Pneumonia Detection in Chest X-Ray Images - CNN using Keras Tensorflow

December 26, 2022

1 Introduction

Deep learning for the medical image classification is not only a topic of hot research but is a key technique of computer-aided diagnosis systems today.

- According to WHO, every year over 150 million people are infected with pneumonia particularly kids below the age of 5 years. One in three deaths in India are caused due to pneumonia as reported by the World Health Organization (WHO).
- Chest X-rays are at the moment, the best available method for diagnosing pneumonia, and therefore play a crucial role in diagnosing and providing clinical care to the ones affected.
- However, detecting pneumonia in chest X-rays is a challenging task that relies on the availability of expert radiologists. Experts are either not available in remote areas or most people can't afford it. Under such circumstances, automating the detection of diseases through AI becomes the need of the hour.
- This study will result into aiding healthcare practioners, physicians, doctors, hospitals to take quick actions if the chest Xray detects Pneumonia. (Based on recent studies it's been observed that pneumonia patients are more prone to have COVID symptoms and their ill-effects.)
- We'll build an end-to-end machine learning & AI pipeline that uses X-ray images of the lungs to detect pneumonia in patients.

2 Load Libraries

[1]:
```
! pip install opencv-python
```

```
WARNING: Value for scheme.headers does not match. Please report this to
<https://github.com/pypa/pip/issues/9617>
distutils: /opt/conda/include/python3.8/UNKNOWN
sysconfig: /opt/conda/include/python3.8
```

```
WARNING: Additional context:

user = False

home = None

root = None

prefix = None
Requirement already satisfied: opencv-python in /opt/conda/lib/python3.8/site-
packages (4.6.0.66)
Requirement already satisfied: numpy>=1.14.5 in /opt/conda/lib/python3.8/site-
packages (from opencv-python) (1.19.5)
WARNING: Value for scheme.headers does not match. Please report this to

<https://github.com/pypa/pip/issues/9617>

distutils: /opt/conda/include/python3.8/UNKNOWN

sysconfig: /opt/conda/include/python3.8
WARNING: Additional context:

user = False

home = None

root = None

prefix = None
```

We have imported OpenCV for preprocessing and loading. For the image classification modeling part, we'll be using Keras with Tensorflow as a backend.

[2]:
```python
import glob
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pathlib import Path
from PIL import Image
from keras.models import Sequential, Model, load_model
from keras.applications.vgg16 import VGG16, preprocess_input
from keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
from keras.layers import *
from keras.optimizers import *
from keras.callbacks import Callback, EarlyStopping
from keras.utils import to_categorical
from sklearn.metrics import confusion_matrix
from keras import backend as k
import tensorflow

%matplotlib inline
```

3 Set source data path to load data

We will split the dataset into three sets - train, validation, and test. Let's define the paths where our data is stored.

There are three separate directories for train, validation, and test data. In each of these directories, there are two folders- one containing pneumonia x-ray images and the other containing normal x-ray images.

[3]:
```
data_dir = Path("../chest_xray/chest_xray")
```

[4]:
```
data_dir
```

[4]:
```
PosixPath('../chest_xray/chest_xray')
```

[5]:
```
train_dir = data_dir/'train'
test_dir = data_dir/'test'
val_dir = data_dir/'val'
```

[6]:
```
train_dir
```

[6]:
```
PosixPath('../chest_xray/chest_xray/train')
```

Let's load the training data in a data frame, where one column would contain the path to images and the other would have image labels.

