Binary Classification Using Convolution Neural Network (CNN) Model

Mayank Verma · Follow
7 min read · May 8, 2022

Binary classification is used in the machine learning domain commonly. It is the simplest way to classify the input into one of the two possible categories. For example, give the attributes of apple-like Color, weight, etc. that classify the fruit as either a green apple or red apple. With the help of effective use of Neural Networks (Deep Learning Models), binary classification problems can be solved to a fairly high degree.

Here we are using Convolution Neural Network(CNN). It is a class of Neural network that has proven very effective in areas of image recognition, processing, and classification. In this blog, we will be focusing on image processing only.

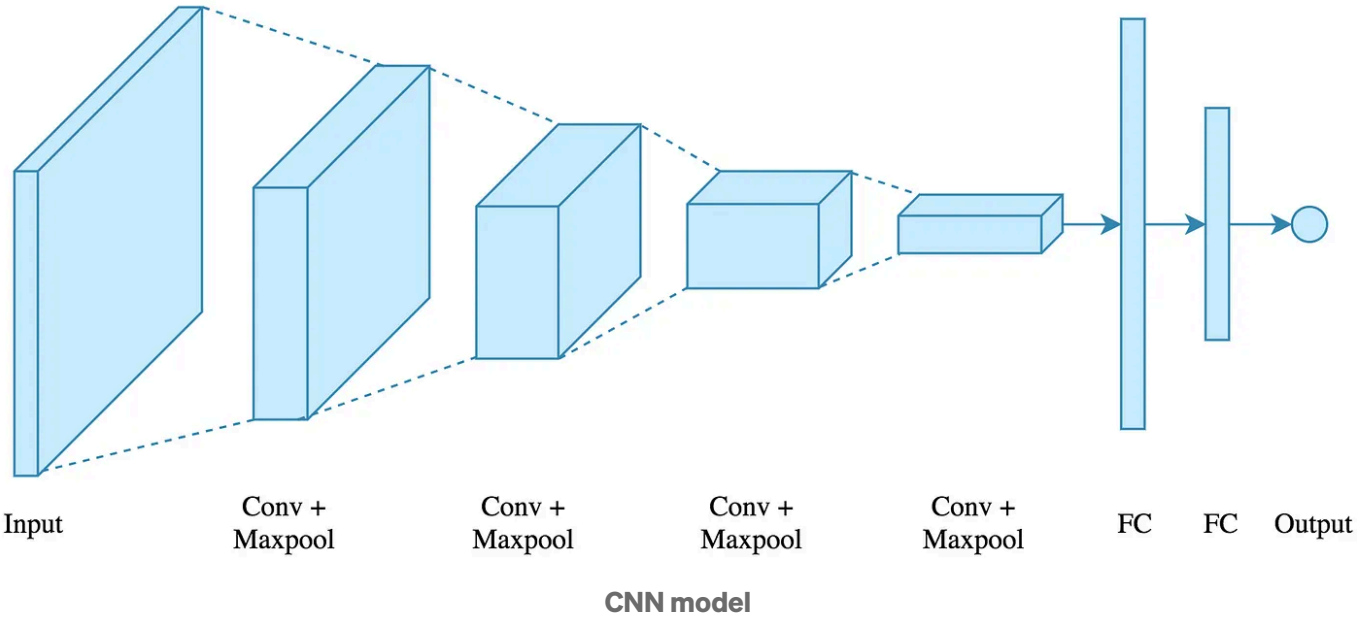
CNN model requires training data for training weights and validation for checking its performance. Each input images passes through a series of

convolution layers with filters(Kernels) :

1. Convolution layers
2. Pooling layers
3. Fully-connected layer (FC) applying the sigmoid function.

the sigmoid function is used to classify an object with a probabilistic value which turns out as 0 or 1 for binary classification.

Here we can see a simple CNN model used for binary classification.



