Mediastinal Lymph Node Detection using Deep Learning

Barak Hostel

Why is AI needed in India

- India has a doctor-patient ratio of 1:1674 against the WHO norm of 1:1000
- Quality diagnosis is thus not readily available to all and is usually quite expensive
- Prognosis can be a tedious process in clinical workflows, and the accuracy is only 60% for pathologists
- There is so much data out in the world that humans can’t possibly go through it all. Machines can use this to find patterns and predict more accurately than many trained professionals

AI in healthcare

AI is getting increasingly sophisticated at doing what humans do, but more efficiently, more quickly and at a lower cost.

Early Detection

AI is already being used to detect diseases, such as cancer, more accurately and in their early stages.

Decision Making

Using pattern recognition to identify patients at risk of developing a condition is another area where AI is beginning to take hold in healthcare.

Lymph node detection

Accurate Lymph Node detection plays a significant role in tumour staging, choice of therapy, and in predicting the outcome of malignant diseases such as lung cancer, lymphadenopathy, lymphoma, and inflammation. These pathologies can cause affected LN's to become enlarged, i.e., swell in size (> 1 cm diameter)

Clinical examination to detect lymph node metastases alone is tedious and error-prone due to the low contrast of surrounding structures in Computed Tomography (CT) and to their varying shapes, poses, sizes, and sparsely distributed locations

Parametric analysis of size, shape and contour, number of nodes and nodal morphology is critical when evaluating nodal disease

Methodology

We propose an efficient two-step method to automatically detect LNs in a patient's chest CT scans.

- Candidate Generation: This step is intended to give 3D coordinates of centroid of expected LNs which may contain a large number of False Positives that need to be rejected
- False Positive Reduction: This step focuses on reducing the False Positives produced in the previous step

Dataset

- Radiologists labeled a total of 388 mediastinal LNs in CT images of 90 patients
- The annotations include a folder for each patient with text files of voxel indices, physical coordinates and size measurements
- For a given CT volume, mask volume is generated (programmatically) by drawing an ellipsoid/sphere with center and shortest axis/diameter as per given annotations
- These mask volumes are used to train model in candidate generation step

Step 1: Pre-processing for Candidate Generation

- Each CT volume (512 X 512 X ~800) is resampled to a voxel size of 1mm X 1mm X 1mm
- A window of width 550 HU and level 25 HU is applied to resampled CT volume. All values less than -250 HU are mapped to -1000 HU, and those above 300 HU are mapped to 1000 HU
- The network is trained on axial slices of CT volume (Annotations in dataset included only shortest axis in the axial view of LN in accordance with RECIST criteria)

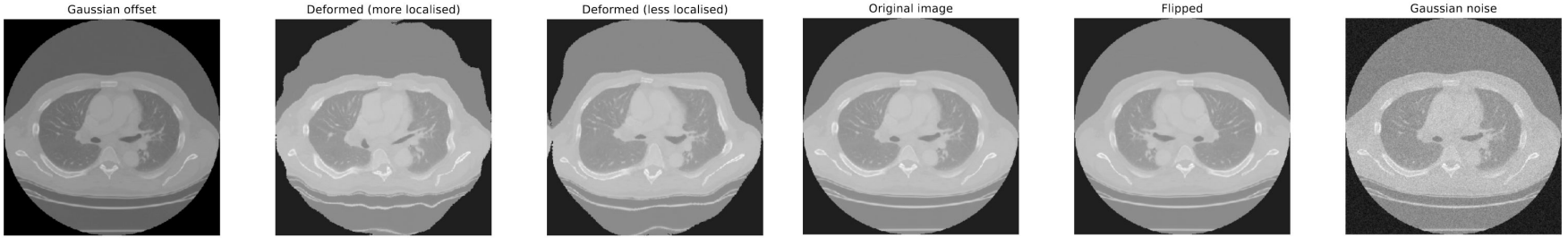
```
RESIZE_SPACING=[1, 1, 1]
resize_factor=spacing/RESIZE_SPACING
new_real_shape=img.shape*resize_factor
new_shape=np.round(new_real_shape)
real_resize=new_shape/img.shape
new_spacing=spacing/real_resize
```

```
# Reads the image using SimpleITK
itkimage = sitk.ReadImage(filename)

# Convert the image to a  numpy array first and then shuffle the dimensions to get axis in the order z,y,x
ct_scan = sitk.GetArrayFromImage(itkimage)

# Read the origin of the ct_scan, will be used to convert the coordinates from world to voxel and vice versa.
origin = np.array(list(reversed(itkimage.GetOrigin())))

# Read the spacing along each dimension
spacing = np.array(list(reversed(itkimage.GetSpacing())))

return ct_scan, origin, spacing
```

Step 1: Pre-processing for Candidate Generation

- Thereafter, axial slices with at least one annotated LN are cropped to 256 X 256 pixels about the center to contain mediastinum with a sufficient margin
- Random rotations, horizontal flips, shear, zoom (0.9-1.1), horizontal and vertical translations, Gaussian Noise, Gaussian Offset and Elastic Transform are used for data augmentation. Both input axial slice and corresponding mask are zipped together and augmented dynamically during training

