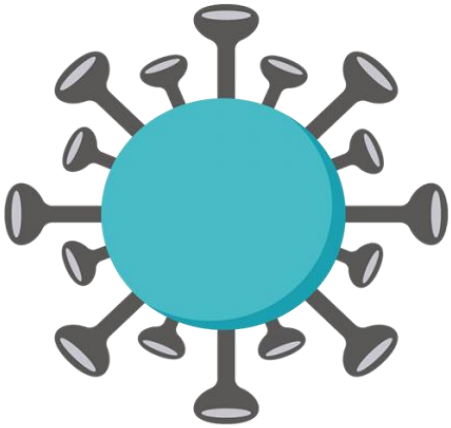
Jawher Sakka

This document shows some analysis of corona propagation in the world and presents a method of building a Face Mask Detector using Convolutional Neural Networks (CNN) Python, Keras, Tensorflow and OpenCV. With further improvements these types of models could be integrated with CCTV or other types cameras to detect and identify people without masks. With the prevailing worldwide situation due to COVID-19 pandemic, these types of systems would be very supportive for many kind of institutions around the world.



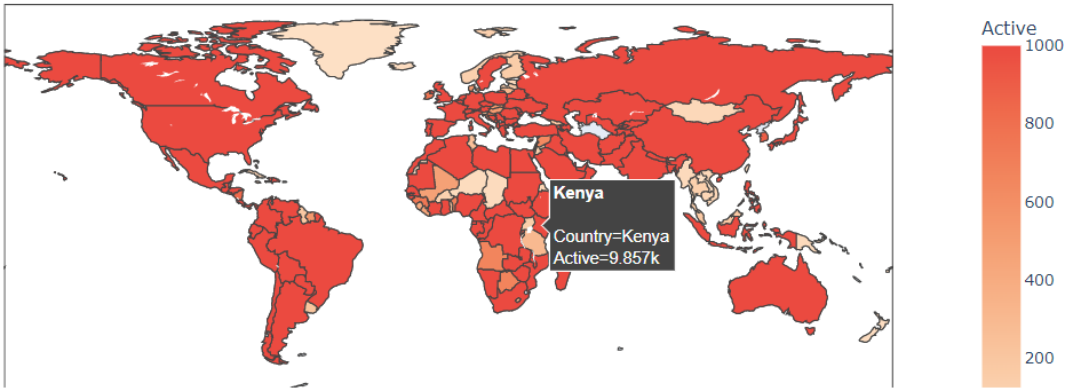
*Read Dataset and show the propagation of Covid-19 in the map:

```
df = pd.read_csv('covid_19_clean_complete.csv',parse_dates=['Date'])
df.head()
```

	Province/State	Country/Region	Lat	Long	Date	Confirmed	Deaths	Recovered	Active	WHO Region
0	NaN	Afghanistan	33.93911	67.709953	2020-01-22	0	0	0	0	Eastern Mediterranean
1	NaN	Albania	41.15330	20.168300	2020-01-22	0	0	0	0	Europe
2	NaN	Algeria	28.03390	1.659600	2020-01-22	0	0	0	0	Africa
3	NaN	Andorra	42.50630	1.521800	2020-01-22	0	0	0	0	Europe
4	NaN	Angola	-11.20270	17.873900	2020-01-22	0	0	0	0	Africa

