Pneumonia Detection using CNN with Implementation in Python

Prof.Manisha Raut
Manisharaut16@gmail.com
Dept.of Artificial Intelligence
Assist.Professor,GHRCE,Nagpur

Prof.Anirudha Bhagwat
anirudha.bhagwat@gmail.com
Dept.of Artificial Intelligence
Assist.Professor,GHRCE,Nagpur

Prof.Priti Gade
Priti.gade@gmail.com
Dept.of Artificial Intelligence
Assist.Professor,GHRCE,Nagpur

Prof.Vaishali Shende
vaishali.shende@gmail.com
Dept.of Artificial Intelligence
Assist.Professor,GHRCE,Nagpur

ABSTRACT

Pneumonia is a life-threatening infectious disease affecting one or both lungs in humans commonly caused by bacteria called Streptococcus pneumoniae. One in three deaths in India is caused due to pneumonia as reported by World Health Organization (WHO). Chest X-Rays which are used to diagnose pneumonia need expert radiotherapists for evaluation. Thus, developing an automatic system for detecting pneumonia would be beneficial for treating the disease without any delay, particularly in remote areas. Due to the success of deep learning algorithms in analyzing medical images, Convolutional Neural Networks (CNNs) have gained much attention for disease classification. In addition, features learned by pre-trained CNN models on large-scale datasets are much more useful in image classification tasks. This paper presents convolutional neural network models to accurately detect pneumonic lungs from chest X-rays, which can be utilized in the real world by medical practitioners to treat pneumonia. Experimentation was conducted on Chest X-Ray Images (Pneumonia) dataset available on Kaggle. Furthermore, recall and F1 scores are calculated from the confusion matrix of each model for better evaluation.

Keywords: - Artificial Intelligence, Pneumonia, CNN, ChestX-rays, Models, Algorithms, Classification

INTRODUCTION

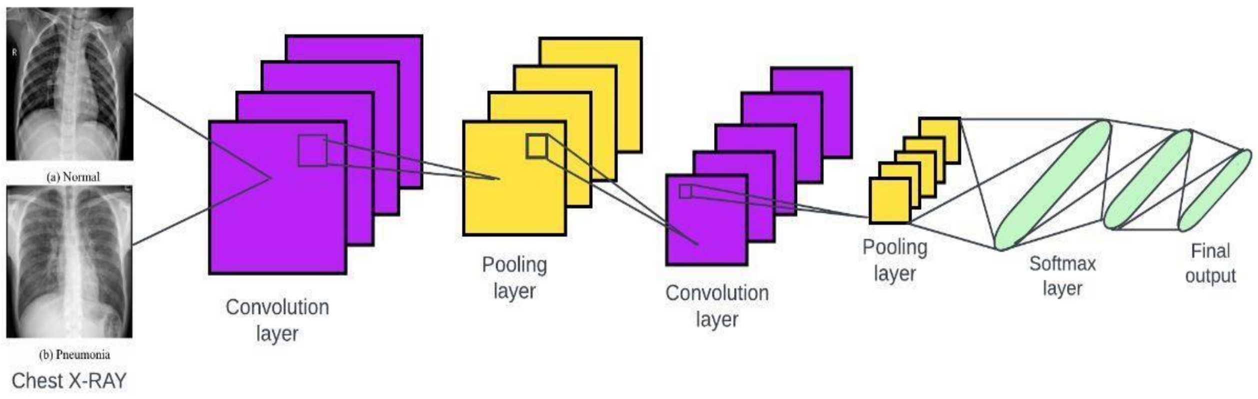
Pneumonia has many variants, it can be viral, bacterial, or fungal, and each of these variants may have various classes. Pneumonia is an infection of one or both of the lungs caused by bacteria, viruses, or fungi. It is a serious infection in which the air sacs fill with pus and other liquid. Pneumonia is often diagnosed via chest X-ray (CXR) or computed tomography (CT) scans, the former being the most used technique due to its reduced costs and availability in India. Timely detection of pneumonia can help to prevent the deaths of children. This paper presents convolutional neural network models to accurately detect pneumonic lungs from chest X-rays, which can be utilized in the real world by medical practitioners to treat pneumonia. These models have been trained to classify chest X-ray images into normal and pneumonia in a few seconds.

The objective of the paper is to develop CNN models from scratch which can classify and thus detect pneumonic patients from their chest X-rays with high validation accuracy, recall, and F1 scores. Recall is often favored in medical imaging cases over other performance-evaluating parameters, as it gives a measure of false negatives in the results. The number of false negatives in the result is very crucial in determining the real-world performance of models. If a model achieves high accuracy but low recall values, it is termed as underperforming, inefficacious, and even unsafe as higher falsenegative values imply a higher number of instances where the model is predicting a patient as normal, but in reality, the person is diseased. Hence, it would risk the patient's life. To prevent this, the focus would be only on models with great recall values, decent accuracies, and F1 scores.

In this study, we propose to build a novel CNN architecture to provide an accurate solution to the problem of pneumonia detection. The objective of the paper is to develop CNN models from scratch which can classify and thus detect pneumonic patients from their chest X-rays with high validation accuracy, recall, and F1 scores.

The paper is organized into 5 sections: Sect. 1 Introduction of Pneumonia, the importance of detection of Pneumonia, the purpose and motive of our research. Section 2 explores the earlier work related to this field that has been accomplished till now. Section 3 explains the methodology of the paper, explaining the architecture of the models, flowchart, and the dataset used to train and test the four models. Section-4 presents the results achieved by the various CNN models and compares the performance of each model using accuracy and loss graphs and confusion matrices. Section 5 provides a brief conclusion to the paper and delivers the best-suited model. Furthermore, the future scope of this work has also been discussed. All the references are mentioned at the end of report.

ARCHITECTURE



LITERATURE SURVEY/STUDY OF EXISTING Solution/Product

In 2016, Redmon et al. YOLO was proposed, which does not require a separate region proposal system, so the detection speed is very fast and can reach 45 FPS. In the same year, Liu et al. proposed the SSD algorithm. SSD and YOLO both win in detection speed, but SSD uses a multi-feature map for self-detection, the spatial resolution of the image in the deep network is greatly reduced, and it will be impossible to find small characters that are difficult to detect; reducing detection accuracy. YOLO does not use multivariate feature maps for independent identification. It smoothes the feature map and divides it with another low-resolution feature map, but defines the detection only as a regression problem, and the detection accuracy is low.

In 2014, Girshik et al. proposed R-CNN, which significantly increased the training speed. In the PASCAL VOC 2010 database, the map increased from 35.1% to 53.7%. In 2015, Ren et al proposed a faster R-CNN algorithm that uses RPN (regional recommendation network) to generate recommendations on feature maps.

In 2018, Lee et al. DetNet proposal, which is designed specifically for target detection and has better detection results with fewer layers. To avoid the large computational complexity and memory usage caused by high-resolution feature mapping, the network adopts a low-complexity extended barrier structure; The high resolution of the feature map is provided in the upper draw area. This paper is based on the concept of DetNet and a faster R-CNN framework for learning pneumonia detection.

