NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA

Mini project report

Analysis of chest x-ray data to create a Deep learning model for classify images.

Submitted by:
Prasad mandave
181ME147

Development Convolution neural network model.

Table of Contents

Problem statement:

Chest X-ray exams are one of the most frequent and cost-effective medical imaging examinations available. However, clinical diagnosis of a chest X-ray can be challenging and sometimes more difficult than diagnosis via chest CT imaging.

So, for the medial purpose and determining if it's a affected by the disease or not, I have developed a convolution neural network with densenet121 and developed an deep learning model.

Dataset used:

https://www.kaggle.com/nih-chest-xrays/sample

this is the link to a data set that is used for training and testing. The model is trained for 4484 training images and 1122 test images.

For the training purpose I have used densenet121 architecture and used transfer learning those weights are get it from here.

https://github.com/prasad2210/Mini_project_CNN/blob/main/history1.txt

and used a transfer learning to get a result.

Libraries used

Keras:

Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result as fast as possible is key to doing good research.

It is used for developing model and getting model architecture of dense net.

Pandas:

pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.

It is used to storing data set from drive into python variable. And reading images and converting them into multi-dimensional arrays.

NumPy:

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays
It's used for manipulation of the data storing arrays etc.

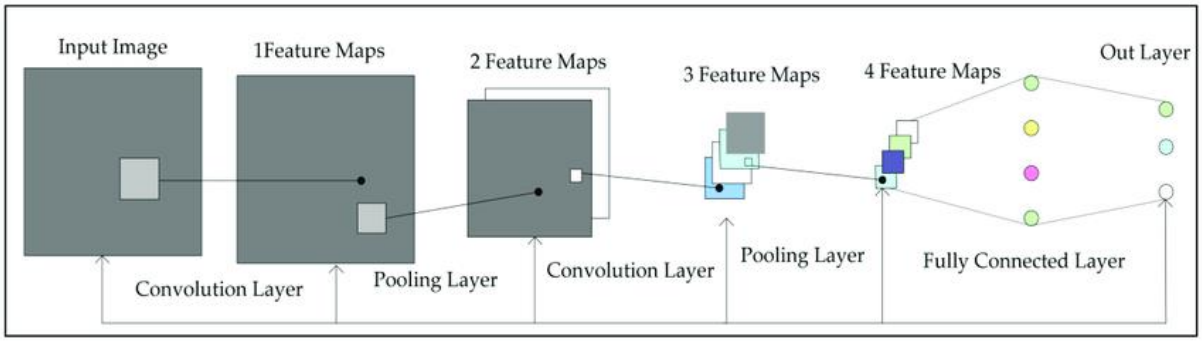
OpenCV:

OpenCV is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez. The library is cross-platform and free for use under the open-source Apache 2 License.

All the code is written in an online platform of google Collaboratory library.

Model architecture

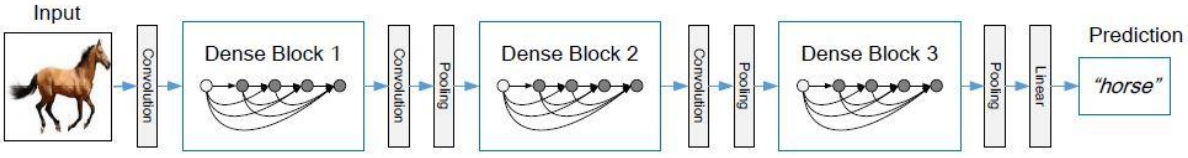
Dense Net is used for this problem statement we will see some of the insights of this architecture.



This is the basic CNN model architecture and its used for getting results.

Dense Net Architecture & Components

components



This is the model architecture which uses dense block and some convolution and pooling layer and at the end it converts to fully connected layer and get output.

Architecture:

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

A summarization of the various architectures implemented for the ImageNet database have been provided in the table above. Stride is the number of pixels shifts over the input matrix. A stride of 'n' (default value being 1), indicates that the filters are moved 'n' pixels at a time.
Using the DenseNet-121 architecture to understand the table, we can see that every dense block has varying number of layers (repetitions) featuring two convolutions each; a 1x1 sized kernel as the bottleneck layer and 3x3 kernel to perform the convolution operation.
Also, each transition layer has a 1x1 convolutional layer and a 2x2 average pooling layer with a stride of 2. Thus, the layers present are as follows:

i. Basic convolution layer with 64 filters of size 7X7 and a stride of 2
ii. Basic pooling layer with 3x3 max pooling and a stride of 2
iii. Dense Block 1 with 2 convolutions repeated 6 times
iv. Transition layer 1 (1 Conv + 1 AvgPool)
v. Dense Block 2 with 2 convolutions repeated 12 times
vi. Transition layer 2 (1 Conv + 1 AvgPool)
vii. Dense Block 3 with 2 convolutions repeated 24 times
viii. Transition layer 3 (1 Conv + 1 AvgPool)
ix. Dense Block 4 with 2 convolutions repeated 16 times
x. Global Average Pooling layer- accepts all the feature maps of the network to perform classification
xi. Output layer

