Face Mask Detection using Deep Learning

MScResearchProject:ConfigurationManual

DataAnalytics

SagarSagar

StudentID:x18134246

SchoolofComputing

NationalCollegeofIreland

Supervisor:

Dr.VladimirMilosavljevic



**National College of Ireland**

**Project Submission Sheet**

**School of Computing**

|  |  |
| --- | --- |
| **Student Name:** | Jaydeep Deka |
| **Student ID:** | x18134246 |
| **Programme:** | Data Analytics |
| **Year:** | 2019 |
| **Module:** | MSc Research Project: Configuration Manual |
| **Supervisor:** | Dr. Vladimir Milosavljevic |
| **Submission Due Date:** | 12/12/2019 |
| **Project Title:** | Image Captioning: Capsule Network vs CNN approach |
| **Word Count:** | 1261 |
| **Page Count:** | 13 |

|  |  |
| --- | --- |
| **Office Use Only** | |
| Signature: |  |
| Date: |  |
| Penalty Applied (if applicable): |  |

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

**ALL** internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author’s written or electronic work is illegal (plagiarism) and may result in disciplinary action.

|  |  |
| --- | --- |
| **Signature:** |  |
| **Date:** | 11th December 2019 |

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:**

|  |  |
| --- | --- |
| Attach a completed copy of this sheet to each project (including multiple copies). |  |
| **Attach a Moodle submission receipt of the online project submission**, to each project (including multiple copies). |  |
| **You must ensure that you retain a HARD COPY of the project**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. |  |

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

# Configuration Manual

Face Mask Detection Using Deep Learning

Sagar XXYYZZ123

## 1 Introduction

This manual will act as a supporting document for the project ‘<Face mask detection>’. This manual will cover detail about the hardware setup, software installation, downloading dataset, running and inferencing the code. All the codes can be found under the *github link.* Datasets, trained models can be downloaded from the following *links*

## 2 Hardware requirements

This project is partially developed on a high performance laptop and high performance cloud, details of which are mentioned below

High performance workstation laptop:

* HP Zbook 15 G3
* Intel Xeon Processor
* 1 Tb HDD, 32GiB RAM
* Nvidia Quadro M2000

High performance Cloud:

* Google Colab-pro / Azure Cognitive Services Cloud
* Nvidia Tesla P6

## 3 Software requirements

* Anaconda IDE – Python 3.7
* Visual Studio Community edition 2015
* NVIDIA Driver, NVIDIA GPU TOOLKIT- CUDA
* PyTorch, Numpy, OpenCV

## 4 Environment Setup

It is always advised to create a virtual environment for a new project. Following steps should be followed to do so:

1. >>conda create –-name facemask python==3.7
2. >>pip install –-r requirement.txt

## 5 Data Collection

There are two datasets used in this project which should be downloaded from link mentioned below. Post downloading those datasets should be put in */data* folder

* Wider Face – [Link 1](http://shuoyang1213.me/WIDERFACE/)
* Real World Face Masked Dataset – [Link 2](https://github.com/X-zhangyang/Real-World-Masked-Face-Dataset)

## 6 Implementation

The project has been planned and executed in an order. While reproducing the solution it’s expected from the developer to follow the same order unless developer is read to make changes in the code