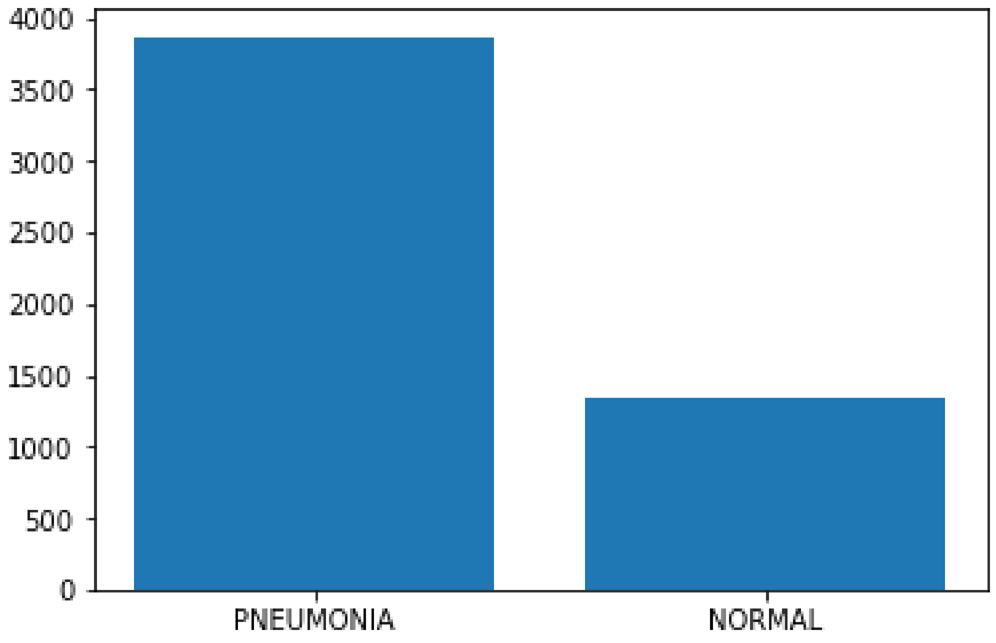
[7]:
```
def train_load():
    normal_case_dir = train_dir/'NORMAL'
    pneumonia_case_dir = train_dir /'PNEUMONIA'

    # Get the list of all the images in train dataset
    normal_cases = normal_case_dir.glob('*.jpeg')
    pneumonia_cases = pneumonia_case_dir.glob('*.jpeg')

    data_train = []
    train_label = []

    for img in normal_cases:
        data_train.append(img)
        train_label.append('NORMAL')

    for img in pneumonia_cases:
        data_train.append(img)
        train_label.append('PNEUMONIA')

    # let's create pandas dataframe
    df_train = pd.DataFrame(data_train)
    #print(df_train.head())
    df_train.columns = ['images']
    df_train['labels'] = train_label
    df_train = df_train.sample(frac = 1).reset_index(drop = True)

    return df_train
```

[8]:
```
train = train_load()
train.shape
```

[8]:
```
(5210, 2)
```

[9]:
```
plt.bar(train['labels'].value_counts().index, train['labels'].value_counts().
 ↪values)
```

[9]:
```
<BarContainer object of 2 artists>
```



As we can see from the above visualization that data for patients with pneomonia chest xrays are 3 times more than patients with normal chest xrays. This data imbalance could be a problematic to accurately train and validate the model.

[10]:
```
def plot_img(image_batch, label_batch):
    plt.figure(figsize = (10,5))
    for i in range(10):
        ax = plt.subplot(2,5,i+1)
        img = cv2.imread(str(image_batch[i]))
        img = cv2.resize(img, (224,224))
        plt.imshow(img)
        plt.title(label_batch[i])
        plt.axis("off")
```

