Real Time Face Mask Detection Using Python

ANISHK YADAV
B. Tech-C.S.E(2018-22)ACED,
yanishkBTECH18@ced.alliance.edu.in

Abstract **- The Face Mask Detection In Python is a simple project developed using Python. The project contains a base model to convert images into arrays and then frame it and compare it with the image dataset. The user can check whether a person is wearing a mask or not with the video stream. The project file contains 2 python scripts (mask_detect_train.py and detect_mask_video.py) and the dataset with 2 categories(with_mask and without_mask). This is a simple command-based project which is very easy to understand and use. Also, this project makes a convenient way for the user to gain an idea of how to perform image processing using python.**

KEYWORDS - Face mask detection, Covid prevention, Covid-19, k-NN,
DNN, MobilenetV2, ADAM, Image Processing, CNN, Artificial Intelligence.

INTRODUCTION

In the new world of coronavirus, multidisciplinary efforts have been organized to slow the spread of the pandemic. The AI community has also been a part of these endeavors. In particular, developments for monitoring social distancing or identifying face masks have made the headlines.

But all this hype and anxiety to show off results as fast as possible, added up to the usual AI overpromising factor, maybe signaling the wrong idea that solving some of these use cases is almost trivial due to the mighty powers of AI.

To paint a more complete picture, we decided to show the creative process behind a solution for a seemingly simple use case in computer vision:

1. **Detect** people that pass through a security-like camera.
2. **Identify** face mask usage.
3. **Collect reliable statistics** (% of people wearing masks).

In a nutshell, the system works by performing two main tasks:

- **Object detection** with a neural network, pre-trained for face detection.

  Output: list of bounding boxes around each detected face.

- **Classification** in two classes (with/without mask), using another neural net (MobileNetV2).

  Output: score from 0 to 1 signifying the probability of a face wearing a mask.

As there was no pre-trained classifier to distinguish faces with and without masks, We trained this model with a dataset. This dataset with nearly ~1400 images in a very clever and effective way: We took a dataset with regular faces, and artificially added masks to some of the images. We leveraged other computer vision techniques to automatically place the masks over the faces.

Some of the images in the dataset are shown below:

MobilenetV2 : MobileNetV2 is a state of the art for mobile visual recognition including classification, object detection and semantic segmentation. This classifier uses Depth wise Separable Convolution which is introduced to dramatically reduce the complexity cost and model size of the network, and hence is suitable to Mobile devices, or devices that have low computational power. In MobileNetV2, another best module that is introduced is inverted residual structure. Non-linearity in narrow layers is deleted. Keeping MobileNetV2 as backbone for feature extraction, best performances are achieved for object detection and semantic segmentation.

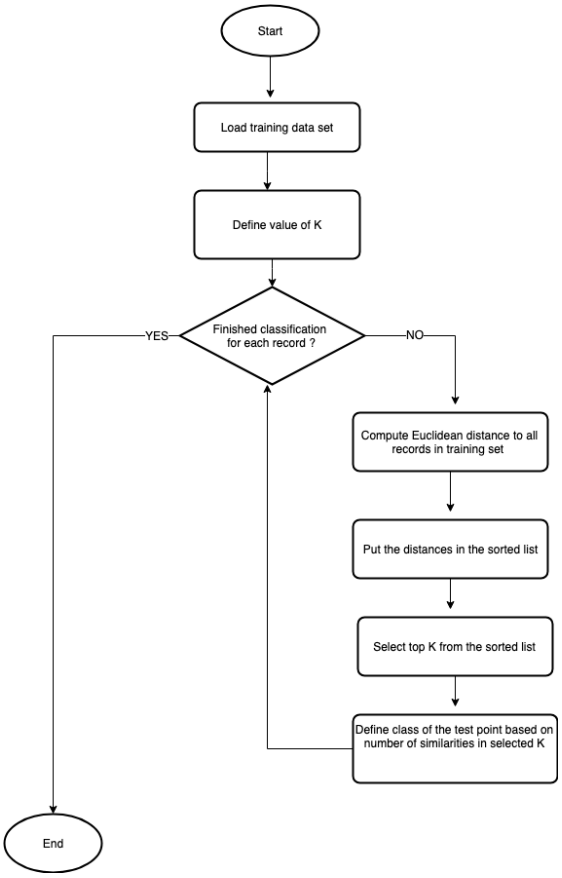
ADAM: Adam, an algorithm for first-order gradient-based optimization of stochastic objective functions, is based on adaptive estimates of lower-order moments. This method is computationally efficient and can execute with little memory requirements. It is invariant to a diagonal rescaling of the gradients, which is well suited for problems that are large in terms of data and/or parameters. The hyper-parameters have intuitive interpretations

and they typically require little tuning. Empirical results show that Adam works well in practice and compares favorably to other stochastic optimization methods.

