FACE MASK DETECTION USING DEEP LEARNING

1. INTRODUCTION

During the spread of COVID-19, the CDC recommended early on in the pandemic that all healthy kids and adults wear a mask. As many people have stepped into new normalcy, it is still important to know the benefits of masking, recognizing that N95, KN95 and KF94 respirators provide the highest level of protection.

If the community level of risk for COVID-19 is "medium" and you are at high risk for severe COVID-19, the CDC recommended you wear a "high-quality mask. If you're getting together with someone who's at higher risk for severe illness, the CDC recommended getting tested prior to seeing them and wearing a mask while you're with them.

Face mask detection refers to detecting whether a person is wearing a mask or not. In fact, the problem involves reverse engineering of face detection where the face is detected using different machine learning algorithms for the purpose of security, authentication and surveillance. Face detection is a key area in the field of Computer Vision and Pattern Recognition. The primary research on face detection was done in 2001 using the design of handcraft feature and application of traditional machine learning algorithms to train effective classifiers for detection and recognition. The problems encountered with this approach include high complexity in feature design and low detection accuracy. In recent years, face detection methods based on deep convolutional neural networks (CNN) have been widely developed to improve detection performance [1].

In order to monitor the usage of masks in public places, we have come up with a Face Mask Detection tool using Convolution Neural network (CNN) Model, which makes life easier to automate the process of detecting people with and without masks and create awareness of mask usage accordingly. This tool is implemented using OpenCV to read the images in Pytorch framework for training and testing the model.

PyTorch is an open-source machine learning library used primarily for applications such as computer vision and natural language processing. PyTorch has a simple Python interface and provides a powerful API. PyTorch can also be easily implemented on both Windows and Linux. PyTorch supports dynamic computational graphs, which means the network behavior can be changed programmatically at runtime. This facilitates more efficient model optimization and gives PyTorch a major advantage over other machine learning frameworks, which treat neural networks as static objects.

Object recognition is a general term describing a collection of related computer vision tasks that involve classification and localization. Image classification involves predicting the class of one object in an image. Object localization refers to identifying the location of one or more objects in an image and drawing a bounding box. This report shall discuss in detail the technical implementation behind the problem.

2. RELATED WORK

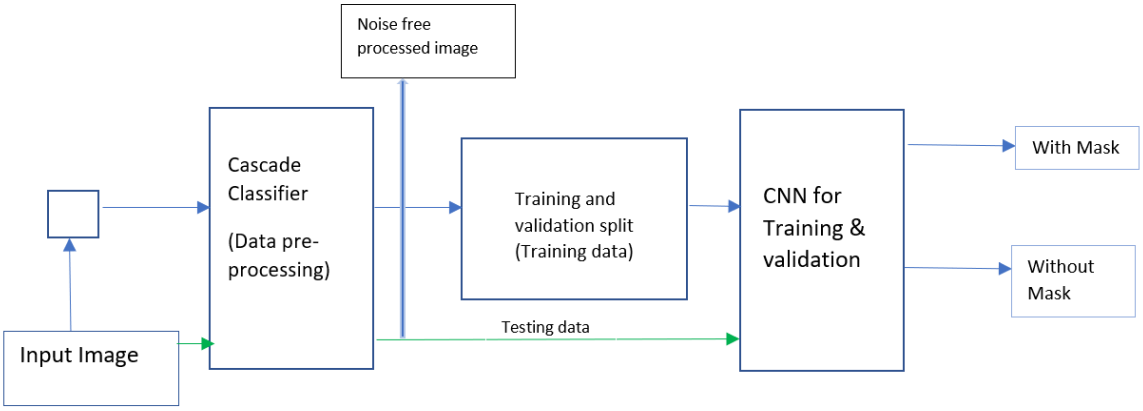
There has already been considerable work on this topic using different algorithms. One such implementation is using YOLOV5. But YOLOV5 has the disadvantage of detecting small objects in images that appear in groups.
Using VGG16, while the number of epoch's required to achieve max accuracy has decreased generally, however the loss is taking much longer to converge to minima. The introduction of more layers in VGG has allowed the model to better understand the features within an image. However constantly learning and relearning is a problem with VGG which is why the loss seems to be so unpredictable (explosion of gradients). But we have better loss values compared to VGG16.

3. TECHNICAL APPROACH :

Dataset:

We collected a comprehensive dataset of 1248 images of people with masks and 1510 images of people without masks to train our data. The data sources include Kaggle and UCI repository and also additional images of people from open-sourced websites. We segregated the images into two folders "with mask" and "without mask", where the folder names were used to identify the classifier labels.

