Data Science 2: Advanced Topics in Data Science

Final Report - Group 7 - Cancer Diagnosis in Medical Imaging

Harvard University
Spring 2019
Instructors: Pavlos Protopapas and Mark Glickman
Group 7 Team Members:

- Mark Carlebach
- Saurabh Rajratna Kulkarni
- Elliot Trilling
- Kwun Lok Yu

I. INTRODUCTION

We were challenged in this project to consider implementations of Deep Learning that would aid radiologists performing cancer diagnoses while avoiding the "black box" problem. That is, we were asked to answer the question: how could Deep Learning provide radiologists insight when making cancer diagonoses rather than just final conclusions?

After reading the initial suggested literature, we decided to evaluate **Semantic Image Segmentation** as a potential answer to this challenge:

- Semantic Image Segmentation classifies each pixel in an image (from potentially many classes) rather than classifying the entire image.
- Given a radiological image and proper training, Semantic Image Segmentation could classify which pixels in the image are concerning and worthy of additional attention by the medical team.
- While Semantic Image Segmentation would not classify the image as "cancer or not cancer", the Semantic Image Segmentation overlay with pixel level classifictions would be Deep Learning insight to aid the radiologist performing cancer diagnoses.

Data:

A real challenge for researchers in this space is to obtain training data. For Semantic Image Segmentation, the training data needs to include both the original medical image along with indications by a radiologist of which portion of the image (i.e., which pixels) are conerning and worthy of highlight by the Deep Learning algorithm. Harvard's teaching staff provided three potential sources of such data:

- DREAM Mamography Dataset
- DDSM Lung Nodule Dataset
- Lung Image Database Consortium (LIDC-IDRI) at The Cancer Imaging Archive

While the first two sources did not appear fruitful, LIDC-IDRI proved to be a source with adaquate access and support. Details about the LIDC-IDIR data and our usage are in the other sections of this report.

Modeling:

As for CNN architectures, Harvard's teaching staff pointed us toward both U-Net and ResNet as reference models with relevance to our problem:

- U-Net is the most prominent architecture developed for fast and accurate Semantic Image Segmentation, especially with a limited size training set.[1] U-Net's main features are the downsampling and upsampling paths (i.e., the "U-shape") with connections from the feature-rich downampling path to the upsampling path, providing context and information to aid in reconstruction of the full resolution image.
- ResNet is a more recent advance than U-Net that enables greater accuracy (on both test and train data) with very deep CNNs, where traditionally extra depth has been associated with both over-fitting and worse performance even during training.[2] The main feature or ResNet is the "skip connection" between layers. With the skip connection, fitting occurs in the skipped level of a residual function which a) appears easier for the neural net to solve and b) from which the original function can be reconstructed.

The highbar for our investigation was to explore Semantic Image Segmentation of the LIDC-IDRI data leveraging features of both U-Net and ResNet. Our progress toward this goal are outlined below.

IIa. DATA: LIDC-IDRI

The data used in this project was obtained from The Cancer Imaging Archive.[1] In particular, we used the LIDC-IDRI data which consists of diagnostic and lung cancer screening thoracic computed tomography (CT) scans with marked-up annotated lesions.

The database includes images and image annotations performed by radiologists for 1,010 patients. For each patient there are one or more imaging studies (i.e., images taken at different times throughout the course of medical treatment). Each study for each patient includes a) a series of images (e.g, slices from a CT scan) stored in medical-industry standard DICOM format (i.e., .dcm files) and b) an associated .xml file containing study-specific tags that encode image annotations provided by a set of radiologists at the time the study was conducted. In our setting of Semantic Image Segmentation, the .dcm files serve as the feature variables (i.e., "X") and the annotations in the .xml files serve as the response variable (i.e., "Y").

As is shown in the data pre-processing section below, converting the image data in .dcm files to an appropriate representation for Deep Learning training is relatively straightforward (as many tools exist in Python to perform transformation from .dcm format to numpy arrays).

Converting the .xml files with annotation information which needs to be converted into pixel level classifications is more challenging. As is suggested by readings at The Cancer Imaging Archive website, most projects analyzing the LIDC-IDRI data have taken on this challenge individually.[2] That is, there is no obvious, universal solution for converting the .xml files into data suitable for Deep Learning traing and testing. In this project, we also developed our own solution to this problem. This solution is described in detail in the data pre-processing section below.

Here is an overview of the information contained in the .xml files:[3]

- For an image study (often consisting of a series of image slices) for a patient, a single .xml file is included iff radiologists identified in the image(s) a nodule (of any size) or a non-nodule $\geq$ 3mm in diameter.
- For nodules with diameters $\geq$ 3mm, the .xml contains tags (i.e., edgemap, xCoord, yCoord) to specify the outline of the nodule identified by radiologists. The outline can be either a) inclusionary to indicate the region inside the outline is a nodule or b) exclusionary to indicate the region inside the outline is not a nodule. These outlines appear on each slice of an image series in which the nodule is present and not on slices where the image is not present.
- For nodules with diameters < 3mm, the .xml contains tags (i.e., edgemap, xCoord, yCoord) to specify a single point corresponding to the nodule's centroid but not an outline.
- For non-nodules with diameters $\geq$ 3mm, the .xml contains tas (e.g., locus, xCoord, yCoord) to specify a single point corresponding to the non-nodule's centroid but not an outline.

For nodules with diameters $\geq$ 3mm, the .xml also contains additional characteristics assessed by the radiologists (on a scale of 1-5 for reach characteristic):

- Subtlety
- Internal structure
- Calcification
- Sphericity
- Margin
- Spiculation
- Texture
- Malignancy

Each set of images was examined independently by 4 radiologists. The .xml for a single set of images contains the above annotations separately for each radiologist that noted an abnormality. That is, each .xml contains up to 4 "sources of truth" to classify the pixels in the associated set of images.