FaceMaskDetection

| convert\_to\_onnx.py

| detect.py

| folder\_tree.doc

| pytorch\_infer.py

| test\_fddb.py

| test\_widerface.py

| train.py

|

+---curve

| FDDB.png

| Widerface.jpg

|

+---data

| | config.py

| | data\_augment.py

| | wider\_face.py

| | \_\_init\_\_.py

+---img

| 0071.jpg

| 0089.jpg

| 0147.jpg

| 0249.jpg

+---layers

| | \_\_init\_\_.py

| |

| +---functions

| | prior\_box.py

| |

| \---modules

| multibox\_loss.py

| \_\_init\_\_.py

|

+---load\_model

| | pytorch\_loader.py

| | \_\_init\_\_.py

+---models

| | MainModel.py

| | model360.pth

| | net.py

| | retinaface.py

| | \_\_init\_\_.py

+---output

| 245.jpg

+---utils

| | anchor\_decode.py

| | anchor\_generator.py

| | box\_utils.py

| | nms\_.py

| | timer.py

| | \_\_init\_\_.py

| |

| +---nms

| | | py\_cpu\_nms.py

| | | \_\_init\_\_.py

+---weights

| mobilenet0.25\_Final.pth

| mobilenetV1X0.25\_pretrain.tar

| Resnet50\_Final.pth

\---widerface\_evaluate

| box\_overlaps.pyx

| evaluation.py

| README.md

| setup.py

\---ground\_truth

wider\_easy\_val.mat

wider\_face\_val.mat

wider\_hard\_val.mat

wider\_medium\_val.mat

## 7. Modular Documentation

### 7.1 Inference Module : Prediction module using deep learning

#### **Facemask\_predict.py**

### Codeflow Diagram :

### Example for calling this code from command line:

*For single image*

>> python facemask\_predict.py --img-path img/10111.jpg

*For using webcam*

>> python facemask\_predict.py --img-mode 0 --video-path 0

*For running analysis on a video*

>> python facemask\_predict.py --img-mode 0 --video-path img\videoplayback.mp4

##### **Detailed Explanation**

1. # -\*- coding:utf-8 -\*-
3. **#Python built in libraries**
4. **import** time
5. **import** argparse
7. **#Python available libraries**
8. **import** cv2
9. **import** numpy as np
10. **from** PIL **import** Image
11. **from** load\_model.pytorch\_loader **import** load\_pytorch\_model, pytorch\_inference
13. **#Custom made modules**
14. **from** utils.anchor\_generator **import** generate\_anchors
15. **from** utils.anchor\_decode **import** decode\_bbox
16. **from** utils.nms\_ **import** single\_class\_non\_max\_suppression
18. **#Loading model from the directory**
19. model = load\_pytorch\_model('models/model360.pth');
21. **# anchor configurations**
22. feature\_map\_sizes = [[45, 45], [23, 23], [12, 12], [6, 6], [4, 4]]
24. **#anchor box sizes, has to be find out using Regional Proposal Networks**
25. anchor\_sizes = [[0.04, 0.056], [0.08, 0.11], [0.16, 0.22], [0.32, 0.45], [0.64, 0.72]]
26. anchor\_ratios = [[1, 0.62, 0.42]] \* 5
28. **# generate anchors**
29. anchors = generate\_anchors(feature\_map\_sizes, anchor\_sizes, anchor\_ratios)
31. **# for inference , the batch size is 1, the model output shape is [1, N, 4],**
32. **# so we expand dim for anchors to [1, anchor\_num, 4]**
33. anchors\_exp = np.expand\_dims(anchors, axis=0)
35. id2class = {0: 'Mask', 1: 'NoMask'}