PROPOSED METHODOLOGY/ SYSTEM ARCHITECTURE

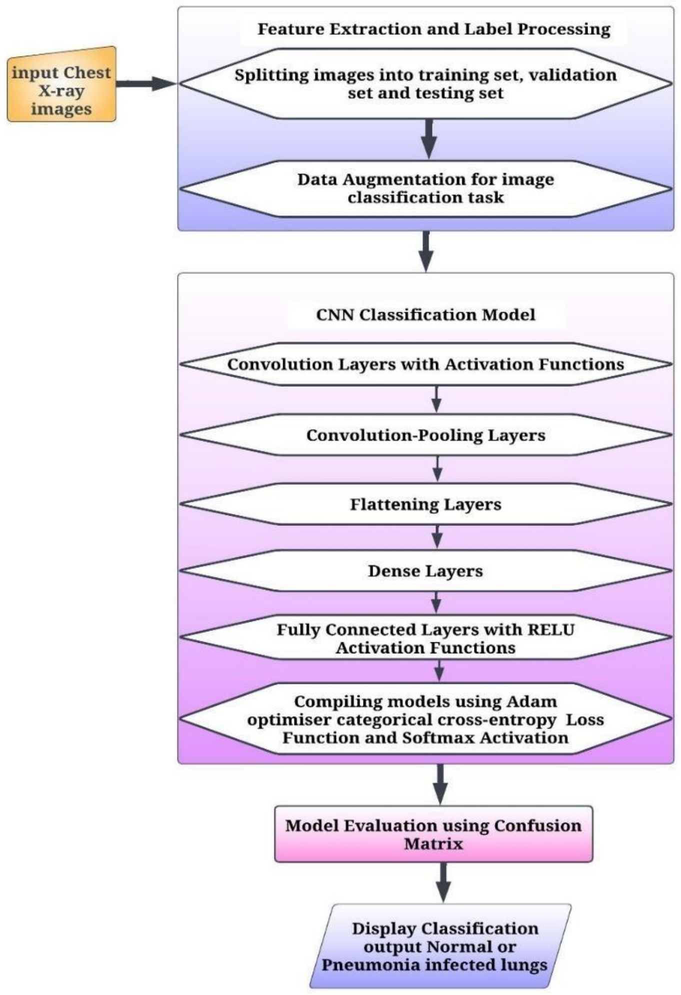

input Chest X-ray images
Feature Extraction and Label Processing
Splitting images into training set, validation set and testing set
Data Augmentation for image classification task
CNN Classification Model
Convolution Layers with Activation Functions
Convolution-Pooling Layers
Flattening Layers
Dense Layers
Fully Connected Layers with RELU Activation Functions
Compiling models using Adam optimiser categorical cross-entropy Loss Function and Softmax Activation
Model Evaluation using Confusion Matrix
Display Classification output Normal or Pneumonia infected lungs

Figure 1: Diagrammatic Representation of the experiment.

Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans: learn by example. Deep learning is a key technology behind driverless cars, enabling them to recognize a stop sign, or to distinguish a pedestrian from a lamppost. A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image, and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets can learn these filters/characteristics.

Dataset:

We will use the Chest X-Ray Images (Pneumonia) dataset by Paul Mooney as the data was already conveniently split into the train, test, and Val:

- Train - contains the training data/images for teaching our model.
- Val - contains images that we will use to validate our model. The purpose of this data set is to prevent our model from overfitting. Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the model's ability to generalize.
- Test - this contains the data that we use to test the model once it has learned the relationships between the images and their label (Pneumonia/Not-Pneumonia)

Models:

We will use the tqdm module to display progress bars - tqdm is a library in Python which is used for creating Progress Meters or Progress Bars. Another import I do here is ImageDataGenerator coming from Keras module. This module is going to help us implement image augmentation techniques during the training process. After loading images and creating labels we need to convert that into one-hot format. Luckily, we got a OneHotEncoder object taken from the Scikit-Learn module which is extremely helpful to do the conversion. Data argumentation-the main point of augmenting data — or more specifically augmenting train data — is that we are going to increase the number of data used for training by creating more samples with some sort of randomness on each of them. That randomness might include translations, rotations, scaling, shearing and flips. Such a technique can help our neural network classifier to reduce overfitting, or in other words, it can make the model generalize data samples better. Then I will have to build the neural network architecture. Let's start with the input layer (input1). So, this layer basically takes all the image samples in our X data. Hence, we need to ensure that the first layer accepts the exact same shape as the image size. It's worth noting that what we need to define is only (width, height, channels), instead of (samples, width, height, channels). We will get a confusion matrix which can tell that from all the images the number of images that suffer with bacteria, are normal and predict with virus pneumonia correctly.

Layers:

1. The Keras Python library makes creating deep learning models fast and easy. The sequential API allows you to create models layer-by-layer for most problems. It is limited in that it does not allow you to create models that share layers or have multiple inputs or outputs.

2. Keras Conv2D is a 2D Convolution Layer, this layer creates a convolution kernel that is wind with layers input which helps produce a tensor of outputs. (Note: - Kernel: In image processing kernel is a convolution matrix or mask which can be used for blurring, sharpening, embossing, edge detection and more by doing a convolution between a kernel and an image.

3. MaxPooling2D from keras.layers, which is used for pooling operations. For building this particular neural network, we are using a Max Pooling function, there exist different types of pooling operations like Min Pooling, Mean Pooling, etc. Here in MaxPooling we need the maximum value pixel from the respective region of interest.

4. Flatten from keras.layers, which is used for Flattening. Flattening is the process of converting all the resultant 2- dimensional arrays into a single long continuous linear vector.

5. Dense from keras.layers, which is used to perform the full connection of the neural network.

6. ImageDataGenerator, which Takes a batch of images and applies a series of random transformations to each image in the batch (including random

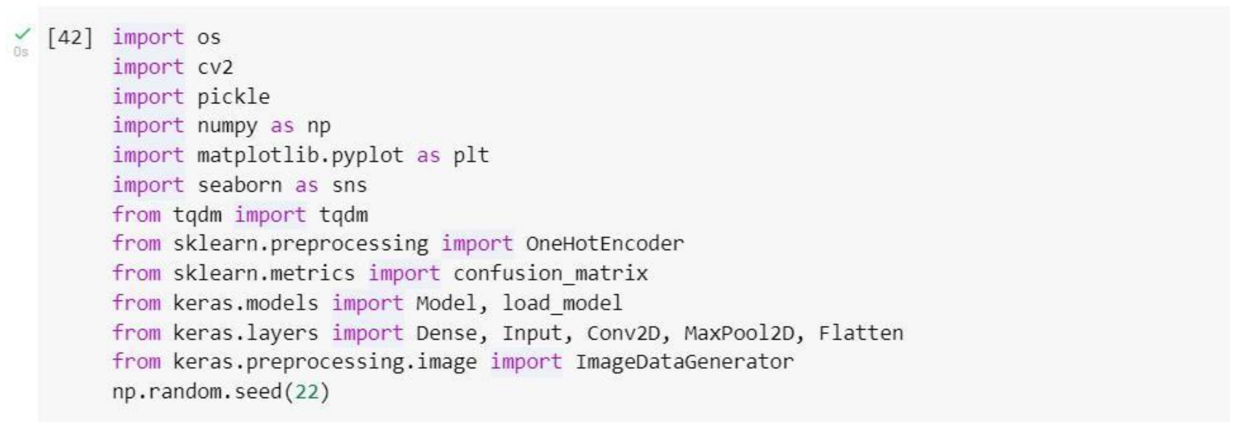
rotation, resizing, shearing, etc.) and then Replacing the original batch with the new, randomly transformed batch for training the CNN.

