Machine Learning and Big Data
DATA 622

City University of New York
School of Professional Studies

Professor Joseph Sabelja

Homework 4 (Final project)

Date: Dec 04, 2022

Integrate Image Recognition technology in Radiology Practice

Prepared by
Anjal Hussan

Table of Contents

Abstract

The purpose of this study is to explore some of the image recognition technology that exists currently and how we can take leverage this technology in radiology. Although this paper does not provide technical details, it will be possible to get the idea of main concept and the technology that handles the task of image processing and classification. The study also seeks to answer various questions for example, how image recognition works? How can we integrate image recognition technology with radiology practice? Is image recognition technology reliable? Can Machine Learning (ML) applications help the Radiologist to detect anomalies as well as other abnormalities?

Keywords

Convolutional Neural Network; Deep learning; Transfer learning; Radiology; Medical Image; CNN; X-Ray Image; Image recognition

Introduction

Various clinical diagnosis requires medical imaging, such as early detection, monitoring, diagnosis, and treatment evaluation of various health conditions. In the health care system, there has been a significant increase in demand for medical image services, e.g., Radiography, endoscopy, Computed Tomography (CT), Mammography Images (MG), Ultrasound images, Magnetic Resonance Imaging (MRI), Magnetic Resonance Angiography (MRA), Nuclear medicine imaging, Positron Emission Tomography (PET) and pathological tests. Due to a shortage of radiologists, medical images can often be challenging to analyze and time-consuming. Machine Learning (ML) applications can help the Radiologist to detect anomalies as well as other abnormalities. ML applications can function without being specifically programmed, which learn from data and make predictions or decisions based on past data.

The term machine learning refers to a wide range of statistical techniques to analyze algorithms that iteratively improve in response to training data to build models for autonomous predictions. In other words, computer programs become more efficient as they learn from data. The objective of a machine learning algorithm is to develop a mathematical model that fits the data. Once this model fits known data, it can be used to predict the labels of new data. The field of radiology involves inherently interpreting data—extracting features from images and analyzing them with the help of a wide knowledge base that includes these features—which provides an excellent opportunity for implementing ML tools in practice.

Feasibility and approach

Medical image analysis using modern technology is growing interest recently due to the need for efficient and objective evaluation of large quantities of data. As the amount of data is growing with the help of modern devices, it's getting inevitable to apply different image recognition methods for the analysis of medical images. With the help of ongoing studies, the methods are becoming more mature and coming closer to routine clinical application. Although there are challenges involved in reading medical images, the qualities of different methods are getting developed that can be applied to standard clinical images. Medical image analysis is very critical when another type of screening provides high false negative results. Modern machine Learning (ML) and deep learning (DL) offer fast, automated, and effective methods to identify abnormalities as well as extract key features from the image which can be used for further study.

The use of medical imaging facilitates early detection, diagnosis, and treatment evaluation of various medical conditions in various clinical applications. Among several methods available in Machine learning, Deep Learning is considered one of the most sophisticated and effective methods in ML to predict the outcome. Deep Learning Approach (DLA) in medical image analysis emerges as a fast-growing research field. DLA has been widely used in medical imaging to detect the presence or absence of the disease. There are multiple papers available that discussed the development of artificial neural networks, and comprehensive analysis of DLA, which delivers promising medical imaging applications. Most of the DLA implementations focus on X-ray images, computerized tomography, mammography images, and digital histopathology images. Those provide a methodical review for the classification, detection, and segmentation of medical images based on DLA. Among all DLA method, the Convolutional Neural Network (CNN) is very suitable for image and speech recognition. The built-in layers reduce the high dimensionality of the

images without losing too much feature. That is why CNNs are especially suited for image recognition and classification. For our study and for the purpose of this paper, we will use CNN to process the images and classify the images.

Image processing and classification

Image classification refers to the labelling of images into one of a few predefined classes. There are potentially n number of classes in which a given image can be classified.

Preprocessing: Steps for image pre-processing includes Reading image, Resizing image, and Data Augmentation (Gray scaling of image, Reflection, Gaussian Blurring, Histogram, Equalization, Rotation, and Translation).

Detection refers to the localization of an object which means the segmentation of the image and identifying the position of the object of interest. The purpose of the segmentation is to divide the image into different parts. These parts are named as region of interest (ROI). Human has a special vision system which separates ROI quickly without any major effort but for computer, it's an complex task to process the image. Luckily there are many methodologies and tools have been developed and we can use them in the industry to process the image and identify the ROI easily with the help of powerful computer. For our study, we used python along with some library provided by TensorFlow.

