SMART INTERNZ – INTERNSHIP

Artificial Intelligence

Project

Title:

Pneumonia Prediction using X-RAY Images

Vishal Parekh

Abstract

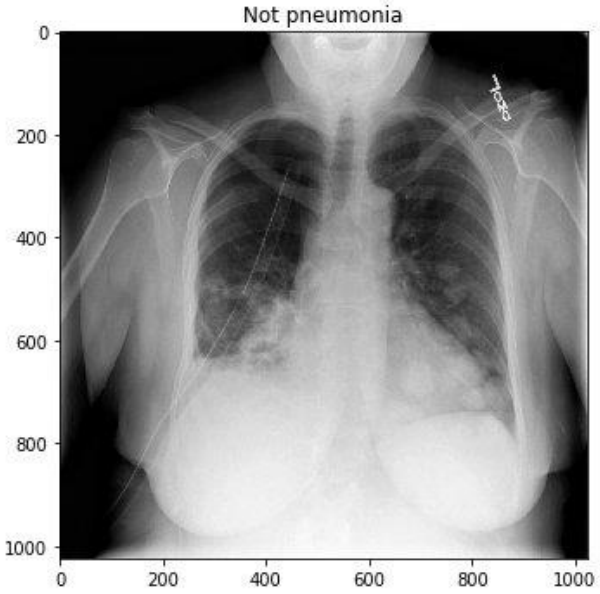
Pneumonia is the leading cause of death among young children and one of the top mortalities causes worldwide. The pneumonia detection is usually performed through examine of chest X-ray radiograph by highly-trained specialists. This process is tedious and often leads to a disagreement between radiologists. Computer-aided diagnosis systems showed the potential for improving diagnostic accuracy. In this work, we develop the computational approach for pneumonia regions detection based on deep convolution neural net-works, augmentations and multi-task learning.

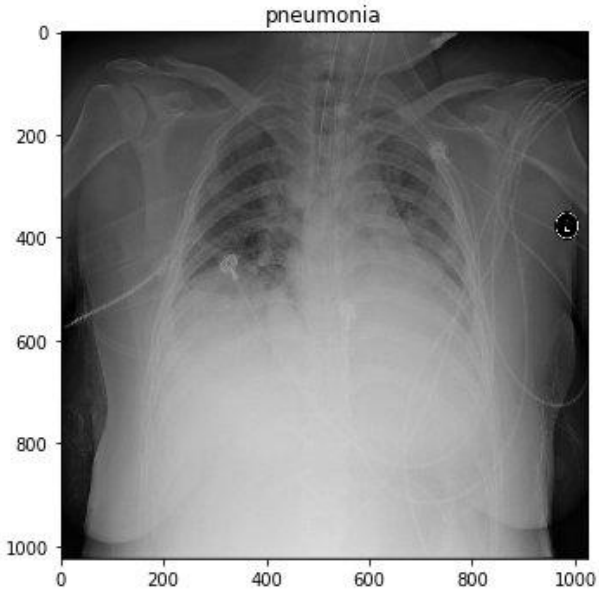
Introduction

Pneumonia accounts for around 16% of all deaths of children under five years worldwide [4], being the world's leading cause of death among young children [1]. In the United States only, about1million adults seek care in a hospital due to pneumonia every year, and 50,000 die from this disease [1]. The pneumonia complicating recent coronavirus disease 2019 (COVID-19) is a life-threatening condition claiming thousands of lives in 2020 [10, 12, 6]. Pneumonia caused by COVID-19 is of huge global concern [6]. The pneumonia detection is commonly performed through examine of chest X-ray radiograph (CXR) by highly-trained specialists. It usually manifests as an area or areas of increased opacity on CXR [11], the diagnosis is further confirmed through clinical history, vital signs and laboratory exams. The diagnosis of pneumonia on CXR is complicated due to the presence of other conditions in the lungs, such as fluid overload, bleeding, volume loss, lung cancer, post-radiation or surgical changes. When available, comparison of CXRs of the patient taken at different timepoints and correlation with clinical symptoms and history is helpful in making the diagnosis. A number of factors such as positioning of the patient and depth of inspiration can alter the appearance of the CXR [18], complicating interpretation even further. There is a known variability between radiologists in the interpretation of chest radiographs [20]. To improve the efficiency and accuracy of diagnostic services computer-aided diagnosis systems for pneumonia detection has been widely exploited in the last decade [22, 21, 28, 35, 25]. Deep learning approaches outperformed conventional machine learning methods in many medical image analysis tasks, including detection [25], classification [26] and segmentation [27, 17]. Here, I present the solution of Pneumonia Detection Challenge for Pneumonia regions detection hosted on SmartInternz platform [3]. Our approach uses deep convolution neural networks (CNNs) [16], augmentations and multi-task learning. The algorithm automatically locates lung opacities on chest radiographs and demonstrated one of the best performances in the challenge.

