COGNITIVE COMPUTING & NEURAL NETWORK

ULTRASOUND NERVE SEGMENTATION

Segmenting images using CNN

Kaggle Competition Link:
https://www.kaggle.com/c/ultrasound-nerve-segmentation#description

Objective:
Segmenting the Brachial Plexus with Deep Learning.
Predict which pixels of an ultrasound image contain the brachial plexus

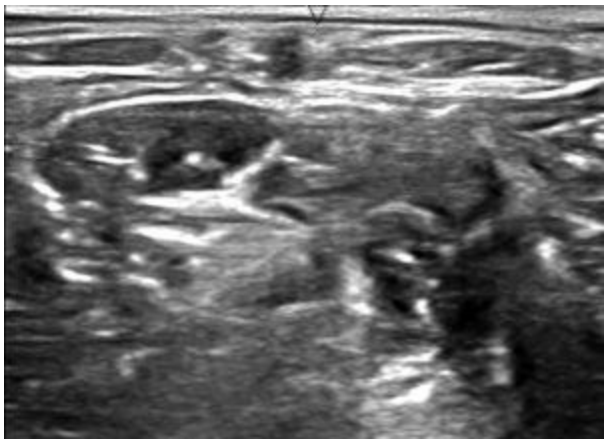
Data:

	TRAIN-X	TRAIN-Y	TEST-X	TEST-Y
# INPUT IMAGES	5635	5635	5508	5508

What to do?

Pixels in each image belonged to one of two possible classes – Nerve, Not-Nerve

So, of the 243,500 pixels in every image of the testing data, the result set must consist of all the pixels that belong to be a part of the Brachial Plexus nerve.



Image

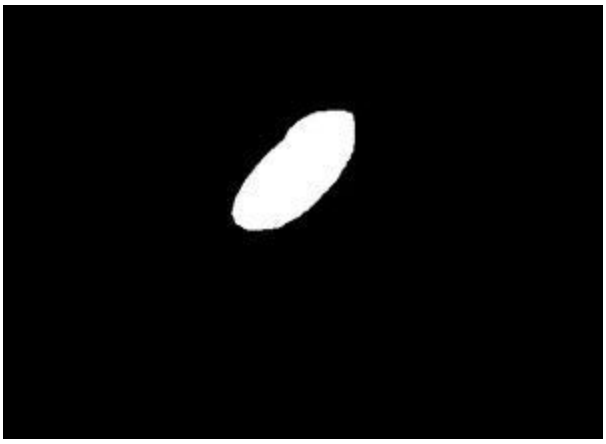


Image Mask

Evaluation

This competition is evaluated on the mean Dice coefficient. The Dice coefficient can be used to compare the pixel-wise agreement between a predicted segmentation and its corresponding ground truth. The formula is given by:

$$\frac{2 * |X \cap Y|}{|X| + |Y|}$$

Where X is the predicted set of pixels and Y is the ground truth. The Dice coefficient is defined to be 1 when both X and Y are empty. The leaderboard score is the mean of the Dice coefficients for each image in the test set.

Understanding Dice coefficient:

For every image i,

If yi=<yi1, yi2, yi3 . . . . . ,yin> is the binary vector identifying the mask for the image i

And $\hat{y}_i$ = <$\hat{y}_{i1}$, $\hat{y}_{i2}$, $\hat{y}_{i3}$,. . ,$\hat{y}_{in}$> be the binary vector predicted

n=420×580=243,600 pixels

$$Dice(y_i, \hat{y}_i) = \frac{2 * y_i \cap \hat{y}_i}{|y_i| + |\hat{y}_i|}$$

$$Score(y, \hat{y}) = \begin{cases} 1 & |y| = |\hat{y}| = 0 \\ Dice(y, \hat{y}) & |y| > 0 \end{cases}$$

```
def dice_coef (y_true, y_pred):
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = K.sum(y_true_f * y_pred_f)
    return (2. * intersection + smooth) / (K.sum(y_true_f) + K.sum(y_pred_f) + smooth)
```