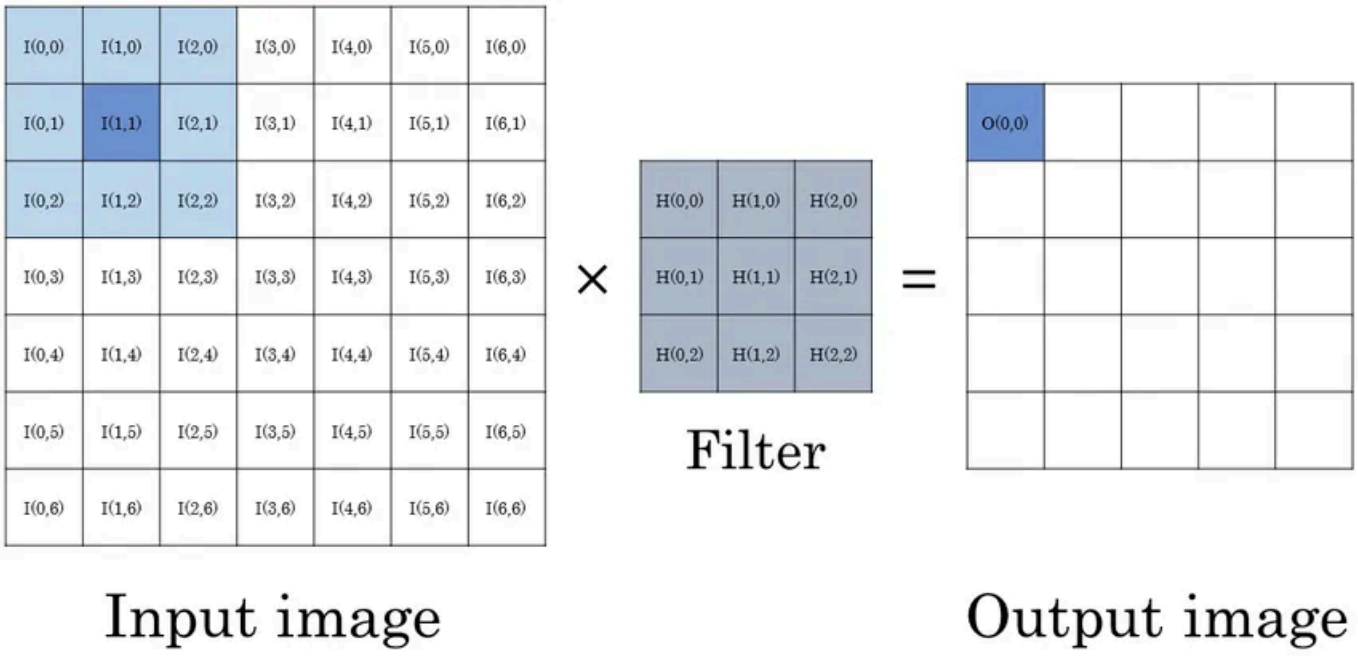
CNN model

the Convolution + MaxPooling layers act as feature extractors from the input image while a fully connected layer acts as a classifier. In the above image figure, on receiving an image as input, the network will assign assigns the highest probability for it and predict which class the input image belongs to.

The operations listed above are the basic building blocks of every Convolutional Neural Network.

Convolution Layers

Image is in the form of a matrix, **Convolution** is the process of adding each element of the image to its local neighbors, weighted by the kernel. In convolution, we use various kinds of filters for extracting features from a given image.



Convolution on an image

In the above image example, we have an image of size 7X7 and we apply a mask or filter of 3X3 and get an output image of 5X5. here we multiply each value of the mask with corresponding values of a portion of the input image under the mask and take the sum of the product of the output and replace the central value with it.

We usually take the size or dimension of the mask as odd numbers so that we have only one central position. this mask the slide all over the input image and generates an output image whose size is dependent on padding is applied or not.

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

n_{in}: number of input features
n_{out}: number of output features
k: convolution kernel size
p: convolution padding size
s: convolution stride size

output size

There are two types of padding: No padding, Same padding.