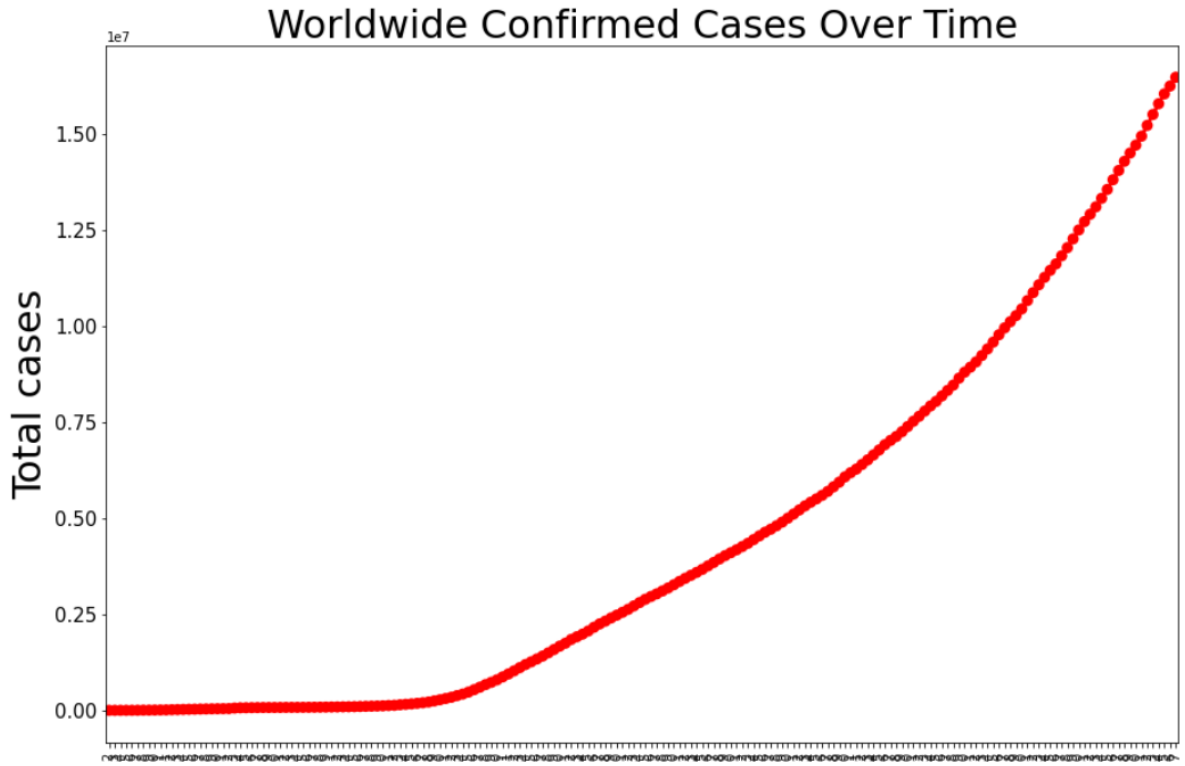
```
df.rename(columns={'Country/Region' :'Country'},inplace=True)
#df['Active']=df['Confirmed']-df['Deaths']-df['Recovered']
```

```
figure= px.choropleth(world,locations='Country',
                      locationmode='country names',color='Active',hover_name='Country',
                      range_color=[1,1000],color_continuous_scale='Peach',title='countries with active cases')
figure.show()
```



```
plt.figure(figsize=(15,10))
plt.xticks(rotation=90,fontsize=10)
plt.yticks(fontsize=15)
plt.xlabel('Dates',fontsize=30)
plt.ylabel('Total cases',fontsize=30)
plt.title('Worldwide Confirmed Cases Over Time',fontsize=30)
total_cases=df.groupby('Date')['Date','Confirmed'].sum().reset_index()
total_cases['Date']=pd.to_datetime(total_cases['Date'])
ax=sns.pointplot(x=total_cases.Date.dt.date,y=total_cases.Confirmed,color='r')
ax.set(xlabel='Dates',ylabel='Total cases')
```

```
[Text(0.5, 0, 'Dates'), Text(0, 0.5, 'Total cases')]
```