Model:

2

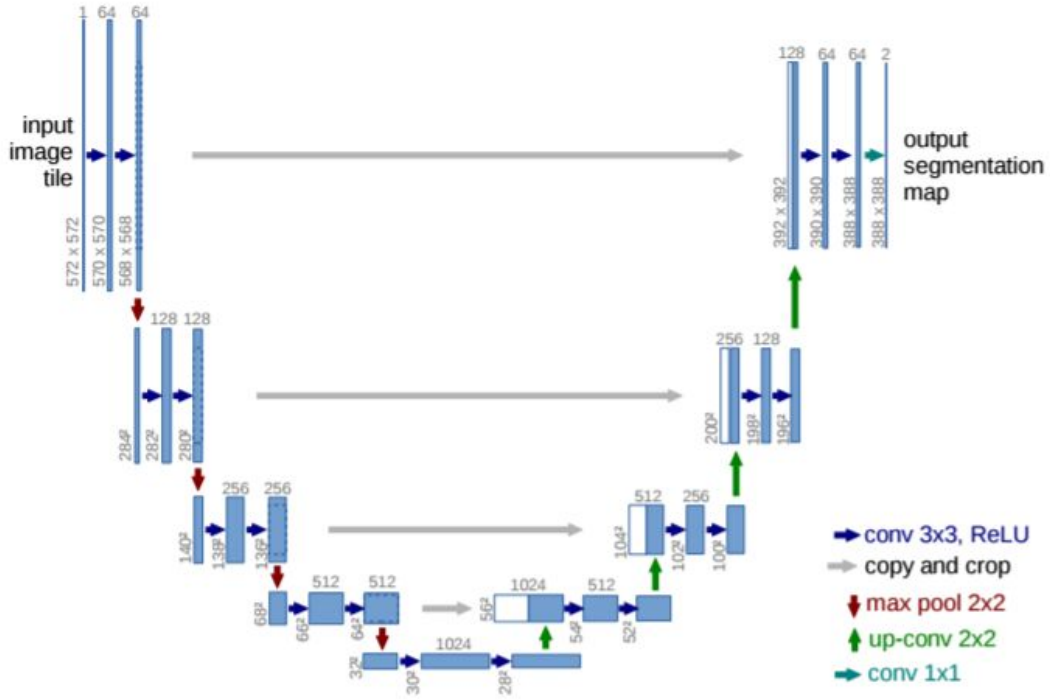


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

The Basic model, was a direct implementation of U-NET as in this paper:
https://arxiv.org/pdf/1505.04597.pdf

The network was built in Keras using Marko Jocic's implementation.

Changes made to the Network but did not seem to be effective:

1. Increased the depth by one more level with 1024 channels
    a. This only increased the training time. There was no significant chnage in the dice coefficients obtained
2. Introduced padding to keep the convolution output size same as the input
    a. Made no difference to the model for 100 epochs run
3. Introduced dropout layers of 25% after convolution layers
4. Following augmentation

```
        rotation_range=5,

        vertical_flip=True,

        horizontal_flip=True,
```