TOOLS AND TECHNOLOGIES APPLIED

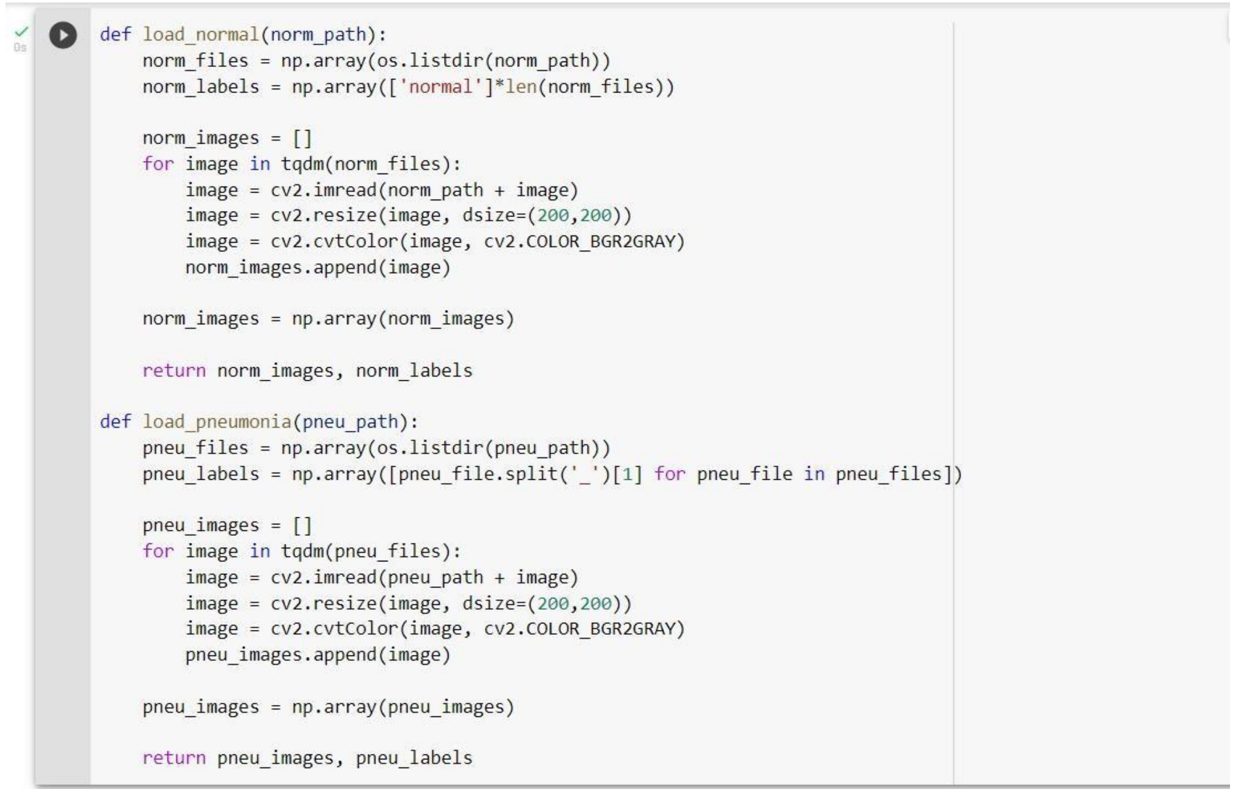
1) **Category:** Machine Learning and Deep Learning
2) **Programming Language:** Python
3) **Tools & Libraries:** ScikitLearn, Keras, MatploLib, ConvNets
4) **IDE:** Google Collaboratory
5) **Prerequisites:** Python, CNN & NLP

IMPLEMENTATION

Implementation of Pneumonia Detection using python



```
[42] import os
     import cv2
     import pickle
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
     from tqdm import tqdm
     from sklearn.preprocessing import OneHotEncoder
     from sklearn.metrics import confusion_matrix
     from keras.models import Model, load_model
     from keras.layers import Dense, Input, Conv2D, MaxPool2D, Flatten
     from keras.preprocessing.image import ImageDataGenerator
     np.random.seed(22)
```



```
def load_normal(norm_path):
    norm_files = np.array(os.listdir(norm_path))
    norm_labels = np.array(['normal']*len(norm_files))

    norm_images = []
    for image in tqdm(norm_files):
        image = cv2.imread(norm_path + image)
        image = cv2.resize(image, dsize=(200,200))
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        norm_images.append(image)

    norm_images = np.array(norm_images)

    return norm_images, norm_labels

def load_pneumonia(pneu_path):
    pneu_files = np.array(os.listdir(pneu_path))
    pneu_labels = np.array([pneu_file.split('_')[1] for pneu_file in pneu_files])

    pneu_images = []
    for image in tqdm(pneu_files):
        image = cv2.imread(pneu_path + image)
        image = cv2.resize(image, dsize=(200,200))
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        pneu_images.append(image)

    pneu_images = np.array(pneu_images)

    return pneu_images, pneu_labels
```

```
[44] norm_images, norm_labels = load_normal('/content/gdrive/MyDrive/chest_xray/train/NORMAL/')
     pneu_images, pneu_labels = load_pneumonia('/content/gdrive/MyDrive/chest_xray/train/PNEUMONIA/')

     100%|██████████| 1341/1341 [01:18<00:00, 17.07it/s]
     100%|██████████| 3875/3875 [02:33<00:00, 25.32it/s]
```

