Second Review Document

DETECTION OF PNEUMONIA FROM CHEST X-RAYS

TEAM MEMBERS:

Name- Saswata Basu

Reg. No.- 20MID0035

Name- Arshi Goyal

Reg. No.- 20MID0109

Name- Nancy Sinha

Reg. No.- 20MID0115

Name- Pankil Kumar Singh

Reg. No.- 20MID0159

Name - Jaideep Singh

Reg. No.- 20MID0161

Guide Name- Dr. N.Poornima
Designation- Supervisor
Mobile No.- +91 98943 82400
Mail ID - poornima.n@vit.ac.in

Int M.Tech.

in

Computer Science and Engineering with Specialization in Data Science

School of Computer Science & Engineering

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

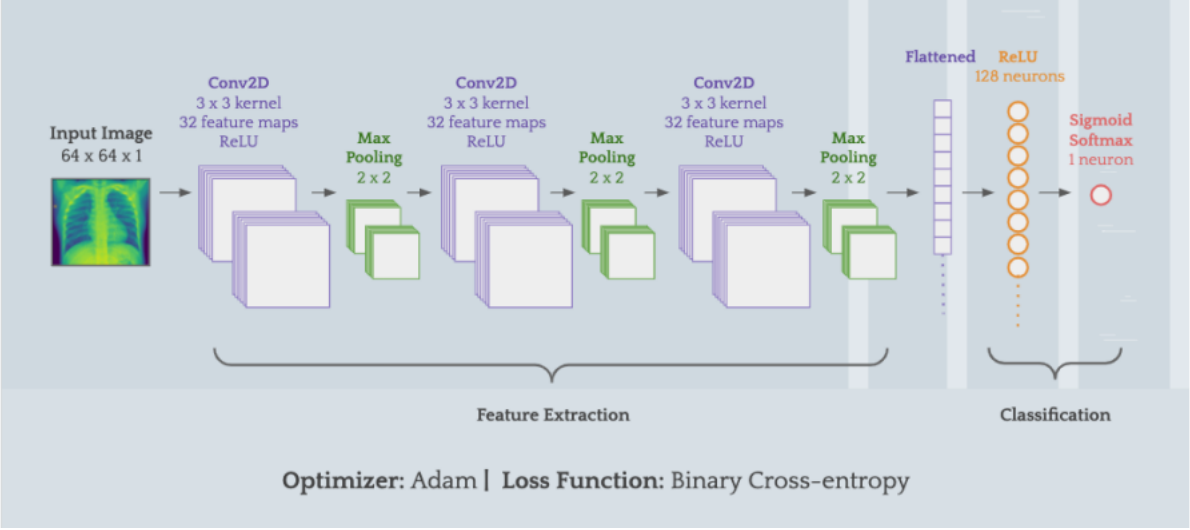
Abstract:

Pneumonia is a serious infectious disease that affects the lungs and can be life-threatening if left untreated. Early detection of pneumonia is crucial for effective treatment and management. Chest X-rays are a common diagnostic tool used to detect pneumonia in patients.

In recent years, machine learning algorithms have been developed to assist radiologists in detecting pneumonia from chest X-rays. These algorithms use deep learning techniques to analyze X-ray images and identify patterns associated with pneumonia.

The detection of pneumonia from chest X-rays using machine learning has shown promising results in terms of accuracy and speed. With further development and refinement, these algorithms could become a valuable tool in the early detection and treatment of pneumonia, potentially improving patient outcomes and reducing healthcare costs.

System Architecture



System Architecture Description

- **Input Layer:** This layer takes in the raw image data as input, typically in the form of a matrix of pixel values.

- **Convolutional Layer:** This layer applies a set of filters (also called kernels) to the input image in order to extract feature maps that represent patterns and shapes within the image. The output of this layer is a set of convolved feature maps.

- **Activation Layer:** This layer applies an activation function to the output of the convolutional layer in order to introduce non-linearity into the model. Common activation functions include **ReLU, sigmoid, and tanh.**

- **Pooling Layer:** This layer downsamples the output of the convolutional layer in order to reduce the spatial dimensions of the feature maps and make the model more computationally efficient. Common pooling methods include max pooling and average pooling.

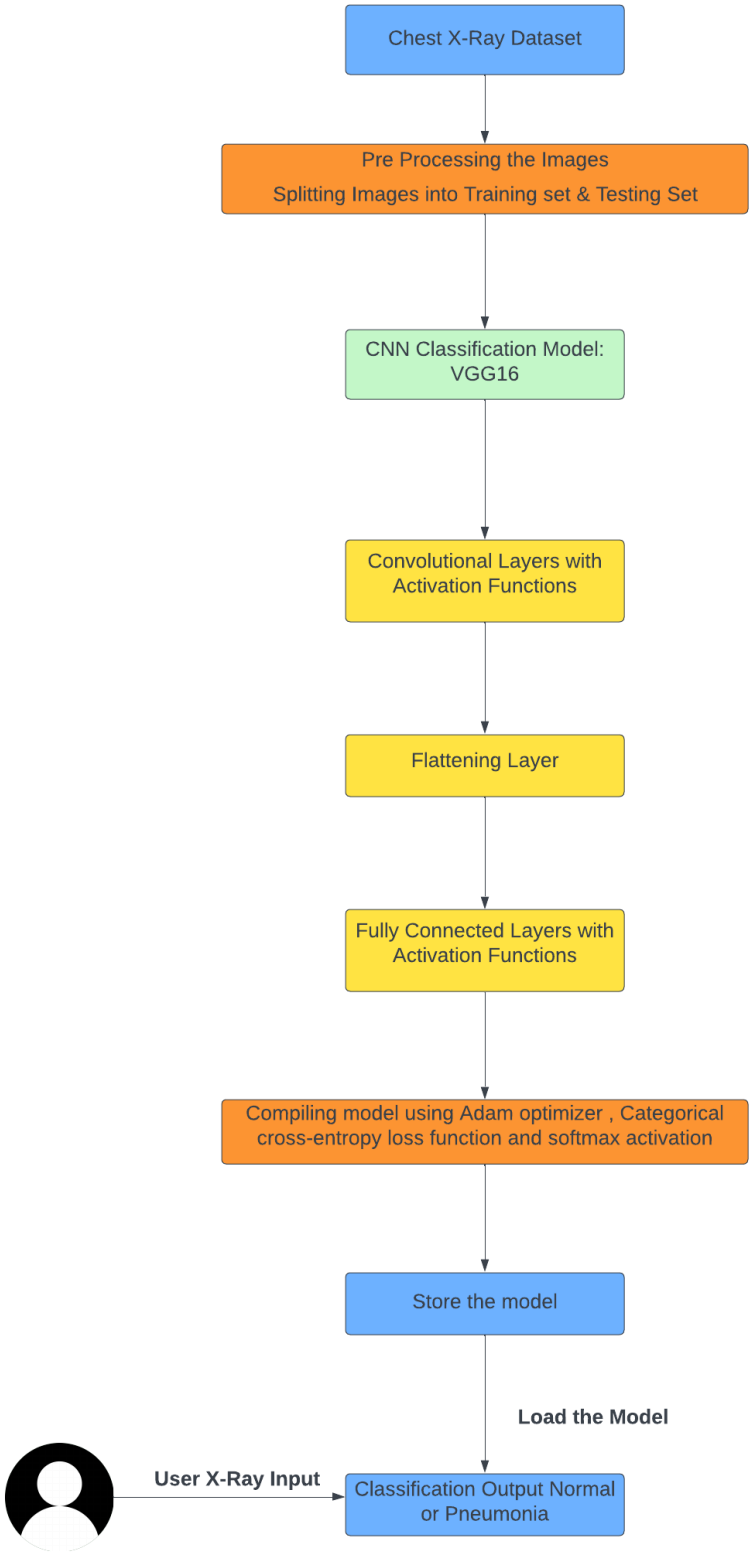
- **Dropout Layer:** This layer randomly drops out some of the neurons in the model during training in order to prevent overfitting and improve generalization performance.