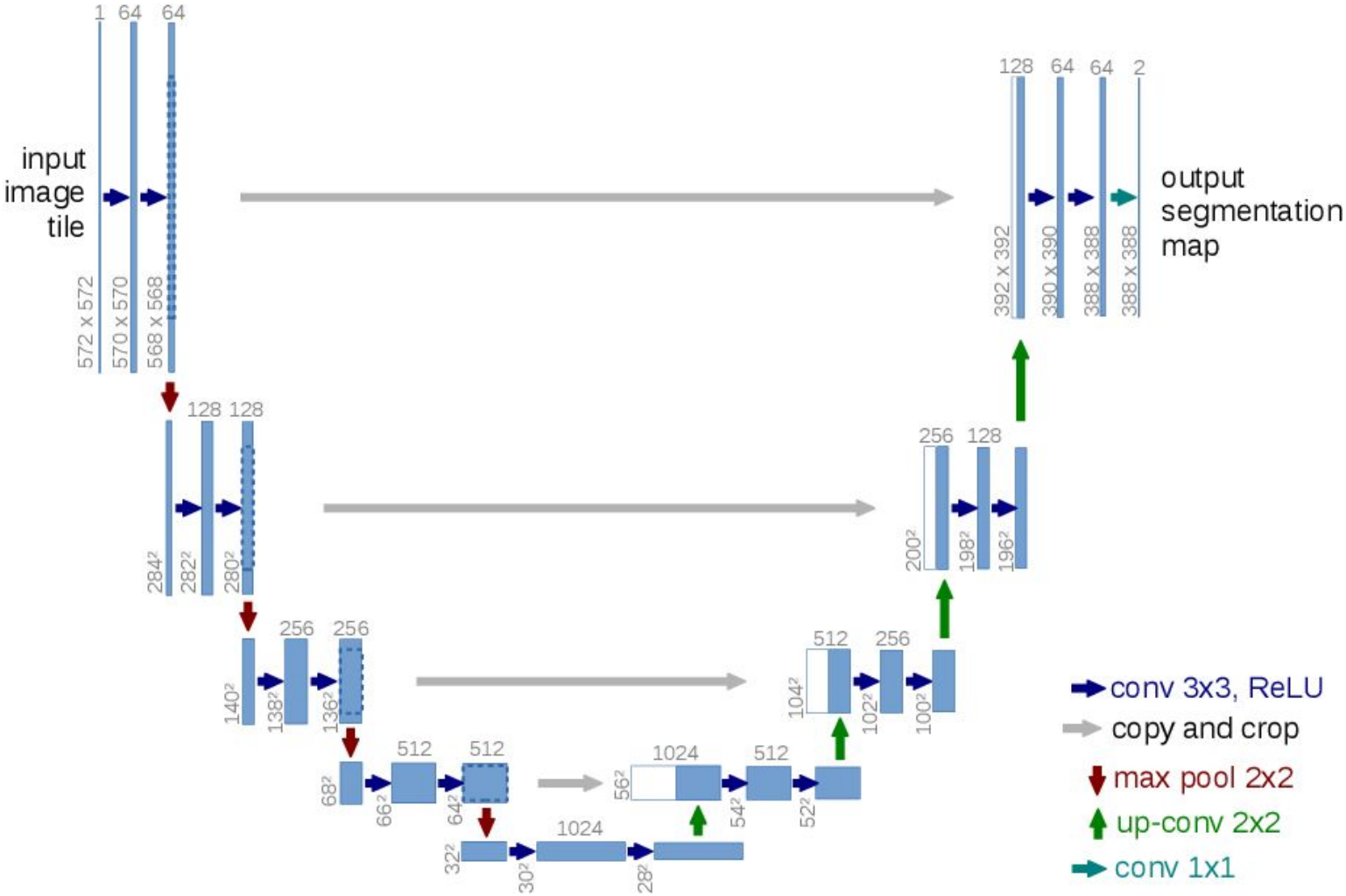
```
# Randomly flip the image along axis 1
t1_flipped1 = flip1(t1_norm.copy(), axis=2)
t1_flipped2 = flip1(t1_norm.copy(), axis=1)

# Add a Gaussian offset (independently for each channel)
t1_offset1 = add_gaussian_offset(t1_norm.copy(), sigma=0.1)
t1_offset2 = add_gaussian_offset(t1_norm.copy(), sigma=0.5)
t1_offset3 = add_gaussian_offset(t1_flipped1.copy(), sigma=0.5)
t1_offset4 = add_gaussian_offset(t1_flipped2.copy(), sigma=0.5)

# Add Gaussian noise
t1_noise1 = add_gaussian_noise(t1_norm.copy(), sigma=0.02)
t1_noise2 = add_gaussian_noise(t1_norm.copy(), sigma=0.05)
t1_noise3 = add_gaussian_noise(t1_flipped2.copy(), sigma=0.05)
t1_noise4 = add_gaussian_noise(t1_flipped1.copy(), sigma=0.05)


# Elastic transforms according to:
# [1] Simard, Steinkraus and Platt, "Best Practices for Convolutional
#     Neural Networks applied to Visual Document Analysis", in Proc. of the
#     International Conference on Document Analysis and Recognition, 2003.
t1_trans_low_s = elastic_transform(t1_norm.copy(), alpha=[1, 1e4, 1e4], sigma=[1, 8, 8])
t1_trans_high_s = elastic_transform(t1_norm.copy(), alpha=[1, 6e4, 6e4], sigma=[1, 16, 16])
```



