

NIH Chest X-rays Image Classification

Neural Network and

Genetic Algorithms

Author:

BENHEDDI Alaeddine

1. **Introduction:**

This project aims to build a convolutional neural network (CNN) for NIH chest x-ray images, which is a dataset of 112,120 frontal-view X-ray images of 30,805 unique patients with fourteen different diseases with a size of 1024x1024, which will be reduced later to train the CNN more efficiently. I experimented different techniques to preprocess the data and CNN models to try and get the best performance possible, and used different libraries in this process (Tensorflow , numpy etc.). The best model accuracy reaches during this process was 55%. And while working on this project I have faced various challenges like limited memory (RAM), very big dataset etc. and due to memory issues I used to python files the first to collect all the images from the directories and filter it to get only the data I will use.

1. **Methodology:**
   1. Data collection:

I used NIH chest x-ray , which consists of 112,120 1024\*1024 images sizes in 15 different classes, one of them is patients with no deceases (‘No Finding’ class) and the rest are with deceases which are Atelectasis, Consolidation, Infiltration,Pneumothorax, Edema, Emphysema, Fibrosis, Effusion, Pneumonia, Pleural\_thickening, Cardiomegaly, Nodule, Mass and Hernia. And there some patients with multiple deceases which won’t be treated. And the number of images is not divided equally to the classes, because we have more than half of the images are with ‘No Finding class’ which causes a problem to classify so I decided to use only the images of the 14 deceases.

* 1. Data Preprocessing:

In the first file (Data\_ Preprocessing.py) I started by importing all the libraries that I will use:

import pandas as pd   
import numpy as np  
import matplotlib.pyplot as plt  
import os   
import cv2  
import pickle as pk

Then read the csv file to get the images names with their Finding Labels (deceases):

csv\_data = pd.read\_csv('Data\_Entry\_2017.csv')  
print(csv\_data.head(5))

Image Index Finding Labels Follow-up # Patient ID \  
0 00000001\_000.png Cardiomegaly 0 1   
1 00000001\_001.png Cardiomegaly|Emphysema 1 1   
2 00000001\_002.png Cardiomegaly|Effusion 2 1   
3 00000002\_000.png No Finding 0 2   
4 00000003\_000.png Hernia 0 3   
  
 Patient Age Patient Gender View Position OriginalImage[Width Height] \  
0 58 M PA 2682 2749   
1 58 M PA 2894 2729   
2 58 M PA 2500 2048   
3 81 M PA 2500 2048   
4 81 F PA 2582 2991   
  
 OriginalImagePixelSpacing[x y] Unnamed: 11   
0 0.143 0.143 NaN   
1 0.143 0.143 NaN   
2 0.168 0.168 NaN   
3 0.171 0.171 NaN   
4 0.143 0.143 NaN

So as we can see we have a column of only NaN so I deleted it:

csv\_data = csv\_data.iloc[:,:-1]

Now we start filtering the data by removing all rows with multiple deceases which were separated by the character ‘|’ and patient age higher than 122:

for i in range(len(csv\_data)):  
 if ('|' in csv\_data['Finding Labels'][i]) or (csv\_data['Patient Age'][i]>122):  
 csv\_data = csv\_data.drop(index=i)  
csv\_data.reset\_index(drop=True, inplace=True)

Visualise the occurrences of each finding labels

labels\_occ = csv\_data['Finding Labels'].value\_counts()  
labels\_occ.plot(kind='bar')  
plt.title("Number of occurrences of Finding Labels")  
plt.xlabel("Finding Labels")  
plt.ylabel("Number of occurrences")  
plt.show()

Une image contenant texte, capture d’écran, diagramme, Tracé

Description générée automatiquement

As we can see on the graphe the no Finding class is the more than the other class combined so I removed it and used only the 14 remaining to have a better managing of the data and less memory to use:

for i in range(len(csv\_data)):  
 if 'No Finding' in csv\_data['Finding Labels'][i]:  
 csv\_data = csv\_data.drop(index=i)  
csv\_data.reset\_index(drop=True, inplace=True

Then I started by getting all the directories to the pictures:

url = 'images\_00'  
Dir = []  
urls = []  
for i in range(1, 13):  
 if i >= 10:  
 url = 'images\_0'  
 Dir.append(os.listdir(url + str(i)+'/images'))

And now we get the URLs to the images.