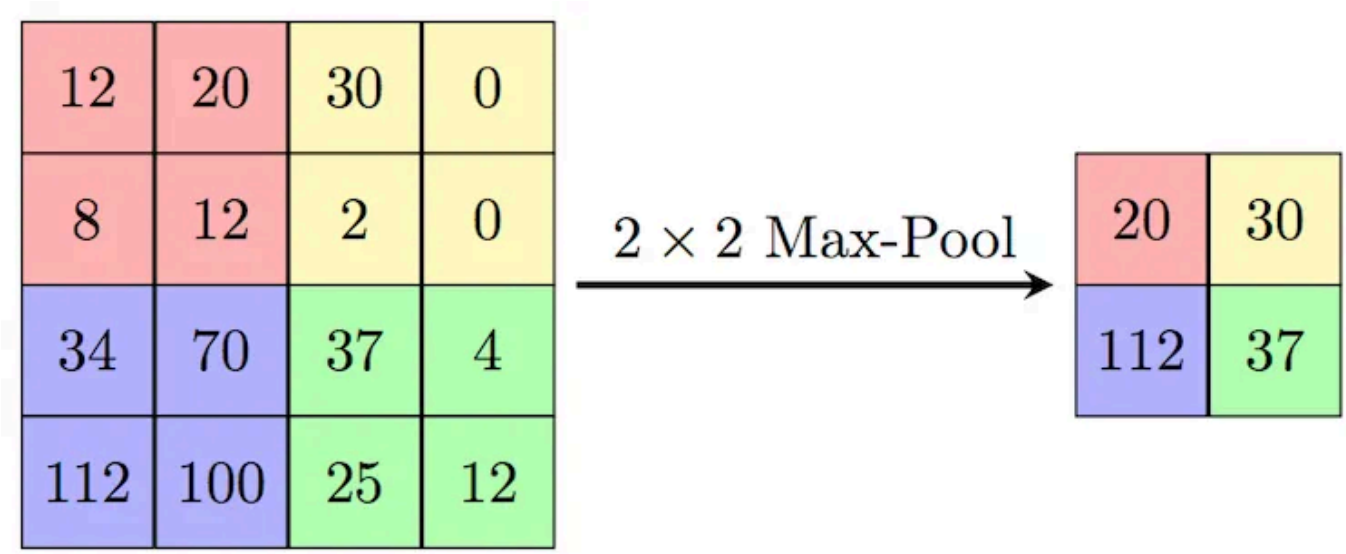
In No padding, the output image is not of the same size as the input as we can see in the above formula padding=0 and the out image is smaller than the input.

but in the same padding output image is equal to the input image.

Here **Stride** is the number of pixels shifts over the input matrix and kernel size is the size of the mask.

Pooling Layer

Also called subsampling or downsampling. The pooling layer does a downsampling operation along the spatial dimensions (width, height) ie. reduces the dimensionality of each feature map but retains the most important information. Max pooling is a type of pooling that extracts only those features which have the highest value.



Max Pooling

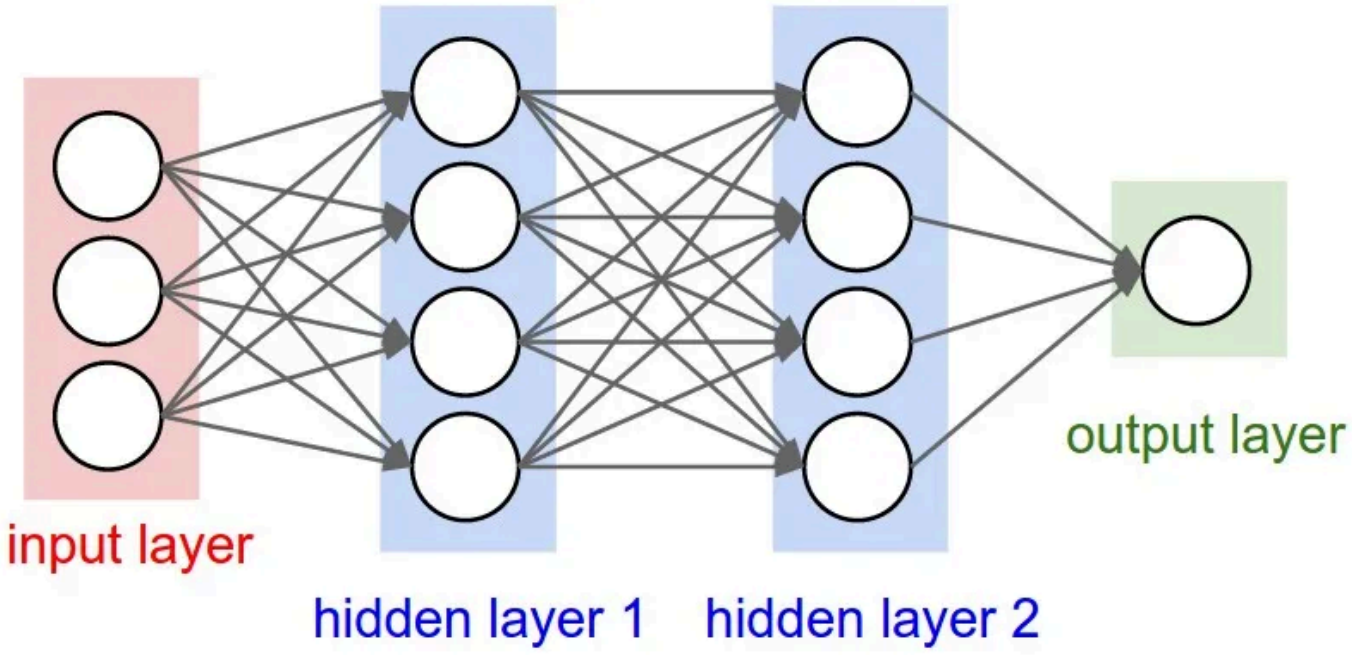
In the above image, max Pooling is choosing maximum value features which means max-pooling is giving more importance to those features which have a higher value.

