“Gheorghe Asachi” Technical University of Iasi
Faculty of Electronics, Telecommunications and Information Technology

Bachelor thesis

Straticiuc Mihai

Detection of the Medical Mask using Image Recognition

Thesis supervisor: Dr. Lecturer Eng. Nicolaie Cleju

Study Program: Telecommunication Technologies and Systems

2021

I declare that I wrote this Bachelor thesis by myself and using only referenced sources.

Date:	Signature:
10rd of July, 2021	Straticiuc Mihai

Table of Contents

Title: Detection of the Medical Mask using Image Recognition

Author: Straticiuc Mihai

Department: Department of Telecommunication

Program of study: Telecommunication Technologies and Systems

Thesis supervisor: Dr. Lecturer Eng. Nicolaie Cleju

Abstract

Recognition if a person wear the medical mask or not become very important in the last year due to the appearance of the COVID-19. Nowadays recognition through the images become one of the most challenging domain that implies image processing, using of the artificial neural networks and pattern recognition.

In this project we will focus on Medical Mask wearing recognition from an live stream video in the real time with the help of trained models. The models for recognition will be trained using python framework named Keras, the image capturing from a camera will be done using OpenCV.
In order to train a model to recognize if in a image the person wear the mask correctly or not we will used a data set where we already have images with people wearing the mask or not.
After the process of training is finished we obtain a model that can be used independently, with the help of the model we have write another program that capture with OpenCV images from the video camera and pass it to the model that decide if the person from the picture wearing the mask or not.

At he end an video stream will be displayed where in real time we can see if thee person wear the mask or not. Do to the slow computers we can’t train all models and we was forced to use per-trained models.

Introduction

The purpose of the project is to train models that will be used to build a program that cam be easily integrated in public places in order to motorize if peoples wear mask when they are in magazines, mall or any other place with high concentration of people.

As we already know COVID-19 is very contagious and one of the solution to decrease the transmission rate is to wear masks in public places. Unfortunately there is a problem “How we detect if a person wear the mask or not ?”.

“Masks are a key measure to suppress transmission and save lives.
Masks should be used as part of a comprehensive **‘Do it all!’** approach including physical distancing, avoiding crowded, closed and close-contact settings, good ventilation, cleaning hands, covering sneezes and coughs, and more. Depending on the type, masks can be used for either protection of healthy persons or to prevent onward transmission.”

If we will look at the global statistics we can observe that the lowest rate of infection through all countries was int Sought Korea , China and Japan where system which recognize if a person wear the mask or not was implemented as soon as possible and integrated everywhere where is a systems with video cameras.
This is the best prove that such systems are very useful in case of pandemic situations and even if we will succeed to pass over current pandemic such systems can be used in the future.

Even if Mask recognition systems seems to be very useful in case of thee COVID-19 there are still a lot of discussion about the legal fairness of integration of such systems as permanent one.
The main problem is that: through such system we can easily collect data about person movements and like with other data it can be used for good purpose or for bad ones.

Even if the legal documentation is not ready and there are a lot of discussion about the fairness the current pandemic prove that in order to be able to deal with such crisis's we need such systems
”Complex problems require complex solutions”.

This thesis paper will be divided in three main chapters as following.

The first chapter entitled **Fundamentals of Deep Learning** where will be explained simple and easy what deep learning means and how it works. Also we will try to explain what is a neural network its characteristics and how it is implemented at he level of the neuron. Briefly we will explain methods used in Deep Learning, the difference between Deep Learning and Machine Learning and what are the future perspective of such technology.

The second chapter is named **Example Training a model**.
In this chapter will be described briefly the steps of the training process of a model with images, what language of programming we used and what modules we use in order to process images.
Also we will describe the process of testing if the model obtained are trained correctly by applying it on a correct image and some issue that could appear during the process of testing .

In the third chapter **Live Application Examples**, we will make a program that can process a video stream from a camera or systems of cameras pass that stream as a separate frames of images to the trained model and after that to show in real time if persons from the input video wear the mask or not. In order to be more objective we will use different models not just the model trained locally and explained at the chapter 2 unfortunately due to the lack of hardware resources these models was per-trained.

