

IZMIR UNIVERSITY OF ECONOMICS FACULTY OF ENGINEERING

SOFTWARE & ELECTRICAL ENGINEERING

FENG 498 PROJECT REPORT

AIX-RAY



Author(s):

Güven Sarı

Metin Güney Öztürk

Doğukan Özyurt

Supervisor:

Dr. Faezeh Yeganli

OUTLINE

Title page

Table of Contents

1. Abstract

2. Introduction

2.1. Problem Statement

2.2. Motivation

3. Literature Review

4. Methodology

4.1. Convenient Data Research

4.2. Researching CNN Architecture

4.3. Model Decision

4.4. Model Modification and Training Testing Process

4.5 Interface Design

5. Conclusion

6. References

7. Appendix

1. Abstract

The aid of smart devices has made our lives easier. But in the specific area that we are focusing on, we believe a little more assistance is required. Our motivation is to make faster chest X-rays examinations. Also, doctors may decide incorrectly due to their long work hours. This study will help the doctor to make proper decisions on disease diagnosis when they feel exhausted. Moreover, we believe computer science should take more part in the health industry.

So that, there will be not only doctor's decisions but there will be machine learning and deep learning algorithms. This will reduce the error caused by the human factor. Instead of directly looking into the x-ray image, the doctor will receive our program's report on this image and decide.

We have already covered some main points of image processing in the order of our schedule. We have covered most of the mathematical operations which are median filter, image sharpening, Laplacian transforms, Gaussian filter, mean filter, etc.

We want to make it easy to detect diseases as quickly as possible. Although raising a radiologist needs years and it has a high cost. Our project will help the radiologist and the doctor in terms of time reduction.

2. Introduction

Uncertainty in the health industry may cause some incorrect diagnoses and it affects the patient in wrong medical treatment. Besides, that is a time-wasting procedure, also going with the wrong treatment can negatively affect the progression of the disease due to misdiagnosis. It is possible that incorrect diagnoses and treatments can even cause death. Therefore, we try to solve this problem with the opportunities brought by the age of technology.

Medical image processing may differ for each person. Every patient has a different lung type, immune system, and disease. According to this situation, the researchers of medical and computer experts had created different types of medical images. (US, X-ray, MRI, CT (computed tomography)). As we want to process medical images on the computer, CNN can provide us to automatically learn significant low-level features (edges, lines, shapes). There are two types of image classification which are supervised and unsupervised.

DenseNet, GoogLeNet, AlexNet, and VGGNet have different classification processes to classify images. GoogleNet with its CNN classified the orientation of the image into frontal or lateral views with near 100% accuracy. [1] DenseNet [2] with 121 convolutional layers called CheXNet to classify 14 different diseases seen on the chest x-rays, using 112,000 images from the ChestXray14 [3] dataset. The VOXCNN and ResNet, which were based on VGGNet and Residual neural network architectures, used the ADNI (Alzheimer Disease Neuroimaging Initiative) database to discriminate between normal and Alzheimer's Disease patients.

Although most of their accuracies are more than 70%, their datasets and CNN layers are highly amounted according to the history of medical image processing experience. According to Korolev [4], he states that their algorithms did not need re-construct and were simpler to implement. In medical image analysis, the lack of data is two-fold and more intense: there is generally a lack of available data in public and even less high-quality labelled data. There are

several datasets, and these datasets involve fewer than 100 patients. The other datasets are not allowed for public usage. However, the small training datasets such as ours include only 5,500 chest X-ray images, which is still satisfactory performance in the similar various tasks.

2.1. Problem Statement

Currently, getting the results of chest X-rays is time-consuming and difficult. There is no illness of the patient mostly. Also, this project will reduce the demand for radiologists. X-rays are analysed by human radiologists who are limited by speed, fatigue, and experience. Training a qualified radiologist takes years and requires huge financial costs. A delayed or incorrect diagnosis harms the patient. For this reason, it is id

deal to perform medical image analysis with an automatic, accurate, and efficient machine learning algorithm.

Currently, CNNs are the most researched machine learning algorithms in medical image analysis. This is because CNNs preserve spatial relationships while filtering their input images. As mentioned, spatial relationships are very important in radiology, such as how the edge of a bone meets muscle or where normal lung tissue intersects with cancerous tissue. With medical image analysis, images are examined pixel by pixel and the clearest results are obtained. It would be ideal for detecting a lung tumor on a patient's chest scan and then localizing and separating it from normal tissue and anticipating various treatment options such as chemotherapy or surgery.

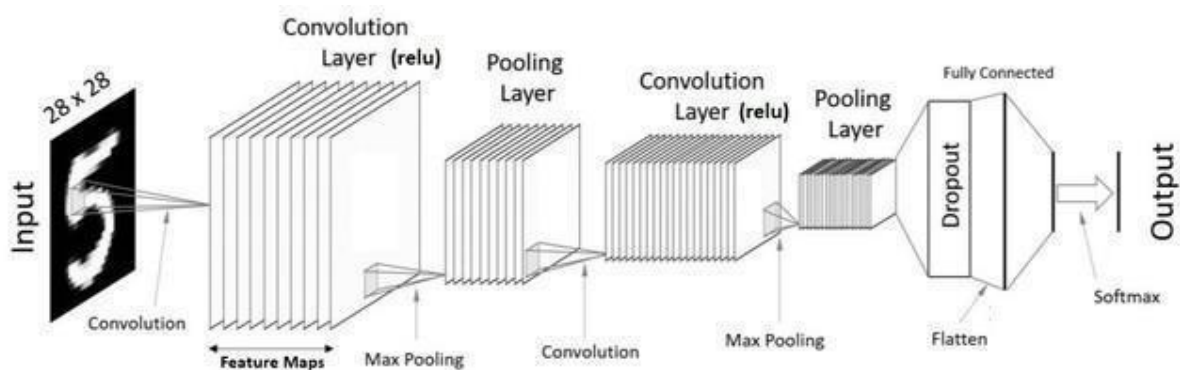
2.2. Motivation

When you go to the hospital, it may take a long time to print out the MRI and chest films. Speeding up this process and checking its accuracy by AI together with the doctor will also facilitate the identification of the disease. Our project aims to increase the reliability and effectiveness of the diagnosis. Creates a list of the 200 most cited articles from Google Scholar in October 2017 using the query terms 'deep learning' and 'medical image analysis'

using citation software. This caused us to turn to the field of medical image analysis when choosing the subject of the project. While doing preliminary research on our project, we examined datasets on the subject. Having many datasets about MRI and chest films made our job easier in terms of accessibility.