38. **def** inference(image,
39. conf\_thresh=0.5,
40. iou\_thresh=0.4,
41. target\_shape=(160, 160),
42. draw\_result=True,
43. show\_result=True
44. ):
45. '''''
46. **Main function of detection inference**
47. **param image       : 3D numpy array of image**
48. **param conf\_thresh : the min threshold of classification probabity.**
49. **param iou\_thresh  : the IOU threshold of NMS**
50. **param target\_shape: the model input size.**
51. **param draw\_result : whether to daw bounding box to the image.**
52. **param show\_result : whether to display the image.**
54. **return:**
55. **output\_info**
56. **'''**
57. **#Initializing list**
58. output\_info = []
60. **#Obtaining height width of the image frame**
61. height, width, \_ = image.shape
63. **#Resizing image using opencv resize function**
64. image\_resized = cv2.resize(image, target\_shape)
66. **#Normalizing the image**
67. image\_np = image\_resized / 255.0
69. **#Flatten the image, converted into a array**
70. image\_exp = np.expand\_dims(image\_np, axis=0)
72. image\_transposed = image\_exp.transpose((0, 3, 1, 2))
74. **#Calling our trained deep learning model for predictions**
75. y\_bboxes\_output, y\_cls\_output = pytorch\_inference(model, image\_transposed)
77. **# remove the batch dimension, for batch is always 1 for inference.**
78. y\_bboxes = decode\_bbox(anchors\_exp, y\_bboxes\_output)[0]
79. y\_cls = y\_cls\_output[0]
81. **# To speed up, do single class NMS, not multiple classes NMS.**
82. bbox\_max\_scores = np.max(y\_cls, axis=1)
83. bbox\_max\_score\_classes = np.argmax(y\_cls, axis=1)
85. **# keep\_idx is the alive bounding box after nms.**
87. keep\_idxs = single\_class\_non\_max\_suppression(y\_bboxes,
88. bbox\_max\_scores,
89. conf\_thresh=conf\_thresh,
90. iou\_thresh=iou\_thresh,
91. )
92. **#Iterating through each predictions**
93. **for** idx **in** keep\_idxs:
94. conf = float(bbox\_max\_scores[idx])
95. class\_id = bbox\_max\_score\_classes[idx]
96. bbox = y\_bboxes[idx]
97. **# clip the coordinate, avoid the value exceed the image boundary**.
98. xmin = max(0, int(bbox[0] \* width))
99. ymin = max(0, int(bbox[1] \* height))
100. xmax = min(int(bbox[2] \* width), width)
101. ymax = min(int(bbox[3] \* height), height)
103. **#Drawing bounding boxes**
104. **if** draw\_result:
105. **if** class\_id == 0:
106. color = (0, 255, 0)
107. **else**:
108. color = (255, 0, 0)
109. cv2.rectangle(image, (xmin, ymin), (xmax, ymax), color, 2)
110. cv2.putText(image, "%s: %.2f" % (id2class[class\_id], conf), (xmin + 2, ymin - 2),
111. cv2.FONT\_HERSHEY\_SIMPLEX, 0.8, color)
112. output\_info.append([class\_id, conf, xmin, ymin, xmax, ymax])
114. **if** show\_result:
115. Image.fromarray(image).show()
116. **## Incase you want to save the image in output folder**
117. **#Image.fromarray(image).save('output/%d.jpg'%(np.random.random()\*1000))**
119. **return** output\_info

122. **def** run\_on\_video(video\_path, output\_video\_name, conf\_thresh):
124. **'''''**
125. **Main function of detection inference on video files**
126. **video\_path : Path of video file, should be relate to master folder**
127. **output\_video\_name : required name for output video**
128. **conf\_thresh : the min threshold of classification probabity.**
130. **return:**
131. **output\_info**
132. **'''**
134. cap = cv2.VideoCapture(video\_path)
135. height = cap.get(cv2.CAP\_PROP\_FRAME\_HEIGHT)
136. width = cap.get(cv2.CAP\_PROP\_FRAME\_WIDTH)
137. fps = cap.get(cv2.CAP\_PROP\_FPS)
138. fourcc = cv2.VideoWriter\_fourcc(\*'XVID')
139. # writer = cv2.VideoWriter(output\_video\_name, fourcc, int(fps), (int(width), int(height)))
140. total\_frames = cap.get(cv2.CAP\_PROP\_FRAME\_COUNT)
141. **if** **not** cap.isOpened():
142. **raise** ValueError("Video open failed.")
143. **return**
144. status = True
145. idx = 0
146. **while** status:
147. start\_stamp = time.time()
148. status, img\_raw = cap.read()
149. img\_raw = cv2.cvtColor(img\_raw, cv2.COLOR\_BGR2RGB)
150. read\_frame\_stamp = time.time()
151. **if** (status):
152. inference(img\_raw,
153. target\_shape=(360, 360),
154. show\_result=False)
155. cv2.imshow('image', img\_raw[:, :, ::-1])
156. cv2.waitKey(1)
157. inference\_stamp = time.time()
158. # writer.write(img\_raw)
159. write\_frame\_stamp = time.time()
160. idx += 1
161. **print**("%d of %d" % (idx, total\_frames))
162. **print**("read\_frame:%f, infer time:%f, write time:%f" % (read\_frame\_stamp - start\_stamp,
163. inference\_stamp - read\_frame\_stamp,
164. write\_frame\_stamp - inference\_stamp))
165. # writer.release()