there are other types of pooling that also exist like average pooling which will take the average of a subsection and update it in the output.

We usually do not count the pooling layer as a layer we consider the convolution +Pooling layer as one layer as the convolution layer generates weights but the Pooling layer only reduces the size of an image by giving importance to the import features and it does not produce any weights.

Fully Connected Layer

In a Fully Connected Layer -each node is connected to every other node in the adjacent layer. The layer that is fully connected uses the sigmoid function for the commuting class of the input image.



Fully Connected Layer

here we can see in the final layer which is the output layer which contains only one node which gives either 1 or 0 as output as it is using sigmoid as an activation function.

Here we can consider the input layer as a flatten layer in the main CNN model, we do not count the flatten layer as a layer because it does not produce any weights it just converts our image into a nested array.

Implementation

This is a demo example, where I have shown how CNN can be used in the medical domain. There are various models which can be used on medical images such as CT scans, X-rays, etc. Here I am taking the X-rays dataset.

Firstly we need to import some libraries into the python notebook. I took data set of Covid 19 patients which contain X-ray images of Covid as well as Normal cases. first, we need to split our data into the training and validation parts.

```
TRAIN_PATH = "Covid_dataset/train"
VAL_PATH = "Covid_dataset/Validation"
```

I already made two folders train and validation which further contains two folders each of positive and normal cases folder.

```
import numpy as np
import matplotlib.pyplot as plt
import keras
from keras.layers import *
from keras.models import *
from keras.preprocessing import image
```

I used some basic libraries like NumPy, Keras, etc for performing tasks on images.

I made a basic CNN model that contains 4 convolutional layers, and 2 fully connected layers. Here, in the first layer, we put 32 nodes with mask size 3X3 and activation function as relu, which is applied per pixel and replaces all negative pixel values in the feature map by zero. and drop out of 25% which means every time 75% of nodes are selected randomly for the operation, it will increase randomness in the model which will reduce bias in output.

```
# CNN Based Model in Keras

model = Sequential()
model.add(Conv2D(32,kernel_size=(3,3),activation='relu',input_shape=(224,224,3)))
model.add(Conv2D(64,(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(64,(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(128,(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(64,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1,activation='sigmoid'))

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

Similarly, I made the 2nd layer which contains 64 nodes having activation function as relu, 3rd layer of 64 nodes, and 4th layer with 128 nodes and the 2 fully connected layer

I used binary cross-entropy as a loss function and adam as an optimizer.

```
model.summary()
```

with the help of model.summary we can analyze our model.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 222, 222, 32)      896
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 220, 220, 64)      18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 110, 110, 64)      0
_________________________________________________________________
dropout_1 (Dropout)          (None, 110, 110, 64)      0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 108, 108, 64)      36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 54, 54, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 54, 54, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 52, 52, 128)       73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 26, 26, 128)       0
_________________________________________________________________
dropout_3 (Dropout)          (None, 26, 26, 128)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 86528)             0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                5537856
_________________________________________________________________
dropout_4 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
=================================================================
Total params: 5,668,097
Trainable params: 5,668,097
Non-trainable params: 0
```