Project Overview :



The project involves the following stages for delivering the required classified output :

1. Data pre-processing
2. Training
3. Model
4. Testing

1. Data Pre-processing:

1. The input image is read along with the location and converted to Grayscale using open CV built in functions – cv2.imread and cv2.cvtcolor.

```
def face_crop(path_link, filename):
    print(path_link)
    image = cv2.imread(path_link)

    #feature extraction
    grayscale = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) #RGB to Grayscale conversion
    face_img = face_cascade.detectMultiScale(grayscale, 1.3, 5) #Detects objects of different sizes in the input image.
```

2. The noise data is cropped and the image is saved in the same location.

```
for (x,y,w,h) in face_img:
    image = image[y:y+h, x:x+w] #crop image for removing noise data
    os.remove(path_link) #remove old image
    cv2.imwrite(filename, image) #save the image
```

3. The processed images are then stored in the same location with a different filename.

```
for i in os.listdir('dataset/'):
    file = os.path.join('dataset/', i) #adding path to image
    file = file + '/'
    count = 0
    for v in os.listdir(file):    #getting processed image for training
        image = os.path.join(file, v)
        print(image)
        filename = str(count) + '_after' +'.jpg'
        filename = os.path.join(file, filename)
        face_crop(image, filename)
        count += 1
    # print(filename, count)
```

2. Training :

1. For Training, the libraries torchvision.datasets, matplotlib and utils are imported. Torchvision.datasets provide built in functions for data transformation, matplotlib is used for visualisation and utils help in importing the utility file components which have the CNN operation.

```
import torchvision.datasets as datasets #for dataset built in functions
import matplotlib.pyplot as plt #for visualization
import utils

from utils import *
```

2. Images are loaded using Image loader function and randomly split into training and validation sets in the ratio 80:20.

```
from utils import *
#load data and applying transform for uniform images
image_dataset = datasets.ImageFolder('./dataset', transform=data_transform)

#define train and validation size
#split train/val : 80/20
train_size= int(0.8*len(image_dataset))
val_size = len(image_dataset) - train_size

training_set, validation_set = torch.utils.data.random_split(image_dataset, [train_size, val_size])
#training_set, validation_set = torch.utils.data.random_split(image_dataset, [1928, 847])

print('Train set:', len(training_set))
print('Validation set:', len(validation_set))
```

3. Training is done by defining the empty lists for loss functions and accuracy and also by defining loss function using crossentropy() function and Stochastic Gradient Descent optimiser and calling the CNN created.

```
model = CNN() #calling the CNN

#create empty loss and accuracy lists
train_lossfunc = []
val_lossfunc = []
train_accuracy = []
val_accuracy = []

def Training_Model(model, epochs, parameters):
    #Using CrossEntropyLoss, SGD optimiser
    loss_func = nn.CrossEntropyLoss()
    optimizer = torch.optim.SGD(parameters, lr=0.01)

    model = model.cuda() #enable GPU operations

    for epoch in range(epochs):
        start = time.time()
        correct = 0
        iterations = 0
        iteration_loss = 0.0
```

4. Input images and labels are wrapped up in variables and train function is called. Optimiser is set to zero gradient and forward and backward propagation is done where the modified loss functions and weights are updated.

```
# predict classes using images from the training set
#forward propogation
    outputs = model(inputs)

    #Calculating loss based on model output and real labels
    loss = loss_func(outputs, labels)
    iteration_loss += loss.item()

    #Backpropagation
    loss.backward()
    optimizer.step()

    # Record the correct predictions for training data
    _, predicted = torch.max(outputs, 1)
    # adjust parameters based on the calculated gradients
    correct += (predicted == labels).sum()
    iterations += 1


 train_lossfunc.append(iteration_loss/iterations)
 train_accuracy.append((100 * correct / len(training_set)))
```

5. Validation set is evaluated the same way as the training set using the codes below and the model is saved.

```
#No_grad on Val_set
with torch.no_grad():
    for i, (inputs, labels) in enumerate(validation_load, 0):

        inputs = Variable(inputs)
        labels = Variable(labels)

        #To Cuda()
        inputs = inputs.cuda()
        labels = labels.cuda()

        #Forward and Caculating loss
        outputs = model(inputs)
        loss = loss_func(outputs, labels)
        loss += loss.item()

        # Record the correct predictions for val data
        _, predicted = torch.max(outputs, 1)
        correct += (predicted == labels).sum()
        iterations += 1
```

```
        stop = time.time()

        print ('Epoch {}/{}, Training Loss: {:.3f}, Training Accuracy: {:.3f}, Val Loss: {:.3f}, Val Accuracy: {:.3f}, Tim
            .format(epoch+1, epochs, train_lossfunc[-1], train_accuracy[-1], val_lossfunc[-1], val_accuracy[-1],stop-start

epochs = 32
Training_Model(model=model, epochs=epochs, parameters=model.parameters())

#Save model
torch.save(model.state_dict(), 'weights/Face-Mask-Model.pt')
```

3. Model :

1. We imported libraries like numpy , torch.nn for neural network torchvision.transform for performing image transforms and convolution. Label IDs are also assigned.

```
import numpy as np
import time
import torch
import torch.nn as nn
import torchvision.transforms as transforms

from torch.autograd import Variable
#perform transformation for CNN
data_transform = transforms.Compose([transforms.Resize((32,32)),
                                     transforms.RandomHorizontalFlip(),
                                     transforms.ToTensor(),
                                     transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])]) # imagenet's para

label2id = {
    0: 'No Mask',
    1: 'Mask'
}
```

2. We created CNN model of 4 layers and 2 layers of max pooling with RELU and Kernel size of 3x3 with 3 RGB channels.

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()

        self.conv = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),

            nn.Conv2d(in_channels=32, out_channels=32, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),  #16

            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),

            nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2)   #8
        )
```

3. Dropout layer of 0.5 is added to avoid overfitting and the output layers are defined.

