(7088CEM)

Coursework
Artificial Neural Networks

MODULE LEADER: Dr. Marwan Fuad

Student Name: Bharti Tayal

SID: 10895366

Face Mask Detection: Classification using Convolutional Neural Network

I can confirm that all work submitted is my own: Yes

Introduction :

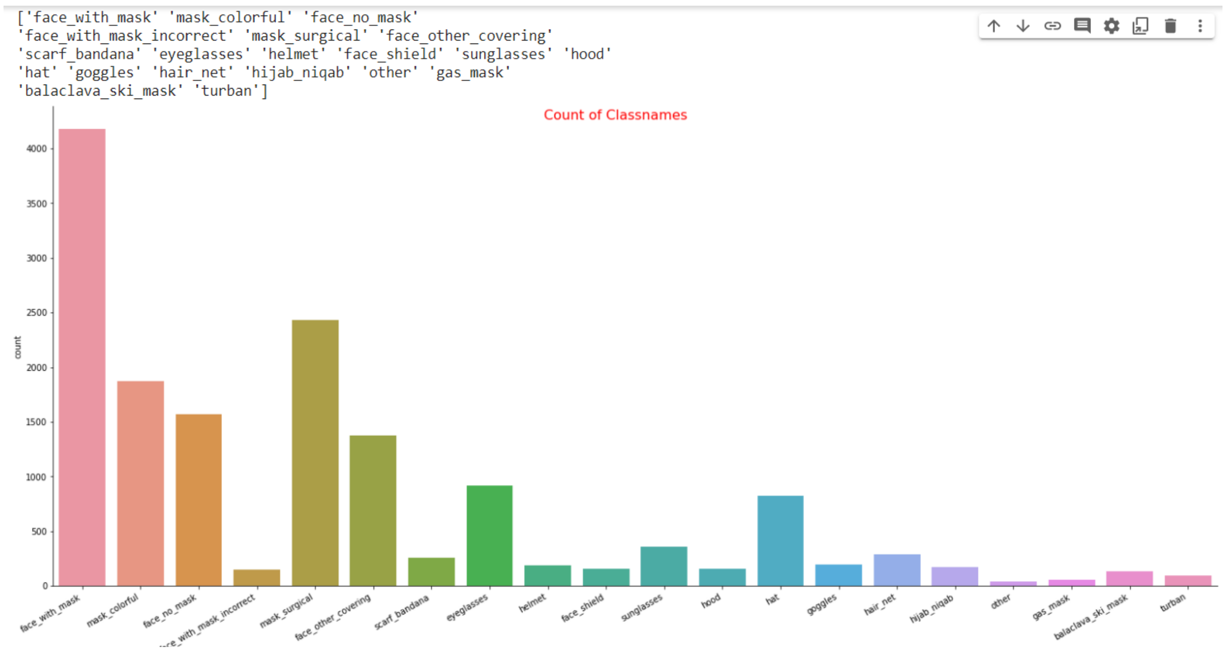
In 2020, the entire world was attacked with a pandemic which was COVID-19. According to World Health Organisation (WHO), there has been more than 247 million coronavirus cases leading to more than 5 million deaths worldwide as of October 2021. This disease has serious side effects on human body causing respiratory illness and kidney issues and many more. The worst part is there has not been any proper treatment which has been proved to control this disease. According to WHO report, The disease can be transmitted when person encounters the person suffering from covid-19. So, to control the flow of transmission, World Health Organisation has issued several guidelines to control the spread of the disease which includes Social Distancing, wearing masks, avoiding crowded places and many more. As per reports, Face covering has been proved useful step to avoid the spread. So, Face coverings were made mandatory in many countries worldwide; people were monitored, and severe actions were taken for those breaking the rules.

This system will detect if the person is wearing mask or not with the aid of Computer Vision. Computer vision enables machines to act and think like humans. It has been proved useful in many tasks such as Image and Video Recognition, Natural language Processing and many more. Computer Vision along with Deep Learning has many applications. Computer Vision is basically used to study and extract features of the subject using inputs provided and deep learning performs operations such as learning, prediction, classification and many more. In this system also, deep learning techniques will be used to detect the face masks.

Face Mask Detection is a classification problem which can be solved using Convolutional Neural Networks (CNN). CNNs have been proved useful for such problems. In this project, Classification and Clustering will be performed on the dataset containing images with people not wearing masks, some wearing different types of face coverings. The dataset has been chosen from an open source website, kaggle which is freely available for all. There are 20 classes in the database but in this project, classification and clustering will be done based on the 2 classes only ,i.e. , 'face_with_mask' and 'face_no_mask'.

```
classes = train_full["classname"].unique()
print("Total Classes: ",len(classes))
print("\n> Classes <\n",classes)

ax = sns.catplot(x='classname',kind='count',data=train_full,orient="h",height=10,aspect=2)
ax.fig.suptitle('Count of Classnames',fontsize=16,color="r")
ax.fig.autofmt_xdate()
```

Code Snippet1: Showing number of different classes in dataset



```
['face_with_mask' 'mask_colorful' 'face_no_mask'
 'face_with_mask_incorrect' 'mask_surgical' 'face_other_covering'
 'scarf_bandana' 'eyeglasses' 'helmet' 'face_shield' 'sunglasses' 'hood'
 'hat' 'goggles' 'hair_net' 'hijab_niqab' 'other' 'gas_mask'
 'balaclava_ski_mask' 'turban']
```

Fig1: Showing number of different classes in dataset

In this project, Classification is performed with the help of two different methods. A sequential CNN model is built initially which would be capable of classifying the images into two classes. Afterwards, a pre-trained VGG16 model is used to perform the same task. The results of both approaches would be compared based on their classification reports and accuracy curves. Another task to be performed on the dataset is Clustering. Clustering groups the images with similar characteristics in one cluster. Though these both tasks sound similar, but the difference is classnames would be given to the model in the former whilst for the latter, classnames would not be passed to the model. Pre-trained models save lot of computational cost especially in large datasets. So, Vgg16 is used for feature extraction which will then be passed to K-Means to form clusters. Two clusters are formed-‘Face_with_mask’ and ‘Face_no_mask’.

In this project, various python machine learning libraries are used. Tensorflow and OpenCV along with Keras API are maily used. Tensorflow is an end-to-end open source platform for deep learning. It offers so many libraries and resources along with tools that help to make deep learning projects and applications easily. It helps in training images, data augmentation, easy model building, etc. It has good community resources available. Being a beginner, these resources provide a lot of guidance in building the model. OpenCV, also known as Open Computer Vision library is another useful library which helps in deep learning and computer vision projects. It provides comprehensive tools for image processing such as reshaping and resizing images which is very helpful in this project. This project requires large RAM and due to massive size of dataset, GPU is preferred more over CPU. Observing the requirements of the project, This project has been run on Google Colab which offers GPU runtime and all the necessary libraries and packages support.

Related Work :

After this pandemic, the use of Computer Vision has been increased rapidly especially for face mask detection[2]. Different architectures of CNN can be used; there are many techniques present in CNN which are quite useful for extracting features and recognising face from an image[3]. Deep Learning techniques work well due to their high robustness. In this project, face mask would be detected by analysing various features of the face. According to [1], face mask detection depends on various features such as expression recognition, pose estimation, face tracking, and many more. Face detection is a quite daunting task as face of everyone varies with different shape, size,and color. [4] used YOLOv3 which is a single-stage architecture of CNN. It can predict bounding boxes and produces faster results but not accurate. The problem generally faced by people is the time consumed for training, feature extraction. Sometimes data is not sufficient enough for training. These problems can be resolved by using pre-trained models such as Vgg16, Vgg19, ResNet50, MobileNet and many more. [5] used different pre-existed models for the same dataset and obtained best results with Vgg16.

Problem :

As specified in the Introduction part, different images of different people are taken. The problem is to analyse if the person in the image is wearing mask or not. There are 6024 images in the dataset which is very huge dataset. Furthermore, annotations are also given which has json file format and contains the classes and bounding boxes of the images. There are two csv files present which are train.csv and submission.csv. Train.csv contains file names, bounding boxes and classes of the images which can be used for training data whereas submission.csv contains only the filenames. One has to predict bounding boxes and classes to which image belongs. There are 20 classes in the given dataset. I have extracted classes with two classnames which are ‘face_with_mask’ and ‘face_with_no_mask’. I will classify images into these two categories. Results of the classification will be stored in a csv file. Additionally, Clusters would be formed based on the similarity of characteristics in the images.

The dataset has been extracted from Kaggle; the link to the dataset is https://www.kaggle.com/wobotintelligence/face-mask-detection-dataset.

Methodology :

Convolutional Neural Network :

There are many types of neural networks available; having its own specialities and usage. It depends on the task which model is suitable; here, I have chosen Convolutional Neural network for my problem which is Image classification.

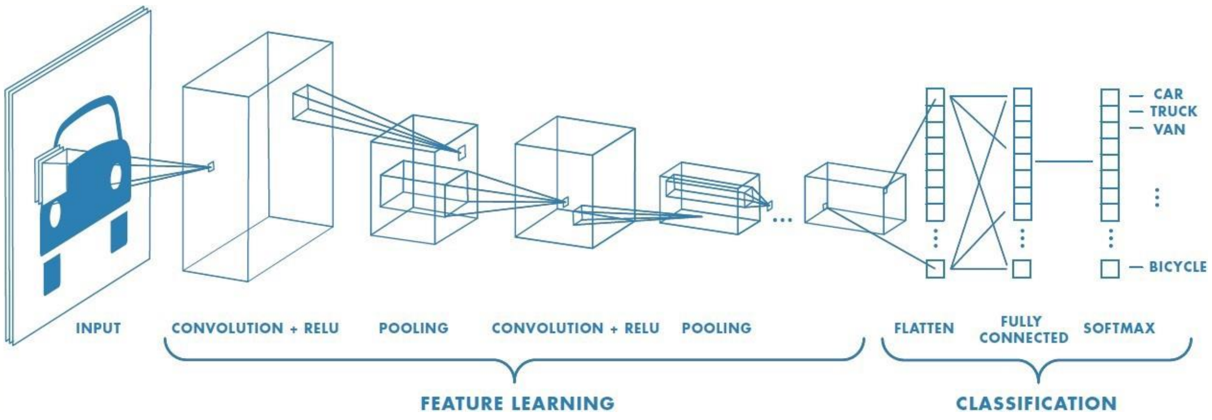


Fig2: A simple Convolutional Neural network
(Source: https://miro.medium.com/max/2000/1*vkQ0hXDaQv57sALXAJquxA.jpeg)

In this network, an input image is passed. Unlike other algorithms, CNNs do not require much preprocessing. They do not require much training also as they will learn themselves.

CNNs are preferred more over simple Artificial Neural networks as for image classification, firstly image has to be converted into a 1-D image from 2-D to train the model. This will increase the complexity of the model. Moreover, spatial features of an image will also be lost with ANNs. Therefore, CNNs are preferred more over multi-level perceptron for image classification.

CNNs have many filters or kernels present which are used for extracting features from the input images. Spatial features are the arrangement and relationship of pixels among them. CNN reserves the spatial features in an image. They reduce images such that features of images are not lost ; and make good predictions.

The main objective of CNN is to extract nearly all the features which can be done by layers present. The first layer captures some low-level features such as edges, gradient orientation etc and high-level features are extracted with more added layers. Feature extraction generates the feature map. This feature map can have increased or decreased dimensions in comparison to the input. The reason behind dimension change is Padding. Padding can be valid or same. Padding is same when the dimensions of input as well as convolved feature is same. It can be achieved by Data Augmentation. On Contrary, In the Valid Padding, the dimensions of convolved feature reduces.