3. Literature Review

There is a fair amount of content in medical image processing. The first one we would like to mention is the data we will use. There are lots of anonymous chest x-ray images online. So we believe it won't be challenging to find the dataset in our study. There is also a study that is similar to our study which is P. Rajpurkar's[1] "ChestXNet: Radiologist-level pneumonia detection on chest X-rays with deep learning". In their study, they've used 112,000 images from ChestXray14[3] dataset. ChestXray achieved a magnificent success in classifying the 14 diseases seen on chest X-ray which is breath-taking. They employed a differentiated version of DenseNet[2]. Their scores are also quite satisfying.



Another study in the area is Shen's study[6]. Even though the study was about CT scans, the methods are still similar. They used CNNs combined with Support Vector Machine (SVM) and Random Forest (RF) classifiers to classify lung nodules into benign or malignant, based

on 1010 labeled CT lung scans from the Lung Image Database Consortium (LIDC-IDRI) dataset. Shen used 3 parallel Convolutional Neural Networks with 2 convolution layers each, with each convolutional neural network taking image patches at different scales to extract output. CNN is basically a way of comparing images. From this comparison, the system can generate output for specific features.

Another study in the area is Shen's study[4]. Even though the study was about CT scans, the methods are still similar. They used CNNs combined with Support Vector Machine (SVM) and Random Forest (RF) classifiers to classify lung nodules into benign or malignant, based on

1010 labeled CT lung scans from the Lung Image Database Consortium (LIDC-IDRI) dataset. Shen used 3 parallel Convolutional Neural Networks with 2 convolution layers each, with each convolutional neural network taking image patches at different scales to extract output. CNN is basically a way of comparing images. From this comparison, system can generate output for specific features.

4. Methodology

4.1 Convenient Data Research

The first step of the project is to find the right datasets. There are plenty of free-to-use datasets on the internet but finding one is a problem. At this stage, we pin the datasets with their classes such as "X-ray Films with pneumonia" etc. But we can't be sure of it now. Whether the datasets are worth using or not will be clear once we run the algorithm.

After getting the data, we need a module to create random data from existing data. The reason for random data creation is to expand total data. We have used Tensor Flow's ImageDataGenerator function to do it.

```
""" DATA AUGMENTATION """

train_datagen = ImageDataGenerator(rescale = 1.0 / 255.0,
                                   zoom_range = 0.4,
                                   validation_split = 0.2)

valid_datagen = ImageDataGenerator(rescale = 1.0 / 255.0,
                                   validation_split = 0.2)

test_datagen = ImageDataGenerator(rescale = 1.0 / 255.0)

✓ 0.5s

train_dataset = train_datagen.flow_from_directory(directory = 'C:/Users/GuvenSari/Desktop/AIXRAY/VSCODE/Data/input/train',
                                                  target_size = (224,224),
                                                  class_mode = 'binary',
                                                  subset = 'training',
                                                  batch_size = 64)

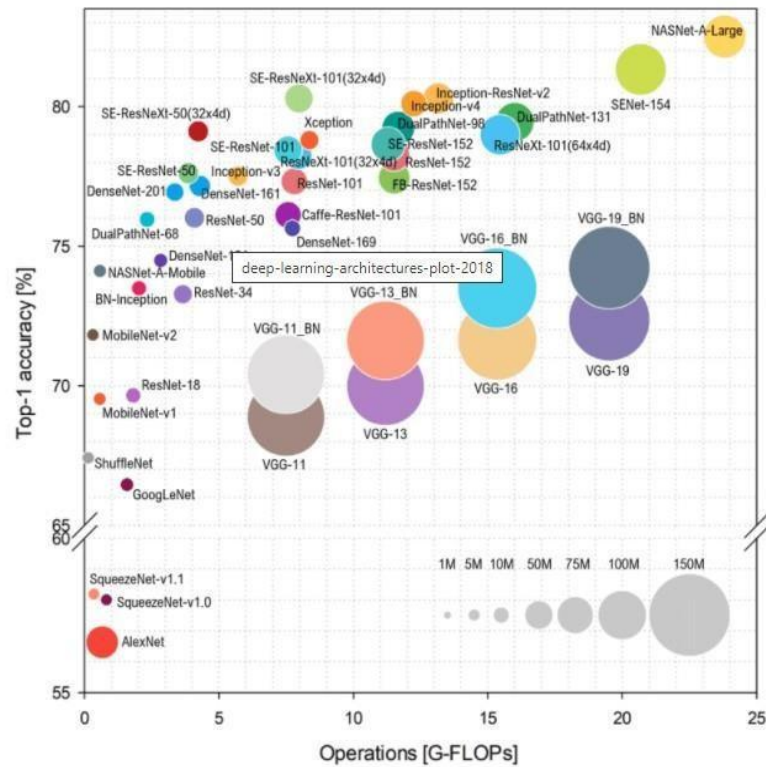
Found 4173 images belonging to 2 classes.
```

The parameter's inside the function means

1. Rescale: Multiplying every pixel by $1/255$ in order to reduce the CPU's workload with high numbers.
2. Zoom_range: This means randomly zooming in and out pictures(keeping the 224x224 size) to create more data.
3. validation_split: Splitting the data to create validation data.

4.2 Researching CNN Architectures

There are plenty of CNN architectures on the internet as can be seen in the figure. It will be clear which one we are going to use after we test the dataset on each architecture. Each CNN architecture has different accuracy, AUC, and ROC values. According to these results, we will pick the most promising architecture.