ALGORITHM

- K-nearest neighbors (KNN) algorithm is a type of supervised ML algorithm that can be used for both classifications as well as regression predictive problems. However, it is mainly used for classification predictive problems in the industry.
- K-nearest neighbors (KNN) algorithm uses 'feature similarity' to predict the values of new datapoints which further means that the new data point will be assigned a value based on how closely it matches the points in the training set. We can understand its working with the help of the following steps:−
- Step 1 − For implementing any algorithm, we need a dataset. So during the first step of KNN, we must load the training as well as test data.
- Step 2 − Next, we need to choose the value of K i.e. the nearest data points. K can be any integer.
- Step 3 − For each point in the test data do the following −
- 3.1 − Calculate the distance between test data and each row of training data with the help of any of the methods.
- 3.2 − Now, based on the distance value, sort them in ascending order.
- 3.3 − Next, it will choose the top K rows from the sorted array.
- 3.4 − Now, it will assign a class to the test point based on the most frequent class of these rows. Step 4 – En**d**

FLOWCHART



PROJECT CODE

i. (mask_detect_train.py)

 # import the necessary packages

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import MobileNetV2

from tensorflow.keras.layers import AveragePooling2D

from tensorflow.keras.layers import Dropout

from tensorflow.keras.layers import Flatten

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import Input

from tensorflow.keras.models import Model

```
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import os

# initialize the initial learning rate, number of epochs to train for,
# and batch size
INIT_LR = 1e-4
EPOCHS = 20
BS = 32

DIRECTORY=r"C:\Users\anshy\Documents\Python\Face_Mask_Project\dataset"
CATEGORIES = ["with_mask", "without_mask"]

# grab the list of images in our dataset directory, then initialize
# the list of data (i.e., images) and class images
print("[INFO] LOADING IMAGES")

data = []
labels = []

for category in CATEGORIES:
    path = os.path.join(DIRECTORY, category)
    for img in os.listdir(path):
        img_path = os.path.join(path, img)
        image = load_img(img_path, target_size=(224, 224))
        image = img_to_array(image)
        image = preprocess_input(image)

        data.append(image)
        labels.append(category)

# perform one-hot encoding on the labels
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)

data = np.array(data, dtype="float32")
labels = np.array(labels)

(trainX, testX, trainY, testY) = train_test_split(data, labels,
        test_size=0.20, stratify=labels, random_state=42)

# construct the training image generator for data augmentation
aug = ImageDataGenerator(
        rotation_range=20,
        zoom_range=0.15,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.15,
        horizontal_flip=True,
        fill_mode="nearest")
```

```
# load the MobileNetV2 network, ensuring the head FC layer sets are
# left off
baseModel = MobileNetV2(weights="imagenet", include_top=False,
        input_tensor=Input(shape=(224, 224, 3)))

# construct the head of the model that will be placed on top of the
# the base model
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(128, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation="softmax")(headModel)

# place the head FC model on top of the base model (this will become
# the actual model we will train)
model = Model(inputs=baseModel.input, outputs=headModel)

# loop over all layers in the base model and freeze them so they will
# *not* be updated during the first training process
for layer in baseModel.layers:
        layer.trainable = False

# compile our model
print("[INFO] compiling model...")
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
        metrics=["accuracy"])

# train the head of the network
print("[INFO] training head...")
H = model.fit(
        aug.flow(trainX, trainY, batch_size=BS),
        steps_per_epoch=len(trainX) // BS,
        validation_data=(testX, testY),
        validation_steps=len(testX) // BS,
        epochs=EPOCHS)

# make predictions on the testing set
print("[INFO] evaluating network...")
predIdxs = model.predict(testX, batch_size=BS)

# for each image in the testing set we need to find the index of the
# label with corresponding largest predicted probability
predIdxs = np.argmax(predIdxs, axis=1)

# show a nicely formatted classification report
print(classification_report(testY.argmax(axis=1), predIdxs,
        target_names=lb.classes_))

# serialize the model to disk
print("[INFO] saving mask detector model...")
model.save("mask_detector1.model", save_format="h5")

# plot the training loss and accuracy
N = EPOCHS
plt.style.use("ggplot")
plt.figure()
```

```
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig("plot1.png")
```