Chapter 1. Fundamentals of Deep Learning

1.1 What is Deep Learning

Deep Learning is a sub-category of machine learning. It was inspired by artificial neural network commonly known as ANN and which in turn was inspired by biological neural network.

The idea behind a deep learning algorithm is to simulate the human brain structure. As we know inside the human brain there are 100 billion neurons, each neuron connects to about 100.000 other neurons near it. Alone neurons are useless but when there are a lot of them connected one to each other some magic happens and as a result we are able to process information, learn, think, fell, etc.

If we will try to simplify the process the process we can divide neurons linking in layers. Each layer have an input, at his level he process that input and generate an output that will be used as input for the next layer. This process is continued successive until a result is obtained according to the problem.

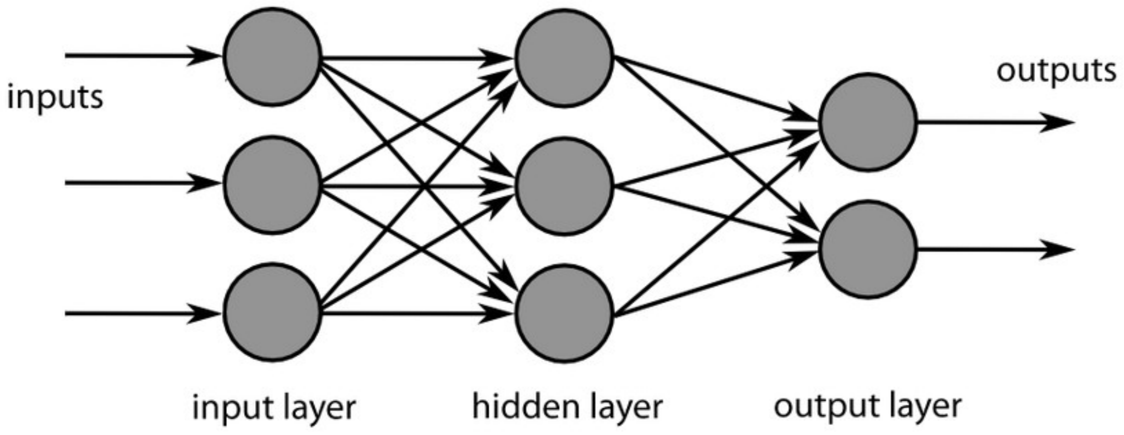


Figure 1.1 Example of layer linking

In the image above we can see an example of a neural network that will always have the following layers:

1. Input layer: It can be pixels of an image or a time series data

2. Hidden layer: Commonly known as weights which are learned while the neural network is trained

3. Output layer: Final layer mainly gives you a prediction of the input you fed into your network.

Also here are an image we we can see the difference of performance between the deep learning and traditional learning algorithms based on the amount of data.

As we can see when the amount of data increase the deep learning methodologies show a better performances compared with the traditional machine learning algorithms where the performance reach a certain saturation level.

In conclusion due to the fact that data increase every day in order to be able to process it deep learning algorithm become more popular and more diverse.

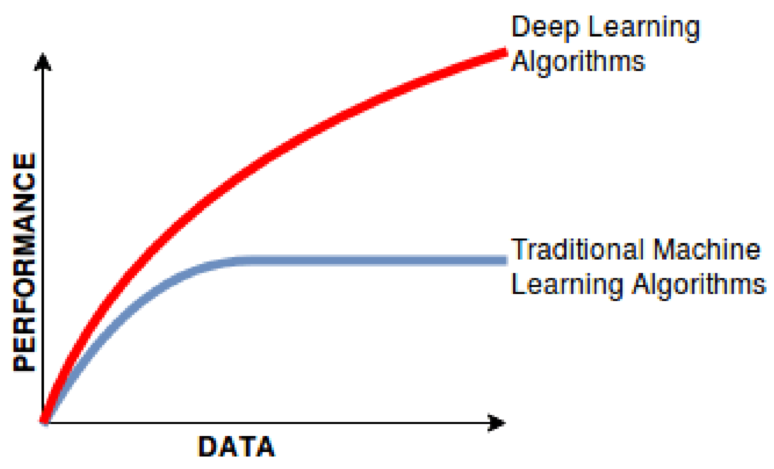


Figure 1.2 Relation between amount of data and performance

1.2 How computer learn automatically

In order to continue firstly we need to explain briefly how computer can learn something automatically.