4.3 Model Decision

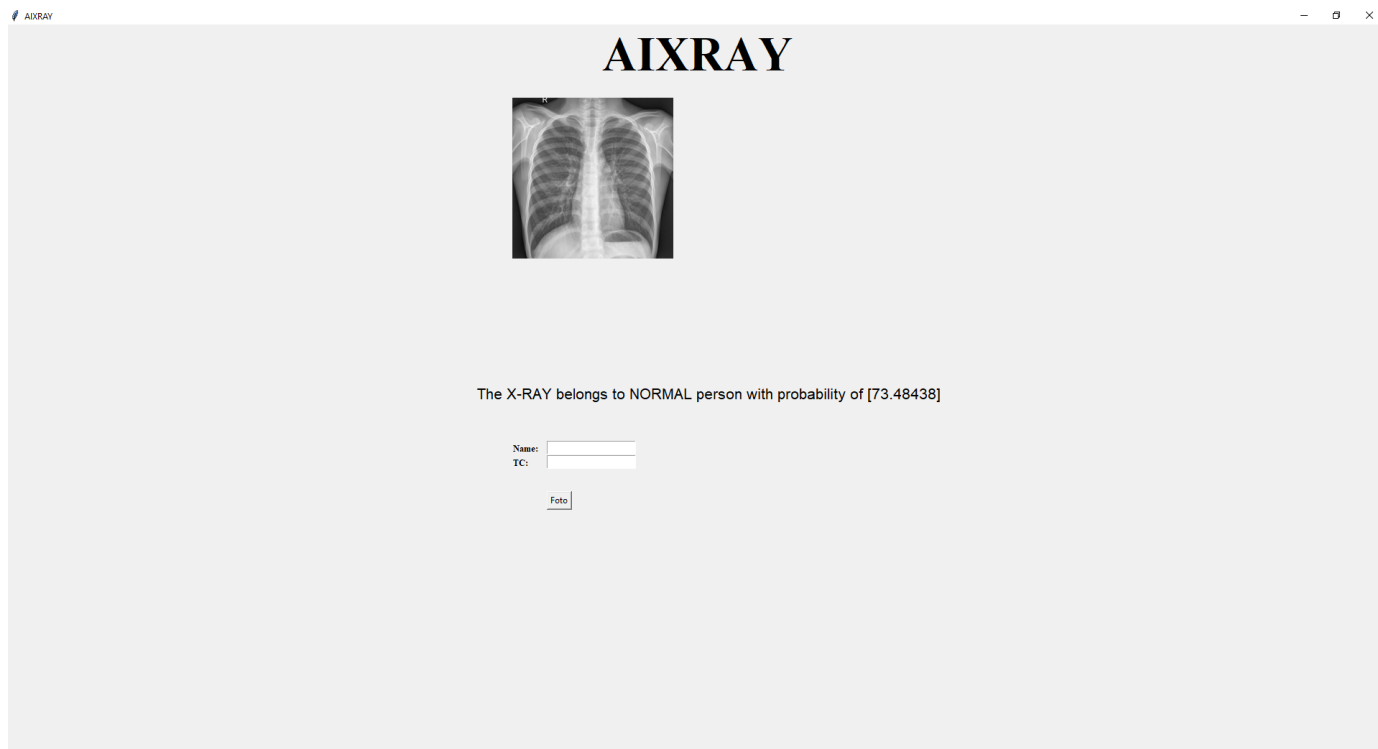
After researching and deciding which architecture to use, we will start to train and implement our model with OpenCV, TensorFlow, Keras, Numpy, Pandas, Matplotlib, and Tkinter, depending on the efficiency of the framework. Statistics at 4.2(Researching CNN architectures), can give us the most effective framework table, but we should consider that it will differ according to the X-ray images we will process. Our dataset, it includes almost 5000 images, and we want to divide it by 75% for training and %25 for testing. In another case of the model, we will try it with 80% for training, %20 for testing and by comparing them we will find out the efficient model.

After the general testing, we will make further tests that contain validation parameters in the order of %50 train %25 validation, and %25 test data. After all testing processes, we will make our decision based on the accuracy and speed of the model.

Model name	Number of parameters [Millions]	ImageNet Top 1 Accuracy	Year
AlexNet	60 M	63.3 %	2012
Inception V1	5 M	69.8 %	2014
VGG 16	138 M	74.4 %	2014
VGG 19	144 M	74.5 %	2014
Inception V2	11.2 M	74.8 %	2015
ResNet-50	26 M	77.15 %	2015
ResNet-152	60 M	78.57 %	2015
Inception V3	27 M	78.8 %	2015
DenseNet-121	8 M	74.98 %	2016
DenseNet-264	22M	77.85 %	2016
BiT-L (ResNet)	928 M	87.54 %	2019
NoisyStudent EfficientNet-L2	480 M	88.4 %	2020
Meta Pseudo Labels	480 M	90.2 %	2021

4.4 Interface Design

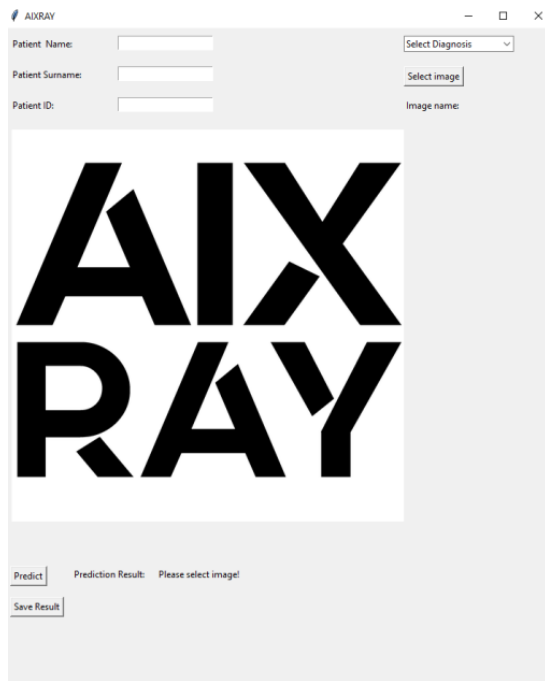
Interface design had been done in Python. First, we have searched for the necessary Python libraries for interface design. The library we used was “Tkinter”. At first, we started with the basics. Placing a button, opening and closing the file, browsing the computer for images, etc. What we have in 2 days we're looking like the picture below.



After further research about the user interfaces led us to create a logo for the program and we designed the logo below.

The logo consists of the words "AIX" and "RAY" stacked vertically. Both words are rendered in a very bold, black, sans-serif typeface. The letters are thick and blocky, with a slightly irregular, hand-drawn quality to the strokes. The "AIX" is positioned directly above the "RAY", and they are centered horizontally relative to each other.

