System Source Code

app.py

```
# app/app.py
from gui import root

print("""Project Lazuli Cross
    Project Lazuli Cross: Personal Protective Equipment
Detection and Safety Features Determination through YOLO
Algorithm
    Copyright (C) 2021 Team Lazuli Cross

    This program is free software: you can redistribute it
and/or modify
    it under the terms of the GNU General Public License as
published by
    the Free Software Foundation, either version 3 of the
License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be
useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty
of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public
License
    along with this program.  If not, see
<http://www.gnu.org/licenses/>.""")
while True and root.winfo_ismapped():
    try:
        root.mainloop()
        break
    except Exception as e:
        print("A runtime error occured: {}".format(str(e)))
        root.destroy()
exit()
```

gui.py

```
# app/gui.py
from tkinter import *
from tkinter import filedialog
import os
from PIL import Image, ImageTk
import pathlib as pl
from modules import capture, detect, test


def image():
    fln = filedialog.askopenfilename(initialdir = os.getcwd(),
title="Select Image file", filetypes=(("JPG File","*.jpg"),
("PNG file", "*.png"),("All Files","*.*")))
    img = Image.open(fln)
    img.thumbnail((300,200))
    img = ImageTk.PhotoImage(img)
    lbl.configure(image = img)
    lbl.image = img
    capture(fln)

def function():
    detect()
    test()

root = Tk()
root.title("Lazuli Cross")
root.geometry("300x400")
root.iconbitmap(str(pl.Path(__file__).parent) + "/logo.ico")

f1 = Frame(root, height=400, width=300)
f1.config(background='#32454a')

f1.pack(fill = 'both', expand = True)

f2 = Frame(f1, height = 300, width = 400)
f2.grid()

lbl = Label(f2, padx = 100, pady = 100, bd = 1)
lbl.pack()
```

```
f2.pack(expand = True)

f3 = Frame(f1)
f3.config(background = '#32454a')
f3.pack(expand = True)


b1 = Button(f3, text="Insert Image", bg = '#5a7982', fg =
'white', command = image)
b1.grid(row=0, column=1, padx= 10, pady=10)

b2 = Button(f3, text = "Test Image", bg = '#5a7982', fg =
'white', command = function)
b2.grid(row=0, column=2, padx=10, pady=10)
```

modules.py

```
# app/modules.py
import argparse
import concurrent.futures as fut
from importlib import import_module
from io import open
import os
import pathlib as pl
from PIL import Image
from shutil import rmtree
import sys
from time import time_ns, sleep

print("Resolving configuration.")

imgproc = import_module("utility.image_process")
yolodetect: None

conf_search = [str(pl.Path(__file__).absolute().parent) +
"/meta/config.conf",
               "meta/config.conf",
               "app/meta/config.conf",
               "G:/Git/Lazuli-Cross/app/meta/config.config",
               str(pl.Path(__file__).absolute().parent) +
"/meta/default.conf"] # automatically includes default
configuration
```

```
config_path: pl.Path
for conf in conf_search:
    if pl.Path(conf).exists():
        config_path = conf
        break

del conf_search


print("Reading configuration file...", end="\r")
config = open(config_path, "r")
options = (config.read()).split("\n")
yolodir: str
dmodeldir: str
tmodeldir: str
tempdir: str
resmode: str
ressz: int
detect_commands = ""
test_commands = ""

for opt in options:
    if (opt.split(" "))[0] == "yolodir:":
        yolodir = " ".join((opt.split(" "))[1:]) if len(opt) >
2 else (opt.split(" "))[1:]
    elif (opt.split(" "))[0] == "tempdir:":
        tempdir = " ".join((opt.split(" "))[1:]) if len(opt) >
2 else (opt.split(" "))[1:]
    elif (opt.split(" "))[0] == "resolutionmode:":
        resmode = opt.split(" ")[1]
    elif (opt.split(" "))[0] == "resolutionmaxwidth:":
        if resmode == "max-width":
            ressz = int(opt.split(" ")[1])
    elif (opt.split(" "))[0] == "resolutionmaxheight:":
        if resmode == "max-height":
            ressz = int(opt.split(" ")[1])
    elif (opt.split(" "))[0] == "autoresolution:":
        if resmode == "auto":
            ressz = int(opt.split(" ")[1])
    elif (opt.split(" "))[0] == "detectmodeldir:":
        dmodeldir = " ".join((opt.split(" "))[1:]) if len(opt)
> 2 else (opt.split(" "))[1:]
    elif (opt.split(" "))[0] == "testmodeldir:":
        tmodeldir = " ".join((opt.split(" "))[1:]) if len(opt)
```

```
> 2 else (opt.split(" "))[1:]
    elif (opt.split(" "))[0] == "detectmodeltargets:":
        temp = detect_commands + "--weights"
        for i in (opt.split(" "))[1:]:
            del detect_commands
            detect_commands = temp + " " + dmodeldir + "/" + i
if not i in [" ", "", "\n"] else temp
            del temp
            temp = detect_commands
    elif (opt.split(" "))[0] == "testmodeltargets:":
        temp = test_commands + "--weights"
        for i in (opt.split(" "))[1:]:
            del test_commands
            test_commands = temp + " " + tmodeldir + "/" + i if
not i in [" ", "", "\n"] else temp
            del temp
            temp = test_commands
    elif (opt.split(" "))[0] == "detectoverride:":
        temp = detect_commands + " " + " ".join(opt.split(" ")
[1:])
        del detect_commands
        detect_commands = temp