The answer is DATA. You give (feed) data having different attributes or features that algorithm will understand and as result it will give you a decision boundary based on provided data. When the algorithm trained itself to interpret the data then you can use it wherever you want. You give the input data point and expect for results but don't forget that there results is based on statistical processing and at some moments of time the result can be wrong just like in case of human processing some data.

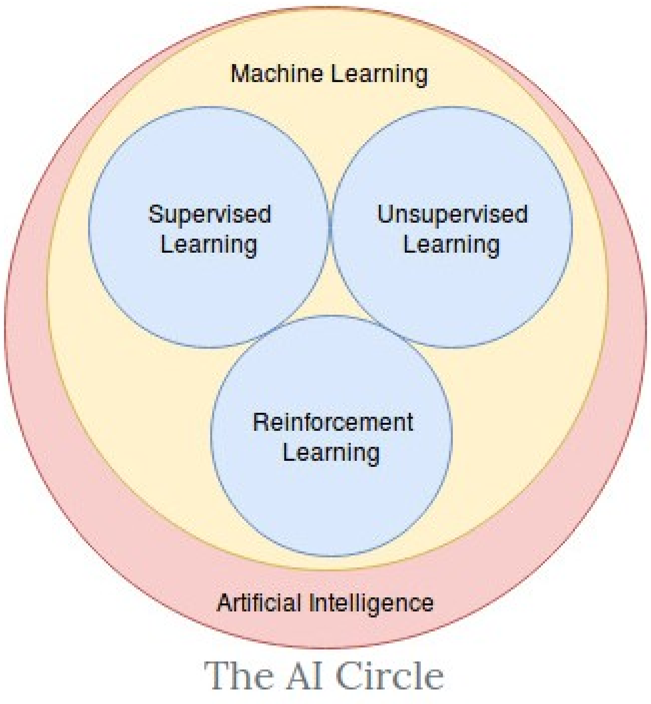


Figure 1.2 Various algorithms for learning

1.2.1 Supervised Learning:

This is the learning technique where the whole process is under control.
The purpose on this algorithm is to find patterns in the train data and classifying it.
As a input data-set include inputs and correct outputs, the algorithm measure the accuracy through loss function and adjusting until the error is minimized as much as possible. Mainly this type of learning can be divided in tow types Classification and Regression.

Regression is used in order to understand relationship between dependent and independent variables, as a consequence it is used in project for sales market.

Classification is used in order to classify assign data into specific categories and as consequence it try to recognize specific patterns and give some conclusions according to which category correspond the input data based to the data set used for training.

Let take a example that we have 10.000 images (5.000 cats and 5.000 dogs) the algorithm will those images in 2 categories dogs with label 1 and cats with label 0 and when the new image come the algorithm will classify it according to those 2 categories.

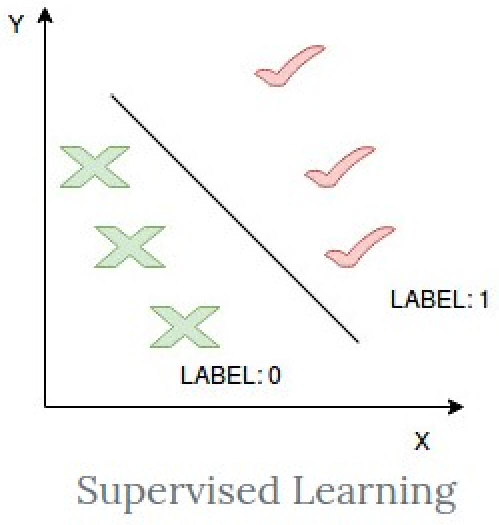


Figure 1.3 Supervised Learning

Challenges of this type of learning are:
Learning models can require a certain levels of expertise about it’s structure accuracy