My model is a 6-layered model which has 4 convolution layers, and 2 fully connected layers.

```
# Train from scratch
train_datagen = image.ImageDataGenerator(
    rescale = 1./255,
    shear_range = 0.2,
    zoom_range = 0.2,
    horizontal_flip = True,
)

test_dataset = image.ImageDataGenerator(rescale=1./255)
```

we can define a train generator and test generator from scratch where we can rescale, and resize our images. here, I applied image augmentation and also converted every image into binary(0 and1 or black and white).

```
train_generator = train_datagen.flow_from_directory(
    'Covid_dataset/train',
    target_size = (224,224),
    batch_size = 32,
    class_mode = 'binary')
```

```
Found 432 images belonging to 2 classes.
```

```
train_generator.class_indices
```

```
{'covid': 0, 'normal': 1}
```

```
validation_generator = test_dataset.flow_from_directory(
    'Covid_dataset/Validation',
    target_size = (224,224),
    batch_size = 32,
    class_mode = 'binary')
```

```
Found 150 images belonging to 2 classes.
```

after initializing generators we can define the flow of our data like, in which size image will be given to our model. here, we define 224X224 as image size, batch size =32, and class mode as binary.

```
hist = model.fit_generator(
    train_generator,
    steps_per_epoch=8,
    epochs = 10,
    validation_data = validation_generator,
    validation_steps=2
)
```

```
Epoch 1/10
8/8 [==============================] - 34s 4s/step - loss: 0.7818 - accuracy: 0.6083 - val_loss: 0.6778 - val_accuracy: 0.5312
Epoch 2/10
8/8 [==============================] - 30s 4s/step - loss: 0.6046 - accuracy: 0.6680 - val_loss: 0.5118 - val_accuracy: 0.9219
Epoch 3/10
8/8 [==============================] - 30s 4s/step - loss: 0.4126 - accuracy: 0.8672 - val_loss: 0.2043 - val_accuracy: 0.9630
Epoch 4/10
8/8 [==============================] - 28s 3s/step - loss: 0.3557 - accuracy: 0.8616 - val_loss: 0.3599 - val_accuracy: 0.9062
Epoch 5/10
8/8 [==============================] - 29s 4s/step - loss: 0.1730 - accuracy: 0.9375 - val_loss: 0.0534 - val_accuracy: 0.9630
Epoch 6/10
8/8 [==============================] - 31s 4s/step - loss: 0.2437 - accuracy: 0.9336 - val_loss: 0.1781 - val_accuracy: 0.9688
Epoch 7/10
8/8 [==============================] - 28s 4s/step - loss: 0.1696 - accuracy: 0.9458 - val_loss: 0.1298 - val_accuracy: 0.9531
Epoch 8/10
8/8 [==============================] - 30s 4s/step - loss: 0.2078 - accuracy: 0.9336 - val_loss: 0.2444 - val_accuracy: 0.9630
Epoch 9/10
8/8 [==============================] - 37s 5s/step - loss: 0.1516 - accuracy: 0.9625 - val_loss: 0.1121 - val_accuracy: 0.9844
Epoch 10/10
8/8 [==============================] - 37s 5s/step - loss: 0.1424 - accuracy: 0.9531 - val_loss: 0.0512 - val_accuracy: 1.0000
```

with the help of a fit generator, we will able to train our model on train data and check its training as well as validation accuracy in each step.

```
model.evaluate_generator(train_generator)
```

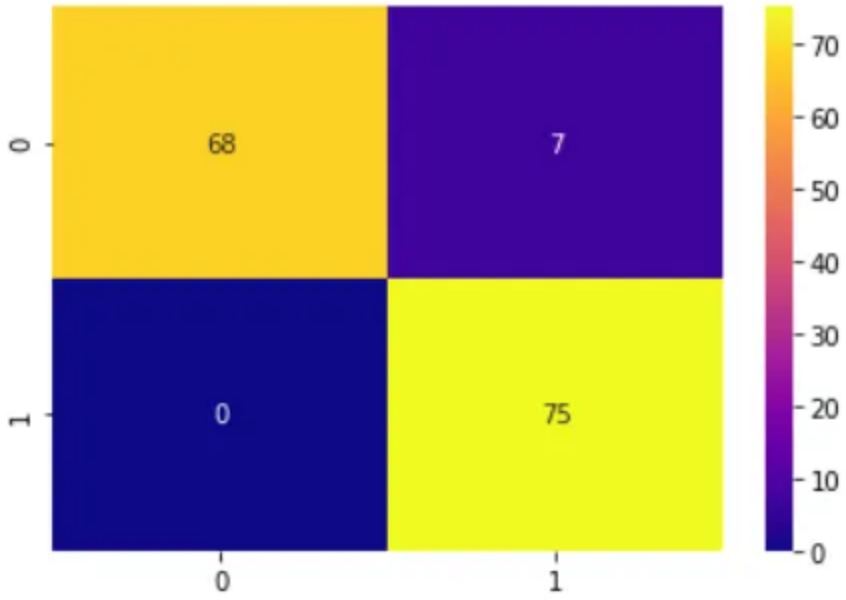
```
[0.2159358710050583, 0.9537037014961243]
```

I got an accuracy of 95% on training data.

```
model.evaluate_generator(validation_generator)
```

```
[0.05385594442486763, 0.9800000190734863]
```

And got 98% accuracy on testing data.