Step 1 : Create the input image arrays by converting the .tif images to ndarray and save them to .npy files to be loaded when required

```
Creating training images...
------------------------------
Done: 0/50 images
Done: 10/50 images
Done: 20/50 images
Loading done.
------------------------------
How the train data looks - This is the X
image name:7
Shape:50,1
------------------------------
[[[  0  98  90 ...,  97  96  94]
  [  0  95  92 ..., 174 173 166]
  [  0 104  93 ..., 178 179 162]
  ...,
  [  0  44  38 ...,  62  62  61]
  [  0  50  36 ...,  62  62  62]
  [  0  52  38 ...,  59  58  59]]]
------------------------------
How the train mask looks - This is Y
image mask name:_
------------------------------
[[[0 0 0 ..., 0 0 0]
  [0 0 0 ..., 0 0 0]
  [0 0 0 ..., 0 0 0]
  ...,
  [0 0 0 ..., 0 0 0]
  [0 0 0 ..., 0 0 0]
  [0 0 0 ..., 0 0 0]]]
Saving to .npy files done.
------------------------------
Creating test images...
------------------------------
Done: 0/50 images
Done: 10/50 images
Done: 20/50 images
Done: 30/50 images
Done: 40/50 images
Loading done.
Saving to .npy files done.
------------------------------
How the test data looks - This is X for Test
Test image ID - This is X - ID
605
------------------------------
[[[  0 158 164 ..., 108 108 109]
  [  0 250 237 ..., 177 182 184]
  [  0 255 221 ..., 225 233 241]
  ...,
  [  0  13   3 ...,  42  41  42]
  [  0  13   2 ...,  44  43  43]
  [  0  13   2 ...,  51  51  51]]]
Out[7]: <matplotlib.image.AxesImage at 0x7eff30316ba8>
```

Step 2: Input data scaling:

All the images are scaled down to 64*80 before inputting to the network to reduce the memory requirement and fasten the training process.
The generated masks for test images need to be scaled up to the original size,

Step 3: Define Evaluation Metrics :

```
def dice_coef(y_true, y_pred):
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = K.sum(y_true_f * y_pred_f)
    return (2. * intersection + smooth) / (K.sum(y_true_f) + K.sum(y_pred_f) + smooth)

def dice_coef_loss(y_true, y_pred):
    return -dice_coef(y_true, y_pred)
```

Model Selection:

Model 1 : Neural Nets using MLP

How well can a model perform when an image recognition problem is fed to only dense layers without convolutions?

Layer (type)	Output Shape	Param #	Connected to
dense_4 (Dense)	(None, 5120)	26219520	dense_input_2[0][0]
dropout_3 (Dropout)	(None, 5120)	0	dense_4[0][0]
dense_5 (Dense)	(None, 512)	2621952	dropout_3[0][0]
activation_2 (Activation)	(None, 512)	0	dense_5[0][0]
dropout_4 (Dropout)	(None, 512)	0	activation_2[0][0]
dense_6 (Dense)	(None, 5120)	2626560	dropout_4[0][0]

Total params: 31468032

Model 2: U-Net with a single contracting and expanding level

This was done to understand the importance of additional convolutions and depth of the network. Keeping all the other hyper parameter consistent, the model consisted of one single layer contraction and expansion.

```
------------------------------
Creating and compiling model...
------------------------------
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
input_3 (InputLayer)             (None, 1, 64, 80)     0
____________________________________________________________________________________________________
dropout_3 (Dropout)              (None, 1, 64, 80)     0           input_3[0][0]
____________________________________________________________________________________________________
convolution2d_15 (Convolution2D) (None, 32, 64, 80)    320         dropout_3[0][0]
____________________________________________________________________________________________________
convolution2d_16 (Convolution2D) (None, 32, 64, 80)    9248        convolution2d_15[0][0]
____________________________________________________________________________________________________
maxpooling2d_5 (MaxPooling2D)    (None, 32, 32, 40)    0           convolution2d_16[0][0]
____________________________________________________________________________________________________
convolution2d_17 (Convolution2D) (None, 64, 32, 40)    18496       maxpooling2d_5[0][0]
____________________________________________________________________________________________________
convolution2d_18 (Convolution2D) (None, 64, 32, 40)    36928       convolution2d_17[0][0]
____________________________________________________________________________________________________
upsampling2d_3 (UpSampling2D)    (None, 64, 64, 80)    0           convolution2d_18[0][0]
____________________________________________________________________________________________________
merge_3 (Merge)                  (None, 96, 64, 80)    0           upsampling2d_3[0][0]
                                                                   convolution2d_16[0][0]
____________________________________________________________________________________________________
convolution2d_19 (Convolution2D) (None, 32, 64, 80)    27680       merge_3[0][0]
____________________________________________________________________________________________________
convolution2d_20 (Convolution2D) (None, 32, 64, 80)    9248        convolution2d_19[0][0]
____________________________________________________________________________________________________
convolution2d_21 (Convolution2D) (None, 1, 64, 80)     33          convolution2d_20[0][0]
====================================================================================================
Total params: 101953
```