Feature extraction and training: Feature extraction is the process of isolating and identifying a set of quantitative quality for region of interest ROI. This is a crucial step wherein statistical or deep learning methods are used to identify the most interesting patterns of the image, features that might be unique to a particular class and that will, later, help the model to differentiate between different classes. This process where the model learns the features from the dataset is called model training.

Classification of the object: After ROI segmentation and feature extraction, each image is uniquely noted using numerical signifiers. Using the unique signifiers, the machine

learning algorithm categorizes detected objects into predefined classes by using a suitable classification technique that compares the image patterns with the target patterns.

Material and methods

The summary of the datasets used and the methodology are as follows.

Datasets: A total of 5855 chest X-Ray images were downloaded from an open-source databases. The Name of the dataset is "Labeled Optical Coherence Tomography (OCT) and Chest X-Ray Images for Classification" the data set is publicly available in the Mendeley.com. and the X-Ray images are from unique individual patients. The source has already divided the images into 3 parts – train, test and val. All three folders contains both normal X-Ray images and the X-Ray images from pneumonia patients. Since the sample data was used in many different studies, we are confident that there is no bad data in there.

Table I

Data Set	Normal	Pneumonia
Training Data Set	1341	3875
Test Data Set	234	390
Val Data Set	8	8

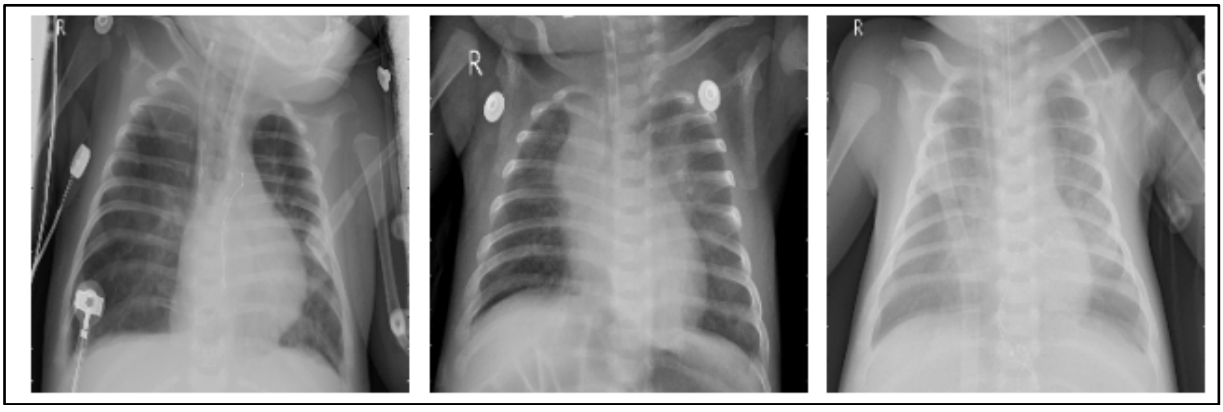


Fig.1: Sample X-Ray Images of pneumonia class

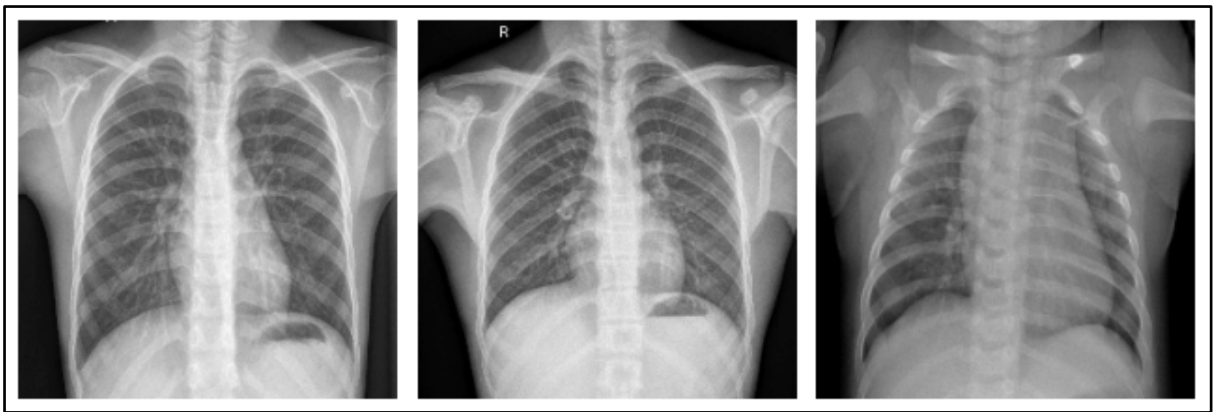


Fig.1: Sample X-Ray Images of Normal class

Methods: Training a CNN model for image classification is typically difficult because it essentially requires large training data which goes through the layers of the model and the model extracts the features from the images and learns from the images. In radiology, it's even more complex and expert annotation is required to apply CNN Model in the radiology. We started with loading the images into the computer memory with the help of ImageDataGenerator from keras.preprocessing.image. We also visualized the image to make sure that the image was loaded properly in the computer memory. Once we are comfortable with the loaded data, we defined a sequential CNN Model. The model we defined is started with a Convolution layer which essentially the main building block of a CNN Model. This layer contains set of filters or kernels which accepts the parameters and pass to the next layer. Before passing to the next layer, the convolution layer extracts the feature and produce a feature map. The feature map is basically all the pixels in its receptive field into a single value. In our case, the convolution layer decreases the image size as well as brings all the information in the field together into a single pixel. The final output of the convolutional layer is a vector.

Right after the convolution layer, we have added a pooling layers which help down sampling feature maps by summarizing the presence of features in patches of the feature map.

Two common pooling methods are average pooling and max pooling that summarize the average presence of a feature and the most activated presence of a feature respectively. In our case, we used MaxPooling methods since MaxPooling selects the maximum value from every pool. Max Pooling retains the most prominent features of the feature map, and the returned image is sharper than the original image.