IIb. DATA: LIMITING SCOPE

The Cancer Imaging Archive website provides additional information about the original study, including a list of patients for whom a malignancy was at some point identified.[1] (For sake of clarity, this classification of the patient's cancer diagnosis is not otherwise idicated in the source images and .xml annotations.) Of the 1,010 patients in the study, diagnoses were provided for 157 patients.

Data Limitations:

Given the constraints of this project, our analysis was of images from these 157 patients which we downloaded from The Cancer Imaging Archcive. Our thought process was that this subset of data would provide many thoracic images clear of abnormalities (i.e., the slices from each patient for which no annotations existed) and many thoracic images with abnormalities (i.e., the slices from each patient for which abnormalities were identified), presumably in higher number than in patients for which a diagnosis was performed).

XML Limitations:

In terms of the .xml conversion described below, we did choose to limit our scope in the following manner for this proof of concept. Given our results, enhancing the pre-preprocessing to include greater complexity from the .xml seems warranted:

- For each .xml, we only generated an overlay with pixel classifications for the first radiologist's readings. Additional overlays with different pixel classifications could be generated from each .xml. In some sense, the multiple readings for each image study by independent radiological experts provide built-in image augmentation with "jitter" associated with each reader's slightly different reading of the same image.
- For each .xml, we only generated pixel classifications based on the the inclusionary outlines for the nodules with diameters $\geq$ 3mm. We did not generate classifications based on the exclusionary outlines, nor did we include pixel classifcation for the centroids of the other abnormalities.

- Lastly, we only included pixel classifications for two classes: "inside an inclusionary outline" or not. We did not create additional classes from the 8 categories (e.g., subtlety, internal structure, etc) that were rated 1-5 by the radiologists. Adding this additional complexity also seems doable with additional time and resources.

III. DATA PRE-PROCESSING

We performed the data pre-processing using three notebooks that are described below:

1. Raw to Processed

See `raw2processed.ipynb`

What file structure is assumed?

- raw data is stored in a root folder (as downloaded from TCIA website) that contains a number of subdirectories, each of which represents the data for a certain patient.
    - each patient directory contains one or more subdirectories, each of which represents a different "study" (collection of scans).
        - each study directory contains a *single* series directory (with a few exceptions) that contains a number of DICOM images as well as an XML file (with a few exceptions) with ROI (region of interest) data.

What steps were needed?

- for each series directory:
    - extract from the XML file the first "readingSession" (data recorded by a single radiologist) tag.
        - find all "unblindedReadNodule" tags (a single nodule) in the given "readingSession".
            - for each "unblindedReadNodule" tag, find all "roi" (region of interest) tags.
                - for each "roi" tag, extract the ID of the corresponding DICOM image on which the given "roi" was marked.
                - for each "roi" tag, also extract a list of x, y coords that outline a nodule on the previously determined DICOM image.
    - for each DICOM image in the series directory:
        - get the image ID and see if it matches anything found in the XML file. If it does, first save the original (unedited) image as a numpy array to a specific subriectory under the root directory "processed", then set each pixel listed in the ROI to some unique (not used elsewhere) arbitrary value. Note: a single image can have multiple ROI's
        - save the DICOM image as a numpy array into one of two subriectories based on whether or not an ROI was present.

Extra steps (completed after the initial extraction - to simplify code / explanation)

- for each processed image, fill the space inside of the ROI to some unique (not used elsewhere) arbitrary value.

2. Processed to Compiled

See `Processed2Compiled.ipynb`

1. read all the input images that have an output image with a marked ROI into an array
2. read the same number of input images that do NOT have and output image with a marked ROI into another array
    - note: we have many more of this type of image. We use only the same number to keep our data balanced.
3. concatenate, then seporate these arrays into input data and output data (some output data has ROI's marked, some does not)
4. save these arrays

3. Compiled to Training

See `Compiled2Training.ipynb`

1. resize and normalize the input images.
2. convert the output images into binary masks. Pixels inside the ROI are set to 1, everything else is set to 0.
3. resize the output images.
4. split the data into training and testing sets.
5. augment the training data
6. save all numpy arrays