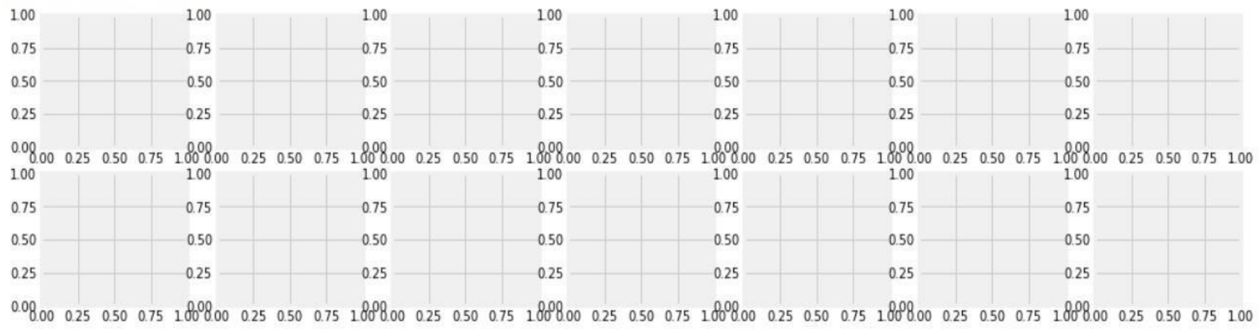
```
[48] X_train = np.append(norm_images, pneu_images, axis=0)
     y_train = np.append(norm_labels, pneu_labels,axis=0)
```

```
[49] print(X_train.shape)

     (5216, 200, 200)
```

```
[50] print(y_train.shape)

     (5216,)
```

```
[51] print(np.unique(y_train, return_counts=True))

     (array(['bacteria', 'normal', 'virus'], dtype='<U8'), array([2530, 1341, 1345]))
```

```
[52] print('Display several images')
     fig, axes = plt.subplots(ncols=7, nrows=2, figsize=(16, 4))

     Display several images
```



```
[53] indices = np.random.choice(len(X_train), 14)
     counter = 0
```

```
[54] for i in range(2):
         for j in range(7):
             axes[i,j].set_title(y_train[indices[counter]])
             axes[i,j].imshow(X_train[indices[counter]], cmap='gray')
             axes[i,j].get_xaxis().set_visible(False)
             axes[i,j].get_yaxis().set_visible(False)
             counter += 1
     plt.show()
```

```
[55] print('Loading test images')
```

```
Loading test images
```

```
[58] norm_images_test, norm_labels_test = load_normal('/content/gdrive/MyDrive/chest_xray/test/NORMAL/')
     pneu_images_test, pneu_labels_test = load_pneumonia('/content/gdrive/MyDrive/chest_xray/test/PNEUMONIA/')
```

```
100%|██████████| 234/234 [00:07<00:00, 29.81it/s]
100%|██████████| 390/390 [00:07<00:00, 49.18it/s]
```

```
[59] X_test = np.append(norm_images_test, pneu_images_test, axis=0)
     y_test = np.append(norm_labels_test, pneu_labels_test)
```

```
[60] with open('pneumonia_data.pickle', 'wb') as f:
         pickle.dump((X_train, X_test, y_train, y_test), f)
```

```
[61] with open('pneumonia_data.pickle', 'rb') as f:
         (X_train, X_test, y_train, y_test) = pickle.load(f)
```

```
[62] print('Label preprocessing')
```

```
Label preprocessing
```

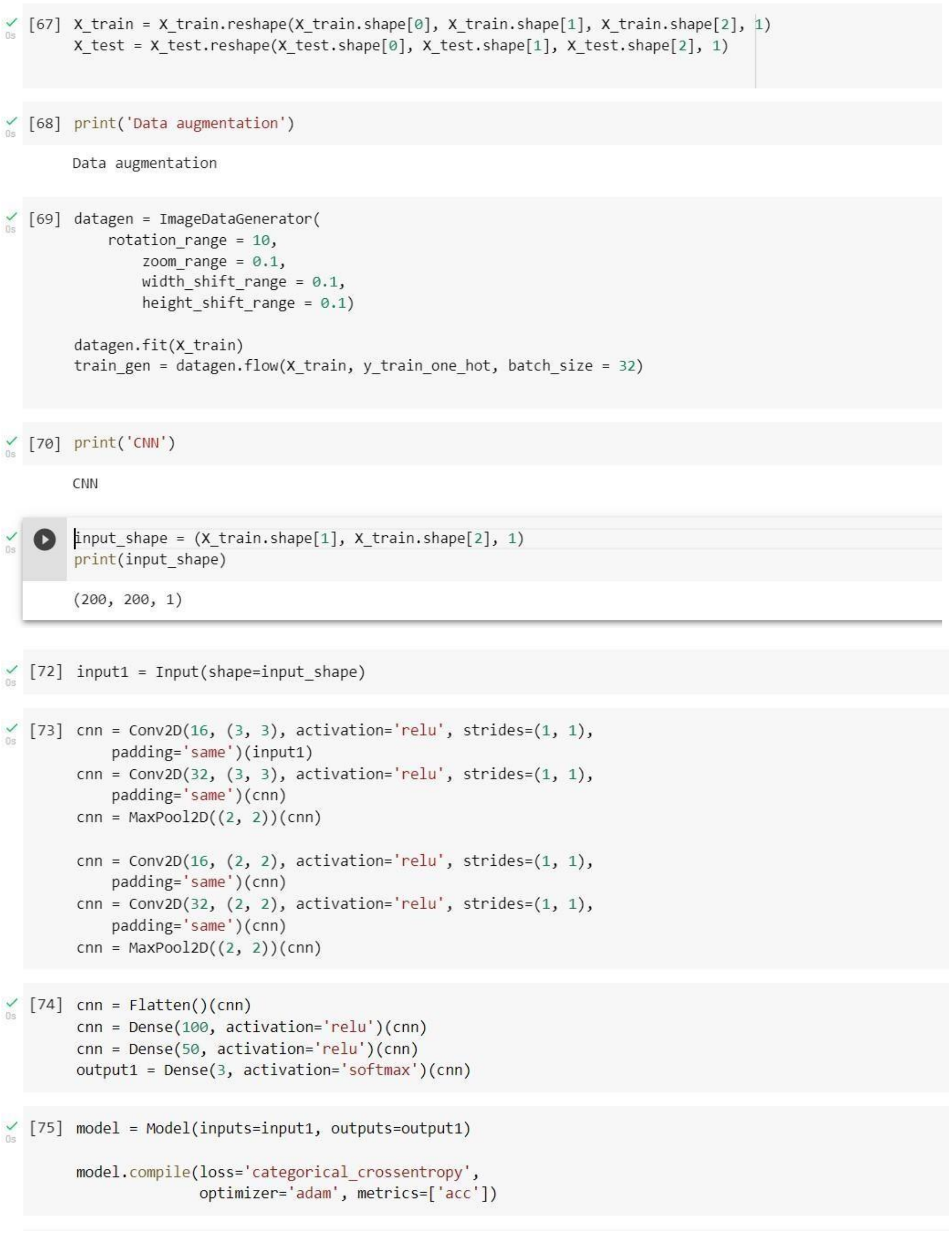
```
[63] y_train = y_train[:, np.newaxis]
     y_test = y_test[:, np.newaxis]
```

```
[64] one_hot_encoder = OneHotEncoder(sparse=False)
```

```
[65] y_train_one_hot = one_hot_encoder.fit_transform(y_train)
     y_test_one_hot = one_hot_encoder.transform(y_test)
```

```
[66] print('Reshaping X data')
```

```
Reshaping X data
```



[67]
```
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], X_train.shape[2], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], X_test.shape[2], 1)
```

[68]
```
print('Data augmentation')
```
```
Data augmentation
```

[69]
```
datagen = ImageDataGenerator(
    rotation_range = 10,
        zoom_range = 0.1,
        width_shift_range = 0.1,
        height_shift_range = 0.1)

datagen.fit(X_train)
train_gen = datagen.flow(X_train, y_train_one_hot, batch_size = 32)
```