- **Flatten Layer:** This layer flattens the output of the previous layer into a 1D vector, which can be fed into a fully connected layer.

- **Fully Connected Layer:** This layer takes in the flattened output from the previous layer and applies a set of weights to produce a final output vector. This output vector represents the class scores for the input image, and can be interpreted as the probability that the image belongs to each of the possible classes.

- **Output Layer:** This layer applies a final activation function (usually softmax) to the class scores in order to produce a probability distribution over the possible classes. The class with the highest probability is then selected as the final classification output.

SYSTEM MODEL: -


Chest X-Ray Dataset
Pre Processing the Images
Splitting Images into Training set & Testing Set
CNN Classification Model: VGG16
Convolutional Layers with Activation Functions
Flattening Layer
Fully Connected Layers with Activation Functions
Compiling model using Adam optimizer , Categorical cross-entropy loss function and softmax activation
Store the model
Load the Model
User X-Ray Input
Classification Output Normal or Pneumonia

Methodology Adapted

CNNs have been displayed to outflank other grouping procedures for picture arrangement assignments by and large. The reasons for this include:

Ability to extract relevant features: CNNs can automatically extract relevant features from the input data, without the need for manual feature engineering. In the case of pneumonia detection, CNNs can extract important features from chest X-ray images that may be indicative of pneumonia, such as infiltrates, consolidation, or nodules.

Hierarchical representation: CNNs can learn a hierarchical representation of the input data, where higher-level features are built upon lower-level features. This can help the model detect more complex patterns and relationships in chest X-ray images that may be indicative of pneumonia.

Translation invariance: CNNs are designed to be translation invariant, meaning that they can recognize the same features regardless of their position in the image. In the case of pneumonia detection, this can be helpful as pneumonia can appear in different regions of the chest X-ray image.

Scalability: CNNs can be easily scaled to handle large datasets and complex models. This is important in the case of pneumonia detection as there may be a large number of chest X-ray images to process.

High accuracy: CNNs have been shown to outperform other classification techniques for image classification tasks in many cases. This means that using a CNN model for pneumonia detection may result in a higher accuracy rate compared to other methods.

CNNs significantly outperformed other image classification methods in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). In the 2012 contest, the triumphant group utilized a CNN design called AlexNet, which accomplished a main 5 mistake pace of 15.3%, contrasted with the second-place blunder pace of 26.2%.

Compared to previous methods, which had top-5 error rates of 25-30%, this was a significant improvement.

CNNs have been used in a wide range of other fields, including TSR, since their victory in the ILSVRC competition established them as a powerful tool for image classification tasks.

Expected Results with discussion.

#Importing Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly
import plotly.express as px

import plotly.graph_objs as go
from plotly.offline import init_notebook_mode, plot, iplot
import cv2
import random
import os
import glob
from tqdm.notebook import tqdm
import albumentations as A
import tensorflow as tf

from tensorflow.keras.layers import Conv2D, Flatten, MaxPooling2D, Dense,
Dropout , BatchNormalization

from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.preprocessing.image import
ImageDataGenerator from keras.applications.vgg16 import
VGG16
from sklearn.model_selection import train_test_split
from sklearn.metrics import
classification_report,confusion_matrix from keras.callbacks
import ReduceLROnPlateau
#Loading Data
train_data = glob.glob('chest_xray/train/**/*.jpeg')
test_data = glob.glob('chest_xray/test/**/*.jpeg')
val_data = glob.glob('chest_xray/val/**/*.jpeg')
print("〜"*20)
print(f"Training Set has: {len(train_data)} images")
print(f"Testing Set has: {len(test_data)} images")
print(f"Validation Set has: {len(val_data)} images")
print("〜"*20)
DIR = "chest_xray/"
sets = ["train", "test", "val"]
all_pneumonia = []
all_normal = []
for cat in sets:
        path = os.path.join(DIR, cat)
        norm = glob.glob(os.path.join(path, "NORMAL/*.jpeg"))
        pneu = glob.glob(os.path.join(path, "PNEUMONIA/*.jpeg"))
        all_normal.extend(norm)
        all_pneumonia.extend(pneu)
print("〜"*20)
print(f"Total Pneumonia Images: {len(all_pneumonia)}")
print(f"Total Normal Images: {len(all_normal)}")
print("〜"*20)
```

#Plot charts for Class distribution

```
labels = ["Normal",'Pneumonia ']
values = [len(all_normal), len(all_pneumonia)]
colors = ['green', 'pink']
fig=go.Figure(data=[go.Pie(labels=l
abels,
values=values,hole=.5)])
fig.update_traces(hoverinfo='value', textinfo='label+percent',
textfont_size=20, marker=dict(colors=colors, line=dict(color='#000000',
width=3)))
fig.update_layout(title="Image Category
Distribution", titlefont={'size': 30},)
iplot(fig)
```

randomly

```
random.shuffle(all_normal)
random.shuffle(all_pneumonia)
images = all_normal[:50] + all_pneumonia[:50]
```

#Viewing Images in X-Ray

```
fig=plt.figure(figsize=(15, 10))
columns = 4; rows = 5
for i in range(1, columns*rows +1):
        img = cv2.imread(images[i])
        img = cv2.resize(img, (128, 128))
        fig.add_subplot(rows, columns, i)
        plt.imshow(img)
        plt.axis(False)
```

#Convert to Grayscale then apply Gaussian Blur

Gaussian filters have the properties of having no overshoot to a step function input while minimizing the rise and fall time. In terms of image processing, any sharp edges in images are smoothed while minimizing too much blurring. OpenCV provides

cv2.gaussianblur() function to apply Gaussian Smoothing on the input source image.

Following is the syntax of GaussianBlur() function :