```
        self.fc = nn.Sequential(
            nn.Linear(in_features=4096, out_features=1024),
            nn.ReLU(),
            nn.Dropout(p=0.5),

            nn.Linear(in_features=1024, out_features=2)
        )

    def forward(self, x):
        x = self.conv(x)
        x = x.view(-1, 4096)
        out = self.fc(x)

        return out
```

4. Testing :

1. We import libraries CV2 to get the inbuilt functions for adding bounding box , argparse to parse the arguments and identify the errors associated with arguments, PIL library to import image processing functions and utility file.
2. We use cv2.putText and cv2.rectangle functions to get the bounding box.
3. To standardize the images, we call the transforms.

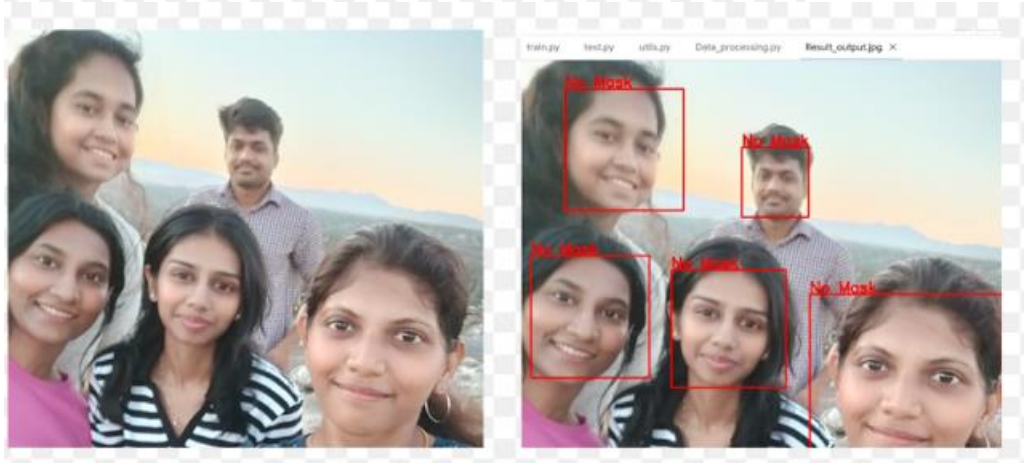
```
import cv2
import argparse

from PIL import Image
from utils import *

#to add bounding box and text
def putface(image, face, x, y, w, h):
    if(face == 'No Mask'):
        cv2.putText(image, face, (x, y-8), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 0, 255), 2)
        cv2.rectangle(image, (x, y-5), (x+w, y+h), (0,0,255), 2)
    elif(face == 'Mask'):
        cv2.putText(image, face, (x, y-8), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 255, 0), 2)
        cv2.rectangle(image, (x, y-5), (x+w, y+h), (0,255,0), 2)

def predict(image_detect, model):
    image_detect = cv2.resize(image_detect, (32, 32)) #Resize 32x32
    image = Image.fromarray(image_detect)

    image = data_transform(image)
    image = image.view(1, 3, 32, 32) #View in tensor
    image = Variable(image)

```

4. While testing, we call the model CNN() to perform the testing and later capture the output image and process the same in cascade classifier and add the bounding boxes.

```
#Load model
model = CNN()
model = model.cuda()
model.load_state_dict(torch.load('weights/Face-Mask-Model.pt'))

if(detectface.mode == "Image"):
    #Load haarlike feature
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades  + 'haarcascade_frontalface_default.xml')

    #Detect face
    img = cv2.imread(detectface.path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)

    for (x,y,w,h) in faces:
        image2 = image[y+2:y+h-2, x+2:x+w-2]
        emo = predict(image2, model)  #face index
        face = label2id[emo.item()]
        putface(image, face, x, y, w, h)

    cv2.imwrite("Result_output.jpg", image)
else:
  pass
```

4. EXPERIMENTAL RESULTS :

Without Mask detection :



With mask Detection:

5. CONCLUSION

We faced challenges in collecting diverse dataset. When we previously trained our data with readily available datasets, the prediction for people with masks were not accurate due to difference in brightness levels and skin tone of people. We then had to diversify the dataset to suit for people of different skin tones. Further improvements can be made to this tool by live web cam detection of masks and also improving its efficiency by training and testing on larger datasets.

6. REFERENCES

6.1. Shilpa Sethi.A, Mamta Kathuria.A, and Trilok Kaushik.b Face mask detection using deep learning: An approach to reduce risk of Coronavirus spread – In National Center for Biotechnology Information – 2021

6.2. P.Viola;M.JonesRapid_object_detection_using_a_boosted_cascade_of_simple_feature s-2001 (IEEE)

6.3. https://www.kaggle.com/code/fanconic/cnn-for-skin-cancer-detection