IV. INITIAL U-NET IMPLEMENTATION

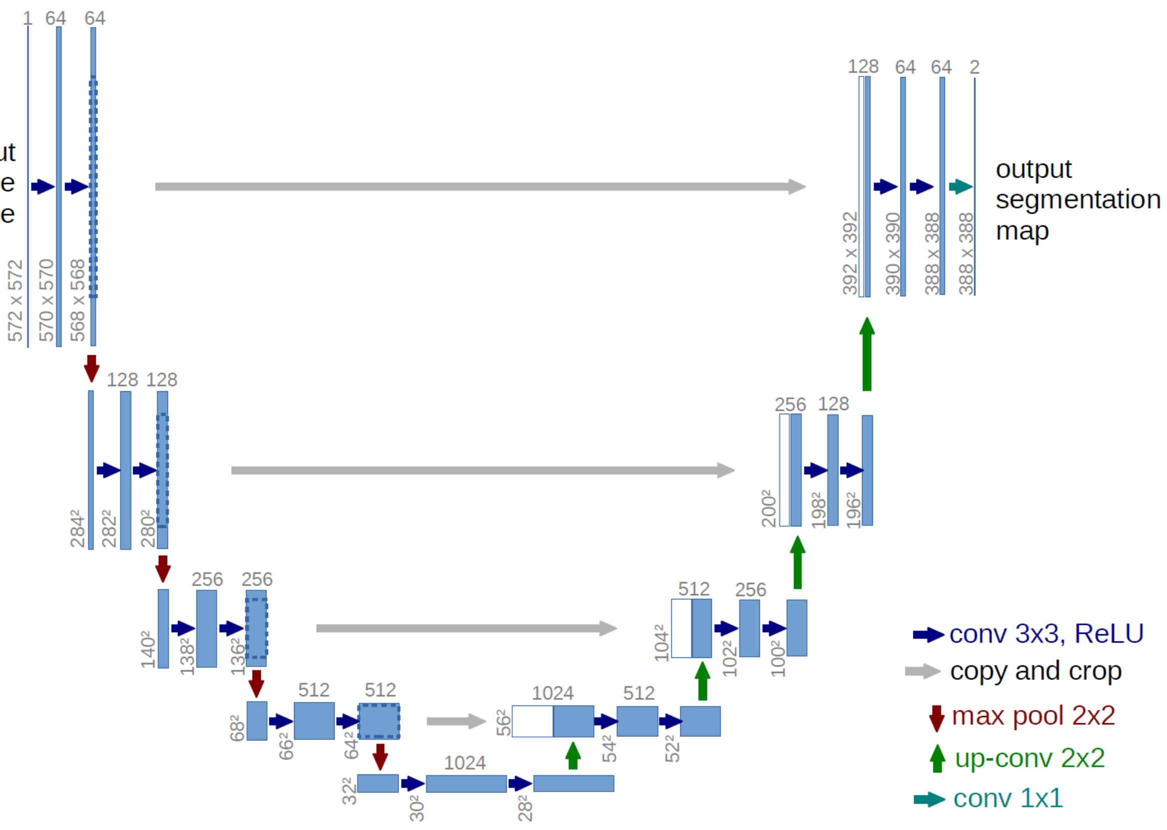
See `UNetImplementation.ipynb`

Motivation (very similar to ResNet)

Deep networks are necessary when training a model on highly complex data. Shorter networks simply don't have enough parameters/layers to capture all of the intricacies of some data sets. Unfortunately, during backpropagation in deep networks, the chain rule tells us to multiply many successive gradients together which frequently leads to either exploding or vanishing gradients.

The idea for the U-Net architecture originated out of the need for deep yet stable autoencoder style model. The one key feature of the U-Net architecture is that it passes intermediate outputs on the encoding path directly to the corresponding part of the decoding path via concatenating the images together on the channels axis (this is represented using grey arrows on the image below). This gives backpropagation an alternate shortcut path to very early layers.

Figure 1: U-Net Architeture[1]



Note: This strategy is very similar in principle to the "skip" connections in ResNet (the model architecture we ended up using).

Why didn't this model work well?

We believe the ultimate problem with our U-Net is that it simply isn't deep/complex enough to learn all the intricacies of our data. While the predicted results show that it is clearly moving in the right direction, it simply can't generalize to all the complexities of this data like a deeper model could. So, why not just make a deeper U-Net style architecture? U-Net is just too memory intensive. The addition of the concatenation connections means that if we wanted to make a sufficiently deep model, the tensors would just end up too big to fit in memory.

The ResNet stile architecture's get around this problem by simply adding (rather than concatenating) intermediate results together. This allows it to be much deeper without using too much memory.

V. RES-NET IMPLEMENTATION

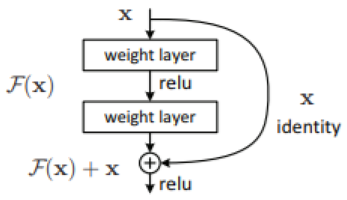
See `MainResNet.ipynb`

A. Motivation & Overview

The idea of Residual Networks originated from the need to have very deep networks. The depth of the networks is very crucial when trying to train your model on complex images and helps in finding challenging patterns. However, as we multiply by the weights on each step during back propagation the gradients can quickly approach to zero, thereby making learning very difficult.

As a solution, the idea of "skip connection" or "shortcut" was introduced - which is basically to shorten the path during back propagation while still having the same number of layer as in a deep network. This allows us to back propagate the gradients directly to earlier layers and to keep the model performance is at least as good as its shallow counterparts.

Figure 1: Skip Connection


x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

As we can see above, we add x to the output of F(x) by using x_identity. Note that, the shape of x and x_identity has to be exactly the same for them add together since they are two matrices.

B. Model Architecture

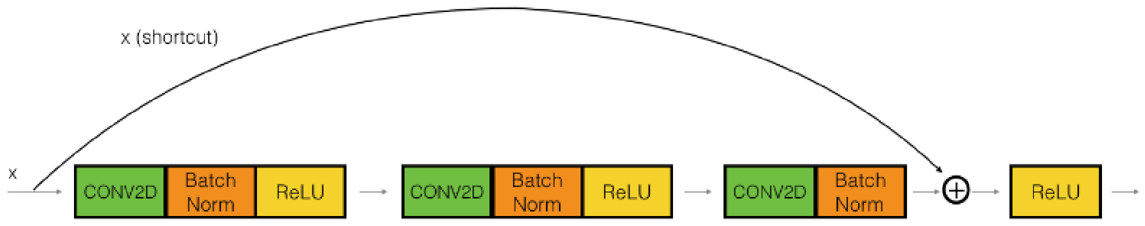
There are two main building blocks that go into the final ResNet that we built for this project:

1. An identity block and
2. a convolution block

Note: The code for these two components was taken from the GitHub repository found in [1] (references)

1. Identity Block:

Figure 2: Identity Block[1]



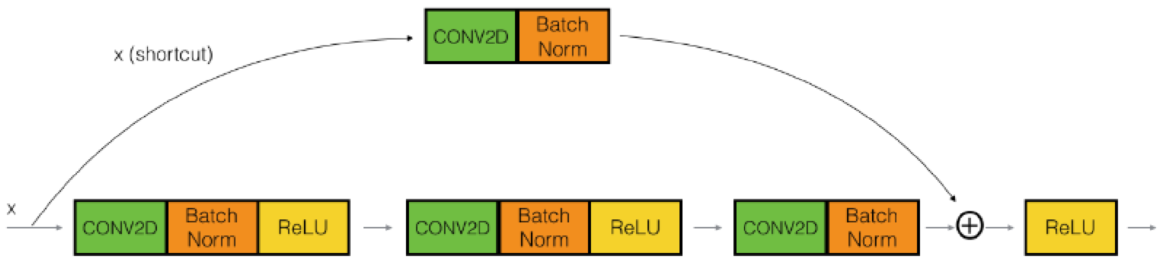
The identity block is introduced to add more depth to the model architecture without changing the input and output activations shapes of the block.