Step 2: Candidate Generation

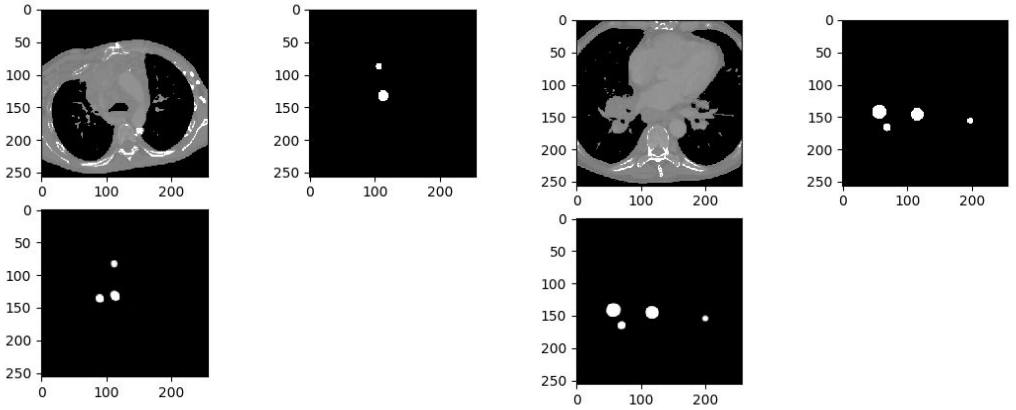
- We use a modified U-Net architecture for detecting LN candidates from mediastinal CT volumes
- The U-Net (Ronneberger et al., 2015) is a convolutional network architecture for fast and precise segmentation of biomedical images
- To avoid overfitting, we use dropout before max pooling operation that behaves as a regularizer when training the network
- After each convolution layer, we use batch normalization to reduce co-variance shift allowing each layer to learn by itself a little bit more independently of other layers

```
s = BatchNormalization()(input_img)
c1 = conv2d_block(s, n_filters=n_filters*1, kernel_size=3, batchnorm=batchnorm)
p1 = MaxPooling2D((2, 2)) (c1)
p1 = Dropout(dropout*0.5)(p1)

c2 = conv2d_block(p1, n_filters=n_filters*2, kernel_size=3, batchnorm=batchnorm)
p2 = MaxPooling2D((2, 2)) (c2)
p2 = Dropout(dropout)(p2)

c3 = conv2d_block(p2, n_filters=n_filters*4, kernel_size=3, batchnorm=batchnorm)
p3 = MaxPooling2D((2, 2)) (c3)
p3 = Dropout(dropout)(p3)

c4 = conv2d_block(p3, n_filters=n_filters*8, kernel_size=3, batchnorm=batchnorm)
p4 = MaxPooling2D(pool_size=(2, 2)) (c4)
p4 = Dropout(dropout)(p4)

c5 = conv2d_block(p4, n_filters=n_filters*16, kernel_size=3, batchnorm=batchnorm)

# expansive path
u6 = Conv2DTranspose(n_filters*8, (3, 3), strides=(2, 2), padding='same') (c5)
u6 = concatenate([u6, c4])
u6 = Dropout(dropout)(u6)
c6 = conv2d_block(u6, n_filters=n_filters*8, kernel_size=3, batchnorm=batchnorm)

u7 = Conv2DTranspose(n_filters*4, (3, 3), strides=(2, 2), padding='same') (c6)
u7 = concatenate([u7, c3])
u7 = Dropout(dropout)(u7)
c7 = conv2d_block(u7, n_filters=n_filters*4, kernel_size=3, batchnorm=batchnorm)

u8 = Conv2DTranspose(n_filters*2, (3, 3), strides=(2, 2), padding='same') (c7)
u8 = concatenate([u8, c2])
u8 = Dropout(dropout)(u8)
c8 = conv2d_block(u8, n_filters=n_filters*2, kernel_size=3, batchnorm=batchnorm)

u9 = Conv2DTranspose(n_filters*1, (3, 3), strides=(2, 2), padding='same') (c8)
u9 = concatenate([u9, c1], axis=3)
u9 = Dropout(dropout)(u9)
c9 = conv2d_block(u9, n_filters=n_filters*1, kernel_size=3, batchnorm=batchnorm)

outputs = Conv2D(1, (1, 1), activation='sigmoid') (c9)
```


input image tile
output segmentation map
1 64 64
572 x 572
570 x 570
568 x 568
128 128
284²
282²
280²
256 256
140²
138²
136²
512 512
68²
66²
64²
1024
32²
30²
28²
1024 512
56²
54²
52²
512 256
104²
102²
100²
256 128
200²
198²
196²
128 64 64 2
392 x 392
390 x 390
388 x 388
388 x 388
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

Step 2: Candidate Generation Results

- The output image is thresholded at 0.35-0.5, depending on the required sensitivity level (>=0.35 represents part of detected LN)
- The network is trained till the “test” dice coefficient reaches [0.60, 0.65] range
- A much higher dice coefficient may defeat the purpose of “Candidate Generation” to maintain high sensitivity while detecting LNs, which could otherwise adversely affect “False Positive Reduction” in the aforementioned pipeline



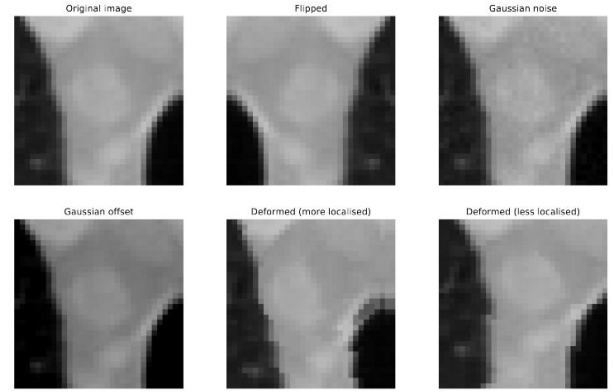
Candidate Generation Results (U-Net): Top Left→Axial Slice; Top Right→Mask; Bottom→Output

- The centroid of detected LN is found by calculating moments of contours in the binarized image and translating coordinates from cropped 256 X 256 image to original 512 X 512 axial view.
- Finally, we get complete list of coordinates indicating locations of potential LNs in CT volume

