Final project

COVID segmentation based on Deep Learning

Presenters:
Adeyemi, Abdulnafiu Olalekan
Khawaja, Muhammad Arsalan
Rodriguez, Joaquin

Evaluators:
Fabrice Meriaudeau
Abdul Qayyum

TABLE OF CONTENTS

Demo

2.1 Models implemented: Introduction

- 20 CT scans with ground-truth data to train
- Multi-class segmentation: Left and right lung and Disease
- Starting point: Unet-Model - state-of-the-art model
- Two approaches:
  - Custom approach - model designed in code and modified
  - Model from Segmentation Models library - Unet - EficientNet B7 - ImageNet weights
- EficientNet B7: Best accuracy and minimum amount of params (75M)
- Optimizer used: Adaptive Moment Estimation - Adam
- Metrics implemented
  - Histogram of dice coefficient
  - Histogram of Hausdorff distance
  - Receiving Operating Characteristic (ROC) curve
  - Area under the ROC curve

2.2 Models implemented: Architecture

```
def put_layer(self, previous_layer, no_nuerons):
    '''
    This Function Adds layer in CNN model
    '''
    first_conv = Conv2D(no_nuerons, (3, 3), activation='relu', padding='same')(previous_layer)
    layer_complete = Conv2D(no_nuerons, (3, 3), activation='relu', padding='same')(first_conv)

    return layer_complete
```

```
def inception_module(self, x,
                     filters_1x1,
                     filters_3x3_reduce,
                     filters_3x3,
                     filters_5x5_reduce,
                     filters_5x5,
                     filters_pool_proj,
                     name=None):

    from keras.initializers import glorot_uniform, Constant
    kernel_init = glorot_uniform()
    bias_init = Constant(value=0.2)

    conv_1x1 = Conv2D(filters_1x1, (1, 1), padding='same', activation='relu',
        kernel_initializer=kernel_init, bias_initializer=bias_init)(x)

    conv_3x3 = Conv2D(filters_3x3_reduce, (1, 1), padding='same', activation='relu',
        kernel_initializer=kernel_init, bias_initializer=bias_init)(x)

    conv_3x3 = Conv2D(filters_3x3, (3, 3), padding='same', activation='relu',
        kernel_initializer=kernel_init, bias_initializer=bias_init)(conv_3x3)

    conv_5x5 = Conv2D(filters_5x5_reduce, (1, 1), padding='same', activation='relu',
        kernel_initializer=kernel_init, bias_initializer=bias_init)(x)

    conv_5x5 = Conv2D(filters_5x5, (5, 5), padding='same', activation='relu',
        kernel_initializer=kernel_init, bias_initializer=bias_init)(conv_5x5)

    pool_proj = MaxPooling2D((3, 3), strides=(1, 1), padding='same')(x)

    pool_proj = Conv2D(filters_pool_proj, (1, 1), padding='same', activation='relu',
        kernel_initializer=kernel_init, bias_initializer=bias_init)(pool_proj)

    output = concatenate([conv_1x1, conv_3x3, conv_5x5, pool_proj], axis=3, name=name)

    return output
```

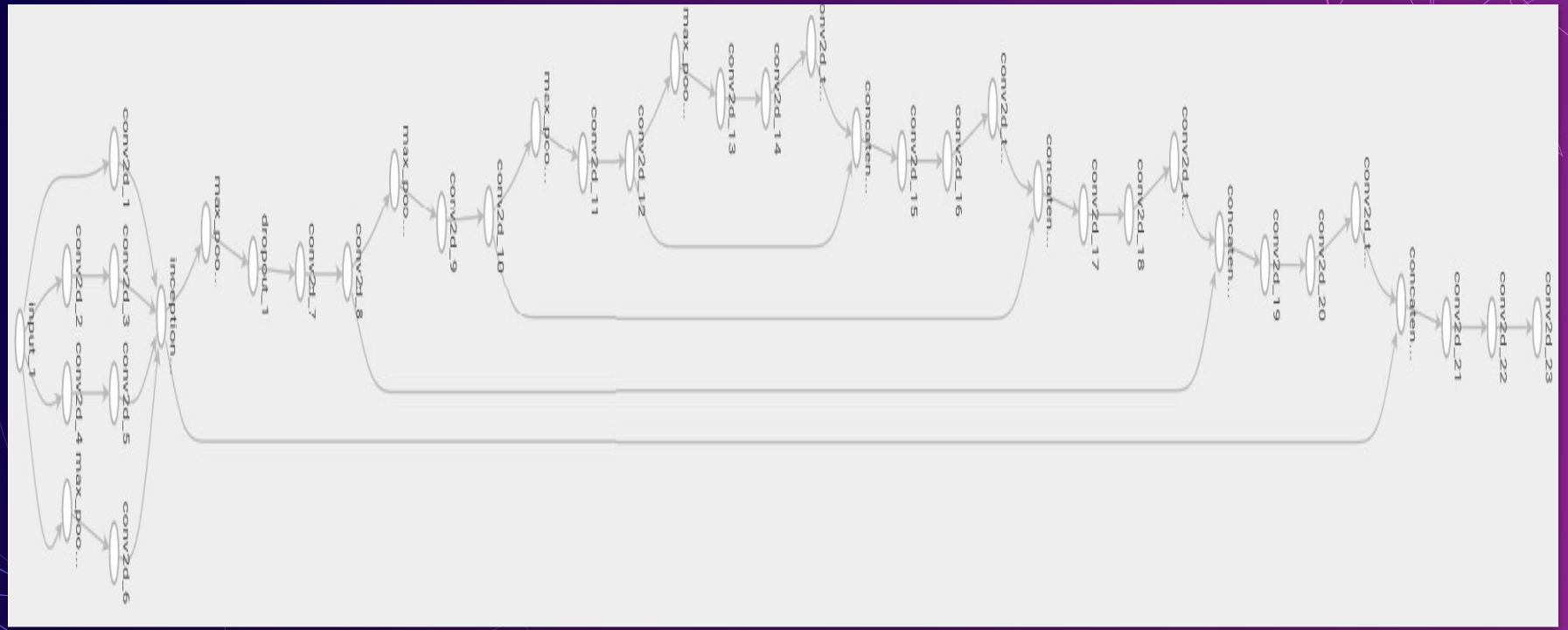
2.2 Models implemented: Architecture

```
def first_sketch_model(self, classes):
    from keras.losses import BinaryCrossentropy, CategoricalCrossentropy
    from keras.optimizers import Adam

    optimizer = Adam(lr = 0.0001)
    loss_func = CategoricalCrossentropy()
    activation_func = 'sigmoid'

    inputs = Input((None, None, 1))

    layer_1 = self.inception_module(inputs,
        64, 96,
        128, 16,
        32 , 32,
        name = 'inception_1')      You, a few seconds ago • Uncommitted ch
    pool1 = MaxPooling2D(pool_size=(2, 2))(layer_1)
    drop = Dropout(0.3)(pool1)

    layer_2 = self.put_layer(drop,64)
    pool2 = MaxPooling2D(pool_size=(2, 2))(layer_2)

    layer_3 = self.put_layer(pool2,128)
    pool3 = MaxPooling2D(pool_size=(2, 2))(layer_3)

    layer_4 = self.put_layer(pool3, 256)
    pool4 = MaxPooling2D(pool_size=(2, 2))(layer_4)

    layer_5 = self.put_layer(pool4, 512)

    up6 = concatenate([Conv2DTranspose(256,
        (2, 2),
        strides=(2, 2),
        padding='same')(layer_5), layer_4], axis=3)
    layer_6 = self.put_layer(up6, 256)

    up7 = concatenate([Conv2DTranspose(128,
        (2, 2),
        strides=(2, 2),
        padding='same')(layer_6), layer_3], axis=3)
    layer_7 = self.put_layer(up7, 128)

    up8 = concatenate([Conv2DTranspose(64,
        (2, 2),
        strides=(2, 2),
        padding='same')(layer_7), layer_2], axis=3)
    layer_8 = self.put_layer(up8, 64)

    up9 = concatenate([Conv2DTranspose(32,
        (2, 2),
        strides=(2, 2),
        padding='same')(layer_8), layer_1], axis=3)
    layer_9 = self.put_layer(up9, 32)

    layer_10 = Conv2D(classes, (1, 1), activation=activation_func)(layer_9)

    model = Model(inputs=[inputs], outputs=[layer_10])
    model.compile(optimizer = optimizer, loss=loss_func, metrics = ['accuracy'])
    return model
```

2.2 Models implemented: Architecture

2.2

Models implemented: Architecture



2.2 Models implemented: Architecture

Key Features of our proposed Hybrid Model

- Primary Architecture: U-net
- Inception module embedded in U-net
- Additional dropout layer to avoid overfitting
- Categorical Cross entropy
- Adam optimizer: Initial LR 0.0001
- Amount of epochs = 8
- Batch size = 8
- Activation function for last layer: Sigmoid

3 Experimentation & results

- **Test realized:**
  - Convolution layers added
  - Single Drop out layer - Change in the dropout rate : 10, 30 and 50%
  - Multiple dropout layers (Encoder section, DO rate: 30 and 50%)
  - Inception modules added
    - Single module added as first layer
    - Multiple modules added in the first and second layer
  - Combination of dropout and inception module
  - Changing epochs and batch size
  - Use of different loss function
    - Binary cross entropy
    - Categorical cross entropy
    - Poisson
    - MSE - MAE
    - Kullback-Leibler Divergence

3 Experimentation & results

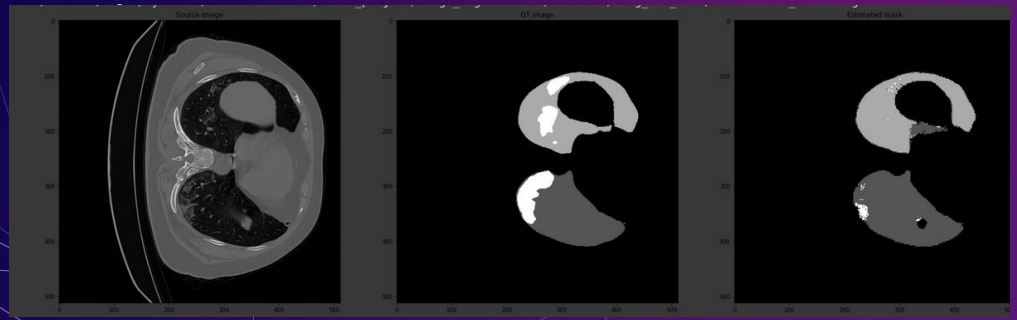
Initial experimental setup

- Results of the initial model:
  - Optimizer = Adam
  - Learning rate = 0.0001
  - Loss function = Binary Cross entropy
  - Activation function in last layer: softmax
  - No drop out layer used
  - No Data Augmentation used

3 Experimentation & results

- **Experiment 1: Adding convolutional layer**

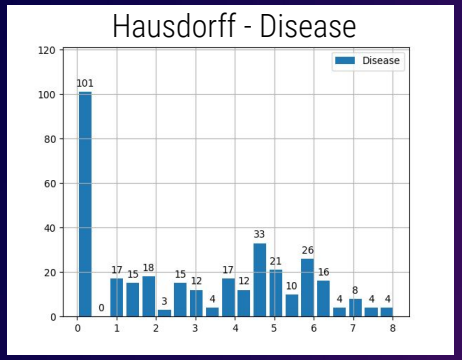
- **Adding one convolutional layer**

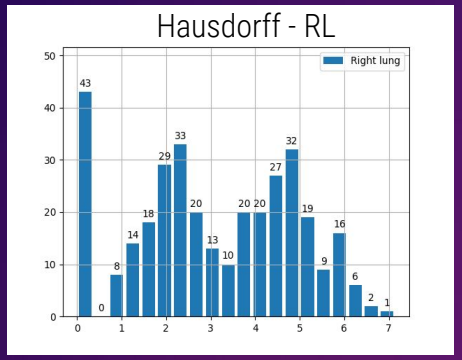


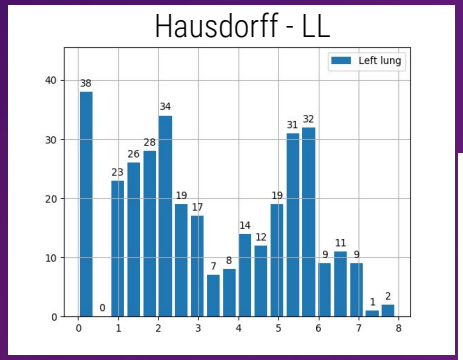
- **Adding two convolutional layer**

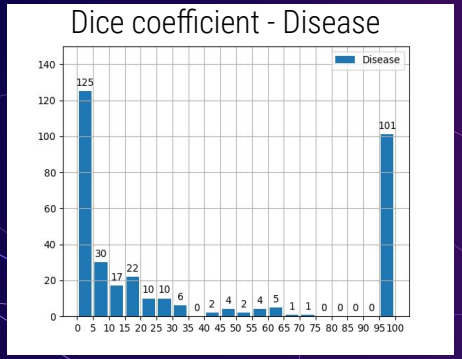
3 Experimentation & results

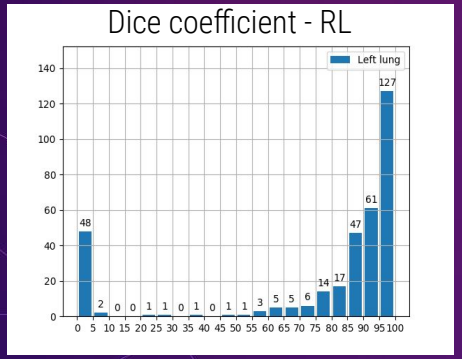
- **Experiment 1: Statistical Analysis adding one convolution layer**