Here is the breakdown of different components within the block:

- Stage 1:
  - CONV2D: Filters = F1, shape = (1,1), padding = valid, stride = (1,1), kernel_initializer = 'glorot_uniform' ;
  - BatchNormalization
  - 'Relu' activation
- Stage 2:
  - CONV2D: Filters = F2, shape = (f,f), padding = same, stride = (1,1), kernel_initializer = 'glorot_uniform' ;
  - BatchNormalization
  - 'Relu' activation
- Stage 3:
  - CONV2D: Filters = F3, shape = (1,1), padding = valid, stride = (1,1), kernel_initializer = 'glorot_uniform' ;
  - BatchNormalization
- Final Step:
  - Concatenate X and X_shortcut (here X_shorcut = input to the block);
  - 'Relu' activation

2. Convolution Block:

Figure 3: Convolution Block[1]



The convolution block is introduced to add more depth to the model architecture when the input and output activations have different shapes. To accomplish this, we add CONV2D and BatchNorm to X_shortcut.

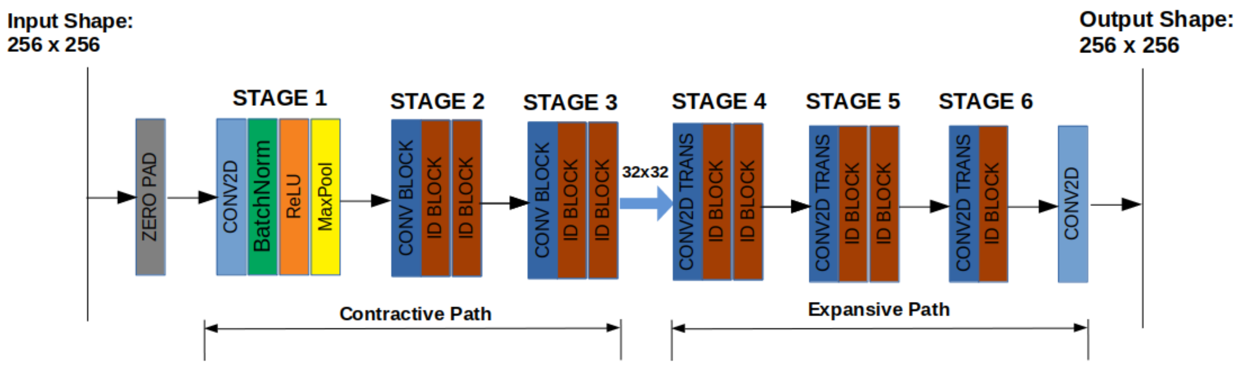
Here is the breakdown of different components within the block:

- Stage 1:
  - CONV2D: Filters = F1, shape = (1,1), padding = valid, stride = (s,s), kernel_initializer = 'glorot_uniform' ;
  - BatchNormalization
  - 'Relu' activation
- Stage 2:
  - CONV2D: Filters = F2, shape = (f,f), padding = same, stride = (1,1), kernel_initializer = 'glorot_uniform' ;
  - BatchNormalization
  - 'Relu' activation
- Stage 3:
  - CONV2D: Filters = F3, shape = (1,1), padding = valid, stride = (1,1), kernel_initializer = 'glorot_uniform' ;
  - BatchNormalization
- X_shortcut:
  - CONV2D: Filters = F3, shape = (1,1), padding = valid, stride = (s,s), kernel_initializer = 'glorot_uniform' ;
  - BatchNormalization
- Final Step:
  - Add() X and X_shortcut;
  - 'Relu' activation

3. Building ResNet:

At this stage, we have the building blocks of our network and we build our final network using architecture shown below:

Figure 4: Final ResNet Model



The first three stages are contractive, where we apply a combination of Conv2D and MaxPool until the shape is 32 x 32 and then expand it back to 256 x 256 by using Conv2DTranspose. The whole architecture is a near resemblance of a variational auto-encoder.

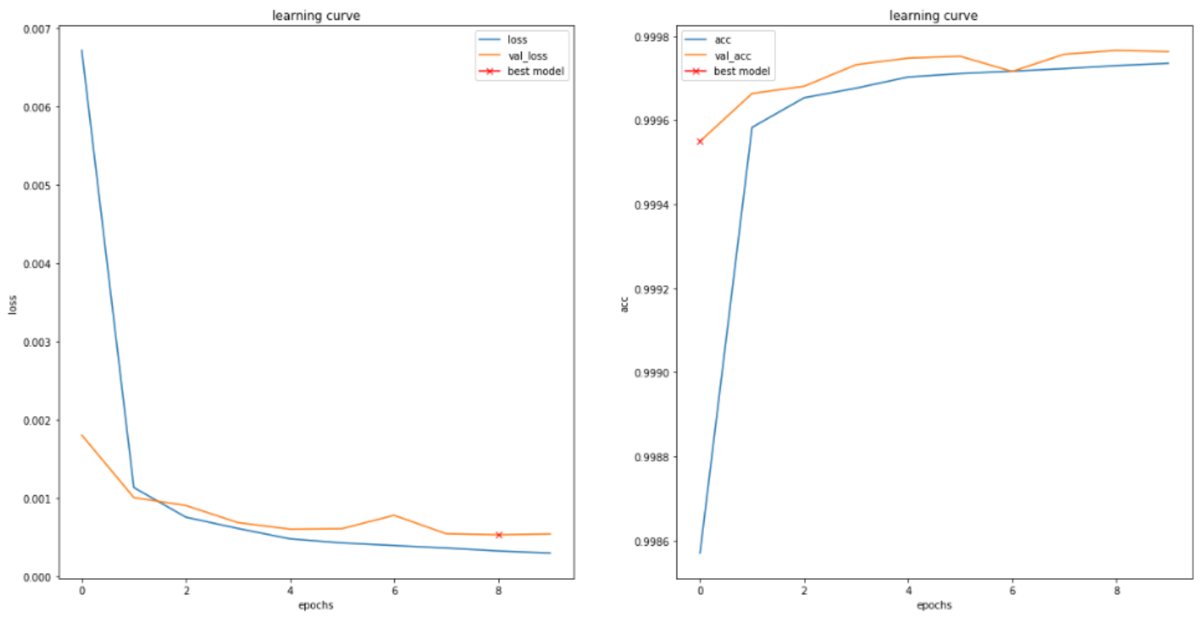
The details are given below:

- Zero Padding: shape = (3,3)
- Stage 1:
  - CONV2D: Filters = 64, shape = (3,3), padding = valid, stride = (1,1), kernel_initializer = 'glorot_uniform' ;
  - BatchNormalization
  - 'Relu' activation
  - MaxPool
- Stage 2:
  - The convolutional block: [F1,F2,F3] = [64,64,256], "f" = 3, "s" = 1.
  - The 2 identity blocks: [F1,F2,F3] = [64,64,256], "f" = 3.
- Stage 3:
  - The convolutional block: [F1,F2,F3] = [128,128,512], "f" = 3, "s" = 1.
  - The 2 identity blocks: [F1,F2,F3] = [128,128,512], "f" = 3.
- Stage 4:
  - The Conv2DTranspose: Filters = 512, shape = (3,3), padding = same, stride = (2,2), kernel_initializer = 'glorot_uniform'
  - The 2 identity blocks: [F1,F2,F3] = [128,128,512], "f" = 3
- Stage 5:
  - The Conv2DTranspose: Filters = 256, shape = (3,3), padding = same, stride = (2,2), kernel_initializer = 'glorot_uniform'
  - The 2 identity blocks: [F1,F2,F3] = [64,64,256], "f" = 3
- Stage 6:
  - The Conv2DTranspose: Filters = 128, shape = (3,3), padding = same, stride = (2,2), kernel_initializer = 'glorot_uniform'
  - The 2 identity blocks: [F1,F2,F3] = [32, 32, 128], "f" = 3.
- Output:
  - Conv2D: Filter = 1 shape = (1,1), activation = sigmoid

C. Model Training

For model training, we used `adam` optimizer with `learning rate` of 0.0002 and `binary_crossentropy` loss with a batch size of 16. With these settings, we get the following learning curve:

Figure 5: Learning Curve



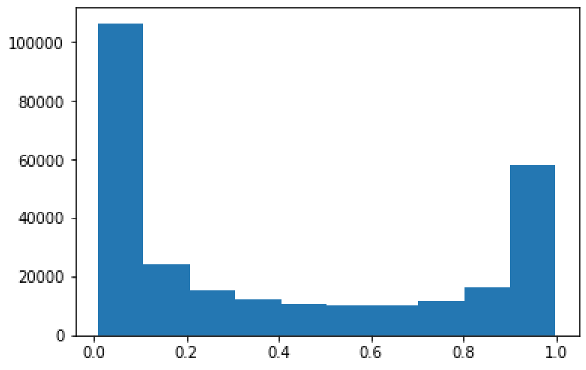
We didn't find training for more than 10 epochs to provide much value as we can see the learning curves do plateau at around 3-5 epochs. This could also be due to the fact that we are using a smaller batch size.

VI. RESULTS

See `MainResNet.ipynb`

One of the major challenges of measuring or interpreting the accuracy of our model numerically is that majority of our true pixels have a value of 0 and hence majority of the predicted pixels also have value very close to zero (see figure below). This might create a false noise in the final output and can take away the attention of the radiologist from the actual nodule.

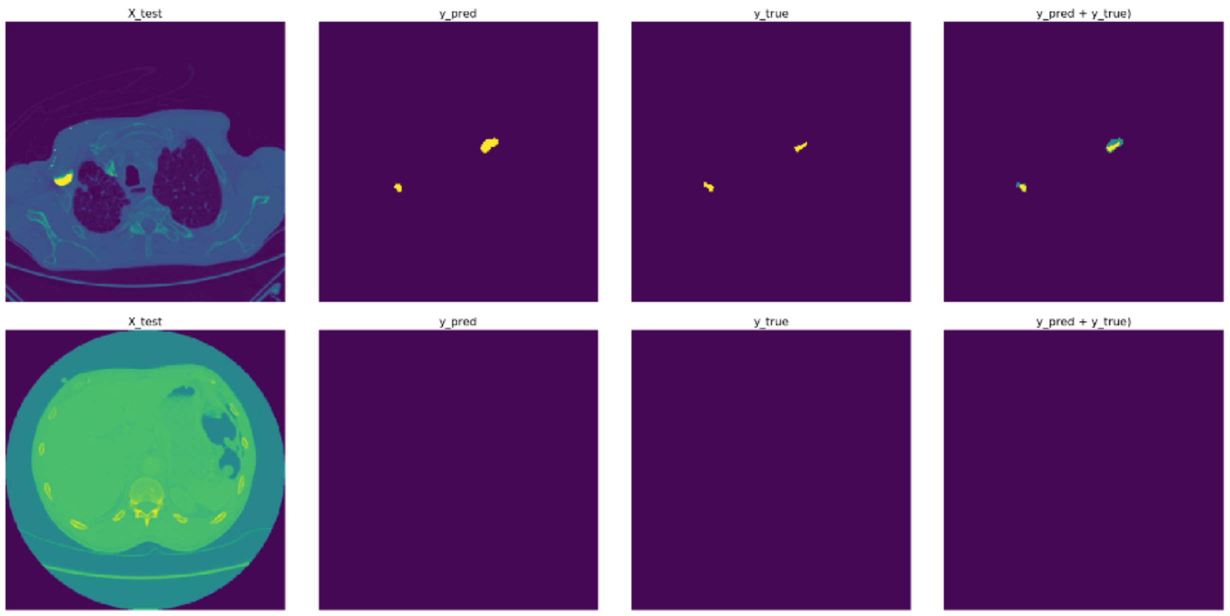
Figure 6: Distribution for Predicted Pixels



To avoid this problem, we convert the true and the predicted response values to binary by setting a particular threshold, above which the value is set to 1 and all values below that are set to 0. To find the threshold, we first define a range of values for y_pred and y_true and then search through that grid to find the best pixel-by-pixel miss-classification rate.

