Face Mask Detection Using CNNs & OpenCV

Yash Gupta
0869229
Software Engineering
ygupta1@lakeheadu.ca

Marko Javorac
1107295
Software Engineering
mjavorac@lakeheadu.ca

Peter Sertic
0694592
Software Engineering
pisertic@lakeheadu.ca

***Abstract*—As COVID-19 ravages the world, businesses need to adjust to political and regulatory conditions in order to survive. Mandatory mask wearing has become an intricate mitigation technique enforced by states all over the world. An opportunity to automate mask enforcement presents itself through relatively simple computer vision and machine learning techniques. This paper proposes and implements a real-timmask detection system built on OpenCV and Tensorflow libraries. An in depth literature review also analyzes the current state of face detection along with neural nets and object detection .**

I. Introduction

The world faces its biggest pandemic since the Spanish Flu, severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2) or Covid-19. After the first confirmed cluster located in China in december of 2019[5], the Corona Virus has garnered the attention of the entire world. World economies are weakening as government responses, such as lockdowns and the shutdown of businesses, have put regular daily life as we know it on hold. With drastic actions to counteract the pandemic, the adverse effects are already being seen. Mental health, the economy, and world politics have all seen negative consequences as a result. It is vital that the pandemic is closed as quickly as possible, which likely will only arrive once an effective vaccine is mass produced and distributed among the public. Until then, counter measures must be employed to slow the spread of the virus.

The most popular and simplest measure for stopping such spread is the application of face masks. As effective as this method is, it is only effective if most people(¿70%) agree to adhere to it[4]. Those who do not will allow the virus to spread and the pandemic will continue. There must be a way to enforce the mask wearing policy without requiring a tremendous amount of resources behind it. The solution lies in technological implementations, such as face detection. By utilizing face detection methods, in conjunction with machine learning models, a program can be created which can detect faces as either having a mask on or not. Such a program, provided as open source or a commercial product, could be implemented in many public sectors and businesses in order to ensure the local population is adhering to the mask wearing policies.

II. Literature Review

Mask detection is fundamentally intertwined with facial recognition due to its visual and technological overlap. The foundation laid by facial recognition technologies over the past decade will directly translate over to new implementations. [7]Ashu Kumar et al. reviews the current landscape of technologies and the challenges associated with their respective applications. Some key challenges in face detection include Complex backgrounds, Poor Resolution , Facial Orientations, and Multiple Faces. All of these challenges can rapidly reduce the performance of the models. The authors break face detection into two major categories, Feature based, and Image Based. Feature based methods attempt to extract features from the image and compare them against its knowledge of facial features. Image based methods try to find the best match between training and testing images which then is translated to new image inputs. The authors proceed to break down each method into more and more specific implementations such as Viola-Jones searching or PDM shape modeling. A central aspect of image detection is the dataset that the model is constructed on. A large amount of high-quality standard image sets for both training and testing purposes are referenced within the text. As the field matures from research to real world deployment, numerous API's and solutions by large tech corporations are highlighted by the authors. This paper is a great starting point for anyone that wishes to explore and understand how contemporary facial recognition works.

Due to the high complexity of facial features, neural networks are employed to extract facial features from face images. Mohammed Loey[3] et.al proposes a hybrid model that uses Resnet50 for feature extraction and traditional classifiers such as decision trees, Support Vector Machines(SVM) and ensemble algorithm for its solution. One of the key strengths of this proposed model is the strong history and performance[6] of ResNet50. This residual neural network is defined by its 50-layer depth at which last layers are replaced with the proposed classifiers. This method of repurposing an excellent and well-documented image classifier for facemack detection allowed the researchers to quickly develop, test, and validate their proposed solution. This is particularly important in an ever evolving pandemic with serious health implications. Coskun et al. mentions other classifiers that can be used to achieve this, such as Deep Belief Networks(DBN) and Stacked Autoencoder, however, CNN is by far the most widely used[2]. The paper aims to produce a highly accurate recognition algorithm by

expanding upon the CNN model with batch normalization processes after two individual layers[C]. This is not necessary for this application, as completing the functionality of the program is the main priority, not increasing the program performance metrics. The paper provides a breakdown of the CNN architecture which is very helpful in understanding what CNNs are and how they function.