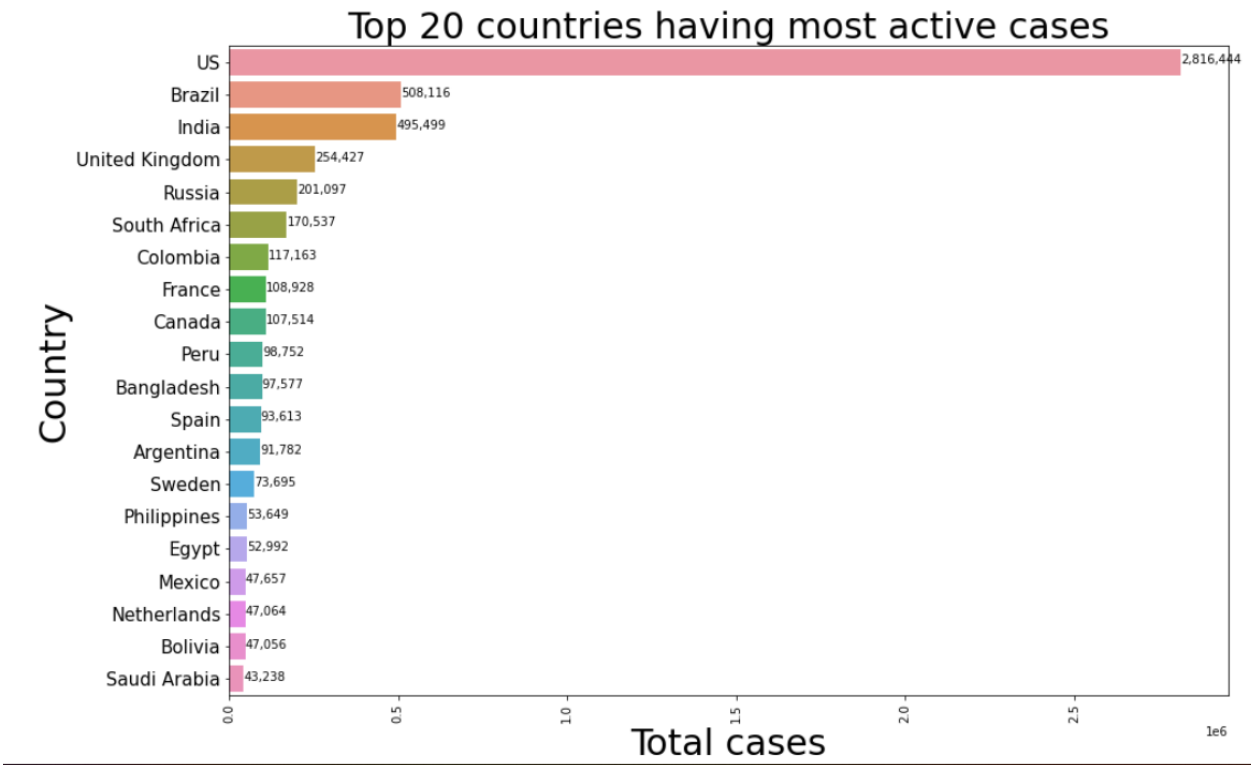

```
tot_cases=df.groupby(by='Country')['Active'].sum().sort_values(ascending=False).to_frame()
tot_cases.style.background_gradient(cmap='Blues_r')
```

Country	Active
US	156981121
Brazil	31094060
United Kingdom	22624595
Russia	19668578
India	15987913
France	10980287
Spain	9277432
Canada	8656985
Peru	7748957
Italy	7363518
Pakistan	5633262

*top 20 countries having most active cases:

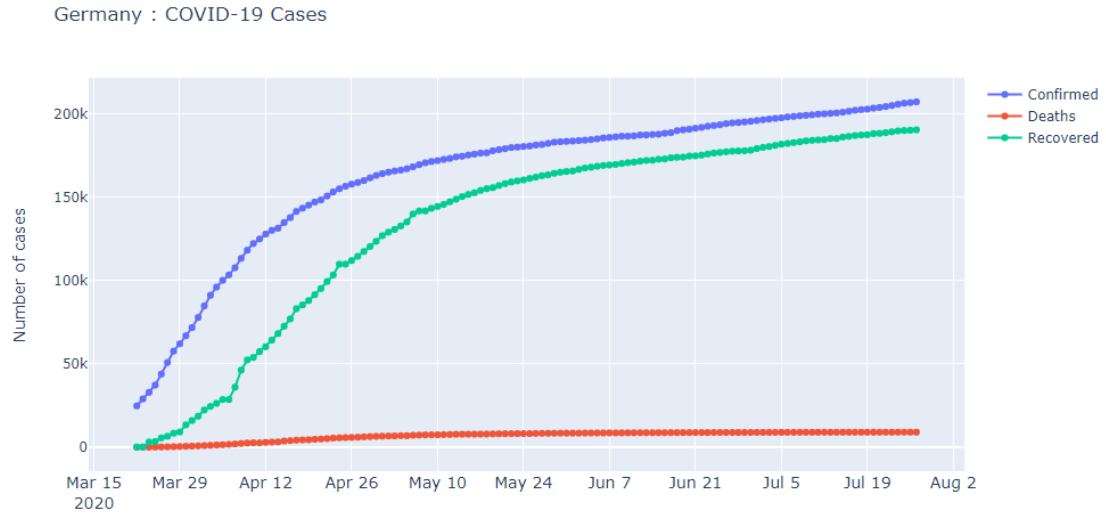
Active cases = confirmed cases – death cases – recovered cases

```
plt.figure(figsize=(15,10))
plt.xticks(rotation=90,fontsize=10)
plt.yticks(fontsize=15)
plt.xlabel('Total cases',fontsize=30)
plt.ylabel('Countries',fontsize=30)
plt.title('Top 20 countries having most active cases',fontsize=30)
top_actives= top.groupby(by='Country')['Active'].sum().sort_values(ascending=False).head(20).reset_index()
ax=sns.barplot(x=top_actives.Active,y=top_actives.Country)
for i , (value,name) in enumerate (zip(top_actives.Active,top_actives.Country)):
    ax.text(value,i-.05,f'{value:,.0f}',size=10, ha='left',va='center')
ax.set(xlabel='Total cases',ylabel='Country')
```

