TABLE OF CONTENTS

LIST OF ABBREVIATIONS

3D	Three-dimensional
ASIR	Adaptive Statistical Iterative Reconstruction
CNN	Convolutional Neural Networks
CSV	Comma-Separated Value
CT	Computed Tomography
GAN	Generative Adversarial Networks
IDE	Integrated Development Environment
LDCT	Low dose Computed Tomography
MBIR	Model Based Iterative Reconstruction Algorithms
ReLu	Rectified Linear Unit
TL	Transfer Learning
VGG	Visual Geometry Group

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

LUNG CANCER PREDICTION FROM LOW DOSE CT IMAGES

Decreasing the quality of images in a deadly type of cancer such as lung cancer can provide the benefit of reducing the amount of radiation. As a result of such a more health-friendly procedure, computer-aided applications and nodule classification algorithms that will support radiologists may pave the way for more successful results with healthier methods.

It has been observed that the use of 3D models in the evaluation of 3D images by CNN models has started to increase in the examined articles. Based on this, a 3D CNN model and the VGG11 version of a model called VGGNet were applied to the data set shared with the Luna16 challenge.

The data in the dataset was divided into 3D voxels and these data were multiplied. Other information about the dataset is given in the article.

In the results section, the metric results, confusion matrices and normalized confusion matrix information we obtained for both models are given.

Keywords: Convolutional Neural Networks, Computed Tomography, Three-Dimensional Voxel, Lung Cancer

ÖZET

DÜŞÜK DOZ BT GÖRÜNTÜLERİNDEN AKCİĞER KANSERİ TAHMİNİ

Akciğer kanseri gibi ölümcül kanser türlerinden birinde görüntülerin kalitesinin düşürülmesi radyasyon miktarlarının da azaltılması gibi bir fayda sağlayabilmektedir. Bu tür sağlığa daha faydalı bir işlem sonucunda bilgisayar destekli uygulamalar ile radyolojistlere destekte bulunacak nodül sınıflandırma algoritmaları daha başarılı sonuçların daha sağlıklı yöntemlerle alınmasının önünü açabilir.

3 boyutlu görüntülerin, CNN modellerince değerlendirilmesinde 3D modeller kullanılmasının incelenen makalelerde artmaya başladığı görülmüştür. Buradan yola çıkarak bir 3D CNN modeli ve VGGNet isimli bir modelin VGG11 isimli versiyonu Luna16 challange ile paylaşılan veri seti için uygulanmıştır.

Dataset içerisinde gelen verileri 3d voksellere ayırma ve bu verileri çoğaltma işlemleri yapılmıştır. Dataset ile ilgili diğer bilgiler makalede verilmiştir.

Sonuç bölümünde ise her iki model için aldığımız metrik sonuçları, confusion matrixleri ve normalize edilmiş confusion matrix bilgileri verilmiştir.

Anahtar Kelimeler:

Evrişimli Sinir Ağları, Bilgisayarlı Tomografi, Üç Boyutlu Voksel, Akciğer Kanseri

1
Introduction

In this chapter brief information about project will be given.

1.1 About CT Images

Computed tomography (CT) imaging is a radiological diagnostic method. The main goal of low-dose CT is to minimize patient's radiation exposure while obtaining diagnostic quality images.

One of the important developments in IT technologies is the integration of CNNs. CNNs are a type of deep learning algorithm that can process and analyze complex visual data; this makes them well-suited for image reconstruction and noise reduction in low-dose CT scans. Using CNNs, radiologists are assisted with supporting applications in the healthcare sector.

Thanks to the algorithms developed by CNNs, successful results can be obtained in CT procedures performed with low doses of radiation, as in high-resolution data. In this way; It can help diagnose a variety of diseases, including lung cancer, cardiovascular diseases and other abnormalities.

Figure 1.1 Lung CT Image Example [1].

At the same time, lower dose radiation application enables CT images to be obtained at more affordable prices, not only for health reasons, but also economically. This reduction in healthcare costs, without compromising diagnostic accuracy, could have

a significant impact on increasing early diagnosis, as more people can be served with similar budgets.

1.2 Data Set

Luna16 dataset was used in the project. It is a dataset derived from the LIDC/IDRI database, containing 888 CT images. Each radiologist made his own marking [1].

For 888 CT images, 36000 markings were made by radiologists. Nodules smaller than 3 mm were not evaluated sufficiently. Numbers 2290, 1602, 1186, and 777 were described by at least 1, 2, 3, or 4 radiologists, respectively. The number of candidates that 3 out of 4 radiologists call nodule is 1186. 11,500 nodules were smaller than 3 mm.

Luna challenge has two tracks first one is "complete nodule detection" and the second one is "false positive reduction". Within the scope of this project, work on the 2nd step was carried out.

The data set also contains annotations.csv file to be used for 'nodule detection' and candidates.csv (has two version) file to be used for 'nodule classification'.

candidatesV2.csv prepared with 5 algorithm. This algorithms are ISICAD, SubsolidCAD, LargeCAD, ETROCAD and M5L algorithms. As a result of these 5 detection algorithms, with the common acceptance of nodules within 5 mm, 1166 of 1186 nodules, 3 or more of which are accepted by the radiologist, will be given as input to our models. At the same time, images outside the lung area are excluded by these algorithms [2].

The complete data set is divided into 10 subsets.

This data set has access to files in raw and dcm format. Each data has its own header files with mhd extension. These files contain information about raw data such as spacing, center, dimension.

2
Preview

Lung cancer is one of the most lethal types of cancer compared to other cancers. It includes symptoms such as cough and chest pain. It is important at what stage the cancer is detected when seeking medical help.

There were an estimated deaths from lung cancer in 2020 is 1.8 million. It has been observed that lung cancer leading mortality rate with 18% [3].

2.1 Literature Review

This article written by Jun Ming at Wuhan Uni. at 2020. There is 100 CT image in their dataset. Also PSNR, RMSE, MAE and SSIM used for result metrics. The proposed method produced the following results. SSIM 0.9435, PSNR 33.69, MAE 1.975, RMSE 5.669 [4].

In 2019, Shiba Kuanar and a group of researchers reconstructed images with the Unsupervised Learning method in their article published in IEEE. GAN based network used to detect the CT images. They use augmentation to multiply images. Also they used SSIM, PSNR and RMSE metrics for results [5]. Their method produced the following results. PSNR 37.76, SSIM 0.944, RMSE 0.0092.

It has been stated that ASIR and MBIR are previously used methods. In the article researchers stated that network types such as GAN, CNN and RNN have been used before [6].

At this article which published in Springer shows an image denoising method. In this article, it is said that FD-VGG, WGAN-VGG, RED-CNN, SAGAN, SMGAN methods give better results than classical noise removal methods [7].

In this paper, Mr. Huang and his colleagues worked with Region-Based Convolutional Neural Networks (RCNN). With 2 methods; SWT was proposed to remove texture

noise, then NDCT model was used for data enhancement. They use 129 LDCT images in dataset [8].

At the article researchers propose a CT-specific perceptual loss scheme and apply it to train a LDCT denoiser. They use AUC and accuracy for the result metrics [9].

In the 2017 IEEE Transactions on Medical Imaging, They combine encoder-decoder and deconvolution networks for low-dose CT imaging [10].

In this article we see the comparison of 2-D and 3-D residual(ResNet) convolutional networks. PSNR, SSIM, RMSE and MAE metrics used for comparing [11].

In that article which published in 2019, researchers optimized performance with WGAN framework [12].

3
Feasibility

3.1 Technical Feasibility

This section consists of topics related to technical feasibility of the project.

3.1.1 Software Feasibility

It can run on every operating system. Created using Google Colab. Therefore, the Python codes in it are operating system independent. The reason why Google Colab was chosen is that it provides convenience with drive support in common use and keeps the code in the cloud.

Visual Studio Code is a program available in Python language developed by Microsoft. This was used to observe the codes taken from different Python compilers while they were translated into Colab Notebook.

- Programming Language: Python
- Tools: Google Colab

3.1.2 Hardware Feasibility

To develop the project, a computer that can open a web browser and has an internet connection is sufficient. Google Colab offers paid and free plans. Systems containing specialized graphics cards such as TPU can be used in train sections as needed.