A CNN consists of four layers: Convolutional layer; Pooling layer; ReLU layer; Fully Connected Layer;[C] The convolutional layer extracts features from the input image by taking small squares of the input image and convoluting them with a set of employable neurons. This produces a feature map which is fed to the following convolution layers while preserving the spatial relation of pixels[C]. The pooling layer reduces the dimensionality of each feature map while preserving the most important information and achieves faster convergence and better generalization, and is placed between convolutional layers[C]. The ReLU layer uses ReLU, an activation function that is a non-linear, element wise operation which is applied to every pixel of the image. It sets all negative values in the feature map to zero[C]. The fully connected layer is the final pooling layer which feeds the extracted image features to a softmax activation feature of a classifier. The layer connects the filters of previous layers with the filters of following layers. The softmax activation feature takes values and scales them between one and zero. These values are then used to classify, with a cutoff value separating the classes(i.e. Mask off ¡= 0.6). This CNN structure allows us to build classifiers to identify facial features. This can be created and implemented originally or can be offloaded to external tools, such as openCV.

Object detection shares a similar workflow with facial recognition as they both employ OpenCV as their primary toolset. Meghna[8] et al. outlines how OpenCV was developed with real time applications in mind. As the need for framework integration grew, libraries such as TensorFlow and PyTorch were directly supported by OpenCV. This cross platform support demonstrates why open source tools are critical to any successful computervison implementation. The authors continue to explain how a Haar-like classifier uses custom filters to generate the feature set needed for the feature based classification. A cascade classifier is proposed that includes multiple stages of recognition which outputs the final prediction. While the authors present an image with a corresponding bounding box around objects, they fail to show the performance of their model in any meaningful way. It is important to validate and do comparative analysis with similar works .

Once face detection and extraction is implemented, there must be a way to analyze the extracted image to classify whether or not it is labeled under “mask” or “no mask”. Milantite et. al proposes DNNs(deep neural network) as a tool for such classification, specifically CNNs(Convolutional Neural Networks)[1]. The authors implemented a VGG-16 model, a modified version of CNN, as it reduces computational complexity[2]. Due to implementation time, it would be better suited for our research to utilize a base CNN implementation. The paper also proposes metrics, accuracy, training time, and learning error, to measure the effectiveness of the classifier[2]. These metrics are simple to acquire and provide enough information for us to access the model with regards to the depth of complexity we are trying to keep. The VGG-16 model, after one hundred epochs, achieved an accuracy of 96%[2]. We will compare our model accuracy to this, and reflect if the choice of a base CNN model was justified. Overall, this paper is a good foundation for understanding mask classification and the different CNN variants for it.

III. Proposed Model

Our solution includes multiple steps in order to generate an accurate prediction. The first step involves preprocessing a database of images. These preprocessed images are then fed into a model for training and validation purposes. Once the model is constructed, we use OpenCV to create a real-time image feed that uses feature recognition to extract a face within the frame. This face is then feed to the classifier which labels whether the individual is wearing a mask or not.

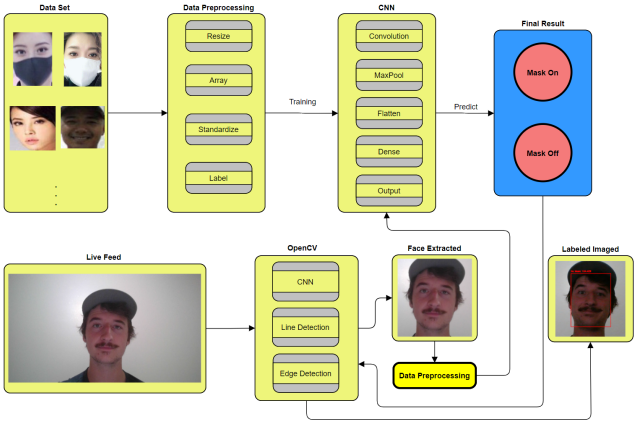

Fig. 1. Model Overview

A. Data Set

For training, validating and Testing our system, two different datasets were used. The first dataset was used train and validate the model. The other dataset was used for testing purposes. The datasets were collected from Github[9][10]. For testing and validation a total of 3832 images where 1916 images has faces with mask and 1916 images had faces without masks. The testing set had a total of 1376 images with 690 images with masks on and 686 images without masks.

1) Data Preprocessing: Before the data can be fed into the training model a series of preprocessing steps were performed.

- Resizing - The images were first resized to a uniform size of 224x224 as CNN requires a uniform input size.
- Vectorizing - After that the images were converted into an array for CNN.
- Standardization - Standardizing the pixel values for best results from the ReLu activation function.
- - The last step of preprocessing is assigning labels to the images 1 (with mask) or 0 (without mask).

B. CNN

Our model was trained on a Convolutional Neural Network. CNN's is suitable for spatial data such as images and hence can be used for classification. In our implementation the CNN had an input of 224 x 224 x 3 input i.e. an RGB image of dimensions 224 x 224. The neural network had a total of 8 layers where the first 5 layers performed feature extraction and the latter layers were used for classification.

- Convolution Layer - This layer performs feature extraction and generates a feature map.
- Pooling Layer - This layer sub samples from the feature map generated from the Convolution layer and generates a pooled feature map.
- Flatten Layer - The flatten layer generates a 1-d array from the matrices generated from the Convolution and the Pooling layer and feeds it to the Dense layer.
- Dense Layer - The Dense layer prepares the data collected from the flatten layer to be fed into the output layer.
- Output Layer - The Output layer generates the final label for the image and has 2 neurons for each classification label.

The activation function used was ReLu which is known for its classification properties. The model used Sparse categorical loss function and used adam optimizer. The model was iterated for 5 epochs and then saved.

C. OpenCV

OpenCV library was used to detect faces in a live stream. This was achieved using cascade filters, Harr features to detect faces based on the pixel intensities on key facial features. After faces were detected in a frame, the region of interest was extracted. The image is then preprocessed and fed to the classifier to be classified.[9]

IV. Experimental Analysis

TABLE I
RESULTS

Data	*Accuracy*
Validation set	95.83%
Testing Set 1	95.65%
Testing Set 2	94.45%

As we can see from the table our model achieved high accuracies on the validation set and unseen test sets.

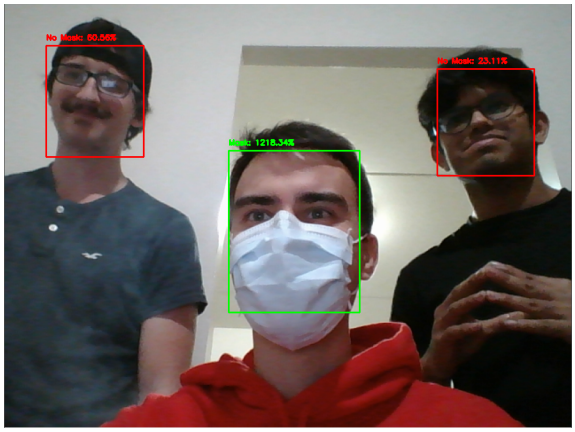

Fig. 2. Model Output

V. Conclusion

With face masks becoming a staple of modern life for the foreseeable future, automating mask enforcement within a business environment will play a key role in maintaining safe working conditions for employers and employees alike. A strong survey of current literature within the computer vision and mask detection domain highlights a unique and ever growing field of research and real world implication. Built on OpenCV and Tensorflow libraries, our proposed solution demonstrates a highly accurate prediction along with a real time camera-based implementation. Future work encompesses a larger and more diverse dataset along with stronger validation techniques.