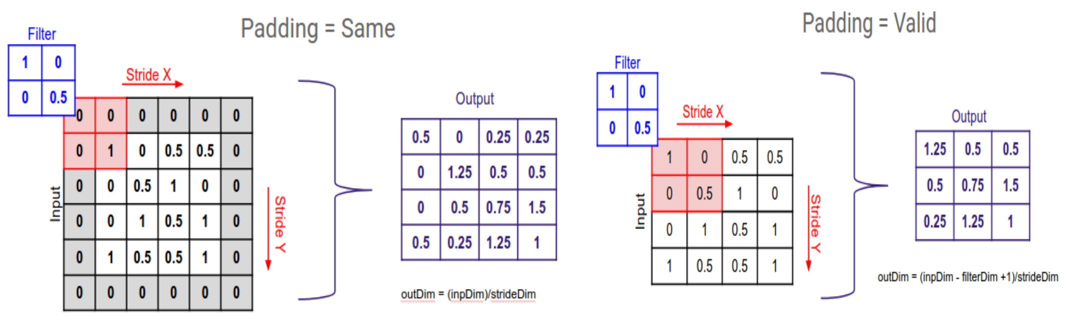

Padding = Same
Filter
Stride X
Stride Y
Input
Output
outDim = (inpDim)/strideDim
Padding = Valid
outDim = (inpDim - filterDim +1)/strideDim

Fig3: Types of Padding
(Source: https://miro.medium.com/max/1300/1*sKTqxILUPNI6P8t-bBIE1w.png)

Activation Functions are really useful for CNNs as they convert input signals of a node into an output signal. The output will be a simple linear function if activation function will not be present. The model will become simple linear regression model with limited power. Moreover, activation functions are useful for handling complicated data such as images, videos, audio , speech, etc. There are many types of Activation functions available but most popular among them are : Sigmoid or Logistic, Tanh-Hyperbolic Tangent , ReLu- Rectified Linear Units. The range of Sigmoid function is between 0 and 1. The output of Sigmoid function was not zero-centered which was then resolved by Hyperbolic Tangent function whose output lies between -1 to 1. Both of them has Vanishing gradient problem. ReLu function has become quite popular recently. It is given by: R(x)=max(0,x). Its output is 0 if x<0 and the output is x if x>0. This function resolves the Vanishing Gradient Descent. But it can only be used with Hidden Layers of a Neural Network. Other Activation functions such as Softmax or Sigmoid can be used for Output layers. Sigmoid function is used when classifying between 2 outputs.

After Convolutional layer, there is a Pooling layer who reduces the spatial size of Convolved Feature. There are two types of Pooling available: Average Pooling, which returns the average of the values from the portion of the image covered by Kernel whilst Max Pooling returns the Maximum value.

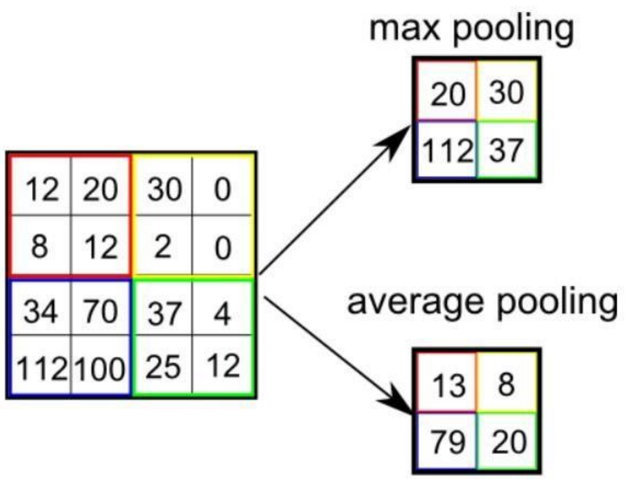


Fig4: Types of Pooling
(Source: https://miro.medium.com/max/1000/1*KQIEqhxzICU7thjaQBfPBQ.png)

The combination of Convolutional Layer and Pooling Layer form the Convolutional Neural networks. The number of these layers can be increased or decreased depending on the required computational power. These all steps have been performed for feature extraction.

The next step is to flatten the final output which will then be given to the fully connected layer to perform the classification. The model is able to classify images after certain number of epochs.

Transfer Learning :

Transfer Learning is an approach that is often used in Computer Vision and Natural language Processing problems. It is a technique in which neural network is initially trained using the pre-existing model. It reduces the training time and resources required for a neural network; and also reduces the errors which can be generated while training, thus producing good results. It can be used either for Feature extraction or Weight Initialisation. The last layer of the pre-trained model is generally removed which is Softmax layer so that the predictions can be done according to own model. There are many pre-existing models available such as Vgg16, Vgg19, ImageNet, InceptronV3, etc.

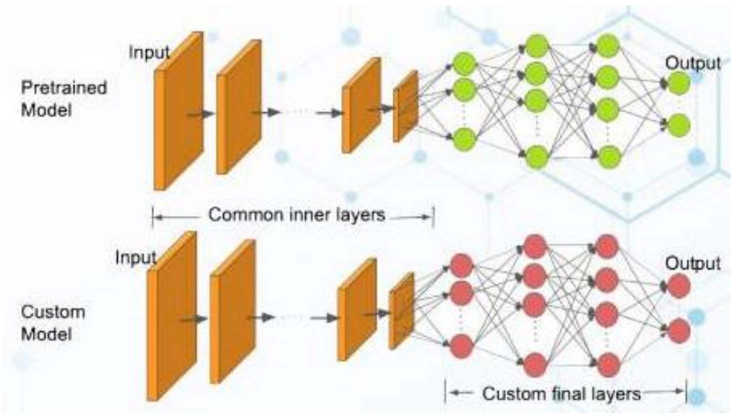


Fig5: Illustration of Pretrained model and Customised Neural Network
(Source: https://www.analyticssteps.com/backend/media/thumbnail/1967565/9315476_1592890541_transfer.jpg)

Experimental Setup:

Task-1: Classification:

This model used Machine Learning libraries such as TensorFlow, OpenCV and Keras API for performing classification and clustering. This model has been implemented on Google Colab which has all in-built packages required. Moreover, The availability of GPU for runtime sessions in Google Colab is very beneficial as it massively reduces the execution time as compared to CPU.

Importing necessary Libraries:

Initially all the packages were loaded which were required to form the model. For code, please refer to Appendix.

Loading dataset:

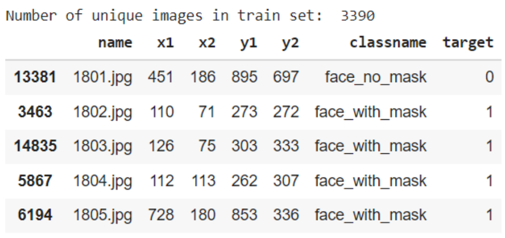
After loading the packages, path to the dataset was set. Dataset folder consists of two sub-folders annotations and images and two csv files. Annotations consist of information about images such as attributes, bounding boxes, classnames in json format. Images folder contains the images that needs to be classified. Two csv files are also loaded that contains information about train and test images in csv format.

Data pre-processing:

The dataset consists of 20 classes out of which only 2 classes have been considered; ‘face_with_mask’ and ‘face_no_mask’. Labels have been added using One-Hot encoding technique. 0 indicates images with ‘face_no_mask’ whereas 1 indicates ‘face_with_mask’.

```python
#Considering classes with face_with_mask or face_no_mask and leaving others
options = ['face_with_mask','face_no_mask']
train = train_full[train_full['classname'].isin(options)].sort_values('name')
#one-hot encoding
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
# encode classname column since it has only two classes
train['target'] = le.fit_transform(train.classname)
print("Number of unique images in train set: ", train.name.nunique())
train.head()
```

Code Snippet2: Considering two classes and Converting them into labels via One-Hot Encoding



```
Number of unique images in train set:  3390
           name   x1   x2   y1   y2       classname  target
13381  1801.jpg  451  186  895  697    face_no_mask       0
 3463  1802.jpg  110   71  273  272  face_with_mask       1
14835  1803.jpg  126   75  303  333  face_with_mask       1
 5867  1804.jpg  112  113  262  307  face_with_mask       1
 6194  1805.jpg  728  180  853  336  face_with_mask       1
```

Fig6: Displaying dataframe containing training data

So, the training dataset has 3390 images and testing set has 1698 images. After that, the csv files containing information about training and testing data were converted to list.

Train.csv contains image name, four coordinates of bounding boxes : x1, x2, y1 and y2, classnames. Now these coordinates will be used to create a list that would contain bounding boxes for each face.

```python
# create a list that would contain bounding boxes for each face
bounding_box=[]
for i in range(len(train)):
    lst = []
    # extract coordinates of bounding box
    for box in train.iloc[i][["x1",'x2','y1','y2']]:
        lst.append(box)
    bounding_box.append(lst)

#add new column with bounding boxes
train["bounding_box"] = bounding_box

# get box(es) for each unique image
def get_boxes(filename):
    boxes = []
    # get bounding_box column for all rows where train["name"] == filename
    for box in train[train["name"] == filename]["bounding_box"]:
        boxes.append(box)
    return boxes
```

Code Snippet3: Creating list containing bounding boxes from train.csv

A random image can be drawn from training dataset with bounding boxes using matplotlib and patches module. Axis can be set using plt.gca( ).

```python
# draw an image with detected objects
def draw_facebox(image, boxes):
    # plot the image
    plt.imshow(image)
    # get the context for drawing boxes
    ax = plt.gca()
    # plot each box
    for box in boxes:
    # get coordinates
        x, y, width, height = box[0], box[1], box[2], box[3],
        # create the shape
        rect = plt.Rectangle((x, y), width-x, height-y,
                             fill=False, color='b', linewidth=1)
        # draw the box
        ax.add_patch(rect)
    # show the plot
    plt.show()

# pick a random index from train_images
i = np.random.choice(np.arange(1801, len(train_images)))
# read image
image = plt.imread(os.path.join(img_dir,train_images[i]))
# get boxes for image
boxes = get_boxes(train_images[i])
print(boxes)
draw_facebox(image, boxes)
```

Code Snippet4: Plotting a random image fron training images with bounding box drawn

The result of above code snippet will produce image as shown below:

```
[[482, 0, 718, 323]]
```

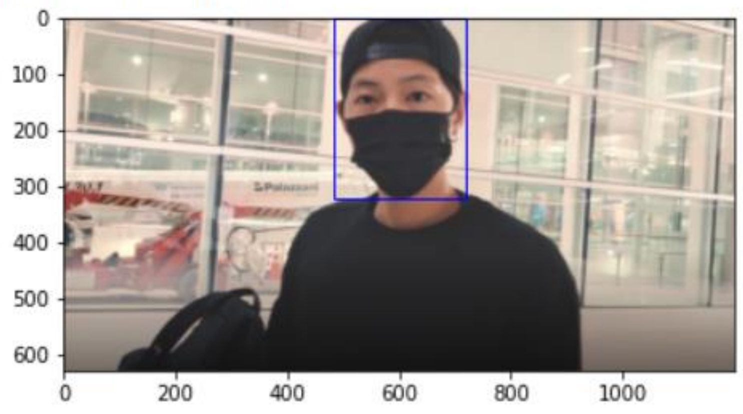


Fig7: Displaying Random Training Data

Image size was reduced to 128 so that all images should be of same size and less computational will be required. An array named, data, was created which will store features and labels of images. Afterwards, features and labels were stored separately in two different arrays, X and Y.

```python
#Rescaling the images by dividing it by 255
X=[]
Y=[]
for features, labels in data:
    X.append(features)
    Y.append(labels)
X = np.array(X)/255
print('Shape of X:', X.shape)
Y = np.array(Y)
print('Shape of Y:', Y.shape)
```

Code Snippet5: Rescaling the images and storing Features and Labels in arrays

The shape of X is (5749, 128,128, 3) and shape of Y is (5749,).

Building the model:

After preprocessing, data should be splitted into training and validation data using train_test_split. The test size or validation set is set to 0.12 i.e. 88% of the dataset would be used for training purposes and remaining 12% as validation data. The next step is to build the sequential model. The model consists of three Convolutional layer along with pooling layers which will be used for Feature Extraction.

The Model takes an input image of size(128,128,3) and maps to 32 feature maps (128x128x32). The Pool size is set to 2x2. Afterwards, MaxPooling2D layer produces output shape as (64x64x32); which is then fed as input to the second Convolution layer. It maps it to 64 feature maps (64x64x64).Another max pooling layer outputs (32x32x64). The last Convolutional layer produces output with 128 feature maps (32x32x128) which is then fed to final max pooling layer that pools output to (16x16x128). The dropout layer is further added for regularisation so that there should not be any problem such as overfitting. Afterwards, the outputs are flattened into a vector and passed to a fully connected hidden layer with ReLu activation function. Then, output is fed to last layer (Dense) having Sigmoid activation function which will classify into categories. The summary of the model has been shown in the Appendix.

Now, the model needs to be compiled. Here, Optimizer is chosen as Adam with epsilon value as 0.01. Binary cross entropy loss is used as there are only two classes for classification. Metrics is set to accuracy as this is classification task.

```python
#compiling the model
model.compile(optimizer=tf.keras.optimizers.Adam(epsilon=0.01),
              loss='binary_crossentropy', metrics=['accuracy'])
# training the model for 30 epochs
history = model.fit(X_train,y_train,batch_size=32,
                    epochs=30,
                    validation_data=(X_val, y_val))
```

Code Snippet6: Compiling and fitting the model

Testing:

The classification is to be done on test images also. As there is no information available regarding the bounding boxes of test images, so, they need to be found out. The Multi-Task Cascaded Convolutional Neural network (MTCNN) is used for drawing the bounding boxes for test images. It automatically detects faces and facial landmarks from images. Similar to the training images, a dataframe containing the names and bounding boxes of images was formed. After rescaling, the prediction is made and results are written to file 'submission1.csv'.