```
dst = cv2.GaussianBlur(src, ksize, sigmaX[, dst[,
sigmaY[, borderType=BORDER_DEFAULT]]] )
fig=plt.figure(figsize=(15, 10))
columns = 4; rows = 2
for i in range(1, columns*rows +1):
        img = cv2.imread(images[i])
        img = cv2.resize(img, (512, 512))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
        img = cv2.addWeighted (img, 4, cv2.GaussianBlur(img, (0,0), 512/10), -4, 128)
        fig.add_subplot(rows, columns, i)
        plt.imshow(img)
        plt.axis(False)
```

 # Fourier Method for viewing pixel distributions

```
fig=plt.figure(figsize=(15, 10))
columns = 4; rows = 2
for i in range(1, columns*rows +1):
        img = cv2.imread(images[i])
        img = cv2.resize(img, (512, 512))
        img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
        f = np.fft.fft2(img)
        fshift = np.fft.fftshift(f)
        magnitude_spectrum = 20*np.log(np.abs(fshift))
        fig.add_subplot(rows, columns, i)
        plt.imshow(magnitude_spectrum)
        plt.axis(False)
```

#Image Erosion

Erosion and Dilation are **morphological image processing** operations.

OpenCV morphological image processing is a procedure for modifying the geometric structure in the image. In morphism, we find the shape and size or structure of an object. Both operations are defined for binary images, but we can also use them on a grayscale image. These are widely used in the following way:

1. Removing Noise
2. Identify intensity bumps or holes in the picture.
3. Isolation of individual elements and joining disparate elements in image.

```
fig=plt.figure(figsize=(15, 10))
columns = 5; rows = 2
for i in range(1, columns*rows +1):
        img = cv2.imread(images[i])
        img = cv2.resize(img, (512, 512))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        kernel = np.ones((5, 5), np.uint8)
        img_erosion = cv2.erode(img, kernel, iterations=3)
        fig.add_subplot(rows, columns, i)
        plt.imshow(img_erosion)
        plt.axis(False)
```

#Dilation of Images

Dilation is a technique where we expand the image. It adds the number of pixels to the boundaries of objects in an image. The structuring element controls it. The structuring element is a matrix of 1's and 0's. The dilation operation is performed by using the **cv2.dilate()** method. The syntax is given below:

```
cv2.dilate(src, dst, kernel)
```

Parameters: The **dilate()** function accepts the following argument:

1. **src** - It represents the input image.
2. **dst** - It represents the output image.
3. **kernel** - It represents the kernel.

```
fig=plt.figure(figsize=(15, 10))
columns = 5; rows = 2
for i in range(1, columns*rows +1):
```

```
    img = cv2.imread(images[i])
    img = cv2.resize(img, (512, 512))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    kernel = np.ones((5, 5), np.uint8)
    img_erosion = cv2.dilate(img, kernel, iterations=3)
    fig.add_subplot(rows, columns, i)
    plt.imshow(img_erosion)
    plt.axis(False)
```

 # Canny Edge Detection for Segmentation

The Canny edge detector is arguably the most well-known and the most used edge detector in all of computer vision and image processing. While the Canny edge detector is not exactly “trivial” to understand, we’ll break down the steps into bite-sized pieces so we can understand what is going on under the hood. Fortunately for us, since the Canny edge detector is so widely used in almost all computer vision applications, OpenCV has already implemented it for us in the cv.Canny() function.

```
fig=plt.figure(figsize=(15, 10))
columns = 5; rows = 2
for i in range(1, columns*rows +1):
    img = cv2.imread(images[i])
    img = cv2.resize(img, (512, 512))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    edges = cv2.Canny(img, 80, 100)
    fig.add_subplot(rows, columns, i)
    plt.imshow(edges)
    plt.axis(False)
```

Building Machine Learning Model for Pneumonia Detection

#Divide data to create a training and validation set using the Keras Image Data Generator with Image Enhancement

```
train_gen = ImageDataGenerator(
    rescale=1/255.,
    horizontal_flip=True,
```

```
        vertical_flip=False,
        rotation_range=0.3,
        zoom_range=0.4)
val_gen = ImageDataGenerator(
rescale=1/255.,)
Train = train_gen.flow_from_directory(
        "chest_xray/train",batch_size=16,
        target_size=(224, 224),#class_mode="binary")
val = train_gen.flow_from_directory(
        "chest_xray/test",batch_size=8,
        target_size=(224, 224),#class_mode="binary")
#Convolution Neural Network Architecture
model = Sequential()
model.add(Conv2D(32,(3,3),strides=(1, 1),activation='relu',padding='same',
input_shape=(224, 224, 3)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(64,(3,3),strides=(1, 1) ,padding='same',activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(128,(3,3),strides=(1, 1),padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(256,(3,3),strides=(1, 1),padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(2, activation='softmax'))
model.summary()
#Model Compilation
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
#Model Training fit_generator() instead of fit() because we are going to take
the train data from the train_gen object.
early_stopping_cb =
tf.keras.callbacks.EarlyStopping(patience=5,restore_best_weights=True)
history =
model.fit_generator(Train,epochs=20,validation_data=val,steps_per_epoch=50,callbacks
=[early_stopping_cb])
#Visualization of model performance
test = train_gen.flow_from_directory(
        "chest_xray/test",batch_size=8,
        target_size=(224, 224),#class_mode="binary")
predict = model.predict_generator(test, steps=np.ceil(400/8))
def show_predictions(y_img_batch, y_true, y_pred, subplot_params, plot_params,
class_map, testing_dir, image_file_name, count=8, sample=True):
        fig, axs = get_fig_axs(subplot_params)
        plt.rcParams.update({'axes.titlesize': plot_params["axes.titlesize"]})
        plt.subplots_adjust(hspace=subplot_params["hspace"],wspace=subplot_params["
        wspace"])
        file_names = test_generator.filenames
        m = {}
        length = len(y_true)
        for i in range(0, count):
                num = i
                if sample:
                        num = random.randint(0, length-1)
                        while num in m:
                                num = int(random.randint(0, length-1))
                        m[num]=1
                plt.subplot(subplot_params["nrows"], subplot_params["ncols"], i+1)
                img = cv2.imread(testing_dir+"\\"+ file_names[num], 1)
```

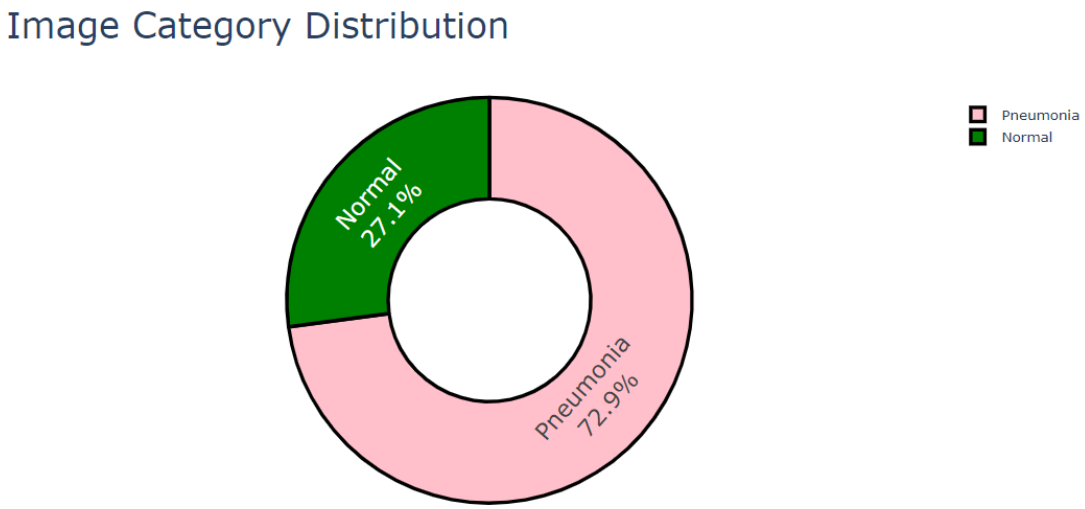
```
                plt.imshow(img)
                plt.xticks([])
                plt.yticks([])
                original = class_map[y_true[num]]
                predicted = class_map[y_pred[num]]
                text = ("%s%s%s%s%s"%("True: ", original, "\n", "Pred: ", predicted))
if original==predicted:
                        plt.title(title_text)
                else:
                        plt.title(title_text, color='red')
                if plot_params["update_image"] and os.path.exists(image_file_name):
                        os.remove(image_file_name)
                fig.savefig(image_file_name, dpi=subplot_params["dpi"])
        plt.tight_layout()
        plt.show()
image_file_name_batch = figure_directory+"/train"
image_file_name_sample = figure_directory+"/test"
batch_size_t = len(y_true_batch)
class_map = {v: k for k, v in test_generator.class_indices.items()}
dpi=100
ncols = 8
nrows = 4
count = ncols*nrows
subplot_params = get_reset_subplot_params(nrows, ncols, dpi)
plot_params = get_reset_plot_params()
show_predictions(y_img_batch, y_true_batch, y_pred_batch, subplot_params,
plot_params, class_map, testing_dir, image_file_name_batch, count=count, sample)
plt.figure(figsize=(8,6))
plt.title('Accuracy scores')
plt.plot(history.history['accuracy'],'go-')
plt.plot(history.history['val_accuracy'],'ro-')
```