[11]:
```
train.columns
```

[11]:
```
Index(['images', 'labels'], dtype='object')
```

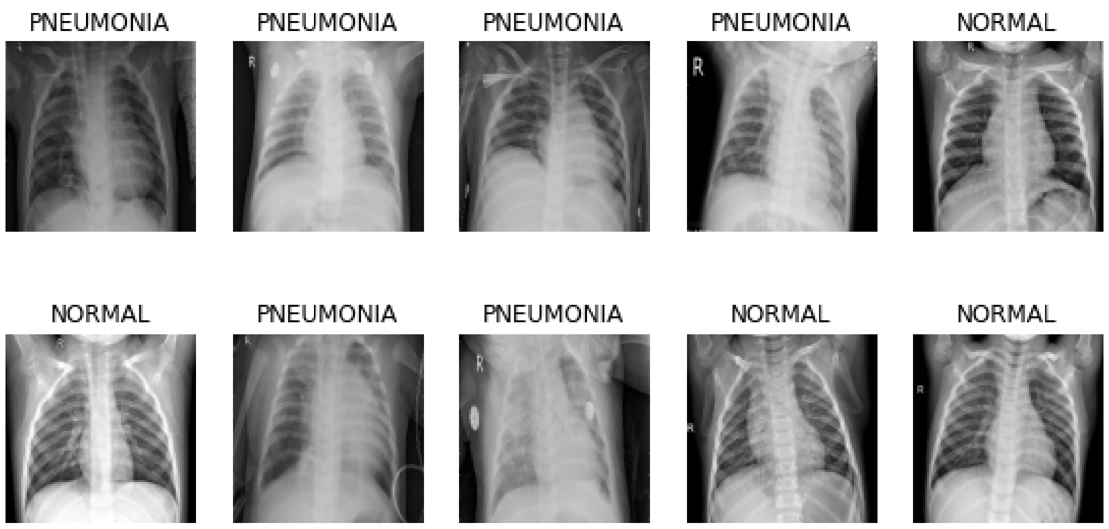
[12]:
```
train.head()
```

[12]:
```
                                              images     labels
0  ../chest_xray/chest_xray/train/NORMAL/NORMAL2-…     NORMAL
1  ../chest_xray/chest_xray/train/PNEUMONIA/perso…  PNEUMONIA
2  ../chest_xray/chest_xray/train/PNEUMONIA/perso…  PNEUMONIA
3  ../chest_xray/chest_xray/train/NORMAL/NORMAL2-…     NORMAL
4  ../chest_xray/chest_xray/train/PNEUMONIA/perso…  PNEUMONIA
```

[13]:
```
shuffled = train.sample(frac = 1, random_state = 42)
shuffled = shuffled.reset_index()
shuffled
```

[13]:
```
      index                                             images     labels
0      2746  ../chest_xray/chest_xray/train/PNEUMONIA/perso…  PNEUMONIA
1       877  ../chest_xray/chest_xray/train/PNEUMONIA/perso…  PNEUMONIA
2      1215  ../chest_xray/chest_xray/train/PNEUMONIA/perso…  PNEUMONIA
3      2154  ../chest_xray/chest_xray/train/PNEUMONIA/perso…  PNEUMONIA
4      3694  ../chest_xray/chest_xray/train/NORMAL/IM-0695-…     NORMAL
…       …                                                …        …
5205    466  ../chest_xray/chest_xray/train/PNEUMONIA/perso…  PNEUMONIA
5206   3092  ../chest_xray/chest_xray/train/NORMAL/IM-0559-…     NORMAL
5207   3772  ../chest_xray/chest_xray/train/PNEUMONIA/perso…  PNEUMONIA
5208   5191  ../chest_xray/chest_xray/train/PNEUMONIA/perso…  PNEUMONIA
5209    860  ../chest_xray/chest_xray/train/PNEUMONIA/perso…  PNEUMONIA

[5210 rows x 3 columns]
```

[14]:
```
image_batch = shuffled['images'][:10] #.tolist()
label_batch = shuffled['labels'][:10] #.tolist()
plot_img(image_batch, label_batch)
```



To the untrained eye, the x-rays with or without pneumonia look the same.

4 Image Preprocessing along with Validation and Test data load

Preprocessing is essential to transform images in a format that can be easily understood by the model and also to make the algorithm work more efficiently.