Classification with Vgg16:

Classification was also performed with the help of pre-trained Vgg16 model. It uses imagenets weights and is efficient when image size is (224x224). So, images are reshaped and resized so that Vgg16 can be used for training the model. Similar to the sequential model, data was split into training and validation. The top layer of model was not loaded. Instead, another dense layer was added with sigmoid activation function.

```
# load model
vgg = VGG16(weights="imagenet", include_top=False, input_shape=(img_size, img_size, 3))
for layer in vgg.layers:
    layer.trainable = False
top = vgg.output
top = GlobalAveragePooling2D()(top)
top = Dense(units=256, activation="relu")(top)
top = Dense(units=128, activation="relu")(top)
top = Dense(units=1, activation="sigmoid")(top)

model1 = Model(inputs=vgg.input, outputs=top)
print(model1.summary())
```

Code Snippet7: Loading Vgg16 model and adding dense layers

Out of 14,879,041 parameters, only 164,353 parameters were trained and remaining 14,714,688 were non-trainable parameters. Afterwards, model was compiled with 'adam' optimizer, 'binary_crossentropy' as loss and due to classification, metrics was selected as 'accuracy'. The model was run for 5 epochs. Code along with results have been specified in Appendix.

Task-2: Clustering:

Annotations folder present in dataset will be used for clustering as it provides information about images in json format; so to access it, getJSON function was built which would access all the files with image information. Another function, adjust_gamma was created to improve brightness and contrast of images so that clusters can be formed efficiently. Pre-trained VGG16 model was built for feature extraction and last layer was removed from the model. As the dataset is huge with lots of features, so, Principal Component Analysis was done to extract important features only to reduce the dimensions and computational cost of clustering. As there was 2 unique labels, so, 2 clusters were formed using K-Means algorithm; assuming as one with mask and another as no mask. For code, please click here.

```
kmeans = KMeans(n_clusters=len(unique_labels))
kmeans.fit(x)
# holds the cluster id and the images { id: [images] }
groups = {}
for file, cluster in zip(X,kmeans.labels_):
    if cluster not in groups.keys():
        groups[cluster] = []
        groups[cluster].append(file)
    else:
        groups[cluster].append(file)
```

Code Snippet8: Forming clusters using k-means

Results and Analysis:

The sequential model was trained and validated. For classification, After 30 epochs, training accuracy came out to be 99% and validation accuracy was 94%. The curves are plotted below showing Accuracy and loss for training as well as validation data.

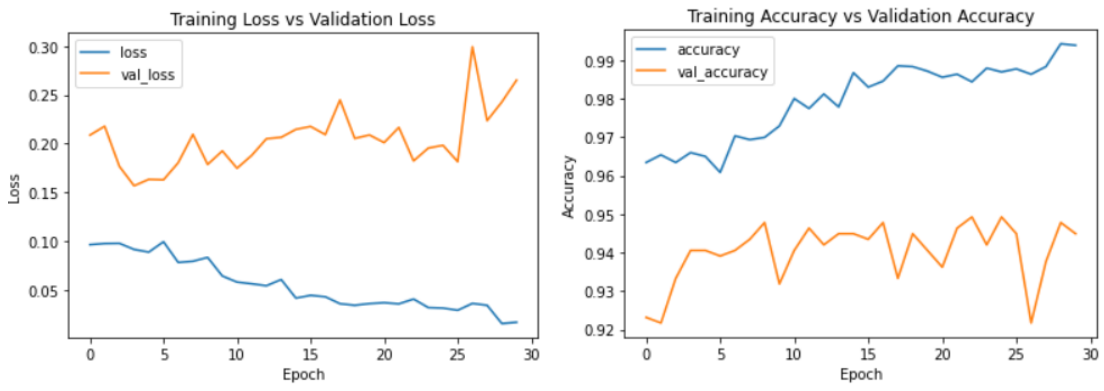


Fig8: Loss and Accuracy Curves for Sequential model

The classification report for classification with CNN is given below:

Classification Report	Precision	Recall	F1 score	Support
With Mask	0.97	0.96	0.97	508
Without mask	0.88	0.93	0.91	182

Table1: Classification report of Sequential model

The curves below show accuracy as well as loss for training and validation data when Vgg16 model was used for prediction after 5 epochs.

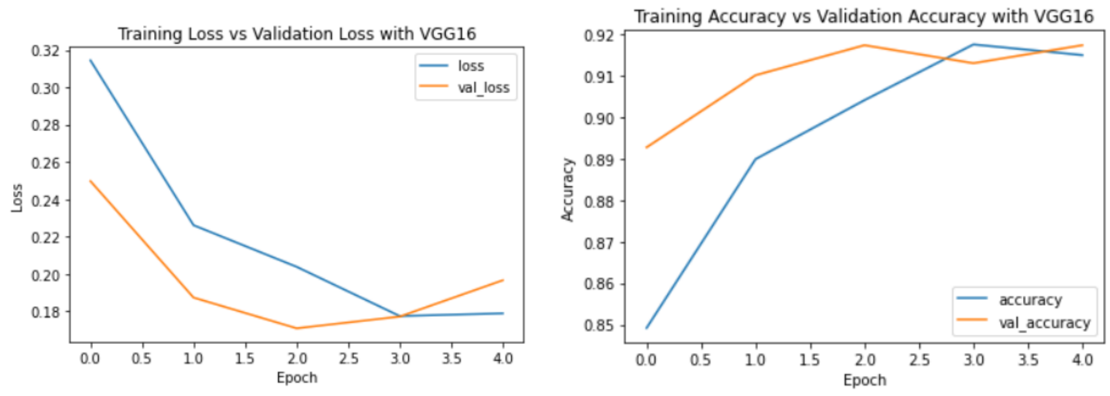


Fig9: Loss and Accuracy Curves for vgg16 model

The classification report for predictions by Vgg16 trained model is given below:

Classification Report	Precision	Recall	F1 score	Support
With Mask	1.00	0.51	0.67	183
Without mask	0.85	1.00	0.92	507

Table2: Classification report of vgg16 model

The confusion matrix was also generated which showed the true accuracy. It shows the false positives, false negatives, true positives, true negatives predicted by the model. The figure below shows the confusion matrix for the Classification task:

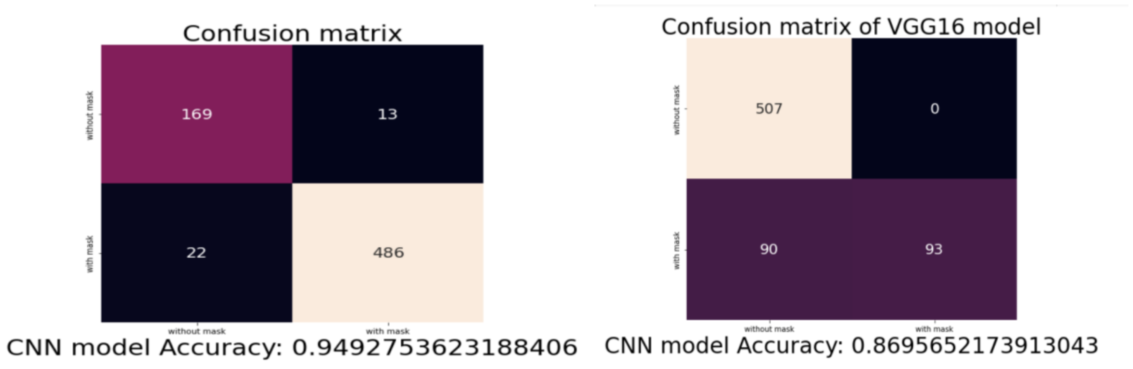


Fig10: Confusion Matrix for both models

From the above fig, it can be concluded that some images were misclassified. This shows model was having some difficulties to classify the images.

The results of Clustering was not expected. The model found it very challenging to form the clusters. The accuracy came out to be just 55%. Some sample images from both the clusters were displayed which are specified in Appendix. After observing those images, it looks like cluster 0 contains images with face masks and little bit similar facial features whilst cluster 1 contains images without face masks. Some more clusters can also be formed in order to enhance the accuracy. The classification report of model performing clustering is given below:

Classification Report	Precision	Recall	F1 score	Support
With Mask	0.27	0.61	0.31	1569
Without mask	0.72	0.38	0.66	4180

Table3: Classification report of clustering

The results above show that model trained by Vgg16 was not able to make good predictions. There can be many reasons for this abnormal behaviour such as the filters of Vgg16 might not be suitable for this dataset, or, the layers need to be more trained. As the training dataset was huge, freezing all the layers might be not be the good solution. Moreover, Vgg16 took so long to train which was highly unexpected. On the Contrary, a sequential Keras model turned out to be more efficient which is very much clear from the classification reports as well specified above.

Conclusion and Future Enhancements:

This system was used to detect if people are wearing face masks or not based on large size dataset obtained from Kaggle. The proposed CNN model was able to classify at some extent but sometimes, incorrect results were produced. There can be many reasons for inaccuracies such as difference in facial expressions for every individual which makes feature extraction quite difficult. Moreover, there is more than one person in picture so it might be possible while reshaping, many features might have lost. However, it was able to majorly classify images correctly with an accuracy of 94%. The sequential model was able to learn the parameters much efficiently.

From the above results of Vgg16 model, it can be concluded that different pretrained models , say, resnet50, vgg19 can be used which claims to provide more accurate results than vgg16 and take less training time.

The model can further be used in many applications which involve face recognition, can be used in real-time detection systems with some modifications. The proposed methodologies and system can be improved by experimenting it with different set of images.

References:

1. D. Meena and R. Sharan, "An approach to face detection and recognition," 2016 International Conference on Recent Advances and Innovations in Engineering (ICRAIE), Jaipur, 2016, pp. 1-6, doi: 10.1109/ICRAIE.2016.7939462.

2. Das, A., Ansari, M. W., & Basak, R. (2020, December). Covid-19 Face Mask Detection Using TensorFlow, Keras and OpenCV. In *2020 IEEE 17th India Council International Conference (INDICON)* (pp. 1-5). IEEE.

3. Costa, V. L., Teixeira, E. H., Mafra, S. B., & Rodrigues, J. J. (2021, June). On the Performance Analysis of a TensorFlow based Neural Network for Face Mask Detection. In *2021 International Wireless Communications and Mobile Computing (IWCMC)* (pp. 342-346). IEEE.

4. S. Li, F. Tao, T. Shi and J. Kuang, "Improvement of YOLOv3 network based on ROI", *2019 IEEE 4th Advanced Information Technology Electronic and Automation Control Conference (IAEAC)*, pp. 2590-2596, 2019.

5. Tomás, J., Rego, A., Viciano-Tudela, S., & Lloret, J. (2021, August). Incorrect Facemask-Wearing Detection Using Convolutional Neural Networks with Transfer Learning. In *Healthcare* (Vol. 9, No. 8, p. 1050). Multidisciplinary Digital Publishing Institute.

6. TensorFlow. 2021. *TensorFlow*. [online] Available at: <https://www.tensorflow.org/> [Accessed 10 November 2021].

7. Pai, A., 2020. *ANN vs CNN vs RNN | Types of Neural Networks*. [online] Analytics Vidhya. Available at: <https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/> [Accessed 6 November 2021].

8. Saha, S., 2021. *A Comprehensive Guide to Convolutional Neural Networks—the ELI5 way*. [online] Medium. Available at: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53> [Accessed 6 November 2021].

9. Brownlee, J., 2019. *How to Improve Performance With Transfer Learning for Deep Learning Neural Networks*. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/how-to-improve-performance-with-transfer-learning-for-deep-learning-neural-networks/> [Accessed 8 November 2021].

10. Dwivedi, R., 2020. *How is Transfer Learning done in Neural Networks and Convolutional Neural Networks? | Analytics Steps*. [online] Analyticssteps.com. Available at: <https://www.analyticssteps.com/blogs/how-transfer-learning-done-neural-networks-and-convolutional-neural-networks> [Accessed 8 November 2021].

11. Brownlee, J., 2019. *A Gentle Introduction to Dropout for Regularizing Deep Neural Networks*. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/#:~:text=regularizers%20are%20present.-,%E2%80%94%20Dropout%3A%20A%20Simple%20Way%20to%20Prevent%20Neural%20Networks%20from%20Overfitting,be%20required%20when%20using%20dropout.> [Accessed 8 November 2021].

12. Dulcic, L., 2021. *Face Recognition with FaceNet and MTCNN*. [online] Arsfutura.com. Available at: <https://arsfutura.com/magazine/face-recognition-with-facenet-and-mtcnn/> [Accessed 8 November 2021].