    elif (opt.split(" "))[0] == "testoverride:":
        temp = test_commands + " " + " ".join(opt.split(" ")
[1:])
        del test_commands
        test_commands = temp
    elif (opt.split(" "))[0] in ["#", " ", "", "\n"]:
        pass
    else:
        raise RuntimeError()
config.close()
del config
print("Reading configuration file... OK")


print("Checking directories...", end="\r")
if not pl.Path(yolodir).exists():
    raise FileExistsError("YOLO directory path [{}] does not
exist in your system".format(yolodir))
elif not pl.Path(yolodir + "/detect.py").exists():
    raise FileExistsError("YOLO detection algorithm path [{}]
does not exist in your system".format(yolodir + "/detect.py"))
else:
    sys.path.append(yolodir)
```

```
    yolodetect = import_module("detect")
    del yolodir
if not pl.Path(tempdir).exists():
    pl.Path(tempdir).mkdir()
if not pl.Path(tempdir + "/detected").exists():
    pl.Path(tempdir + "/detected").mkdir()
if not pl.Path(tempdir + "/testing").exists():
    pl.Path(tempdir + "/testing").mkdir()
print("Checking directories... OK")


temp = detect_commands + " --exist-ok --project " + tempdir + "
--name detected --view-img --augment --save-crop"
del detect_commands
detect_commands = temp
del temp
temp = test_commands + " --exist-ok --project " + tempdir + " -
-name testing --view-img --save-crop"
del test_commands
test_commands = temp
del temp

# Argument Parser copy from the detect.py located at the YOLO
directory
# modify accordingly if other YOLO detection algorithm
implementation is used
# presently resonates to @ultralytics YOLO implementation
parser = argparse.ArgumentParser()
parser.add_argument('--weights', nargs='+', type=str,
default='yolov5s.pt', help='model.pt path(s)')
parser.add_argument('--source', type=str,
default='data/images', help='file/dir/URL/glob, 0 for webcam')
parser.add_argument('--imgsz', '--img', '--img-size', type=int,
default=640, help='inference size (pixels)')
parser.add_argument('--conf-thres', type=float, default=0.25,
help='confidence threshold')
parser.add_argument('--iou-thres', type=float, default=0.45,
help='NMS IoU threshold')
parser.add_argument('--max-det', type=int, default=1000,
help='maximum detections per image')
parser.add_argument('--device', default='', help='cuda device,
i.e. 0 or 0,1,2,3 or cpu')
parser.add_argument('--view-img', action='store_true',
help='show results')
parser.add_argument('--save-txt', action='store_true',
```

```
help='save results to *.txt')
parser.add_argument('--save-conf', action='store_true',
help='save confidences in --save-txt labels')
parser.add_argument('--save-crop', action='store_true',
help='save cropped prediction boxes')
parser.add_argument('--nosave', action='store_true', help='do
not save images/videos')
parser.add_argument('--classes', nargs='+', type=int,
help='filter by class: --class 0, or --class 0 2 3')
parser.add_argument('--agnostic-nms', action='store_true',
help='class-agnostic NMS')
parser.add_argument('--augment', action='store_true',
help='augmented inference')
parser.add_argument('--update', action='store_true',
help='update all models')
parser.add_argument('--project', default='runs/detect',
help='save results to project/name')
parser.add_argument('--name', default='exp', help='save results
to project/name')
parser.add_argument('--exist-ok', action='store_true',
help='existing project/name ok, do not increment')
parser.add_argument('--line-thickness', default=3, type=int,
help='bounding box thickness (pixels)')
parser.add_argument('--hide-labels', default=False,
action='store_true', help='hide labels')
parser.add_argument('--hide-conf', default=False,
action='store_true', help='hide confidences')
parser.add_argument('--half', action='store_true', help='use
FP16 half-precision inference')

def  flush():
    if pl.Path(tempdir + "/target.jpg").exists():
        os.remove(tempdir + "/target.jpg")
    loc = tempdir + "/detected"
    floc = os.listdir(loc)
    for i in floc:
        if pl.Path(loc + "/" + i).is_file():
            os.remove(loc + "/" + i)
        elif pl.Path(loc + "/" + i).is_dir():
            rmtree(loc + "/" + i)
    del loc
    loc = tempdir + "/testing"
    floc = os.listdir(loc)
    for i in floc:
        if pl.Path(loc + "/" + i).is_file():
```

```
            os.remove(loc + "/" + i)
        elif pl.Path(loc + "/" + i).is_dir():
            rmtree(loc + "/" + i)

print("Cleaning temporary directory...", end="\r")
flush()
print("Cleaning temporary directory... OK")
print("Configuration resolved
[{}]".format(str(pl.Path(config_path))))
print("Temporary directory: {}".format(str(pl.Path(tempdir))))
print("Detection models directory:
{}".format(str(pl.Path(dmodeldir))))
print("Testing models directory:
{}".format(str(pl.Path(tmodeldir))))
print("Full detection commands (excludes source): " +
detect_commands)
print("Full testing commands (excludes source): " +
test_commands)

def capture(path):
    global resmode, ressz, detect_commands
    flush()
    name, extension = os.path.splitext(path)
    del name
    if not extension.lower() in [".png", ".jpg", ".jpeg"]:
        raise Exception("Only PNG and JPEG file types are
allowed.")