Compiled with the chosen hyper parameter

```
model.compile(optimizer=Adam(lr=1e-5), loss=dice_coef_loss, metrics=[dice_coef])
```

Model 3: U-Net

Network Parameter chosen:

```
model.compile(optimizer=Adam(lr=1e-5), loss=dice_coef_loss, metrics=[dice_coef])
```

Network Parameters tried:

```
#sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
#model.compile(optimizer=sgd, loss='binary_crossentropy')
#rmsprop = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08)
#model.compile(optimizer=rmsprop, loss='binary_crossentropy')
#loss='binary_crossentropy'
loss=[dice_coef_loss_batchwise, 'binary_crossentropy']
metrics={'outmap': dice_coef, 'outbin': 'accuracy'}
model.compile(optimizer='adam',
              loss=loss,
              loss_weights=[1., 0.01],
              metrics=metrics)

print_summary(model.layers)
```

4. Model : U-Net with Image Augmentation

In order to make the most of our few training examples, we will "augment" them via a number of random transformations, so that our model would never see twice the exact same picture. This helps prevent overfitting and helps the model generalize better.

In Keras this can be done via the keras.preprocessing.image.ImageDataGenerator class. This class allows you to:

- Configure random transformations and normalization operations to be done on your image data during training
- instantiate generators of augmented image batches (and their labels) via .flow(data, labels) or .flow_from_directory(directory)

These generators can then be used with the Keras model methods that accept data generators as inputs, fit_generator, evaluate_generator and predict_generator.

```
print('-'*30)
print('Fitting model...')
print('-'*30)
#model.fit(imgs_train, imgs_mask_train, batch_size=32, nb_epoch=1, verbose=1, shuffle=True,
 #         callbacks=[model_checkpoint,csv_logger], validation_split=0.25)
datagen = ImageDataGenerator(
            rotation_range=5,
            vertical_flip=True,
            horizontal_flip=True,
            )
model.fit_generator(datagen.flow(imgs_train, imgs_mask_train, batch_size=32, shuffle=True),
           samples_per_epoch=len(imgs_train),nb_epoch=100, verbose=1,callbacks=[model_checkpoint,csv_logger])
```

Data Preprocessing:

Before using the compiled model to predict for data, some basic data processing is done to make sure we get accurate results.

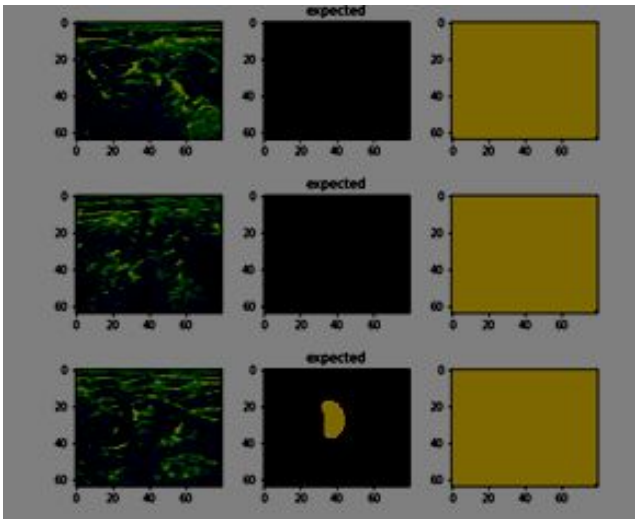
Zero-centering and Standardizing:

The first and simple pre-processing approach is zero-center the data, and then normalize them, which is presented as two lines codes as follows:

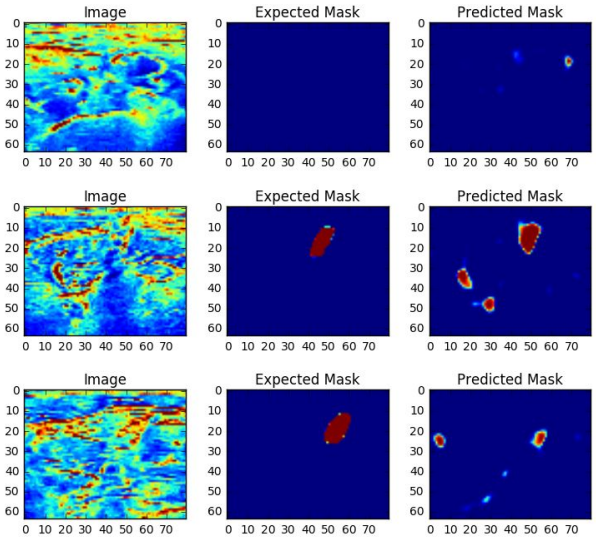
```
imgs_train = imgs_train.astype('float32')
mean = np.mean(imgs_train)  # mean for data centering
std = np.std(imgs_train)  # std for data normalizati
print("Mean of images:",mean)
print("Std Deviation of images:",std)

imgs_train -= mean
imgs_train /= std

imgs_mask_train = imgs_mask_train.astype('float32')
imgs_mask_train /= 255.  # scale masks to [0, 1]
print(imgs_mask_train[1,0])
```

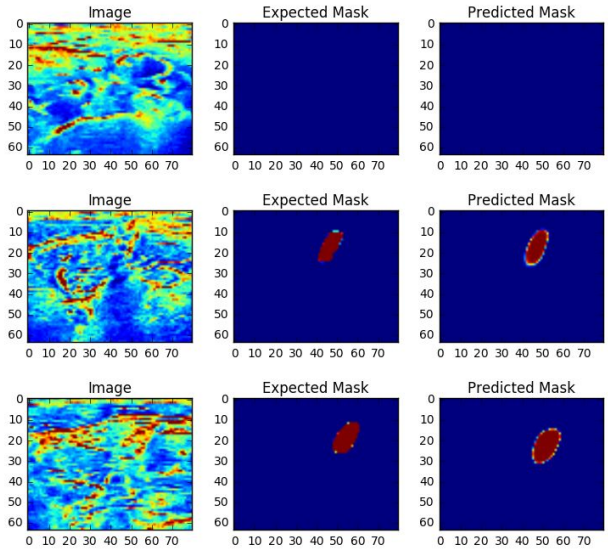
Results: These are the expected and the predicted masks on seen images - Images taken from the training set

Regular Neural Nets : Network using only MLPs & one hidden layer.