Appendix:

<u>Project Proposal :</u>

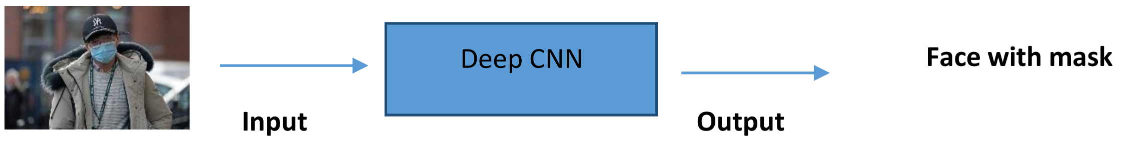
Title of Project:
Face Mask Detection : Analysis using Convolutional Neural Network

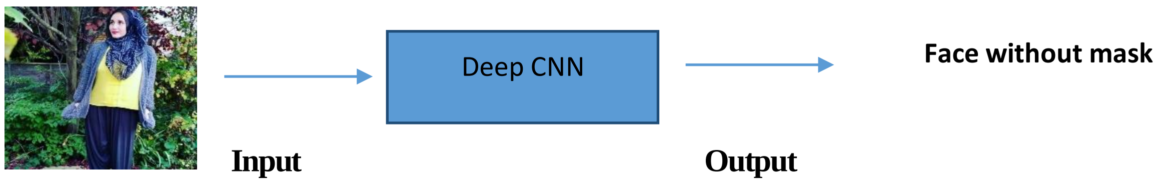
Objectives:
The aim of this project is to build a system which will detect if the person is wearing a face mask or not.

Motivation for the project:
In 2020, the Covid-19 pandemic hit the world. Many of the safety measures were taken to avoid the spread of virus. Due to the transmission of the virus through air, many countries decided to use mask as mandatory measure to avoid the transmission. The use of masks was very beneficial in controlling the infection. Therefore, methods based on Artificial Intelligence were required to detect if the person is wearing mask or not.

Plan and Expected Outputs:
<u>Task-1: Classification:</u>

This project aims to classify if the person is wearing the mask or not. So, the model will be able to classify images with the help of Convolutional Neural Network. The advantage of CNN over Neural Network is that there is no need to flatten the input image to 1D as they can work with 2D image data also which will help in retaining the spatial properties of image.





<u>Task-2: Clustering:</u>

A CNN can also act as a feature extractor which can be used to group similar images in a cluster. In my dataset, there are large number of images which can be grouped together based on some features such as person wearing glasses.

The Deep CNN will be implemented in Python using TensorFlow and Keras libraries. TensorFlow has a comprehensive, flexible ecosystem of tools, libraries and community resources that helps researchers to develop state-of-the-art Machine Learning applications.
Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow to define and train neural networks. Keras empowers to take full advantage of the scalability and cross-platform capabilities of TensorFlow.

Dataset:
The dataset has been chosen from Kaggle. The link to the dataset is:
https://www.kaggle.com/wobotintelligence/face-mask-detection-dataset

Coding:

Task-1 : Classification :-

```python
#importing necessary libraries
import numpy as np
import pandas as pd
import os

#used for visualisation
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import seaborn as sns
from PIL import Image as im

#importing Deep Learning libraries for models
import tensorflow as tf
from keras.layers import Flatten, Dense, Conv2D, MaxPooling2D, Dropout
from keras.models import Sequential
import pathlib
import cv2
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
from tensorflow.keras.layers import GlobalAveragePooling2D
from keras.models import Model

# for loading/processing the images
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input

# Pre-trained model
from keras.applications.vgg16 import VGG16
from keras.models import Model

# clustering and dimension reduction
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

#train test split
from sklearn.model_selection import train_test_split

#Loading Dataset
os.environ['KAGGLE_CONFIG_DIR']="/content/"
!kaggle datasets download wobotintelligence/face-mask-detection-dataset
import zipfile
! unzip "face-mask-detection-dataset.zip"
directory="/content/Medical mask/Medical mask/Medical Mask/annotations"
img_dir = "/content/Medical mask/Medical mask/Medical Mask/images"
train_full = pd.read_csv('/content/train.csv')
submission = pd.read_csv('/content/submission.csv')
```

```python
#Checking the file has been loaded or not
train_full.head()
```

```python
#Checking the file has been loaded or not
train_full.head()
```

	name	x1	x2	y1	y2	classname
0	2756.png	69	126	294	392	face_with_mask
1	2756.png	505	10	723	283	face_with_mask
2	2756.png	75	252	264	390	mask_colorful
3	2756.png	521	136	711	277	mask_colorful
4	6098.jpg	360	85	728	653	face_no_mask

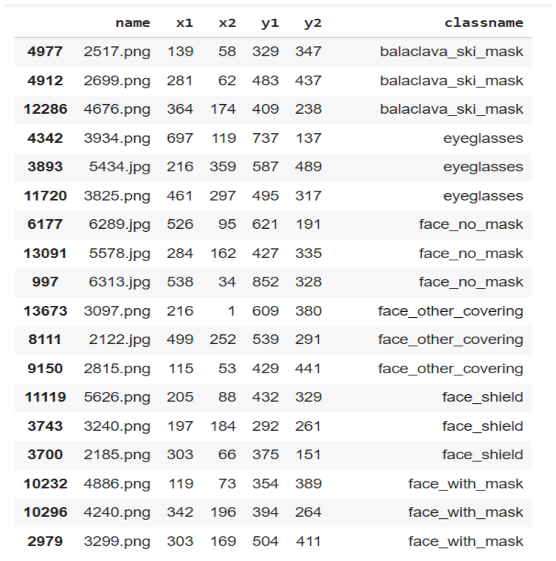
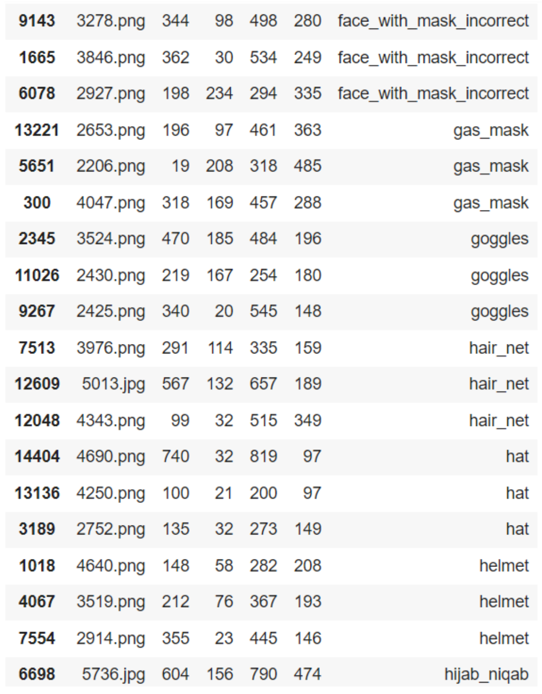
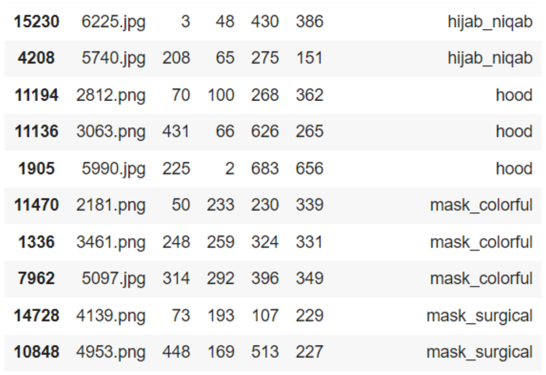
```python
#Display number of unique images in training list
train_full.name.unique().shape
```

```python
#Displaying number of classes and their count
train_full.classname.value_counts()
```

```
face_with_mask              4180
mask_surgical               2430
mask_colorful               1876
face_no_mask                1569
face_other_covering         1372
eyeglasses                   914
hat                          823
sunglasses                   358
hair_net                     287
scarf_bandana                260
goggles                      192
helmet                       187
hijab_niqab                  173
face_shield                  160
hood                         159
face_with_mask_incorrect     150
balaclava_ski_mask           134
turban                        94
gas_mask                      55
other                         39
Name: classname, dtype: int64
```

```python
classes = train_full["classname"].unique()
print("Total Classes: ",len(classes))
print("\n> Classes <\n",classes)

ax = sns.catplot(x='classname',kind='count',data=train_full,orient="h",height=10,aspect=2)
ax.fig.suptitle('Count of Classnames',fontsize=16,color="r")
ax.fig.autofmt_xdate()
```

```python
sample_per_class = train_full.groupby('classname').sample(3)
sample_per_class

```

	name	x1	x2	y1	y2	classname
4977	2517.png	139	58	329	347	balaclava_ski_mask
4912	2699.png	281	62	483	437	balaclava_ski_mask
12286	4676.png	364	174	409	238	balaclava_ski_mask
4342	3934.png	697	119	737	137	eyeglasses
3893	5434.jpg	216	359	587	489	eyeglasses
11720	3825.png	461	297	495	317	eyeglasses
6177	6289.jpg	526	95	621	191	face_no_mask
13091	5578.jpg	284	162	427	335	face_no_mask
997	6313.jpg	538	34	852	328	face_no_mask
13673	3097.png	216	1	609	380	face_other_covering
8111	2122.jpg	499	252	539	291	face_other_covering
9150	2815.png	115	53	429	441	face_other_covering
11119	5626.png	205	88	432	329	face_shield
3743	3240.png	197	184	292	261	face_shield
3700	2185.png	303	66	375	151	face_shield
10232	4886.png	119	73	354	389	face_with_mask
10296	4240.png	342	196	394	264	face_with_mask
2979	3299.png	303	169	504	411	face_with_mask
9143	3278.png	344	98	498	280	face_with_mask_incorrect
1665	3846.png	362	30	534	249	face_with_mask_incorrect
6078	2927.png	198	234	294	335	face_with_mask_incorrect
13221	2653.png	196	97	461	363	gas_mask
5651	2206.png	19	208	318	485	gas_mask
300	4047.png	318	169	457	288	gas_mask
2345	3524.png	470	185	484	196	goggles
11026	2430.png	219	167	254	180	goggles
9267	2425.png	340	20	545	148	goggles
7513	3976.png	291	114	335	159	hair_net
12609	5013.jpg	567	132	657	189	hair_net
12048	4343.png	99	32	515	349	hair_net
14404	4690.png	740	32	819	97	hat
13136	4250.png	100	21	200	97	hat
3189	2752.png	135	32	273	149	hat
1018	4640.png	148	58	282	208	helmet
4067	3519.png	212	76	367	193	helmet
7554	2914.png	355	23	445	146	helmet
6698	5736.jpg	604	156	790	474	hijab_niqab
15230	6225.jpg	3	48	430	386	hijab_niqab
4208	5740.jpg	208	65	275	151	hijab_niqab
11194	2812.png	70	100	268	362	hood
11136	3063.png	431	66	626	265	hood
1905	5990.jpg	225	2	683	656	hood
11470	2181.png	50	233	230	339	mask_colorful
1336	3461.png	248	259	324	331	mask_colorful
7962	5097.jpg	314	292	396	349	mask_colorful
14728	4139.png	73	193	107	229	mask_surgical
10848	4953.png	448	169	513	227	mask_surgical

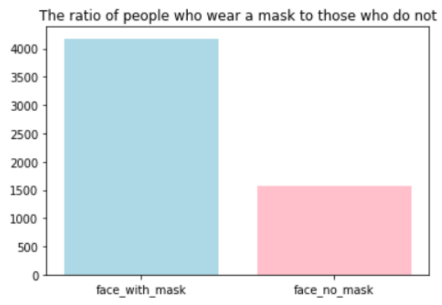
```python
#Considering classes with face_with_mask or face_no_mask and leaving others
options = ['face_with_mask','face_no_mask']
train = train_full[train_full['classname'].isin(options)].sort_values('name')
#one-hot encoding
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
# encode classname column since it has only two classes
train['target'] = le.fit_transform(train.classname)
print("Number of unique images in train set: ", train.name.nunique())
train.head()
```

```python
print("Number of unique images in submission set: ", submission.name.nunique())
submission.head()
```

```
Number of unique images in train set:  3390
        name      x1   x2   y1   y2   classname       target
13381   1801.jpg  451  186  895  697  face_no_mask    0
3463    1802.jpg  110   71  273  272  face_with_mask  1
14835   1803.jpg  126   75  303  333  face_with_mask  1
5867    1804.jpg  112  113  262  307  face_with_mask  1
6194    1805.jpg  728  180  853  336  face_with_mask  1
```

```
Number of unique images in submission set:  1698
   name      x1   x2   y1   y2   classname
0  1800.jpg  NaN  NaN  NaN  NaN  NaN
1  1800.jpg  NaN  NaN  NaN  NaN  NaN
2  1800.jpg  NaN  NaN  NaN  NaN  NaN
3  1799.jpg  NaN  NaN  NaN  NaN  NaN
4  1799.jpg  NaN  NaN  NaN  NaN  NaN
```

```python
plt.bar(['face_with_mask','face_no_mask'], train.classname.value_counts(), color = ['lightblue','pink']);
plt.title('The ratio of people who wear a mask to those who do not');
```



