# Mask Detection

Before moving to coding part. Actually, what we are trying to solve this project? We are developing an image classification task, for example classifying dog or cat. Perhaps in this particular project the need for retaining the network is very less as compared to other images classification tasks like face recognition, emotion analysis and so on. Face recognition and other project need to train new faces and age variant training is needed to time. But in the case of mask detection, we set this trained model on a particular camera and once it is done then there is less chance to retrain the network for improving accuracy. So, it is important to keep in mind that our main model need better accuracy and f1-score. So that it will last for long period of time.

Since, here we use inceptionv3 model for classification. Basically, inceptionv3 is a large network and good for image classification model. Because of the complexity of network, we use a pretrained model and transfer lean it into generate our model. By using pre-trained model, we will achieve grater performance after training the model

Ok, let’s start,

First thing first, before starting the project we need clear idea about what we are do

1. Installing libraries
2. Prepare dataset
   1. I recommend you to choose the same images as we use real implementation. Maybe it is a footage from CCTV or any other source of camera.
3. Preprocessing dataset
   1. Cleaning noise, resizing image, augmentation…
4. Import inception v3 model
5. Training
6. Testing

## Prepare dataset

Here, we extract images from a video using yolo face detection. If we dealing with CCTV it is better to use yolo face detection model instead opencv.

YoloFaceDetection.py

import argparse

import sys

import os

from utils import \*

from datetime import datetime

# Give the configuration and weight files for the model and load the network

# using them.

model\_cfg = './cfg/yolov3-face.cfg'

model\_weights = './model-weights/yolov3-wider\_16000.weights'

net = cv2.dnn.readNetFromDarknet(model\_cfg, model\_weights)

net.setPreferableBackend(cv2.dnn.DNN\_BACKEND\_OPENCV)

net.setPreferableTarget(cv2.dnn.DNN\_TARGET\_CPU)

path\_dateset = os.path.join("dataset")

if not os.path.exists(path\_dateset):

    os.mkdir(path\_dateset)

import pdb

#face detection

class FaceDetection:

    def detect(self, frame):

        # Create a 4D blob from a frame.

        blob = cv2.dnn.blobFromImage(frame, 1 / 255, (IMG\_WIDTH, IMG\_HEIGHT),

                                     [0, 0, 0], 1, crop=False)

        # Sets the input to the network

        net.setInput(blob)

        # Runs the forward pass to get output of the output layers

        outs = net.forward(get\_outputs\_names(net))

        # Remove the bounding boxes with low confidence

        faces, faces\_o = post\_process(frame, outs, CONF\_THRESHOLD, NMS\_THRESHOLD)

        # save the face images into dataset folder

        for face in faces:

            x = abs(int(face[0]))

            y = abs(int(face[1]))

            w = abs(int(face[2]))

            h = abs(int(face[3]))

            # print(type(x))

            f\_face = frame[y:y+h, x:x+w]

            now = datetime.now()

            time = now.strftime("%Y\_%m\_%d\_%H\_%M\_%S\_%f.jpg")

            output\_file = os.path.join(path\_dateset, time)

            try:

                cv2.imwrite(os.path.join(output\_file), f\_face.astype(np.uint8))

            except :

                print("Frame error")

                pdb.set\_trace()

        return faces, faces\_o

# obj\_face = Face()

# test\_img = os.path.join("samples", "outside\_000001.jpg")

# frame = cv2.imread(test\_img)

# obj\_face.detection(frame)

After creating the dataset we need to manually sort images with and without mask. If we have lot of images in dataset it is better to use any dataset from Kaggle.

## Preprocessing dataset

We create a separate python file for data preprocessing because of the reusability of code.