References

[1] S. V. Militante and N. V. Dionisio, "Real-Time Facemask Recognition with Alarm System using Deep Learning," 2020 11th IEEE Control and System Graduate Research Colloquium (ICSGRC), Shah Alam, Malaysia, 2020, pp. 106-110, doi: 10.1109/ICSGRC49013.2020.9232610.

[2] M. Coşkun, A. Uçar, Ö. Yildirim and Y. Demir, "Face recognition based on convolutional neural network," 2017 International Conference on Modern Electrical and Energy Systems (MEES), Kremenchuk, 2017, pp. 376-379, doi: 10.1109/MEES.2017.8248937

[3] Loey, Mohamed et al. A hybrid deep transfer learning model with machine learning methods for face mask detection in the era of the COVID-19 pandemic." Measurement : journal of the International Measurement Confederation vol. 167 (2021): 108288. doi:10.1016/j.measurement.2020.108288

[4] Bai, Nina. Still Confused About Masks? Heres the Science Behind How Face Masks Prevent Coronavirus. Still Confused About Masks? Heres the Science Behind How Face Masks Prevent Coronavirus — UC San Francisco, 19 Nov. 2020, www.ucsf.edu/news/2020/06/417906/still-confused-about-masks-heres-science-behind-how-face-masks-prevent.

[5] Holshue, Michelle L., et al. First Case of 2019 Novel Coronavirus in the United States: NEJM. New England Journal of Medicine, 7 May 2020, www.nejm.org/doi/full/10.1056/NEJMoa2001191.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet classification with deep convolutional neural networks. Commun. ACM 60, 6 (June 2017), 84–90. DOI:https://doi.org/10.1145/3065386

[7] Kumar, A., Kaur, A. & Kumar, M. Face detection techniques: a review. Artif Intell Rev 52, 927–948 (2019). https://doi.org/10.1007/s10462-018-9650-2

[8] Saxena, Meghna Raj, et al. "REAL TIME OBJECT DETECTION USING MACHINE LEARINING AND OPENCV."International Journal of Information Sciences and Application (IJISA). ISSN 0974-2255, Vol.11, No.1, 2019

[9] https://github.com/balajisrinivas/Face-Mask-Detection
[10] https://github.com/aieml/face-mask-detection-keras

VI. Appendix

A. Data Preprocessing & Training

```
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
import tensorflow as tf
import keras
from keras.models import model_from_json

from tensorflow.keras.layers import Input
from tensorflow.keras.models import Sequential
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.losses import
    SparseCategoricalCrossentropy
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
import matplotlib.pyplot as plt
from tensorflow.keras import datasets, layers, models
import numpy as np
import os
from sklearn.pipeline import Pipeline
from keras.models import model_from_json

# Directory The image dataset is stored
DIRECTORY = r"C:\Users\yashb"
CATEGORIES = ["with_mask", "without_mask"]
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)
      if (category == CATEGORIES[1]):
         labels.append(1)
      else:
         labels.append(0)

data = np.array(data, dtype="float32")
labels = np.array(labels)
(trainX, testX, trainY, testY) = train_test_split(data,
    labels,
   test_size=0.20, random_state=42)

# creating a model
model = Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
    input_shape=(224, 224, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(2))
model.compile(optimizer='adam',
          loss=SparseCategoricalCrossentropy(from_logits=True),
          metrics=['accuracy'])
history = model.fit(trainX, trainY, epochs=5,
              validation_data=(testX, testY))
model.save("cnnmodel.h5")
```

B. Mask Detection

```
# import the necessary packages
import tensorflow as tf
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"face_detector\deploy.prototxt"
weightsPath =
     r"face_detector\res10_300x300_ssd_iter_140000.caffemodel"
faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)
model = tf.keras.models.load_model("cnnmodel.h5")
# load the face mask detector model from disk
maskNet = 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, height=1000,width=1000)

   # detect faces in the frame and check if they are
       wearing a
   # 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()
```