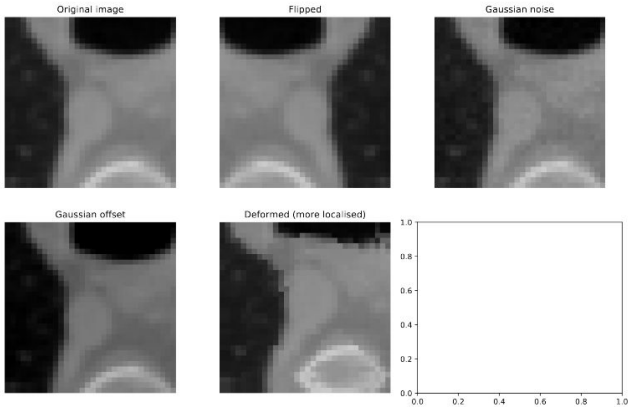
```
from cv2 import cv2
import numpy as np
import matplotlib.pyplot as plt
import sys

img=np.load("./result.npy", allow_pickle=True)
print(img.shape)

img[img<0.2]=int(0)
img[img>=0.2]=int(255)
img=np.array(img).astype('uint8')


# convert image to grayscale image
#gray_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

ret,thresh = cv2.threshold(img,127,255,0)

# find contours in the binary image
contours, hierarchy = cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
print(len(contours))
for c in contours:
   # calculate moments for each contour
   M = cv2.moments(c)

   # calculate x,y coordinate of center
   cX = int(M["m10"] / M["m00"])
   cY = int(M["m01"] / M["m00"])
   print(cX, cY)
```

Step 3: Pre-processing for False Positive Reduction

- About each candidate centroid, a 3D VOI (volume of interest) of shape 32 X 32 X 32 voxels is extracted from original resampled CT volume
- In order to increase training data variation and to avoid overfitting, each normalized VOI is also flipped (horizontal and vertical), translated, and rotated along a random vector in 3D space.
- Furthermore, each flipped, translated, and rotated VOI is augmented with Gaussian Noise, Gaussian Offset, and Elastic Transform different number of times depending on the scale of data augmentation
- Different sample and augmentation rates for positive and negative VOIs are used to obtain a reasonably balanced training set



Axial positive node



Axial negative node

Test Time Augmentation (TTA)

When classifying an unseen VOI, we make use of Test Time Augmentation (TTA). It involves creating multiple augmented copies of each VOI during testing, having the model make a prediction for each, then returning an ensemble of those predictions. Augmentations are chosen to give the model the best opportunity for correctly classifying a given VOI and reduce generalization error. Apart from the aforementioned VOI augmentations, we also use shear angle in counter-clockwise direction and zoom (0.8-1.2).

```
def tta_prediction(datagen, model, image, n_examples):
    samples=np.expand_dims(image, 0)
    it=datagen.flow(samples, batch_size=n_examples)
    yhats=model.predict_generator(it, steps=n_examples, verbose=0)
    summed=np.sum(yhats, axis=0)
    probs=np.mean(yhats, axis=0)
    return probs[1], np.argmax(summed)
```

```
datagen=ImageDataGenerator(**data_gen_args)
n_examples_per_image=21
yhats=[]
probs=[]
for i in range(x_test.shape[0]):
    prob, yhat=tta_prediction(datagen, model, x_test[i], n_examples_per_image)
    yhats.append(yhat)
    probs.append(prob)
```

Step 4: False Positive Reduction

Depending on the scale of data augmentation, we divide the analysis into two broad categories with sub-categories in each. Let 3x be the number of candidate LNs (obtained from Section 1), both positive and negative combined.

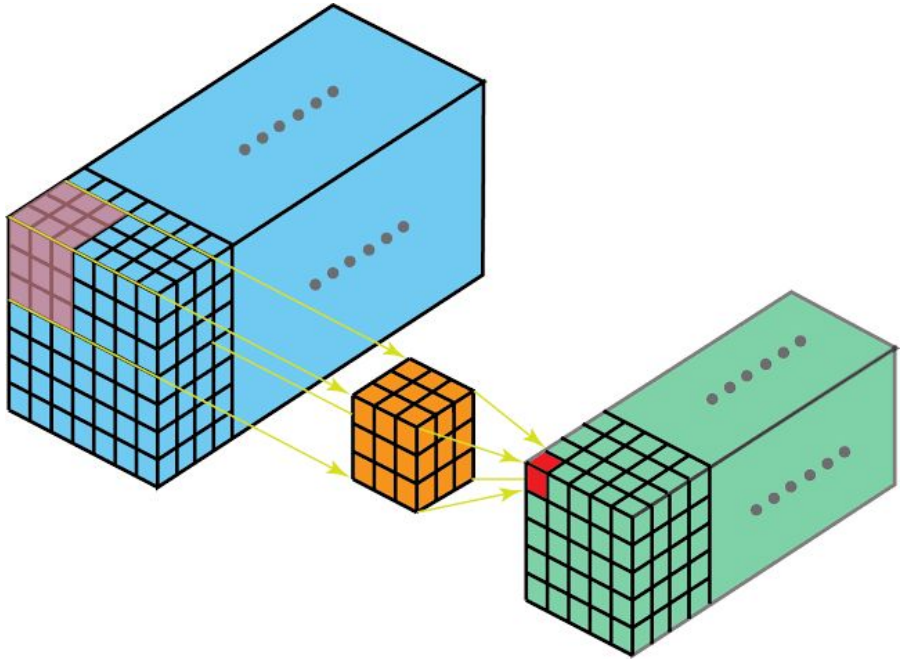
- 18x: 3x increased to 18x using data augmentation
- 34x: 3x increased to 34x using data augmentation

For each broad category following are the sub-categories:

- 2.5D-I: VOI decomposed into three orthogonal slices (axial, coronal and sagittal) through the center. 32 X 32 X 3 input to CNN;
- 2.5D-II: VOI decomposed into 12 slices (4 axial, 4 coronal and 4 sagittal) through the center. 32 X 32 X 12 input to CNN;
- 3D: VOI as input to 3D CNN (CNN with 3D convolutions);
- 2.5D-I TTA: 2.5D-I used with test time augmentation;
- 2.5D-I SVM: SVM trained on features extracted by CNN in 2.5D-I;
- 2.5D-II SVM: SVM trained on features extracted by CNN in 2.5D-II;
- 3D SVM: SVM trained on features extracted by 3D CNN in 3D;
- 2.5D-I TTA SVM: SVM trained on features extracted by CNN in 2.5D-I TTA;
- Axial: Axial slice through center as input (32 X 32 X 1) to CNN;
- Coronal: Coronal slice through center as input to CNN;
- Sagittal: Sagittal slice through center as input to CNN.