Since the interface is about user experience, we have designed the interface in a way that even a child can run the program and get the result. After getting the result, now we have to save it and use it in order to use the data for further improvements. We have saved the results in “.txt” format. But this format is not very useful. But saving data is done in “.json” format in these projects. The reason for that is to read data from a server for a website or web application to display — and change data given the correct permissions. But, that is not the only thing it is used for. Computer applications, programs, mobile apps, and much more all use JSON files. The last version of the program looks like the picture below.



4.4 Model Modification and Training - Testing Process

The selection of the model depends on the numbers mentioned in Section 4.2. The modification of the model consists of labeling diseases and adding or extracting layers. Selected Model trained to be tested.

5. Conclusions

At the end of this project, it will still be hard to diagnose the disease just by considering the X-Ray. So our aim is to assist the doctor at the moment. But with further research and including the blood test results, may allow the program to diagnose more accurately. Even at that point, it would be still hard to leave all responsibility to the program. So doctors cannot be left out of the diagnosis process.

Also, the application's disease capability can be increased with more data. In time, it is possible to add new diseases to the application. Increasing the range of diseases that the program defines will reduce the accuracy. Finding the right number of diseases will be a new challenge.

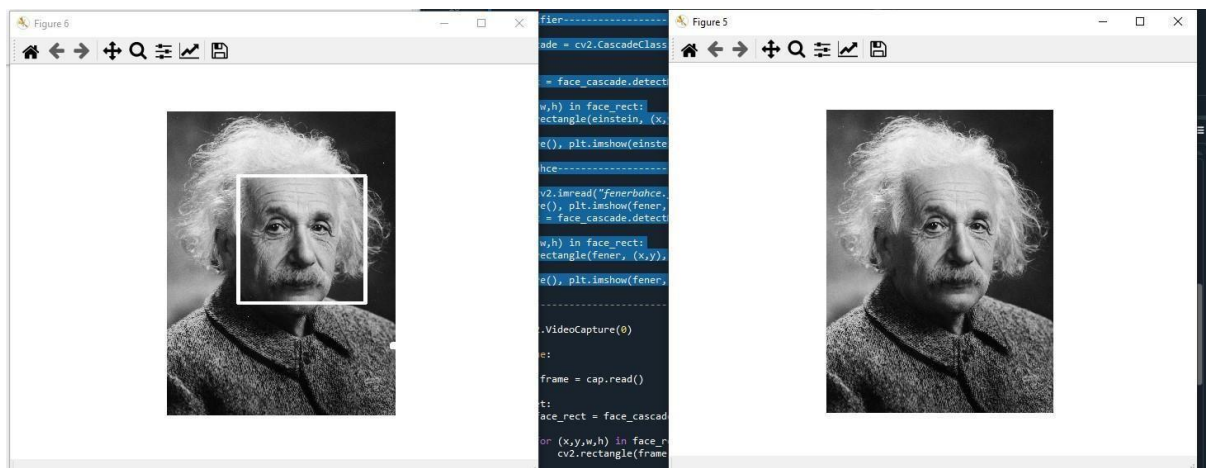
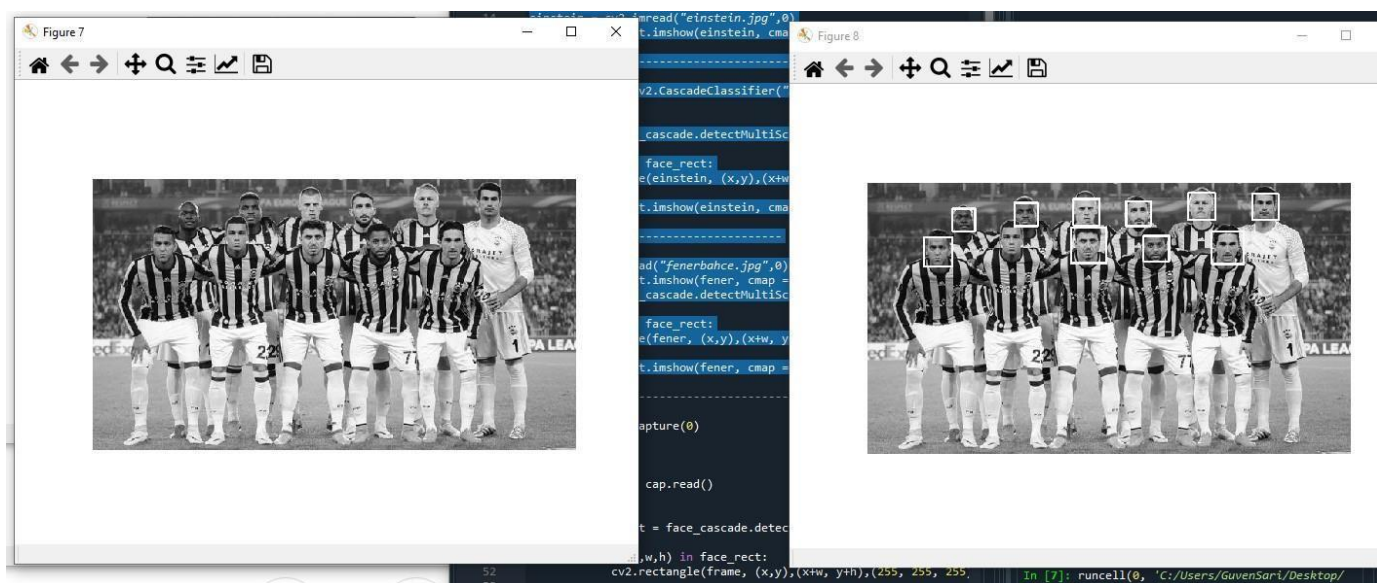
6. REFERENCES

- [1] P. Rajpurkar et al. (Dec. 2017). "CheXNet: Radiologist-level pneumonia detection on chest X-rays with deep learning"
- [2] X. Wang, Y. Peng, L. Lu, Z. Lu, M. Bagheri, and R. M. Summers. (Dec. 2017). "ChestXray8: Hospital-scale chest X-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases."
- [3] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. (Aug. 2016). "Densely connected convolutional networks"
- [4] Shen, M. Zhou, F. Yang, C. Yang, and J. Tian, "Multi-scale convolutional neural networks for lung nodule classification,"

7. Appendix

Face Detection Sample:

At the beginning of the image processing, we want to gain some basic experiences. So we try to implement how to detect faces using the OpenCV library.