```
plt.legend(['accuracy', 'val_accuracy'])
plt.show()
plt.figure(figsize=(8,6))
plt.title('Loss value')
plt.plot(history.history['loss'],'go-')
plt.plot(history.history['val_loss'],'ro-')
plt.legend(['loss', 'val_loss'])
plt.show()
```

SCREENSHOTS OF OUTPUT

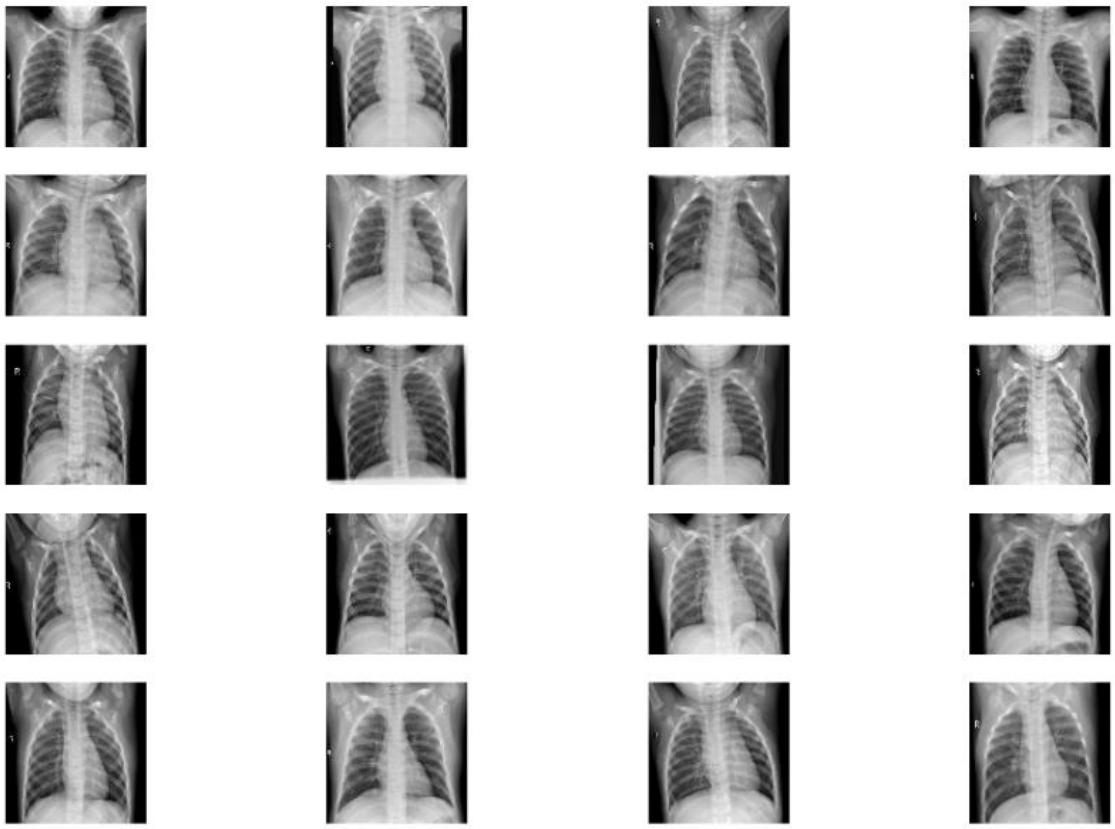
1. Image Category Distribution