3.2 Schedule Feasibility

It is expected to complete project in three months period by two students. Below is the Gantt chart of the topics.

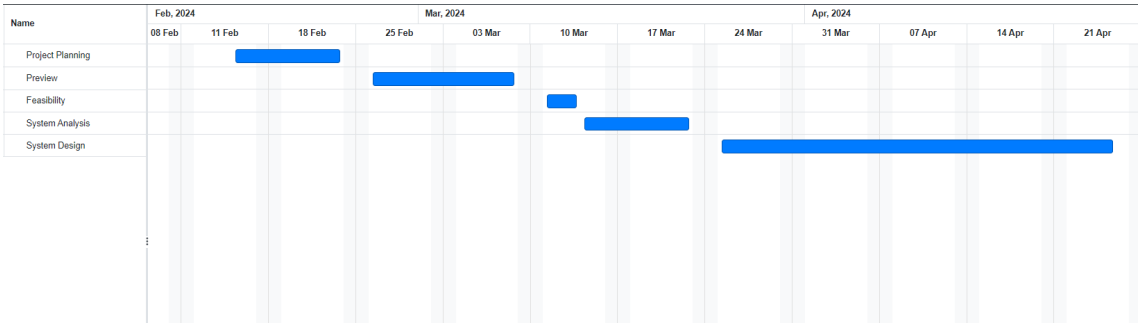

Figure 3.1 Gant Diagram

3.3 Legal Feasibility

It is believed that there is nothing against the law or the law regarding the project, as stated in the "Crimes in the field of informatics and the criminal sanctions to be applied in case of commission of these crimes" section of the Turkish Penal Code No. 5237, articles 243 and 246. The rules to be followed for the working environment and the libraries used have been made by adhering to the legal usage rights of the required application. The libraries used are open source libraries.

There is no legal requirement or liability for tools and environments used during development of the project. Python programming language and all the libraries used are open source.

3.4 Financial Feasibility

The total budget required to realize the project is divided into three items: "Hardware Requirements", "Software Requirements" and "Employees". The relevant amounts are listed in Table 3.1.

Since the Computer Engineer working on the project worked for 120 days, expense information was entered in the table based on 4 months' salary. Values were entered considering the salaries of new software developers. In order to be compatible with other expense entries in the table, it has been converted to US Dollars and entered. Salaries of Working Engineers are taken into account at 200 USD. And the calculations were made on the basis that 1 American Dollar is equal to 30 Turkish Liras.

Table 3.1 System Requirements

Hardware Requirements	**Cost**
Lenovo IdeaPad V15	$ 1200
Software Requirements	
Microsoft Windows 11	$ 150
Google Colab	$ 50
Employees	
Computer Engineer	$ 6000
Total	$ 7400

4
System Analysis

This chapter includes the project's goals, requirements and performance metrics.

4.1 Goals

The aim of the project is to classify lung scans with deep learning, observe the results on different models applied and reach satisfactory accuracy values for the models. In this way, by achieving higher success; It is to help perform CT procedures that can be performed more affordably and using less radiation.

4.2 Requirements

- Node extraction from dataset
- Augmenting the dataset. Especially positive data
- Splitting dataset to train, validation and test
- Implementing 3D CNN models
- Implementing CNN model with VGG architecture
- Training and testing CNN models with k-fold cross validation

5
System Design

This chapter describes the design models and architecture that will enable us to meet all project requirements.

5.1 Software Design

This section contains the software design steps of the system. These steps are explained under subheadings.

5.1.1 Data Preprocessing

5.1.1.1 Node Extraction

Under normal conditions, the entire input image is used for image classification operations. But for 3D CNN, this creates computational difficulties and it becomes difficult to receive the entire image as input due to GPU memory constraint. Smaller 3D pieces were extracted out of the images to get around this issue. Afterwards, these pieces are given as input to the network one by one.

The "candidates" csv file included in the dataset was used for this process. This file contains the serialUIDs, x y z coordinates and classes of the nodes. As an example row: 1.3.6.1.4.1.14519.5.2.1.6279.6001.100225287222365663678666836860, 68.42,-74.48,-288.7,0

In the node extraction process, data was saved as .npy extension by using the .csv file and center and spacing information in .mhd files, which is the header file of the raw data.

While the file contained 754975 candidates in total, There are 1186 nodes in the dataset. This situation causes imbalance between classes and data augmentation is required.

5.1.1.2 Data Augmentation

In the Augmentation process, the amount of data was increased by performing the following operations:

- Random Rotating and Mirroring
- Transpose
- Random Brightness Contrast
- Random Scale

Then processed data is divided into training and testing.

5.1.2 Model Design

5.1.2.1 3D CNN Model

In this model max-pooling is used for spatial downsampling reducing the size of image map and softmax is used in the output layer to produce class probabilities.

3D convolution is a mathematical operation used in processing three-dimensional data. In this operation, a filter or kernel is applied to an input volume, sliding across it to compute product weight input values at each position. This results in an output volume. Parameters such as size, stride, and padding affect the properties of the output volume. 3D convolutional neural networks are usually used in tasks involving volumetric data, such as volumetric image segmentation and medical image analysis, proving effective in capturing spatial dependencies.

```
def CNNT5_3D():
    model = keras.Sequential()
    model.add(Convolution3D(filters=8, kernel_size=(5, 5, 5), strides=1, activation='relu', input_shape=(32,32,32,1)))
    model.add(MaxPooling3D(pool_size=(2, 2,2), strides=2))
    model.add(Convolution3D(filters=16, kernel_size=(5, 5, 5),strides=1, activation='relu'))
    model.add(MaxPooling3D(pool_size=(2, 2,2), strides=2))
    model.add(Flatten())
    model.add(Dense(units=150, activation='relu'))
    model.add(Dropout(0.3))
    model.add(Dense(units=100, activation='relu'))
    model.add(Dropout(0.3))
    model.add(Dense(units=50, activation='relu'))
    model.add(Dense(units=2, activation = 'softmax'))

    return model
```

Figure 5.1 3D CNN Model Impl

5.1.2.2 VGGNet

VGGNet, which stands for Visual Geometry Group Network, is a convolutional neural network. It was put forward by the University of Visual Geometry Group (Oxford) in 2014. VGGNet allows successful results in complex images. It effectively organizes features and hierarchies in input images. There are versions such as VGG11 VGG16 VGG19 that will indicate the number of layers. The numbers in the version name here indicate the number of layers. In this way, flexibility is offered in the model. VGGNet uses 3x3 convolutional filters. After the convolution process, it uses the max pooling process to reduce the dimensions.

Convolutional layers in VGGNet are followed by fully connected layers. Typical steps involved in the VGGNet architecture include: The input image passes through a 3x3 layer. Feature extraction is performed here. Relu activation function is applied after all convolution layers. Afterwards, maxpooling used for reducing the size. Finally, the feature maps from the convolutional and pooling layers are flattened and passed through fully connected layers to perform classification tasks. With these steps, VGGNet completes the operations and provides a CNN model that can perform classification.

VGGNet architecture as can be seen in VggNet Figure 5.3.