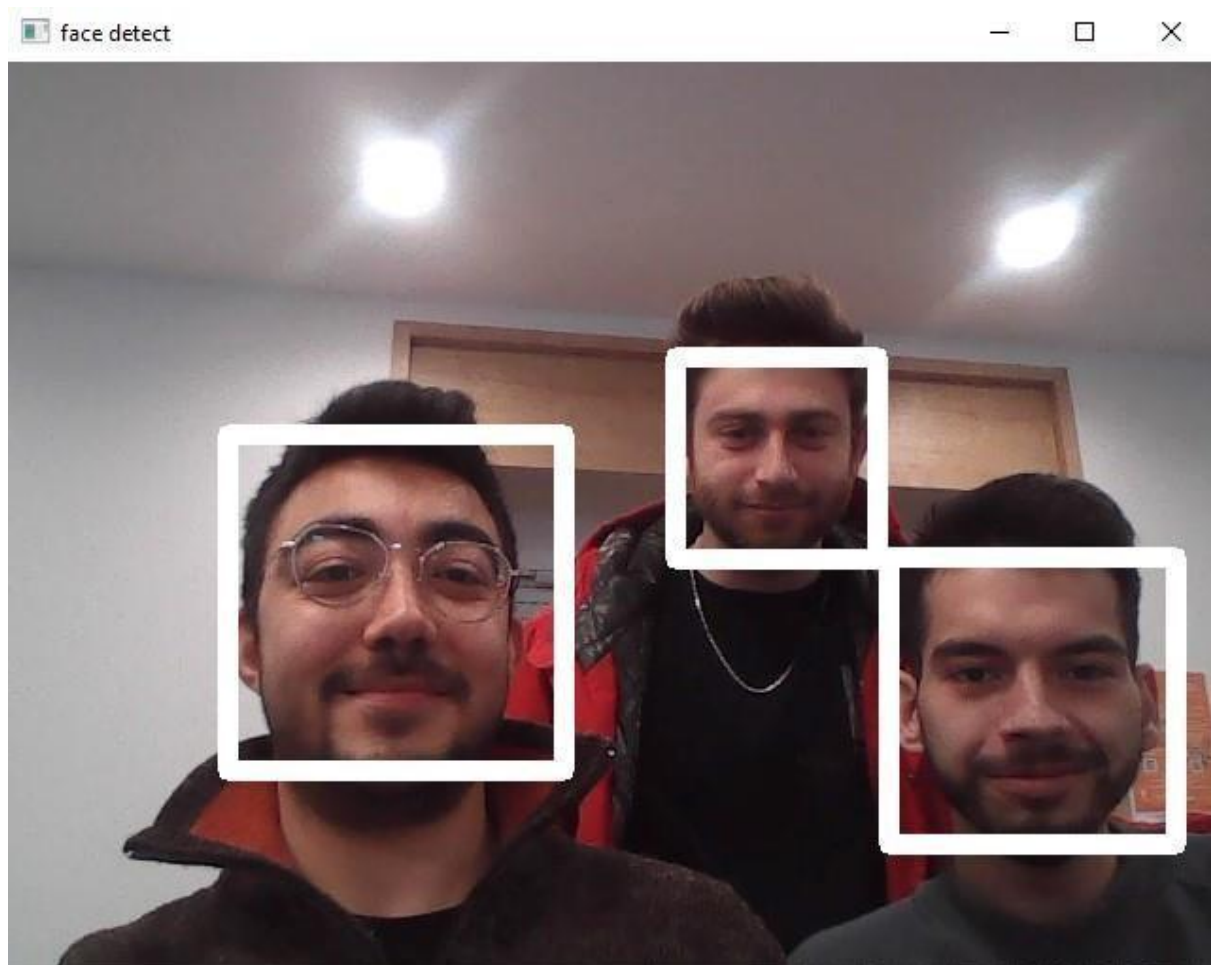


```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Sun Jan 23 14:59:21 2022
4
5  @author: GuvenSari
6  """
7
8
9  import cv2
10 import matplotlib.pyplot as plt
11
12 # import-----
13
14 einstein = cv2.imread("einstein.jpg",0)
15 plt.figure(), plt.imshow(einstein, cmap = "gray"), plt.axis("off")
16
17 ## classifier-----
18
19 face_cascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
20
21
22 face_rect = face_cascade.detectMultiScale(einstein, minNeighbors = 8)
23
24 for (x,y,w,h) in face_rect:
25     cv2.rectangle(einstein, (x,y),(x+w, y+h),(255, 255, 255), 50)
26
27 plt.figure(), plt.imshow(einstein, cmap = "gray"), plt.axis("off")
28
29 # fenerbahce-----
30
31 fener = cv2.imread("fenerbahce.jpg",0)
32 plt.figure(), plt.imshow(fener, cmap = "gray"), plt.axis("off")
33 face_rect = face_cascade.detectMultiScale(fener, minNeighbors = 8)
34
35 for (x,y,w,h) in face_rect:
36     cv2.rectangle(fener, (x,y),(x+w, y+h),(255, 255, 255), 2)
37
38 plt.figure(), plt.imshow(fener, cmap = "gray"), plt.axis("off")
39

```

Video Capture Sample: After we detected faces on some images, also we try to implement it on video