168. **if** \_\_name\_\_ == "\_\_main\_\_":
169. parser = argparse.ArgumentParser(description="Face Mask Detection")
170. parser.add\_argument('--img-mode', type=str, default='1', help='set 1 to run on image, 0 to run on video.')
171. parser.add\_argument('--img-path', type=str, help='path to your image.')
172. parser.add\_argument('--video-path', type=str, default='0', help='path to your video, `0` means to use camera.')
174. args = parser.parse\_args()
175. **#If you want to run inference on single image**
176. **if** int(args.img\_mode):
177. imgPath = args.img\_path
178. img = cv2.imread(imgPath)
179. img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)
180. inference(img, show\_result=True, target\_shape=(360, 360))
181. **#If you want to run inference on the whole video**
182. **else**:
183. video\_path = args.video\_path
184. **if** args.video\_path == '0':
185. video\_path = 0
186. run\_on\_video(video\_path, '', conf\_thresh=0.5)

### 7.2 Model : Building pytorch for retinaface

#### **Model.py**

### Codeflow Diagram :

### Face Detectors: Understand DSFD and the State-of-the-art Algorithms

##### **Detailed Explanation**

1. **#Importing python based libraries**
2. **import** torch
3. **import** torch.nn as nn
4. **import** torchvision.models.detection.backbone\_utils as backbone\_utils
5. **import** torchvision.models.\_utils as \_utils
6. **import** torch.nn.functional as F
7. **from** collections **import** OrderedDict
8. **#Importing all the custom made modules**
10. **from** models.net **import** MobileNetV1 as MobileNetV1
11. **from** models.net **import** FPN as FPN
12. **from** models.net **import** SSH as SSH