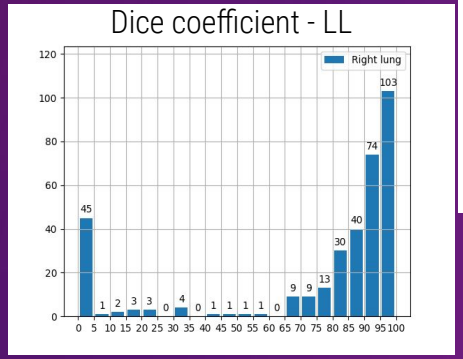


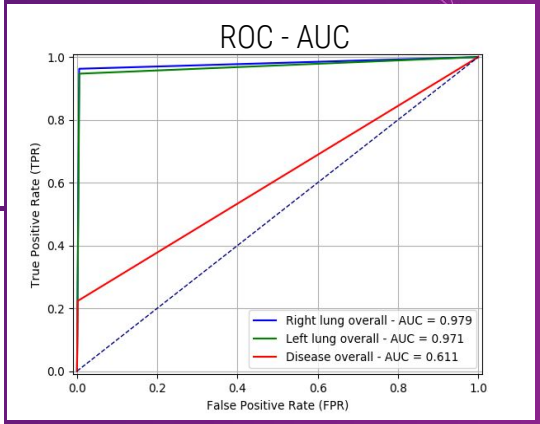










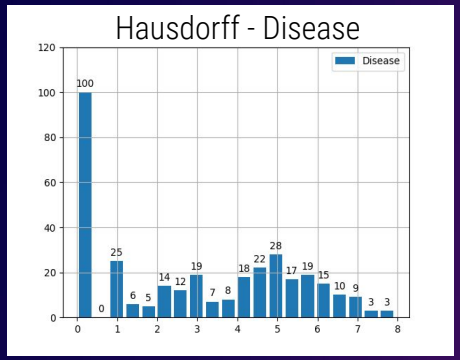


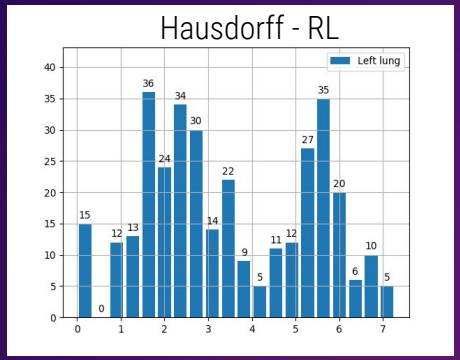
- **Experiment 2: Adding dropout layer to initial model**

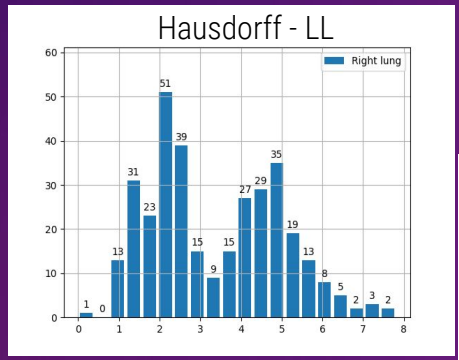
- **Dropout of 10% on layer 4**
- **Dropout of 30% on layer 4**
- **Adding DO (30%) on first 5 layers**

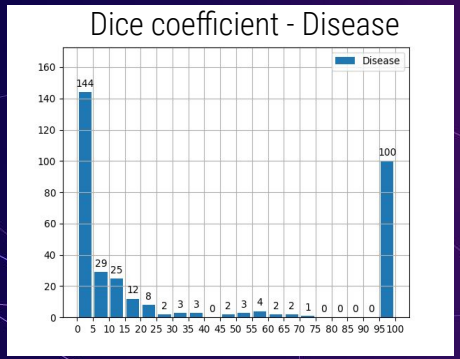
3 Experimentation & results

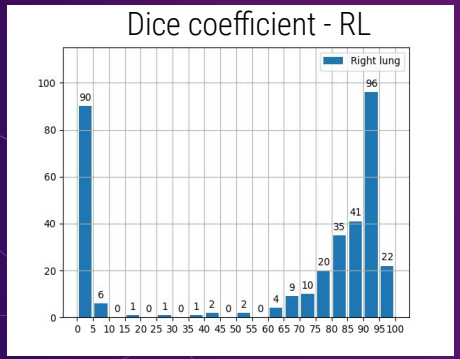
- **Experiment 2: Statistical Analysis for DO 10% on first layer**

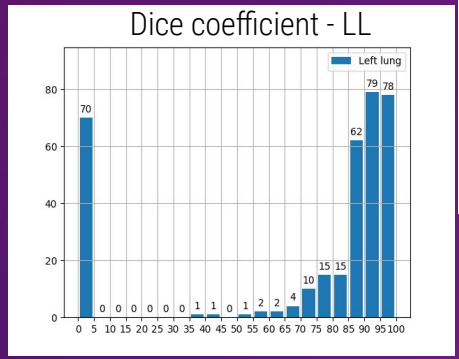


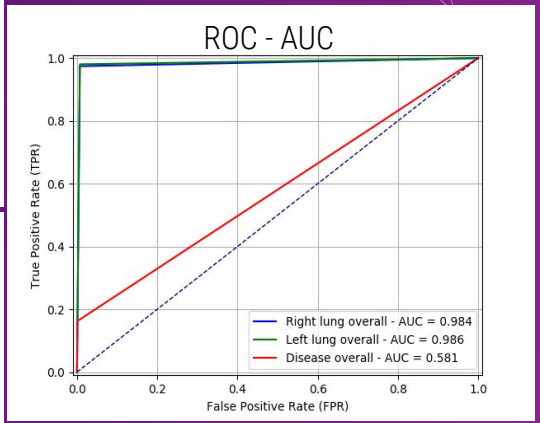






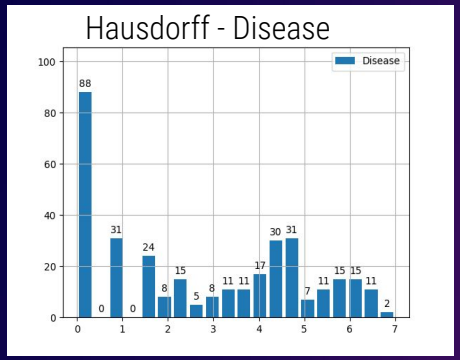


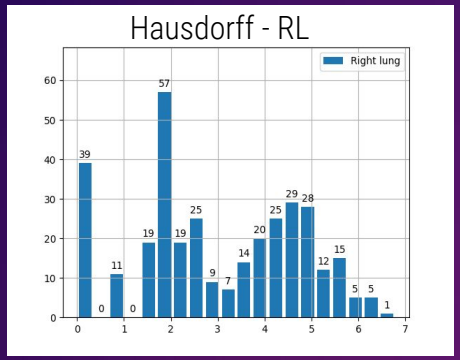


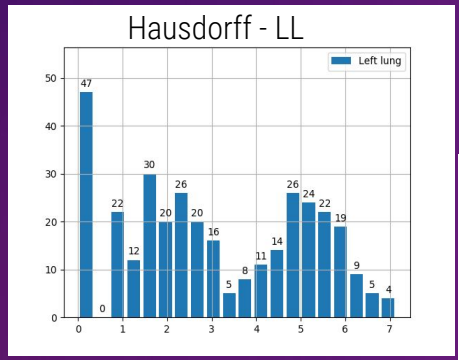


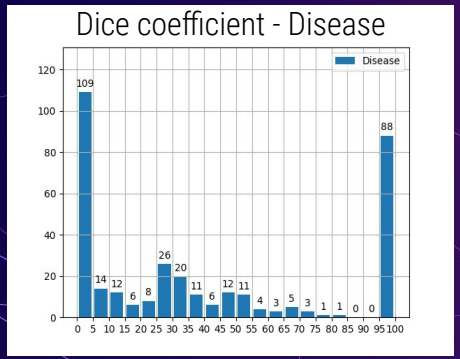
3 Experimentation & results

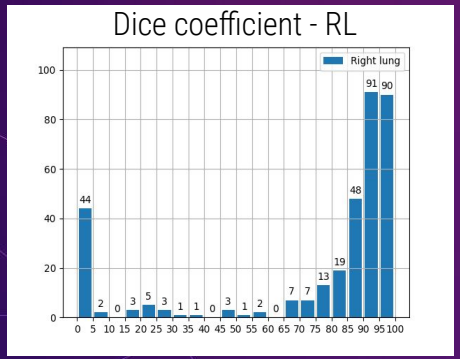
- **Experiment 2: Statistical Analysis for DO 30% on first layer**

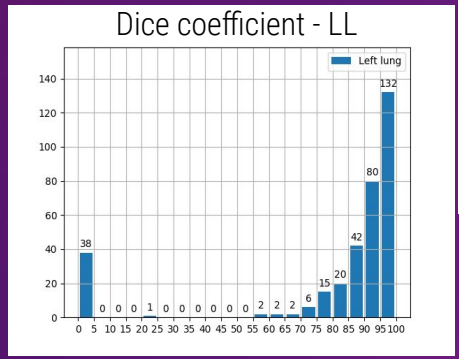


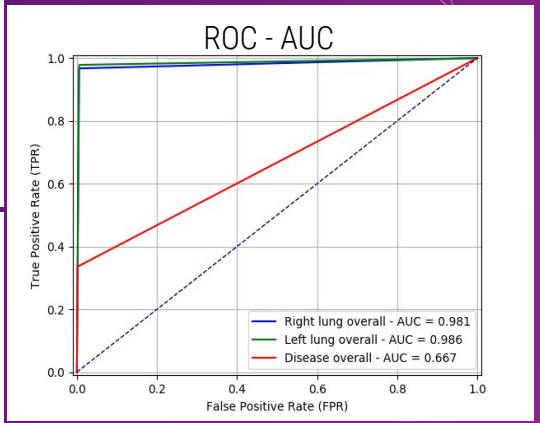






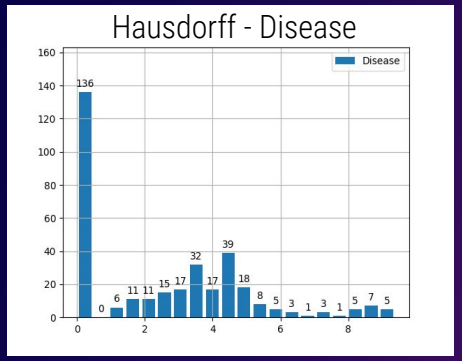


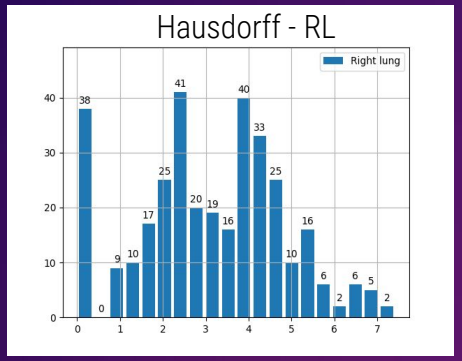


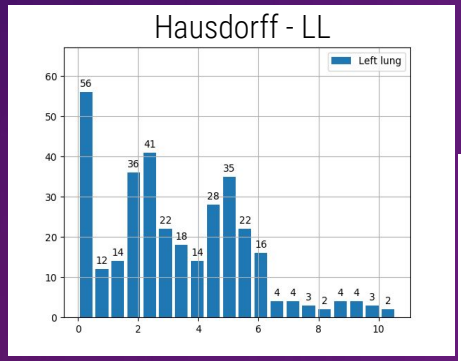


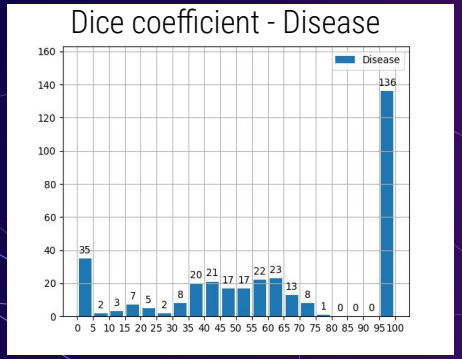
3 Experimentation & results

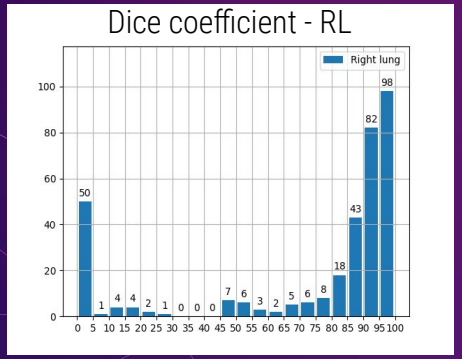
- **Experiment 2: Statistical Analysis of adding 30% DO on first 5 layers**