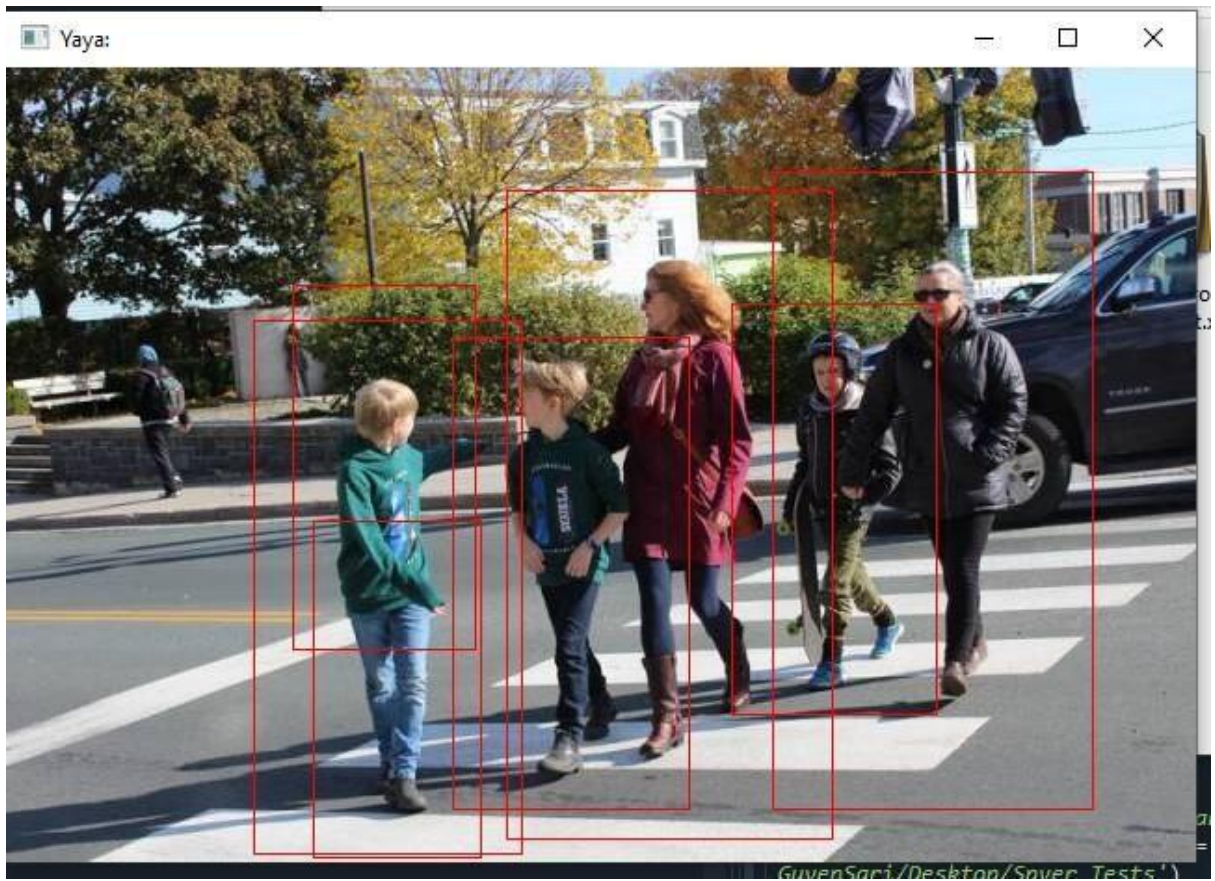

```

40 # video -----
41
42 cap = cv2.VideoCapture(0)
43
44 while True:
45     ret, frame = cap.read()
46
47     if ret:
48         face_rect = face_cascade.detectMultiScale(frame, minNeigh
49
50
51         for (x,y,w,h) in face_rect:
52             cv2.rectangle(frame, (x,y),(x+w, y+h),(255, 255, 255)
53
54             cv2.imshow("face detect", frame)
55
56             if cv2.waitKey(1) & 0xFF == ord("q"): break
57
58 cap.release()
59 cv2.destroyAllWindows()
60
61
62
63
64

```

Pedestrian Detection Sample:

To expand our understanding of image processing we've used a small program that detects pedestrians.

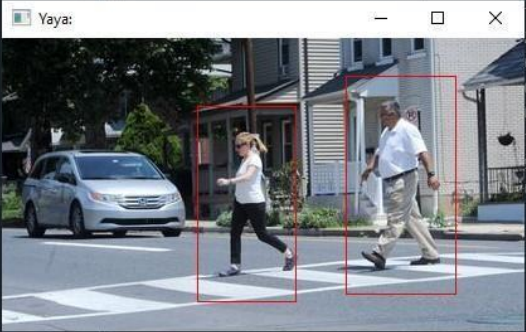


C:\Users\GuvenSari\Desktop\Spyer_Tests\yayaTespit.py

temp.py x unutil1.py x yayaTespit.py x

```
1  #-*- coding: utf-8 -*-
2  """
3  Created on Mon Jan 24 15:44:31 2022
4
5  @author: GuvenSari
6  """
7
8  import cv2
9  import os
10
11
12  files = os.listdir()
13  img_path_list = []
14
15
16  for f in files:
17      if f.endswith(".jpg"):
18          img_path_list.append(f)
19
20
21  print(img_path_list)
22
23  # hog descriptor
24
25  hog = cv2.HOGDescriptor()
26
27  # adding SVM to descriptor.
28  hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())
29  for imagePath in img_path_list:
30      print(imagePath)
31
32      image = cv2.imread(imagePath)
33
34      (rects, weights) = hog.detectMultiScale(image, padding = (8,8),
35
36
37      for(x,y,w,h) in rects:
38          cv2.rectangle(image, (x,y),(x+w,y+h),(0,0,255),1)
39
40
41  cv2.imshow("Yaya: ",image)
42  if cv2.waitKey(0) & 0xFF == ord("q"): continue
```

Yaya:



Variable explorer Help Plots Files

Console 1/A x

```
img2.jpg , img3.jpg
einstein.jpg
fenerbahce.jpg
img1.jpg
img2.jpg
img3.jpg

In [4]: runfile('C:/Users/GuvenSari/Desktop/
Spyer_Tests/yayaTespit.py', wdir='C:/Users/
GuvenSari/Desktop/Spyer_Tests')
['einstein.jpg', 'fenerbahce.jpg', 'img1.jpg',
'img2.jpg', 'img3.jpg']
einstein.jpg
fenerbahce.jpg
img1.jpg
img2.jpg
img3.jpg

In [5]: runfile('C:/Users/GuvenSari/Desktop/
Spyer_Tests/yayaTespit.py', wdir='C:/Users/
GuvenSari/Desktop/Spyer_Tests')
['einstein.jpg', 'fenerbahce.jpg', 'img1.jpg',
'img2.jpg', 'img3.jpg']
einstein.jpg
fenerbahce.jpg
img1.jpg
img2.jpg
```

Model Training and Results

1. CNN_Sequential_Model Pneumonia (Sequential_model_1_224x224x3input.h5):

SEQUENTIAL MODEL : %73 ACCURACY -> MODEL.SUMMARY()

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 222, 222, 32)	896
max_pooling2d (MaxPooling2D)	(None, 111, 111, 32)	0
conv2d_1 (Conv2D)	(None, 109, 109, 32)	9248
max_pooling2d_1 (MaxPooling 2D)	(None, 54, 54, 32)	0
flatten (Flatten)	(None, 93312)	0
dense (Dense)	(None, 128)	11944064
dense_1 (Dense)	(None, 1)	129