The different preprocessing steps that we'll use here are:

- Since the images are of different lengths and widths, resize them to 224,224,3.
- Some images are in greyscale (1 channel), therefore convert them to 3 channel
- Images read using cv2 are in BGR format(by default), convert it to RGB.
- Normalize the image pixels by dividing them by 255 (an essential math trick for better performance).
- to_categorical is used to convert labels to one-hot encoded format.

[15]:
```
def preprocess_and_load(isval):
    if isval == True:
        normal_case_dir = val_dir/'NORMAL'
        pneumonia_case_dir = val_dir /'PNEUMONIA'
    else:
        normal_case_dir = test_dir/'NORMAL'
        pneumonia_case_dir = test_dir /'PNEUMONIA'

    # based on the above if else condition load all the validation or test data
    normal_cases = normal_case_dir.glob('*.jpeg')
    pneumonia_cases = pneumonia_case_dir.glob('*.jpeg')

    # empty lists to store the data
#     data = []
#     labels = []
    data,labels = ([] for x in range(2))

    def preprocess(cases):
        for img in cases:
            img = cv2.imread(str(img))
            img = cv2.resize(img, (224,224))
            if img.shape[2] == 1:
                img = np.dstack([img,img,img])
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            img = img.astype(np.float32)/255.

            if cases == normal_cases:
                label = to_categorical(0, num_classes = 2)
            else:
                label = to_categorical(1, num_classes = 2)

            data.append(img)
            labels.append(label)
#             print(data[:3])
#             print(labels[:3])
        return(data,labels)
    #dn,ln =
    preprocess(normal_cases)
#     dn1 = np.asarray(dn)
#     ln1 = np.asarray(ln)

    dp, lp = preprocess(pneumonia_cases)
#     dp1 = np.asarray(dp)
#     lp1 = np.asarray(lp)

#     d = np.append(dn1, dp1)
#     l = np.append(ln1, lp1)
    dp = np.asarray(dp)
    lp = np.asarray(lp)

    return dp,lp
```

[16]:
```
val_data, val_labels = preprocess_and_load(isval = True)
```

[17]:
```
test_data, test_labels = preprocess_and_load(isval = False)
```

[18]:
```
print(f"No. of Validation Images: {len(val_data)}")
print(f"No. of Test Images: {len(test_data)}")
```

```
No. of Validation Images: 16
No. of Test Images: 622
```

We have 16 images for validation and 622 images for testing.

In Neural Networks, training takes place in batches.

```
- The model takes the first batch, passes it through the network,
- A loss is calculated in the end, then the gradients travel backward to update the parameters
- This process is repeated until we reach our desired loss and training stops.
```

Let’s write a function to generate images in batches from the train set.

5 Train data preprocessing

[19]:
```
def train_data_gen(data, batch_size):

    #get tot samples in the data
    n = len(data)
    steps = n//batch_size

    #define two numpy arrays for batch data and batch labels
    batch_data = np.zeros((batch_size, 224, 224, 3), dtype = np.float32)
    batch_labels = np.zeros((batch_size, 2), dtype = np.float32)

    # a numpy array for all the indices of the input data
    indices = np.arange(n)

    # initialize the counter
    i = 0

    while True:
        np.random.shuffle(indices)
        # get next batch
        count = 0
        next_batch = indices[(i*batch_size):(i+1)*batch_size]
        for j, idx in enumerate(next_batch):
            img_name = data.iloc[idx]['images']
            label = data.iloc[idx]['labels']
            if label == 'NORMAL':
                label = 0
            else:
                label = 1

            # one hot encoding
            encoded_label = to_categorical(label, num_classes = 2)

            # read the images & resize
            img = cv2.imread(str(img_name))
            img = cv2.resize(img,(224,224))

            # for grayscale images
            if img.shape[2] == 1:
                img = np.dstack([img,img,img])

            # since cv2 reads in BGR mode by default lets convert into RGB
            orig_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

            # scale the images for pixel range 0 to 255 i.e normalize them
            orig_img = orig_img.astype(np.float32)/255.

            batch_data[count] = orig_img
            batch_labels[count] = encoded_label

            count+=1

            if count == batch_size - 1:
                break

        i+=1
        yield batch_data, batch_labels

        if i>=steps:
            i=0
```

Batch size refers to the number of images fed to the network in a single forward pass. On the other hand, the number of epochs determines the number of times our network would process the whole training data. The number of training steps is the number of steps in one epoch and is calculated by dividing the size of our training data by the batch size.