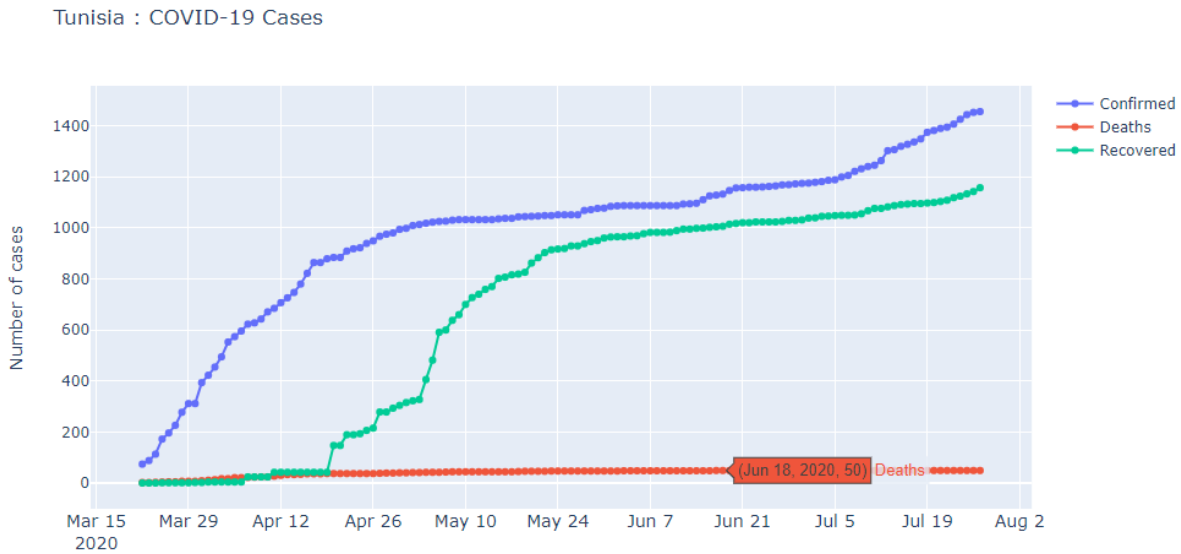


COVID-19 Cases in GERMANY

In [288]:
```
Germany=df[df.Country=='Germany']
Germany = Germany.groupby(by='Date')['Recovered','Deaths','Confirmed','Active'].sum().reset_index()
Germany=Germany.iloc[60:].reset_index().drop('index',axis=1)
fig=go.Figure()
fig.add_trace(go.Scatter(x=Germany['Date'],y=Germany['Confirmed'],mode='lines+markers',name='Confirmed'))
fig.add_trace(go.Scatter(x=Germany['Date'],y=Germany['Deaths'],mode='lines+markers',name='Deaths'))
fig.add_trace(go.Scatter(x=Germany['Date'],y=Germany['Recovered'],mode='lines+markers',name='Recovered'))
fig.update_layout(title='Germany : COVID-19 Cases',xaxis_tickfont_size=14,yaxis=dict(title='Number of cases'))
fig.show()
```



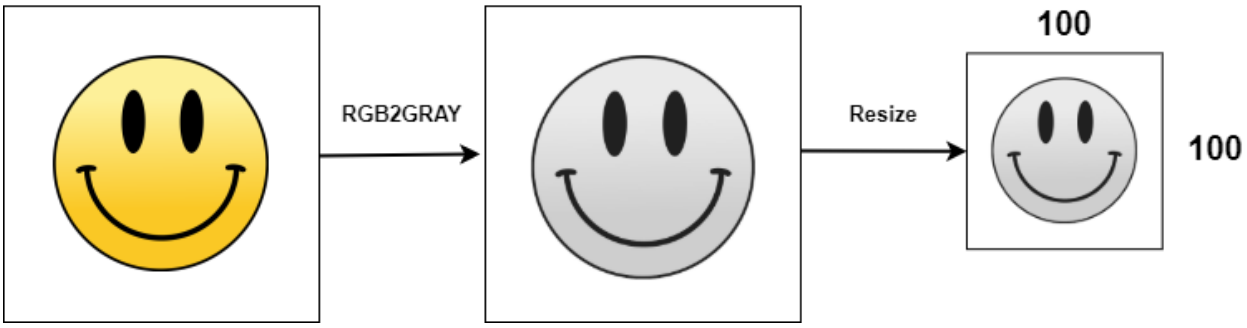
COVID-19 Cases in TUNISIA



Jawher Sakka

Real Time Face Face Mask Detection

1-Data Preprocessing

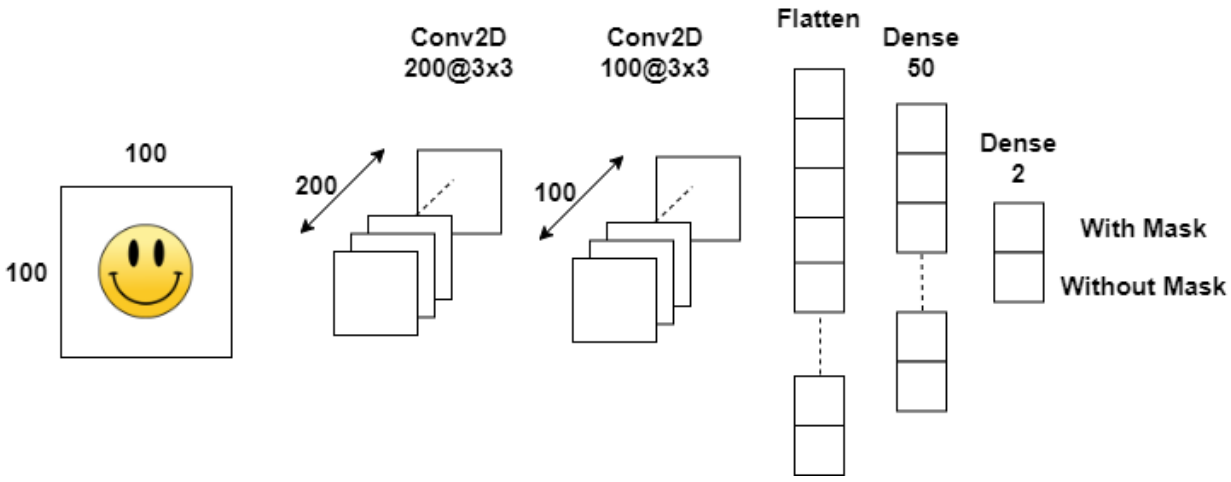


In [77]:
```
import cv2,os

data_path='dataset'
categories=os.listdir(data_path)
labels=[i for i in range(len(categories))]

label_dict=dict(zip(categories,labels)) #empty dictionary

print(label_dict)
print(categories)
print(labels)
```

```
{'with mask': 0, 'without mask': 1}
['with mask', 'without mask']
[0, 1]
```