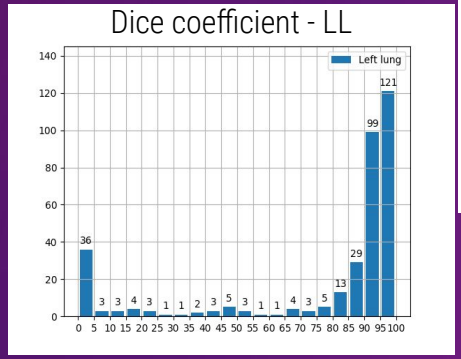


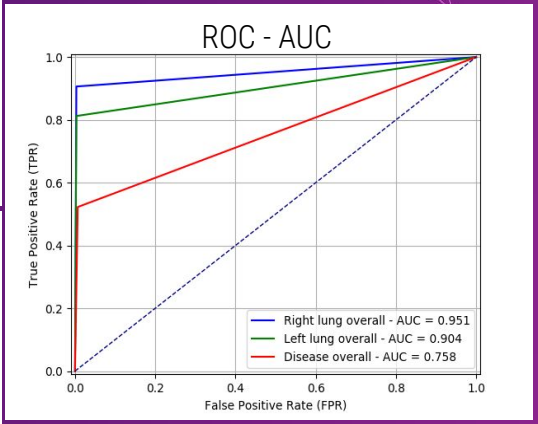






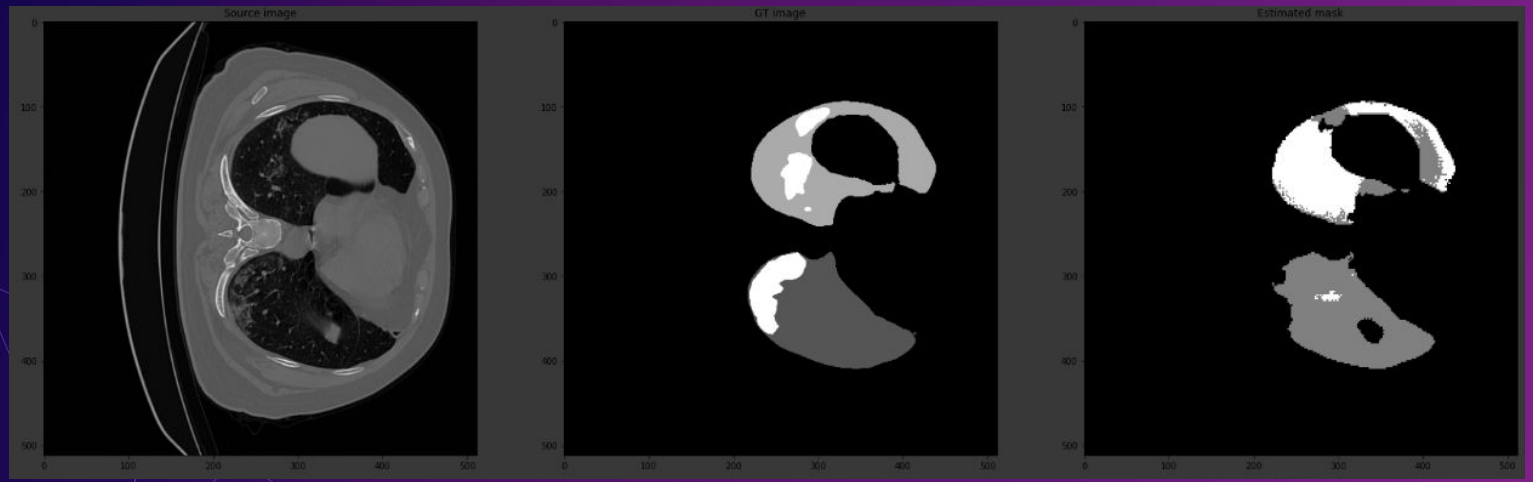






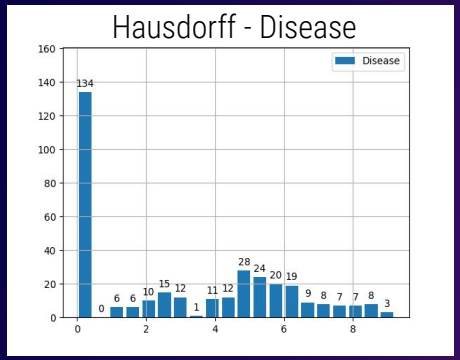
3 Experimentation & results

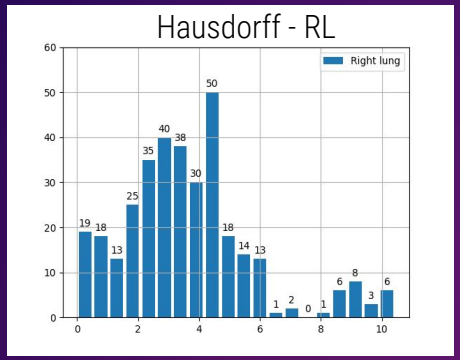
- **Experiment 4: Adding DO (50%) on first 5 layers**

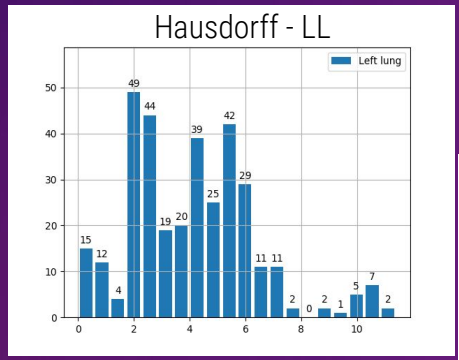


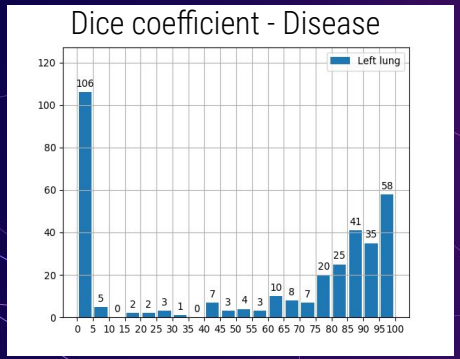
3 Experimentation & results

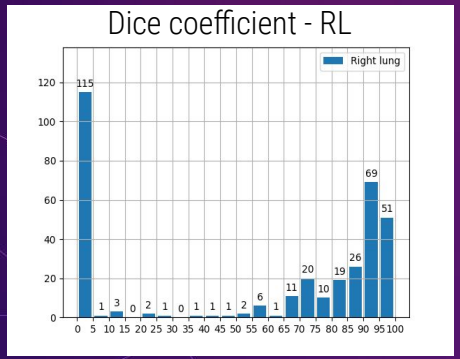
- **Experiment 4: Statistical analysis Adding DO(50%) on first 5 layers**

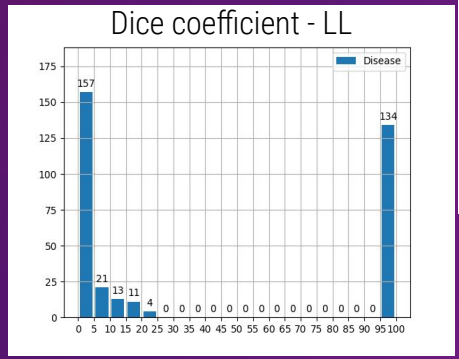


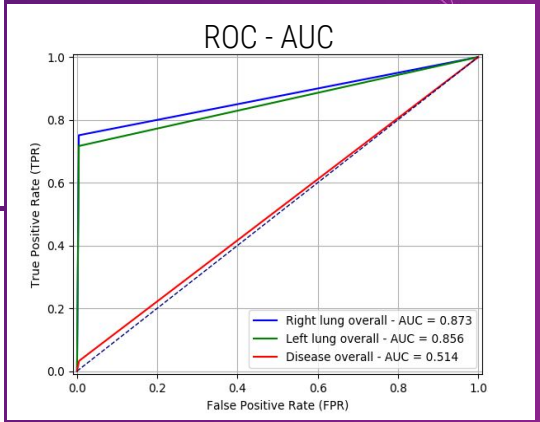






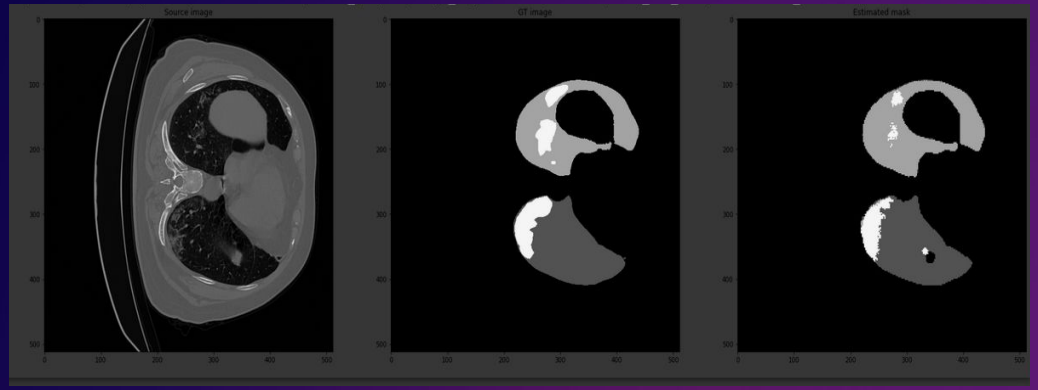






3 Experimentation & results

- **Experiment 3: Trying different activation functions at the last layer**

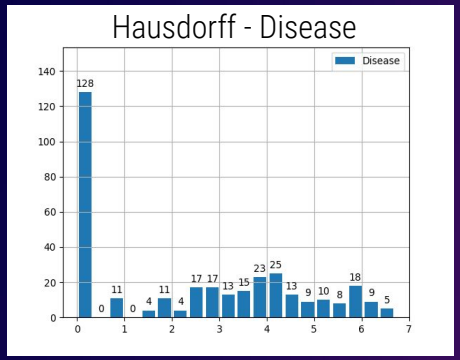


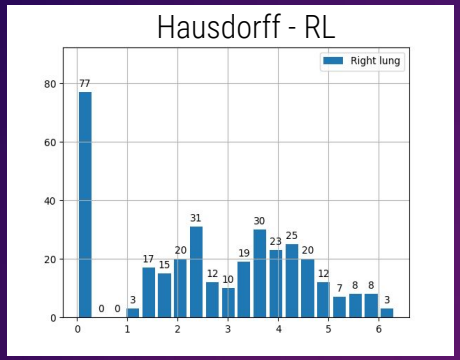
- **Sigmoid**

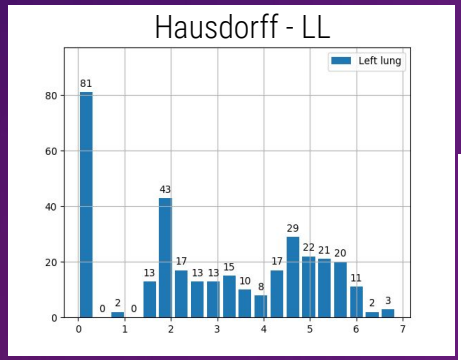
- **Softmax**

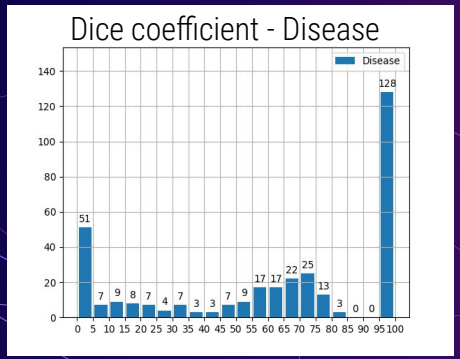
3 Experimentation & results

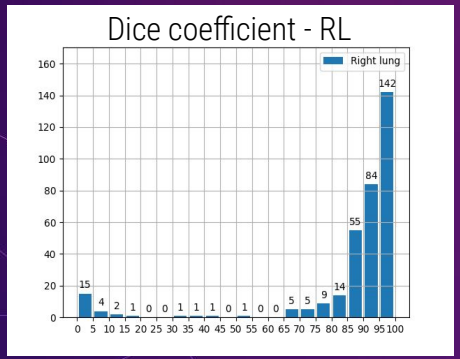
- **Experiment 3: Statistical Analysis of Sigmoid Activation function**

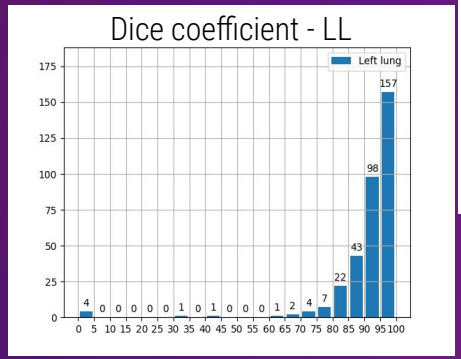


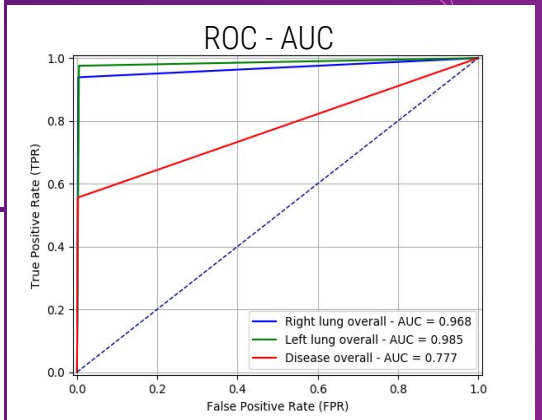






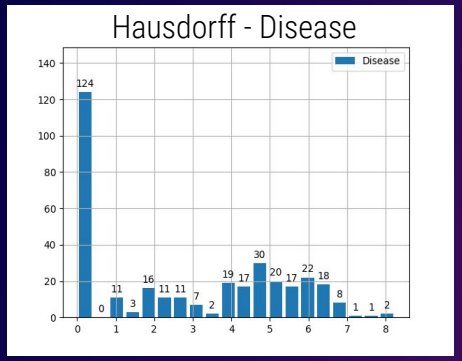


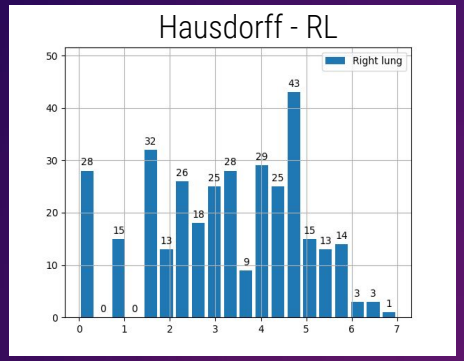


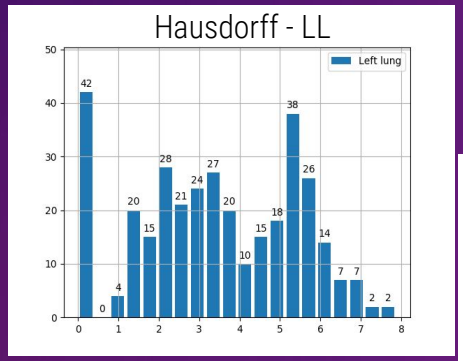


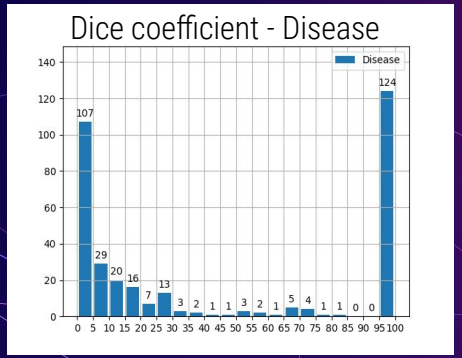
3 Experimentation & results

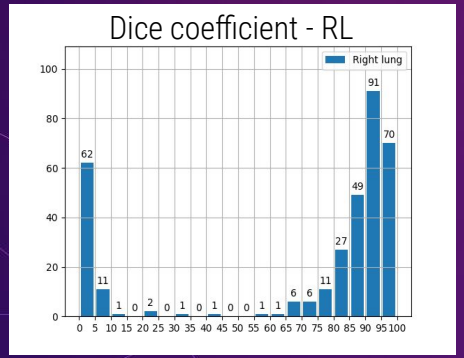
- **Experiment 3: Statistical Analysis of Softmax Activation function**