Learning process can take a lot and data set used for training can contain errors provided by humans

Unlike unsupervised learning it can’t classify data by it’s own

1.2.2 Unsupervised Learning:

In this case we don’t have training labels the algorithm can find suitable structures and patterns in train data. When patterns become apparent that similar data point can be grouped in clusters tougher. The main advantage is that grouping data don’t require intervention of the human.

Mostly this algorithm is used to make high- dimensional data in low- dimension data for visualization, cross-selling strategies, cluster segmentation and image recognition.

Clustering is a data mining technique where data are grouped based on their similarities and differences.

Clustering algorithms can be classified in few types:

Exclusive clustering stipulates that the data point can exist in only one cluster. One of the most representative example is K-means clustering where data points are assigned into K groups, where K represents the number of clusters based on distances from each group centroid. Bigger K means smaller grouping and less granularity

Overlapping clustering allows data point to belong to multiple clusters with separate degrees of membership.

Below we have an graphical representation for better understanding how data are grouped.

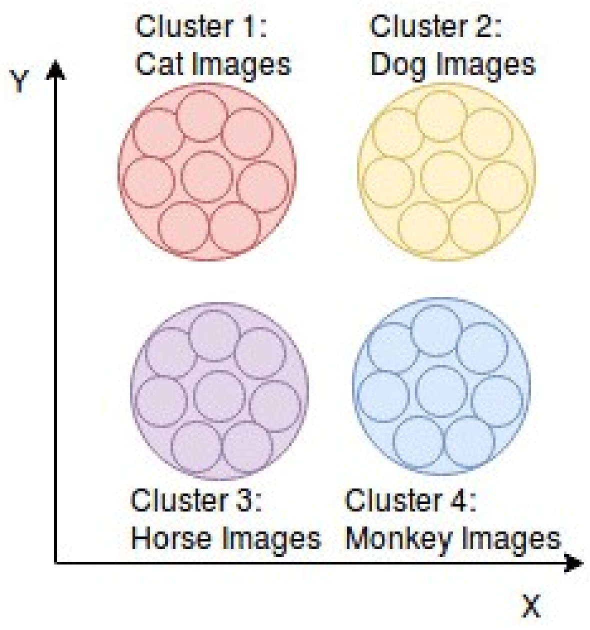


Figure 1.3 Exclusive and Overlapping clustering

Hierarchical clustering also known as HCA and cam be categories in 2 types agglomerative considered as “bottoms-up approach ” or diverse considered “top-down” approach.
In case of agglomerative approach data point are isolated as separate groupings initially and after they are merged toughener iteratively based on their similarity until one cluster is achieved.

In case of diverse clustering a single data is divided based on differences between data points, mainly it is opposite with agglomerative clustering also this type of clustering is not commonly used.