[20]:
```
batch_size = 16
nb_epochs = 3

# preprocess the train data
train_data_gen = train_data_gen(data = train, batch_size = batch_size)

# Define the number of training steps
nb_train_steps = train.shape[0]//batch_size
```

6 CNN Model

- The basic building block of any model working on image data is a Convolutional Neural Network.
- Convolutions were designed specifically for images.
- There is a filter or weights matrix (n x n-dimensional) where n is usually smaller than the image size.
- A multiplication or dot product is taken of this matrix with the filter size patch of the input. The filter is applied systematically to each overlapping part or filter-sized patch of the input data, moving from left to right and then top to bottom.
- The result of this dot product between two matrices is a single value and through repetition of this process on different input patches, we get a matrix in the end.
- There is also a bias value that is added after every dot product.
- The weight matrix and the bias value are the parameters of the neural network that are updated throughout training.
- Stride is the value by which the filter shifts on the image.
- In a convolutional layer, there are multiple filters- this value is decided and fed by the developer when defining a layer. The use and significance of these convolutions might not be intuitive at first- it is hypothesized that they learn different things at different stages.
- The convolutions in the earlier layers learn to detect abstract things like edges, textures, etc. Towards the final layer, they learn to detect more specific objects based on classification categories.

[21]:
```
model = Sequential()
model.add(Conv2D(32, (3,3), input_shape = (224,224,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))

model.add(Conv2D(32, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))

model.add(Conv2D(64, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))

model.add(Flatten())

model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dense(2))
model.add(Activation('softmax'))
```

A ReLu activation is applied after every convolution to transform the output values between the range 0 to 1. Max pooling is used to downsample the input representation. It helps the model to deal with overfitting by providing an abstract representation and also reduces the computational cost. The way Max Pooling works can be illustrated by the image below:

We'll use binary cross-entropy as our loss function because we have only 2 classes. Rmsprop will be our optimizer function.

[22]:
```
model.compile(optimizer = 'rmsprop', loss = 'binary_crossentropy', metrics =␣
 ↪['accuracy'])
model.fit(train_data_gen, epochs = nb_epochs, steps_per_epoch = nb_train_steps,␣
 ↪validation_data = (val_data, val_labels))
```

```
Epoch 1/3
325/325 [==============================] - 101s 281ms/step - loss: 0.5174 -
accuracy: 0.7811 - val_loss: 0.3152 - val_accuracy: 0.8750
Epoch 2/3
325/325 [==============================] - 86s 267ms/step - loss: 0.1257 -
accuracy: 0.9044 - val_loss: 0.1361 - val_accuracy: 0.9375
Epoch 3/3
325/325 [==============================] - 88s 271ms/step - loss: 0.1171 -
accuracy: 0.9022 - val_loss: 0.3549 - val_accuracy: 0.8125
```

[22]:
```
<tensorflow.python.keras.callbacks.History at 0x7f90422d1940>
```

In oder to increase the validation accuracy and performance metrics overall we can take following steps -

- Change the batch size.
- Train for more epochs.
- Use a different optimizer.
- Tweak the neural network by adding/ removing layers.