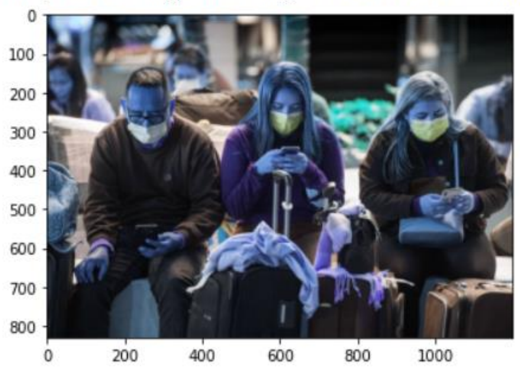
```python
#Converting dataframes to list having just unique names of images
train_images = train.name.unique().tolist()
test_images = submission.name.unique().tolist()
```

```python
#Sample image from train set
# pick a random index from train_images
i = np.random.choice(np.arange(1801, len(train_images)))
# read image
img=plt.imread(os.path.join(img_dir,train_images[i]))
plt.imshow(img)
```

```python
#Sample image from test set
# pick a random index from test_images
i = np.random.choice(len(test_images))
# read image
img = cv2.imread(os.path.join(img_dir,test_images[i]))
# plot image
plt.imshow(img)
```

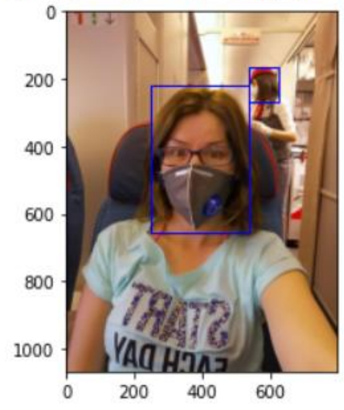
```
<matplotlib.image.AxesImage at 0x7fbeb3beb150>
```

```
<matplotlib.image.AxesImage at 0x7fbeb236e0d0>
```



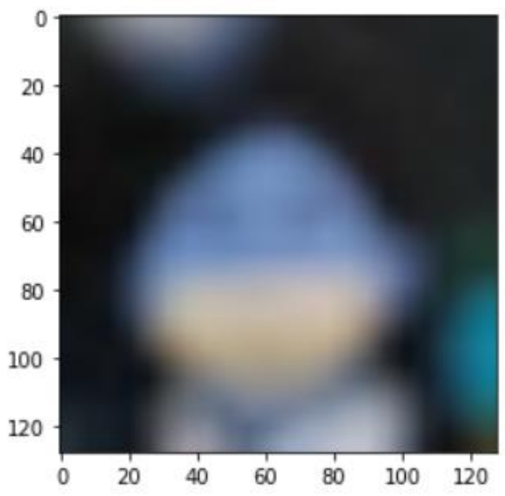
```python
# create a list that would contain bounding boxes for each face
bounding_box=[]
for i in range(len(train)):
    lst = []
    # extract coordinates of bounding box
    for box in train.iloc[i][["x1",'x2','y1','y2']]:
        lst.append(box)
    bounding_box.append(lst)

#add new column with bounding boxes
train["bounding_box"] = bounding_box

# get box(es) for each unique image
def get_boxes(filename):
    boxes = []
    # get bounding_box column for all rows where train["name"] == filename
    for box in train[train["name"] == filename]["bounding_box"]:
        boxes.append(box)
    return boxes

# draw an image with detected objects
def draw_facebox(image, boxes):
    # plot the image
    plt.imshow(image)
    # get the context for drawing boxes
    ax = plt.gca()
    # plot each box
    for box in boxes:
    # get coordinates
        x, y, width, height = box[0], box[1], box[2], box[3],
        # create the shape
        rect = plt.Rectangle((x, y), width-x, height-y,
                             fill=False, color='b', linewidth=1)
        # draw the box
        ax.add_patch(rect)
    # show the plot
    plt.show()

# pick a random index from train_images
i = np.random.choice(np.arange(1801, len(train_images)))
# read image
image = plt.imread(os.path.join(img_dir,train_images[i]))
# get boxes for image
boxes = get_boxes(train_images[i])
print(boxes)
draw_facebox(image, boxes)
```

```
[[249, 221, 537, 653], [537, 164, 626, 268]]
```



```
#Creation of training data- Resize, reshape
img_size=128
data=[]
for i in range(len(train)):
    x,y,width,height = train.iloc[i]['bounding_box']
    image = train.iloc[i]['name']
    # read image with green channel
    img_array = cv2.imread(os.path.join(img_dir,image), 1)
    # crop image with bounding box
    img_cropped = img_array[y:height,x:width]
    # resize cropped image
    img = cv2.resize(img_cropped,(img_size,img_size))
    data.append([img,train.iloc[i]['target']])
```

```
#Plot a random cropped image from data
# Pick a random index from data
i = np.random.choice(range(len(data)))
plt.imshow(data[i][0]);
```



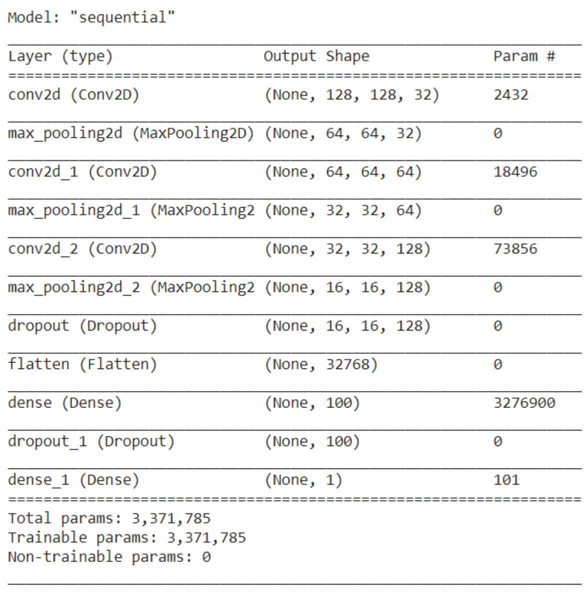
```
#Rescaling the images by dividing it by 255 and converting features and labels in arrays
X=[]
Y=[]
for features, labels in data:
    X.append(features)
    Y.append(labels)
```

```python
X = np.array(X)/255
print('Shape of X:', X.shape)
Y = np.array(Y)
print('Shape of Y:', Y.shape)
```

```
Shape of X: (5749, 128, 128, 3)
Shape of Y: (5749,)
```

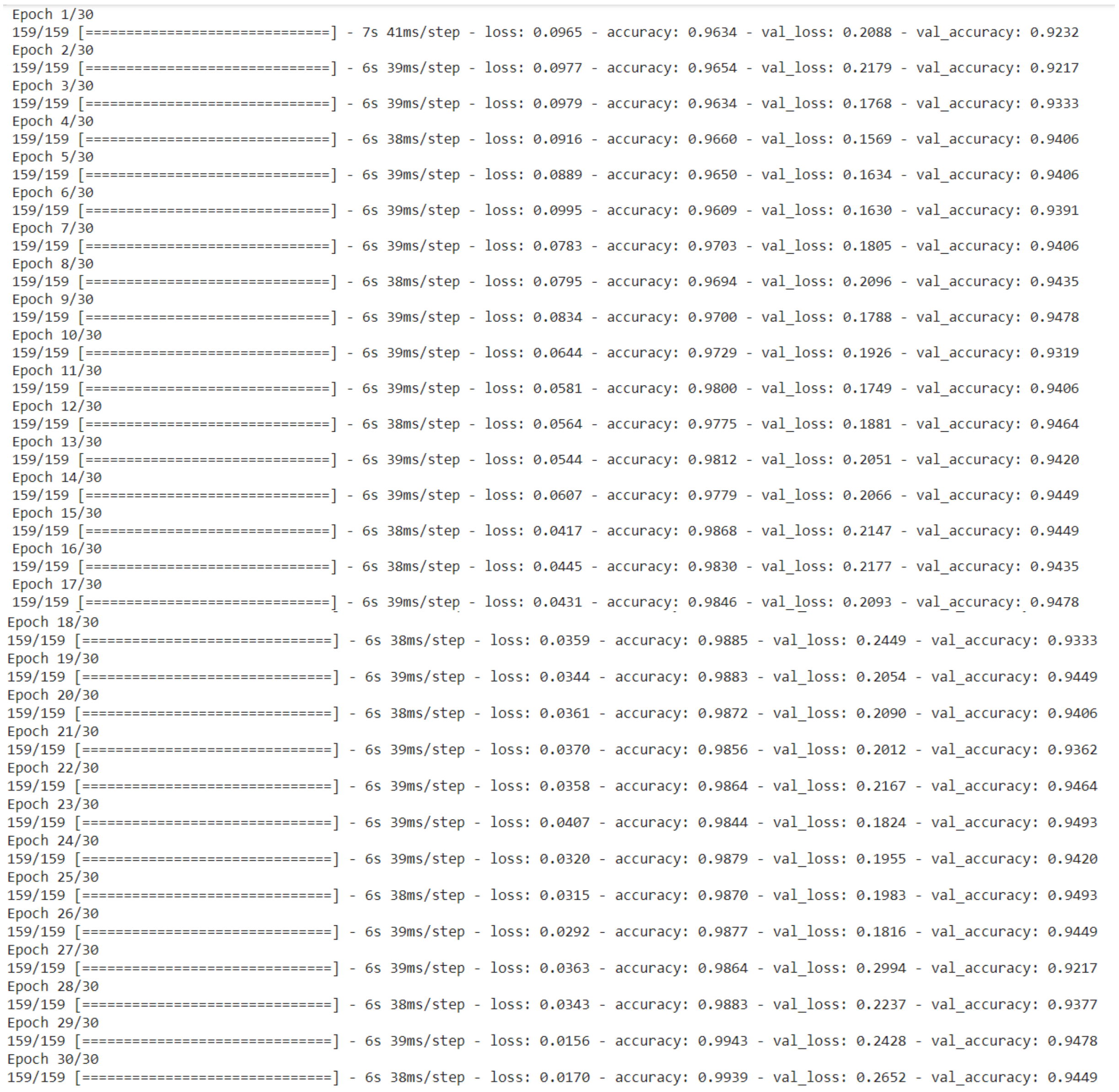
```python
# split our data into train and validation sets
X_train,X_val,y_train,y_val = train_test_split(X, Y,test_size=0.12,random_state=42)
```

```python
# building a linear stack of layers with the sequential model
model = Sequential()
#three convolutional layers
#First conv block
model.add(Conv2D(filters=32, kernel_size=5, activation="relu", padding='same', input_shape=(128,128,3)))
model.add(MaxPooling2D(pool_size=(2,2)))
#Second conv block
model.add(Conv2D(filters=64, kernel_size=3, activation="relu", padding='same'))
model.add(MaxPooling2D(pool_size=(2,2)))
#Third Conv block
model.add(Conv2D(filters=128, kernel_size=3, activation="relu", padding='same'))
model.add(MaxPooling2D(pool_size=(2,2)))
#Dropout for regularization
model.add(Dropout(0.25))
#flatten output of conv
model.add(Flatten())
#hidden layer
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.5))
#output layer
model.add(Dense(1, activation='sigmoid'))
model.summary()
```



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 128, 128, 32)      2432
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 64, 64, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 16, 16, 128)       0
_________________________________________________________________
dropout (Dropout)            (None, 16, 16, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 32768)             0
_________________________________________________________________
dense (Dense)                (None, 100)               3276900
_________________________________________________________________
dropout_1 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 101
=================================================================
Total params: 3,371,785
Trainable params: 3,371,785
Non-trainable params: 0
_________________________________________________________________
```

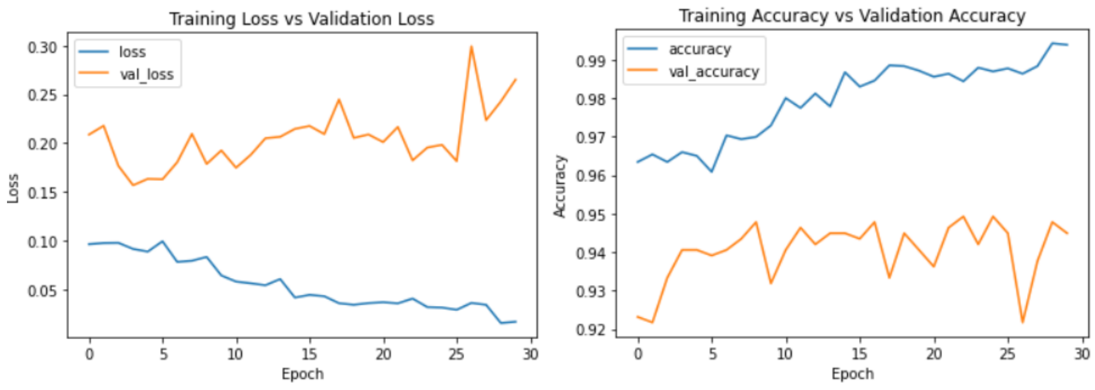
```python
#compiling the model
model.compile(optimizer=tf.keras.optimizers.Adam(epsilon=0.01),
              loss='binary_crossentropy', metrics=['accuracy'])
# training the model for 30 epochs
history = model.fit(X_train,y_train,batch_size=32,
                    epochs=30,
                    validation_data=(X_val, y_val))
```