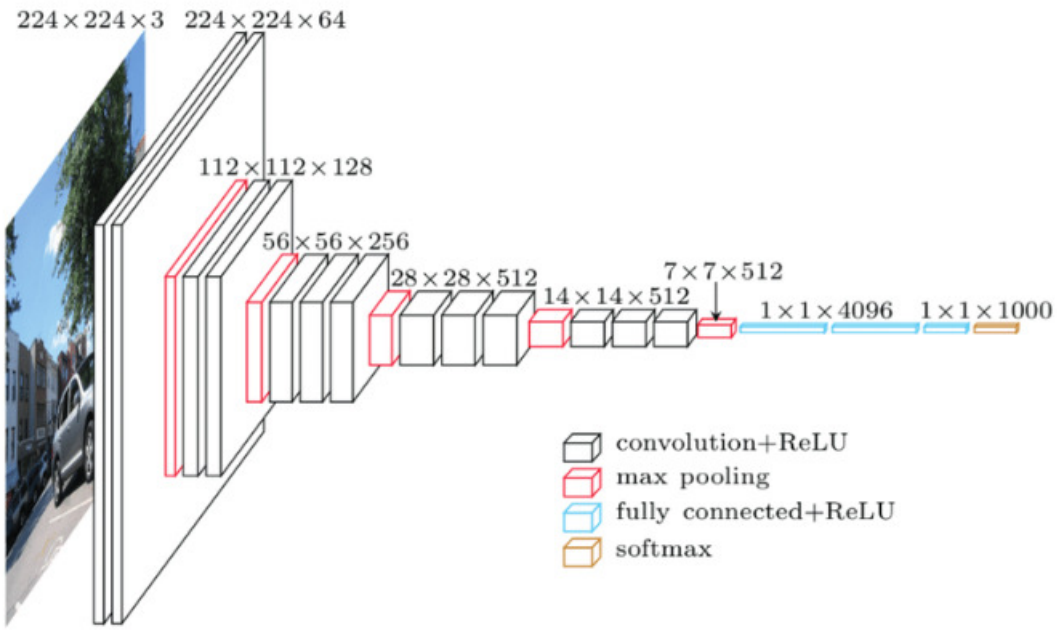

Figure 5.2 VggNet Figure

```
def VGG11_3D(weights_path=None):
    model = Sequential()
    model.add(ZeroPadding3D((0,1,1), input_shape=(32,32,32,1)))
    model.add(Convolution3D(64, (3,3,3), activation='relu', data_format='channels_last'))
    model.add(MaxPooling3D(pool_size=(1,2,2), strides=(1,2,2), data_format='channels_last'))

    model.add(ZeroPadding3D((0,1,1)))
    model.add(Convolution3D(128, (3,3,3), activation='relu', data_format='channels_last'))
    model.add(MaxPooling3D(pool_size=(1,2,2), strides=(1,2,2), data_format='channels_last'))

    model.add(ZeroPadding3D((0,1,1)))
    model.add(Convolution3D(256, (3,3,3), activation='relu', data_format='channels_last'))
    model.add(ZeroPadding3D((0,1,1)))
    model.add(Convolution3D(256, (3,3,3), activation='relu', data_format='channels_last'))
    model.add(MaxPooling3D(pool_size=(1,2,2), strides=(1,2,2), data_format='channels_last'))

    model.add(ZeroPadding3D((0,1,1)))
    model.add(Convolution3D(512, (3,3,3), activation='relu', data_format='channels_last'))
    model.add(ZeroPadding3D((0,1,1)))
    model.add(Convolution3D(512, (3,3,3), activation='relu', data_format='channels_last'))
    model.add(MaxPooling3D(pool_size=(2,2,2), strides=(2,2,2), data_format='channels_last'))

    model.add(ZeroPadding3D((0,1,1)))
    model.add(Convolution3D(512, (3,3,3), activation='relu', data_format='channels_last'))
    model.add(ZeroPadding3D((0,1,1)))
    model.add(Convolution3D(512, (3,3,3), activation='relu', data_format='channels_last'))
    model.add(MaxPooling3D(pool_size=(2,2,2), strides=(2,2,2), data_format='channels_last'))

    model.add(Flatten())
    model.add(Dense(4096, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(4096, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(2, activation='softmax'))

    return model
```

Figure 5.3 VggNet Model Imp

5.1.2.3 Relu

ReLU means Rectified Linear Unit and this activation function used at neural networks for getting better results in deep learning models.

ReLU has many advantages. Firstly, it offers computational efficiency. Secondly, it can introduce sparsity in the network. This means that some neurons can output zero, which can help prevent overfitting by adding regularization to the network. Lastly, ReLU offers ease of optimization. The derivative of ReLU is either 0 or 1, making it easier for optimization algorithms like gradient descent to converge during training.

Imagine a simple graph with an x-axis and a y-axis. The ReLU function starts at the origin (0,0) and extends upwards as a straight line with a positive slope.

For any input that is positive,function will output a value along the upward line. If the input value is negative or zero, the ReLU function will output zero.

This activation function introduces non-linearity into the neural network, which is essential for enabling the network to learn from the training data and make complex decisions.

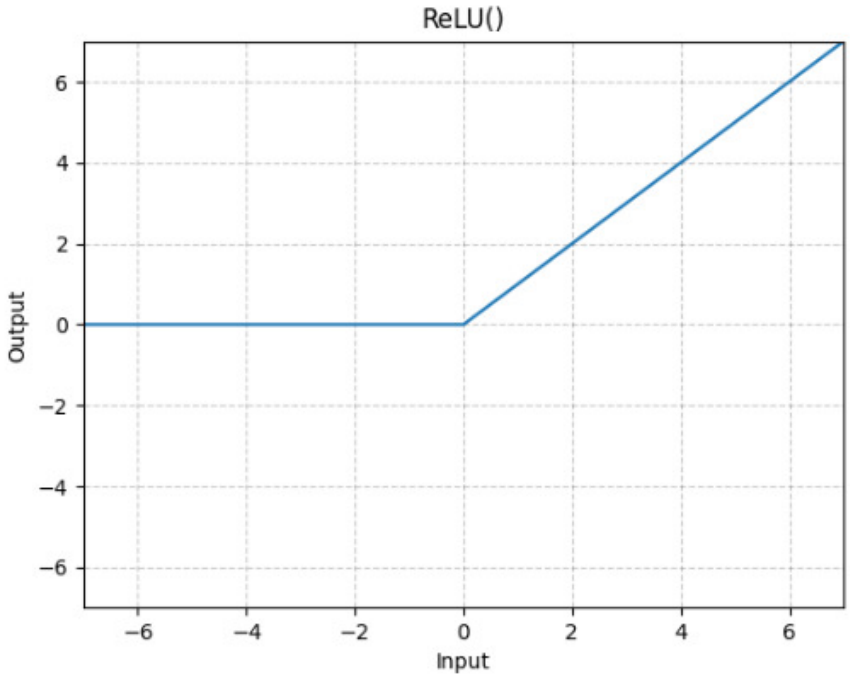

Figure 5.4 Relu

5.1.2.4 Maxpooling and Softmax

Maximum pooling preserves the features of an image while preserving its properties, such as width and depth. It reduces complexity by reducing the number of parameters. Spatial dimensions are added after reduction.

Softmax is an activation function used to produce probability distributions over multiple classes. It takes as input a vector of raw scores (also known as logits) and transforms them into probabilities that sum to one. The softmax activation function is used to calculate the probabilities of each class. The one with the highest probability is chosen as the output.

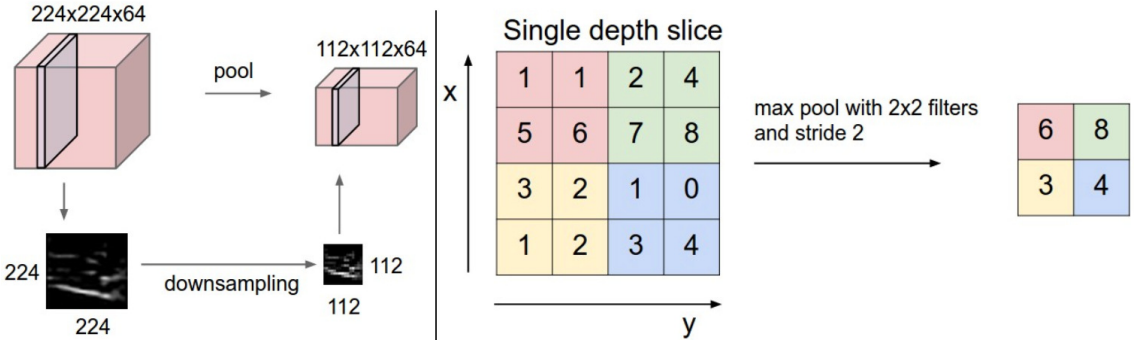

Figure 5.5 Maxpooling

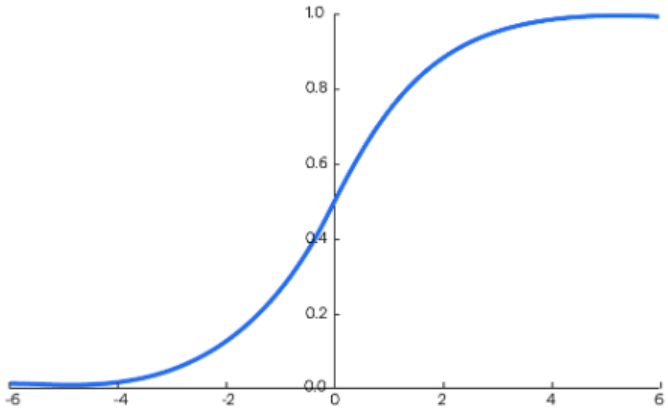

Figure 5.6 Softmax Function

5.1.3 Training and Validation

Training and test set will be detailed in the results section(at final report); specifying the training process, measurement metrics, optimization strategies and learning rates.