ii. (detect_mask_video.py)

```
# import the necessary packages
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
from imutils.video import VideoStream
import numpy as np
import imutils
import time
import cv2
import os

def detect_and_predict_mask(frame, faceNet, maskNet):
	# grab the dimensions of the frame and then construct a blob
	# from it
	(h, w) = frame.shape[:2]
	blob = cv2.dnn.blobFromImage(frame, 1.0, (224, 224),
		(104.0, 177.0, 123.0))

	# pass the blob through the network and obtain the face detections
	faceNet.setInput(blob)
	detections = faceNet.forward()
	print(detections.shape)

	# initialize our list of faces, their corresponding locations,
	# and the list of predictions from our face mask network
	faces = []
	locs = []
	preds = []

	# loop over the detections
	for i in range(0, detections.shape[2]):
		# extract the confidence (i.e., probability) associated with
		# the detection
		confidence = detections[0, 0, i, 2]

		# filter out weak detections by ensuring the confidence is
		# greater than the minimum confidence
		if confidence > 0.5:
			# compute the (x, y)-coordinates of the bounding box for
			# the object
			box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
			(startX, startY, endX, endY) = box.astype("int")
```

```
                # ensure the bounding boxes fall within the dimensions of
                # the frame
                (startX, startY) = (max(0, startX), max(0, startY))
                (endX, endY) = (min(w - 1, endX), min(h - 1, endY))

                # extract the face ROI, convert it from BGR to RGB channel
                # ordering, resize it to 224x224, and preprocess it
                face = frame[startY:endY, startX:endX]
                face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
                face = cv2.resize(face, (224, 224))
                face = img_to_array(face)
                face = preprocess_input(face)

                # add the face and bounding boxes to their respective
                # lists
                faces.append(face)
                locs.append((startX, startY, endX, endY))

        # only make a predictions if at least one face was detected
        if len(faces) > 0:
                # for faster inference we'll make batch predictions on *all*
                # faces at the same time rather than one-by-one predictions
                # in the above `for` loop
                faces = np.array(faces, dtype="float32")
                preds = maskNet.predict(faces, batch_size=32)

        # return a 2-tuple of the face locations and their corresponding
        # locations
        return (locs, preds)

# load our serialized face detector model from disk
prototxtPath=r"C:\Users\anshy\Documents\Python\Face_Mask_Project\face_detector\deploy.prototxt"
weightsPath=r"C:\Users\anshy\Documents\Python\Face_Mask_Project\face_detector\res10_300x300_ssd_iter_140000.caffemodel"
faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)

# load the face mask detector model from disk
maskNet = load_model("mask_detector1.model")

# initialize the video stream
print("[INFO] STARTING VIDEO STREAM")
vs = VideoStream(src=0).start()

# loop over the frames from the video stream
while True:
        # grab the frame from the threaded video stream and resize it
        # to have a maximum width of 400 pixels
        frame = vs.read()
        frame = imutils.resize(frame, width=400)

        # detect faces in the frame and determine if they are wearing a
        # face mask or not
```

(locs, preds) = detect_and_predict_mask(frame, faceNet, maskNet)

 # loop over the detected face locations and their corresponding

 # locations

for (box, pred) in zip(locs, preds):

 # unpack the bounding box and predictions

(startX, startY, endX, endY) = box

(mask, withoutMask) = pred

 # determine the class label and color we'll use to draw

 # the bounding box and text

label = "Mask" if mask > withoutMask else "No Mask"

color = (0, 255, 0) if label == "Mask" else (0, 0, 255)

 # include the probability in the label

label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)

 # display the label and bounding box rectangle on the output

 # frame

cv2.putText(frame, label, (startX, startY - 10),

cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)

cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)

 # show the output frame

cv2.imshow("Frame", frame)

key = cv2.waitKey(1) & 0xFF

 # if the `q` key was pressed, break from the loop

if key == ord("q"):

break

 # do a bit of cleanup

cv2.destroyAllWindows()

vs.stop()

RESULT

THE WHOLE EXECUTION IS RECORDED AND ATTACHED WITH THE PROJECT DOCUMENT SEPARATELY

SUMMARY

The Project Can Be Implemented Easily In Only Few Steps:

- Install The Dependencies
- Write The Python Script
- Load The Dataset
- Pre Process The Data
- Train The Model
- Run And View The Model Accuracy
- Use Model In Real Time Camera
- Get The Final Result

FUTURE SCOPE

The current system is evaluated with different classifiers. The best system may be implemented along with interfacing with alarm and alerting systems in the near future. This system may be integrated with a system which can integrate with a system implementing social distancing which can make it a wholesome system which can bring dramatic impact on the spread.

We're headed for a faceless future as masks become the new normal. That can be a big security concern, experts say" - By Luke McGee, CNN - Though using face masks is proved to be the best solution to mitigate the spread of air borne viruses like Corona; it poses a big security challenge to the nation ahead as it could create opportunities for people who cover their faces for nefarious reasons.

Experts say mass mask-wearing could complicate crime investigations in the coming days, as facial recognition becomes an important part in tracking criminals. Human beings have proved out to be very good at recognizing familiar faces and facial recognition algorithms are getting better in identifying patterns. These masks throw new challenges into the mix. This challenge may create a scope to new face detection algorithms which can identify faces which are covered with greater accuracies.