15. **#Defining python class for neural net module based**
16. **#on pytorch framework**
17. **class** ClassHead(nn.Module):
18. **def** \_\_init\_\_(self,inchannels=512,num\_anchors=3):
19. super(ClassHead,self).\_\_init\_\_()
20. self.num\_anchors = num\_anchors
21. self.conv1x1 = nn.Conv2d(inchannels,self.num\_anchors\*2,kernel\_size=(1,1),stride=1,padding=0)
23. **def** forward(self,x):
24. out = self.conv1x1(x)
25. out = out.permute(0,2,3,1).contiguous()
27. **return** out.view(out.shape[0], -1, 2
28. **#Defining python class for box head class based**
29. **#on pytorch framework. This box will be used for replacing the frontal**
30. **#Head of the retinaface network**
31. **class** BboxHead(nn.Module):
32. **def** \_\_init\_\_(self,inchannels=512,num\_anchors=3):
33. super(BboxHead,self).\_\_init\_\_()
34. self.conv1x1 = nn.Conv2d(inchannels,num\_anchors\*4,kernel\_size=(1,1),stride=1,padding=0)
36. **def** forward(self,x):
37. out = self.conv1x1(x)
38. out = out.permute(0,2,3,1).contiguous()
40. **return** out.view(out.shape[0], -1, 4)
41. **#Defining facial landmark class. This will create face landmark datatype**
43. **class** LandmarkHead(nn.Module):
44. **def** \_\_init\_\_(self,inchannels=512,num\_anchors=3):
45. super(LandmarkHead,self).\_\_init\_\_()
46. self.conv1x1 = nn.Conv2d(inchannels,num\_anchors\*10,kernel\_size=(1,1),stride=1,padding=0)
48. **def** forward(self,x):
49. out = self.conv1x1(x)
50. out = out.permute(0,2,3,1).contiguous()
52. **return** out.view(out.shape[0], -1, 10)
53. **#Defining python class for retina net module based**
54. **#on pytorch framework. This datatype will hold all the information of the defined**
55. **#Neural network framework**
56. **class** RetinaFace(nn.Module):
57. **def** \_\_init\_\_(self, cfg = None, phase = 'train'):
58. """
59. :param cfg:  Network related settings.
60. :param phase: train or test.
61. """
62. super(RetinaFace,self).\_\_init\_\_()
63. self.phase = phase
64. backbone = None
65. **if** cfg['name'] == 'mobilenet0.25':
66. backbone = MobileNetV1()
67. **if** cfg['pretrain']:
68. checkpoint = torch.load("./weights/mobilenetV1X0.25\_pretrain.tar", map\_location=torch.device('cpu'))
69. **from** collections **import** OrderedDict
70. new\_state\_dict = OrderedDict()
71. **for** k, v **in** checkpoint['state\_dict'].items():
72. name = k[7:]  # remove module.
73. new\_state\_dict[name] = v
74. # load params
75. backbone.load\_state\_dict(new\_state\_dict)
76. **elif** cfg['name'] == 'Resnet50':
77. **import** torchvision.models as models
78. backbone = models.resnet50(pretrained=cfg['pretrain'])
80. self.body = \_utils.IntermediateLayerGetter(backbone, cfg['return\_layers'])
81. in\_channels\_stage2 = cfg['in\_channel']
82. in\_channels\_list = [
83. in\_channels\_stage2 \* 2,
84. in\_channels\_stage2 \* 4,
85. in\_channels\_stage2 \* 8,
86. ]
87. out\_channels = cfg['out\_channel']
88. self.fpn = FPN(in\_channels\_list,out\_channels)
89. self.ssh1 = SSH(out\_channels, out\_channels)
90. self.ssh2 = SSH(out\_channels, out\_channels)
91. self.ssh3 = SSH(out\_channels, out\_channels)
93. self.ClassHead = self.\_make\_class\_head(fpn\_num=3, inchannels=cfg['out\_channel'])
94. self.BboxHead = self.\_make\_bbox\_head(fpn\_num=3, inchannels=cfg['out\_channel'])
95. self.LandmarkHead = self.\_make\_landmark\_head(fpn\_num=3, inchannels=cfg['out\_channel'])
97. **def** \_make\_class\_head(self,fpn\_num=3,inchannels=64,anchor\_num=2):
98. classhead = nn.ModuleList()
99. **for** i **in** range(fpn\_num):
100. classhead.append(ClassHead(inchannels,anchor\_num))
101. **return** classhead
103. **def** \_make\_bbox\_head(self,fpn\_num=3,inchannels=64,anchor\_num=2):
104. bboxhead = nn.ModuleList()
105. **for** i **in** range(fpn\_num):
106. bboxhead.append(BboxHead(inchannels,anchor\_num))
107. **return** bboxhead
109. **def** \_make\_landmark\_head(self,fpn\_num=3,inchannels=64,anchor\_num=2):
110. landmarkhead = nn.ModuleList()
111. **for** i **in** range(fpn\_num):
112. landmarkhead.append(LandmarkHead(inchannels,anchor\_num))
113. **return** landmarkhead
115. **def** forward(self,inputs):
116. out = self.body(inputs)
118. # FPN
119. fpn = self.fpn(out)
121. # SSH
122. feature1 = self.ssh1(fpn[0])
123. feature2 = self.ssh2(fpn[1])
124. feature3 = self.ssh3(fpn[2])
125. features = [feature1, feature2, feature3]
127. bbox\_regressions = torch.cat([self.BboxHead[i](feature) **for** i, feature **in** enumerate(features)], dim=1)
128. classifications = torch.cat([self.ClassHead[i](feature) **for** i, feature **in** enumerate(features)],dim=1)
129. ldm\_regressions = torch.cat([self.LandmarkHead[i](feature) **for** i, feature **in** enumerate(features)], dim=1)
131. **if** self.phase == 'train':
132. output = (bbox\_regressions, classifications, ldm\_regressions)
133. **else**:
134. output = (bbox\_regressions, F.softmax(classifications, dim=-1), ldm\_regressions)
135. **return** output

### 7.3 Training code

#### train.py

### Codeflow Diagram :

### Example for calling this code from command line:

##### **Detailed Explanation**