[70]
```
print('CNN')
```
```
CNN
```

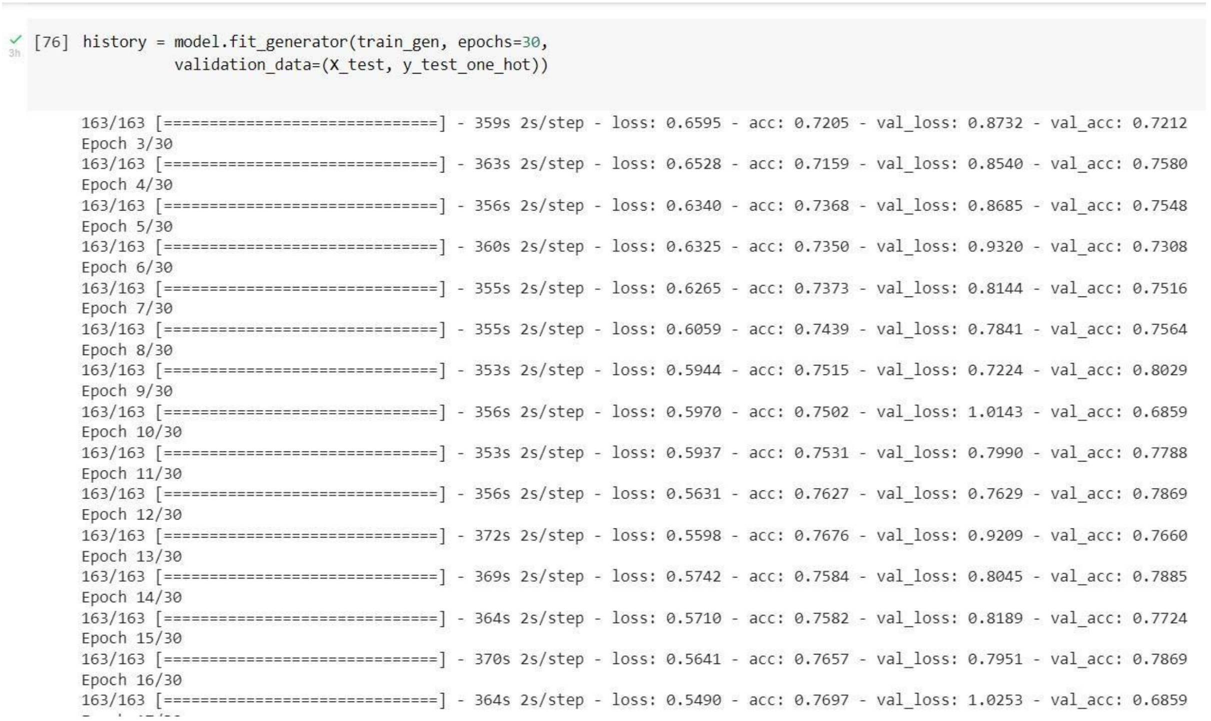
```
input_shape = (X_train.shape[1], X_train.shape[2], 1)
print(input_shape)
```
```
(200, 200, 1)
```

[72]
```
input1 = Input(shape=input_shape)
```

[73]
```
cnn = Conv2D(16, (3, 3), activation='relu', strides=(1, 1),
    padding='same')(input1)
cnn = Conv2D(32, (3, 3), activation='relu', strides=(1, 1),
    padding='same')(cnn)
cnn = MaxPool2D((2, 2))(cnn)

cnn = Conv2D(16, (2, 2), activation='relu', strides=(1, 1),
    padding='same')(cnn)
cnn = Conv2D(32, (2, 2), activation='relu', strides=(1, 1),
    padding='same')(cnn)
cnn = MaxPool2D((2, 2))(cnn)
```

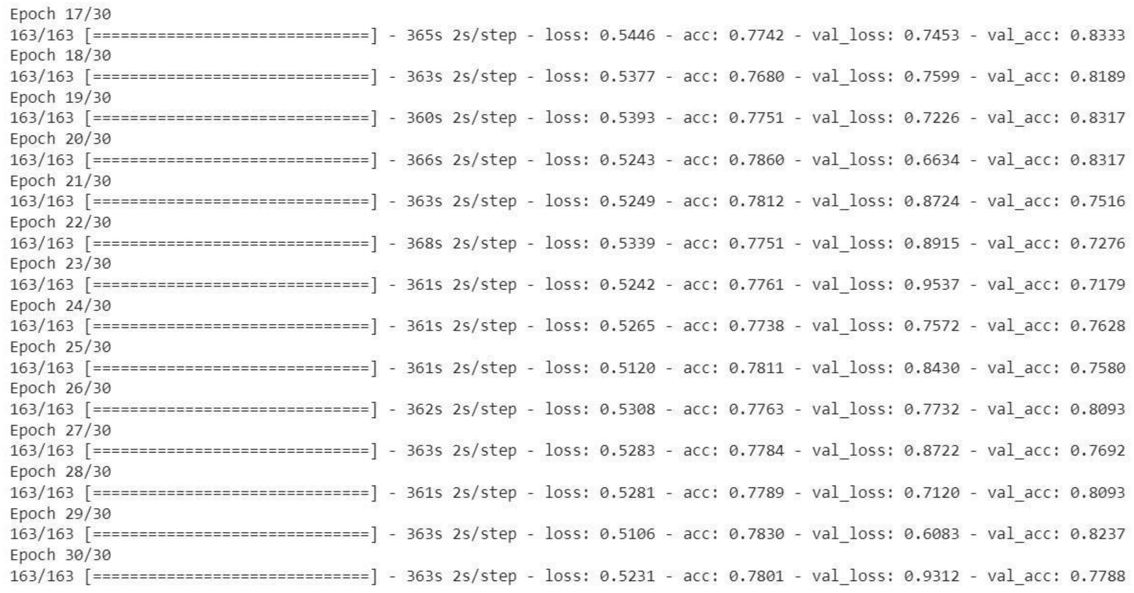
[74]
```
cnn = Flatten()(cnn)
cnn = Dense(100, activation='relu')(cnn)
cnn = Dense(50, activation='relu')(cnn)
output1 = Dense(3, activation='softmax')(cnn)
```

[75]
```
model = Model(inputs=input1, outputs=output1)

model.compile(loss='categorical_crossentropy',
              optimizer='adam', metrics=['acc'])
```



```
[76] history = model.fit_generator(train_gen, epochs=30,
               validation_data=(X_test, y_test_one_hot))
```

```
163/163 [==============================] - 359s 2s/step - loss: 0.6595 - acc: 0.7205 - val_loss: 0.8732 - val_acc: 0.7212
Epoch 3/30
163/163 [==============================] - 363s 2s/step - loss: 0.6528 - acc: 0.7159 - val_loss: 0.8540 - val_acc: 0.7580
Epoch 4/30
163/163 [==============================] - 356s 2s/step - loss: 0.6340 - acc: 0.7368 - val_loss: 0.8685 - val_acc: 0.7548
Epoch 5/30
163/163 [==============================] - 360s 2s/step - loss: 0.6325 - acc: 0.7350 - val_loss: 0.9320 - val_acc: 0.7308
Epoch 6/30
163/163 [==============================] - 355s 2s/step - loss: 0.6265 - acc: 0.7373 - val_loss: 0.8144 - val_acc: 0.7516
Epoch 7/30
163/163 [==============================] - 355s 2s/step - loss: 0.6059 - acc: 0.7439 - val_loss: 0.7841 - val_acc: 0.7564
Epoch 8/30
163/163 [==============================] - 353s 2s/step - loss: 0.5944 - acc: 0.7515 - val_loss: 0.7224 - val_acc: 0.8029
Epoch 9/30
163/163 [==============================] - 356s 2s/step - loss: 0.5970 - acc: 0.7502 - val_loss: 1.0143 - val_acc: 0.6859
Epoch 10/30
163/163 [==============================] - 353s 2s/step - loss: 0.5937 - acc: 0.7531 - val_loss: 0.7990 - val_acc: 0.7788
Epoch 11/30
163/163 [==============================] - 356s 2s/step - loss: 0.5631 - acc: 0.7627 - val_loss: 0.7629 - val_acc: 0.7869
Epoch 12/30
163/163 [==============================] - 372s 2s/step - loss: 0.5598 - acc: 0.7676 - val_loss: 0.9209 - val_acc: 0.7660
Epoch 13/30
163/163 [==============================] - 369s 2s/step - loss: 0.5742 - acc: 0.7584 - val_loss: 0.8045 - val_acc: 0.7885
Epoch 14/30
163/163 [==============================] - 364s 2s/step - loss: 0.5710 - acc: 0.7582 - val_loss: 0.8189 - val_acc: 0.7724
Epoch 15/30
163/163 [==============================] - 370s 2s/step - loss: 0.5641 - acc: 0.7657 - val_loss: 0.7951 - val_acc: 0.7869
Epoch 16/30
163/163 [==============================] - 364s 2s/step - loss: 0.5490 - acc: 0.7697 - val_loss: 1.0253 - val_acc: 0.6859
```