The Challenge

Build an algorithm to automatically identify whether a patient is suffering from pneumonia or not by looking at chest X-ray images. The algorithm had to be extremely accurate because lives of people is at stake.





Environment and Tools

1. Jupyter Lab – Python
2. Keras
3. Skimage
4. NumPy
5. Cv2
6. Flask

Data

The dataset can be downloaded from the Kaggle website which can be found here.

The dataset includes 3 folders namely train, test, validation.
The train folder contains around 5000+ x-ray images which will be used to train our convolutional neural network model.
The test folder contains around 600+ x-ray images which will be used to test our model simultaneously.
The validation folder contains samples images for which our model will predict whether the x-ray image belongs to the pneumonia category or the normal category.

Code

(1) Train

We create a. ipynb file named as train. ipynb.
Let's start with loading all the libraries and dependencies.

```
[1]: import keras
     from keras.preprocessing.image import ImageDataGenerator
```

```
[4]: import keras
     from keras.models import Sequential
     from keras.layers import Dense,Flatten
     from keras.layers import Conv2D, MaxPooling2D
```

The practice of data augmentation is an effective way to increase the size of the training set. Augmenting the training examples allow the network to "see" more diversified, but still representative, data points during training. Then I defined a couple of data generators: one for training data, and the other for validation data. A data generator is capable of loading the required amount of data (a mini batch of images) directly from the source folder, convert them into training data (fed to the model) and training targets (a vector of attributes — the supervision signal).

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

     test_datagen=ImageDataGenerator(rescale=1./255)
```

```
[3]: training_set=train_datagen.flow_from_directory(r'F:\Project\train',target_size=(64,64),batch_size=8,class_mode='binary')

     val_set=train_datagen.flow_from_directory(r'F:\Project\val',target_size=(64, 64),batch_size=8,shuffle=True,class_mode='binary')

     test_set=test_datagen.flow_from_directory(r'F:\Project\test',target_size=(64,64),batch_size=8,class_mode='binary')

     Found 5216 images belonging to 2 classes.
     Found 16 images belonging to 2 classes.
     Found 624 images belonging to 2 classes.
```

The next step was to build the model. This can be described in the following steps.

1. I used convolutional blocks comprised of convolutional layer, max-pooling and batch-normalization.
2. On top of it I used a flatten layer and followed it by other fully connected layers.
3. Also, I have used the dense layer which includes kernel_regularizer, bias_regularizer, activity_regularizer.
4. The activation function was ‘Relu’ in the first dense layer and in the second dense layer it was ‘Sigmoid’ as this is a binary classification problem.
5. I have used ‘Adam’ as the optimizer and binary_crossentropy as the loss.

[5]:
```
model=Sequential()
model.add(Conv2D(32,kernel_size=(3,3),activation='relu',input_shape=(64,64,3)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
```

[6]:
```
from keras import regularizers
model.add(Dense(128,activation='relu',kernel_regularizer=regularizers.l1_l2(l1=1e-5, l2=1e-4),bias_regularizer=regularizers.l2(1e-4),activity_regularizer=regularizers.l2(1e-5)))
model.add(Dense(1,activation='sigmoid',kernel_regularizer=regularizers.l1_l2(l1=1e-5, l2=1e-4),bias_regularizer=regularizers.l2(1e-4),activity_regularizer=regularizers.l2(1e-5)))
```

Next, I trained the convolutional neural network model for 15 epochs with a batch size of 8. Please note that usually a higher batch size gives better results but at the expense of higher computational burden. Some research also claims that there is an optimal batch size for best results which could be found by investing some time on hyper-parameter tuning. The steps_per_epochs were 326 with validation_steps as 39.

