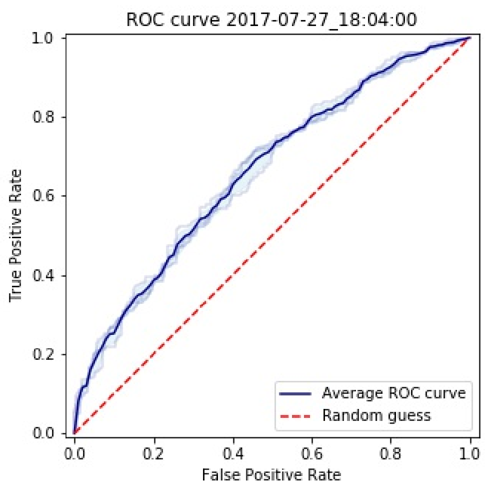
2017-07-27

- Results from last run have an average specificity of 0.54, and an average sensitivity of 0.62. The AUC average is 0.60.
    - 

- This is with

```
    # Neural net (two-channel)

    sess = tf.InteractiveSession()
    tf.reset_default_graph()
    tflearn.initializations.normal()

    # Input layer:
    net = tflearn.layers.core.input_data(shape=[None,34,34,34,2])

    # First layer:
    net = tflearn.layers.conv.conv_3d(net, 32, [10,10,10],
activation="leaky_relu")
    net = tflearn.layers.conv.max_pool_3d(net, [2,2,2], strides=[2,2,2])

    # Second layer:
    net = tflearn.layers.conv.conv_3d(net, 64, [5,5,5],
activation="leaky_relu")
    net = tflearn.layers.conv.max_pool_3d(net, [2,2,2], strides=[2,2,2])

    # Fully connected layers
    net = tflearn.layers.core.fully_connected(net, 2048, regularizer="L2",
weight_decay=0.01, activation="leaky_relu")
    #net = tflearn.layers.core.dropout(net, keep_prob=0.5)

    net = tflearn.layers.core.fully_connected(net, 1024, regularizer="L2",
weight_decay=0.01, activation="leaky_relu")
    #net = tflearn.layers.core.dropout(net, keep_prob=0.5)

    net = tflearn.layers.core.fully_connected(net, 512, regularizer="L2",
weight_decay=0.01, activation="leaky_relu")
    #net = tflearn.layers.core.dropout(net, keep_prob=0.5)

    # Output layer:
    net = tflearn.layers.core.fully_connected(net, 2, activation="softmax")

    net = tflearn.layers.estimator.regression(net, optimizer='adam',
learning_rate=0.000001, loss='categorical_crossentropy')
    model = tflearn.DNN(net, tensorboard_verbose=0)

    # Train the model, leaving out the kfold not being used
    dummyData = np.reshape(np.concatenate(kfoldData[:i] + kfoldData[i+1:],
axis=0), [-1,34,34,34,2])
    dummyLabels = np.reshape(np.concatenate(kfoldLabelsOH[:i] +
kfoldLabelsOH[i+1:], axis=0), [-1, 2])
    model.fit(dummyData, dummyLabels, batch_size=100, n_epoch=500,
show_metric=True)
```

- I am convinced that the CNN is finding something. Will push the new cnn.py to github so that we can test the trained nets on real data.
- New CNN further reduces regularisation, and increases learning rate from 0.000001 to

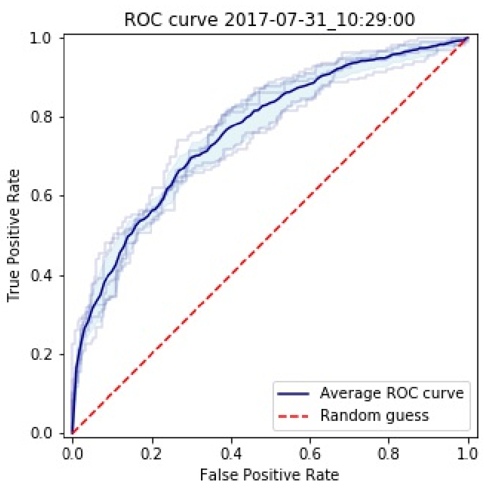
0.0001.
- Writing a python script that finds the part(s) of the cube that the CNN uses for diagnosis.
    - I will need to test it when I have some models, but it looks like it will work. It is saved as `getDiagArea.py`.
- Latest CNN results:
    - AVG spec 0.62, AVG sens 0.61, AVG AUC 0.66. (Over k=3 folds).
    - 