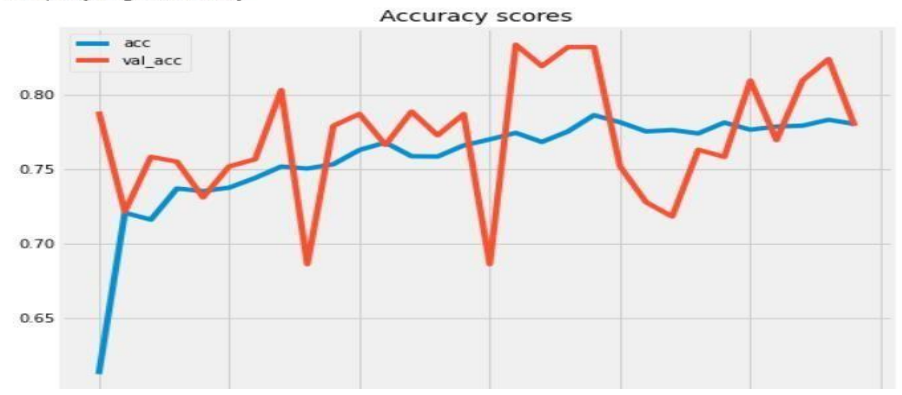


```
Epoch 17/30
163/163 [==============================] - 365s 2s/step - loss: 0.5446 - acc: 0.7742 - val_loss: 0.7453 - val_acc: 0.8333
Epoch 18/30
163/163 [==============================] - 363s 2s/step - loss: 0.5377 - acc: 0.7680 - val_loss: 0.7599 - val_acc: 0.8189
Epoch 19/30
163/163 [==============================] - 360s 2s/step - loss: 0.5393 - acc: 0.7751 - val_loss: 0.7226 - val_acc: 0.8317
Epoch 20/30
163/163 [==============================] - 366s 2s/step - loss: 0.5243 - acc: 0.7860 - val_loss: 0.6634 - val_acc: 0.8317
Epoch 21/30
163/163 [==============================] - 363s 2s/step - loss: 0.5249 - acc: 0.7812 - val_loss: 0.8724 - val_acc: 0.7516
Epoch 22/30
163/163 [==============================] - 368s 2s/step - loss: 0.5339 - acc: 0.7751 - val_loss: 0.8915 - val_acc: 0.7276
Epoch 23/30
163/163 [==============================] - 361s 2s/step - loss: 0.5242 - acc: 0.7761 - val_loss: 0.9537 - val_acc: 0.7179
Epoch 24/30
163/163 [==============================] - 361s 2s/step - loss: 0.5265 - acc: 0.7738 - val_loss: 0.7572 - val_acc: 0.7628
Epoch 25/30
163/163 [==============================] - 361s 2s/step - loss: 0.5120 - acc: 0.7811 - val_loss: 0.8430 - val_acc: 0.7580
Epoch 26/30
163/163 [==============================] - 362s 2s/step - loss: 0.5308 - acc: 0.7763 - val_loss: 0.7732 - val_acc: 0.8093
Epoch 27/30
163/163 [==============================] - 363s 2s/step - loss: 0.5283 - acc: 0.7784 - val_loss: 0.8722 - val_acc: 0.7692
Epoch 28/30
163/163 [==============================] - 361s 2s/step - loss: 0.5281 - acc: 0.7789 - val_loss: 0.7120 - val_acc: 0.8093
Epoch 29/30
163/163 [==============================] - 363s 2s/step - loss: 0.5106 - acc: 0.7830 - val_loss: 0.6083 - val_acc: 0.8237
Epoch 30/30
163/163 [==============================] - 363s 2s/step - loss: 0.5231 - acc: 0.7801 - val_loss: 0.9312 - val_acc: 0.7788
```

```
[77] model.save('pneumonia_cnn.h5')
```

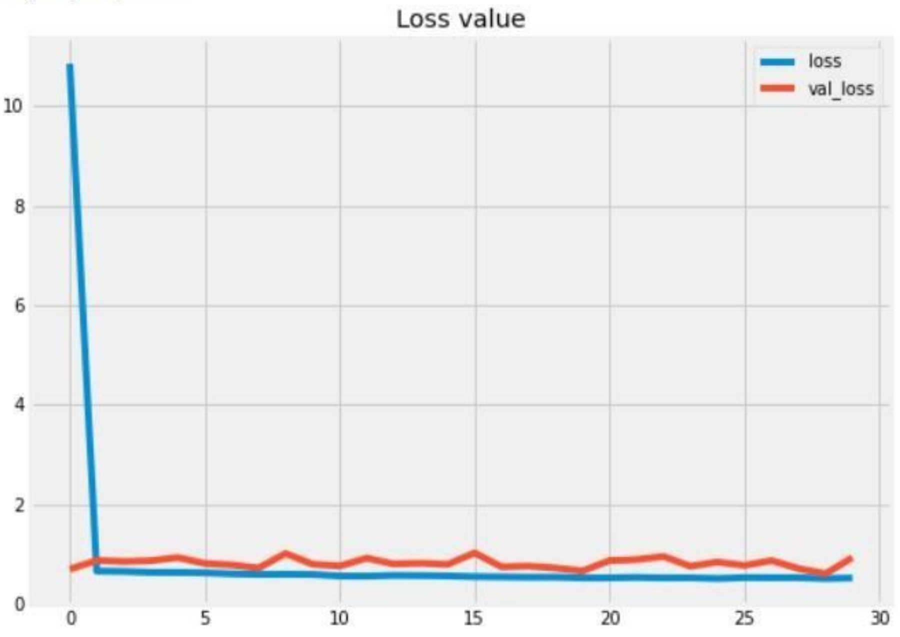
```
[81] print('Displaying accuracy')
     plt.figure(figsize=(8,6))
     plt.title('Accuracy scores')
     plt.plot(history.history['acc'])
     plt.plot(history.history['val_acc'])
     plt.legend(['acc', 'val_acc'])
     plt.show()
```

```
Displaying accuracy
```



```
[80] print('Displaying loss')
     plt.figure(figsize=(8,6))
     plt.title('Loss value')
     plt.plot(history.history['loss'])
     plt.plot(history.history['val_loss'])
     plt.legend(['loss', 'val_loss'])
     plt.show()
```

```
Displaying loss
```



```
[82] predictions = model.predict(X_test)
     print(predictions)
```

```
20/20 [==============================] - 11s 543ms/step
[[1.45663962e-01 7.38606095e-01 1.15730025e-01]
 [1.43420711e-01 7.45171189e-01 1.11408107e-01]
 [5.30836225e-01 1.86655045e-01 2.82508671e-01]
 ...
 [1.83226131e-02 5.40175506e-06 9.81671989e-01]
 [6.11001775e-02 4.02910780e-04 9.38496947e-01]
 [5.19407511e-01 1.52530791e-02 4.65339422e-01]]
```

```
[83] predictions = one_hot_encoder.inverse_transform(predictions)
```

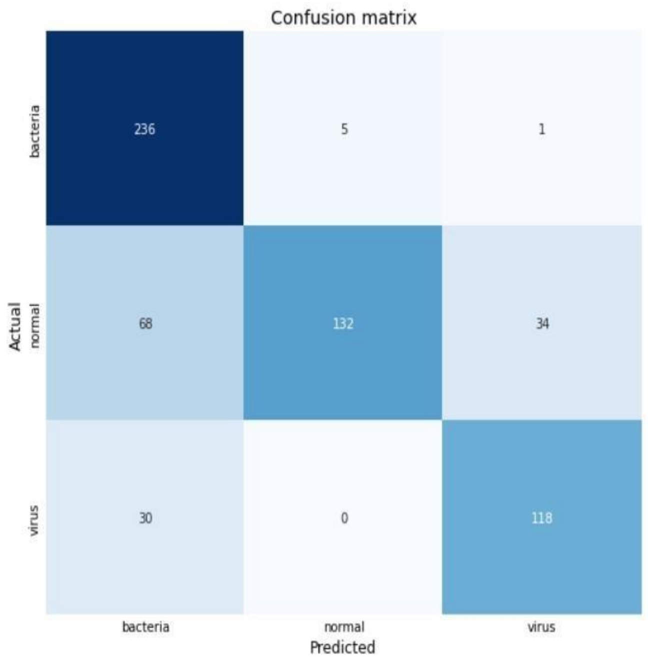
```
[84] print('Model evaluation')
     print(one_hot_encoder.categories_)
```

```
Model evaluation
[array(['bacteria', 'normal', 'virus'], dtype='<U8')]
```

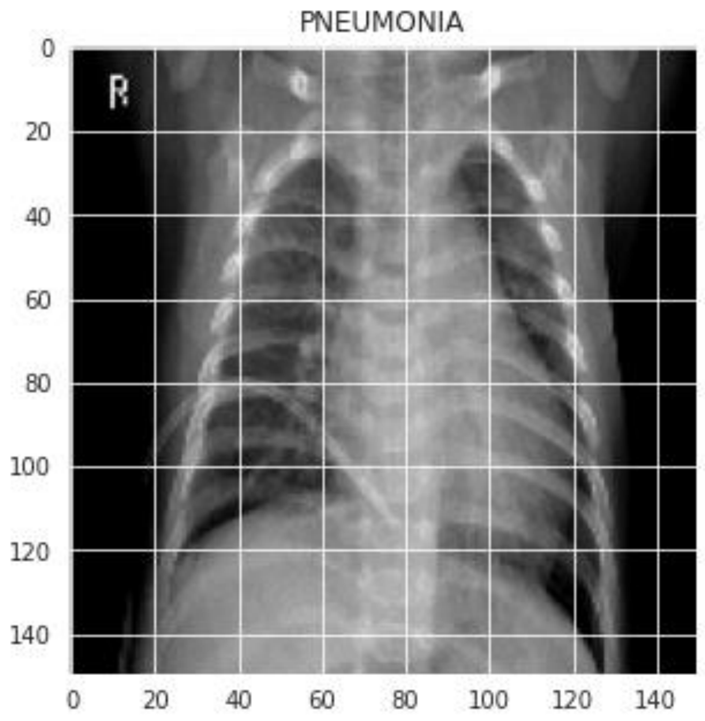
```
[85] classnames = ['bacteria', 'normal', 'virus']
```

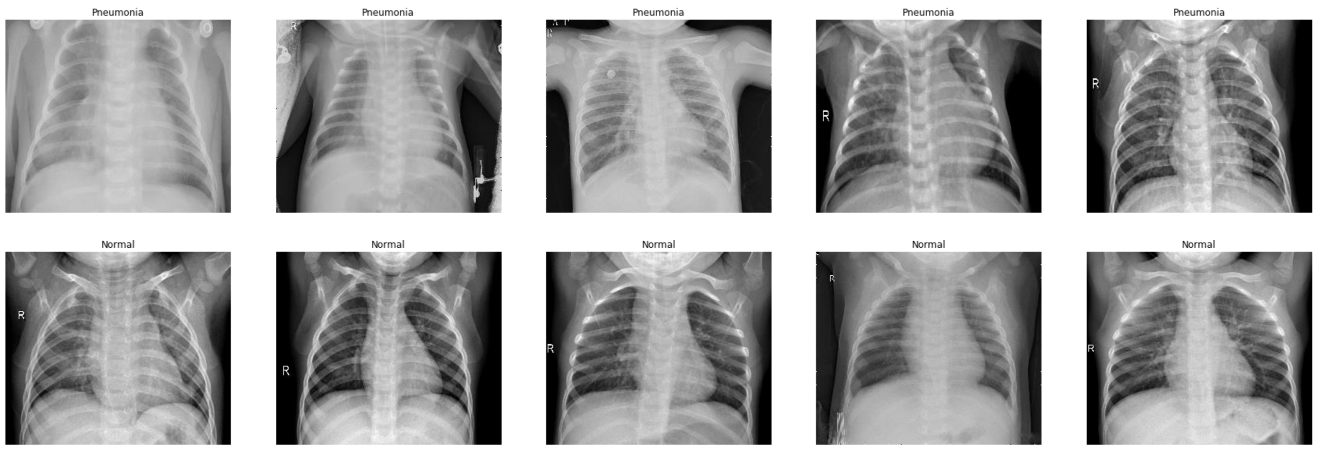
```
[86] cm = confusion_matrix(y_test, predictions)
```

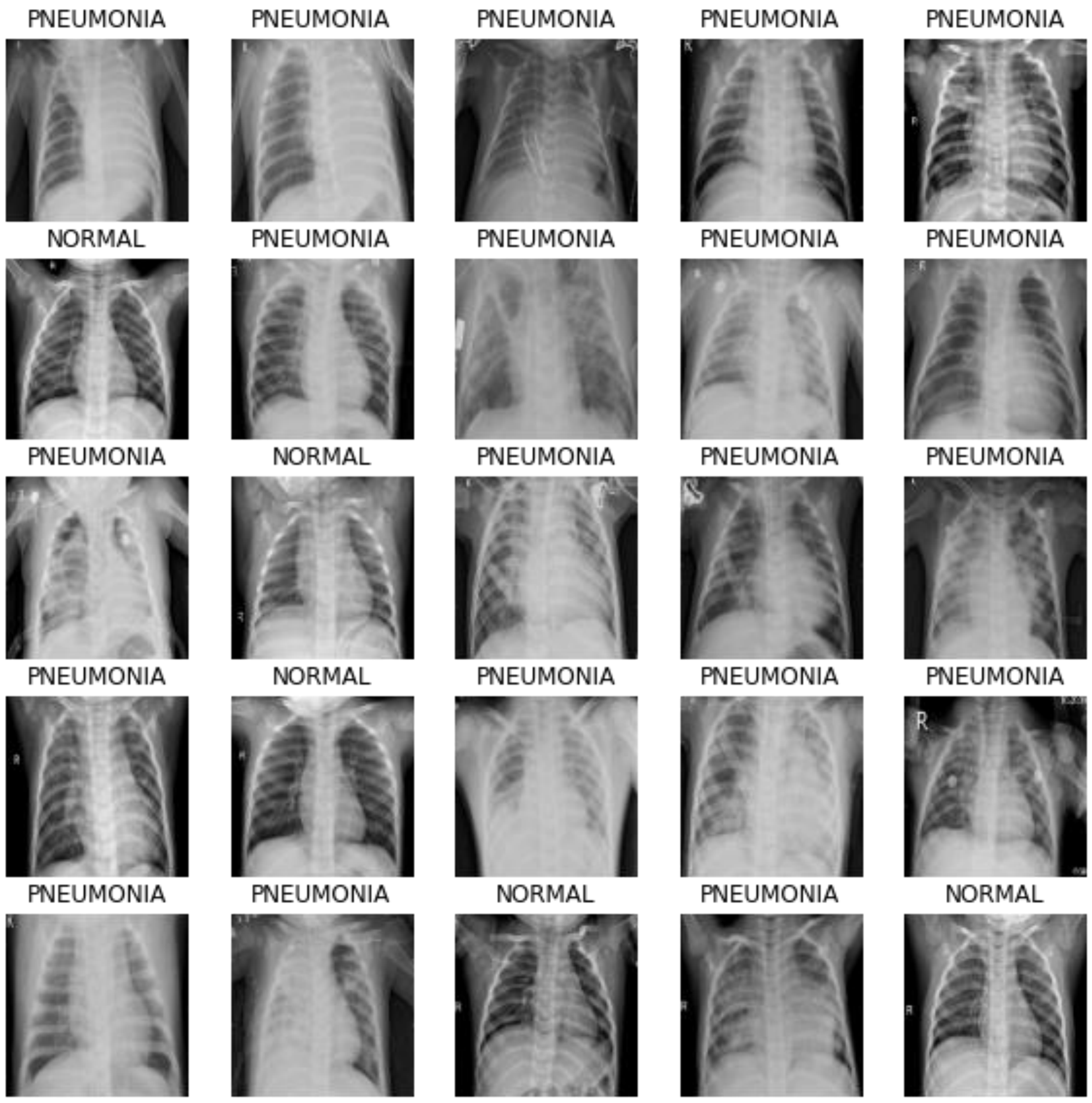
```
[87] plt.figure(figsize=(8,8))
     plt.title('Confusion matrix')
     sns.heatmap(cm, cbar=False, xticklabels=classnames, yticklabels=classnames, fmt='d', annot=True, cmap=plt.cm.Blues)
     plt.xlabel('Predicted')
     plt.ylabel('Actual')
     plt.show()
```



Output


PNEUMONIA
R
0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


Pneumonia
Pneumonia
Pneumonia
Pneumonia
Pneumonia
Normal
Normal
Normal
Normal
Normal



In this project, the prediction of pneumonia was conducted utilizing a dataset comprising various images. The model's accuracy was measured at 79%, signifying its ability to accurately predict pneumonia when the input image closely resembles those present in the dataset. This indicates a notable level of performance within the confines of the dataset used for training. Moreover, there is a suggestion that employing a larger and more diverse dataset could potentially lead to further improvements in the model's accuracy. This approach is commonly adopted in machine learning to enhance the robustness and generalizability of predictive models.

RESULT

We proposed a CNN model to provide an efficient and accurate solution for the pneumonia detection problem based on X-ray images. Instead of deploying pretrained networks via transfer learning, we can create a CNN model from scratch. Experiments have shown that the proposed model performs better than its counter candidates in terms of accuracy and efficiency, achieving recall and precision well above 97% with predictions produced in only 122 ms. Thus, it is concluded that the CNN classifier can be effectively used by medical officers for diagnostics purposes for early detection of pneumonia in children as well as adults. A large number of X-ray images can be processed very quickly to provide highly precise diagnostics results, thus helping healthcare systems provide efficient patient care services and reduce mortality rates. These convolution neural network models can be achieved by employing various methods of parameters adding dropout, changing learning rates, changing the batch size, and the number of epochs, adding more complex fully connected layers, and changing various stochastic gradient optimizers. The presence of expert radiologists is the topmost necessity to properly diagnose any kind of thoracic disease. This study primarily aims to improve medical adeptness in areas where the availability of radiotherapists is still limited.

CONCLUSION

Our study facilitates the early diagnosis of Pneumonia to prevent adverse consequences (including death) in such remote areas. So far, not much work has been contributed specifically to detect Pneumonia from the Kaggle dataset. With the series of experiments conducted, we aim to provide the dominating pre-trained CNN model and classifier for future work in a similar research domain.

Our study will likely lead to the development of better algorithms for detecting Pneumonia in the foreseeable future. Lastly, It is fairly easy for any developer with decent programming skills to create Machine Learning models which could be useful to millions of people. With future development, improvement in ML technology, and most importantly — the involvement of more and more people working on it — we will be able to solve more of such problems.

REFERENCES

1. https://www.analyticsvidhya.com/blog/2020/09/pneumonia-detection-usingcnn-with-implementation-in-python/

2. https://ieeexplore.ieee.org/document/8869364/keywords#keywords

3. https://www.researchgate.net/publication/340961287_Pneumonia_Detection_U
sing_Convolutional_Neural_Networks_CNNs#:~:text=Timely%20detection%20of%
20pneumonia%20in,medical%20practitioners%20to%20treat%20pneumonia.

4. https://www.sciencedirect.com/science/article/pii/S0208521622000742

5. https://data.mendeley.com/datasets/rscbjbr9sj/2

6. L. Liu, S. Oza, D. Hogan et al., "Global, regional, and national causes of under-5 mortality in 2000-15: an updated systematic analysis with implications for the Sustainable Development Goals," The Lancet, vol. 388, no. 10063, pp. 3027–3035, 2016. View at: Publisher Site | Google Scholar

7. P. Rajpurkar, J. Irvin, K. Zhu et al., "Chexnet: radiologist-level pneumonia detection on chest X-rays with deep learning," 2017, https://arxiv.org/abs/1711.05225. View at: Google Scholar

8. S.-A. Zhou and A. Brahme, "Development of phase-contrast X-ray imaging techniques and potential medical applications," Physica Medica, vol. 24, no. 3, pp. 129–148, 2008. View at: Google Scholar

9. T. Paulraj, K. S. Chelliah, and S. Chinnasamy, "Lung computed axial tomography image segmentation using possibilistic fuzzy C-means approach for computer aided diagnosis system," International Journal of Imaging Systems and Technology, vol. 29, no. 3, pp. 374–381, 2019. View at: Google Scholar

10. R. H. Kallet, “The vexing problem of ventilator-associated pneumonia: observations on pathophysiology, public policy, and clinical science,” Respiratory Care, vol. 60, no. 10, pp. 1495–1508, 2015. View at: Google Scholar