url = 'images\_00'  
for i in range(len(Dir)):  
 for j in range(len(Dir[i])):  
 if i == 9:  
 url = 'images\_0'  
 urls.append(url + str(i+1) + '/images/' + Dir[i][j])

Then I took only the urls to images that I will be using to build the model

images = csv\_data['Image Index']  
j = 0  
filter\_urls = []  
for i in range(len(urls)):  
 if images[j] in urls[i]:  
 filter\_urls.append(urls[i])  
 j += 1  
 if j>=len(images):  
 break

The training and testing data split was provided, so I create variables to put into files for the training.

test\_list = pd.read\_csv('test\_list.txt',header=None , names=['test\_list'])  
train\_list = pd.read\_csv('train\_val\_list.txt',header=None , names=['train\_list'])  
  
train\_list = np.array(train\_list)  
test\_list = np.array(test\_list)  
  
images\_used = np.array(csv\_data['Image Index'])  
  
test\_imgs = test\_list[np.isin(test\_list,images\_used)]  
train\_imgs = train\_list[np.isin(train\_list,images\_used)]  
  
train\_imgs = train\_imgs.tolist()  
test\_imgs = test\_imgs.tolist()

In this step I read all the pictures and transformed them to 64\*64 because of the limited memory I had and then split them into train and test data with their corresponding labels

Labels = csv\_data['Finding Labels']  
input\_train = []  
target\_train = []  
input\_test = []  
target\_test = []  
j = 0  
for i in range(len(Labels)):  
 if j < len(train\_imgs) and train\_imgs[j] in filter\_urls[i]:  
 tmp\_img = cv2.imread(filter\_urls[i])  
 img = cv2.resize(tmp\_img, (64, 64))  
 input\_train.append(img)  
 target\_train.append(Labels[i])  
 j += 1  
 else:  
 tmp\_img = cv2.imread(filter\_urls[i])  
 img = cv2.resize(tmp\_img, (64, 64))  
 input\_test.append(img)  
 target\_test.append(Labels[i])

final\_data = (input\_train, target\_train), (input\_test, target\_test)

In the end I put them into a file to use later in the second python file which will be for the model building   
with open('data\_2.pkl', 'wb') as f:  
 pk.dump(final\_data, f)

Move on the second file ‘Model\_creation.py’ I started as usual by importing the libraries needed:

import pickle as pkl  
import numpy as np  
import tensorflow as tf  
import matplotlib.pyplot as plt  
  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout  
from tensorflow.keras.losses import sparse\_categorical\_crossentropy  
from tensorflow.keras.optimizers import RMSprop  
from tensorflow.keras.preprocessing.image import ImageDataGenerator  
from sklearn.metrics import ConfusionMatrixDisplay  
from keras.utils import to\_categorical

Then opening the date created earlier.

with open('data\_2.pkl', 'rb') as f:  
 (input\_train, target\_train), (input\_test, target\_test) = pkl.load(f)

to transform each label into an int I created a dictionary for each label a number between 0 and 13:

deceases = []  
deceases\_dic = {}  
for i in range(len(target\_test)):  
 if target\_test[i] not in deceases:  
 deceases.append(target\_test[i])  
   
for i in range(len(deceases)):  
 deceases\_dic.update({deceases[i]: i})

{'Hernia': 0, 'Pleural\_Thickening': 1, 'Infiltration': 2, 'Pneumothorax': 3, 'Effusion': 4, 'Mass': 5, 'Emphysema': 6, 'Cardiomegaly': 7, 'Consolidation': 8, 'Edema': 9, 'Atelectasis': 10, 'Fibrosis': 11, 'Pneumonia': 12,Nodule': 13}

And then replaced them in the target data:

for i in range(len(target\_train)):  
 if i < len(target\_test):  
 target\_test[i] = deceases\_dic[target\_test[i]]  
 target\_train[i] = deceases\_dic[target\_train[i]]

and in the end of the data preprocessing when divide de input data by 255 to have values between 0 and 1 and turn them into numpy array , plus converting target data to categorical :

target\_train = to\_categorical(target\_train,num\_classes=14)  
target\_test = to\_categorical(target\_test,num\_classes=14)

input\_train = np.array(input\_train, dtype=np.float32)  
input\_test = np.array(input\_test, dtype=np.float32)  
target\_train = np.array(target\_train ,dtype=np.float32)  
target\_test = np.array(target\_test,dtype=np.float32)

input\_test = input\_test / 255  
input\_train = input\_train / 255

* 1. Model:

First, initializing a data generator to apply some transformation on the input images:

imgen = ImageDataGenerator(  
 width\_shift\_range=0.2,  
 height\_shift\_range=0.2,  
 zoom\_range=0.2,  
 horizontal\_flip=True,  
)  
imgen.fit(input\_train)

and then starting to build the model:

net = Sequential()  
  
net.add(Conv2D(32, kernel\_size=(3,3), activation='relu',input\_shape=(64, 64, 3)))  
net.add(MaxPooling2D(pool\_size=(2, 2)))  
net.add(Dropout(0.25))  
net.add(Conv2D(64, kernel\_size=(3,3),activation='relu' ))  
net.add(MaxPooling2D(pool\_size=(2, 2)))  
net.add(Dropout(0.25))  
net.add(Conv2D(128, kernel\_size=(3,3),activation='relu'))  
net.add(MaxPooling2D(pool\_size=(2, 2)))  
net.add(Dropout(0.25))  
net.add(Flatten())  
net.add(Dense(128, activation='relu'))  
net.add(Dropout(0.5))  
net.add(Dense(256, activation='relu'))  
net.add(Dropout(0.5))  
net.add(Dense(14, activation='softmax'))  
  
net.summary()

the model is composed of 3 convolutional layers with size 32 ,64 and 128 and ‘relu’ as an activation.

And each one of these layers is followed by MaxPooling to reduce the dimension of the input. And a Dropout layer that 25% of the neurons during the training to avoid overfitting.

Then the data flattened to 1D vector to put into the connected layers which are three, the first two are hidden layer and the last one is to predict the percentage of each class in an image with a SoftMax activation.

The summary of the model :  
┃ Layer (type) ┃ Output Shape ┃ Param # ┃  
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩  
│ conv2d (Conv2D) │ (None, 62, 62, 32) │ 896 │  
├─────────────────────────────────┼────────────────────────┼───────────────┤  
│ max\_pooling2d (MaxPooling2D) │ (None, 31, 31, 32) │ 0 │  
├─────────────────────────────────┼────────────────────────┼───────────────┤  
│ dropout (Dropout) │ (None, 31, 31, 32) │ 0 │  
├─────────────────────────────────┼────────────────────────┼───────────────┤  
│ conv2d\_1 (Conv2D) │ (None, 29, 29, 64) │ 18,496 │  
├─────────────────────────────────┼────────────────────────┼───────────────┤  
│ max\_pooling2d\_1 (MaxPooling2D) │ (None, 14, 14, 64) │ 0 │  
├─────────────────────────────────┼────────────────────────┼───────────────┤  
│ dropout\_1 (Dropout) │ (None, 14, 14, 64) │ 0 │  
├─────────────────────────────────┼────────────────────────┼───────────────┤  
│ conv2d\_2 (Conv2D) │ (None, 12, 12, 128) │ 73,856 │  
├─────────────────────────────────┼────────────────────────┼───────────────┤  
│ max\_pooling2d\_2 (MaxPooling2D) │ (None, 6, 6, 128) │ 0 │  
├─────────────────────────────────┼────────────────────────┼───────────────┤  
│ dropout\_2 (Dropout) │ (None, 6, 6, 128) │ 0 │  
├─────────────────────────────────┼────────────────────────┼───────────────┤  
│ flatten (Flatten) │ (None, 4608) │ 0 │  
├─────────────────────────────────┼────────────────────────┼───────────────┤  
│ dense (Dense) │ (None, 128) │ 589,952 │  
├─────────────────────────────────┼────────────────────────┼───────────────┤  
│ dropout\_3 (Dropout) │ (None, 128) │ 0 │  
├─────────────────────────────────┼────────────────────────┼───────────────┤  
│ dense\_1 (Dense) │ (None, 256) │ 33,024 │  
├─────────────────────────────────┼────────────────────────┼───────────────┤  
│ dropout\_4 (Dropout) │ (None, 256) │ 0 │  
├─────────────────────────────────┼────────────────────────┼───────────────┤  
│ dense\_2 (Dense) │ (None, 14) │ 3,598 │  
└─────────────────────────────────┴────────────────────────┴───────────────┘

Total params: 719,822 (2.75 MB)

Trainable params: 719,822 (2.75 MB)

Non-trainable params: 0 (0.00 B)