```
Epoch 1/30
159/159 [==============================] - 7s 41ms/step - loss: 0.0965 - accuracy: 0.9634 - val_loss: 0.2088 - val_accuracy: 0.9232
Epoch 2/30
159/159 [==============================] - 6s 39ms/step - loss: 0.0977 - accuracy: 0.9654 - val_loss: 0.2179 - val_accuracy: 0.9217
Epoch 3/30
159/159 [==============================] - 6s 39ms/step - loss: 0.0979 - accuracy: 0.9634 - val_loss: 0.1768 - val_accuracy: 0.9333
Epoch 4/30
159/159 [==============================] - 6s 38ms/step - loss: 0.0916 - accuracy: 0.9660 - val_loss: 0.1569 - val_accuracy: 0.9406
Epoch 5/30
159/159 [==============================] - 6s 39ms/step - loss: 0.0889 - accuracy: 0.9650 - val_loss: 0.1634 - val_accuracy: 0.9406
Epoch 6/30
159/159 [==============================] - 6s 39ms/step - loss: 0.0995 - accuracy: 0.9609 - val_loss: 0.1630 - val_accuracy: 0.9391
Epoch 7/30
159/159 [==============================] - 6s 39ms/step - loss: 0.0783 - accuracy: 0.9703 - val_loss: 0.1805 - val_accuracy: 0.9406
Epoch 8/30
159/159 [==============================] - 6s 38ms/step - loss: 0.0795 - accuracy: 0.9694 - val_loss: 0.2096 - val_accuracy: 0.9435
Epoch 9/30
159/159 [==============================] - 6s 39ms/step - loss: 0.0834 - accuracy: 0.9700 - val_loss: 0.1788 - val_accuracy: 0.9478
Epoch 10/30
159/159 [==============================] - 6s 39ms/step - loss: 0.0644 - accuracy: 0.9729 - val_loss: 0.1926 - val_accuracy: 0.9319
Epoch 11/30
159/159 [==============================] - 6s 39ms/step - loss: 0.0581 - accuracy: 0.9800 - val_loss: 0.1749 - val_accuracy: 0.9406
Epoch 12/30
159/159 [==============================] - 6s 38ms/step - loss: 0.0564 - accuracy: 0.9775 - val_loss: 0.1881 - val_accuracy: 0.9464
Epoch 13/30
159/159 [==============================] - 6s 39ms/step - loss: 0.0544 - accuracy: 0.9812 - val_loss: 0.2051 - val_accuracy: 0.9420
Epoch 14/30
159/159 [==============================] - 6s 39ms/step - loss: 0.0607 - accuracy: 0.9779 - val_loss: 0.2066 - val_accuracy: 0.9449
Epoch 15/30
159/159 [==============================] - 6s 38ms/step - loss: 0.0417 - accuracy: 0.9868 - val_loss: 0.2147 - val_accuracy: 0.9449
Epoch 16/30
159/159 [==============================] - 6s 38ms/step - loss: 0.0445 - accuracy: 0.9830 - val_loss: 0.2177 - val_accuracy: 0.9435
Epoch 17/30
159/159 [==============================] - 6s 39ms/step - loss: 0.0431 - accuracy: 0.9846 - val_loss: 0.2093 - val_accuracy: 0.9478
Epoch 18/30
159/159 [==============================] - 6s 38ms/step - loss: 0.0359 - accuracy: 0.9885 - val_loss: 0.2449 - val_accuracy: 0.9333
Epoch 19/30
159/159 [==============================] - 6s 39ms/step - loss: 0.0344 - accuracy: 0.9883 - val_loss: 0.2054 - val_accuracy: 0.9449
Epoch 20/30
159/159 [==============================] - 6s 38ms/step - loss: 0.0361 - accuracy: 0.9872 - val_loss: 0.2090 - val_accuracy: 0.9406
Epoch 21/30
159/159 [==============================] - 6s 39ms/step - loss: 0.0370 - accuracy: 0.9856 - val_loss: 0.2012 - val_accuracy: 0.9362
Epoch 22/30
159/159 [==============================] - 6s 39ms/step - loss: 0.0358 - accuracy: 0.9864 - val_loss: 0.2167 - val_accuracy: 0.9464
Epoch 23/30
159/159 [==============================] - 6s 39ms/step - loss: 0.0407 - accuracy: 0.9844 - val_loss: 0.1824 - val_accuracy: 0.9493
Epoch 24/30
159/159 [==============================] - 6s 39ms/step - loss: 0.0320 - accuracy: 0.9879 - val_loss: 0.1955 - val_accuracy: 0.9420
Epoch 25/30
159/159 [==============================] - 6s 38ms/step - loss: 0.0315 - accuracy: 0.9870 - val_loss: 0.1983 - val_accuracy: 0.9493
Epoch 26/30
159/159 [==============================] - 6s 39ms/step - loss: 0.0292 - accuracy: 0.9877 - val_loss: 0.1816 - val_accuracy: 0.9449
Epoch 27/30
159/159 [==============================] - 6s 39ms/step - loss: 0.0363 - accuracy: 0.9864 - val_loss: 0.2994 - val_accuracy: 0.9217
Epoch 28/30
159/159 [==============================] - 6s 38ms/step - loss: 0.0343 - accuracy: 0.9883 - val_loss: 0.2237 - val_accuracy: 0.9377
Epoch 29/30
159/159 [==============================] - 6s 39ms/step - loss: 0.0156 - accuracy: 0.9943 - val_loss: 0.2428 - val_accuracy: 0.9478
Epoch 30/30
159/159 [==============================] - 6s 38ms/step - loss: 0.0170 - accuracy: 0.9939 - val_loss: 0.2652 - val_accuracy: 0.9449
```

```python
#curves for accuracy and loss
history_frame = pd.DataFrame(history.history)
history_frame.loc[:, ['loss', 'val_loss']].plot()
plt.title('Training Loss vs Validation Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')

```

```
history_frame.loc[:, ['accuracy', 'val_accuracy']].plot();
plt.title('Training Accuracy vs Validation Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
```



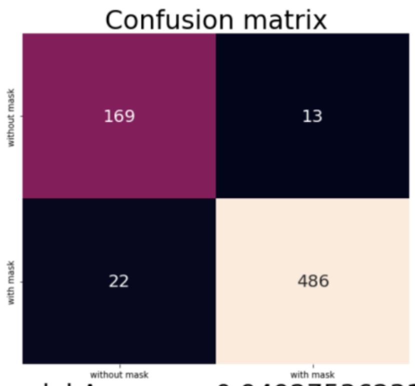
```
y_pred=model.predict(X_val)
df1=pd.DataFrame()
df1['classname'] = [1 if i > 0.8 else 0 for i in y_pred]
df1
```

```
#Converting y prediction dataframe into array
y_pred=np.array(df1)
y_pred
```

```
from sklearn.metrics import classification_report
y_true = y_val
target_names = ['without mask', 'with mask']
print(classification_report(y_true, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

without mask       0.88      0.93      0.91       182
   with mask       0.97      0.96      0.97       508

    accuracy                           0.95       690
   macro avg       0.93      0.94      0.94       690
weighted avg       0.95      0.95      0.95       690
```

```
from sklearn.metrics import confusion_matrix, accuracy_score
plt.subplots(figsize=(8,7))
sns.heatmap(confusion_matrix(y_true,y_pred),xticklabels=target_names, yticklabels=target
_names, annot=True,fmt="1.0f",cbar=False,annot_kws={"size": 20})
plt.title("Confusion matrix",fontsize=30)
plt.xlabel(f"CNN model Accuracy: {accuracy_score(y_val,y_pred)}",fontsize=30)
plt.show()
```



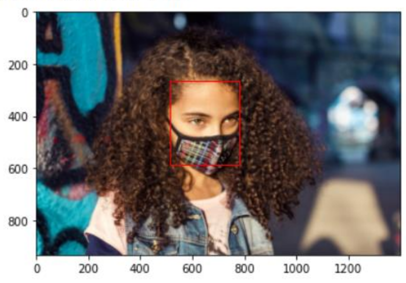
CNN model Accuracy: 0.9492753623188406

```
#Preprocessing test images
pip install mtcnn
```

```
def draw_facebox(image, boxes):
    # plot the image
    plt.imshow(image)
    # get the context for drawing boxes
    ax = plt.gca()
    # plot each box
    for box in boxes:
    # get coordinates
        x, y, width, height = box[0], box[1], box[2], box[3],
        # create the shape
        rect = plt.Rectangle((x, y), width, height,
                             fill=False, color='r', linewidth=1)
        # draw the box
        ax.add_patch(rect)
    # show the plot
    plt.show()
```

```
#Plotting image randomly with bounding box
from mtcnn.mtcnn import MTCNN
detector = MTCNN()

i = np.random.choice(len(test_images))
# load image from file
image = plt.imread(os.path.join(img_dir, df.name[i]))
# detect faces in the image
faces = detector.detect_faces(image)
boxes = [face['box'] for face in faces if face['confidence']>0.99]
print(boxes)
# display faces on the original image
draw_facebox(image, boxes)
```

```
[[514, 263, 268, 321]]
```

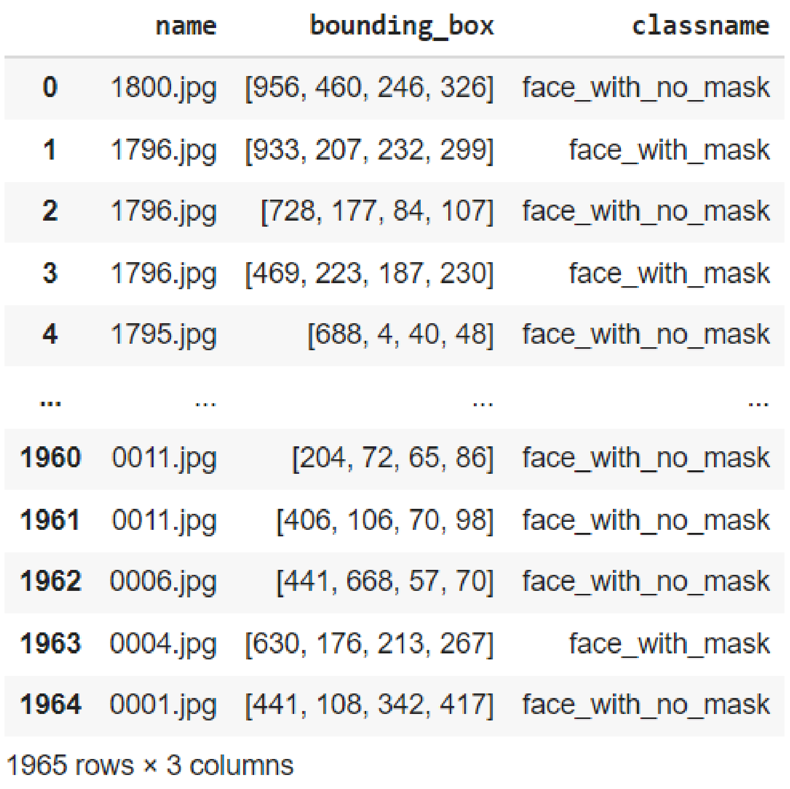


```python
# create lists which would contain filenames of images and bounding boxes
names = []
bboxes = []
for img_name in test_images:
    # load image from file
    image = plt.imread(os.path.join(img_dir, img_name))
    # detect faces in the image
    faces = detector.detect_faces(image)
    for face in faces:
        if face['confidence']>0.99:
            names.append(img_name)
            bboxes.append(face['box'])

df = pd.DataFrame({'name' : names, 'bounding_box' : bboxes})
df.head()
```

	name	bounding_box
0	1800.jpg	[956, 460, 246, 326]
1	1796.jpg	[933, 207, 232, 299]
2	1796.jpg	[728, 177, 84, 107]
3	1796.jpg	[469, 223, 187, 230]
4	1795.jpg	[688, 4, 40, 48]