6
Implementation

In this chapter, the outputs of the working version of the project are shown.

The implementation of this project was made in Python3 language and Google Colaboratory as a web IDE.

6.1 Data Preprocessing

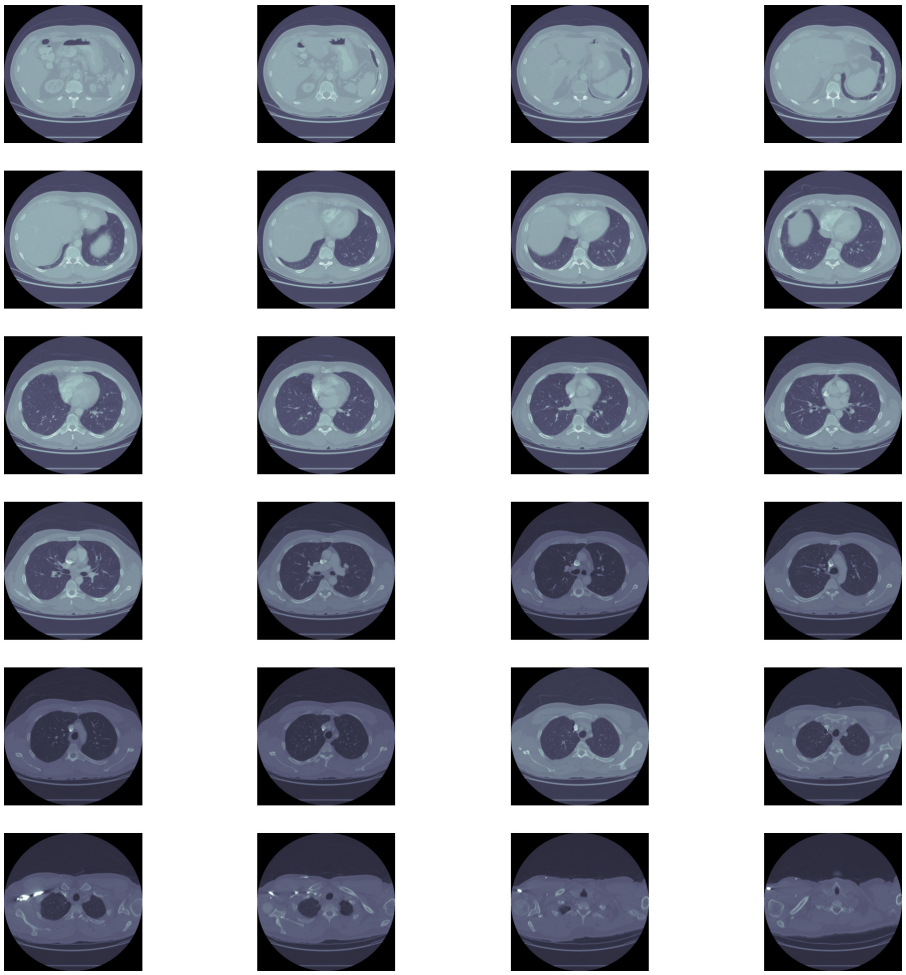

Figure 6.1 Sample Raw Data from LUNA16

```python
num_subsets = 10
image_size = 32

for subsetindex in tqdm(range(num_subsets)):
    luna_path = "/content/drive/MyDrive/Luna16/" # location of the dataset
    luna_subset_path = luna_path + "subset" + str(subsetindex) + "/"
    output_path = "/content/drive/MyDrive/output_path_32/"
    file_list = glob(luna_subset_path + "*.mhd") # header files of raw data
    luna_csv_path = "/content/drive/MyDrive/Luna16/"
    df_node = pd.read_csv('/content/drive/MyDrive/Luna16/candidates_V2.csv') # locations of the nodes
    print(df_node)
    df_node["file"] = df_node["seriesuid"].map(lambda file_name: [f for f in file_list if file_name in f][0] if any(file_name in f for f in file_list) else None)
    df_node = df_node.dropna()
    for fcount, img_file in enumerate(tqdm(file_list)): # looping over the image files
        temp_df = df_node[df_node["file"] == img_file] # get all nodules associate with file
        if temp_df.shape[0] > 0: # some files may not have a nodule
            img_array, origin, spacing = load_itk_version2(img_file) # origin and spacing of voxels in world coordinates (mm)
            index = 0
            for node_idx, row in temp_df.iterrows(): # go through all nodes
                x, y, z = row["coordX"], row["coordY"], row["coordZ"]
                label = row["class"]
                center = np.array([x, y, z]) # nodule center
                v_center = np.rint((center - origin) / spacing) # nodule center in voxel space
                v_center[0], v_center[1], v_center[2] = v_center[2], v_center[1], v_center[0] # convert x,y,z order to z,y,x
                node_cube = get_cube_from_img(img_array, v_center, image_size) # get cub (size of image_size)
                node_cube.astype(np.uint8)
                # save as npy file
                if label == 1:
                    filepath = output_path + "1/"
                    if not os.path.exists(filepath):
                        os.makedirs(filepath)
                    filename = str(subsetindex) + "_" + str(fcount) + "_" + str(index) + '_' + str(row['seriesuid']) + '_' + str(x) + '_' + str(y) + '_' + str(z)
                    np.save(filepath + filename + ".npy", node_cube)
                if label == 0:
                    filepath = output_path + "0/"
                    if not os.path.exists(filepath):
                        os.makedirs(filepath)
                    filename = str(subsetindex) + "_" + str(fcount) + "_" + str(index) + '_' + str(row['seriesuid']) + '_' + str(x) + '_' + str(y) + '_' + str(z)
                    np.save(filepath + filename + ".npy", node_cube)
                index += 1
```

Figure 6.2 Code Snippet of Node Extraction

```
                np.save(filepath + filename + ".npy", node_cube)
            index += 1
```

```
  0%|          | 0/1 [00:00<?, ?it/s]
0       1.3.6.1.4.1.14519.5.2.1.6279.6001.100225287222...  68.420000
1       1.3.6.1.4.1.14519.5.2.1.6279.6001.100225287222... -95.209361
2       1.3.6.1.4.1.14519.5.2.1.6279.6001.100225287222... -24.766755
3       1.3.6.1.4.1.14519.5.2.1.6279.6001.100225287222... -63.080000
4       1.3.6.1.4.1.14519.5.2.1.6279.6001.100225287222...  52.946688
...                                                   ...        ...
754970  1.3.6.1.4.1.14519.5.2.1.6279.6001.997611074084... -33.400000
754971  1.3.6.1.4.1.14519.5.2.1.6279.6001.997611074084...  56.236359
754972  1.3.6.1.4.1.14519.5.2.1.6279.6001.997611074084... -97.104221
754973  1.3.6.1.4.1.14519.5.2.1.6279.6001.997611074084... -65.470000
754974  1.3.6.1.4.1.14519.5.2.1.6279.6001.997611074084...  -3.100000

            coordY      coordZ  class
0       -74.480000 -288.700000      0
1       -91.809406 -377.426350      0
2      -120.379294 -273.361539      0
3       -65.740000 -344.240000      0
4       -92.688873 -241.067872      0
...            ...         ...    ...
754970  -64.200000 -115.560000      0
754971   70.352400 -203.446236      0
754972   55.738289 -203.879785      0
754973   59.670000 -136.370000      0
754974  -10.660000  -27.470000      0

[754975 rows x 5 columns]

  0%|          | 0/89 [00:00<?, ?it/s]
  1%|          | 1/89 [02:11<3:12:37, 131.33s/it]
  2%|▏         | 2/89 [02:29<1:33:40, 64.60s/it]
  3%|▎         | 3/89 [02:40<57:47, 40.32s/it]
  4%|▍         | 4/89 [02:55<42:55, 30.29s/it]
  6%|▌         | 5/89 [03:07<32:59, 23.56s/it]
  7%|▋         | 6/89 [03:27<30:49, 22.29s/it]
  8%|▊         | 7/89 [03:35<24:19, 17.80s/it]
  9%|▉         | 8/89 [03:56<25:22, 18.79s/it]
 10%|▉         | 9/89 [04:11<23:19, 17.49s/it]
 11%|█         | 10/89 [04:27<22:46, 17.30s/it]
 12%|█▏        | 11/89 [04:46<23:09, 17.81s/it]
```