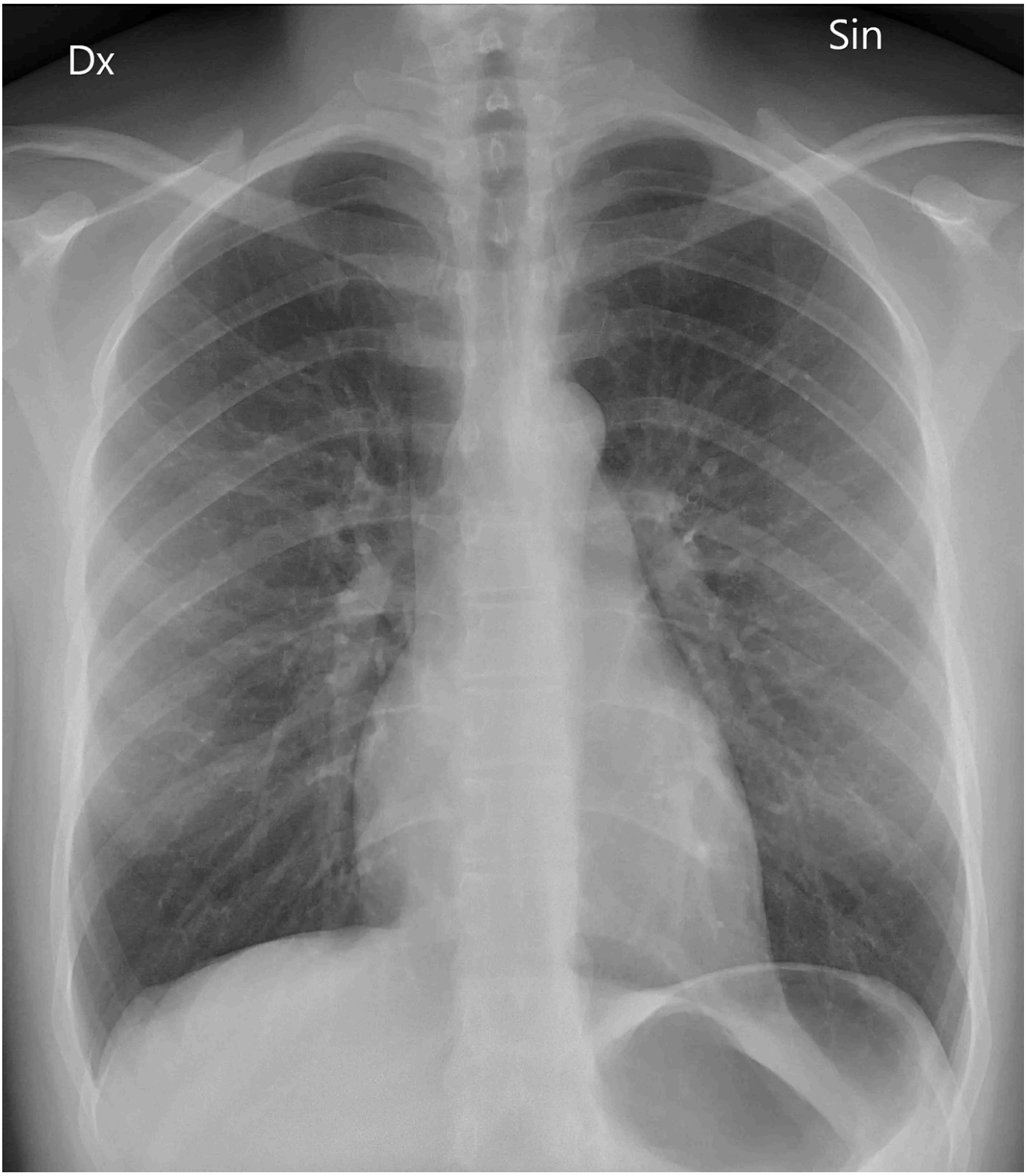
confusion Matrix

this is the confusion matrix of the validation set which contain 75 images of both covid and normal cases