```
[7]: model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
     model.fit(training_set,steps_per_epoch=5216//16,epochs=15,validation_data=test_set,validation_steps=624//16)

     Epoch 1/15
     326/326 [==============================] - 94s 284ms/step - loss: 0.6701 - accuracy: 0.7700 - val_loss: 0.6317 - val_accuracy: 0.7147
     Epoch 2/15
     326/326 [==============================] - 62s 191ms/step - loss: 0.3851 - accuracy: 0.8657 - val_loss: 0.4201 - val_accuracy: 0.8494
     Epoch 3/15
     326/326 [==============================] - 54s 166ms/step - loss: 0.3138 - accuracy: 0.8990 - val_loss: 0.3932 - val_accuracy: 0.8846
     Epoch 4/15
     326/326 [==============================] - 53s 161ms/step - loss: 0.3292 - accuracy: 0.8837 - val_loss: 0.6497 - val_accuracy: 0.7500
     Epoch 5/15
     326/326 [==============================] - 52s 158ms/step - loss: 0.3115 - accuracy: 0.9072 - val_loss: 0.8669 - val_accuracy: 0.7372
     Epoch 6/15
     326/326 [==============================] - 48s 147ms/step - loss: 0.3039 - accuracy: 0.9024 - val_loss: 0.4914 - val_accuracy: 0.8141
     Epoch 7/15
     326/326 [==============================] - 48s 147ms/step - loss: 0.2594 - accuracy: 0.9263 - val_loss: 0.4915 - val_accuracy: 0.8269
     Epoch 8/15
     326/326 [==============================] - 48s 146ms/step - loss: 0.2935 - accuracy: 0.9119 - val_loss: 0.4193 - val_accuracy: 0.8814
     Epoch 9/15
     326/326 [==============================] - 47s 144ms/step - loss: 0.2968 - accuracy: 0.9101 - val_loss: 0.3398 - val_accuracy: 0.9103
     Epoch 10/15
     326/326 [==============================] - 46s 142ms/step - loss: 0.2745 - accuracy: 0.9170 - val_loss: 0.3920 - val_accuracy: 0.8654
     Epoch 11/15
     326/326 [==============================] - 47s 144ms/step - loss: 0.2769 - accuracy: 0.9156 - val_loss: 0.5625 - val_accuracy: 0.8365
     Epoch 12/15
     326/326 [==============================] - 47s 145ms/step - loss: 0.2609 - accuracy: 0.9184 - val_loss: 0.4280 - val_accuracy: 0.8718
     Epoch 13/15
     326/326 [==============================] - 48s 146ms/step - loss: 0.2670 - accuracy: 0.9217 - val_loss: 0.3956 - val_accuracy: 0.8910
     Epoch 14/15
     326/326 [==============================] - 47s 144ms/step - loss: 0.2539 - accuracy: 0.9195 - val_loss: 0.5649 - val_accuracy: 0.8333
     Epoch 15/15
     326/326 [==============================] - 47s 143ms/step - loss: 0.2672 - accuracy: 0.9278 - val_loss: 0.3668 - val_accuracy: 0.8782
[7]: <tensorflow.python.keras.callbacks.History at 0x1a317e4d220>
```

Thereafter the trained model is saved in the current working directory for later prediction use and the accuracy of the model is calculated which was found to be around 94%.

```
[8]: model.save('cnnmodel.h5')

[11]: scores = model.evaluate(training_set)
      print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))

      652/652 [==============================] - 59s 91ms/step - loss: 0.2199 - accuracy: 0.9440

      accuracy: 94.40%
```

(2) Predict Without Interface

After the model is saved in current directory, we create another. ipynb file named as predict. ipynb.
Let's start with loading all the libraries and dependencies.

```
[1]: from keras.models import load_model
     import numpy as np
     import cv2
```

Next, we load the trained and saved model from the current directory.

```
[2]: model=load_model('cnnmodel.h5')
```

```
[3]: model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

Now we define a list with the two categories of the classification namely Normal and Pneumonia.
We define a function which will take image as it's parameter and perform some digital image operations such as resize, dimension reduction etc.
Then we load the image given by the user and predict the class of image using our pre trained model and print out the class of category to the user.

```
[14]: from skimage.transform import resize
      CATEGORIES = ["Normal","Pneumonia"]
      def detect(frame):
              img=resize(frame,(64,64))
              img=np.expand_dims(img,axis=0)
              if(np.max(img)>1):
                  img=img/255.0
              prediction=model.predict(img)
              prediction_class=model.predict_classes(img)
              print(CATEGORIES[int(prediction_class[0][0])])
```

```
[15]: frame=cv2.imread("NORMAL2-IM-1427-0001.jpeg")
      data=detect(frame)
```

```
Normal
```