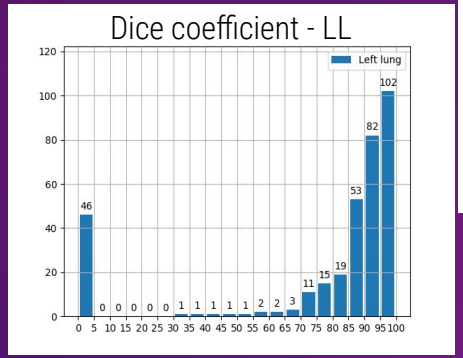


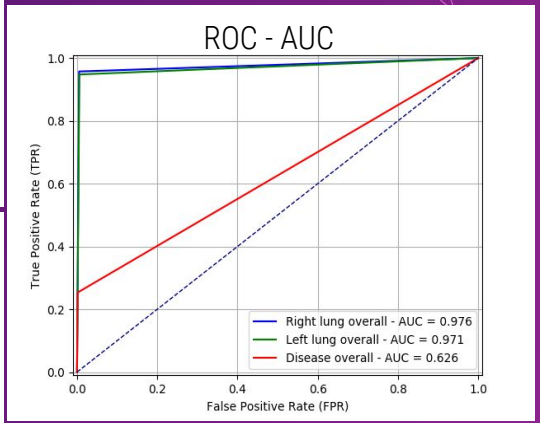












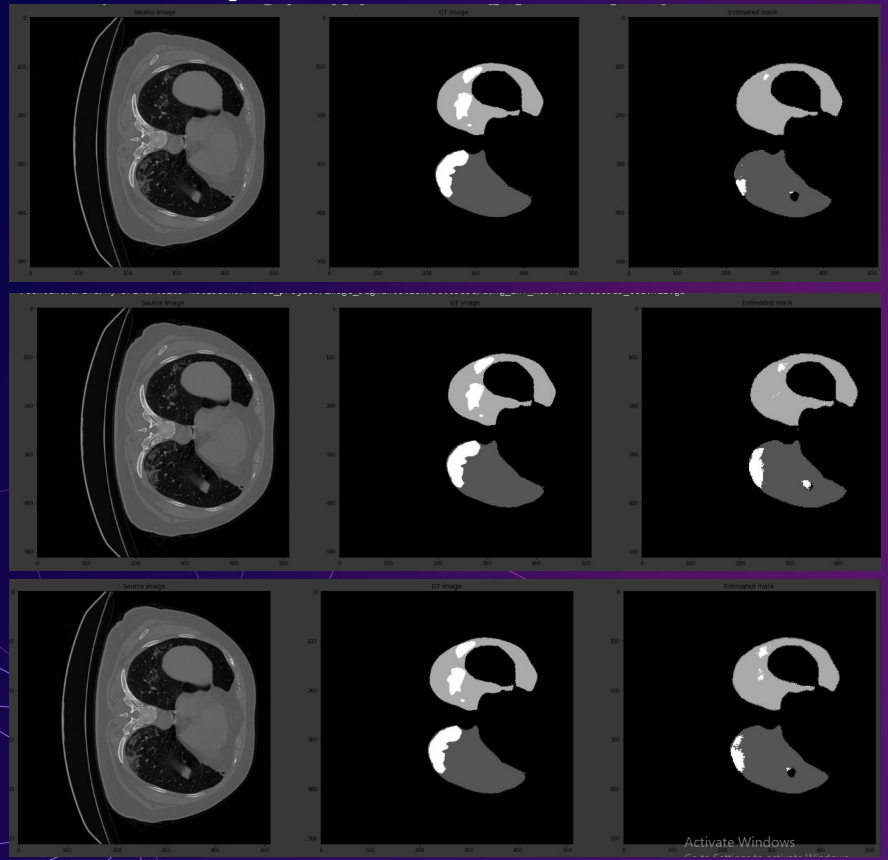
- **Experiment 4: Trying different loss functions**

Using different loss functions on original U-Net model:

- Probabilistic Loss functions
  - Categorical Cross entropy
  - Poisson
  - KL - Divergence
- Regression loss functions
  - Mean squared error
  - Mean absolute error

3 Experimentation & results

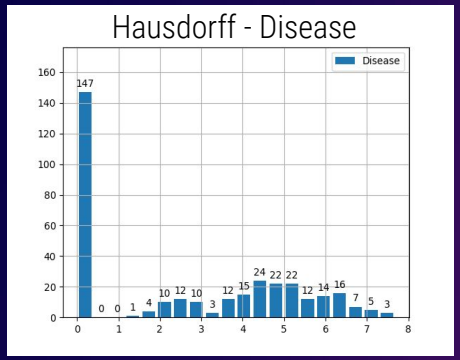
- **Experiment 4 - A: Probabilistic loss functions**

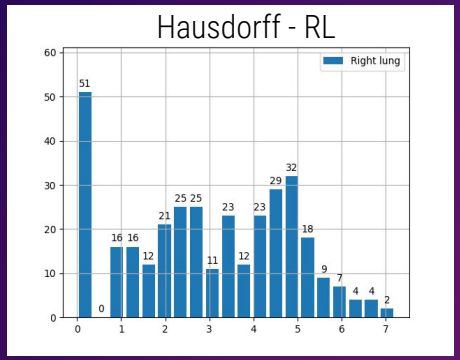


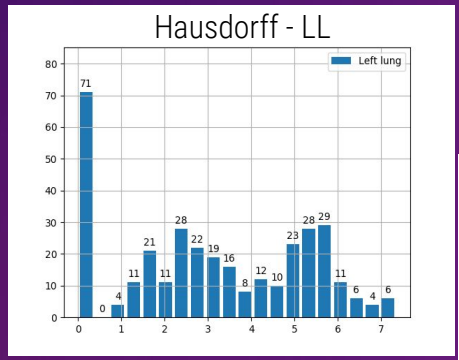
- **Categorical cross entropy**
- **Poisson**
- **Kullback-Leibler divergence**

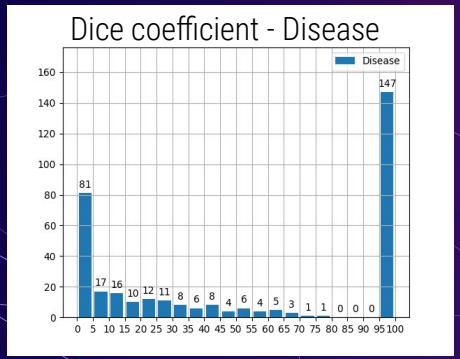
3 Experimentation & results

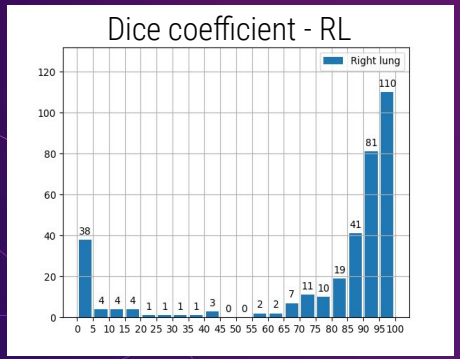
- **Experiment 4 - A: Statistical Analysis of Categorical Cross Entropy LF**

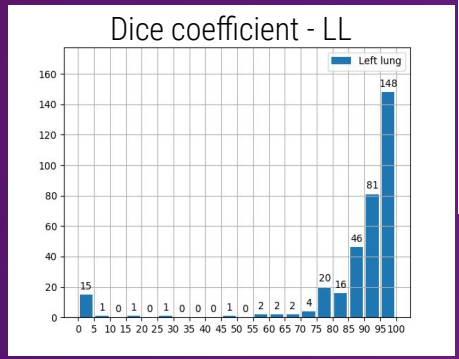


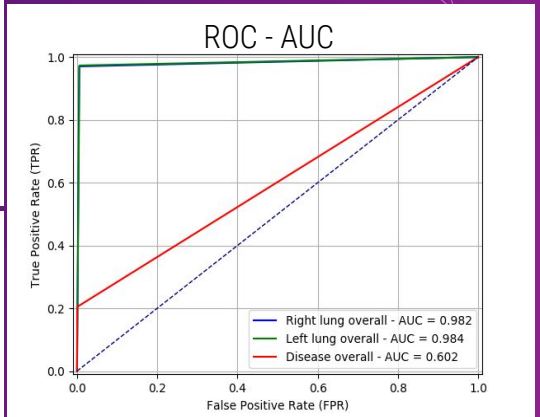






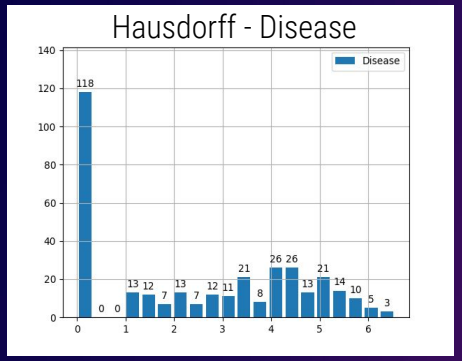


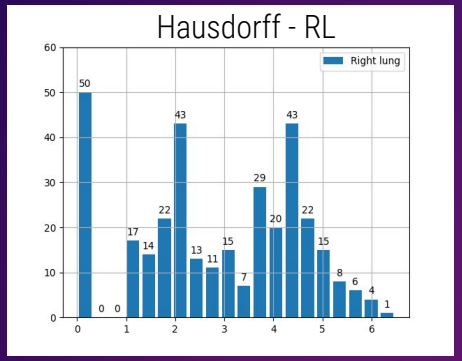


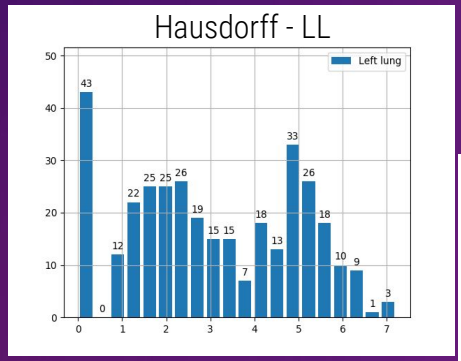


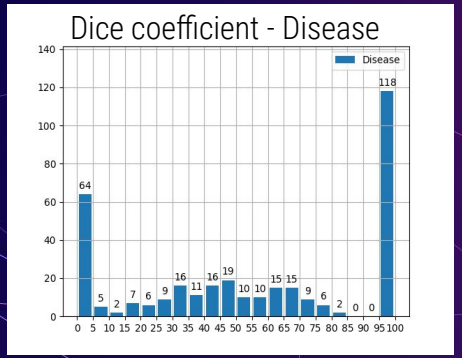
3 Experimentation & results

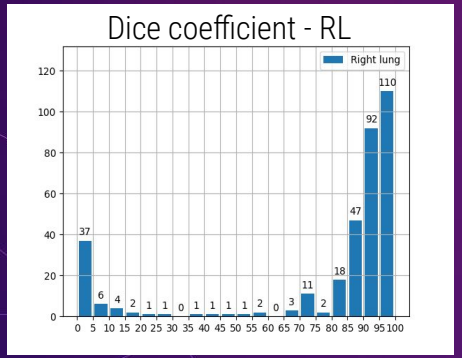
- **Experiment 4 - A: Statistical Analysis of Poisson LF**

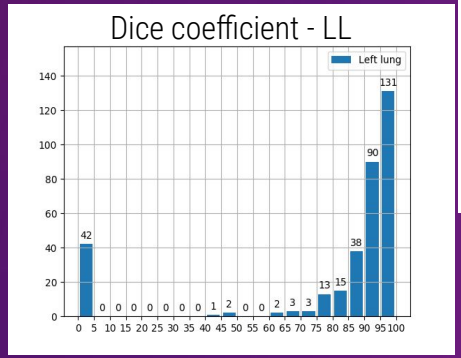


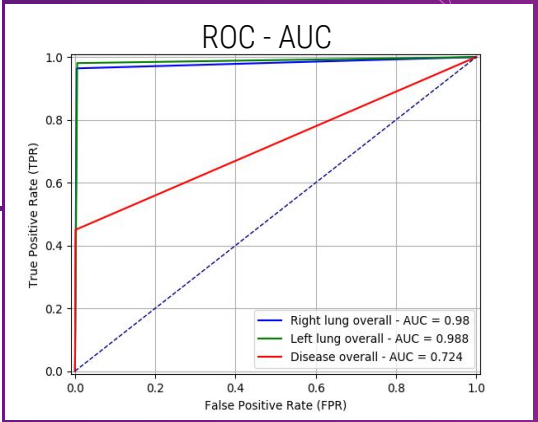






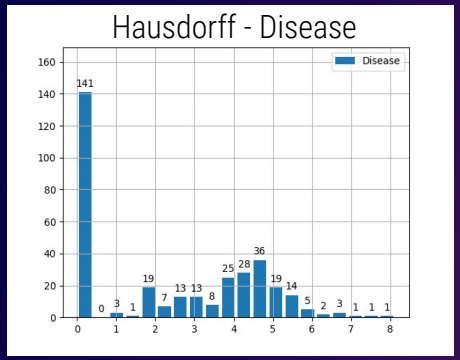


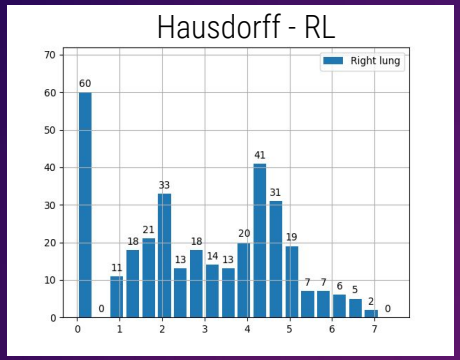


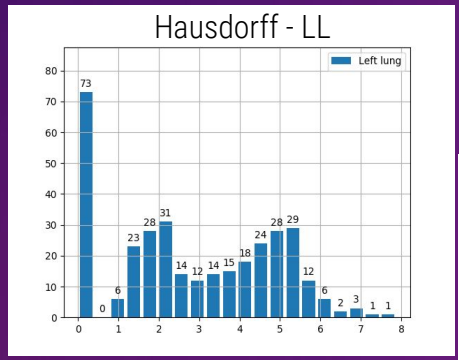


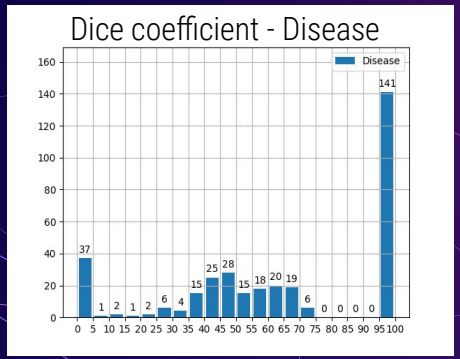
3 Experimentation & results

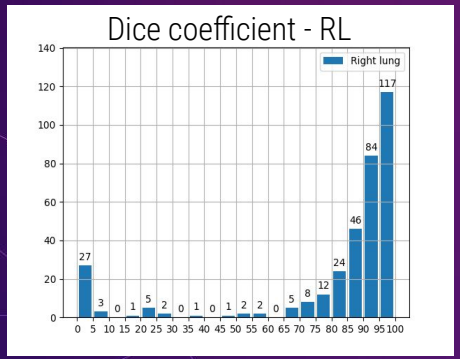
- **Experiment 4 - A: Statistical Analysis of Kullback-Leibler divergence LF**