Total params: 11,954,337

Trainable params: 11,954,337

Non-trainable params: 0

2. CNN with Transfer Learning - VGG16: MODEL.SUMMARY ->

Model: "VGG16"

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 7, 7, 512)	14714688
dropout (Dropout)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
batch_normalization (BatchNormalization)	(None, 25088)	100352
dense (Dense)	(None, 1024)	25691136
batch_normalization_1 (BatchNormalization)	(None, 1024)	4096
activation (Activation)	(None, 1024)	0
dropout_1 (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 1024)	1049600
batch_normalization_2 (Batc	(None, 1024)	4096
...		
Total params: 41,564,993		
Trainable params: 26,796,033		
Non-trainable params: 14,768,960		

```
# Model Fitting

model_history=transferlearning_model.fit(train_dataset,
                                         validation_data=valid_dataset,
                                         epochs = 3,
                                         callbacks = callback_list,
                                         verbose = 1)

[25] ✓ 29m 57.1s

... Epoch 1/3
66/66 [=====] - ETA: 0s - loss: 0.2356 - auc: 0.9641
Epoch 1: val_auc improved from -inf to 1.00000, saving model to .\best_weights.hdf5
66/66 [=====] - 730s 11s/step - loss: 0.2356 - auc: 0.9641 - val_loss: 1.2516 - val_auc: 1.0000
Epoch 2/3
66/66 [=====] - ETA: 0s - loss: 0.1344 - auc: 0.9842
Epoch 2: val_auc did not improve from 1.00000
66/66 [=====] - 556s 8s/step - loss: 0.1344 - auc: 0.9842 - val_loss: 0.8235 - val_auc: 1.0000
Epoch 3/3
66/66 [=====] - ETA: 0s - loss: 0.1188 - auc: 0.9879
Epoch 3: val_auc did not improve from 1.00000
66/66 [=====] - 509s 8s/step - loss: 0.1188 - auc: 0.9879 - val_loss: 1.4807 - val_auc: 1.0000
```

Epoch 1/3

163/163 [=====] - ETA: 0s - loss: 0.4225 - accuracy:

0.8083WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least `steps_per_epoch * epochs` batches (in this case, 100 batches). You may need to use the repeat() function when building your dataset.

163/163 [=====] - 46s 279ms/step - loss: 0.4225 - accuracy:

0.8083 - val_loss: 0.6712 - val_accuracy: 0.6875

Epoch 2/3

163/163 [=====] - 44s 271ms/step - loss: 0.2495 - accuracy:

0.8988

Epoch 3/3

163/163 [=====] - 50s 308ms/step - loss: 0.2544 - accuracy:

0.8988

78/100 [=====>.....] - ETA: 2s - loss: 0.7401 - accuracy:

0.7372WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least `steps_per_epoch * epochs` batches (in this case, 100 batches). You may need to use the repeat() function when building your dataset.

100/100 [=====] - 7s 73ms/step - loss: 0.7401 - accuracy: 0.7372

The testing accuracy is : 73.71794581413269 %

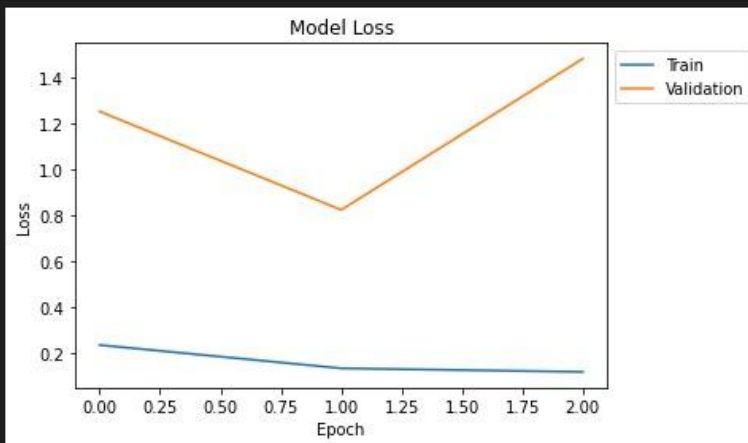
PS C:\Users\GuvenSari\Desktop\AIXRAY\VSCODE>

- **MODEL LOSS AND AUC**

```
# Summarize the model loss

plt.plot(model_history.history['loss'])
plt.plot(model_history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left', bbox_to_anchor=(1,1))
plt.show()
```

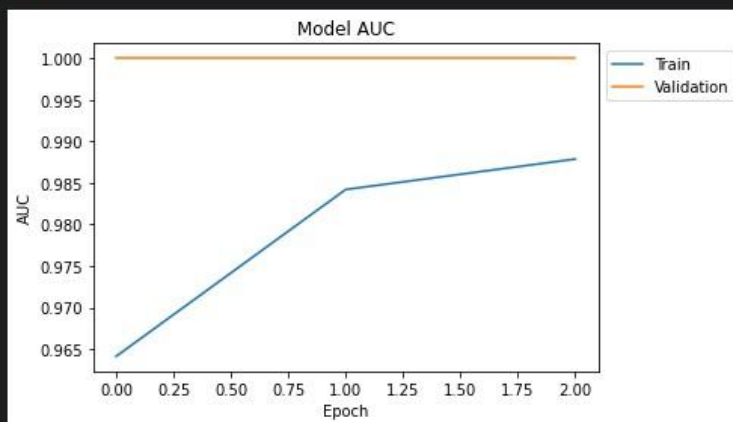
[26] ✓ 0.2s



```
# Summarize models auc

plt.plot(model_history.history['auc'])
plt.plot(model_history.history['val_auc'])
plt.title('Model AUC')
plt.ylabel('AUC')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left', bbox_to_anchor=(1,1))
plt.show()
```