Step 4: False Positive Reduction

3D CNN uses 3D convolutions, which apply a 3-dimensional filter to the input volume, and the filter moves in three directions to calculate low-level representations. Their output is a 3-dimensional volume space such as cube or cuboid. A 2D convolution filter on the other hand, moves in two directions to give a 2-dimensional matrix as output. The 3D activation map produced during the convolution of a 3D CNN is necessary for analyzing medical data where temporal or volumetric context is important. The ability to leverage inter-slice context in a volumetric patch can lead to improved performance but comes with a computational cost as a result of the increased number of parameters



Step 4: False Positive Reduction

```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same', input_shape=x_train.shape[1:]))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(128, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(128, (3, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))


model.add(Flatten())
model.add(Dense(1024, name="SVM"))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(2))
model.add(Activation('softmax'))
```

CNN

```
## input layer
input_layer = Input((32, 32, 32, 1))

## convolutional layers
conv_layer1 = Conv3D(filters=32, kernel_size=(3, 3, 3))(input_layer)
conv_layer1 = BatchNormalization()(conv_layer1)
conv_layer1 = Activation('relu')(conv_layer1)
conv_layer2 = Conv3D(filters=32, kernel_size=(3, 3, 3))(conv_layer1)
conv_layer2 = BatchNormalization()(conv_layer2)
conv_layer2 = Activation('relu')(conv_layer2)

## add max pooling to obtain the most imformatic features
pooling_layer1 = MaxPool3D(pool_size=(2, 2, 2))(conv_layer2)
pooling_layer1 = Dropout(0.3)(pooling_layer1)

conv_layer3 = Conv3D(filters=64, kernel_size=(3, 3, 3))(pooling_layer1)
conv_layer3 = BatchNormalization()(conv_layer3)
conv_layer3 = Activation('relu')(conv_layer3)
conv_layer4 = Conv3D(filters=64, kernel_size=(3, 3, 3))(conv_layer3)
conv_layer4 = BatchNormalization()(conv_layer4)
conv_layer4 = Activation('relu')(conv_layer4)


pooling_layer2 = MaxPool3D(pool_size=(2, 2, 2))(conv_layer4)
pooling_layer2 = Dropout(0.3)(pooling_layer2)

## perform batch normalization on the convolution outputs before feeding it to MLP architecture
flatten_layer = Flatten()(pooling_layer2)

## add dropouts to avoid overfitting / perform regularization
dense_layer2 = Dense(units=512, activation='relu', name="SVM")(flatten_layer)
dense_layer2 = Dropout(0.5)(dense_layer2)
output_layer = Dense(units=2, activation='softmax')(dense_layer2)
```