ImagePrepocessing.py

# Preprocessing image using tensorlow keras preprocessor

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import os

import shutil

import random

path\_train\_dir = os.path.join("dataset", "train")

path\_test\_dir = os.path.join("dataset", "validation")

class Preprocess:

    def train\_and\_test\_split(self, class\_names, percentage1 = 80):

        if not os.path.exists(path\_train\_dir):

            os.mkdir(path\_train\_dir)

        if not os.path.exists(path\_test\_dir):

            os.mkdir(path\_test\_dir)

        for label in class\_names:

            path = os.path.join("dataset", label)

            images = os.path.listdir(path)

            random.shuffle(images)

            thesh = int(len(images) \* percentage1 / 100)

            path\_train\_class\_dir = os.path.join(path\_train\_dir, label)

            path\_test\_class\_dir = os.path.join(path\_test\_dir, label)

            if not os.path.exists(path\_class\_dir):

                os.mkdir(path\_class\_dir)

            for img in images[:thesh]:

                p\_img = os.path.join(path, img)

                d\_train\_dir = os.path.join(path\_train\_class\_dir, img )

                d\_test\_dir = os.path.join(path\_test\_class\_dir, img )

                # move images into train dir

                shutil.move(p\_img, d\_train\_dir)

                shutil.move(p\_img, d\_test\_dir)

    def dataAugment(self, train\_dir, validation\_dir):

        # set data augmentation parameter to ImageDataGenerator

        train\_datagen = ImageDataGenerator(rescale = 1./255.,

                        rotation\_range = 40,

                        width\_shift\_range = 0.2,

                        height\_shift\_range = 0.2,

                        shear\_range = 0.2,

                        zoom\_range = 0.2,

                        horizontal\_flip = True)

        # we not applying augmentation to validation data

        test\_datagen = ImageDataGenerator(rescale = 1./255.)

        # training images creation with batch 20 using train\_datagen

        train\_generator = train\_datagen.flow\_from\_directory(train\_dir,

                                                            batch\_size = 20,

                                                            class\_mode = 'binary',

                                                            target\_size = (150, 150)

                                                            )

        # testing images creation with batch 20 using test\_datagen

        validation\_generator = test\_datagen.flow\_from\_directory(validation\_dir,

                                                            batch\_size = 20,

                                                            class\_mode = 'binary',

                                                            target\_size = (150, 150)

                                                            )

        return train\_generator, validation\_generator

## Import inception v3 model

Import inception v3

from tensorflow.keras.applications.inception\_v3 import InceptionV3

load pretrained model

here we use 150x150x3 images. For transfer learning we remove the fully connected layer(final layer) form inception v3.

pretrained\_model = InceptionV3(input\_shape = (150,150,3), # shape of input image

                                include\_top = False, # Removing fully connected layer

                                weights = 'imagenet'

                                )

Make all the layers non trainable

for layer in pretrained\_model.layers:

    layer.trainable = False

Here we using RMSprop as optimizer with learning rate 0.0001 and loss metrics as binary\_crossentropy

# flatten output layer to 1D

x = layers.Flatten()(pretrained\_model.output)

# add FC layer with 1024 hidden units and RELU activation

x = layers.Dense(1024, activation = 'relu')(x)

# add a dropout rate of 0.2

x = layers.Dropout(0.2)(x)

# add a final sigmoid for classification

x = layers.Dense(1, activation = 'sigmoid')(x)

model = Model(pretrained\_model.input, x)

model.compile(optimizer = RMSprop(lr = 0.0001),

                loss = 'binary\_crossentropy',

                metrics = ['acc'])

## Training

Before training, we need to preprocess data. At preprocessing as I explained above is not calling the method.

# train and test split

obj\_preprocess.train\_and\_test\_split("mask", "no mask")

train\_dataset\_dir = os.path.join("dataset", "train")

validation\_dataset\_dir = os.path.join("dataset", "validation")

train\_generator, validation\_generator = obj\_preprocess.dataAugment(train\_dataset\_dir, validation\_dataset\_dir)

training the model

history = model.fit\_generator(

    train\_generator,

    validation\_data = validation\_generator,

    step\_per\_epoch = 100,

    epoches = 100,

    validation\_steps = 50,

    verbose = 2

)