Figure 6.3 Output of Node Extraction

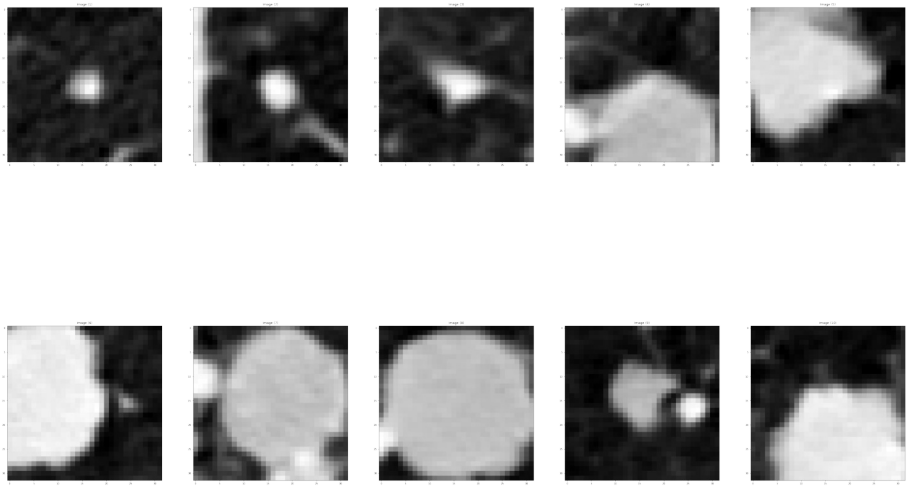

Figure 6.4 Sample Nodule Images from LUNA16 Candidates List

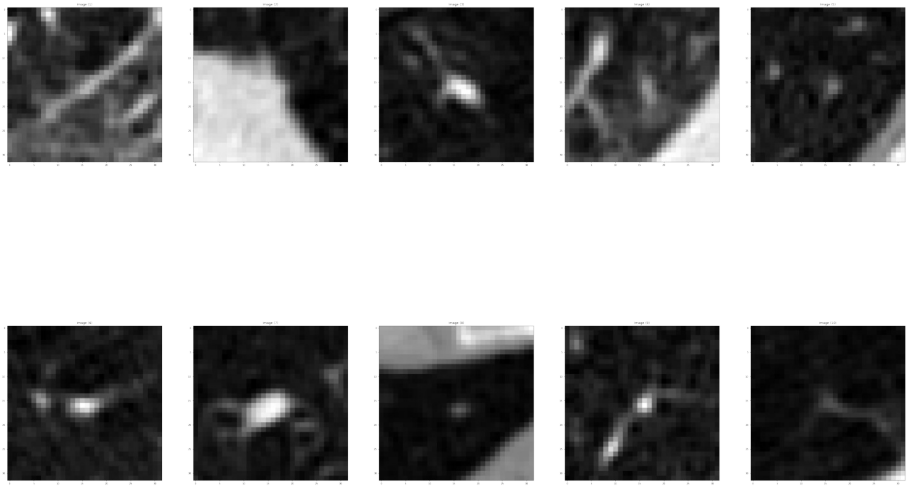

Figure 6.5 Sample Nonnodule Images from LUNA16 Candidates List

```
augm = Compose([
        RandomRotate90(),
        Transpose(),
        RandomBrightnessContrast(brightness_limit=0.4, contrast_limit=0.4, p=0.8),
    VerticalFlip(p=0.7),HorizontalFlip(p=0.7),
        RandomScale(scale_limit=0.3)
    ])
```

Figure 6.6 Augmentation Parameters

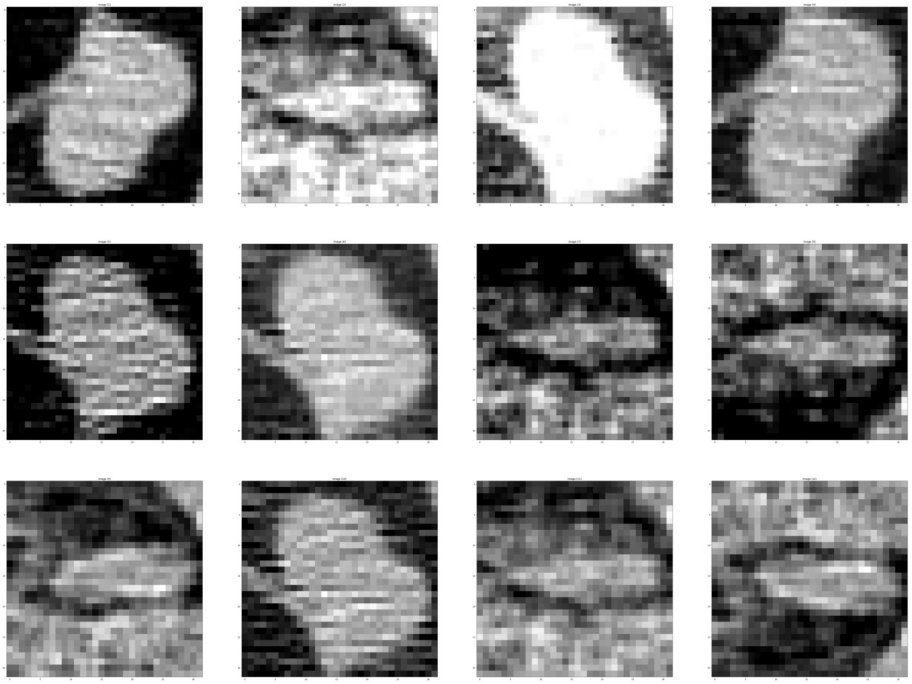

Figure 6.7 Sample Augmentation Output

6.2 Model Implementation

The model which implemented for 3D CNN and VggNet shared below.

CNN_3D

Train a model

```
opt = keras.optimizers.RMSprop(lr=0.0001, rho=0.95)
model = CNNT5_3D()

model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])

callback = [EarlyStopping(monitor='val_loss', patience=7),
            ReduceLROnPlateau(patience=5, verbose=1)]


history = model.fit(x=X_train_range, y=Y_train, epochs=30, validation_data=(X_test_range, Y_val),
          batch_size=32, callbacks=callback)
```

Figure 6.8 3D CNN Model

```
opt = keras.optimizers.Adam(learning_rate=0.0001)
model = VGG11_3D()

model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])

callback = [EarlyStopping(monitor='val_loss', patience=7),
            ReduceLROnPlateau(patience=5, verbose=1)]


history = model.fit(train_generator,
                    steps_per_epoch=len(candidates_train) // batch_size,
                    epochs=30,
                    validation_data=val_generator,
                    validation_steps=len(candidates_val) // batch_size,
                    callbacks=callback)
```

Figure 6.9 VggNet Model

6.3 Training

6.3.1 3D CNN Model

For the 3D CNN Model, 1 subset was used for testing, 1 subset for validation and finally 8 subsets for training. The metric results of that model for each epoch shared below.