7 Prediction and Model Evaluation Parameters

[32]:
```
prediction = model.predict(test_data, batch_size = 16)
pred_labels = np.argmax(prediction, axis = -1)

test_labels1 = np.argmax(test_labels, axis = -1)


from sklearn.metrics import classification_report

print(classification_report(test_labels1, pred_labels))
```

```
              precision    recall  f1-score   support

           0       0.97      0.32      0.48       234
           1       0.71      0.99      0.83       388

    accuracy                           0.74       622
   macro avg       0.84      0.66      0.65       622
```

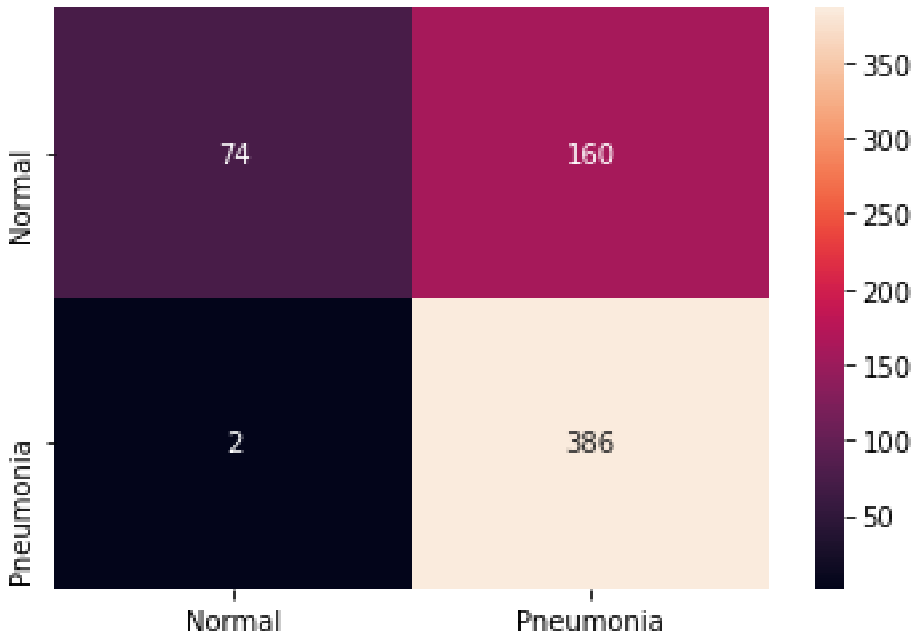
```
weighted avg       0.81      0.74      0.70       622
```

- Precision is the fraction of relevant instances among the retrieved instances. In our case, it is the number of people actually having pneumonia divided by all those predicted by the model as having pneumonia.
- Recall on the other hand refers to the relevant instances that were retrieved. Here, it is the fraction of people actually having pneumonia and are predicted positive by the model to the total number of people having pneumonia. It measures the potential of a test to recognize patients with the disease.
- F1 score is just the harmonic mean of precision and recall.

[33]:
```
cm = confusion_matrix(labels, pred)
```

[35]:
```
import seaborn as sns
sns.heatmap(cm, annot = True, fmt = 'g', xticklabels = ['Normal', 'Pneumonia'],␣
 ↪yticklabels = ['Normal', 'Pneumonia'])
```

[35]:
```
<AxesSubplot:>
```



8 Results & Conclusion

The y-axis of the chart is for true labels and the x-axis is for predicted ones.

- The number of people who are actually Normal and are predicted as Normal by our model is 74. These cases are called True Negatives.
- The number of people with Pneumonia but diagnosed as Normal are called False Negatives and there are just 2 patients for that.
- The number of people who were Normal but are diagnosed with Pneumonia by the model are called False Positives and these cases are 160.
- The number of people with Pneumonia who are also diagnosed with Pneumonia by the model are True Positives, these cases are 386.

While training an ML algorithm to diagnose whether a patient has a disease or not, it is far more fatal to predict "Normal"

for a person who actually has the ailment when compared to the other type of error i.e. predicting Pneumonia for Normal

Patients. Thus, while training our aim should be to minimize False Negatives and we have successfully done that.