```python
data=[]
for i in range(len(df)):
    # replace any negative value with zero
    x,y,width,height = [0 if value < 0 else value for value in df.iloc[i]['bounding_box']]
    image = df.iloc[i]['name']
    # read image with green channel
    img_array = cv2.imread(os.path.join(img_dir,image), 1)
    # crop image with bounding box
    img_cropped = img_array[y:y+height,x:x+width]
    # resize cropped image
    img = cv2.resize(img_cropped,(img_size,img_size))
    data.append(img)
```

```python
#predict classes
X = np.array(data)/255
predict = model.predict(X)
df['classname'] = ['face_with_mask' if i > 0.8 else 'face_with_no_mask' for i in predict]
df
```



	name	bounding_box	classname
0	1800.jpg	[956, 460, 246, 326]	face_with_no_mask
1	1796.jpg	[933, 207, 232, 299]	face_with_mask
2	1796.jpg	[728, 177, 84, 107]	face_with_no_mask
3	1796.jpg	[469, 223, 187, 230]	face_with_mask
4	1795.jpg	[688, 4, 40, 48]	face_with_no_mask
...	...	...	...
1960	0011.jpg	[204, 72, 65, 86]	face_with_no_mask
1961	0011.jpg	[406, 106, 70, 98]	face_with_no_mask
1962	0006.jpg	[441, 668, 57, 70]	face_with_no_mask
1963	0004.jpg	[630, 176, 213, 267]	face_with_mask
1964	0001.jpg	[441, 108, 342, 417]	face_with_no_mask

1965 rows × 3 columns

```
# save the result to csv
df.to_csv('submission_1.csv')
```

Classification Using Vgg16 Pre-trained model:

```
#Setting path for json file
def getJSON(filePathandName):
    with open(filePathandName,'r') as f:
        return json.load(f)
jsonfiles= []
for i in os.listdir(directory):
    jsonfiles.append(getJSON(os.path.join(directory,i)))
jsonfiles[0]
```

```
{'Annotations': [{'Attributes': {},
   'BoundingBox': [414, 236, 477, 319],
   'Confidence': 1,
   'ID': 139738413658914480,
   'classname': 'face_with_mask',
   'isProtected': False},
  {'Attributes': {},
   'BoundingBox': [290, 69, 391, 192],
   'Confidence': 1,
   'ID': 990667606318341760,
   'classname': 'face_with_mask',
   'isProtected': False},
  {'Attributes': {},
   'BoundingBox': [420, 288, 471, 310],
   'Confidence': 1,
   'ID': 470075866928222592,
   'classname': 'mask_surgical',
   'isProtected': False},
  {'Attributes': {},
   'BoundingBox': [310, 149, 354, 191],
   'Confidence': 1,
   'ID': 438931347460153472,
   'classname': 'mask_colorful',
   'isProtected': False}],
 'FileName': '3933.png',
 'NumOfAnno': 4}
```

```
#Accessing and resizing data for vgg16
data = []
img_size = 224
mask = ['face_with_mask']
non_mask = ["face_no_mask"]
labels={'mask':0,'without mask':1}
for i in train_full["name"].unique():
    f = i+".json"
    for j in getJSON(os.path.join(directory,f)).get("Annotations"):
        if j["classname"] in mask:
            x,y,w,h = j["BoundingBox"]
            img = cv2.imread(os.path.join(img_dir,i),1)
            img = img[y:h,x:w]
            img = cv2.resize(img,(img_size,img_size))
            img = preprocess_input(img)
            data.append([img,labels["mask"]])
        if j["classname"] in non_mask:
            x,y,w,h = j["BoundingBox"]
            img = cv2.imread(os.path.join(img_dir,i),1)
            img = img[y:h,x:w]
            img = cv2.resize(img,(img_size,img_size))
            img = preprocess_input(img)
            data.append([img,labels["without mask"]])
random.shuffle(data)
```

```
X = []
Y = []
for features,label in data:
    X.append(features)
    Y.append(label)
X = np.array(X)/255
print('Shape of X:', X.shape)
Y = np.array(Y)
print('Shape of Y:', Y.shape)
```

```
Shape of X: (5749, 224, 224, 3)
Shape of Y: (5749,)
```

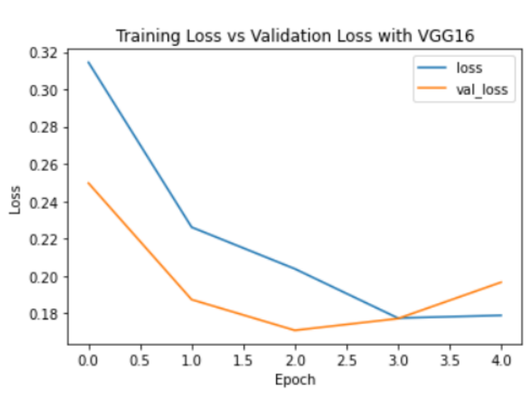
```python
# split our data into train and validation sets
X_train,X_val,y_train,y_val = train_test_split(X, Y,test_size=0.12,random_state=42)
```

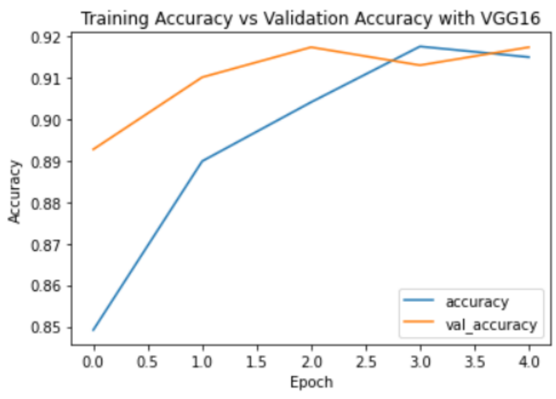
```python
# load model
vgg = VGG16(weights="imagenet", include_top=False, input_shape=(img_size, img_size, 3))
for layer in vgg.layers:
    layer.trainable = False
top = vgg.output
top = GlobalAveragePooling2D()(top)
top = Dense(units=256, activation="relu")(top)
top = Dense(units=128, activation="relu")(top)
top = Dense(units=1, activation="sigmoid")(top)

model1 = Model(inputs=vgg.input, outputs=top)
print(model1.summary())
```

```python
#Compiling and fitting the model
model1.compile(
  loss='binary_crossentropy',
  optimizer='adam',
  metrics=['accuracy']
)
hist = model1.fit(X_train,y_train,batch_size=32,
                 epochs=5,
                 validation_data=(X_val, y_val))
```

```
Epoch 1/5
159/159 [==============================] - 2536s 16s/step - loss: 0.3144 - accuracy: 0.8492 - val_loss: 0.2497 - val_accuracy: 0.8928
Epoch 2/5
159/159 [==============================] - 2530s 16s/step - loss: 0.2261 - accuracy: 0.8899 - val_loss: 0.1874 - val_accuracy: 0.9101
Epoch 3/5
159/159 [==============================] - 2529s 16s/step - loss: 0.2039 - accuracy: 0.9041 - val_loss: 0.1709 - val_accuracy: 0.9174
Epoch 4/5
159/159 [==============================] - 2533s 16s/step - loss: 0.1775 - accuracy: 0.9176 - val_loss: 0.1772 - val_accuracy: 0.9130
Epoch 5/5
159/159 [==============================] - 2517s 16s/step - loss: 0.1789 - accuracy: 0.9150 - val_loss: 0.1967 - val_accuracy: 0.9174
```

```python
history_frame = pd.DataFrame(hist.history)
history_frame.loc[:, ['loss', 'val_loss']].plot()
plt.title('Training Loss vs Validation Loss with VGG16')
plt.ylabel('Loss')
plt.xlabel('Epoch')

history_frame.loc[:, ['accuracy', 'val_accuracy']].plot();
plt.title('Training Accuracy vs Validation Accuracy with VGG16')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
```



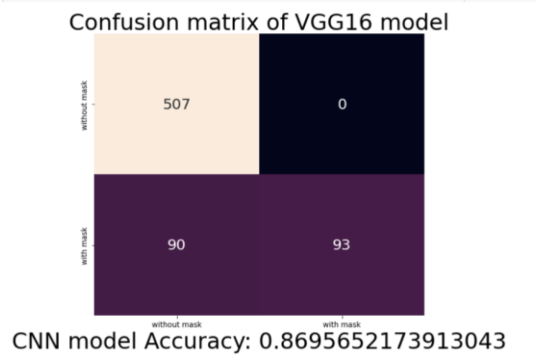


```python
y_pred=model1.predict(X_val)
df1=pd.DataFrame()
df1['classname'] = [1 if i > 0.8 else 0 for i in y_pred]
df1
y_pred=np.array(df1)
y_pred
```

```python
y_true = y_val
target_names = ['without mask', 'with mask']
print(classification_report(y_true, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

without mask       0.85      1.00      0.92       507
   with mask       1.00      0.51      0.67       183

    accuracy                           0.87       690
   macro avg       0.92      0.75      0.80       690
weighted avg       0.89      0.87      0.85       690
```

```python
from sklearn.metrics import confusion_matrix, accuracy_score
plt.subplots(figsize=(8,7))
sns.heatmap(confusion_matrix(y_true,y_pred),xticklabels=target_names, yticklabels=target_names, annot=True,fmt="1.0f",cbar=False,annot_kws={"size": 20})
plt.title("Confusion matrix of VGG16 model",fontsize=30)
plt.xlabel(f"CNN model Accuracy: {accuracy_score(y_val,y_pred)}",fontsize=30)
plt.show()
```



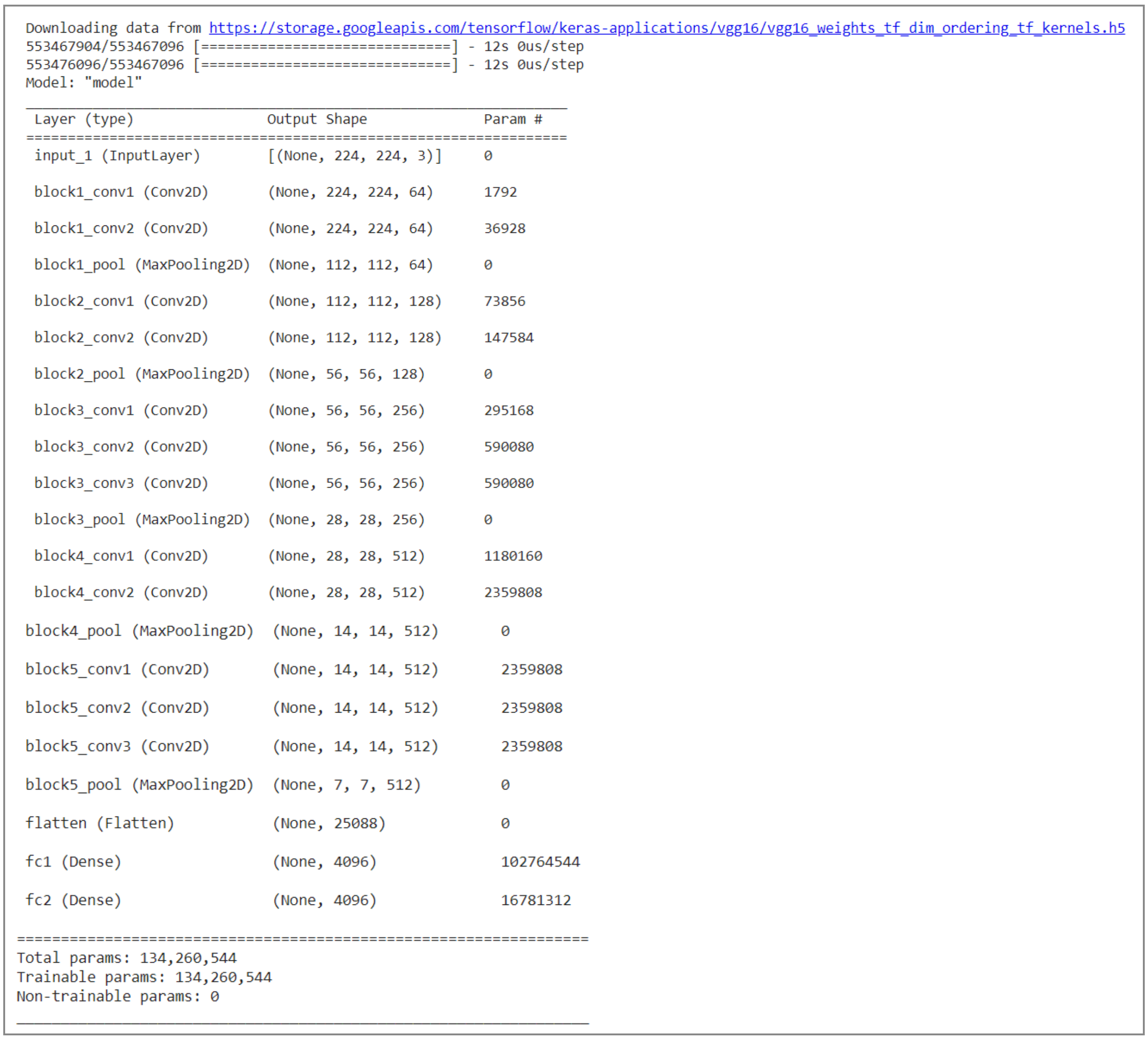
Task-2 : Clustering :-

For clustering, Pretrained vgg16 model was used for extracting features. So, model made for classification was used.