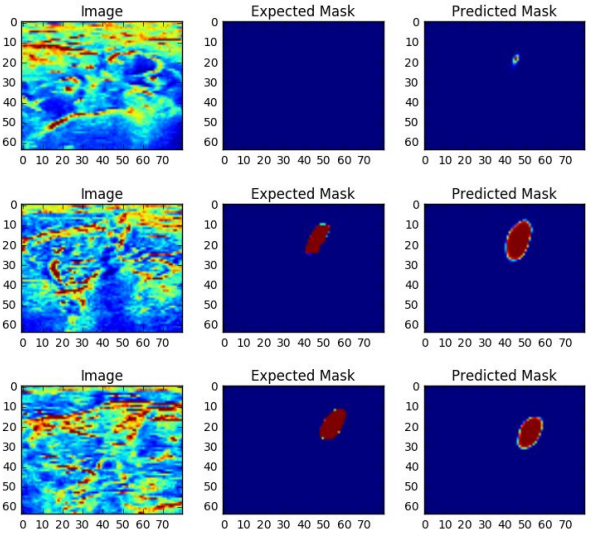


Model 1: Modified U-Net for a single Level of contraction & expansion





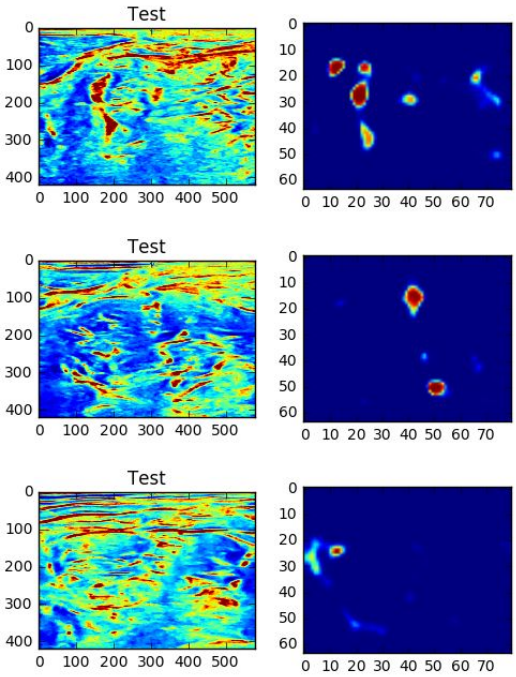
Model 3 - U-Net without Augmentation

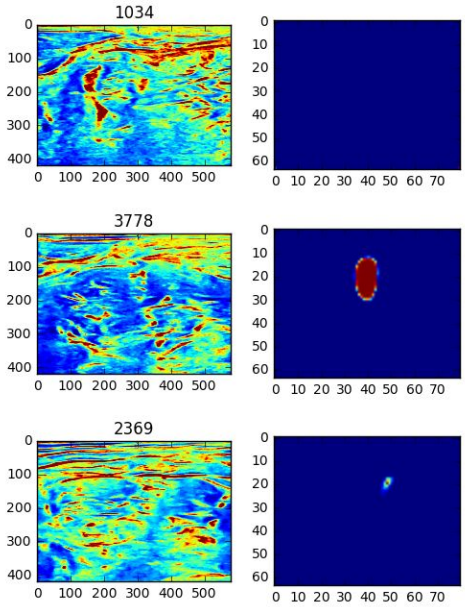
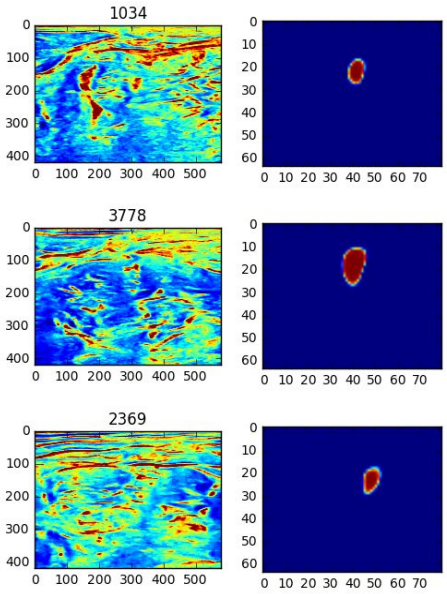


Model 4 - U-Net with Augmentation

Results: These are the expected and the predicted masks on seen images - Images taken from the training set