```
In [5]: labels = ["Normal",'Pneumonia ']
values = [len(all_normal), len(all_pneumonia)]
colors = ['green', 'pink']
fig = go.Figure(data=[go.Pie(labels=labels,values=values,hole=.5)])
fig.update_traces(hoverinfo='value', textinfo='label+percent', textfont_size=20,marker=dict(colors=colors, line=dict(color='#000
fig.update_layout(title="Image Category Distribution",
titlefont={'size': 30},)
iplot(fig)
```



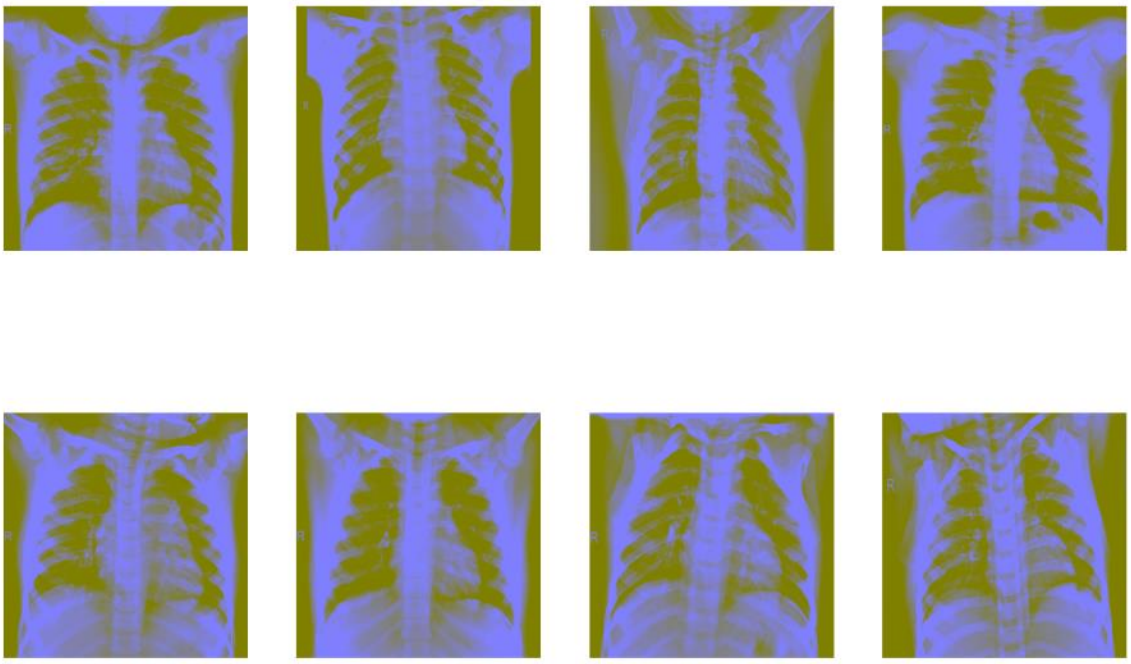
2. On viewing the images in X-RAY:

In [14]:
```
fig=plt.figure(figsize=(15, 10))
columns = 4; rows = 5
for i in range(1, columns*rows +1):
    img = cv2.imread(images[i])
    img = cv2.resize(img, (128, 128))
    fig.add_subplot(rows, columns, i)
    plt.imshow(img)
    plt.axis(False)
```



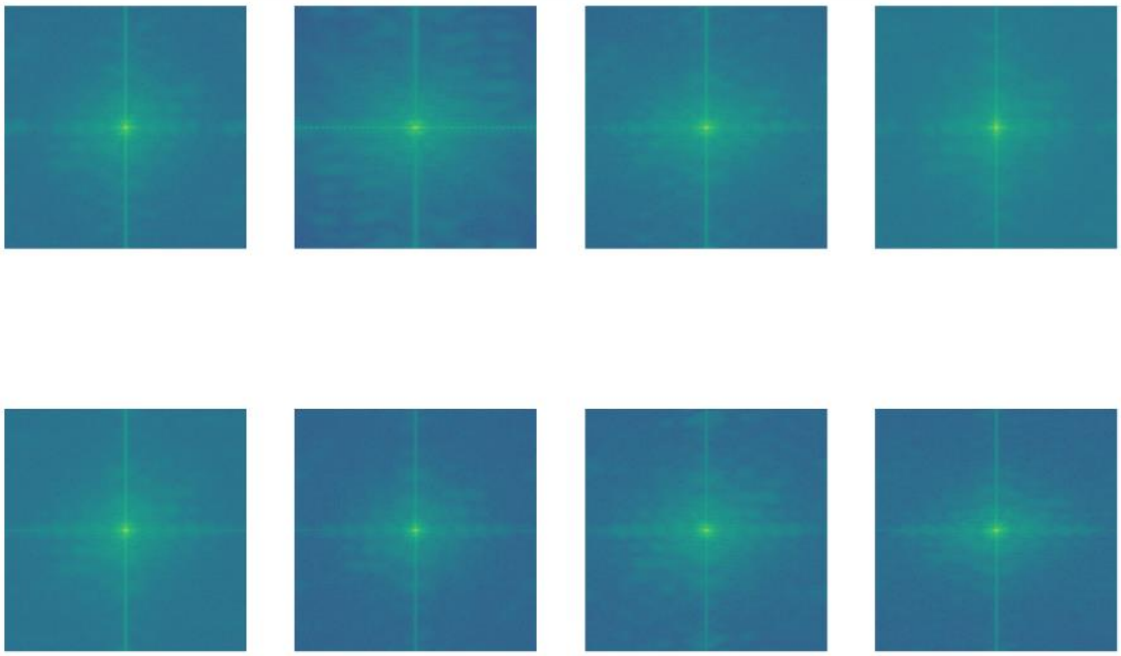
3. Gaussian Blurred Images:

In [15]:
```
fig=plt.figure(figsize=(15, 10))
columns = 4; rows = 2
for i in range(1, columns*rows +1):
    img = cv2.imread(images[i])
    img = cv2.resize(img, (512, 512))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    img = cv2.addWeighted (img, 4, cv2.GaussianBlur(img, (0,0), 512/10), -4, 128)