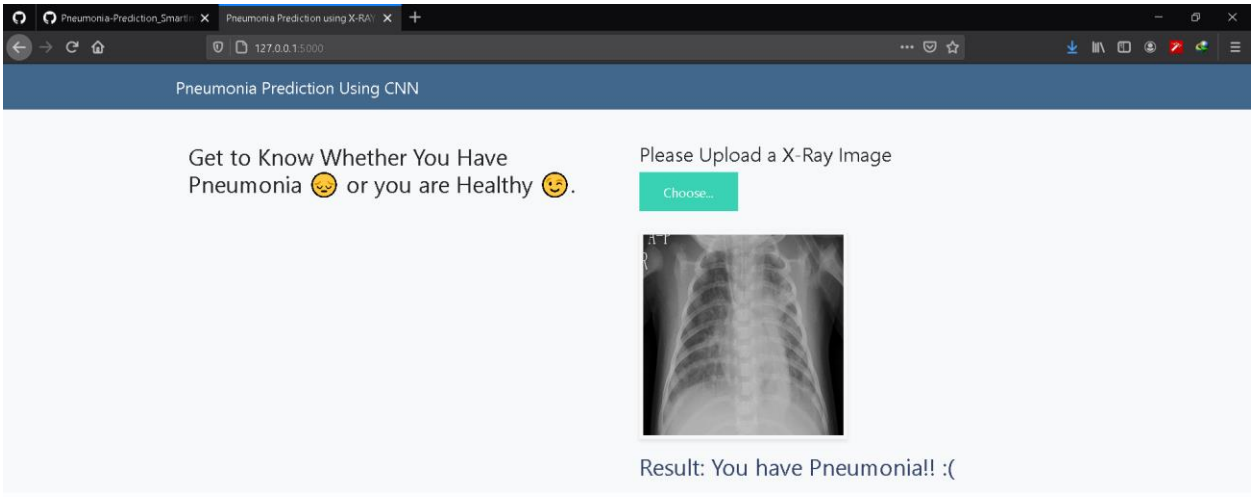
(3) Predict with Interface – Flask

We create a html file for user interface which includes option to upload the x-ray image and a button to submit the image. The button then redirects to another file named app.py which includes our pretrained model and some other code which is necessary to provide the output. The app.py provides back the class of category to html file which intend displays the class of category to the user.

App.py File –

```
import numpy as np
import os
import tensorflow as tf
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
global graph
graph = tf.compat.v1.get_default_graph()
from flask import Flask , request, render_template
app = Flask(__name__)

@app.route('/')
def index():
    return render_template('base.html')


@app.route('/predict',methods = ['GET','POST'])
def preds():
    CATEGORIES = ["You are Normal! :)","You have Pneumonia!! :("]
    if request.method == 'POST':
        f = request.files['image']
        print("current path")
        basepath = os.path.dirname(__file__)
        print("current path", basepath)
        filepath = os.path.join(basepath,'uploads',f.filename)
        print("upload folder is ", filepath)
        f.save(filepath)

        img = image.load_img(filepath,target_size = (64,64))
        img=np.expand_dims(img,axis=0)
        if(np.max(img)>1):
            img=img/255.0

        with graph.as_default():
            model = load_model(r"F:\Project\spyder\models\cnnmodel.h5")
            p=model.predict(img)
            p=model.predict_classes(img)
            print(p)

    return (CATEGORIES[int(p[0][0])])

if  __name__ == "__main__":
    app.run(debug=True)
```

The Interface User Experiences –



Conclusions

Although this project is far from complete but it is remarkable to see the success of deep learning in such varied real-world problems. I have demonstrated how to classify positive and negative pneumonia data from a collection of X-ray images. The model was made from scratch, which separates it from other methods that rely heavily on transfer learning approach. In the future this work could be extended to detect and classify X-ray images consisting of lung cancer and pneumonia. Distinguishing X-ray images that contain lung cancer and pneumonia has been a big issue in recent times, and our next approach should be to tackle this problem.

References

[1] Whitepaper: Top20pneumoniafacts.
http://www.thoracic.org/patients/patient-resources/resources/top-pneumonia-facts.pdf
[2] Evaluation metric:
http://www.kaggle.com/c/rsna-pneumonia-detection-challenge/overview/evaluation,2018.