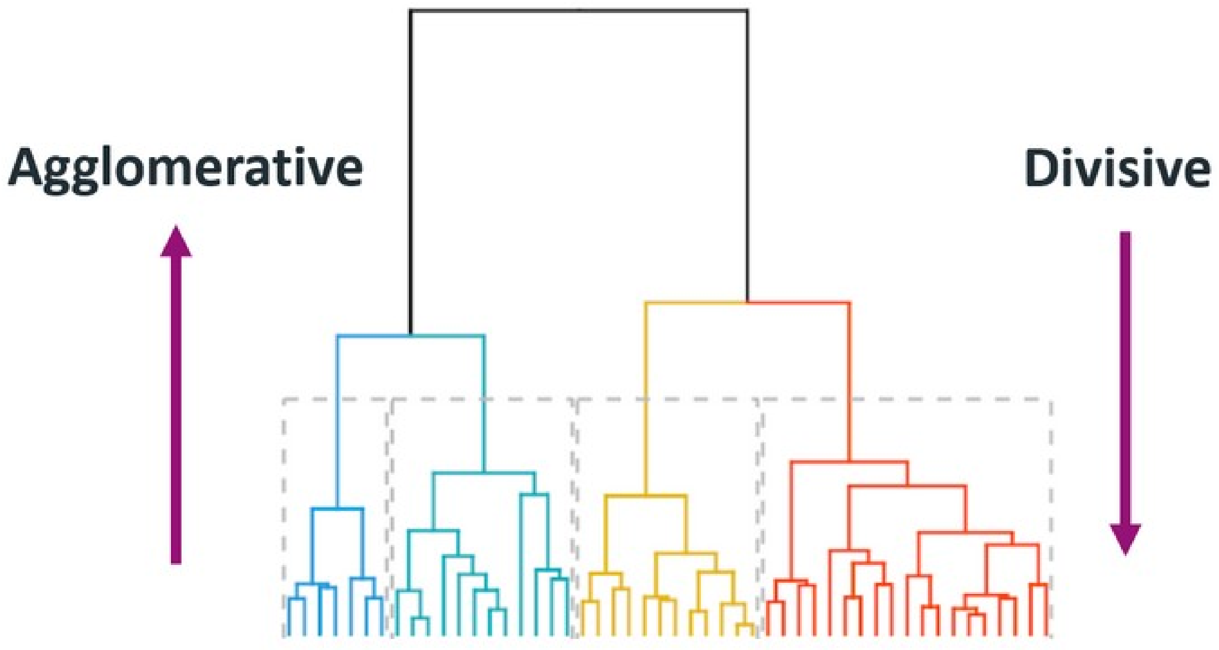


Figure 1.4 Agglomerative and diverse clustering

Probabilistic clustering is an unsupervised technique used to solve density estimation clustering problems. Here data point are grouped based on probability that they belong to a particular distribution model. One of the most used model is Gaussian Mixture Models (GMM) where if we know the mean or variance that we can which cluster a given data point belongs to.

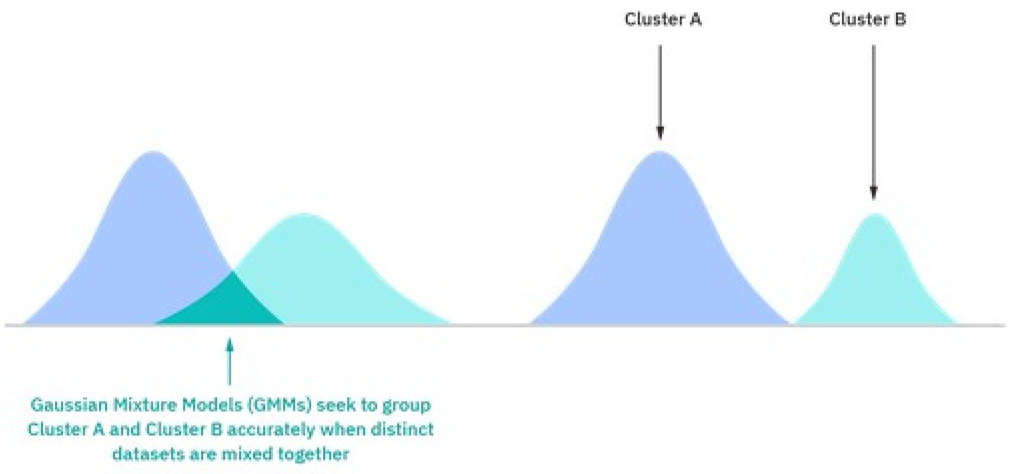

Cluster A
Cluster B
Gaussian Mixture Models (GMMs) seek to group Cluster A and Cluster B accurately when distinct datasets are mixed together

Figure 1.5 Gaussian Mixture Models (GMM)

There are also another methods used in this type of learning such *Apriori algorithms w*here algorithm use hash tree to count items and navigate through them, *Dimensionality reduction u*sed mainly to reduce the number of data inputs if it is too high, *Principal component analysis a*lso used to reduce redundancies and compress data sets, *Singular value decomposition* used when we work with low-rank matrices and *Autoencoders.*

1.2.3 Reinforcement Learning:

These type of learning has an agent who behave like a robot. It learns how to make decision by taking actions and quantifying the results. The learning process can be spited cycles, during that cycle a millions of agents are make a decision according to some problem if the answer is correct the the agent move to the next cycle if not it is destroyed. At the end of thousands of cycles remain only one agent whose which trained in the best way.