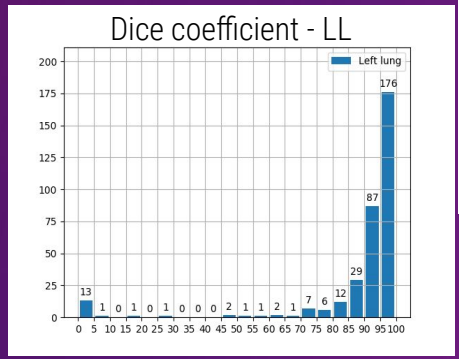


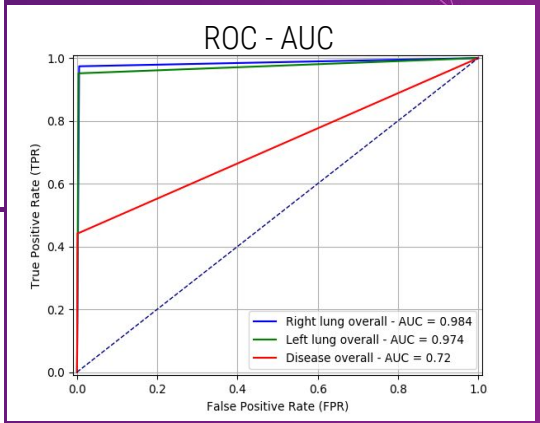




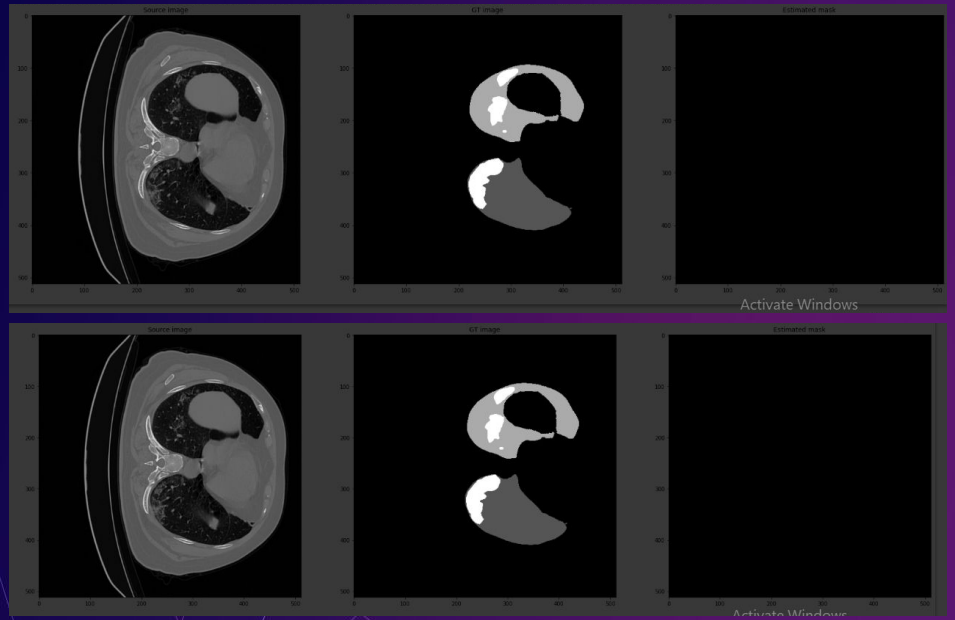








- **Experiment 4 - B: Regression models LF**

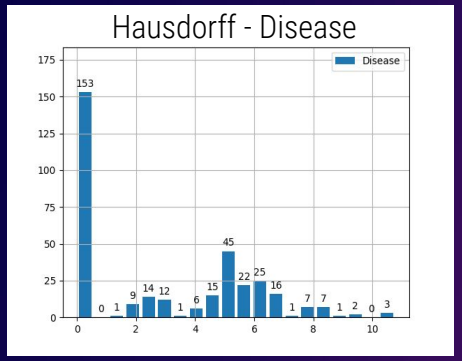


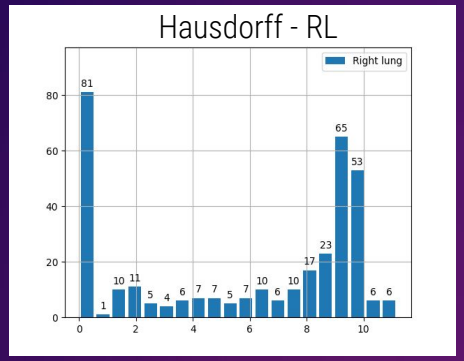
- **Mean square error**
- **Mean absolute error**

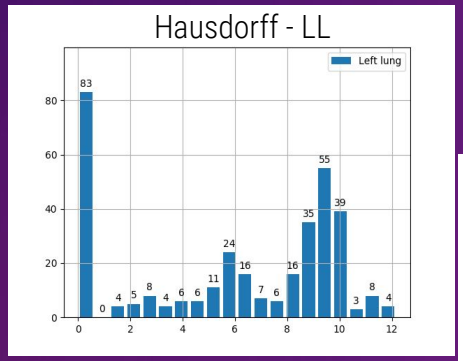
We can see that, even though the PLF work for the training, the absolutely fail into segmenting the image (It succeeded in segmenting only background)

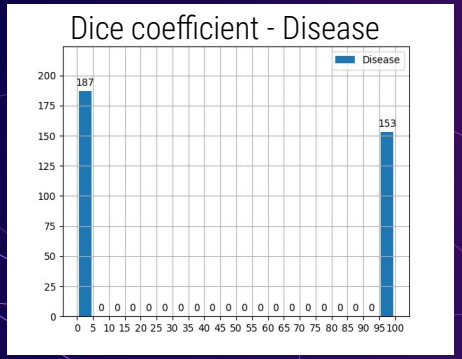
3 Experimentation & results

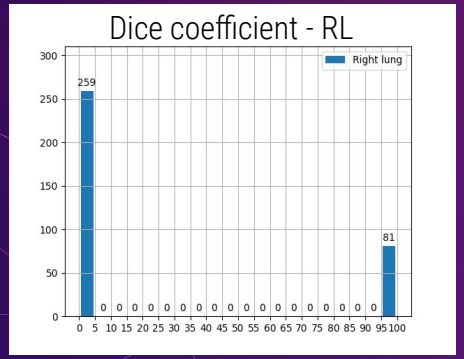
- **Experiment 5 - B: Statistical Analysis of Absolute / Mean Squared error LF**

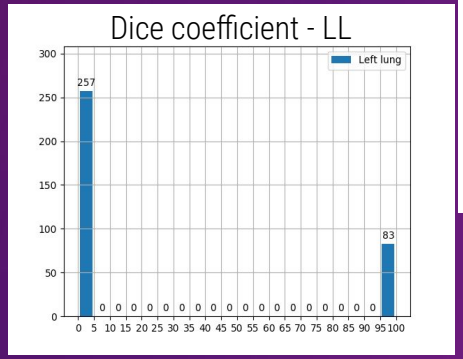


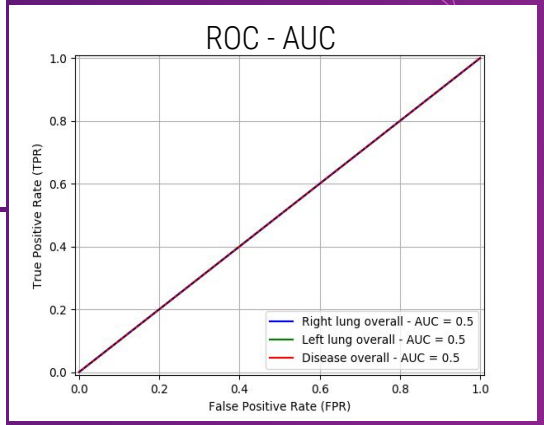






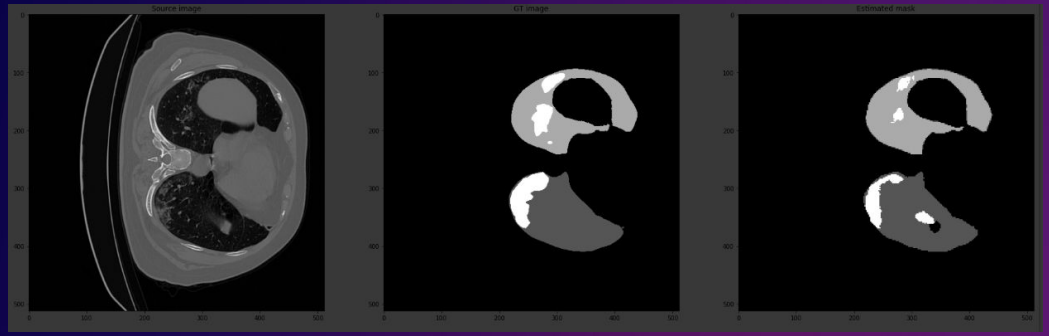




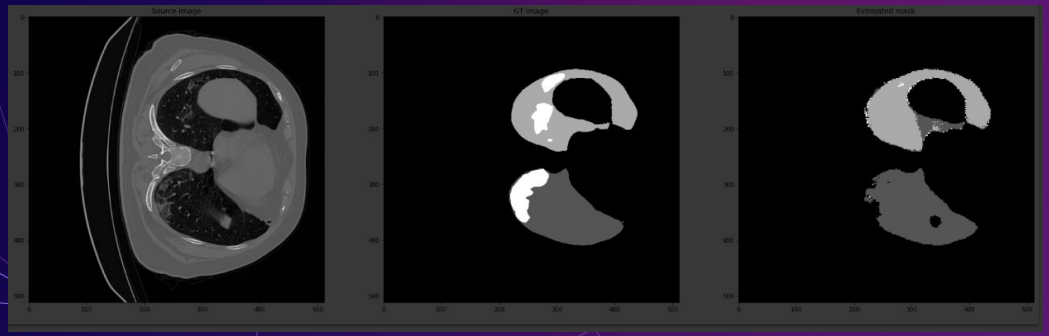


3 Experimentation & results

- **Experiment 5: Using Different Learning rates (Adam optimizer)**



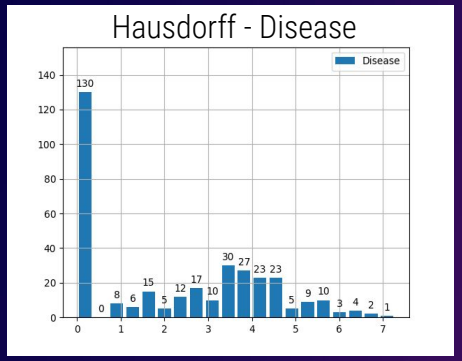
- **Learning rate = 0.001**

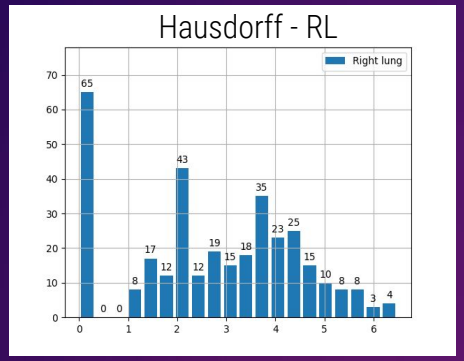


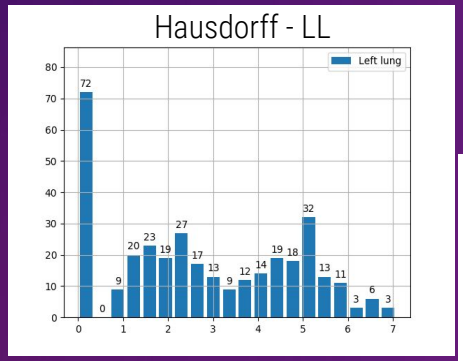
- **Learning rate = 0.00001**

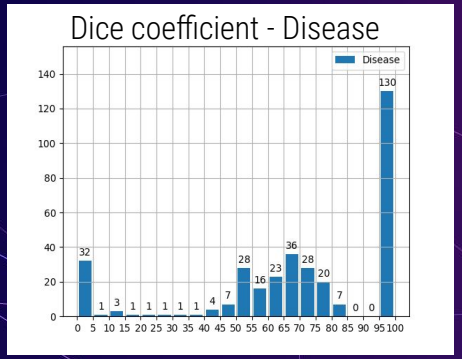
3 Experimentation & results

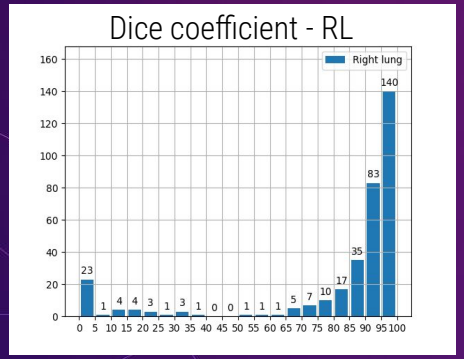
- **Experiment 5: Statistical Analysis of 0.001 learning rate**