    local = Image.open(path)
    if resmode == "full":
        pass
    elif resmode == "half":
        height, width = local.size
        local = local.resize((int(height // 2), int(width //
2)), Image.ANTIALIAS)
    elif resmode == "max-width":
        height, width = local.size
        ratio = height / width
        newh = int(ressz * ratio)
        local = local.resize((newh, ressz), Image.ANTIALIAS)
    elif resmode == "max-height":
        height, width = local.size
        ratio = width / height
        neww = int(ressz * ratio)
        local = local.resize((ressz, neww), Image.ANTIALIAS)
    elif resmode == "auto":
```

```
        height, width = local.size
        ratio = min(height, width) / max(height, width)
        newh = ressz if height >= width else int(ressz * ratio)
        neww = ressz if width >= height else int(ressz * ratio)
        local = local.resize((newh, neww), Image.ANTIALIAS)
    local.convert("RGB")
    local.save(tempdir + "/target.jpg", "JPEG")
    temp = detect_commands
    del detect_commands
    detect_commands = temp + " --source " + tempdir +
"/target.jpg"

def detect():
    global detect_commands
    temp = detect_commands
    del detect_commands
    detect_commands = temp + " --source " + tempdir +
"/target.jpg"
    del temp
    dopt = (parser.parse_known_args(detect_commands.split("
")))[0]
    print(dopt)
    yolodetect.detect(**vars(dopt))

def _process(imgpath, loc):
    global tempdir
    capture = Image.open(loc + "/" + imgpath)
    edges =
imgproc.ImprovedSecondDerivativeEdgeDetection(capture)
    deisolated = (imgproc.ColorDeisolationRoutine(capture,
edges))[0]
    fpath = tempdir + "/testing/deisolated_" + imgpath
    deisolated.save(fpath, "JPEG")
    return "Processed " + imgpath + ": OK"

def test():
    topt = (parser.parse_known_args((test_commands + " --source
" + tempdir + "/testing").split(" ")))[0]
    loc = tempdir + "/detected/crops/Facemask"
    if not pl.Path(loc).exists():
        print("Found no detected supported images
[class=\'Facemask\'] to test.")
    else:
        if not pl.Path(tempdir + "/detected/test").exists():
            pl.Path(tempdir + "/detected/test").mkdir()
```

```
        floc = os.listdir(loc)
        print("Found {} detected supported
images".format(len(floc)))
        print("Processing...")
        localthreadpool = fut.ThreadPoolExecutor()
        start = time_ns()
        threads = {localthreadpool.submit(_process, i, loc): i
for i in floc}
        pending_work = localthreadpool._work_queue.qsize()
        pending = pending_work -
localthreadpool._work_queue.qsize() if (pending_work -
localthreadpool._work_queue.qsize()) >= 0 else 0
        ongoing = localthreadpool._work_queue.qsize()
        completed = len(floc) - (pending + ongoing)
        while ((pending > 0 or ongoing > 0) and completed !=
len(floc)):
            print(" " * 90, end="\r")
            print("Pending {0} of {1} images | Ongoing: {2} |
Completed: {3} | Time: {4:.2f}".format(pending, len(floc),
ongoing, completed, (time_ns() - start) / 1000000000),
end="\r")
            pending_work = localthreadpool._work_queue.qsize()
            pending = pending_work -
localthreadpool._work_queue.qsize() if (pending_work -
localthreadpool._work_queue.qsize()) >= 0 else 0
            ongoing = localthreadpool._work_queue.qsize()
            completed = len(floc) - (pending + ongoing)
            sleep(0.5)
        print(" " * 90, end="\r")
        print("Pending {0} of {1} images | Ongoing: {2} |
Completed: {3} | Time: {4:.2f}".format(pending, len(floc),
ongoing, completed, (time_ns() - start) / 1000000000),
end="\r")
        localthreadpool.shutdown(wait=True)
        print(" " * 90, end="\r")
        for future in fut.as_completed(threads):
            thr = threads[future]
            result = {}
            try:
                result = future.result()
                print(result)
            except Exception as e:
                print("Process [%r] raised error: %s" % (thr,
e))
        print("Processed {0} images | Time:
```

```
{1:.2f}".format(len(floc), (time_ns() - start) / 1000000000))
        print("Finished Processing Images")
        print(topt)
        yolodetect.detect(**vars(topt))
```