One of the challenge of reinforcement learning consist in preparing the environment which is dependent to the task that is going to be performed. Even if the training environment is made and the model is trained transferring the model into the real world is where usually appear problems and the only solution to make this transfer possible is to make the training environment similar with real world environment as much as possible.

Another challenge is scaling because there a no another way to communicate other than through system on rewards and penalties (Interpreter). That constrain may lead to the situation when acquiring the new knowledge cause erasing the old one.

There are also a problem with optimization, we can arrive with an agent that perform the task but in to slow for example. In that case there a no another way then to train another agent.

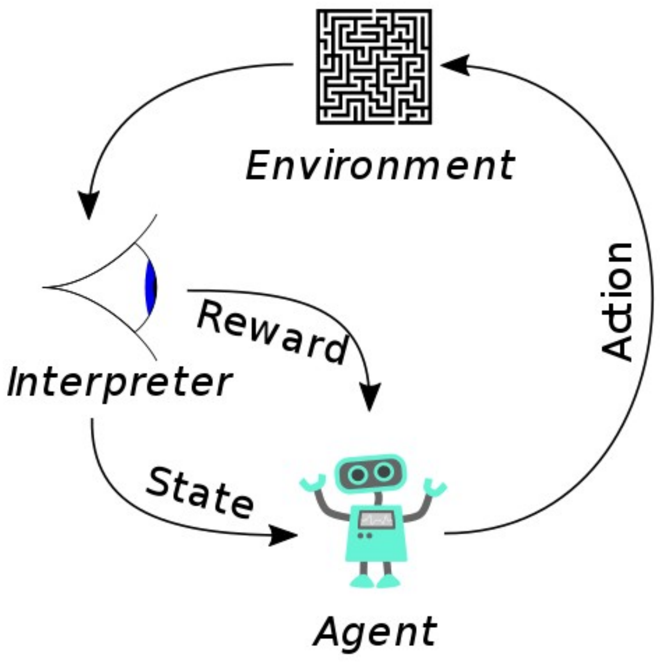


Figure 1.4 Reinforcement Learning

1.3 Inside the neural network

The main advantage of neural networks is that they can learn itself, all what we need to do is to define the architecture, tell the input data and what we expect as the result. One way to understand how it works is to think of each individual node as its own linear regression model where formula looks like:

$$\sum_{i=1}^{m} w_i x_i + bias = w_1 x_1 + w_2 x_2 + w_3 x_3 + bias$$

∑wixi + bias = w1x1 + w2x2 + w3x3 + bias

$$output = f(x) = \begin{cases} 1 \text{ if } \sum w_1 x_1 + b \geq 0 \\ 0 \text{ if } \sum w_1 x_1 + b < 0 \end{cases}$$

output = f(x) = 1 if ∑w1x1 + b>= 0; 0 if ∑w1x1 + b < 0

Figure 1.5

The neural networks can be classified into different types based of the propose but due to the fact that nowadays neural networks are used in a lot of cases it leads to a very comprehensive list of types this is why we will mention juts few of them mainly the most common one.

Feed forward neural networks, or multi-layer perceptions (MLPs) composed on an input, hidden and output layer.

Convolutional neural networks (CNNs) similar to feed forward networks, but they’re usually utilized for image recognition, pattern recognition, and/or computer vision and are based on linear algebra and matrix manipulation in order to determine patterns in an image.

Recurrent neural networks (RNNs) that can be recognized by their feedback loops. Mainly used in order to prediction about future outcomes in areas such stock market.