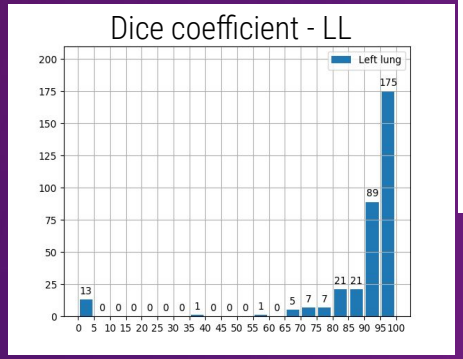


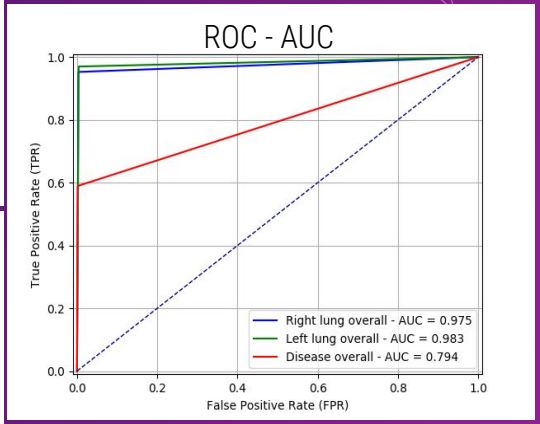






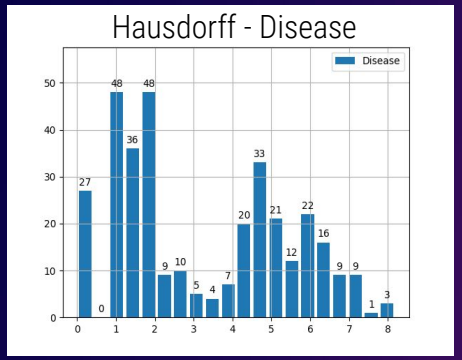


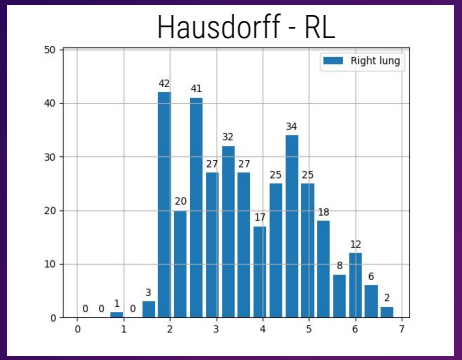


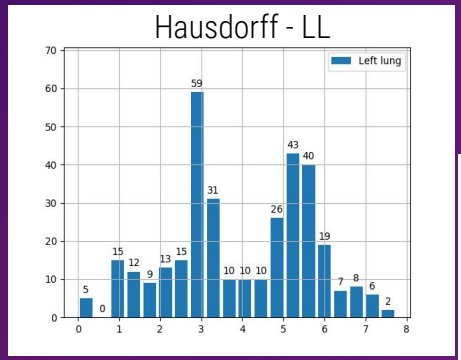


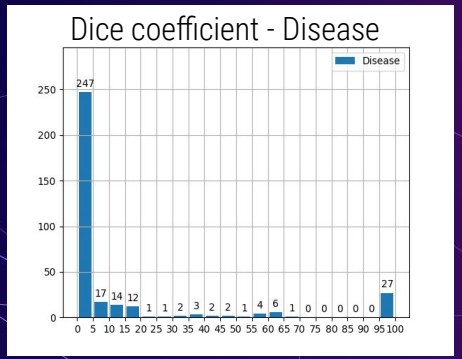
3 Experimentation & results

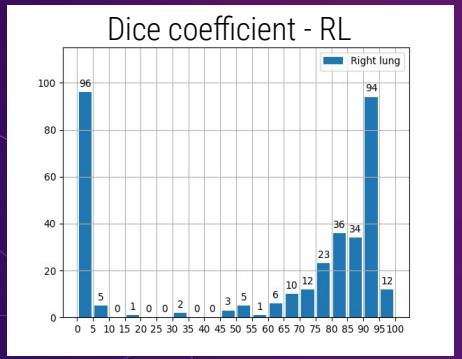
- **Experiment 5: Statistical Analysis of 0.00001 learning rate**

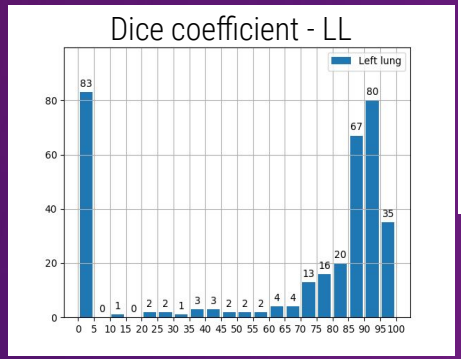


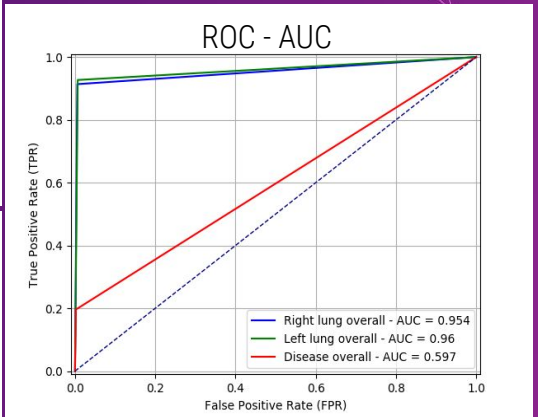




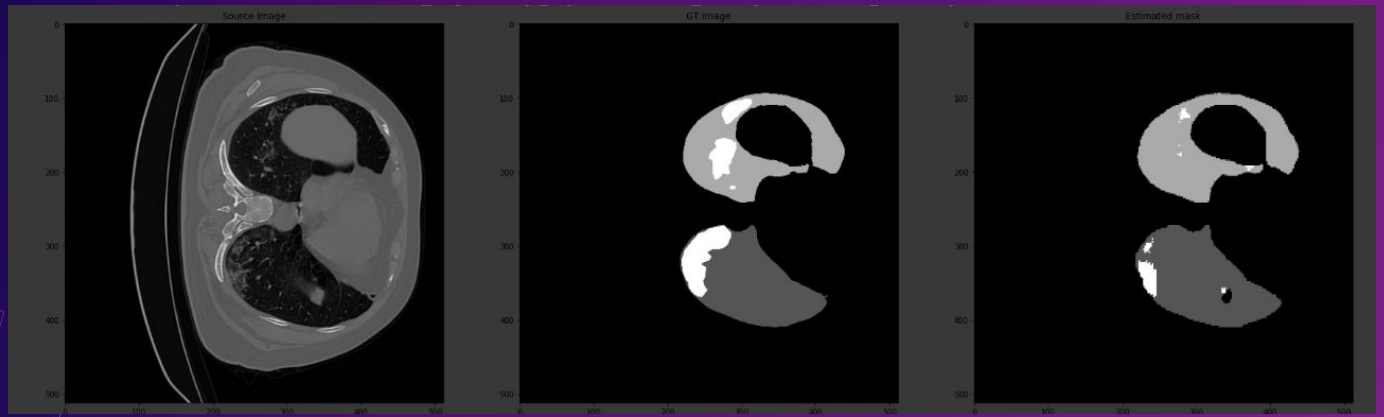






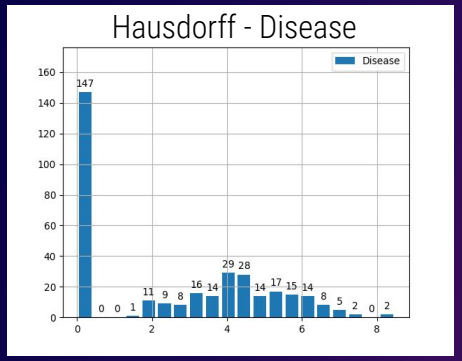


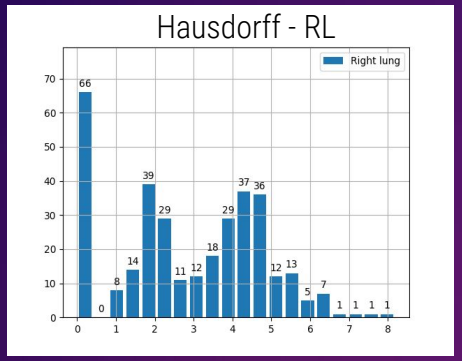
- **Experiment 6: Introducing inception module/layer to the U-Net model**

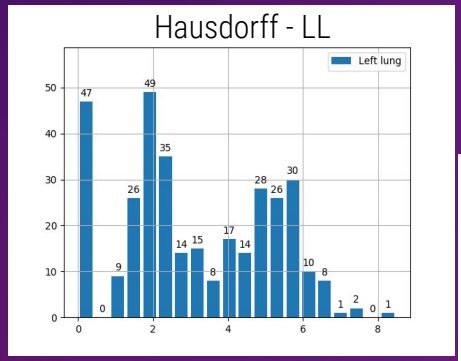


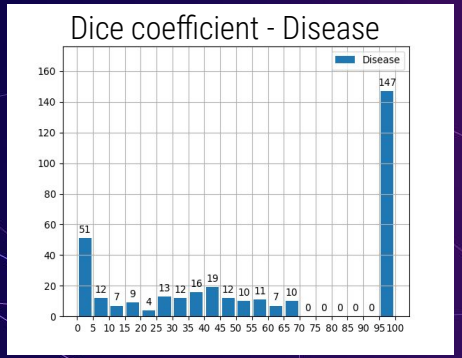
3 Experimentation & results

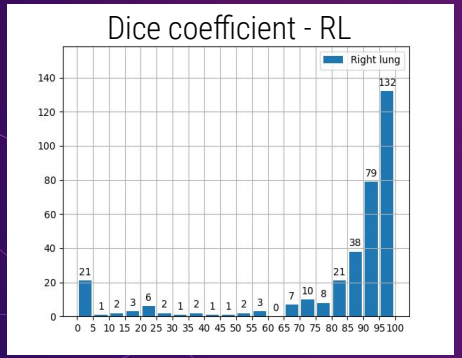
- **Experiment 6: Statistical Analysis of Inception Layer 1**

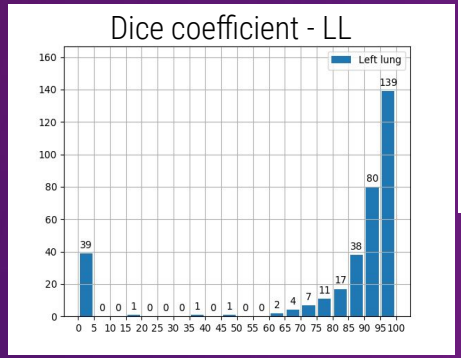


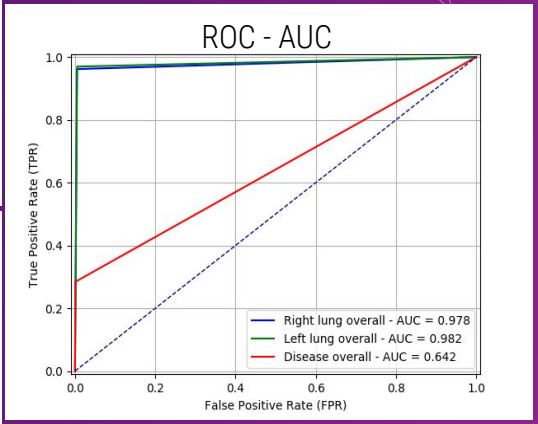












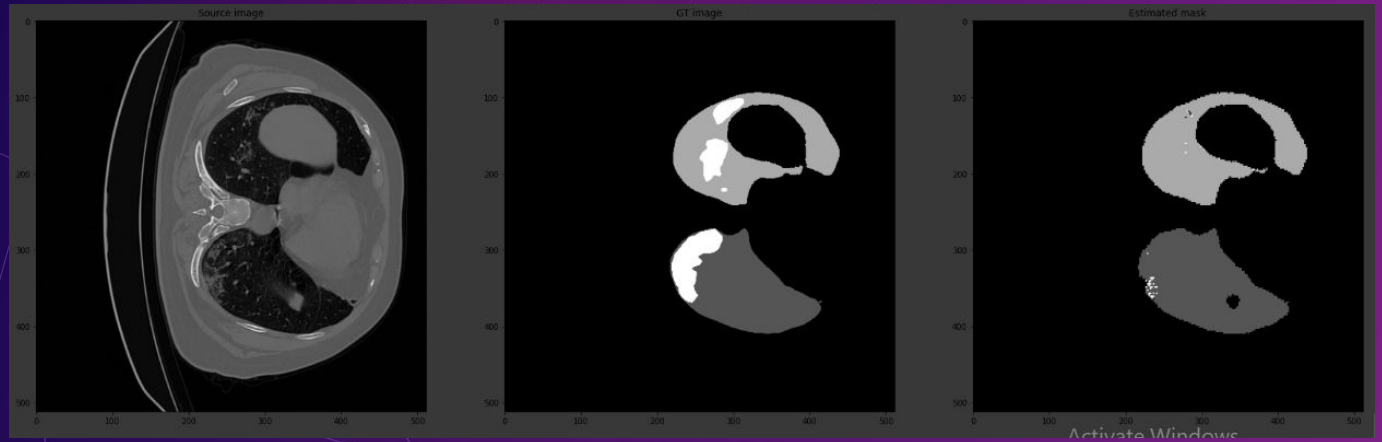
3 Experimentation & results

- So far, we saw that the inception module works well with U-Net model. The biggest challenge here is to decide the size of the filters for this module
- Also, the value of 0.001 for the learning rate increases the performance of the model
- Some overfitting might be happening
- Further experiments made for testing the performance of the model with different batch size and epochs value

- **Experiment 7: Reducing Epoch and increasing Batch size**

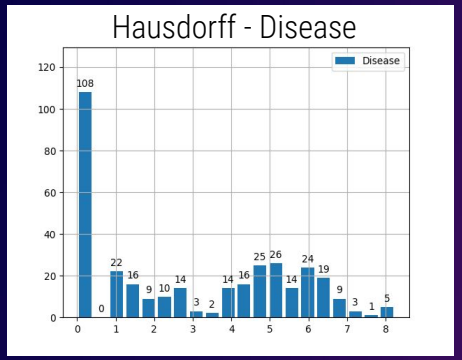
Experiment details

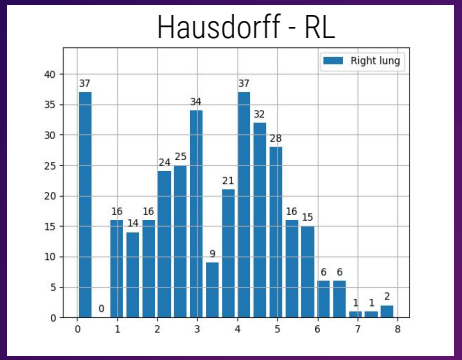
- Epoch = 4
- Batch size = 12

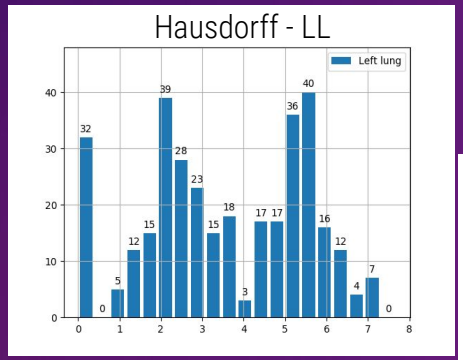


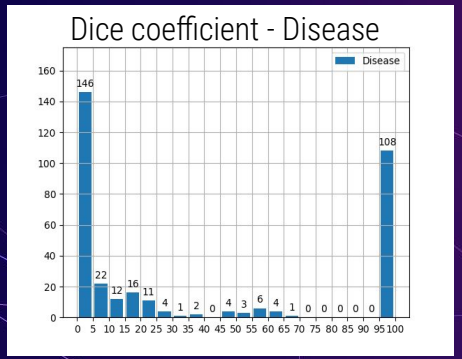
3 Experimentation & results

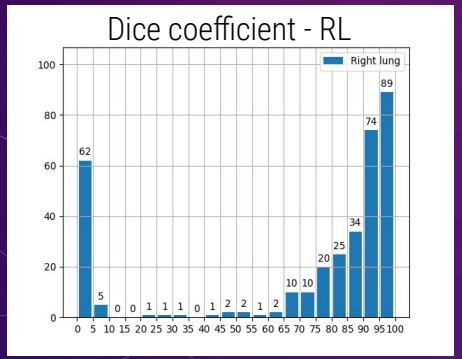
- **Experiment 7: Statistical Analysis of Reducing Epoch and increasing Batch size**

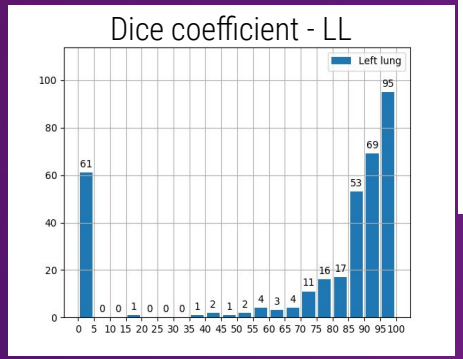


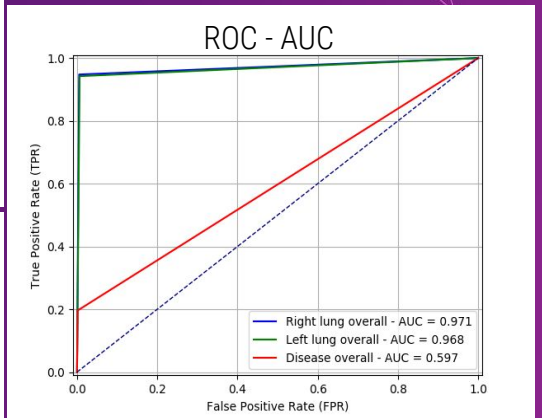




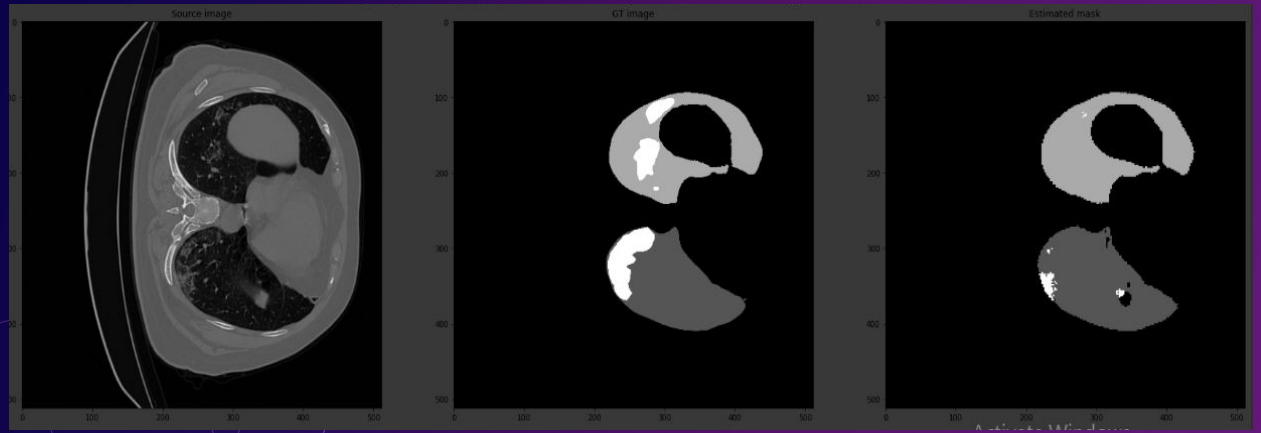








- **Experiment 8: Using categorical cross entropy for inception U-net model**

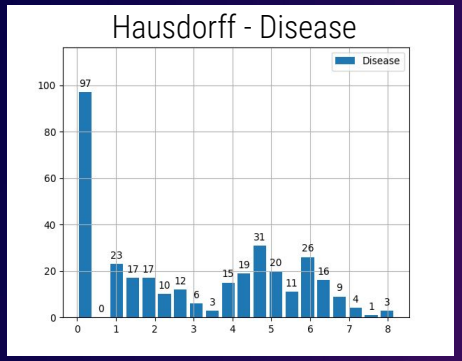


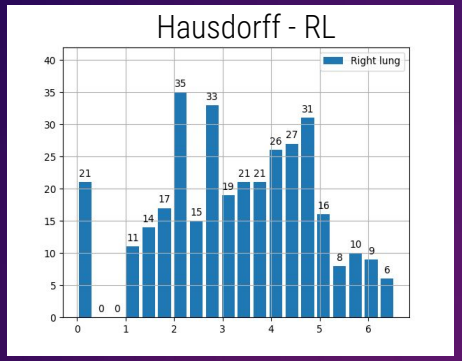
Experimental Details

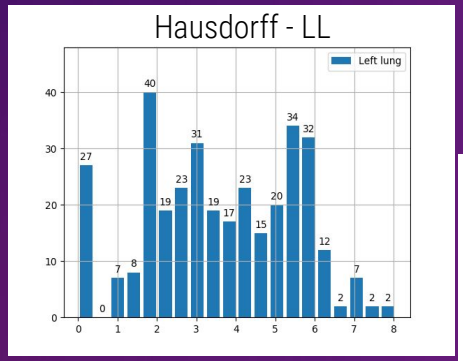
- Using Categorical cross entropy instead of binary cross entropy as a loss function
- Epochs = 5
- Batch size = 16

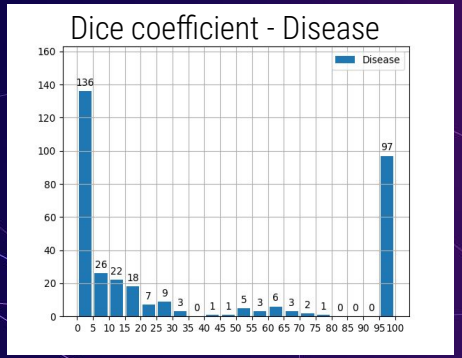
3 Experimentation & results

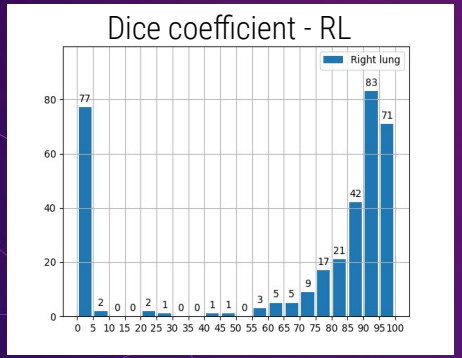
- **Experiment 8: Statistical Analysis of using categorical cross entropy for inception U-net model**

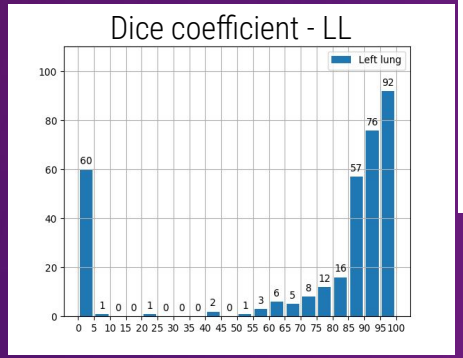


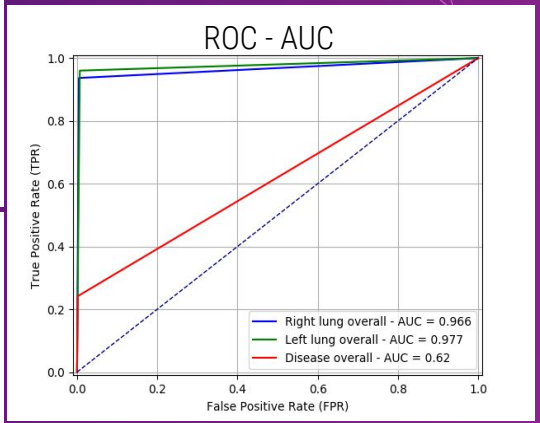




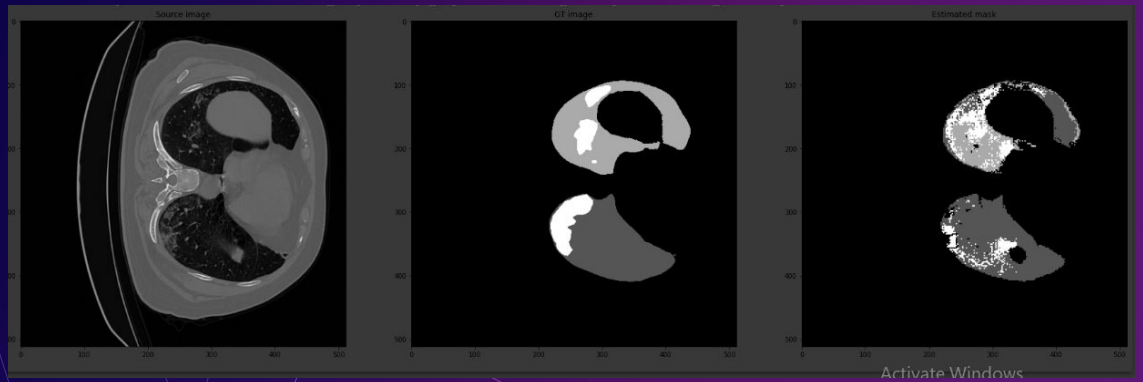








- **Experiment 9: Introducing second layer of Inception in U-net**

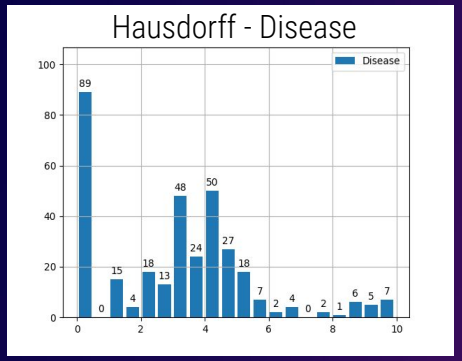


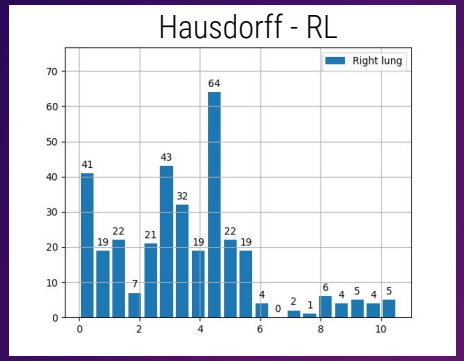
Experimental Details

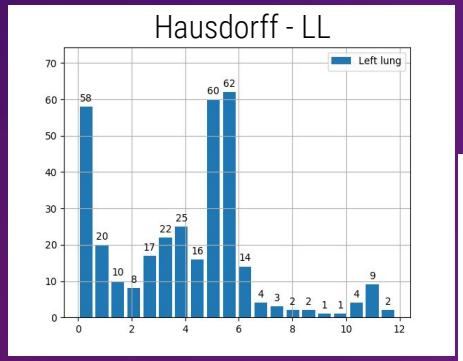
- Total Params: 8,990,692
- 2 layers of inception
- 2 drop out layers
- Epochs = 5
- Batch size = 12

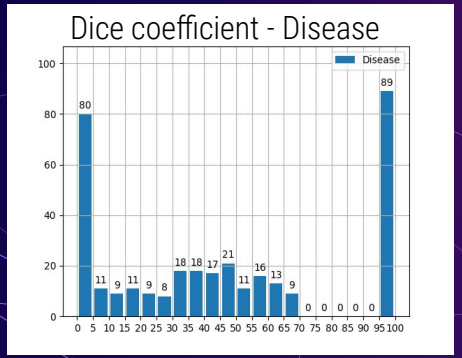
3 Experimentation & results

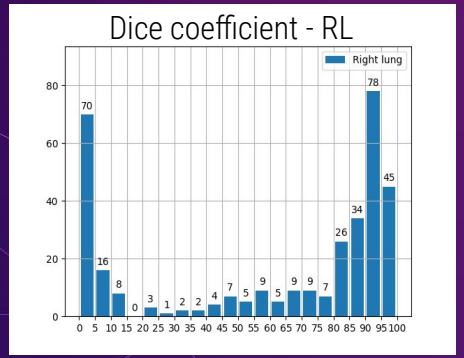
- **Experiment 9: Statistical Analysis of Introducing second layer of Inception in U-net**

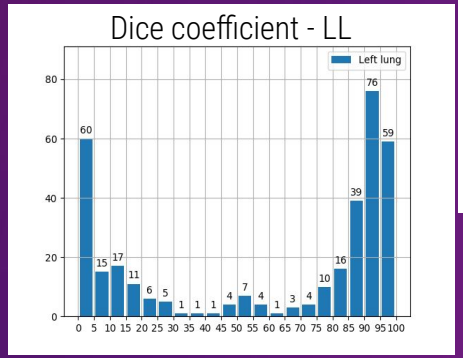


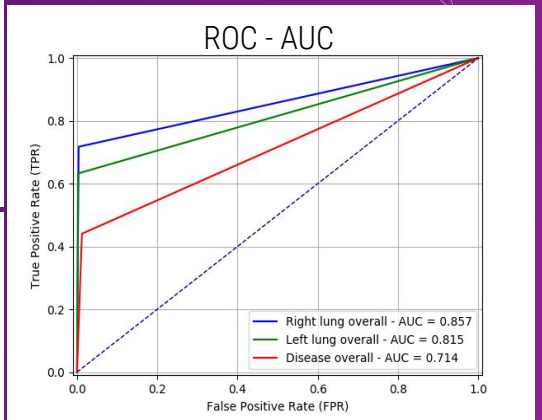




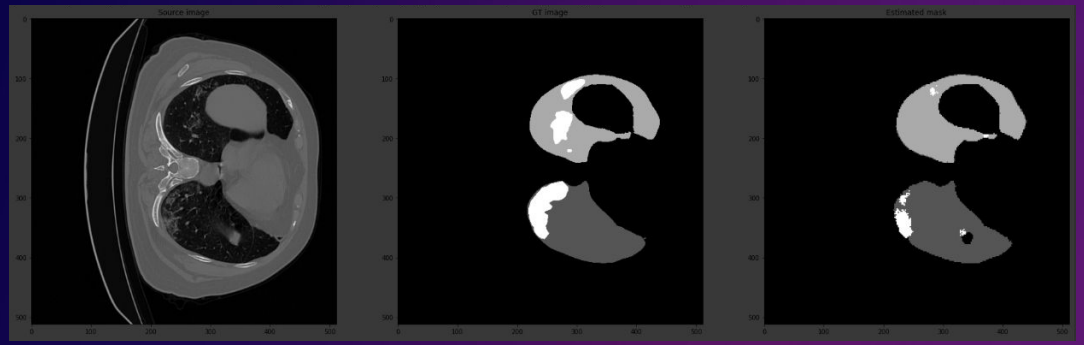








- **Experiment 10: Adding One inception layer and one DO (30%) in initial U-net**

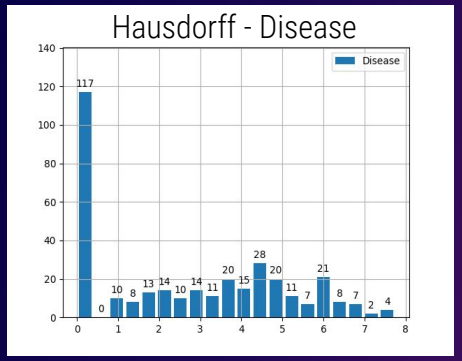


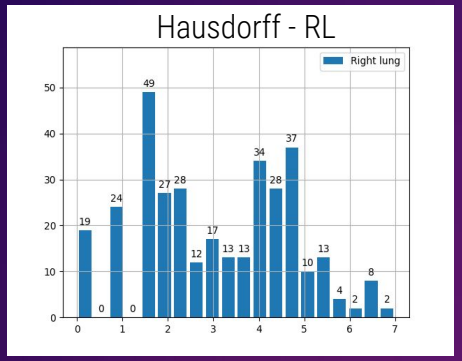
Experimental details

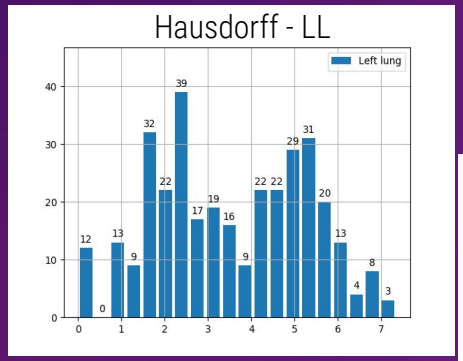
- 1 Drop out layer
- 1 Inception layer in unet
- Categorical cross entropy
- epochs = 8
- Batch size = 8
- Adam optimizer. LR 0.0001

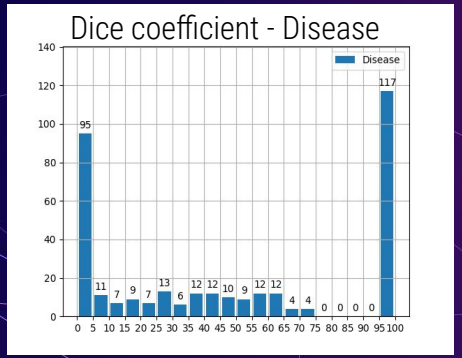
3 Experimentation & results

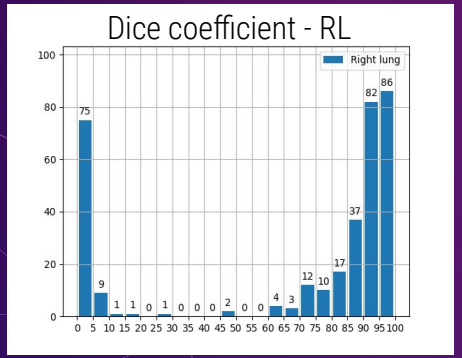
- **Experiment 10: Statistical Analysis of Adding One inception layer and one DO (30%) in initial U-net**

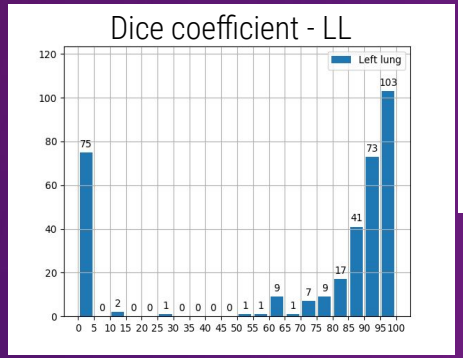


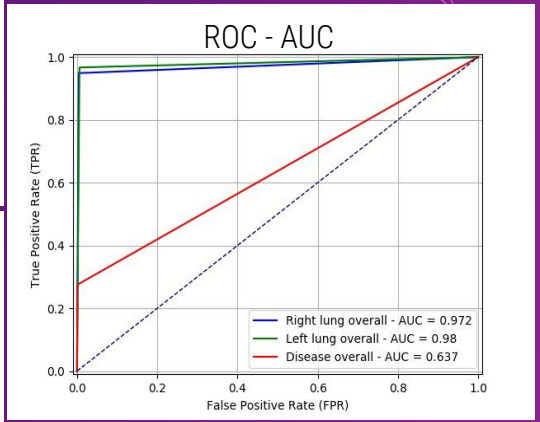






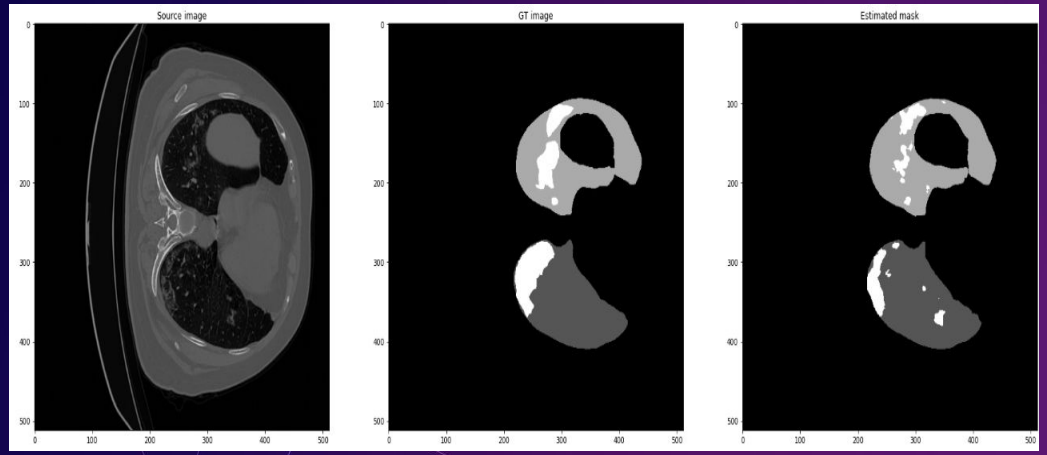






3 Experimentation & results

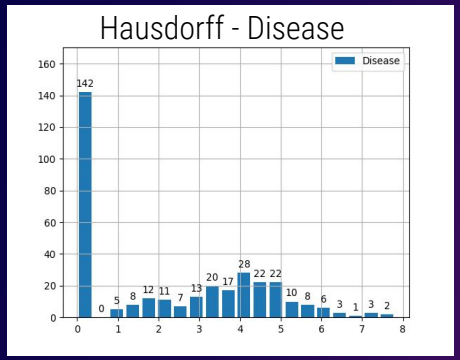
- **Experiment 11: Base U-Net model using EfficientNet pre-trained weights**

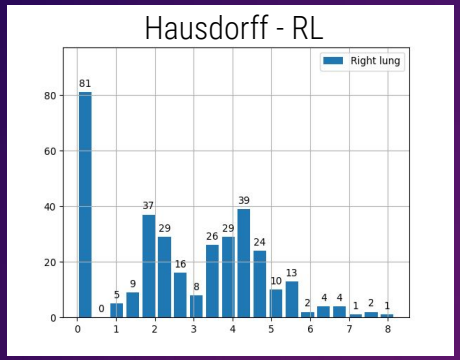


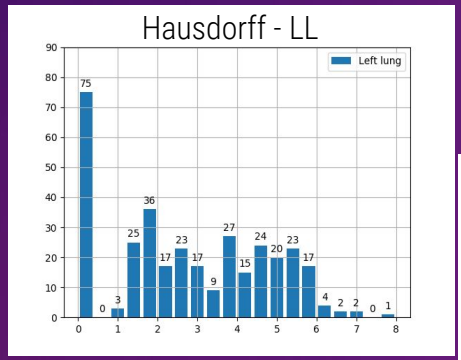
- **Optimizer: Adam - LR = 0.0001**
- **Activation Func.: Sigmoid**
- **LF: Categorical Cross Entropy**
- **Epochs: 10**
- **Batch size: 8**

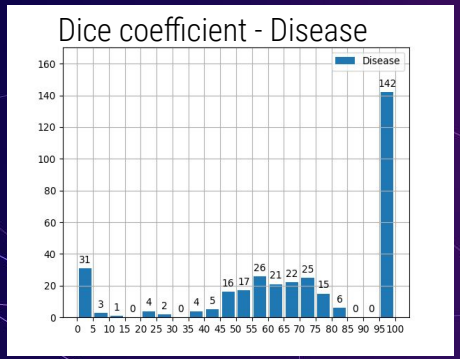
3 Experimentation & results

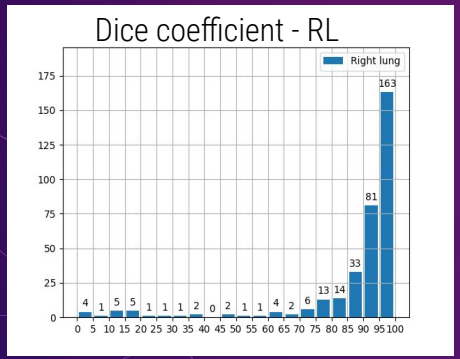
- **Experiment 11: Base U-Net model using EfficientNet pre-trained weights**

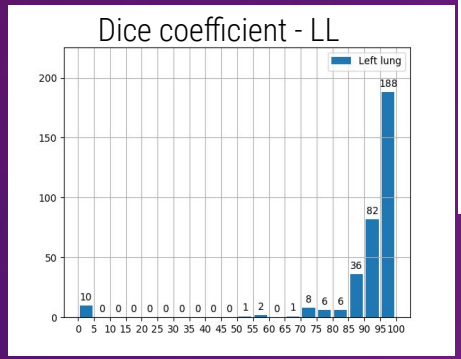


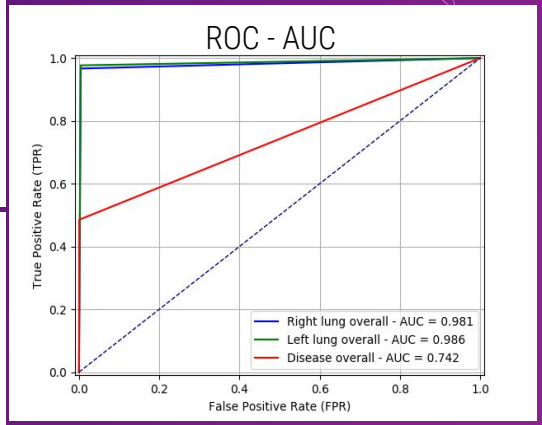












4 Conclusions & Further improvements

- The EfficientNet model performs as good as our custom model
- Since there are more pixels about the lungs than for the disease, the model performs better on them than in the disease area
- Even only 20 CT scans are provided, it is easy to get 99% accuracy for the lungs using deep learning
- Inception modules showed to improve the segmentation quality
- Further work should include:
  - Further studies of dropout layers are required. Interdependent learning
  - Data augmentation is strongly suggested
  - Batch normalization when using sigmoid activation function
  - Reduction of the background area at each image

Questions?

Thank you!