In our CNN Model, we added another Convolution layer which accepts input from the 1st pooling layer and transforms the image to extract features from it. Each time the image goes through the layers, our model learns some features from the image.

After the 2nd convolution layer, we added another pooling layer with MaxPooling methods to filter out the prominent feature of the feature map.

Then, we add a layer called Flatten layer which used to make the multidimensional input one-dimensional, commonly used in the transition from the convolution layer to the full connected layer.

Lastly, we compile the model with an optimizer. The optimizer we used is the Adam optimizer. Adam is a replacement optimization algorithm for stochastic gradient descent for training deep learning models. Adam combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems. Adam is relatively easy to configure where the default configuration parameters do well on most problems. Once the model is compiled, we trained the model using the "fit_generator" method. This fit_generator accepts the train data set which we prepared earlier using ImageDataGenerator method from Tensorflow.Keras library. We tried 25 epochs which essentially means that the fit method will run through the training data set 25 time to learn about the images to classify between normal X-ray images and images with pneumonia. Next the models were tested using the test data and let the model predict the class of the images in the test data. Using the class of test data and the predicted class of the test

data, we calculated the accuracy of the CNN Model. In our case, the accuracy of the CNN Model is: 92.63%

In comparison, we also trained a SVM model with our training data with the help of the sklearn library. With the SVM Model, we also predicted the classification using xtest and the accuracy of the SVM model is 76.44%. We also plotted the confusion matrix for the SVM model which indicated that 147 labels were wrong out of 624 rows of data. That's a lot of wrong classification.

Model	**Accuracy**
CNN	92.63%
SVM	76.44%

Limitations and challenges

Using our training data, although it was possible to apply deep learning algorithms like CNN to medical imaging, there are many challenges that is hindering the progress. Some of the challenges are listed below:

- Poor Data Quality – low resolution, low contrast
- Limited data – there are only non-standard medical data available for analysis.
- Medical data privacy – For deep learning algorithms, we need huge number of data, but medical data privacy put a hard stop in data acquisition.
- Medical image classification needs expert knowledge which makes it harder to obtain annotated medical data.

Ethical dilemmas

Although the results of using Machine Learning technology to review medical images is phenomenal, the technologist should consider ethical and legal risk very seriously. The radiologists, on the other hand, should be able to evaluate machine learning projects, including common algorithms, supervised as opposed to unsupervised techniques, statistical pitfalls, and data considerations for training and evaluation.

Geis and Spencer et al, published an article focusing on the ethical use of AI in radiology and mentioned that using AI in radiology should bring well-being, minimize harm. The article also highlights that the "AI in radiology should be appropriately transparent and highly dependable, curtail bias in decision making, and ensure that responsibility and accountability remain with human designers or operators". In addition to the data scientist, the radiologists will remain responsible to acquire new skills to do their best for patients in the new AI ecosystem.