In effect we get y_pred and y_true as shown below for visual comparison (the last cell is an overlay of y_pred and y_true):

Figure 7: Sample Output



We know that the accuracy among the pixels that are truly 0's is going to be extremely high - what we are interested in is the accuracy among the pixels that are true 1's or miss-classified as 1's as these will tell us how good the model is in identifying true nodules and when does it confuses a non-nodule to actually be a nodule. For doing this, we ignore all the true positive where the true response is 0 and plot a confusion matrix for train and test sets with the remaining data points.

Figure 8: Confusion Matrix (Train)

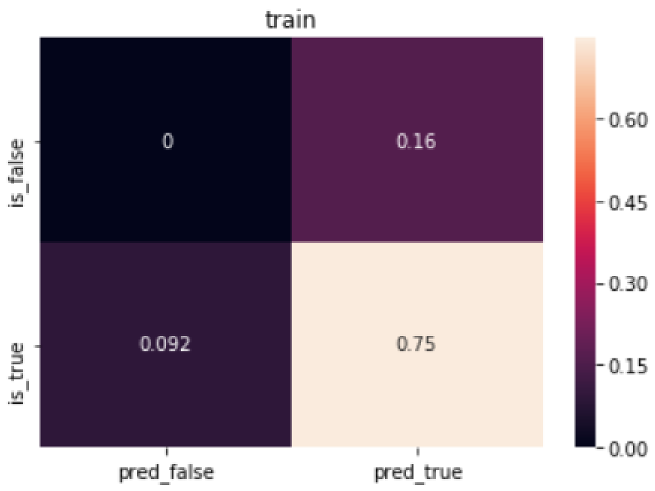
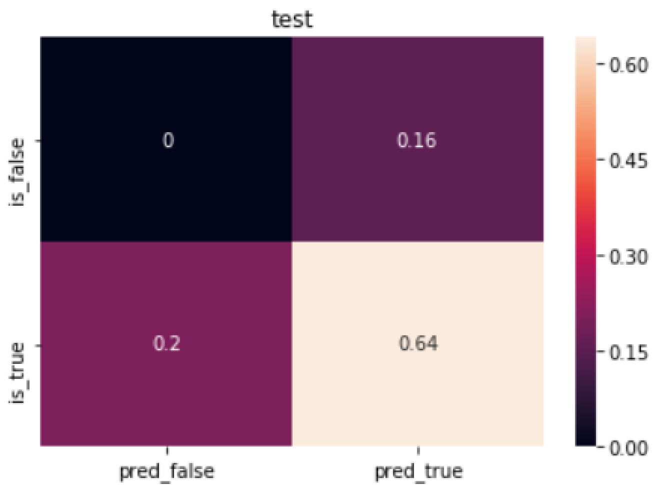


Figure 9: Confusion Matrix (Test)



Another challenge to interpret the accuracy, is that we are doing a pixel-by-pixel comparison. So if there is a certain amount of area in our y_pred that gives the radiologists a general idea of the nodule - that would still serve our utlimate purpose but numerically that would not result in a perfect accuracy for that sample. Hence, the actual results are supposed to be better than what we can see in the confusion matrix above.

VII. CONCLUSIONS

In our project, we were able to perform Semantic Image Segmentation with medical imaging data. Rather than classifying the image as being "diseased" or "healthy", the image overlay that our model predicts highlights areas on the image that warrant additional review by a trained radiologist.

With additional time and resources, we would incorporate more complexity from the LIDC-IDRI .xml files to perform multi-class Semantic Image Segmentation. We would also refine and enhance our "best model". We believe we could use this approach to provide meaningful Deep Learning input to radiologists to enhance their abilities to identify and diagnosis potentially fatal diseases.

The discussion above represents the main thrust of our project and the clearest arc of our analysis. That said, we did spend some additional time considering alternatives that might improve the performance of our final model. Those additional areas we explored are reviewed in Appendix A and Appendix B.

APPENDIX A: IMAGE AUGMENTATION

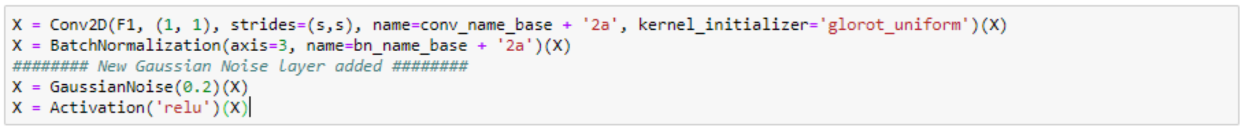
See `ImageAugmentation.ipynb`

For most deep learning models, performance can be improved by two main avenues- 1) by regularizing the model training[1,2] and 2) regularizing through data augmentation. These are techniques that our team identified that may be worthy to explore but were not implemented into the final model due to time constraints.

1) Regularization through adding Gaussian noise

Rather than having fixed weights, the probability distribution chooses weights at random based on a Gaussian distribution centered around the weight you would get without the noise layer. In terms, this model averaging behavior is similar to the effects of regularization, given that the noise layer does not overwhelm the true signals in the input data.

In our model, a variance of 0.2 is used to test the potential performance gain from adding a Gaussian noise layer[3,4], the gaussian noise layers are added within each of the convolutional 2D hidden layer after batch normalization. The outputs from batch normalization provide a standard normal ourput profile that should facilitate the addition of noise from a Gaussian Distribution.