Table 6.1 Epoch Results Table

Model	*Loss*	*valLoss*	*Acc*	*valAcc*
Epoch 1	0.64	0.54	0.64	0.74
Epoch 2	0.46	0.44	0.78	0.80
Epoch 3	0.35	0.43	0.85	0.81
Epoch 4	0.30	0.48	0.87	0.80
Epoch 5	0.26	0.35	0.89	0.87
Epoch 6	0.21	0.45	0.91	0.85
Epoch 7	0.17	0.34	0.93	0.86
Epoch 8	0.13	0.43	0.94	0.87
Epoch 9	0.11	0.46	0.96	0.87
Epoch 10	0.09	0.66	0.96	0.86
Epoch 11	0.08	0.78	0.97	0.85
Epoch 12	0.06	0.55	0.98	0.87
Epoch 13	0.02	0.65	0.99	0.88
Epoch 14	0.01	0.75	0.99	0.88

6.3.2 VGGNet

For the training of VggNet model, k-fold cross validation used.

When the Vggnet model was trained using the k-fold cross validation method for 10 subsets. val_acc results varied between 0.95 and 0.97, with an average value of 0.96. train_acc results also varied between 0.97 and 0.99, with an average value of 0.98. Early stopping method was used by monitoring validation loss. It ended in an average of 12 epochs.

7
Performance Analysis

7.1 Performance Metrics

- Performance during the training phase will be observed with loss and accuracy values with validation data.
- After the training of the models, the results will be obtained with the separated test data and will be displayed with visuals such as the confusion matrix (Figure 4.1) and classification report.

		Actual Class	
		Positive (**P**)	Negative (**N**)
Predicted Class	Positive (**P**)	True Positive (**TP**)	False Positive (**FP**)
	Negative (**N**)	False Negative (**FN**)	True Negative (**TN**)

Figure 7.1 Confusion Matrix

After our model trained, classification report method from sklearn.metrics library. With that method we reached precision, recall and f1 score metric results. Here is the formulas for those metrics.

Precision	$\textbf{Precision} = \frac{tp}{tp + fp}$
Recall	$\textbf{Recall} = \frac{tp}{tp + fn}$
F1-score	$\textbf{F1-score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$

Figure 7.2 Confusion Matrix

Results for that metrics given at Results chapter.

8
Experimental Results

In this section, 2D images of 32x32x32 voxels will be shared. Among these images called Candidate, first of all, screenshots will be shared for the augmented samples with and without nodules used in the train section.

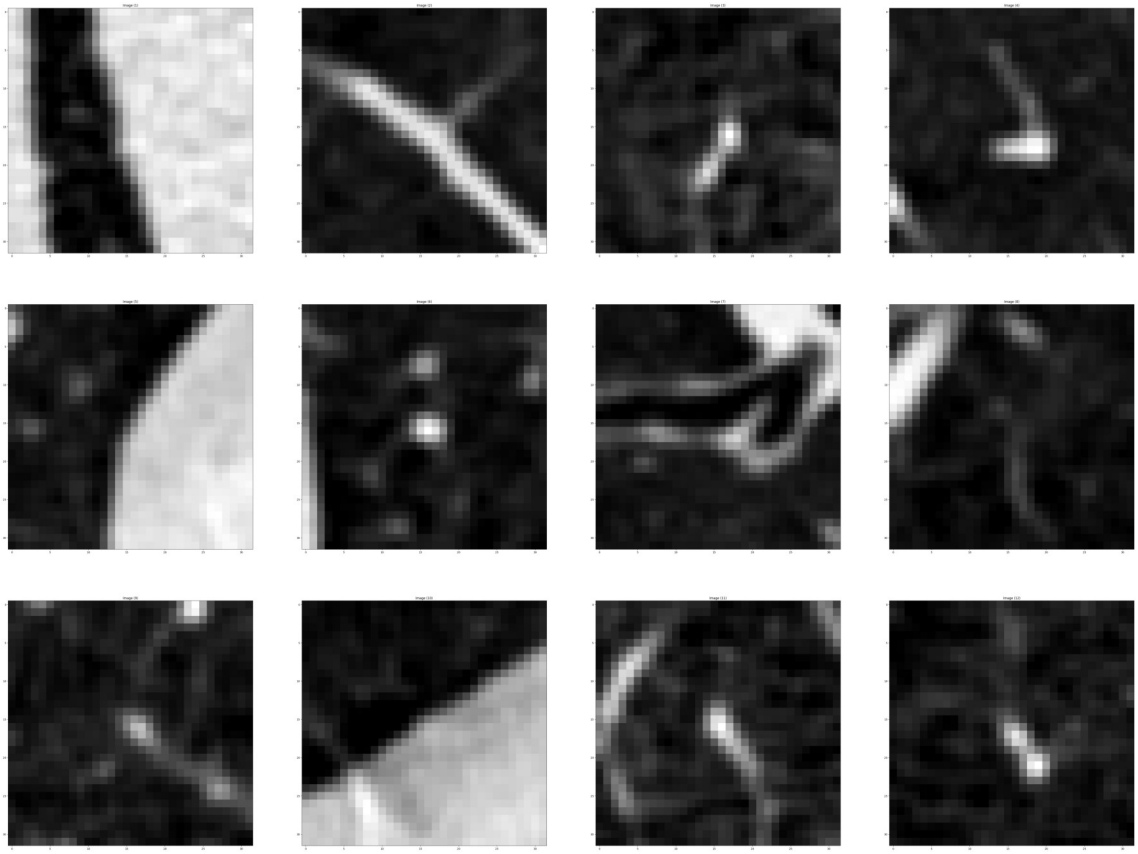

Figure 8.1 Augmented Images for Non Nodules

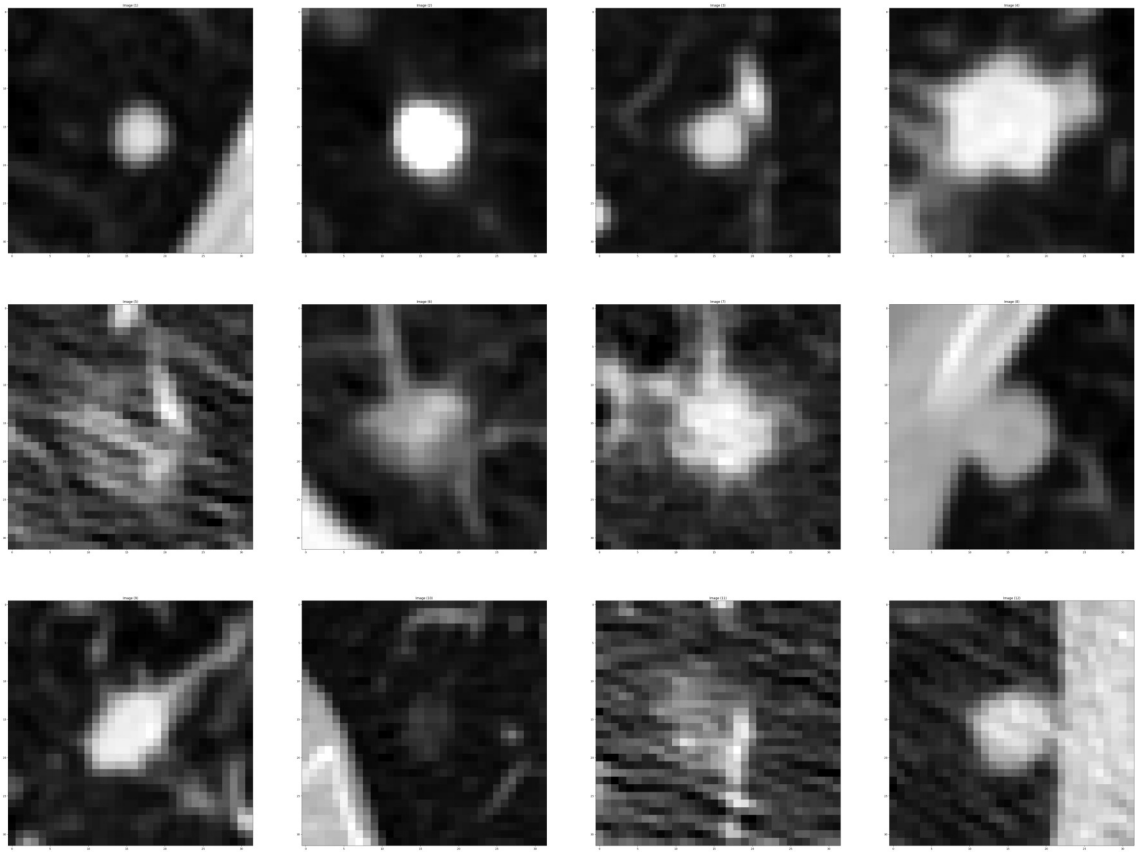

Figure 8.2 Augmented Images for Nodules

In this part, the results of the CNN 3D model without augmentation will be shared. When compared with the results with augmented data in the Results chapter, the positive effect of augmentation of the images to be trained on the results was seen.