In [124]:
```
img_size=100
data=[]
target=[]


for category in categories:
    folder_path=os.path.join(data_path,category)
    img_names=os.listdir(folder_path)

    for img_name in img_names:
        img_path=os.path.join(folder_path,img_name)
        img=cv2.imread(img_path)

        try:
            gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
            #Coverting the image into gray scale
            resized=cv2.resize(gray,(img_size,img_size))
            #resizing the gray scale into 50x50, since we need a fixed common size for all the images in the dataset
            data.append(resized)
            target.append(label_dict[category])
            #appending the image and the label(categorized) into the list (dataset)

        except Exception as e:
            print('Exception:',e)
```

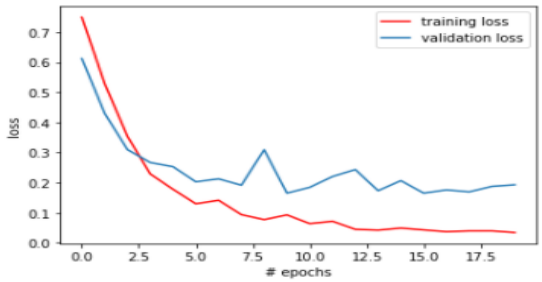
2-Training the CNN:



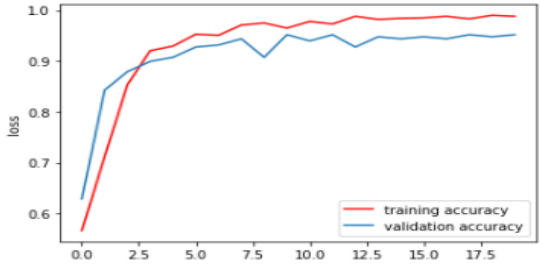

```
import numpy as np

data=np.load('data.npy')
target=np.load('target.npy')
```

```
from keras.models import Sequential
from keras.layers import Dense,Activation,Flatten,Dropout
from keras.layers import Conv2D,MaxPooling2D
from keras.callbacks import ModelCheckpoint

model=Sequential()

model.add(Conv2D(200,(3,3),input_shape=data.shape[1:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
#The first CNN layer followed by Relu and MaxPooling layers

model.add(Conv2D(100,(3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
#The second convolution layer followed by Relu and MaxPooling layers

model.add(Flatten())
model.add(Dropout(0.5))
#Flatten layer to stack the output convolutions from second convolution layer
model.add(Dense(50,activation='relu'))
#Dense layer of 64 neurons
model.add(Dense(2,activation='softmax'))
#The Final layer with two outputs for two categories

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

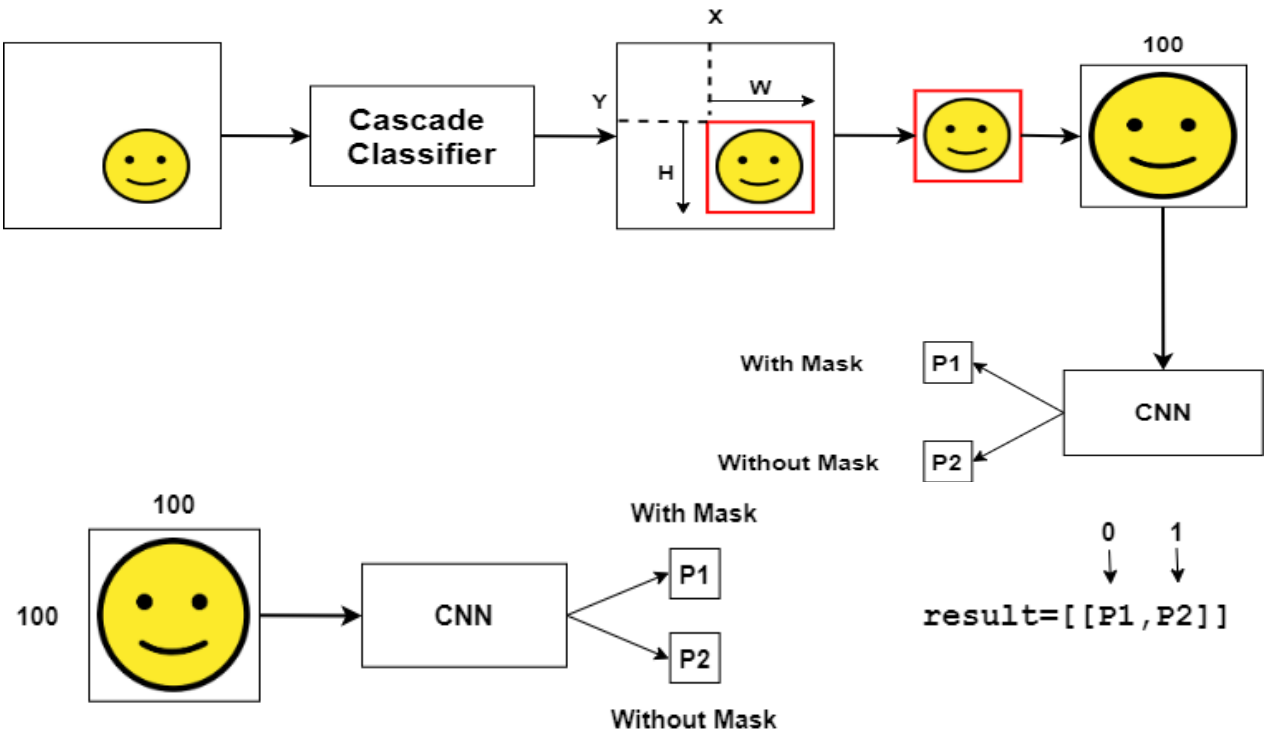
```
In [85]: from matplotlib import pyplot as plt