Therefore, DenseNet-121 has the following layers:

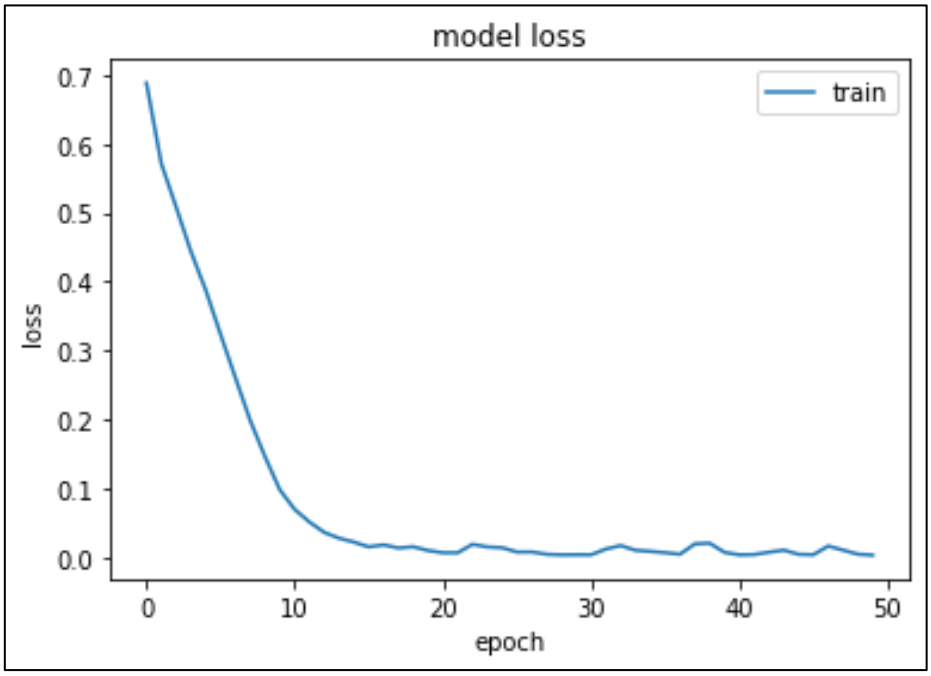
- 1 7x7 Convolution
- 58 3x3 Convolution
- 61 1x1 Convolution
- 4 AvgPool
- 1 Fully Connected Layer

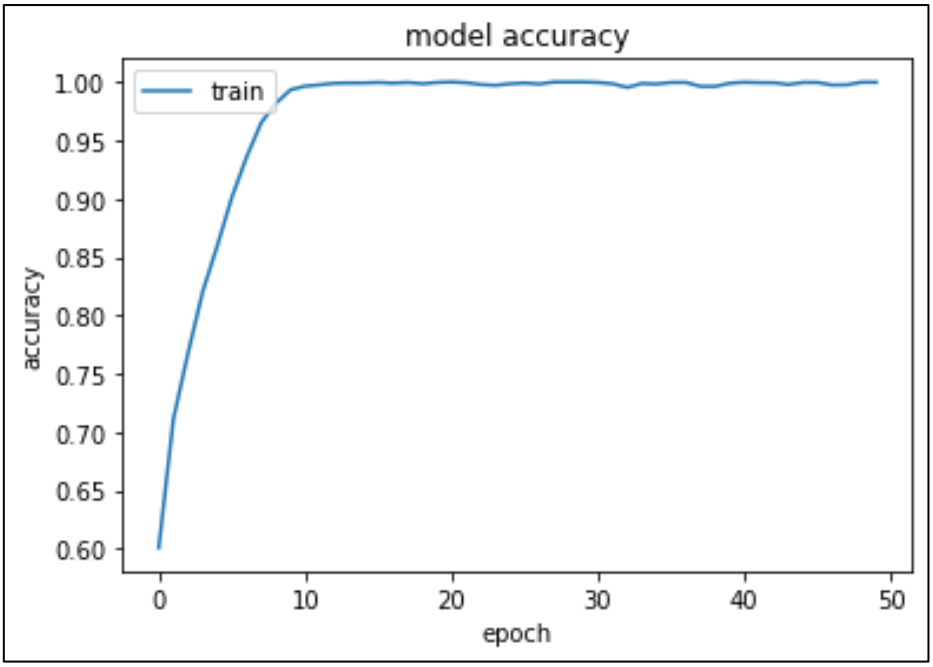
In short, DenseNet-121 has 120 Convolutions and 4 AvgPool.
All layers i.e. those within the same dense block and transition layers, spread their weights over multiple inputs which allows deeper layers to use features extracted early on.
Since transition layers outputs many redundant features, the layers in the second and third dense block assign the least weights to the output of the transition layers.
Also, even though the weights of the entire dense block are used by the final layers, there still may be more high-level features generated deeper into the model as there seemed to be a higher concentration towards final feature maps in experiments.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 densenet121 (Functional)    (None, 1024)              7037504