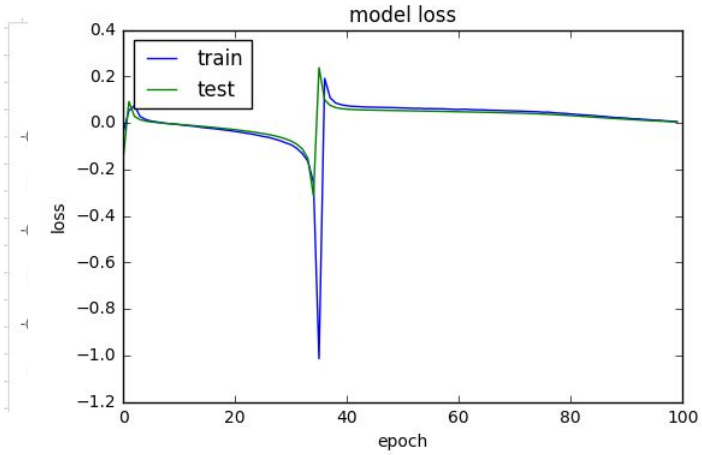
Model 1 - Single layer U-Net

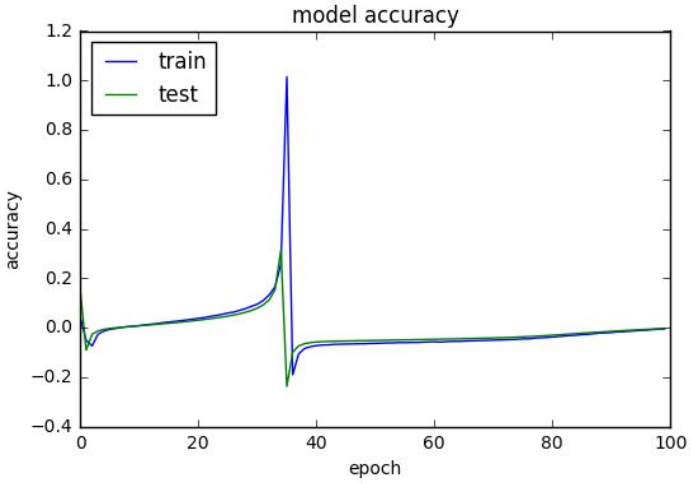


Without Augmentation	With Augmentation
	

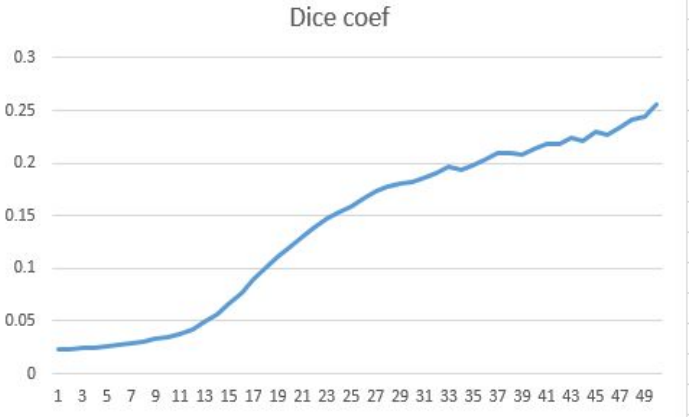
Model History:

Model 1 - Regular Neural Network with MLPs

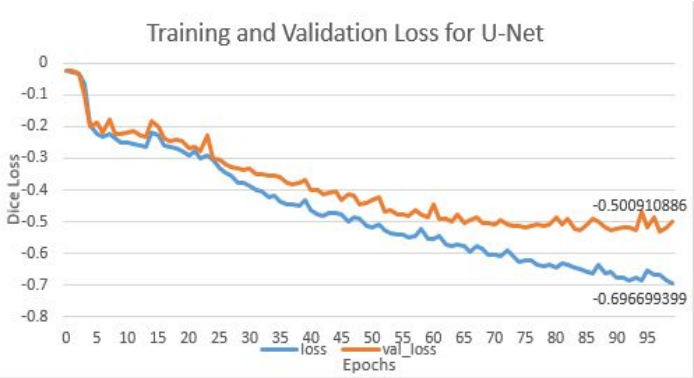


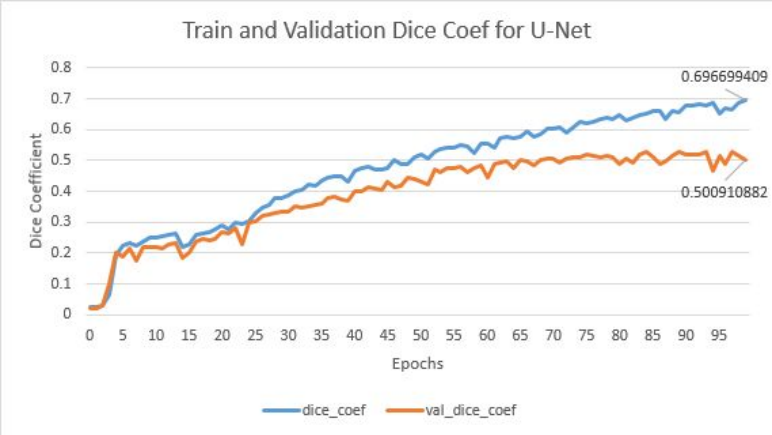


Model 2 - U-Net with one level (Train)

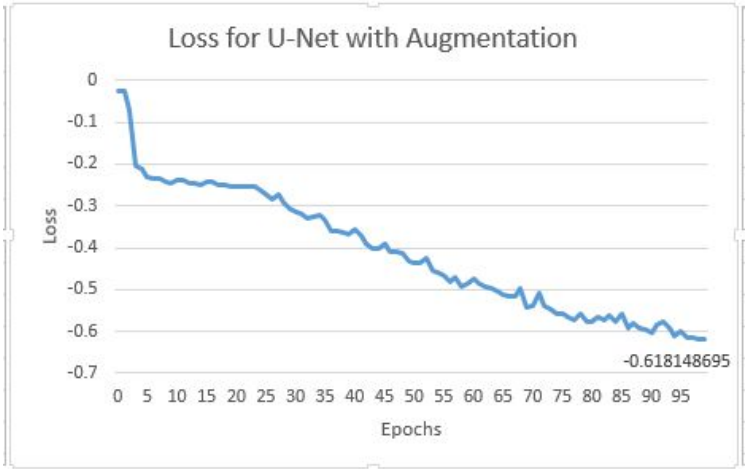


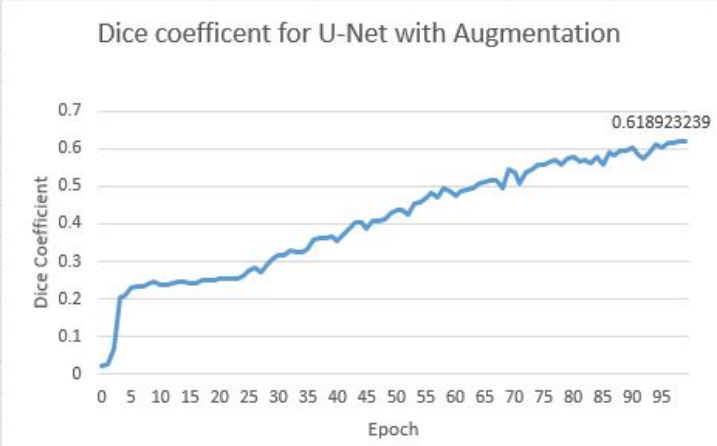
Model 3 - U-Net





Model 4 - U-Net with Augmentation





Conclusion: Comparing Models:

- **X_train = 1000 Training images**
- **Y_train = 1000 Training images, with mask value**
- **X_test = 200 test images**

Output of the predict() function, returning a mask for each image is also a scaled image saved as n-d array in a .npy file

The 'Train Dice Coefficient' and 'Dice Coef Loss' is the value after 100 epochs for each model

Model	Avg. Time Per Epoch	Train Dice Coefficient	Dice Coef Loss	Input Type	Input Size
MLP	5.8 s	1.01	-1.01	1-D Array	5120
Single Layer U-Net	271.6 s	0.4652	-0.4652	2-D Array	64x80
U-Net	498.3 s	0.6967	-0.6967	2-D Array	64x80
U-Net with augmentation	600.8 s	0.6189	-0.6189	2-D Array	64x80