plt.plot(history.history['loss'],'r',label='training loss')
plt.plot(history.history['val_loss'],label='validation loss')
plt.xlabel('# epochs')
plt.ylabel('loss')
plt.legend()
plt.show()
```




```
In [86]: plt.plot(history.history['accuracy'],'r',label='training accuracy')
plt.plot(history.history['val_accuracy'],label='validation accuracy')
plt.xlabel('# epochs')
plt.ylabel('loss')
plt.legend()
plt.show()
```



3-Detecting Masks in Real Time:



In [131]:
```
from keras.models import load_model
import cv2
import numpy as np
```

In [132]:
```
model = load_model('model-017.model')

face_clsfr=cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

source=cv2.VideoCapture(0)

labels_dict={0:'MASK',1:'NO MASK'}
color_dict={0:(0,255,0),1:(0,0,255)}
```

In [133]:
```
while True:

    ret,img=source.read()
    gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    faces=face_clsfr.detectMultiScale(gray,1.3,5)

    for (x,y,w,h) in faces:

        face_img=gray[y:y+w,x:x+w]
        resized=cv2.resize(face_img,(100,100))
        normalized=resized/255.0
        reshaped=np.reshape(normalized,(1,100,100,1))
        result=model.predict(reshaped)

        label=np.argmax(result,axis=1)[0]

        cv2.rectangle(img,(x,y),(x+w,y+h),color_dict[label],2)
        cv2.rectangle(img,(x,y-40),(x+w,y),color_dict[label],-1)
        cv2.putText(img, labels_dict[label], (x, y-10),cv2.FONT_HERSHEY_SIMPLEX,0.8,(255,255,255),2)


    cv2.imshow('LIVE',img)
    key=cv2.waitKey(1)

    if(key==27):
        break

cv2.destroyAllWindows()
source.release()
```