[27] ✓ 0.2s



• TEST RESULTS :

```
# Test Data

test_dataset = test_datagen.flow_from_directory(directory = 'C:/Users/GuvenSari/Desktop/AIXRAY/VSCODE/Data/input/test',
                                                target_size = (224,224),
                                                class_mode = 'binary',
                                                batch_size = 64)

[28] ✓ 0.9s

... Found 624 images belonging to 2 classes.

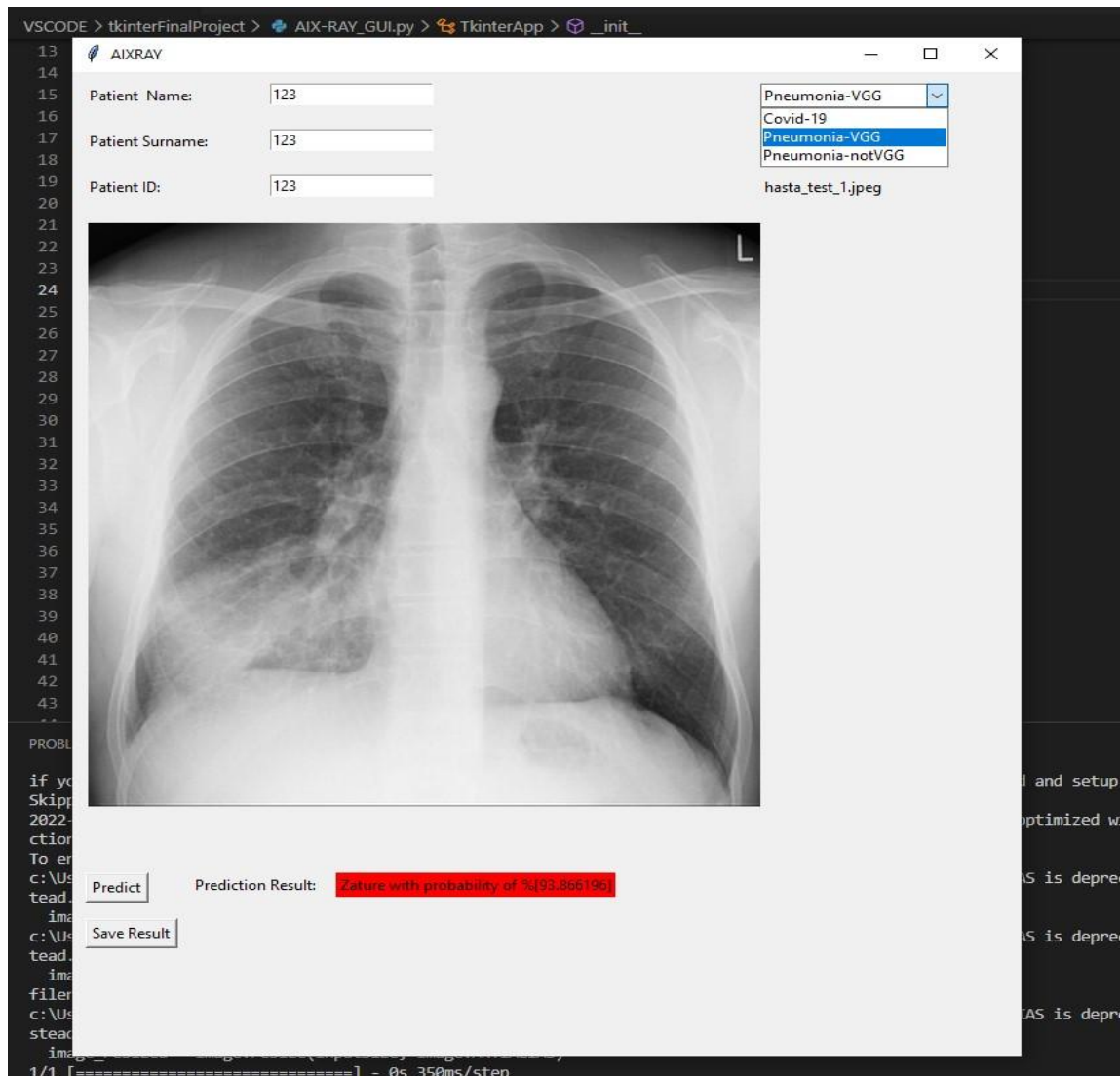
# Evaluating Loss and AUC - Test Data

transferlearning_model.evaluate(test_dataset)

[31] ✓ 1m 9.4s

... 10/10 [=====] - 69s 7s/step - loss: 0.3660 - auc: 0.9528

[0.3659727871417999, 0.9527503848075867]
```



3. Covid Model With Transfer Learning VGG16 :

Model.summary() is the same results with the 2.model.

Epoch results :

```
# Model Fitting

model_history=model.fit(train_dataset,
                        validation_data=valid_dataset,
                        epochs = 5,
                        callbacks = callback_list,
                        verbose = 1)

✓ 184m 24.6s

Epoch 1/5
172/172 [=====] - ETA: 0s - loss: 0.2876 - auc: 0.9353
Epoch 1: val_auc improved from -inf to 0.98098, saving model to .\best_weights.hdf5
172/172 [=====] - 2199s 13s/step - loss: 0.2876 - auc: 0.9353 - val_loss: 0.1914 - val_auc: 0.9810
Epoch 2/5
172/172 [=====] - ETA: 0s - loss: 0.1812 - auc: 0.9721
Epoch 2: val_auc improved from 0.98098 to 0.98957, saving model to .\best_weights.hdf5
172/172 [=====] - 2171s 13s/step - loss: 0.1812 - auc: 0.9721 - val_loss: 0.1107 - val_auc: 0.9896
Epoch 3/5
172/172 [=====] - ETA: 0s - loss: 0.1584 - auc: 0.9784
Epoch 3: val_auc did not improve from 0.98957
172/172 [=====] - 2263s 13s/step - loss: 0.1584 - auc: 0.9784 - val_loss: 0.1214 - val_auc: 0.9888
Epoch 4/5
172/172 [=====] - ETA: 0s - loss: 0.1387 - auc: 0.9828
Epoch 4: val_auc did not improve from 0.98957
172/172 [=====] - 2218s 13s/step - loss: 0.1387 - auc: 0.9828 - val_loss: 0.1368 - val_auc: 0.9877
Epoch 5/5
172/172 [=====] - ETA: 0s - loss: 0.1238 - auc: 0.9867
Epoch 5: val_auc did not improve from 0.98957
172/172 [=====] - 2213s 13s/step - loss: 0.1238 - auc: 0.9867 - val_loss: 0.1359 - val_auc: 0.9827
Epoch 5: early stopping
```

Model's Area Under Curve:



Model's Loss:

