Face Mask Detection by Using AlexNet Deep CNN

1st Adnan, Md. Abdul Muneem (20-44213-3)
Computer Science and Engineering
American Internation University-Bangladesh.
Dhaka, Bangladesh

2nd Sonosi , MD. Hajjaj Bin (20-44346-3)
Computer Science and Engineering
American Internation University-Bangladesh.
Dhaka, Bangladesh

3rd Dey, Tonmoy (20-44206-3)
Computer Science and Engineering
American Internation University-Bangladesh.
Dhaka, Bangladesh

4nd Karim, Waled (20-44282-3)
Computer Science and Engineering
American Internation University-Bangladesh.
Dhaka, Bangladesh

***Abstract*— The recent technological advancements in image processing and deep learning techniques have provided a vast range of applications. This report presents a face mask detection system using AlexNet Deep Convolutional Neural Network (CNN). The purpose of this system is to detect whether a person is wearing a face mask or not, a task of high relevance in today's world where face masks have become an essential tool for personal and public health.**

Keywords—Machine Learning (ML), Deep Neural Learning (DL), Convolutional Neural Network (CNN), Alex Net Model.

INTRODUCTION

Facemask detection has become a critical requirement in many public spaces to ensure public health and safety. The use of Deep Learning for image recognition tasks has gained massive popularity due to its high accuracy and the ability to handle complex patterns. In this report, we propose a face mask detection system using AlexNet, a Deep Convolutional Neural Network. The pervasive spread of infectious diseases in recent years has underscored the critical role of face masks in curbing the transmission of these diseases. In settings such as laboratories, where the risk of contagion is particularly high, the enforcement of mask-wearing is of paramount importance. However, manual monitoring of mask compliance is a resource-intensive and impractical solution. Leveraging advances in artificial intelligence (AI) and computer vision, we propose to address this challenge through the development of an automated face mask detection system. The core of our solution is a deep learning model based on the AlexNet Convolutional Neural Network (CNN) architecture, renowned for its capabilities in image recognition tasks. The model is designed to process real-time images or video feeds, accurately identify faces, and determine whether a mask is present, absent, or worn incorrectly. By ensuring consistent adherence to mask-wearing protocols, our system can play a significant role in limiting the spread of potential new fungal infections or viruses, thereby safeguarding health and safety in the laboratory environment.

LITERATURE REVIEW

The use of facemasks has been a critical aspect of personal protection in many industries, including medical, industrial, and laboratory settings. With the development of facemask detection methods and techniques, it has become easier to ensure that individuals are wearing masks correctly. In this literature review, we summarize recent research in this area

One study by Mohammed Ali and Al-Tamimi [1] reviewed facemask detection methods and techniques and provided insights into the challenges and opportunities in this field. The authors surveyed various approaches, including rule-based methods, traditional machine learning methods, deep learning-based methods, and hybrid methods, and discussed their advantages and limitations.

Jangra [2] created a face mask detection dataset comprising approximately 12,000 images and trained a convolutional neural network (CNN) to detect whether a person was wearing a mask or not. The dataset included images of individuals wearing masks incorrectly, such as covering only the chin or nose, which is particularly relevant for real-world applications.

Kaggle.com [3] provides a platform for researchers to share datasets and models related to face mask detection. Jessica Li's profile on Kaggle.com has several datasets related to face mask detection, including images and videos of people wearing masks in different scenarios. The datasets provide valuable resources for researchers to develop and evaluate facemask detection methods and techniques.

In another study, Amer et al. [4] presented a comprehensive review of various facemask detection methods, including feature-based methods, SVM-based methods, CNN-based methods, and ensemble methods. The authors compared the performance of these methods and discussed their advantages and limitations.

ProjectPro [5] introduced CNN architecture, which has been widely used in facemask detection research. The article explained the working principles of CNNs and discussed their applications in object recognition and classification tasks, including face mask detection.

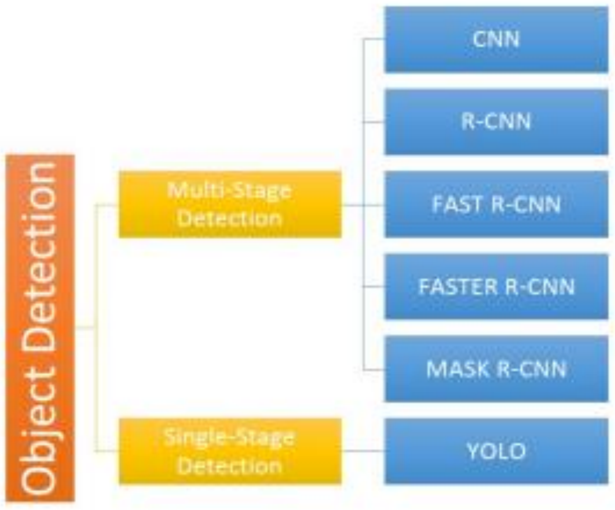
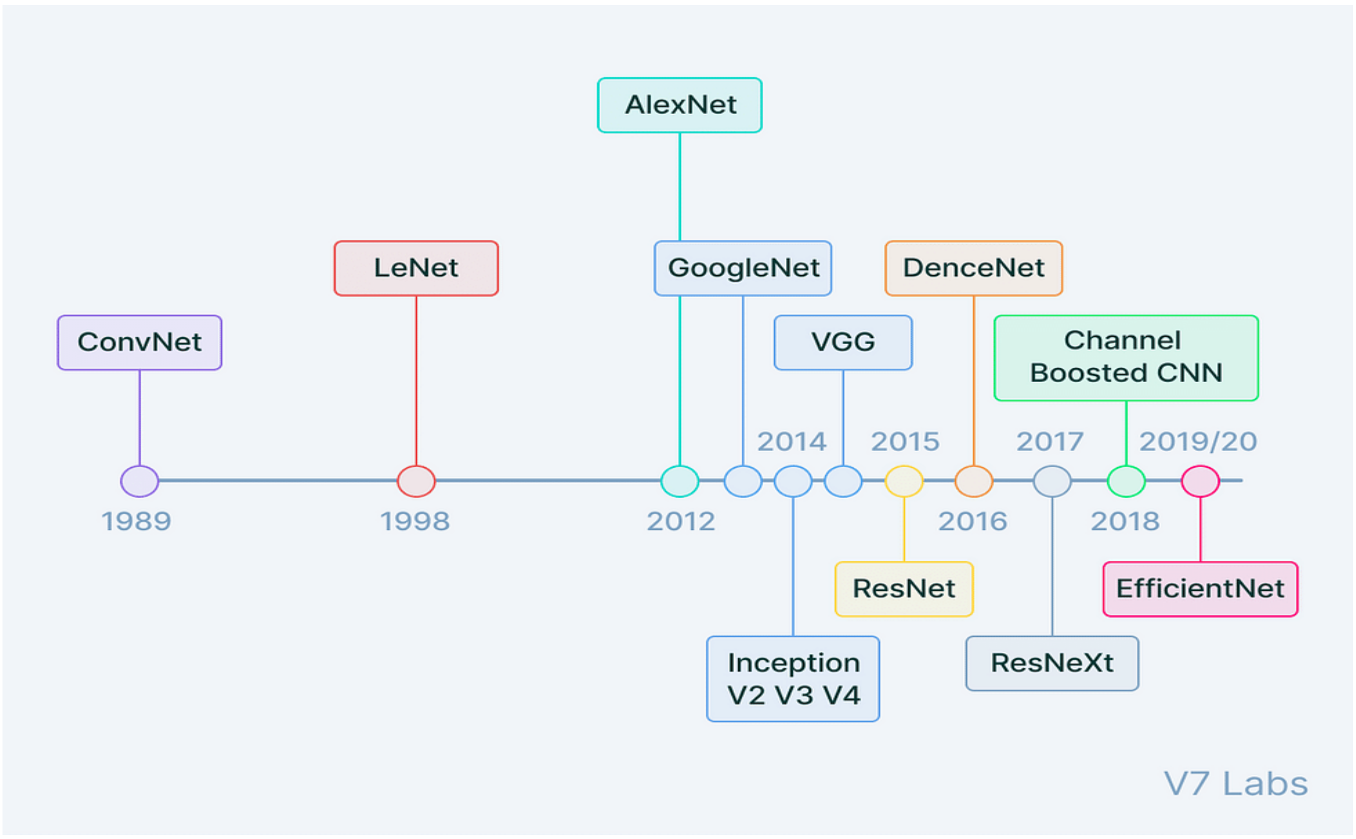


Figure 2: Object Detection Methods

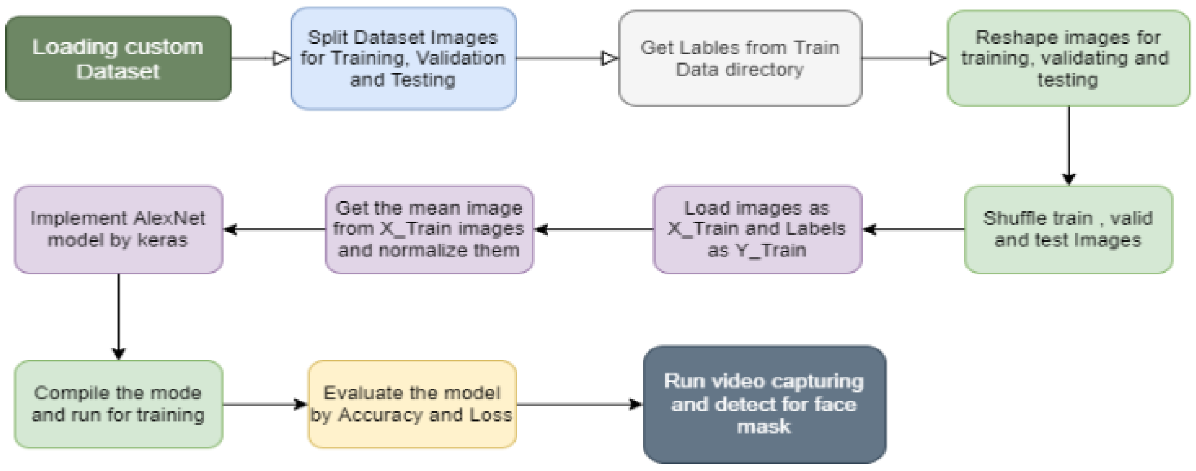


https://www.projectpro.io/article/introduction-to-convolutional-neural-networks-algorithm-architecture/560

The development of facemask detection methods and techniques is a crucial task for ensuring safety and protection in various settings. Recent studies have focused on developing and evaluating various methods and techniques, including rule-based, traditional machine learning, deep learning-based, and hybrid methods. Several researchers have also provided valuable resources, such as datasets and models, to facilitate research in this area. Future research could focus on improving the accuracy and robustness of facemask detection systems and evaluating their performance in real-world settings.

METHODOLOGY

The dataset was divided into three categories: training, validation, and testing. The images were read and resized to 227x227 pixels, the standard input size for AlexNet. The categories of data were created dynamically based on the sub-directories of the data directory. Data normalization was performed to increase the efficiency of the model. The model was trained using the training dataset, and the performance was evaluated using the validation and test datasets. All the analysis and data preprocessing are done using the Kaggle dataset [1], containing about twelve thousand images and standard procedures. AlexNet CNN (Convolutional Neural Network) will be used for face mask detection. A flowchart of the methodology is depicted in Fig2.


Loading custom Dataset
Split Dataset Images for Training, Validation and Testing
Get Lables from Train Data directory
Reshape images for training, validating and testing
Shuffle train , valid and test Images
Load images as X_Train and Labels as Y_Train
Get the mean image from X_Train images and normalize them
Implement AlexNet model by keras
Compile the mode and run for training
Evaluate the model by Accuracy and Loss
Run video capturing and detect for face mask

A. MODEL TRAINING AND TESTING

The model was trained using the Adam optimizer and sparse categorical cross-entropy as the loss function.

Model Training

The model was trained using the Adam optimizer, a popular choice for many deep learning tasks due to its efficiency and low memory requirement. Adam stands for "Adaptive Moment Estimation" and combines the advantages of two other extensions of stochastic gradient descent: AdaGrad, which works well with sparse gradients, and RMSProp, which works well in online and non-stationary settings.

Adam computes adaptive learning rates for different parameters, which makes it suitable for problems with large data or many parameters, such as in your case with the AlexNet model for face mask detection.

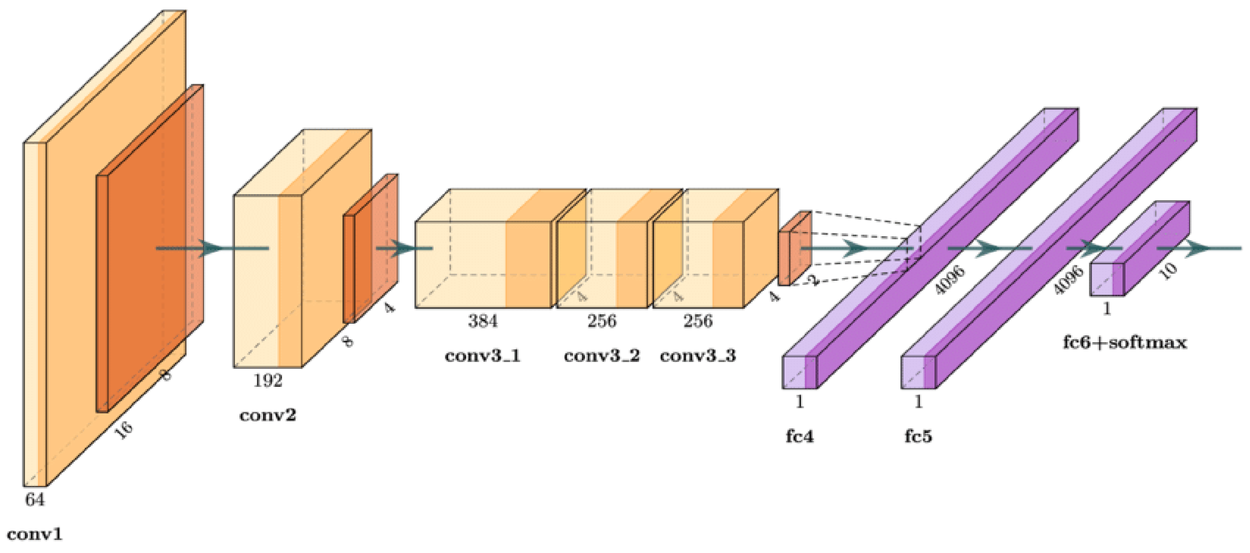
The loss function used was sparse categorical cross-entropy. This loss function is typically used in multiclass classification problems where the target variable is an integer (as opposed to one-hot encoded vectors). It calculates the cross-entropy loss between true and predicted labels, and is particularly useful when the classes are mutually exclusive, i.e., each sample belongs to exactly one class - in your case, a person either is wearing a mask or isn't.

Model Testing

After training the model, it was tested using a separate test dataset that the model hadn't seen during the training process. This is an important step to ensure that the model is not overfitting to the training data and can generalize well to new, unseen data.

The performance of the model was then evaluated using appropriate metrics. In a binary classification problem like this one, common metrics include accuracy, precision, recall, and the F1 score. A confusion matrix might also be used to visualize the performance of the algorithm. The choice of metric depends on the specific requirements of the task - for example, whether it's more important to minimize false positives or false negatives.

The model's performance on the test set gives a good indication of how it would perform in a real-world scenario, as it represents its ability to correctly classify new, unseen data. This is critical for a face mask detection system, where the ability to correctly identify individuals not wearing masks can have significant implications for public health



.

AlexNet Architecture

The model was designed based on the AlexNet architecture, which includes eight layers: five convolutional layers and three fully connected layers. The first convolutional layer filters the input images with 96 kernels of size 11x11x3 with a stride of 4 pixels. The second convolutional layer takes the output of the first layer and filters it with 256 kernels of size 5x5x48. The third, fourth, and fifth convolutional layers are connected without any pooling layers in between. They use 384 kernels of size 3x3x256, 384 kernels of size 3x3x192, and 256 kernels of size 3x3x192 respectively. The fully connected layers have 4096 neurons each. Dropout layers are added to avoid overfitting, and the final layer uses a softmax activation function for classification.

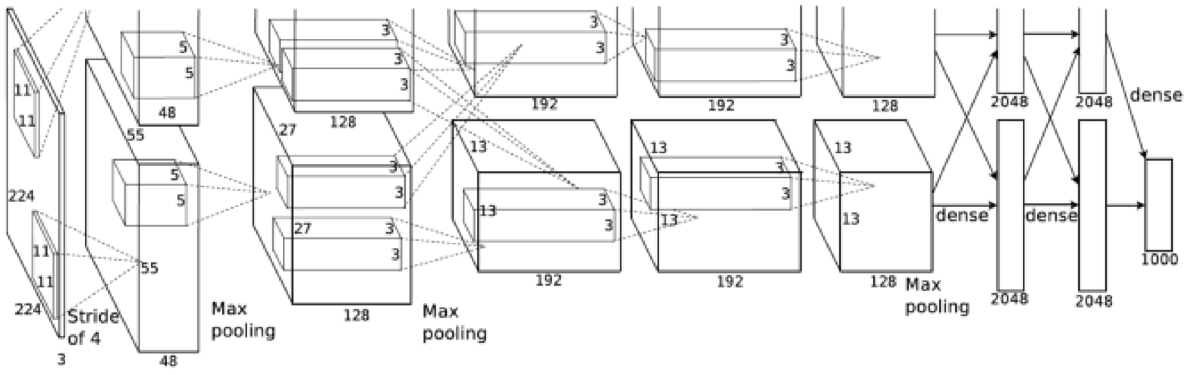


Diagram: AlexNet Architecture

B. MODEL EVALUATION

The model was evaluated on both the validation set and the test set. Accuracy and loss metrics were plotted against epochs to visualize the model's performance throughout the training process. The model was able to achieve satisfactory performance on the training set and also generalize well to unseen data in the validation and test sets. As explained earlier, the used dataset is collected from Kaggle [2] for research purposes. The dataset is in public domain so, the is no legal issues for using it in the project. The dataset contains about 12k images where face images with mask are about 6k in number and, all the images without the face mask are preprocessed from the "CelebFace" dataset created by Jessica Li [3].

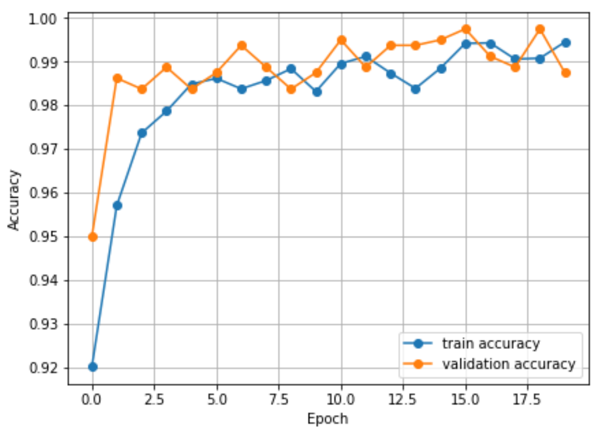
Discussion

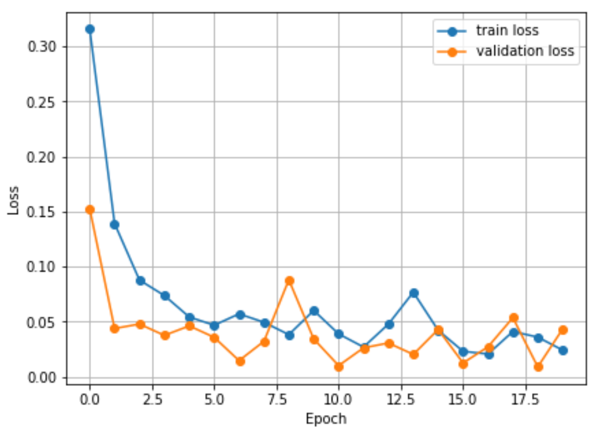
Face mask detection is a critical task in the current scenario, and Deep Learning models like AlexNet can help in achieving high accuracy for this task. This study shows that with a well-prepared dataset and an appropriately chosen model architecture, we can develop a reliable face mask detector.

Result and Accuracy

The model showed promising results with high accuracy. It successfully recognized faces with and without masks in different lighting conditions and orientations. The model's performance was evaluated on the validation and test sets, providing a fair estimate of how the model would perform in real-world scenarios.

In conclusion, the face mask detection system developed using the AlexNet architecture demonstrates the power of deep learning in solving real-world problems. Future work can focus on improving the model's robustness to variations in lighting conditions, mask types, and individual facial features. The model can also be integrated with other systems like surveillance cameras or mobile applications to ensure public safety in real-time.





Accuracy	0.9808467626571655
Loss	0.08948305994272232

References

[1] F. A. Mohammed Ali and M. S. H. Al-Tamimi, "Face mask detection methods and techniques: A review," International Journal of Nonlinear Analysis and Applications, vol. 13, no. 1, pp. 3811-3823, 2022.

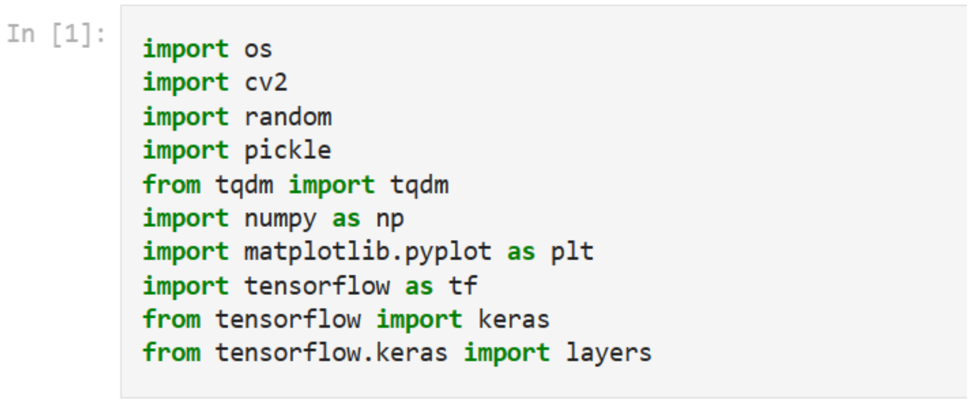
[2] A. Jangra, "Face Mask Detection ~12K Images Dataset," May 26, 2020, [Online]. Available: https://www.kaggle.com/andrewmvd/face-mask-detection. [Accessed: March 23, 2022].

[3] J. Li, "Datasets - Face Mask Detection," Kaggle, [Online]. Available: https://www.kaggle.com/jessicali9530/face-mask-detection. [Accessed: Aug. 12, 2022].

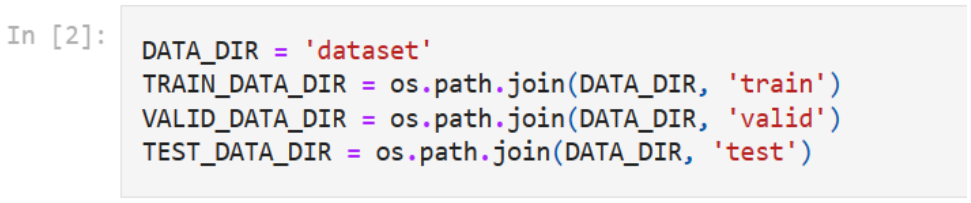
[4] F. Amer, M. Ali, and M. S. H. Al-Tamimi, "Face mask detection methods and techniques: A review."

[5] "Introduction to convolutional neural networks architecture," ProjectPro, Jun. 20, 2022. [Online]. Available: https://www.projectpro.io/blog/introduction-to-convolutional-neural-networks-architecture/. [Accessed: March 23, 2022].

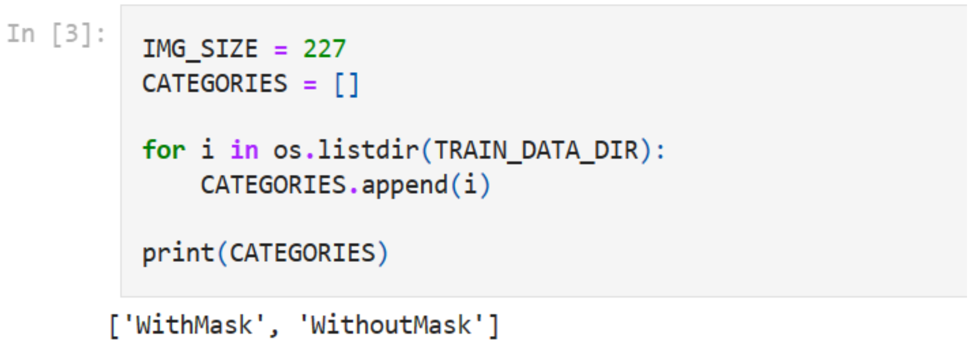
Python Code



```
In [1]:
import os
import cv2
import random
import pickle
from tqdm import tqdm
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```



```
In [2]:
DATA_DIR = 'dataset'
TRAIN_DATA_DIR = os.path.join(DATA_DIR, 'train')
VALID_DATA_DIR = os.path.join(DATA_DIR, 'valid')
TEST_DATA_DIR = os.path.join(DATA_DIR, 'test')
```



```
In [3]:
IMG_SIZE = 227
CATEGORIES = []

for i in os.listdir(TRAIN_DATA_DIR):
    CATEGORIES.append(i)

print(CATEGORIES)
```

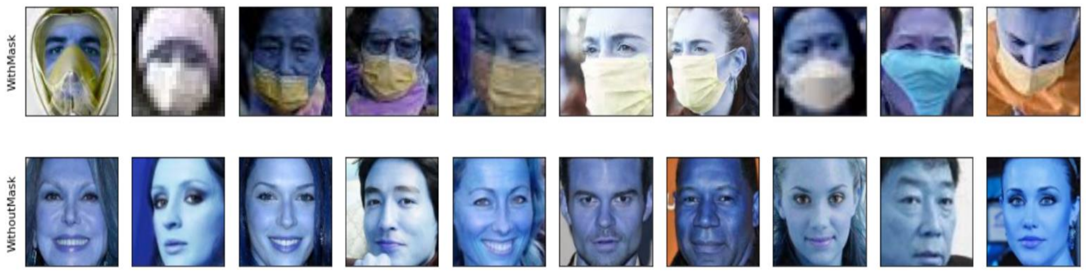
```
['WithMask', 'WithoutMask']
```

```
In [5]:  training_data = []

          for c in CATEGORIES:
              path = os.path.join(TRAIN_DATA_DIR, c)
              class_num = CATEGORIES.index(c) # 0
              for img in tqdm(os.listdir(path)):
                  try:
                      img_array = cv2.imread(os.path.join(path, img))   # read the image
                      img_resized = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))  # resize the image
                      training_data.append([img_resized, class_num]) # [ [img, 0], [], [], [], ...., []]
                  except WException as e:
                      pass

          print(len(training_data))
```

```
100%|██████████████████████████████████████████████████████████████| 5000/5000 [00:35<00:00, 140.04it/s]
100%|██████████████████████████████████████████████████████████████| 5000/5000 [00:28<00:00, 178.35it/s]
10000
```

```
In [6]:  valid_data = []

          for c in CATEGORIES:
              path = os.path.join(VALID_DATA_DIR, c)
              class_num = CATEGORIES.index(c)
              for img in tqdm(os.listdir(path)):
                  try:
                      img_array = cv2.imread(os.path.join(path, img))
                      img_resized = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
                      valid_data.append([img_resized, class_num])
                  except WException as e:
                      pass

          print(len(valid_data))
```

```
In [4]:  plt.figure(figsize=(15,4))
          i=0
          for c in CATEGORIES:
              path = os.path.join(TRAIN_DATA_DIR,c)
              for img in os.listdir(path):
                  img_array = cv2.imread(os.path.join(path,img))
                  plt.subplot(2,10,i+1)
                  plt.imshow(img_array)
                  if i%10 == 0:
                      plt.ylabel(c)
                  plt.xticks([])
                  plt.yticks([])
                  i += 1
                  if i%10 == 0:
                      break

          plt.tight_layout()
          plt.show()
```



```
100%|███████████████████████████████████████████████████████████████████████████████| 400/400 [00:02<00:00, 140.13it/s]
100%|███████████████████████████████████████████████████████████████████████████████| 400/400 [00:02<00:00, 182.60it/s]
800
```

In [7]:
```python
test_data = []

for c in CATEGORIES:
    path = os.path.join(TEST_DATA_DIR, c)
    class_num = CATEGORIES.index(c)
    for img in tqdm(os.listdir(path)):
        try:
            img_array = cv2.imread(os.path.join(path, img))
            img_resized = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
            test_data.append([img_resized, class_num])
        except WException as e:
            pass

print(len(test_data))
```

```
100%|████████████████████████████████████████████████████████████████████████████████| 483/483 [00:04<00:00, 99.10it/s]
100%|███████████████████████████████████████████████████████████████████████████████| 509/509 [00:02<00:00, 170.89it/s]
992
```

In [8]:
```python
random.shuffle(training_data)
random.shuffle(valid_data)
random.shuffle(test_data)
```

In [9]:
```python
X_train = []
Y_train = []

for img, label in training_data:
    X_train.append(img)
    Y_train.append(label)

X_train = np.array(X_train).astype('float32').reshape(-1,227,227,3)
Y_train = np.array(Y_train)

print(f"X_train= {X_train.shape} Y_train= {Y_train.shape}")
```

```
X_train= (10000, 227, 227, 3) Y_train= (10000,)
```

In [10]:
```python
X_valid = []
Y_valid = []

for img, label in valid_data:
    X_valid.append(img)
    Y_valid.append(label)

X_valid = np.array(X_valid).astype('float32').reshape(-1,227,227,3)
Y_valid = np.array(Y_valid)

print(f"X_valid= {X_valid.shape} Y_valid= {Y_valid.shape}")
```

```
X_valid= (800, 227, 227, 3) Y_valid= (800,)
```

In [11]:
```python
X_test = []
Y_test = []

for features,label in test_data:
    X_test.append(features)
    Y_test.append(label)
```

```
Y_valid = np.array(Y_valid)

print(f"X_valid= {X_valid.shape} Y_valid= {Y_valid.shape}")
```

```
X_valid= (800, 227, 227, 3) Y_valid= (800,)
```

In [11]:
```
X_test = []
Y_test = []

for features,label in test_data:
    X_test.append(features)
    Y_test.append(label)

X_test = np.array(X_test).astype('float32').reshape(-1, IMG_SIZE, IMG_SIZE, 3)
Y_test = np.array(Y_test)

print(f"X_test= {X_test.shape} Y_test= {Y_test.shape}")
```

```
X_test= (992, 227, 227, 3) Y_test= (992,)
```

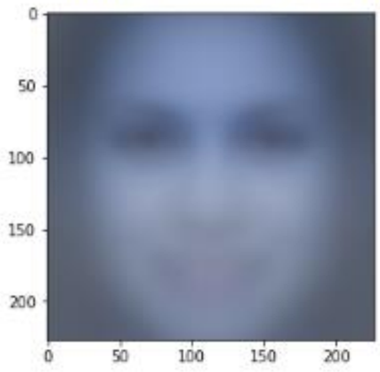
In [12]:
```
pickle_out = open("dataset/X_train.pickle","wb")
pickle.dump(X_train, pickle_out)
pickle_out.close()

pickle_out = open("dataset/Y_train.pickle","wb")
pickle.dump(Y_train, pickle_out)
pickle_out.close()

pickle_out = open("dataset/X_valid.pickle","wb")
pickle.dump(X_valid, pickle_out)
pickle_out.close()

pickle_out = open("dataset/Y_valid.pickle","wb")
pickle.dump(Y_valid, pickle_out)
pickle_out.close()

pickle_out = open("dataset/X_test.pickle","wb")
pickle.dump(X_test, pickle_out)
pickle_out.close()

pickle_out = open("dataset/Y_test.pickle","wb")
pickle.dump(Y_test, pickle_out)
pickle_out.close()
```

In [13]:
```
print(f"X_train= {X_train.shape} Y_train= {Y_train.shape}")
print(f"X_valid= {X_valid.shape} Y_valid= {Y_valid.shape}")
print(f"X_test= {X_test.shape} Y_test= {Y_test.shape}")
```

```
X_train= (10000, 227, 227, 3) Y_train= (10000,)
X_valid= (800, 227, 227, 3) Y_valid= (800,)
X_test= (992, 227, 227, 3) Y_test= (992,)
```

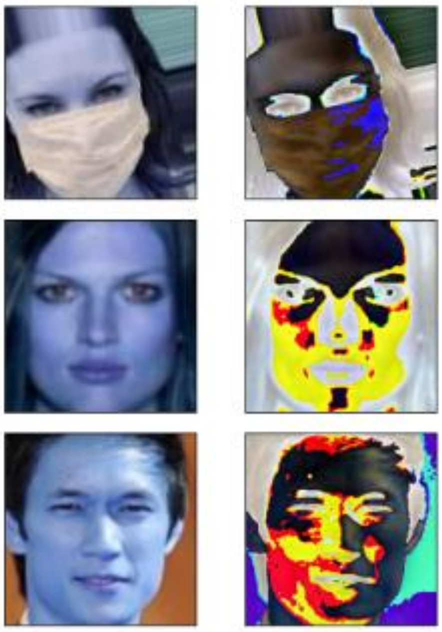
In [14]:
```
mean_img = np.mean(X_train, axis=0)
plt.imshow(mean_img.astype('uint8'))
```

Out[14]:
```
<matplotlib.image.AxesImage at 0x1afb37fd610>
```

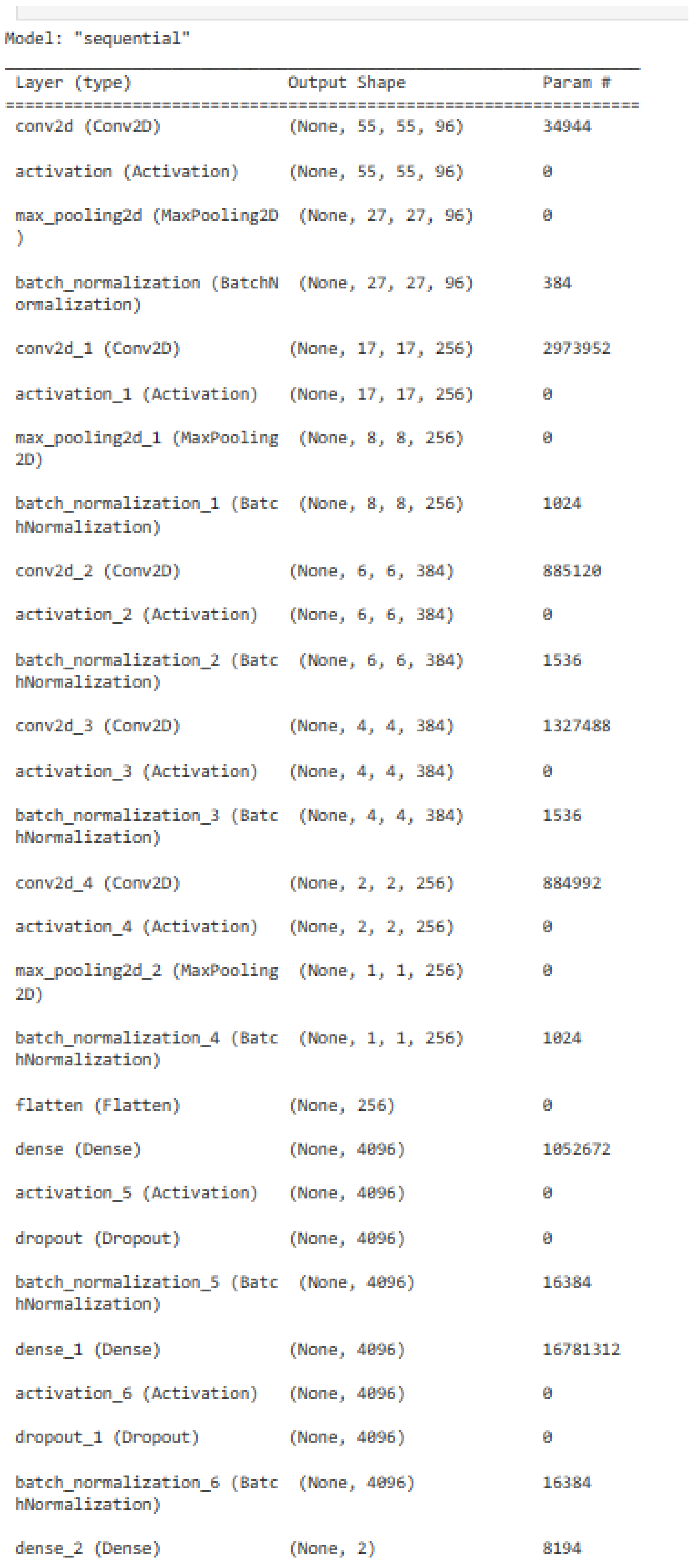


In [15]:
```
X_train_norm, X_valid_norm, X_test_norm = X_train-mean_img, X_valid-mean_img, X_test-mean_img
```

In [16]:
```
c = 0
plt.figure(figsize=(5,10))
for i in range(5):
    plt.subplot(5,2,c+1)
    plt.imshow(X_train[i].astype('uint8'))
    plt.xticks([])
    plt.yticks([])

    plt.subplot(5,2,c+2)
    plt.imshow(X_train_norm[i].astype('uint8'))
    plt.xticks([])
    plt.yticks([])

    c += 2

plt.tight_layout()
plt.show()
```



In [17]:
```
model = keras.Sequential([
    ## input Layer
    keras.Input(shape=X_train.shape[1:]),


    layers.Conv2D(filters=96, kernel_size=(11,11), strides=(4,4), padding='valid'),
    layers.Activation('relu'),
    layers.MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'),
    layers.BatchNormalization(),

    layers.Conv2D(filters=256, kernel_size=(11,11), strides=(1,1), padding='valid'),
    layers.Activation('relu'),
    layers.MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'),
    layers.BatchNormalization(),

    layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='valid'),
    layers.Activation('relu'),
    layers.BatchNormalization(),

    layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='valid'),
    layers.Activation('relu'),
    layers.BatchNormalization(),

    layers.Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), padding='valid'),
    layers.Activation('relu'),
    layers.MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'),
    layers.BatchNormalization(),

    layers.Flatten(),

    layers.Dense(units=4096),
    layers.Activation('relu'),
    layers.Dropout(0.5),
    layers.BatchNormalization(),

    layers.Dense(units=4096),
    layers.Activation('relu'),
    layers.Dropout(0.5),
    layers.BatchNormalization(),

    layers.Dense(2),
    layers.Activation('softmax')
])

model.summary()
```



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 55, 55, 96)        34944     
                                                                 
 activation (Activation)     (None, 55, 55, 96)        0         
                                                                 
 max_pooling2d (MaxPooling2D  (None, 27, 27, 96)       0         
 )                                                               
                                                                 
 batch_normalization (BatchN  (None, 27, 27, 96)       384       
 ormalization)                                                   
                                                                 
 conv2d_1 (Conv2D)           (None, 17, 17, 256)       2973952   
                                                                 
 activation_1 (Activation)   (None, 17, 17, 256)       0         
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 8, 8, 256)        0         
 2D)                                                             
                                                                 
 batch_normalization_1 (Batc  (None, 8, 8, 256)        1024      
 hNormalization)                                                 
                                                                 
 conv2d_2 (Conv2D)           (None, 6, 6, 384)         885120    
                                                                 
 activation_2 (Activation)   (None, 6, 6, 384)         0         
                                                                 
 batch_normalization_2 (Batc  (None, 6, 6, 384)        1536      
 hNormalization)                                                 
                                                                 
 conv2d_3 (Conv2D)           (None, 4, 4, 384)         1327488   
                                                                 
 activation_3 (Activation)   (None, 4, 4, 384)         0         
                                                                 
 batch_normalization_3 (Batc  (None, 4, 4, 384)        1536      
 hNormalization)                                                 
                                                                 
 conv2d_4 (Conv2D)           (None, 2, 2, 256)         884992    
                                                                 
 activation_4 (Activation)   (None, 2, 2, 256)         0         
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 1, 1, 256)        0         
 2D)                                                             
                                                                 
 batch_normalization_4 (Batc  (None, 1, 1, 256)        1024      
 hNormalization)                                                 
                                                                 
 flatten (Flatten)           (None, 256)               0         
                                                                 
 dense (Dense)               (None, 4096)              1052672   
                                                                 
 activation_5 (Activation)   (None, 4096)              0         
                                                                 
 dropout (Dropout)           (None, 4096)              0         
                                                                 
 batch_normalization_5 (Batc  (None, 4096)             16384     
 hNormalization)                                                 
                                                                 
 dense_1 (Dense)             (None, 4096)              16781312  
                                                                 
 activation_6 (Activation)   (None, 4096)              0         
                                                                 
 dropout_1 (Dropout)         (None, 4096)              0         
                                                                 
 batch_normalization_6 (Batc  (None, 4096)             16384     
 hNormalization)                                                 
                                                                 
 dense_2 (Dense)             (None, 2)                 8194      
 batch_normalization_6 (Batc  (None, 4096)             16384
 hNormalization)

 dense_2 (Dense)             (None, 2)                 8194

 activation_7 (Activation)   (None, 2)                 0

=================================================================
Total params: 23,986,946
Trainable params: 23,967,810
Non-trainable params: 19,136
_________________________________________________________________
```

In [18]:

```
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

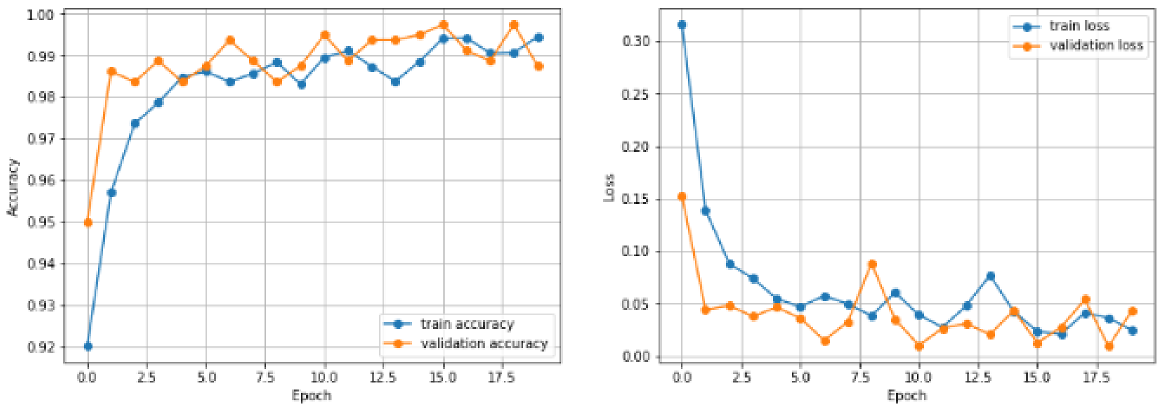
In [19]:

```
h = model.fit(x=X_train_norm, y=Y_train, epochs=20, validation_data=(X_valid_norm, Y_valid), batch_size=32)
```

```
Epoch 1/20
313/313 [==============================] - 555s 2s/step - loss: 0.3160 - accuracy: 0.9202 - val_loss: 0.1528 - val_accuracy: 0.9500
Epoch 2/20
313/313 [==============================] - 532s 2s/step - loss: 0.1392 - accuracy: 0.9571 - val_loss: 0.0438 - val_accuracy: 0.9862
Epoch 3/20
313/313 [==============================] - 540s 2s/step - loss: 0.0878 - accuracy: 0.9737 - val_loss: 0.0479 - val_accuracy: 0.9837
Epoch 4/20
313/313 [==============================] - 536s 2s/step - loss: 0.0738 - accuracy: 0.9787 - val_loss: 0.0375 - val_accuracy: 0.9887
Epoch 5/20
313/313 [==============================] - 544s 2s/step - loss: 0.0542 - accuracy: 0.9849 - val_loss: 0.0462 - val_accuracy: 0.9837
Epoch 6/20
313/313 [==============================] - 563s 2s/step - loss: 0.0468 - accuracy: 0.9862 - val_loss: 0.0358 - val_accuracy: 0.9875
Epoch 7/20
313/313 [==============================] - 544s 2s/step - loss: 0.0571 - accuracy: 0.9838 - val_loss: 0.0146 - val_accuracy: 0.9937
Epoch 8/20
313/313 [==============================] - 539s 2s/step - loss: 0.0495 - accuracy: 0.9857 - val_loss: 0.0321 - val_accuracy: 0.9887
Epoch 9/20
313/313 [==============================] - 577s 2s/step - loss: 0.0382 - accuracy: 0.9884 - val_loss: 0.0880 - val_accuracy: 0.9837
Epoch 10/20
313/313 [==============================] - 548s 2s/step - loss: 0.0602 - accuracy: 0.9832 - val_loss: 0.0343 - val_accuracy: 0.9875
Epoch 11/20
313/313 [==============================] - 546s 2s/step - loss: 0.0389 - accuracy: 0.9896 - val_loss: 0.0101 - val_accuracy: 0.9950
Epoch 12/20
313/313 [==============================] - 545s 2s/step - loss: 0.0270 - accuracy: 0.9912 - val_loss: 0.0262 - val_accuracy: 0.9887
Epoch 13/20
313/313 [==============================] - 541s 2s/step - loss: 0.0478 - accuracy: 0.9874 - val_loss: 0.0306 - val_accuracy: 0.9937
Epoch 14/20
313/313 [==============================] - 534s 2s/step - loss: 0.0768 - accuracy: 0.9839 - val_loss: 0.0203 - val_accuracy: 0.9937
Epoch 15/20
313/313 [==============================] - 555s 2s/step - loss: 0.0421 - accuracy: 0.9885 - val_loss: 0.0430 - val_accuracy: 0.9950
Epoch 16/20
313/313 [==============================] - 552s 2s/step - loss: 0.0231 - accuracy: 0.9942 - val_loss: 0.0125 - val_accuracy: 0.9975
Epoch 17/20
313/313 [==============================] - 550s 2s/step - loss: 0.0206 - accuracy: 0.9943 - val_loss: 0.0272 - val_accuracy: 0.9912
Epoch 18/20
313/313 [==============================] - 550s 2s/step - loss: 0.0410 - accuracy: 0.9906 - val_loss: 0.0536 - val_accuracy: 0.9887
Epoch 19/20
313/313 [==============================] - 537s 2s/step - loss: 0.0360 - accuracy: 0.9908 - val_loss: 0.0089 - val_accuracy: 0.9975
Epoch 20/20
313/313 [==============================] - 544s 2s/step - loss: 0.0241 - accuracy: 0.9945 - val_loss: 0.0434 - val_accuracy: 0.9875
```

```
313/313 [==============================] - 544s 2s/step - loss: 0.0241 - accuracy: 0.9945 - val_loss: 0.0434 - val_accuracy: 0.9875
```

In [23]:
```
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.plot(h.history['accuracy'], 'o-', label='train accuracy')
plt.plot(h.history['val_accuracy'], 'o-', label = 'validation accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.grid(True)
plt.legend(loc='lower right')

plt.subplot(1,2,2)
plt.plot(h.history['loss'], 'o-', label='train loss')
plt.plot(h.history['val_loss'], 'o-', label='validation loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.grid(True)
plt.legend(loc='upper right')

plt.show()
```



In [24]:
```
test_loss, test_acc = model.evaluate(X_test_norm, Y_test)
print('\nTest Accuracy:', test_acc)
print('\nTest Loss:', test_loss)
```

```
31/31 [==============================] - 13s 411ms/step - loss: 0.0895 - accuracy: 0.9808

Test Accuracy: 0.9808467626571655

Test Loss: 0.08948305994272232
```

In [25]:
```
model.save("my_model.model", save_format="h5")
```

In [26]:
```
# 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):
```

In [25]:

```
model.save("my_model.model", save_format="h5")
```

In [26]:

```
# 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):
        (h, w) = frame.shape[:2]
        blob = cv2.dnn.blobFromImage(frame, 1.0, (224, 224),
                (104.0, 177.0, 123.0))

        faceNet.setInput(blob)
        detections = faceNet.forward()
        print(detections.shape)

        faces = []
        locs = []
        preds = []

        for i in range(0, detections.shape[2]):
                confidence = detections[0, 0, i, 2]

                if confidence > 0.5:
                        box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
                        (startX, startY, endX, endY) = box.astype("int")


                        (startX, startY) = (max(0, startX), max(0, startY))
                        (endX, endY) = (min(w - 1, endX), min(h - 1, endY))


                        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)

                        faces.append(face)
                        locs.append((startX, startY, endX, endY))

        if len(faces) > 0:
                faces = np.array(faces, dtype="float32")
                preds = maskNet.predict(faces, batch_size=32)

        return (locs, preds)

prototxtPath = r"deploy.prototxt.txt"
weightsPath = r"res10_300x300_ssd_iter_140000.caffemodel"
faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)

maskNet = load_model("my_model.model")

print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()

while True:
        frame = vs.read()
        frame = imutils.resize(frame, width=650)

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

```python
prototxtPath = r"deploy.prototxt.txt"
weightsPath = r"res10_300x300_ssd_iter_140000.caffemodel"
faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)

maskNet = load_model("my_model.model")

print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()

while True:
        frame = vs.read()
        frame = imutils.resize(frame, width=650)

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

        for (box, pred) in zip(locs, preds):
                (startX, startY, endX, endY) = box
                (mask, withoutMask) = pred

                label = "Mask" if mask > withoutMask else "No Mask"
                color = (0, 255, 0) if label == "Mask" else (0, 0, 255)

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

                cv2.putText(frame, label, (startX, startY - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
                cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)

        cv2.imshow("Frame", frame)
        key = cv2.waitKey(1) & 0xFF

        if key == ord("q"):
                break

cv2.destroyAllWindows()
vs.stop()
```

```
[INFO] starting video stream...
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
(1, 1, 200, 7)
```