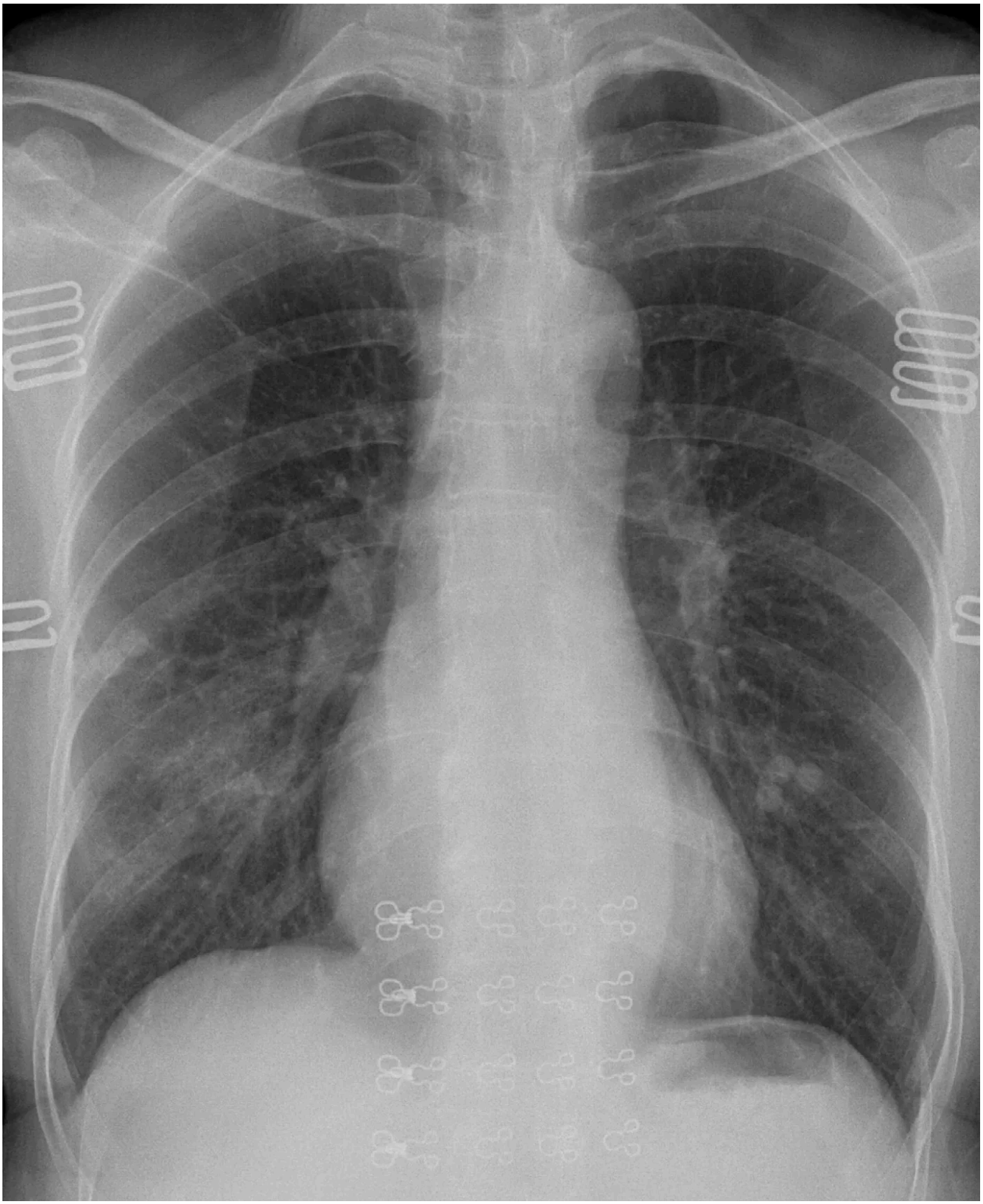
Here 0 means covid positive and 1 means the normal case.

Dataset

I made a dataset from the various repository on keggle.com which contained 290 images of X-ray of covid 19 positive cases and 292 images of normal cases.



Normal case X-ray



Covid +ve case

As we can see we are not able to distinguish between them which one is positive and which one is negative ourselves. with the help of deep learning