Conclusion

There are several deep learning methods and associated algorithms / learning available now a days. Several optimized techniques and frameworks also developed around those deep learning methods. In this study we have discovered that the CNN based models have higher accuracy when it comes to apply deep learning methods to medical image applications, including classification, segmentation, and detection. Using the modern deep learning algorithm to clinical decisions, the workflow for a radiologist can be automated and help making decision faster. But it should be noted that technology to automate identify and classify is only to help the radiologist and medical provider not to replace them. Successful integration between deep learning application to radiology will certainly require the radiologist to supervise and accurately interpreted the output.

References

Corradini, D., Brizi, L., Gaudiano, C., Bianchi, L., Marcelli, E., Golfieri, R., Schiavina, R., Testa, C., & Remondini, D. (2021, August 5). Challenges in the Use of Artificial Intelligence for Prostate Cancer Diagnosis from Multiparametric Imaging Data. Cancers, 13(16), 3944. https://doi.org/10.3390/cancers13163944

Coudray, N., Ocampo, P. S., Sakellaropoulos, T., Narula, N., Snuderl, M., Fenyö, D., Moreira, A. L., Razavian, N., & Tsirigos, A. (2018). Classification and mutation prediction from non-small cell lung cancer histopathology images using deep learning. Nature medicine, 24(10), 1559–1567. https://doi.org/10.1038/s41591-018-0177-5

Geis, J. R., Brady, A. P., Wu, C. C., Spencer, J., Ranschaert, E., Jaremko, J. L., Langer, S. G., Borondy Kitts, A., Birch, J., Shields, W. F., van den Hoven van Genderen, R., Kotter, E., Wawira Gichoya, J., Cook, T. S., Morgan, M. B., Tang, A., Safdar, N. M., & Kohli, M. (2019, November). Ethics of Artificial Intelligence in Radiology: Summary of the Joint European and North American Multisociety Statement. Radiology, 293(2), 436–440. https://doi.org/10.1148/radiol.2019191586

Kohli, Marc, et al. "Implementing Machine Learning in Radiology Practice and Research." American Journal of Roentgenology, vol. 208, no. 4, 2017, pp. 754–760., https://doi.org/10.2214/ajr.16.17224.

Tripathi, S., & Musiolik, T. H. (2022). Fairness and Ethics in Artificial Intelligence-Based Medical Imaging. In T. Musiolik, & A. Dingli (Ed.), Ethical Implications of Reshaping Healthcare With Emerging Technologies (pp. 71-85). IGI Global. https://doi.org/10.4018/978-1-7998-7888-9

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

https://machinelearningmastery.com

Appendix A. Exploratory data analysis

The dataset contains Chest X-Ray images and are already split into a training set and a testing set of independent patients. The dataset was used in many studies and have been labelled properly. We will explore the data sets and perform the exploratory data analysis in python.

First, we start by importing libraries in the Python notebook:

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
import plotly.figure_factory as ff

import keras
from keras.models import Sequential
from keras.layers import Dense, Conv2D , MaxPool2D , Flatten , Dropout , BatchNormalization
from keras.callbacks import ReduceLROnPlateau
from keras.preprocessing.image import ImageDataGenerator

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import accuracy_score


import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import callbacks
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator


import cv2
import warnings
warnings.filterwarnings('ignore')

import random
import os
import glob
from numpy.random import
```

Once we import the libraries it's time to read the data from our local directory. The data is already divided into train and test and val directory. Let's define the path for the directories.

```
main_path = "chest_xray_images/chest_xray/"


train_path = os.path.join(main_path,"train")
test_path=os.path.join(main_path,"test")
val_path=os.path.join(main_path,"val")

train_normal = glob.glob(train_path+"/NORMAL/*.jpeg")
train_pneumonia = glob.glob(train_path+"/PNEUMONIA/*.jpeg")

test_normal = glob.glob(test_path+"/NORMAL/*.jpeg")
test_pneumonia = glob.glob(test_path+"/PNEUMONIA/*.jpeg")

val_normal = glob.glob(val_path+"/NORMAL/*.jpeg")
val_pneumonia = glob.glob(val_path+"/PNEUMONIA/*.jpeg")
```

Now that we have the directories defined, we can combine the normal and pneumonia datasets for each of the data pool