The results obtained from the train process with non-augmented images are as shown in the images.

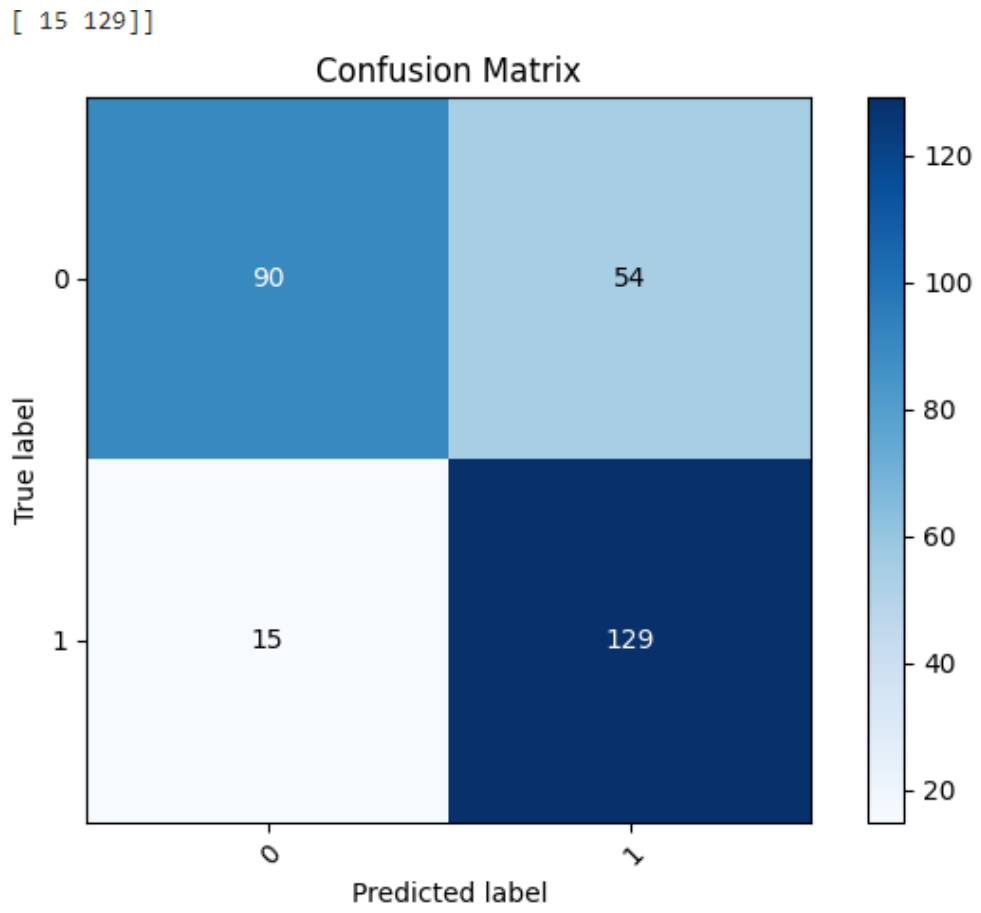

Figure 8.3 Confusion Matrix for none augmented trainset for CNN 3D Model

Classification Report:

	precision	recall	f1-score	support
Nonnodule	0.86	0.62	0.72	144
Nodule	0.70	0.90	0.79	144
accuracy			0.76	288
macro avg	0.78	0.76	0.76	288
weighted avg	0.78	0.76	0.76	288

Figure 8.4 Classification Report for none augmented trainset for CNN 3D Model

9
Results

In this section, the results of the CNN model for different measurement metrics are listed.

With the application, radiologists were supported by classifying chest CT images.

A table showing the results of the classification model is shared below.

With this application, it has been seen that the use of computer science in computerized tomography images (along with the development of diagnostic systems) can increase the diagnostic accuracy in computerized tomography images and help diagnose images in which the amount of radiation is less.

In future studies, studies can be carried out to comparatively determine the most successful CNN models at computed tomography by increasing the number of 3D models.

Model	***Precision***	***f1 score***	***recall***
3D CNN MODEL	0.87	0.88	0.90
VggNet MODEL	0.94	0.93	0.93

Table 9.1 Results Table

9.0.1 3D CNN Model Results

```
Confusion matrix, without normalization
[[141   3]
 [ 37 107]]
```

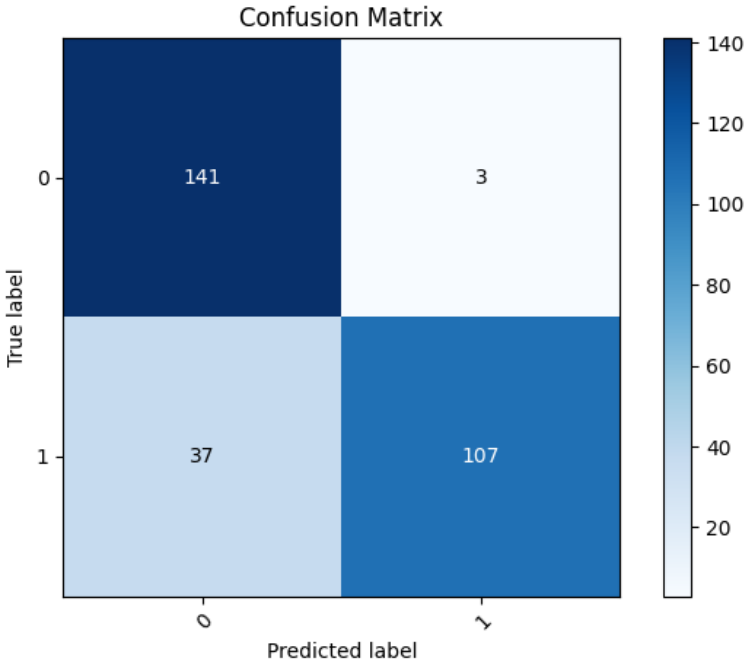

Figure 9.1 ConfusionMatrix for CNN 3D Model

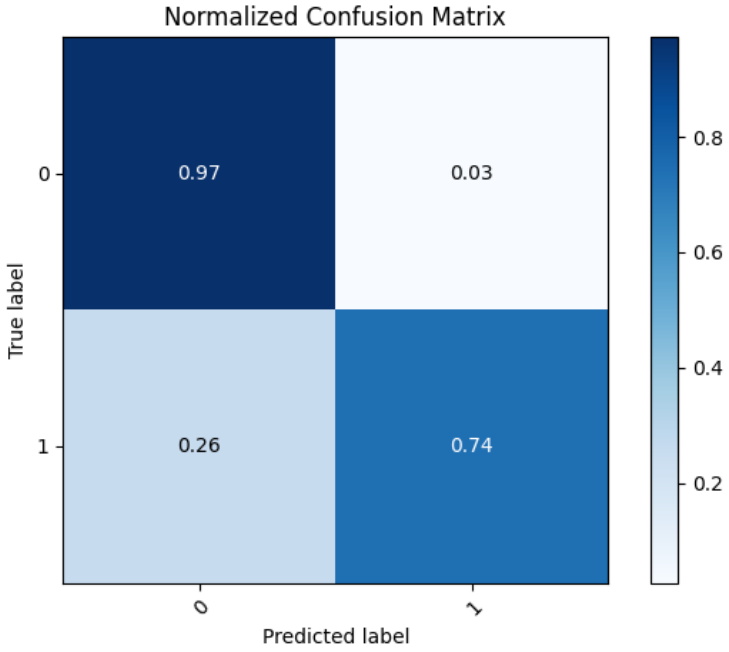

Figure 9.2 Normalized ConfusionMatrix for CNN 3D Model

Classification Report:

	precision	recall	f1-score	support
Nonnodule	0.79	0.97	0.87	144
Nodule	0.96	0.74	0.84	144
accuracy			0.86	288
macro avg	0.88	0.86	0.86	288
weighted avg	0.88	0.86	0.86	288