3D CNN

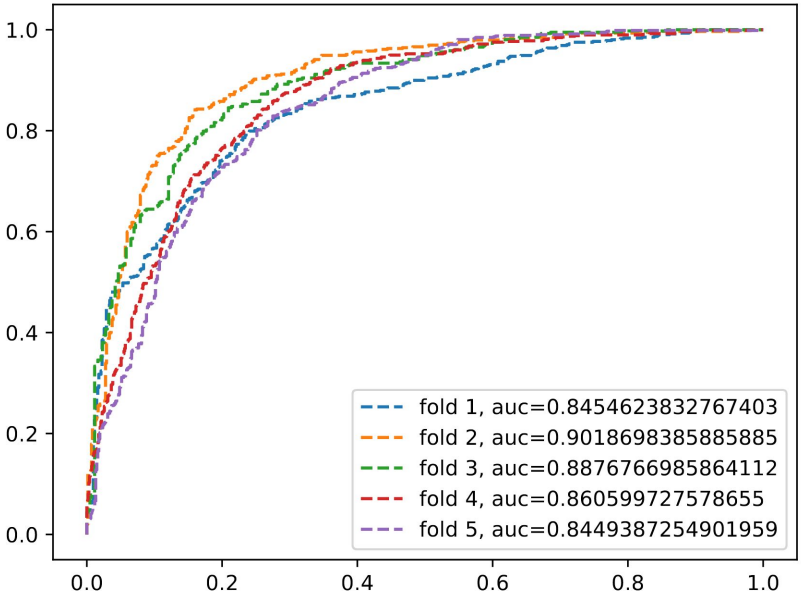
Results and Evaluation

BS = 32	PRECISION	RECALL	F1 SCORE	ROC AUC	FP/vol.
2.5D-I (18x)	0.74 (+/- 0.03)	0.82 (+/- 0.06)	0.77 (+/- 0.03)	0.86 (+/- 0.02)	3.08
2.5D-I (34x)	0.79 (+/- 0.04)	0.83 (+/- 0.03)	0.81 (+/- 0.01)	0.87 (+/- 0.01)	3.74
2.5D-II (18x)	0.78 (+/- 0.04)	0.78 (+/- 0.08)	0.78 (+/- 0.03)	0.86 (+/- 0.03)	2.61
2.5D-II (34x)	0.78 (+/- 0.06)	0.82 (+/- 0.06)	0.80 (+/- 0.30)	0.87 (+/- 0.01)	2.97
3D (18x)	0.79 (+/- 0.05)	0.80 (+/- 0.08)	0.80 (+/- 0.05)	0.88 (+/- 0.03)	2.31
3D (34x)	0.80 (+/- 0.04)	0.84 (+/- 0.03)	0.82 (+/- 0.03)	0.89 (+/- 0.03)	2.88
2.5D-I TTA (18x)	0.82 (+/- 0.04)	0.80 (+/- 0.04)	0.81 (+/- 0.01)	0.90 (+/- 0.01)	2.11
2.5D-I TTA (34x)	0.85 (+/- 0.05)	0.83 (+/- 0.05)	0.84 (+/- 0.01)	0.93 (+/- 0.01)	2.78
2.5D-I SVM (18x)	0.79 (+/- 0.04)	0.75 (+/- 0.08)	0.76 (+/- 0.03)	0.87 (+/- 0.02)	2.49
2.5D-I SVM (34x)	0.81 (+/- 0.04)	0.77 (+/- 0.03)	0.79 (+/- 0.01)	0.88 (+/- 0.01)	2.77
2.5D-II SVM (18x)	0.81 (+/- 0.06)	0.74 (+/- 0.10)	0.77 (+/- 0.04)	0.87 (+/- 0.03)	2.37
2.5D-II SVM (34x)	0.82 (+/- 0.06)	0.75 (+/- 0.02)	0.78 (+/- 0.02)	0.88 (+/- 0.03)	2.39
3D SVM (18x)	0.81 (+/- 0.05)	0.76 (+/- 0.09)	0.78 (+/- 0.05)	0.89 (+/- 0.03)	2.20
3D SVM (34x)	0.84 (+/- 0.05)	0.78 (+/- 0.05)	0.81 (+/- 0.03)	0.89 (+/- 0.03)	2.38
2.5D-I TTA SVM (18x)	0.79 (+/- 0.03)	0.75 (+/- 0.02)	0.77 (+/- 0.02)	0.87 (+/- 0.01)	2.24
2.5D-I TTA SVM (34x)	0.83 (+/- 0.03)	0.77 (+/- 0.03)	0.80 (+/- 0.01)	0.89 (+/- 0.01)	2.42

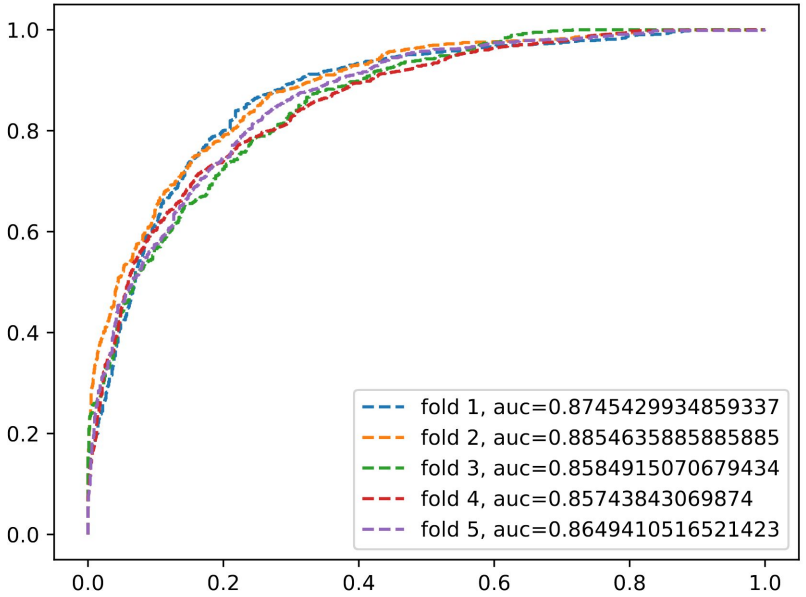
Results and Evaluation

To justify that 3 channels, 1 each of the 3 orthogonal views in 2.5D-I; 12 channels, 4 each of the 3 orthogonal views in 2.5D-II and VOI in 3D are not redundant, we also train CNN model with 1 channel input under each category of data augmentation containing 1 of the 3 orthogonal views at a time. As expected, in any of the single views, performance is not satisfactory with F1 score and ROC in [0.74, 0.76] and [0.82, 0.86] range, respectively. Recall is comparable with other corresponding sub-categories but with much larger FP/vol. In [8.48, 9.82] range

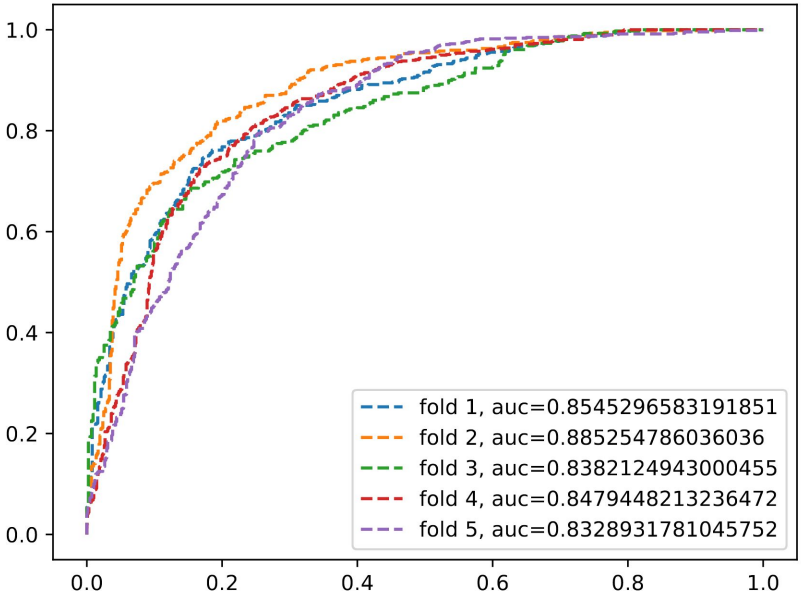
BS = 32	PRECISION	RECALL	F1 SCORE	ROC AUC	FP/vol.
Axial (18x)	0.67 (+/- 0.04)	0.83 (+/- 0.05)	0.74 (+/- 0.02)	0.84 (+/- 0.02)	9.82
Axial (34x)	0.72 (+/- 0.05)	0.82 (+/- 0.05)	0.76 (+/- 0.02)	0.86 (+/- 0.01)	9.20
Coronal (18x)	0.69 (+/- 0.06)	0.80 (+/- 0.04)	0.74 (+/- 0.03)	0.82 (+/- 0.02)	9.80
Coronal (34x)	0.73 (+/- 0.04)	0.81 (+/- 0.07)	0.76 (+/- 0.02)	0.84 (+/- 0.02)	9.15
Sagittal (18x)	0.72 (+/- 0.06)	0.78 (+/- 0.07)	0.74 (+/- 0.02)	0.83 (+/- 0.02)	8.56
Sagittal (34x)	0.75 (+/- 0.06)	0.78 (+/- 0.07)	0.76 (+/- 0.02)	0.85 (+/- 0.02)	8.48