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 512)               524800

 dense_1 (Dense)             (None, 1)                 513

=================================================================
Total params: 7,562,817
Trainable params: 7,479,169
Non-trainable params: 83,648
```

This is the model summary and get total parameters that need to be trained.

Results





Precision	0.6155507559395248
recall	0.5654761904761905
F1score	0.5894519131334022
Accuracy	0.6461675579322638

Testing results

Threshold value	Correct prediction	Total prediction	Accuracy (%)
0.3	724	1122	64.5276
0.4	728	1122	64.8841
0.5	725	1122	64.6167
0.6	723	1122	64.4385
0.7	724	1122	64.5275

Conclusion and results

So, the training for chest x-ray is done using Dense Net 121 model and got a transfer learning done for pre trained model and got good accuracy.

References

Git hub link:

https://github.com/prasad2210/Mini_project_CNN

drive link:

https://drive.google.com/drive/folders/16HD4sFuNK9YHtHuzpekLOq30ktUFKTHD?usp=sharing

Appendix

Code for the project is given below.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
import numpy as np
from keras.models import Model,Sequential
import pandas as pd
from sklearn.model_selection import train_test_split
import os
import matplotlib.pyplot as plt
from tensorflow.keras.optimizers import Adam
import keras
from scipy import signal
import cv2
from sklearn.metrics import precision_recall_fscore_support
import tensorflow as tf
from keras.layers import Dense, Flatten
from sklearn.metrics import accuracy_score
from natsort import natsorted
```

```
data=pd.read_csv('drive/MyDrive/archive/sample_labels.csv')
x=[]
y=[]
for i in range(0, len(data["Finding Labels"])):
  a=cv2.imread('drive/MyDrive/archive/images/'+data["Image Index"][i])
  a=cv2.resize(a,(224,224))
  x.append(a)
  if(data["Finding Labels"][i]=='No Finding'):
    y.append(0)
  else:
```

```
    y.append(1)


x=np.array(x)
y=np.array(y)
print(x.shape, y.shape)


xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.2)
print(xtrain.shape)
print(ytrain.shape)


encoder = tf.keras.applications.ResNet50(include_top=False,weights='ima
genet',pooling= 'avg', input_shape=(224, 224, 3))
model1=keras.Sequential()
model1.add(encoder)
model1.add(Flatten())
model1.add(Dense(512, activation='relu'))
model1.add(Dense(1, activation='sigmoid'))
model1.summary()
opt=Adam(learning_rate=0.000009,
    beta_1=0.9,
    beta_2=0.999,
    epsilon=1e-07)
model1.compile(loss='binary_crossentropy', optimizer=opt, metrics=['acc
uracy'])

history1= model1.fit(xtrain,ytrain,epochs=50)

print("\n")
ypred1=model1.predict(xtest)
for i in range(0,len(ypred1)):
  if(ypred1[i,0]>0.5):
    ypred1[i,0]=1
  else:
    ypred1[i,0]=0
acc1=accuracy_score(ytest, ypred1)
pr1, re1, f1, _ =precision_recall_fscore_support(ytest, ypred1, average
='binary')
print("Precision:", pr1)
print("Recall:", re1)
print("F1score:", f1)
print("Accuracy:", acc1)
file = open("drive/MyDrive/history1.txt", "w")
var = repr(history1.history)
file.write(var)
file.close()
```

```
print("\n")
plt.plot(history1.history['accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history1.history['loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right')
plt.show()
model1.save('drive/MyDrive/csMinorProject/xray.hdf5')


yhat = model1.predict(xtest,verbose = 0)


print(ytest)
print(yhat)
# print(ytest.shape)
# print(len(ytest))
count = 0

for i in range(len(ytest)):
  temp = 0
  if(yhat[i] >= 0.3):
      temp = 1
  if(temp == ytest[i]):
      count +=1
print(count)



print("accuracy on test data is " + str(count/(len(yhat))))
```