And then fitting the model where the data is divided into 32 batches and 20% of the input data is for validation :

fit = net.fit(input\_train,target\_train,  
 batch\_size=32,validation\_split=0.2,  
 verbose=1,epochs=100)

after that we evaluate the model using test data:

score = net.evaluate(input\_test,target\_test,batch\_size=32,verbose=1)  
print(f'Test loss: {score[0]} / Test accuracy: {score[1]}')

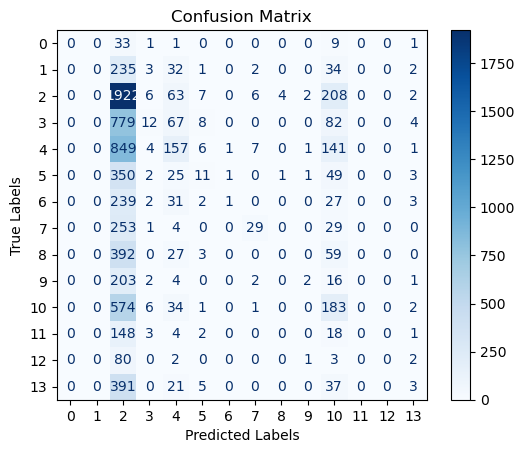
where we got as Test loss: 2.2547 / Test accuracy: 0.2903.

which is not that good.

Then we build a confusion matrix to visualize where the data is overfitting,

pred = net.predict(input\_test)  
pred = np.argmax(pred , axis=1)  
target\_test = np.argmax(target\_test , axis=1)  
  
conf\_matrix = tf.math.confusion\_matrix(target\_test, pred)

conf\_matrix = conf\_matrix.numpy()  
ConfusionMatrixDisplay(confusion\_matrix=conf\_matrix).plot(cmap='Blues')  
plt.xlabel('Predicted Labels')  
plt.ylabel('True Labels')  
plt.title('Confusion Matrix')  
plt.show()



Then plot the graphs of the accuracy and the loss of training and validation:

plt.plot(fit.history['accuracy'])  
plt.plot(fit.history['val\_accuracy'])  
plt.title('Accuracy history')  
plt.ylabel('accuracy')  
plt.xlabel('Epoch')  
plt.show()

Une image contenant texte, Tracé, ligne, capture d’écran

Description générée automatiquement

plt.plot(fit.history['loss'])  
plt.plot(fit.history['val\_loss'])  
plt.title('Loss history')  
plt.ylabel('Loss')  
plt.xlabel('Epoch')  
plt.show()

Une image contenant texte, Tracé, capture d’écran, ligne

Description générée automatiquement

1. **Results and Challenges:**
   1. Results

The accuracy of model wasn’t good because of the difference of the number of images provided for each class as we can see there are some with over 5000 and other with just few hundred which causes the model to not train very good and get the problem of overfitting. Because as we can see on the graphs the validation isn’t improving while the training one does.

* 1. Challenges:

During this process I have faced multiple difficulties for example:

In the beginning I tried to use all the data with ‘No finding’ class but the model wasn’t improving at all which consumed a good amount of time to train without any positive result because I had all the prediction in the same columns of the confusion matrix and the reason for that is very large number of ‘No Finding label’ Compared to the other which was 2 times more than the rest of them combined. On the other hand, there was some data that had a very small (less than 1000 image) number of images like Hernia and Pneumonia, so I had to remove them to get a better accuracy of the model.

Then I had a problem of memory sometimes the model doesn’t start training or stops in the middle, because of the size of the data, which was big, so I had to reduce it which causes the loss of some data for training, so I had less precision.

And in the beginning, I tried to use some complex model to train which weren’t effective and was just giving weird results, so I had taken make simpler models and build from there.

The training of the model took a lot of time so most of the time it was pointless to let it train for several hours, so I tried to put small number of iterations and analyze the graph to see if it will be better or not.

1. **Conclusion:**

This project developed a CNN for classifying NIH chest X-ray images into fourteen disease categories. Despite preprocessing efforts to balance the dataset, the model achieved a test accuracy of only 29%. Key challenges included handling a large dataset with limited memory, managing class imbalance, and avoiding overfitting. The necessity to reduce image resolution to 64x64 affected model performance. Simplifying the model architecture and employing data augmentation were partially effective strategies.

1. References:

[1] https://www.tensorflow.org/

[2] keras.io

[3] https://www.geeksforgeeks.org/

[4] <https://stackoverflow.com/>

[5] https://docs.python.org/