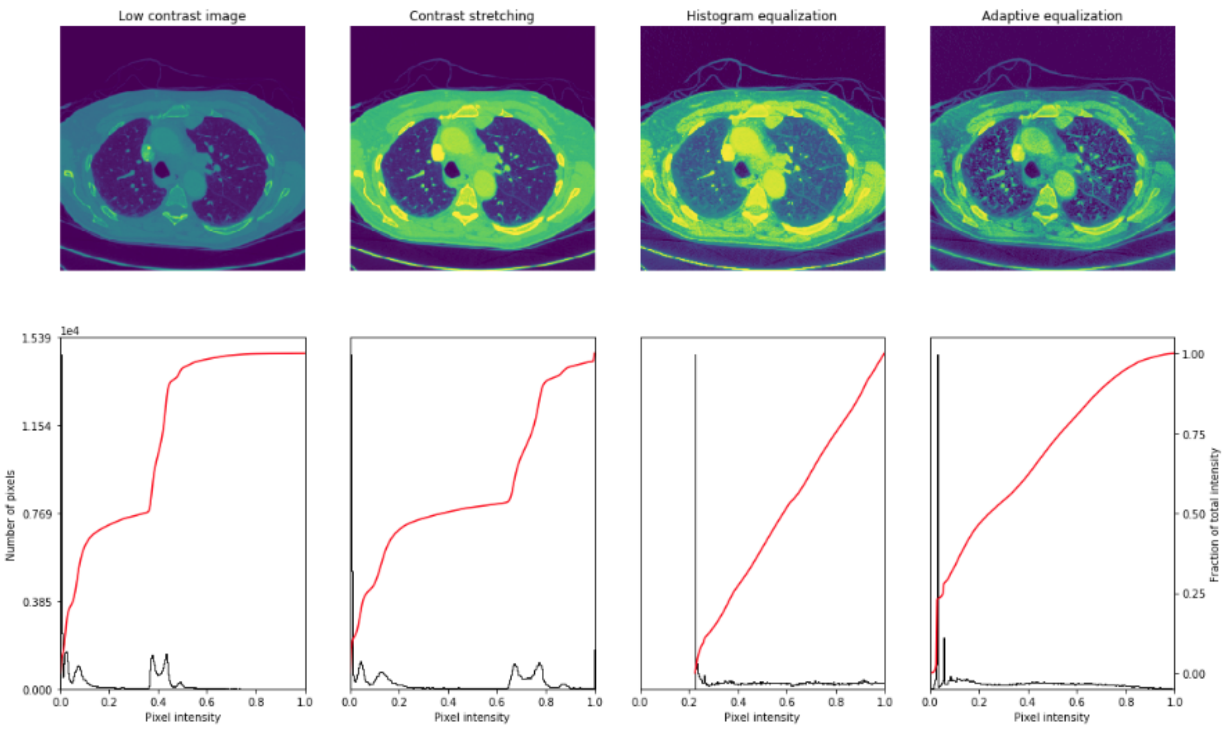


```
X = Conv2D(F1, (1, 1), strides=(s,s), name=conv_name_base + '2a', kernel_initializer='glorot_uniform')(X)
X = BatchNormalization(axis=3, name=bn_name_base + '2a')(X)
######## New Gaussian Noise layer added ########
X = GaussianNoise(0.2)(X)
X = Activation('relu')(X)
```

2) Regularization through image augmentation

The CT scans used in this model have relatively low contrast, and since we are approaching this problem with semantic image segmentation that classifies images pixel by pixel, our team wants to evaluate whether there are any benefits from altering the contrast of the input images[5].

An easy way to perform this task is by manipulating the histogram of the images[6,7]. The histogram of an image shows the number of pixels in an image at each different intensity value found in that image. In the world of deep learning, the scikit-image package provides an easy way to process input images. For our model, we considered the following image-processing techniques:



- Contrast-stretching : The image is rescaled to include all intensities that fall within the 2nd and 98th percentiless
- Equalization : The most frequent intensity values are stretched out, such that the cumulative histogram is linear
- Adaptive equaliztion: the adaptive method computes several histograms, each corresponding to a distinct section of the image, and uses them to redistribute the lightness values of the image. It is therefore suitable for improving the local contrast and enhancing the definitions of edges in each region of an image.

APPENDIX B: LOSS FUNCTION AND CONVOLUTIONAL LOGIC

See `LossFunctionAndSeparableConvolution.ipynb`

We did identify articles and research that suggested other potential enhancements to the above models:[1,2,3]

- Replacement of our binarycrossentropy loss function with Dice or Jaccord loss functions (especially for classification with classes, which often applies with Semantic Image Segmentation)
- Substitution of Separable Convolutional operations for Convolutional operations (both to simplify computation and possibly improve accurcy)
- Use of Dilated Convolution as an alternative for upsampling

With respect to changing the loss function, we did exerpiment with implementations of both Dice loss and Jaccord loss.[4,5] We include a notebook in Appendix C which shows the implementation and training with Dice loss. While we cannot compare the numerical Dice loss with the numerical binarycrossentropy loss on the validation set, we can visually compare results on the test set from models trained using each loss function. By this measure, our implementations show binarycrossentropy as a loss function results in a better trained model than Dice loss.

With respect to substitution of Separable Convolution (i.e., Keras SeparableConv2d) for Convolutional operations in the downsampling portion of the model, we include a notebook in Appendix C which shows the implementation with both the binarycrossentropy loss and the Dice loss. Here are few observations from these results:

- Use of Separable Convolution did reduce the number of trainable weights (from 5,716,417 to 5,554,762) but increased processing time from approximately 700s per epoch to 800s. We also were only able to make the substitution in the Convolutional block. When we substituted in the Identify block, we repeatedly encountered resource exhaustion errors.
- Comparing the binarycrossentropy loss from both implementations, regular convolution produced less loss. This is consistent with regular convolution appearing visually to match more closely the true overlay values.
- As with Dice loss substitution, it does not appear the simple substitution of Separable Convolution for regular convolution in model improves results.

Finally, with respect to Dilated Convolution, we did identify more recent online implemenetations to perform Semantic Image Segmentation that differed significantly from our U-Net & ResNet.[6] We were unable to implement for this project but think this would be an interesting follow-up at a later point.