Figure 9.3 Classification Report with Metrics

9.0.2 VGGNet Model Results

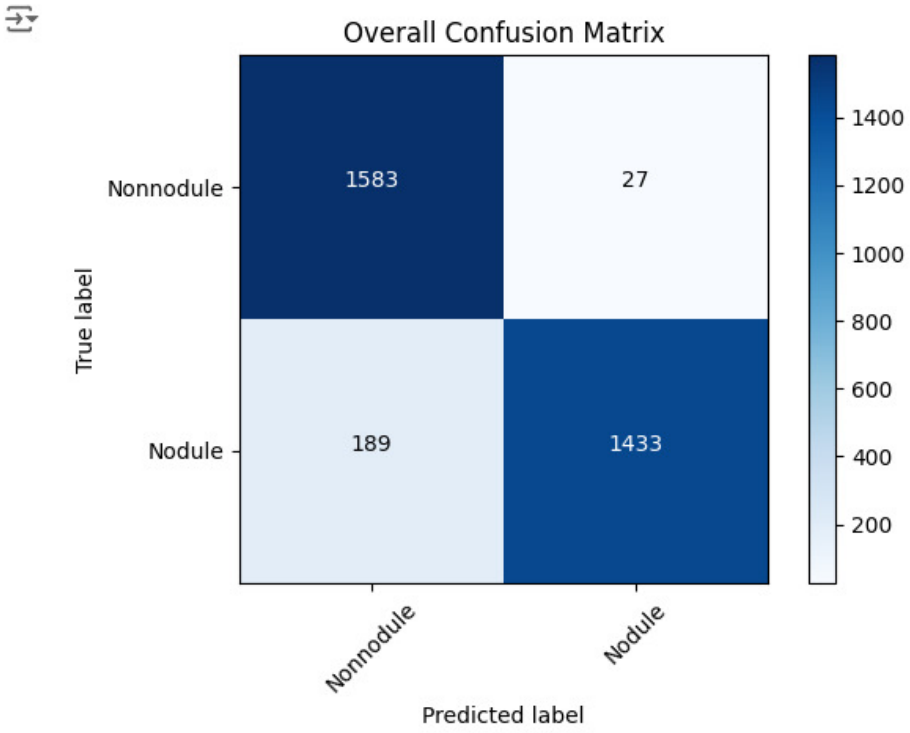

Figure 9.4 ConfusionMatrix for VGGNet Model

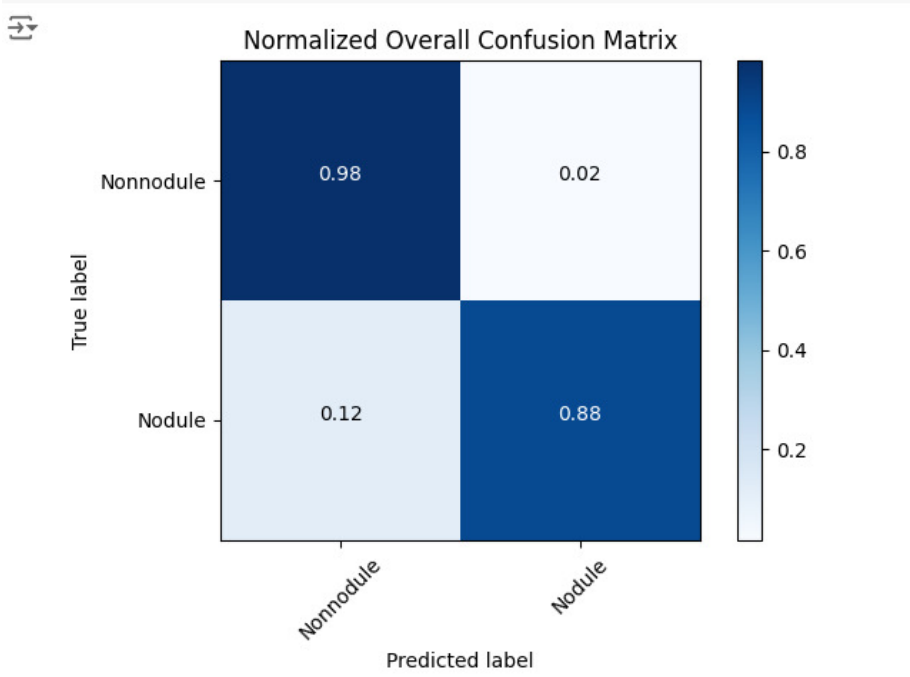

Figure 9.5 Normalized ConfusionMatrix for VGGNet Model

Overall Classification Report:

	precision	recall	f1-score	support
Nonnodule	0.89	0.98	0.94	1610
Nodule	0.98	0.88	0.93	1622
accuracy			0.93	3232
macro avg	0.94	0.93	0.93	3232
weighted avg	0.94	0.93	0.93	3232

Figure 9.6 Classification Report with Metrics for VGGNet

References

[1] "Luna 16 data set," [Online]. Available: https://luna16.grand-challenge.org/Data/.

[2] A. A. A. Setio *et al.*, "Validation, comparison, and combination of algorithms for automatic detection of pulmonary nodules in computed tomography images: The luna16 challenge," *Medical image analysis*, vol. 42, pp. 1–13, 2017.

[3] "Who lung cancer overview," [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/lung-cancer.

[4] J. Ming, B. Yi, Y. Zhang, and H. Li, "Low-dose ct image denoising using classification densely connected residual network," *KSII Transactions on Internet and Information Systems (TIIS)*, vol. 14, no. 6, pp. 2480–2496, 2020.

[5] S. Kuanar, V. Athitsos, D. Mahapatra, K. Rao, Z. Akhtar, and D. Dasgupta, "Low dose abdominal ct image reconstruction: An unsupervised learning based approach," pp. 1351–1355, 2019.

[6] E. Immonen *et al.*, "The use of deep learning towards dose optimization in low-dose computed tomography: A scoping review," *Radiography*, vol. 28, no. 1, pp. 208–214, 2022.

[7] N. T. Trung, D.-H. Trinh, N. L. Trung, and M. Luong, "Low-dose ct image denoising using deep convolutional neural networks with extended receptive fields," *Signal, Image and Video Processing*, vol. 16, no. 7, pp. 1963–1971, 2022.

[8] L. Huang, H. Jiang, S. Li, Z. Bai, and J. Zhang, "Two stage residual cnn for texture denoising and structure enhancement on low dose ct image," *Computer methods and programs in biomedicine*, vol. 184, p. 105 115, 2020.

[9] M. Han, H. Shim, and J. Baek, "Perceptual ct loss: Implementing ct image specific perceptual loss for cnn-based low-dose ct denoiser," *IEEE Access*, vol. 10, pp. 62 412–62 422, 2022.

[10] H. Chen *et al.*, "Low-dose ct with a residual encoder-decoder convolutional neural network," *IEEE transactions on medical imaging*, vol. 36, no. 12, pp. 2524–2535, 2017.

[11] W. Yang *et al.*, "Improving low-dose ct image using residual convolutional network," *IEEE access*, vol. 5, pp. 24 698–24 705, 2017.

[12] C. You, L. Yang, Y. Zhang, and G. Wang, "Low-dose ct via deep cnn with skip connection and network-in-network," in *Developments in X-Ray tomography XII*, SPIE, vol. 11113, 2019, pp. 429–434.