    fig.add_subplot(rows, columns, i)
    plt.imshow(img)
    plt.axis(False)
```



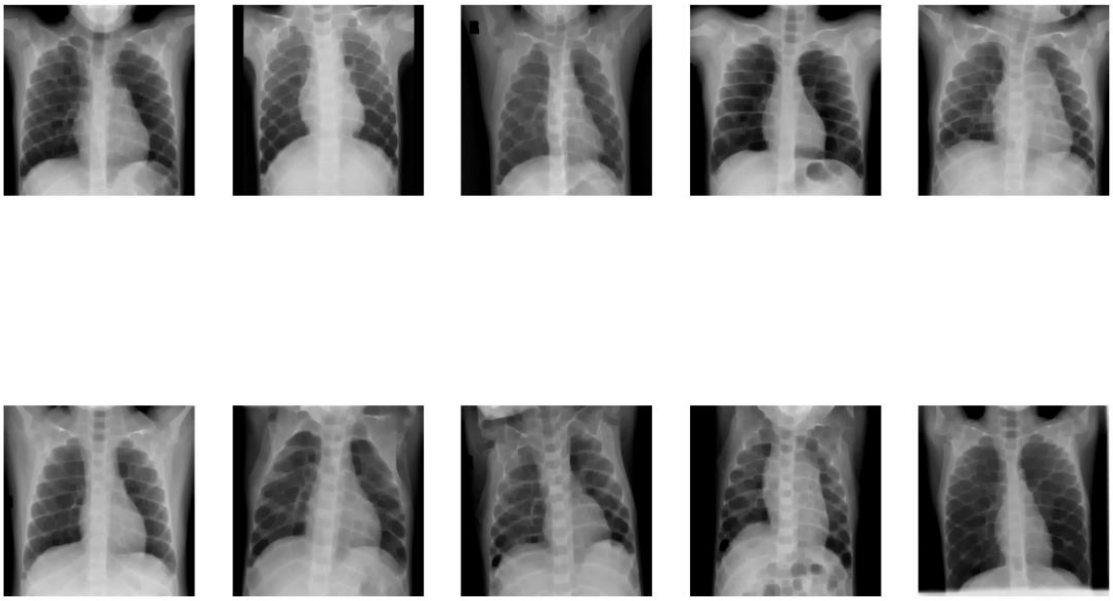
4. Viewing Pixel Distribution

In [16]:
```
fig=plt.figure(figsize=(15, 10))
columns = 4; rows = 2
for i in range(1, columns*rows +1):
    img = cv2.imread(images[i])
    img = cv2.resize(img, (512, 512))
    img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    f = np.fft.fft2(img)
    fshift = np.fft.fftshift(f)
    magnitude_spectrum = 20*np.log(np.abs(fshift))
    fig.add_subplot(rows, columns, i)
    plt.imshow(magnitude_spectrum)
    plt.axis(False)
```



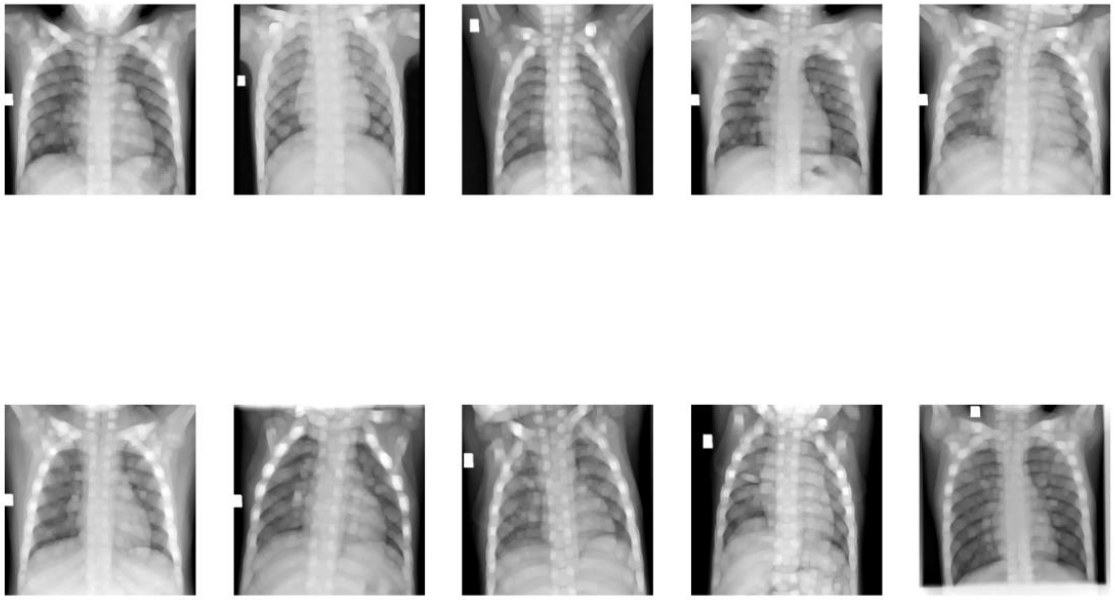
5. On Applying Erosion to Images

In [17]:
```
fig=plt.figure(figsize=(15, 10))
columns = 5;
rows = 2
for i in range(1, columns*rows +1):
    img = cv2.imread(images[i])
    img = cv2.resize(img, (512, 512))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    kernel = np.ones((5, 5), np.uint8)
    img_erosion = cv2.erode(img, kernel, iterations=3)
    fig.add_subplot(rows, columns, i)
    plt.imshow(img_erosion)
    plt.axis(False)