Feed-forward network is a network that contains inputs, outputs, and hidden layers. The signals can only travel in one direction (forward). Input data passes into a layer where calculations are performed. Each processing element computes based upon the weighted sum of its inputs. The new values become the new input values that feed the next layer (feed-forward). This continues through all the layers and determines the output. Feed-forward networks are often used in, for example, data mining.

Feedback network (for example, a recurrent neural network) has feedback paths this means that they can have signals traveling in both directions using loops and all possible connections between neurons are allowed.
Since loops are present in this type of network, it becomes a non-linear dynamic system which changes continuously until it reaches a state of equilibrium.
Feedback networks are often used in optimization problems where the network looks for the best arrangement of interconnected factors.

As it was mentioned many times in order to do that at the stages of calculations and defining the architecture we use the following concepts at the level of a neuron:

1.3.1 Weighted sum

Each connection between two neurons has a unique synapse with a unique weight attached. If you want to get from one neuron to the next, you have to travel along the synapse and pay the “toll” (weight). The neuron then applies an activation function to the sum of the weighted inputs from each incoming synapse. It passes the result on to all the neurons in the next layer. When we talk about updating weights in a network, we’re talking about adjusting the weights on these synapses.

A neuron’s input is the sum of weighted outputs from all the neurons in the previous layer. Each input is multiplied by the weight associated with the synapse connecting the input to the current neuron. If there are 3 inputs or neurons in the previous layer, each neuron in the current layer will have 3 distinct weights: one for each synapse.

In a nutshell, the activation function of a node defines the output of that node. The activation function (or transfer function) translates the input signals to output signals. It maps the output values on a range like 0 to 1 or -1 to 1. It’s an abstraction that represents the rate of action potential firing in the cell. It’s a number that represents the likelihood that the cell will fire. At it’s simplest, the function is binary: **yes** (the neuron fires) or **no** (the neuron doesn’t fire). The output can be either 0 or 1 (on/off or yes/no), or it can be anywhere in a range.

1.3.2 Activation Function

The activation function (or transfer function) translates the input signals to output signals. It maps the output values on a range like 0 to 1 or -1 to 1. It’s an abstraction that represents the rate of action potential firing in the cell. It’s a number that represents the likelihood that the cell will fire. At it’s simplest, the function is binary: **yes** (the neuron fires) or **no** (the neuron doesn’t fire). The output can be either 0 or 1 (on/off or yes/no), or it can be anywhere in a range.

$$\text{Y} = \text{Activation}(\Sigma(weight * input) + bias)$$

Figure 1.6

There can be a lot of type for the activation function and in order to chose the right one we must to take into account the proprieties of the problem. Because there are a lot of functions and a lot of implementations we will describe just few of them.

1.3.2.1 Binary Step Function

This is the most basic type of function that can be easily implemented by using an if else statement. As the mane says if the input of the activation function is greater then a certain threshold then the neuron is activated else is deactivated. In the image below a graphical representation of the output.

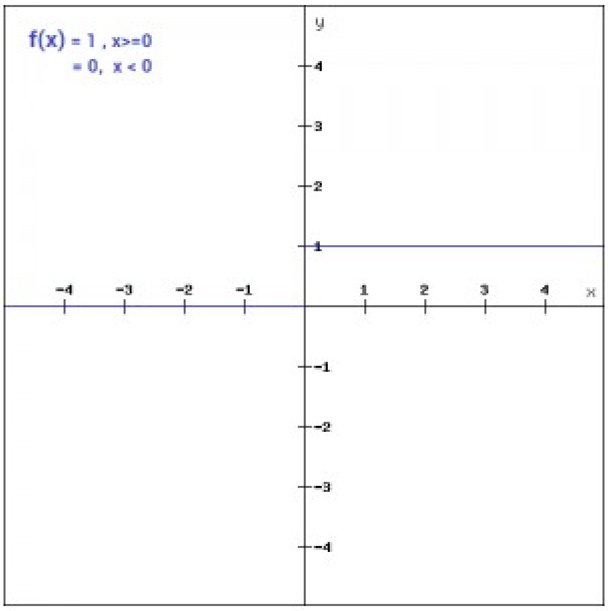


Figure 1.7

This type of function can be used in case of binary classifier but it become useless when there are a multiple classes on the input variables, these aspect make binary step function very limited in usage moreover there are cases when the result is 0 and it can cause the back propagation process.