APPENDIX C: SOURCE PYTHON NOTEBOOKS

The following notebooks and .py files (referenced above) are submitted in a zip file on Canvas with our final report:

- Raw2Processed.ipynb
- Processed2Compiled.ipynb
- Compiled2Training.ipynb
- UNetImplementation.ipynb
- MainResNet.ipynb
- ImageAugmentation.ipynb
- LossFunctionAndSeparableConvolution.ipynb
- model_functions.py
- utility_functions.py

CITATIONS

Section I:

[1] Sankesara, *Towards Data Science*, UNet Introducting Symmetry in Segmentation, https://towardsdatascience.com/u-net-b229b32b4a71 (https://towardsdatascience.com/u-net-b229b32b4a71)

[2] He, Zhang, Ren, Sun, *Microsoft Research*, Deep Residual Learning for Image Recognition, https://arxiv.org/pdf/1512.03385.pdf (https://arxiv.org/pdf/1512.03385.pdf)

Section IIa:

[1] The Cancer Imaging Archive LIDC-IDRI Home Page, https://wiki.cancerimagingarchive.net/display/Public/LIDC-IDRI (https://wiki.cancerimagingarchive.net/display/Public/LIDC-IDRI)

[2] Federov, Hancock, Clunie, et. al., *PeerJPrePrints*, Standardized Representation of the LIDC Annotations using DICOM, https://peerj.com/preprints/27378/ (https://peerj.com/preprints/27378/)

[3] See link for "XML File Documentation" @ https://wiki.cancerimagingarchive.net/display/Public/LIDC-IDRI#2d925967055b4b66a1b7ea8b1b4b25e0 (https://wiki.cancerimagingarchive.net/display/Public/LIDC-IDRI#2d925967055b4b66a1b7ea8b1b4b25e0)

Section IIb:

[1] See link for "tcia-diagnosis-data-2012-04-20.xls" @ https://wiki.cancerimagingarchive.net/display/Public/LIDC-IDRI#2d925967055b4b66a1b7ea8b1b4b25e0 (https://wiki.cancerimagingarchive.net/display/Public/LIDC-IDRI#2d925967055b4b66a1b7ea8b1b4b25e0)

Section IV:

[1] U-Net: Convolutional Networks for Biomedical Image Segmentation. https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/ (https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/)

Section V & VI:

[1] Understanding and Coding a ResNet in Keras. https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33 (https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33)

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. *Deep Residual Learning for Image Recognition.* arXiv:1512.03385. https://arxiv.org/pdf/1512.03385.pdf (https://arxiv.org/pdf/1512.03385.pdf).

Appendix A:

[1] Li, Liu, Whiteout: Gaussian Adaptive Noise Injection Regularization in Deep Neural Networks, https://arxiv.org/pdf/1612.01490.pdf (https://arxiv.org/pdf/1612.01490.pdf)

[2] Greydanus, *Sam's Blog*, The art of regularization, https://greydanus.github.io/2016/09/05/regularization/ (https://greydanus.github.io/2016/09/05/regularization/)

[3] Brownlee, How to Improve Deep Learning Model Robustness by Adding Noise, https://machinelearningmastery.com/how-to-improve-deep-learning-model-robustness-by-adding-noise/ (https://machinelearningmastery.com/how-to-improve-deep-learning-model-robustness-by-adding-noise/)

[4] Keras Documentation, Noise Layers, https://keras.io/layers/noise/ (https://keras.io/layers/noise/)

[5] Singh, Bansal, Bansal, Medical Image enhancement using histogram processing techniques followed by median filter, https://pdfs.semanticscholar.org/244c/3239c8c2be16340cb2b27f2de21860c5a4e6.pdf (https://pdfs.semanticscholar.org/244c/3239c8c2be16340cb2b27f2de21860c5a4e6.pdf)

[6] Scikit-image documentation, Histogram Equalization, https://scikit-image.org/docs/dev/auto_examples/color_exposure/plot_equalize.html (https://scikit-image.org/docs/dev/auto_examples/color_exposure/plot_equalize.html)

[7] Lerman, Raicu, Furst, Intelligent Multimedia Processing Laboratory, Contrast enhancement of soft tissues in Computed Tomography images, https://facweb.cdm.depaul.edu/research/vc/publications/BinningManuscript2Roman_JF.pdf (https://facweb.cdm.depaul.edu/research/vc/publications/BinningManuscript2Roman_JF.pdf)

Appendix B:

[1] Jordan, *Jeremy Jordan*, Evaluating Image Segmentation Models, https://www.jeremyjordan.me/evaluating-image-segmentation-models/ (https://www.jeremyjordan.me/evaluating-image-segmentation-models/)

[2] Chen, Zhu, Papandreou, Schriff, Adam, *GroundAI*, Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation, https://www.groundai.com/project/encoder-decoder-with-atrous-separable-convolution-for-semantic-image-segmentation/1 (https://www.groundai.com/project/encoder-decoder-with-atrous-separable-convolution-for-semantic-image-segmentation/1)

[3] Silva, *Thalle's Blog*, Deeplab Image Semantic Segmentation Network, https://sthalles.github.io/deep_segmentation_network/ (https://sthalles.github.io/deep_segmentation_network/)

[4] Clark, *GitHub*, dice_loss_for_keras, https://gist.github.com/wassname/7793e2058c5c9dacb5212c0ac0b18a8a (https://gist.github.com/wassname/7793e2058c5c9dacb5212c0ac0b18a8a)

[5] Clark, *GitHub*, jaccard_coef_loss for keras, https://gist.github.com/wassname/f1452b748efcbeb4cb9b1d059dce6f96 (https://gist.github.com/wassname/f1452b748efcbeb4cb9b1d059dce6f96)

[6] Yu, *GitHub*, Dilated Convolution for Semantic Image Segmentation, https://github.com/fyu/dilation (https://github.com/fyu/dilation)