```



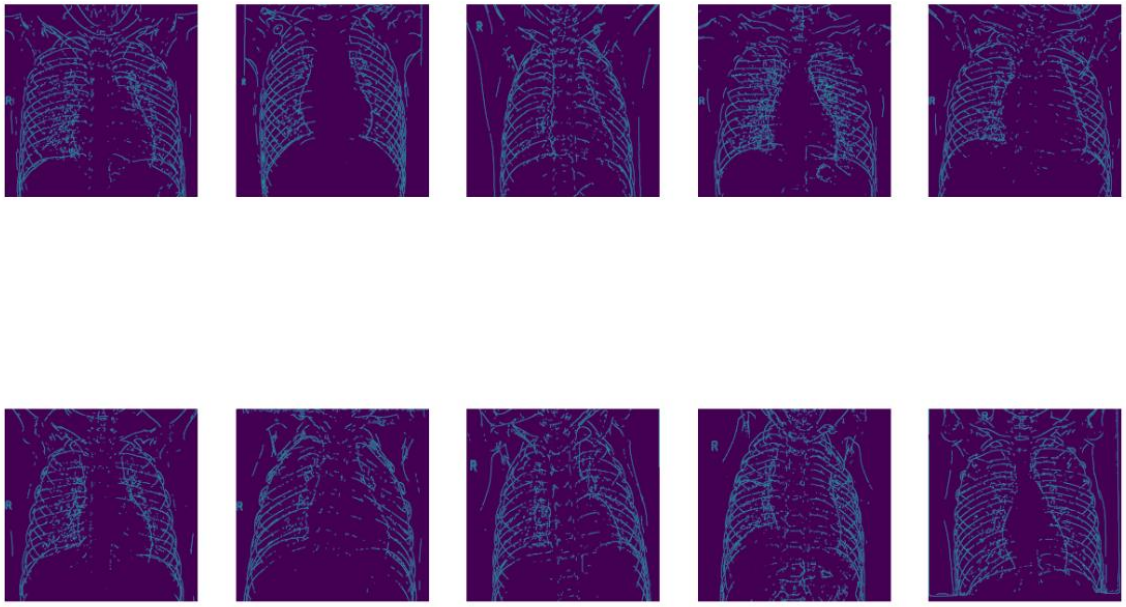
6. After Dilation

In [18]:
```
fig=plt.figure(figsize=(15, 10))
columns = 5;
rows = 2
for i in range(1, columns*rows +1):
    img = cv2.imread(images[i])
    img = cv2.resize(img, (512, 512))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    kernel = np.ones((5, 5), np.uint8)
    img_erosion = cv2.dilate(img, kernel, iterations=3)
    fig.add_subplot(rows, columns, i)
    plt.imshow(img_erosion)
    plt.axis(False)
```



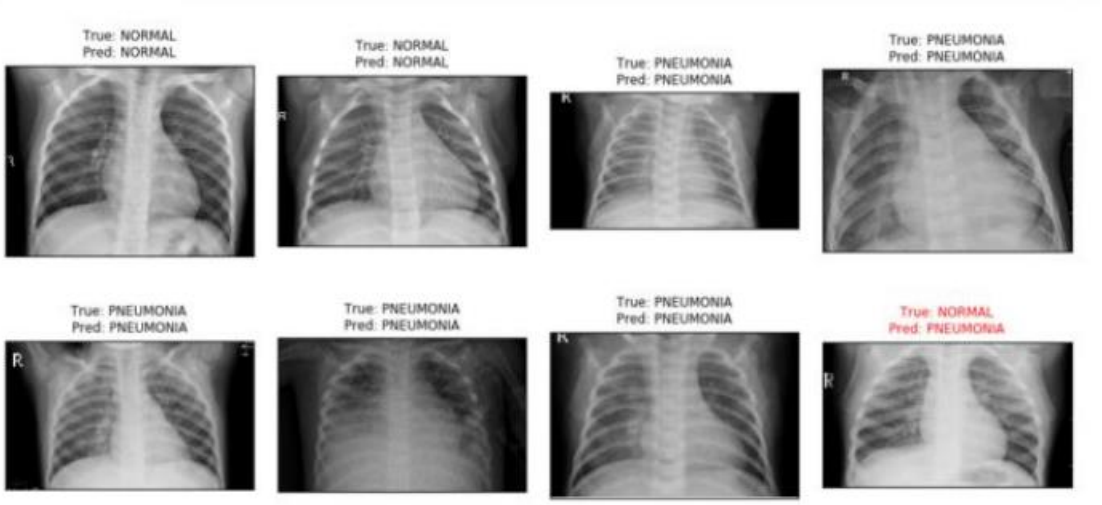
7. On implementing Canny Edge Detection

In [19]:
```
fig=plt.figure(figsize=(15, 10))
columns = 5; rows = 2
for i in range(1, columns*rows +1):
    img = cv2.imread(images[i])
    img = cv2.resize(img, (512, 512))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    edges = cv2.Canny(img, 80, 100)
    fig.add_subplot(rows, columns, i)
    plt.imshow(edges)
    plt.axis(False)
```



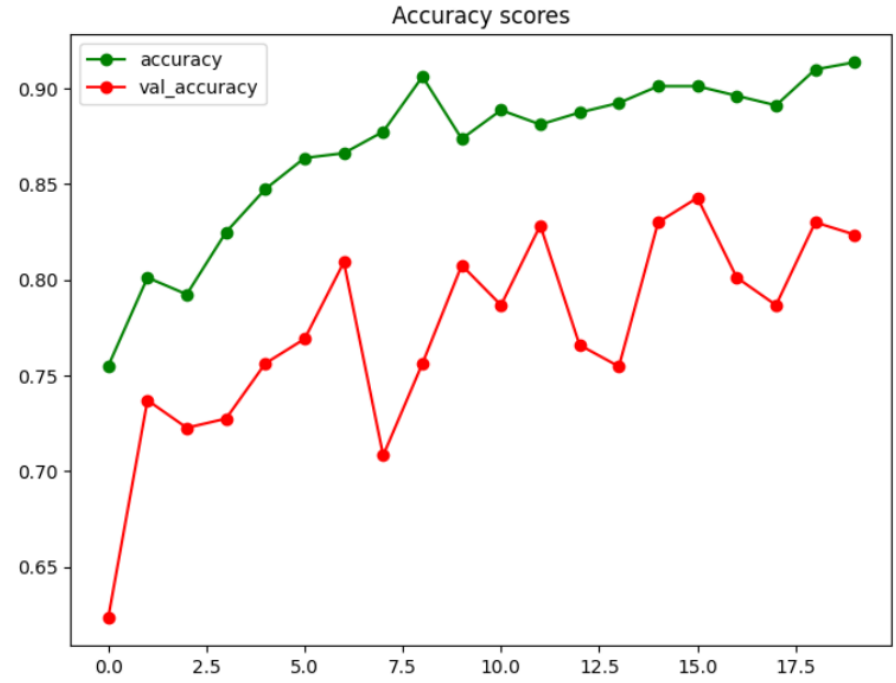
Model Predictions:

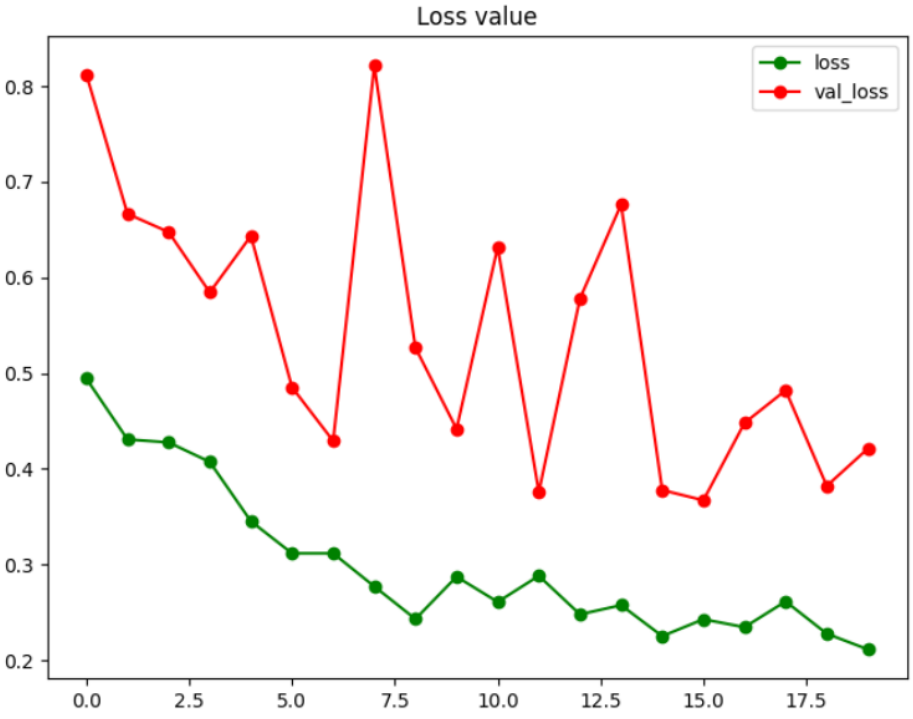
One false prediction out of 8 image predictions



Accuracy Score of the model:

```
Found 624 images belonging to 2 classes.
50/50 [==============================] - 7s 132ms/step
```





Details of Hardware and Software

Software Configuration: -

This software package is developed using java as front end which is supported by sun micro system. Microsoft SQL Server as the back end to store the database. Operating System: Windows NT, windows 98

Database: MS SQL Server (back end)

PYTHON BASED SOFTWARE EXAMPLE: ANACONDA, JUPYTER, R. STUDIO, IDLE ETC o Processor: Preferably 1.0 GHz or Greater. o RAM: 2 GB or Greater.

Hardware Configuration: -

Processor: Pentium(R)Dual-core CPU

Hard Disk: 40GB

RAM: 256 MB or more

References: (API format)

Weblinks:

1. https://www. sciencedirect.com/science/article/abs/pii/S0263224119305202
2. https://ieeexplore.ieee.org/abstract/document/9787593
3. https://ieeexplore.ieee.org/abstract/document/8869364
4. https://ieeexplore.ieee.org/document/9512631
5. https://ieeexplore.ieee.org/abstract/document/9231235
6. https:// ieeexplore.ieee.org/document/9381118
7. https://jestec.taylors.edu.my/Vol%2016%20issue%201%20February%202021/16_1_61.pdf
8. https://ieeexplore.ieee. org/abstract/document/9175594
9. https://ieeexplore.ieee.org/abstract/document/9241305
10. https://ieeexplore.ieee.org/abstract/document/9214289
11. https://ieeexplore.ieee.org/abstract/document/9389754
12. https://ieeexplore.ieee.org/document/9478246
13. https://ieeexplore.ieee.org/abstract/document/8861969
14. https:// ieeexplore.ieee.org/abstract/document/9272749

15. https://doi. org/10.3390/app10093233
16. https:// ieeexplore.ieee.org/document/9300429
17. https://ieeexplore.ieee.org/abstract/document/8857277
18. https: //ieeexplore.ieee.org/abstract/document/9091755
19. https://www.medrxiv.org/content/10.1101/2020.08.20.20178913.abstract
20. https://ieeexplore.ieee.org/abstract/document/9676146
21. https://ieeexplore.ieee.org/document/9474444
22. https://ieeexplore.ieee.org/abstract/document/8325607
23. http://cs229.stanford.edu/proj2017/final-reports/5231221.pd

Journal:

1. Antin, B., Kravitz, J., & Martayan, E. (n.d.). Detecting Pneumonia in Chest X-Rays with Supervised Learning.
2. Deo, G., Totlani, J., & Mahamuni, C. (2021). Detection of COVID-19 and Prediction of Pneumonia from Chest X-Rays using Deep Learning. IEEE Access, 9, 105737-105749.
3. Garstka, J., & Strzelecki, M. (n.d.). Pneumonia detection in X-ray chest images based on convolutional neural networks and data augmentation methods.
4. Han, Y., Chen, C., Tewfik, A., Ding, Y., & Peng, Y. (n.d.). Pneumonia Detection On Chest X-Ray Using Radiomic Features And Contrastive Learning. [IEEE Xplore Digital Library].
5. Haque, K. F., Haque, F. F., Gandy, L., & Abdelgawad, A. (2020). Automatic Detection of COVID-19 from Chest X-ray Images with Convolutional Neural Networks. IEEE Access, 8, 221775-221783.
6. Haritha, D., Krishna Pranathi, M., & Reethika, M. (n.d.). COVID Detection from Chest X-rays with DeepLearning: CheXNet. [IEEE Xplore Digital Library].
7. Harshvardhan GM, Mahendra Kumar, Gourisaria Siddharth Swarup Rautaray, Manjusha Pandey. Pneumonia Detection Using CNN through Chest X-RAY. Journal of Engineering Science and Technology Vol. 16, No. 1 (2021) 861 -

876 © School of Engineering, Taylor's University.

8. Ihssan S. Masad, Amin Alqudah, Ali Mohammad Alqudah, Sami Almashaqbeh. A hybrid deep learning approach towards building an intelligent system for pneumonia detection in chest X-ray images. International Journal of Electrical and Computer Engineering (IJECE) Vol. 11, No. 6, December 2021, pp. 5530~5540 ISSN: 2088-8708, DOI: 10.11591/ijece.v11i6.pp5530-5540
9. Irfan, A., Adivishnu, A. L., Sze-To, A., Dehkharghanian, T., Rahnamayan, S., & Tizhoosh, H. R. (2020). Classifying Pneumonia among Chest X-Rays Using Transfer Learning. IEEE Access, 8, 165957-165994.
10. Islam, S. R., Maity, S. P., Ray, A. K., & Mandal, M. (n.d.). Automatic Detection of Pneumonia on Compressed Sensing Images using Deep Learning.
11. Jaiswal, A. K., Tiwari, P., Kumar, S., Gupta, D., Khanna, A., & Rodrigues, J. J. P. C. (n.d.). Identifying pneumonia in chest X-rays: A deep learning approach.
12. Khan, W., Zaki, N., & Ali, L. (2020). Intelligent Pneumonia Identification From Chest X-Rays: A Systematic Literature Review. IEEE Access, 8, 221226-221244.
13. Labhane, G., Pansare, R., Maheshwari, S., Tiwari, R., & Shukla, A. (n.d.). Detection of Pediatric Pneumonia from Chest X-Ray Images using CNN and Transfer Learning. [IEEE Xplore Digital Library].
14. Labhane, G., Pansare, R., Maheshwari, S., Tiwari, R., & Shukla, A. (n.d.). Detection of Pediatric Pneumonia from Chest X-Ray Images using CNN and Transfer Learning.
15. Li, B., Kang, G., Cheng, K., & Zhang, N. (n.d.). Attention-Guided Convolutional Neural Network for Detecting Pneumonia on Chest X-Rays.
16. M. B. Darici, Z. Dokur, and T. Olmez. Pneumonia Detection and Classification Using Deep Learning on Chest X-Ray Images. Int J Intell Syst Appl Eng, vol. 8, no. 4, pp. 177–183, Dec. 2020.
17. Rahman T, Chowdhury MEH, Khandakar A, Islam KR, Islam KF, Mahbub ZB, Kadir MA, Kashem S. Transfer Learning with Deep Convolutional Neural Network (CNN) for Pneumonia Detection Using Chest X-ray. Applied Sciences. 2020; 10(9):3233.

18. Rajaraman, S., Siegelman, J., Alderson, P. O., Folio, L. S., Folio, L. R., & Antani, S. K. (2022). Iteratively Pruned Deep Learning Ensembles for COVID-19 Detection in Chest X-Rays. [IEEE Xplore Digital Library].
19. Shah, S., Mehta, H., & Sonawane, P. (n.d.). Pneumonia Detection Using Convolutional Neural Networks.
20. Sharma, A., Raju, D., & Ranjan, S. (n.d.). Detection of pneumonia clouds in chest X-ray using image processing approach.
21. Terry Gao. Chest X-ray image analysis and classification for COVID-19 pneumonia detection using Deep CNN. Counties Manukau Health, Auckland, 1640, New Zealand.
22. Tyagi, K., Pathak, G., Nijhawan, R., & Mittal, A. (2020). Detecting Pneumonia using Vision Transformer and comparing with other techniques. IEEE Access, 8, 228836-228843.
23. Varshni, D., Thakral, K., Agarwal, L., Nijhawan, R., & Mittal, A. (n.d.). Pneumonia Detection Using CNN based Feature Extraction. [IEEE Xplore Digital Library].
24. Varshni, D., Thakral, K., Agarwal, L., Nijhawan, R., & Mittal, A. (n.d.). Pneumonia Detection Using CNN based Feature Extraction.
25. Zhang, J., Xie, Y., Pang, G., Liao, Z., Verjans, J., Li, W., Sun, Z., He, J., & Li, Y. (n.d.). Viral Pneumonia Screening on Chest X-Rays Using Confidence-Aware Anomaly Detection.