```
train_list = [x for x in train_normal]
train_list.extend([x for x in train_pneumonia])

df_train = pd.DataFrame(np.concatenate([['Normal']*len(train_normal) , ['Pneumonia']*len(train_pneumonia)]), columns = ['class'])
df_train['image'] = [x for x in train_list]



test_list = [x for x in test_normal]
test_list.extend([x for x in test_pneumonia])

df_test = pd.DataFrame(np.concatenate([['Normal']*len(test_normal) , ['Pneumonia']*len(test_pneumonia)]), columns = ['class'])
df_test['image'] = [x for x in test_list]


val_list = [x for x in val_normal]
val_list.extend([x for x in val_pneumonia])

df_val = pd.DataFrame(np.concatenate([['Normal']*len(val_normal) , ['Pneumonia']*len(val_pneumonia)]), columns = ['class'])
df_val['image'] = [x for x in val_list]
```

Now that we have our dataset combined. We will explore the dimension of the data frames:

Training data set:

```
[6]: df_train
```

[6]:

	class	image
0	Normal	chest_xray_images/chest_xray/train/NORMAL/NORM...
1	Normal	chest_xray_images/chest_xray/train/NORMAL/NORM...
2	Normal	chest_xray_images/chest_xray/train/NORMAL/IM-0...
3	Normal	chest_xray_images/chest_xray/train/NORMAL/NORM...
4	Normal	chest_xray_images/chest_xray/train/NORMAL/IM-0...
...	...	...
5211	Pneumonia	chest_xray_images/chest_xray/train/PNEUMONIA/p...
5212	Pneumonia	chest_xray_images/chest_xray/train/PNEUMONIA/p...
5213	Pneumonia	chest_xray_images/chest_xray/train/PNEUMONIA/p...
5214	Pneumonia	chest_xray_images/chest_xray/train/PNEUMONIA/p...
5215	Pneumonia	chest_xray_images/chest_xray/train/PNEUMONIA/p...

5216 rows × 2 columns

Looks like the training data set has 5216 rows and 2 columns. One column is the label and the other column is the path to the image.

Testing data set:

```
[14]: df_test
```

[14]:

	class	image
0	Normal	chest_xray_images/chest_xray/test/NORMAL/IM-00...
1	Normal	chest_xray_images/chest_xray/test/NORMAL/IM-00...
2	Normal	chest_xray_images/chest_xray/test/NORMAL/NORMA...
3	Normal	chest_xray_images/chest_xray/test/NORMAL/NORMA...
4	Normal	chest_xray_images/chest_xray/test/NORMAL/NORMA...
...	...	...
619	Pneumonia	chest_xray_images/chest_xray/test/PNEUMONIA/pe...
620	Pneumonia	chest_xray_images/chest_xray/test/PNEUMONIA/pe...
621	Pneumonia	chest_xray_images/chest_xray/test/PNEUMONIA/pe...
622	Pneumonia	chest_xray_images/chest_xray/test/PNEUMONIA/pe...
623	Pneumonia	chest_xray_images/chest_xray/test/PNEUMONIA/pe...

624 rows × 2 columns

Looks like the testing data set has 624 rows.

Val data set:

```
[15]: df_val
```

[15]:

	class	image
0	Normal	chest_xray_images/chest_xray/val/NORMAL/NORMAL...
1	Normal	chest_xray_images/chest_xray/val/NORMAL/NORMAL...
2	Normal	chest_xray_images/chest_xray/val/NORMAL/NORMAL...
3	Normal	chest_xray_images/chest_xray/val/NORMAL/NORMAL...
4	Normal	chest_xray_images/chest_xray/val/NORMAL/NORMAL...
5	Normal	chest_xray_images/chest_xray/val/NORMAL/NORMAL...
6	Normal	chest_xray_images/chest_xray/val/NORMAL/NORMAL...
7	Normal	chest_xray_images/chest_xray/val/NORMAL/NORMAL...
8	Pneumonia	chest_xray_images/chest_xray/val/PNEUMONIA/per...
9	Pneumonia	chest_xray_images/chest_xray/val/PNEUMONIA/per...
10	Pneumonia	chest_xray_images/chest_xray/val/PNEUMONIA/per...
11	Pneumonia	chest_xray_images/chest_xray/val/PNEUMONIA/per...
12	Pneumonia	chest_xray_images/chest_xray/val/PNEUMONIA/per...
13	Pneumonia	chest_xray_images/chest_xray/val/PNEUMONIA/per...
14	Pneumonia	chest_xray_images/chest_xray/val/PNEUMONIA/per...
15	Pneumonia	chest_xray_images/chest_xray/val/PNEUMONIA/per...

The Val data set has only 15 rows of data. Among the 15 rows, there 7 rows with normal x-ray image and 8 rows with x-ray image of the patient who had pneumonia.

Let's explore more with some visualization:

From the bar charts above, it appears that the number of pneumonia cases are higher than normal cases in both the training and testing data set.

Using Python, we are now going to create a function which will essentially read the data from the given path and plot the image.

```
]: def plotFirstThreeImage(dataSet):
       plt.figure(figsize=(12,12))
       for i in range (0, 3):
           plt.subplot(3, 4, i+1)
           img = cv2.imread(dataSet[i])
           img = cv2.resize(img, (244, 244))
           plt.imshow(img)
           plt.axis('off')
       plt.tight_layout()
       plt.show
```

Now that we have the function ready, let's plot first three images of the train_normal data set:

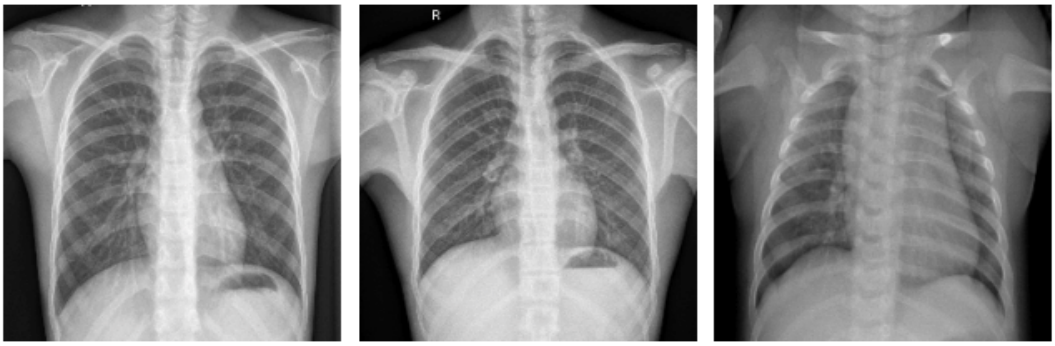
```
[13]: plotFirstThreeImage(train_normal)
```

Plot first three images from train_pneumonia data set:

```
[14]: plotFirstThreeImage(train_pneumonia)
```

Plot first three images from test_normal data set:

```
[58]: plotFirstThreeImage(test_normal)
```



Plot first three images from test_pneumonia data set:

```
[59]: plotFirstThreeImage(test_pneumonia)
```

Appendix B. Data augmentation

Keras ImageDataGenerator is used for getting the input of the original data and further, it makes the transformation of this data on a random basis and gives the output resultant containing only the data that is newly transformed. It does not add the data. Keras image data generator class is also used to carry out data augmentation where we aim to gain the overall increment in the generalization of the model. Operations such as rotations, translations, shearin, scale changes, and horizontal flips are carried out randomly in data augmentation using an image data generator. Let's take a look how the ImageDataGenerator code looks like:

[24]:
```
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_dataframe(
    df_train,
    x_col = 'image',
    y_col = 'class',
    target_size=(64, 64),
    batch_size=32,
    class_mode='binary')

test_generator = test_datagen.flow_from_dataframe(
    df_test,
    x_col = 'image',
    y_col = 'class',
    target_size=(64, 64),
    batch_size=32,
    class_mode='binary',
    shuffle = False)

val_generator = test_datagen.flow_from_dataframe(
    df_val,
    x_col = 'image',
    y_col = 'class',
    target_size=(64, 64),
    batch_size=16,
    class_mode='binary')
```

```
Found 5216 validated image filenames belonging to 2 classes.
Found 624 validated image filenames belonging to 2 classes.
Found 16 validated image filenames belonging to 2 classes.
```

Appendix C. Code for SVM model and model performance

The SVM model started with reading the data and convert the data into NumPy array. The code looks like below:

```
def create_data(data):
    data_set = data['image']
    mydata = []
    for i in range (0, data_set.size):
        img = cv2.imread(data_set[i], 0)
        img_resized = cv2.resize(img, (50,50))
        image = np.array(img_resized).flatten()

        mydata.append(image)
    return mydata
```

```
xtrain = create_data(df_train)
```

```
len(xtrain)
```

```
5216
```

Code for SVM Classifier

```
[40]: from sklearn import svm
      svmClassifier = svm.SVC()
      svmClassifier.fit(xtrain, ytrain)
```

```
[40]: ▾ SVC
      SVC()
```

Code for prediction and accuracy of SVM code:

```
[44]: prediction = svmClassifier.predict(xtest)
```

```
[45]: accuracy = svmClassifier.score(xtest, ytest)
```

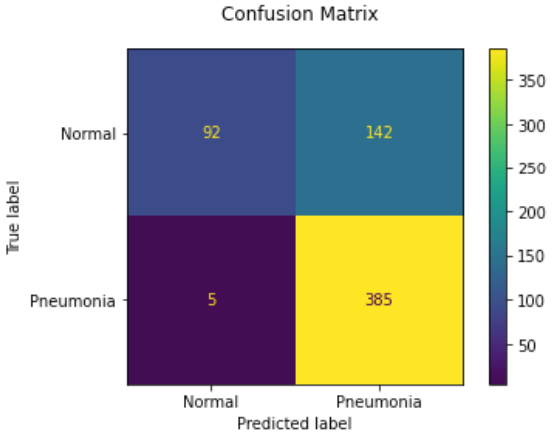
```
[46]: accuracy
```

```
[46]: 0.7644230769230769
```

Confusion matrix for the SVM Model:

```
[47]: disp = metrics.ConfusionMatrixDisplay.from_predictions(ytest, prediction)
disp.figure_.suptitle("Confusion Matrix")
print(f"Confusion matrix:\n{disp.confusion_matrix}")
plt.show()
```

```
Confusion matrix:
[[ 92 142]
 [  5 385]]
```



Report for SVM Classifier:

```
[51]: print(
    f"Classification report for classifier {svmClassifier}:\n"
    f"{metrics.classification_report(ytest, prediction)}\n"
)
```

```
Classification report for classifier SVC():
              precision    recall  f1-score   support

      Normal       0.95      0.39      0.56       234
   Pneumonia       0.73      0.99      0.84       390

    accuracy                           0.76       624
   macro avg       0.84      0.69      0.70       624
weighted avg       0.81      0.76      0.73       624
```

Appendix D. Code for CNN model and model performance

The CNN model we used is the Sequential model. A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor.

Our Model looks like this code:

```
# let's build the CNN model

cnn = Sequential()

#Convolution
cnn.add(Conv2D(32, (3, 3), activation="relu", input_shape=(64, 64, 3)))

#Pooling
cnn.add(MaxPooling2D(pool_size = (2, 2)))

# 2nd Convolution
cnn.add(Conv2D(64, (3, 3), activation="relu"))

# 2nd Pooling layer
cnn.add(MaxPooling2D(pool_size = (2, 2)))

# Flatten the layer

cnn.add(layers.Flatten())
cnn.add(layers.Dense(64, activation='relu'))
cnn.add(layers.Dense(10))

# Compile the Neural network
cnn.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

When we compile the model and print the summary, the summary prints pertinent information of the model:

```
[37]: cnn.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)           (None, 62, 62, 32)        896

 max_pooling2d_4 (MaxPooling  (None, 31, 31, 32)       0
 2D)

 conv2d_6 (Conv2D)           (None, 29, 29, 64)        18496

 max_pooling2d_5 (MaxPooling  (None, 14, 14, 64)       0
 2D)

 flatten_2 (Flatten)         (None, 12544)             0

 dense_4 (Dense)             (None, 64)                802880

 dense_5 (Dense)             (None, 10)                650

=================================================================
Total params: 822,922
Trainable params: 822,922
Non-trainable params: 0
_________________________________________________________________
```

Now that we have a model and our data is ready to be provided as input, let's try to train the model.

```
[53]: cnn_model = cnn.fit_generator(train_generator,
                                    steps_per_epoch = 163,
                                    epochs = 25,
                                    validation_data = val_generator)
```

```
Epoch 1/25
163/163 [==============================] - 68s 415ms/step - loss: 0.1463 - accuracy: 0.9410 - val_loss: 0.4515 - val_accuracy: 0.6875
Epoch 2/25
163/163 [==============================] - 68s 418ms/step - loss: 0.1409 - accuracy: 0.9471 - val_loss: 0.2804 - val_accuracy: 0.8750
Epoch 3/25
163/163 [==============================] - 87s 533ms/step - loss: 0.1444 - accuracy: 0.9450 - val_loss: 0.2762 - val_accuracy: 0.8750
Epoch 4/25
163/163 [==============================] - 76s 465ms/step - loss: 0.1309 - accuracy: 0.9509 - val_loss: 0.4118 - val_accuracy: 0.7500
Epoch 5/25
163/163 [==============================] - 96s 592ms/step - loss: 0.1207 - accuracy: 0.9530 - val_loss: 0.4904 - val_accuracy: 0.6875
```

After 25th iteration, the accuracy seems to be consistent around 96.70% and the accuracy for validation data is around 93.75%

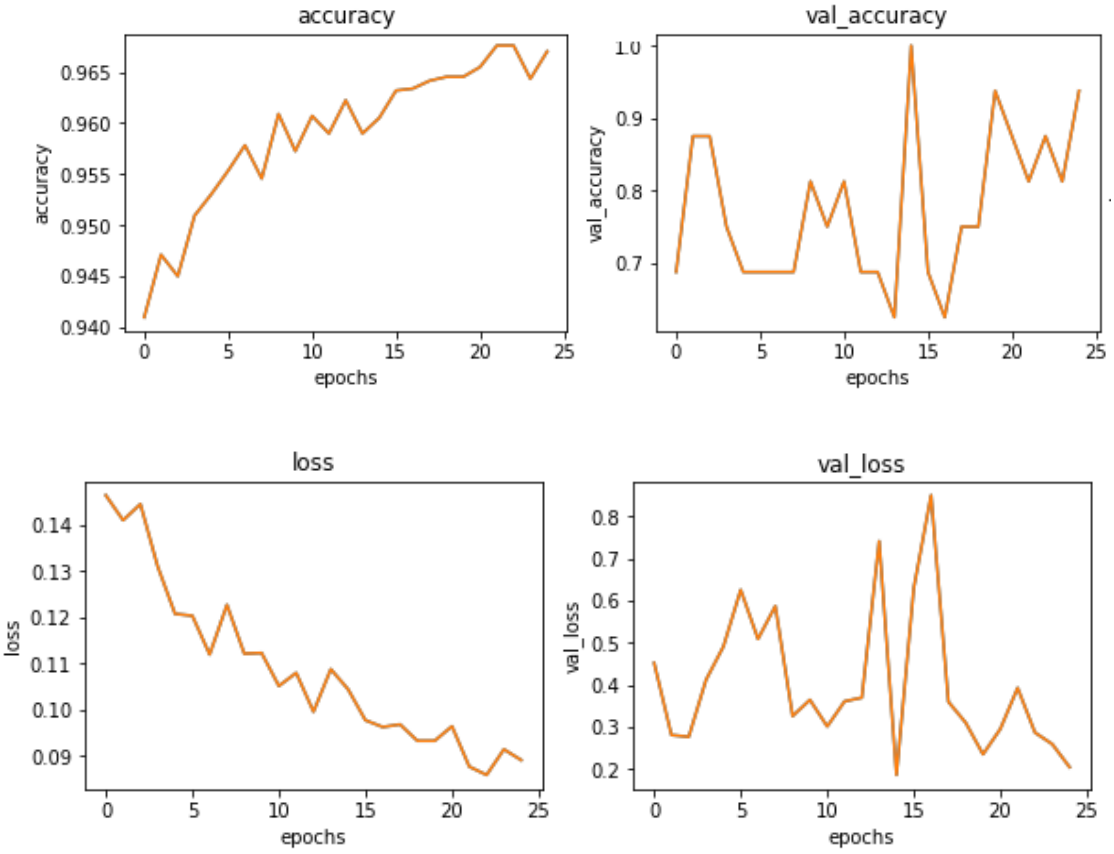
```
Epoch 22/25
163/163 [==============================] - 73s 449ms/step - loss: 0.0876 - accuracy: 0.9676 - val_loss: 0.3924 - val_accuracy: 0.8125
Epoch 23/25
163/163 [==============================] - 129s 793ms/step - loss: 0.0858 - accuracy: 0.9676 - val_loss: 0.2863 - val_accuracy: 0.8750
Epoch 24/25
163/163 [==============================] - 119s 729ms/step - loss: 0.0913 - accuracy: 0.9643 - val_loss: 0.2588 - val_accuracy: 0.8125
Epoch 25/25
163/163 [==============================] - 122s 750ms/step - loss: 0.0890 - accuracy: 0.9670 - val_loss: 0.2041 - val_accuracy: 0.9375
```

Now that we have a model which is trained, let's visualize the model performance

```
fig, ax = plt.subplots(1, 4, figsize=(20, 3))
ax = ax.ravel()

for i, x in enumerate(['accuracy', 'val_accuracy', 'loss', 'val_loss']):
    ax[i].plot(cnn_model.history[x])
    ax[i].plot(cnn_model.history[x])
    ax[i].set_title(x)
    ax[i].set_xlabel('epochs')
    ax[i].set_ylabel(x)
```

The plots look like this:



Let's evaluate our model against our test data:

```
[56]: test_accuracy = cnn.evaluate_generator(test_generator)
      test_accuracy

[56]: [0.2157781571149826, 0.9262820482254028]

[81]: print('The testing accuracy is :',test_accuracy[1]*100, '%')

      The testing accuracy is : 92.62820482254028 %
```