```
#improve brightness and contrast
def adjust_gamma(image, gamma=1.0):
    invGamma = 1.0 / gamma
    table = np.array([((i / 255.0) ** invGamma) * 255 for i in np.arange(0, 256)])
    return cv2.LUT(image.astype(np.uint8), table.astype(np.uint8))
```

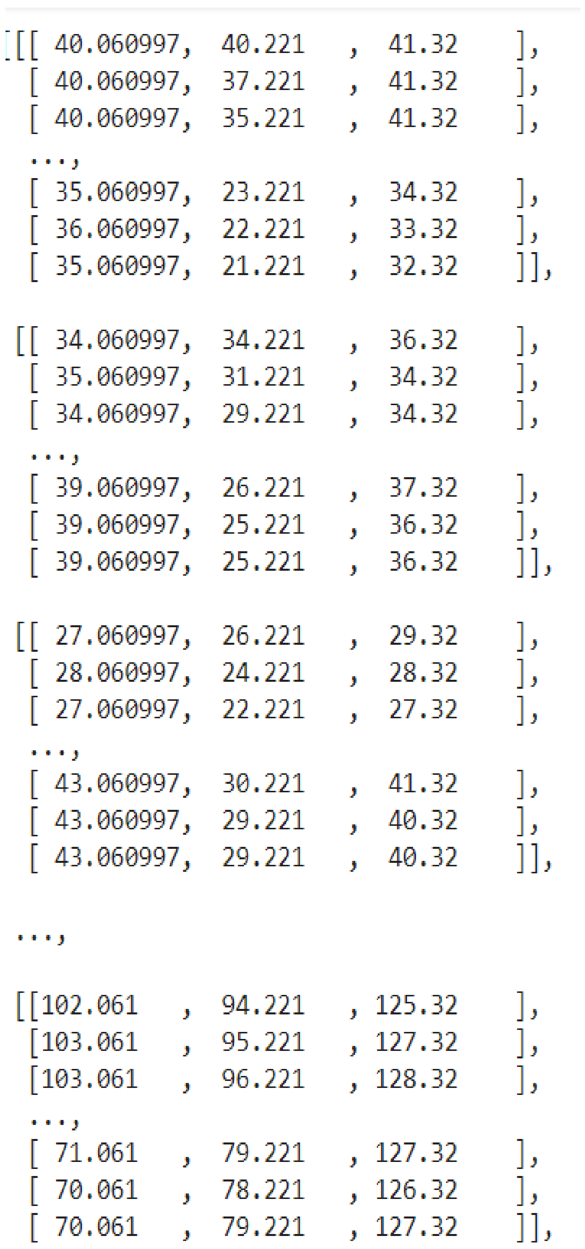
```
# load model
model = VGG16()
# remove the output layer
model = Model(inputs=model.inputs, outputs=model.layers[-2].output)
model.summary()
```

```
#defining functions for extracting features
def extract_features(file, model):
    features = model.predict(file, use_multiprocessing=True)
    return features
data = {}
for facemask in range(len(X)):

    # try to extract the features and update the dictionary
      print(facemask)
      dat=X[facemask]
      print(dat.shape)
      feat = extract_features(dat,model)
      data[facemask] = feat
```

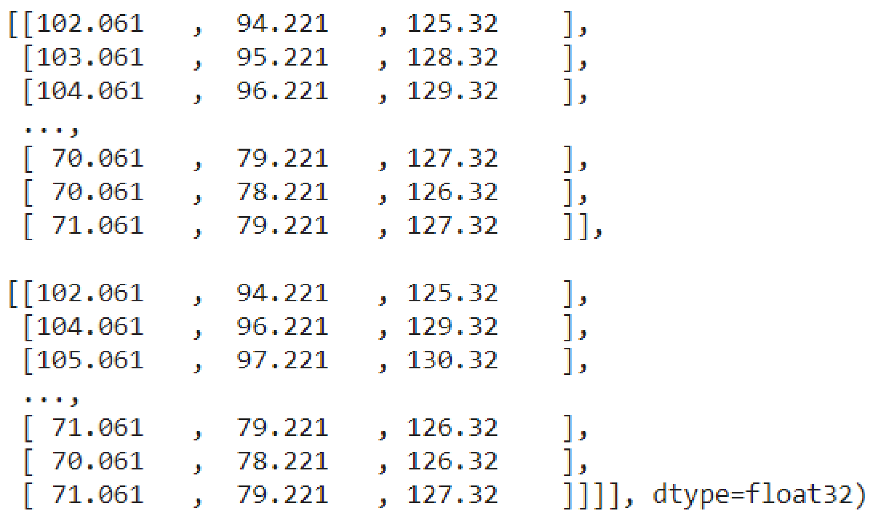


```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels.h5
553467904/553467096 [==============================] - 12s 0us/step
553476096/553467096 [==============================] - 12s 0us/step
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

block4_pool (MaxPooling2D)  (None, 14, 14, 512)        0

block5_conv1 (Conv2D)       (None, 14, 14, 512)        2359808

block5_conv2 (Conv2D)       (None, 14, 14, 512)        2359808

block5_conv3 (Conv2D)       (None, 14, 14, 512)        2359808

block5_pool (MaxPooling2D)  (None, 7, 7, 512)          0

flatten (Flatten)           (None, 25088)              0

fc1 (Dense)                 (None, 4096)               102764544

fc2 (Dense)                 (None, 4096)               16781312

=================================================================
Total params: 134,260,544
Trainable params: 134,260,544
Non-trainable params: 0
_________________________________________________________________
```

```
1.  features
```



```
[[[ 40.060997,  40.221    ,  41.32     ],
  [ 40.060997,  37.221    ,  41.32     ],
  [ 40.060997,  35.221    ,  41.32     ],
  ...,
  [ 35.060997,  23.221    ,  34.32     ],
  [ 36.060997,  22.221    ,  33.32     ],
  [ 35.060997,  21.221    ,  32.32     ]],

 [[ 34.060997,  34.221    ,  36.32     ],
  [ 35.060997,  31.221    ,  34.32     ],
  [ 34.060997,  29.221    ,  34.32     ],
  ...,
  [ 39.060997,  26.221    ,  37.32     ],
  [ 39.060997,  25.221    ,  36.32     ],
  [ 39.060997,  25.221    ,  36.32     ]],

 [[ 27.060997,  26.221    ,  29.32     ],
  [ 28.060997,  24.221    ,  28.32     ],
  [ 27.060997,  22.221    ,  27.32     ],
  ...,
  [ 43.060997,  30.221    ,  41.32     ],
  [ 43.060997,  29.221    ,  40.32     ],
  [ 43.060997,  29.221    ,  40.32     ]],


 ...,

 [[102.061    ,  94.221    , 125.32     ],
  [103.061    ,  95.221    , 127.32     ],
  [103.061    ,  96.221    , 128.32     ],
  ...,
  [ 71.061    ,  79.221    , 127.32     ],
  [ 70.061    ,  78.221    , 126.32     ],
  [ 70.061    ,  79.221    , 127.32     ]],
```



```
 [[102.061    ,  94.221    , 125.32     ],
  [103.061    ,  95.221    , 128.32     ],
  [104.061    ,  96.221    , 129.32     ],
  ...,
  [ 70.061    ,  79.221    , 127.32     ],
  [ 70.061    ,  78.221    , 126.32     ],
  [ 71.061    ,  79.221    , 127.32     ]],

 [[102.061    ,  94.221    , 125.32     ],
  [104.061    ,  96.221    , 129.32     ],
  [105.061    ,  97.221    , 130.32     ],
  ...,
  [ 71.061    ,  79.221    , 126.32     ],
  [ 70.061    ,  78.221    , 126.32     ],
  [ 71.061    ,  79.221    , 127.32     ]]]], dtype=float32)
```

```
# get a list of the filenames
filenames = np.array(list(data.keys()))

# get a list of just the features
feat = np.array(list(data.values()))

# reshape so that there are 210 samples of 4096 vectors
feat = feat.reshape(-1,4096)

label=labels.keys()
unique_labels = list(set(label))

#Principal Component Analysis
pca = PCA(n_components=100, random_state=22)
pca.fit(feat)
x = pca.transform(feat)
```

```
#Forming clusters
kmeans = KMeans(n_clusters=len(unique_labels))
km=kmeans.fit(x)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=2, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

```
# holds the cluster id and the images { id: [images] }
groups = {}
for file, cluster in zip(X,kmeans.labels_):
    if cluster not in groups.keys():
        groups[cluster] = []
        groups[cluster].append(file)
    else:
        groups[cluster].append(file)
```

```
groups[0][0].shape
```

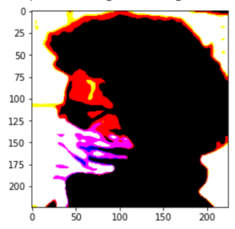
```
(1, 224, 224, 3)
```

```
groups[0][0]
```

```
array([[[[-57.939003 , -66.779    , -75.68     ],
         [-57.939003 , -66.779    , -75.68     ],
         [-57.939003 , -66.779    , -75.68     ],
         ...,
         [-53.939003 , -59.779    , -72.68     ],
         [-53.939003 , -59.779    , -72.68     ],
         [-53.939003 , -59.779    , -72.68     ]],

        [[-57.939003 , -66.779    , -75.68     ],
         [-57.939003 , -66.779    , -75.68     ],
         [-57.939003 , -66.779    , -75.68     ],
         ...,
         [-53.939003 , -59.779    , -72.68     ],
         [-53.939003 , -59.779    , -72.68     ],
         [-53.939003 , -59.779    , -72.68     ]],

        [[-57.939003 , -66.779    , -75.68     ],
         [-57.939003 , -66.779    , -75.68     ],
         [-57.939003 , -66.779    , -75.68     ],
         ...,
         [-53.939003 , -59.779    , -72.68     ],
         [-53.939003 , -59.779    , -72.68     ],
         [-53.939003 , -59.779    , -72.68     ]],

        ...,

        [[-65.939    , -82.779    , -84.68     ],
         [-65.939    , -82.779    , -84.68     ],
         [-65.939    , -82.779    , -84.68     ],
         ...,
         [ 12.060997 ,  -3.7789993,  -4.6800003],
         [ 12.060997 ,  -2.7789993,  -4.6800003],
         [ 12.060997 ,  -2.7789993,  -4.6800003]],

        [[-65.939    , -82.779    , -84.68     ],
         [-65.939    , -82.779    , -84.68     ],
         [-65.939    , -82.779    , -84.68     ],
         ...,
         [ 11.060997 ,  -3.7789993,  -5.6800003],
         [ 11.060997 ,  -3.7789993,  -5.6800003],
         [ 11.060997 ,  -3.7789993,  -5.6800003]],

        [[-65.939    , -82.779    , -84.68     ],
         [-65.939    , -82.779    , -84.68     ],
         [-65.939    , -82.779    , -84.68     ],
         ...,
         [ 11.060997 ,  -3.7789993,  -5.6800003],
         [ 11.060997 ,  -3.7789993,  -5.6800003],
         [ 11.060997 ,  -3.7789993,  -5.6800003]]]], dtype=float32)
```

```
plt.imshow(groups[0][70][0])
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
<matplotlib.image.AxesImage at 0x7f5a1a0bdcd0>
```



```
def view_cluster(cluster):
    for i in range(1, 7):
        plt.figure(figsize = (5,5))
        plt.imshow(groups[cluster][i][0])
        plt.axis('off')
view_cluster(0)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
view_cluster(1)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```python
y_true = Y
y_pred = kmeans.labels_
target_names = ['without mask', 'with mask']
print(classification_report(y_true, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

without mask       0.72      0.61      0.66      4180
   with mask       0.27      0.38      0.31      1569

    accuracy                           0.55      5749
   macro avg       0.49      0.49      0.49      5749
weighted avg       0.60      0.55      0.57      5749
```