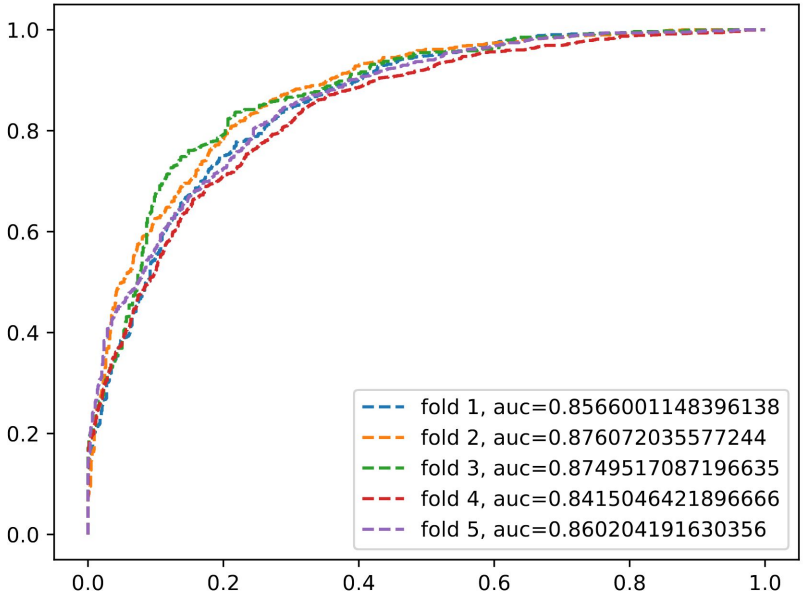
2.5D-I (18x)



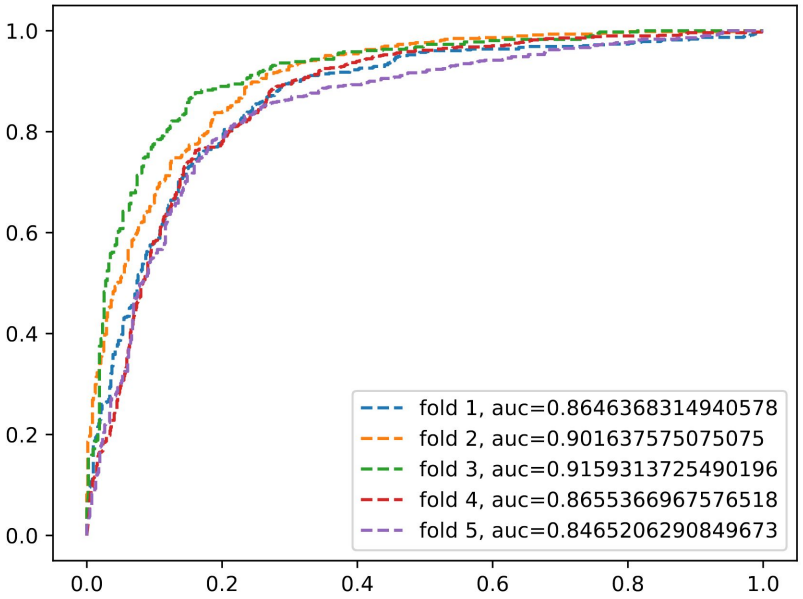
2.5D-I (34x)



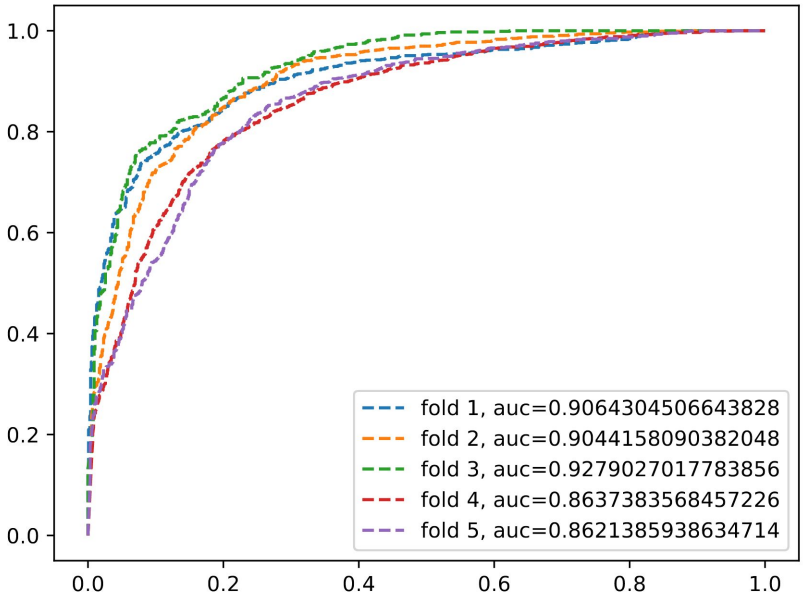
2.5D-II (18x)



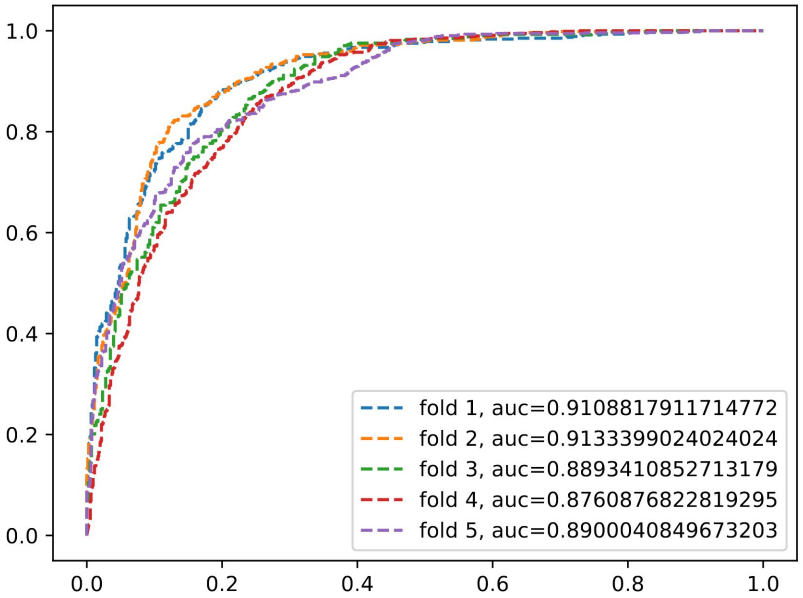
2.5D-II (34x)


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
fold 1, auc=0.8646368314940578
fold 2, auc=0.901637575075075
fold 3, auc=0.9159313725490196
fold 4, auc=0.8655366967576518
fold 5, auc=0.8465206290849673

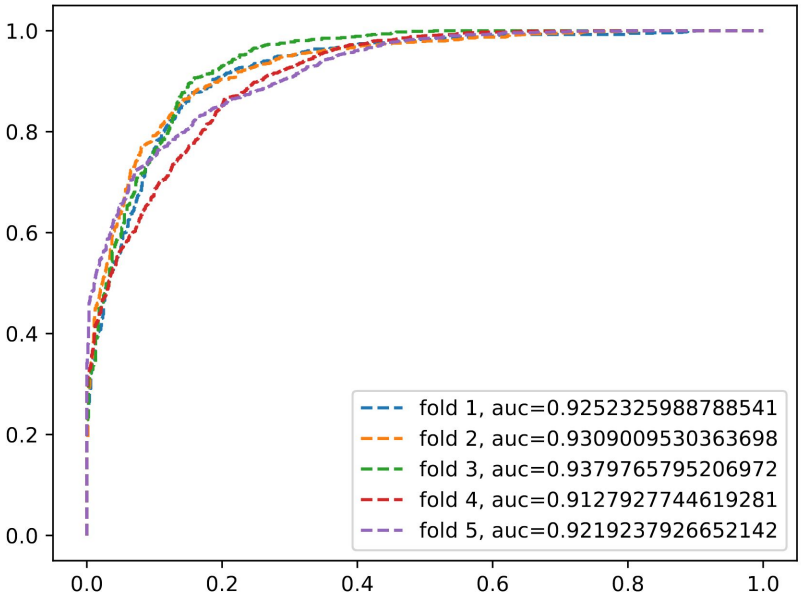
3D (18x)


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
fold 1, auc=0.9064304506643828
fold 2, auc=0.9044158090382048
fold 3, auc=0.9279027017783856
fold 4, auc=0.8637383568457226
fold 5, auc=0.8621385938634714

3D (34x)



2.5D-I TTA (18x)



2.5D-I TTA (34x)

Results and Evaluation

The performance improvement using more data augmentation (34x) shows that it is beneficial for CNN to have larger, more varied, and comprehensive datasets (which is coherent to the computer vision literature (Krizhevsky et al., 2012), (Zeiler and Fergus, 2014)).

3D CNN seems to perform better than 2D CNN in 2.5D-I and 2.5D-II because of 3D convolutions exploiting temporal/volumetric information. 2.5D-I with test time augmentation outperforms all other subcategories with maximum F1 score and AUC being 0.85 and 0.94, respectively, shown by 2.5D-I TTA (34x). AUC and ROC exhibit significant improvement in sensitivity levels at the range of clinically relevant FP/vol. rates.

(Roth et al., 2014) report 70% sensitivity at 3 FP/vol. in the mediastinum (3 fold cross validation on a dataset of 90 patients). (Liu et al., 2016) report 88% sensitivity at 8 FP/vol. (chest CT volumes from 70 patients with 316 enlarged mediastinal lymph nodes are used for validation), while our best approach achieves **80% sensitivity at 2.11 FP/vol.** in the mediastinum (dataset published by (Roth et al., 2014)).

Thus, the proposed method of candidate generation deploying U-Net followed by false positive reduction using different approaches involving CNN, 3D CNN, SVM, varying input representations of VOI, can be used for efficient detection of lymph nodes in a patient's chest CT scans. However, if 3D representations are exploited further for the volumetric context in serial medical CT data with current advancements in computing and memory hardware, results can be greatly improved.