2017-07-31

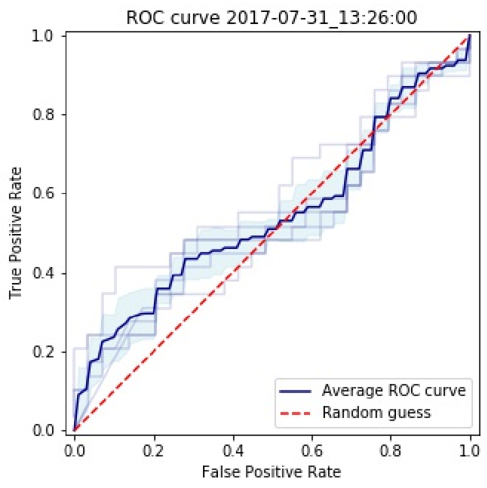
- Newest results are in with the following CNN:

```
    # Neural net (two-channel)

    sess = tf.InteractiveSession()
    tf.reset_default_graph()
    tflearn.initializations.normal()

    # Input layer:
    net = tflearn.layers.core.input_data(shape=[None,34,34,34,2])

    # First layer:
    net = tflearn.layers.conv.conv_3d(net, 32, [10,10,10],
activation="leaky_relu")
    net = tflearn.layers.conv.max_pool_3d(net, [2,2,2], strides=[2,2,2])

    # Second layer:
    net = tflearn.layers.conv.conv_3d(net, 64, [5,5,5],
activation="leaky_relu")
    net = tflearn.layers.conv.max_pool_3d(net, [2,2,2], strides=[2,2,2])

    # Third layer:
    net = tflearn.layers.conv.conv_3d(net, 128, [2,2,2],
activation="leaky_relu") # This was added for CNN 2017-07-28
```

```
    # Fully connected layers
    net = tflearn.layers.core.fully_connected(net, 2048,
activation="leaky_relu") # regularizer="L2", weight_decay=0.01,
    #net = tflearn.layers.core.dropout(net, keep_prob=0.5)

    net = tflearn.layers.core.fully_connected(net, 1024,
activation="leaky_relu") # regularizer="L2", weight_decay=0.01,
    #net = tflearn.layers.core.dropout(net, keep_prob=0.5)

    net = tflearn.layers.core.fully_connected(net, 512,
activation="leaky_relu") # regularizer="L2", weight_decay=0.01,
    #net = tflearn.layers.core.dropout(net, keep_prob=0.5)

    # Output layer:
    net = tflearn.layers.core.fully_connected(net, 2, activation="softmax")

    net = tflearn.layers.estimator.regression(net, optimizer='adam',
learning_rate=0.0001, loss='categorical_crossentropy')
    model = tflearn.DNN(net, tensorboard_verbose=0)

    # Train the model, leaving out the kfold not being used
    dummyData = np.reshape(np.concatenate(kfoldData[:i] + kfoldData[i+1:],
axis=0), [-1,34,34,34,2])
    dummyLabels = np.reshape(np.concatenate(kfoldLabelsOH[:i] +
kfoldLabelsOH[i+1:], axis=0), [-1, 2])
    model.fit(dummyData, dummyLabels, batch_size=100, n_epoch=150,
show_metric=True) # In practice learning stops ~150 epochs.
    dt = str(datetime.datetime.now().replace(second=0,
microsecond=0).isoformat("_"))
    model.save("./models/"+dt+"_3d-2channel-fakedata_"+str(i)+"-of-
"+str(k)+".tflearn")
```

- Avg AUC, spec, sens (over 5 k-folds): 0.762, 0.630, 0.735.
    - 

- Results for the models applied to real data (avg AUC, spec, sens): 0.544, 0.814, 0.207.


ROC curve 2017-07-31_13:26:00
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Average ROC curve
Random guess