1.3.2.2 Sigmoid function

This type of function in one of the most widely used non-linear activation function. The mathematical expression of that function is **f(x) = 1/(1+e^-x)**. Being a non-linear function as a consequence it leads to a non- linear output from the neuron system these means practically that the output we get from a neuron can not be represented in a linear combination.

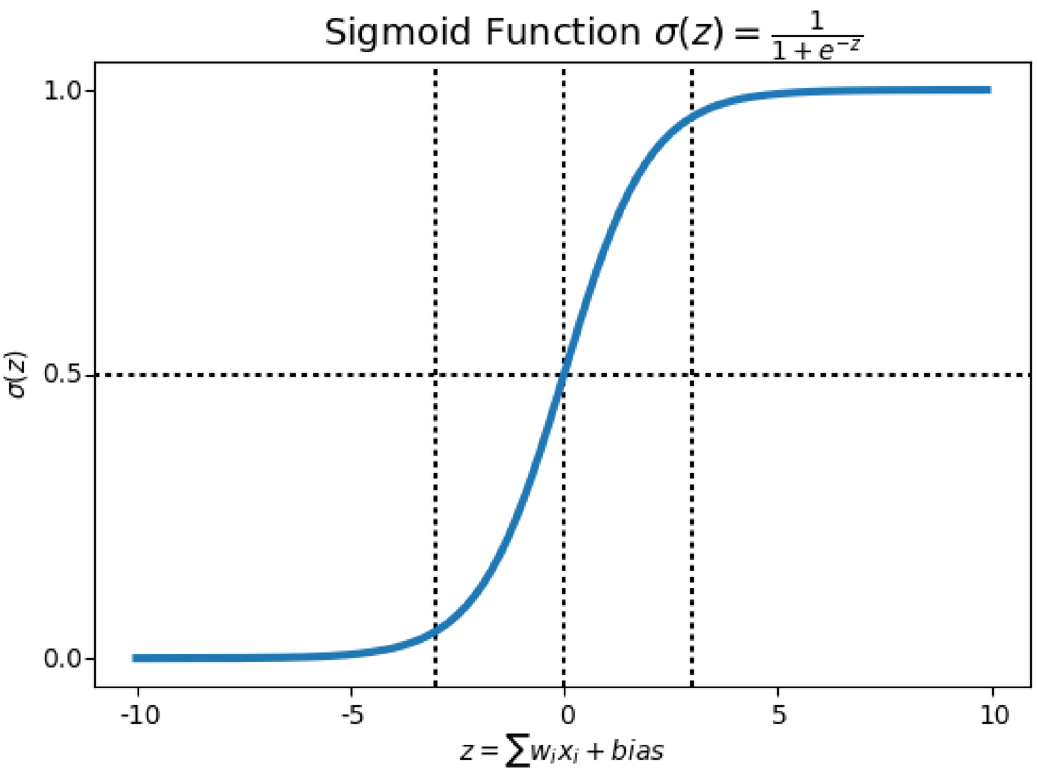


Figure 1.8

Unfortunately the sigmoid function is not commonly used due to tow disadvantages. Firstly sigmoids can be easily saturated if the input is too large or small this make the gradient of the neuron close to 0. As consequence we can’t learn the input data recursively. Second disadvantage is that the output is not 0 centered, it will cause the input of the neurons in the back layer to be a signal with a non-zero mean, which will affect the gradient.

1.3.2.3 Than function

This type of function is very similar to sigmoid function the only difference is that it is symmetric respect to the origin. The range of the values are between -1 and 1 and as consequence the input for the next layer can be with different signs. The formula if the than function is **tanh(x)=2sigmoid(2x)-1** and as in the case of sigmoid function there are also the same problem with the gradient.

$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

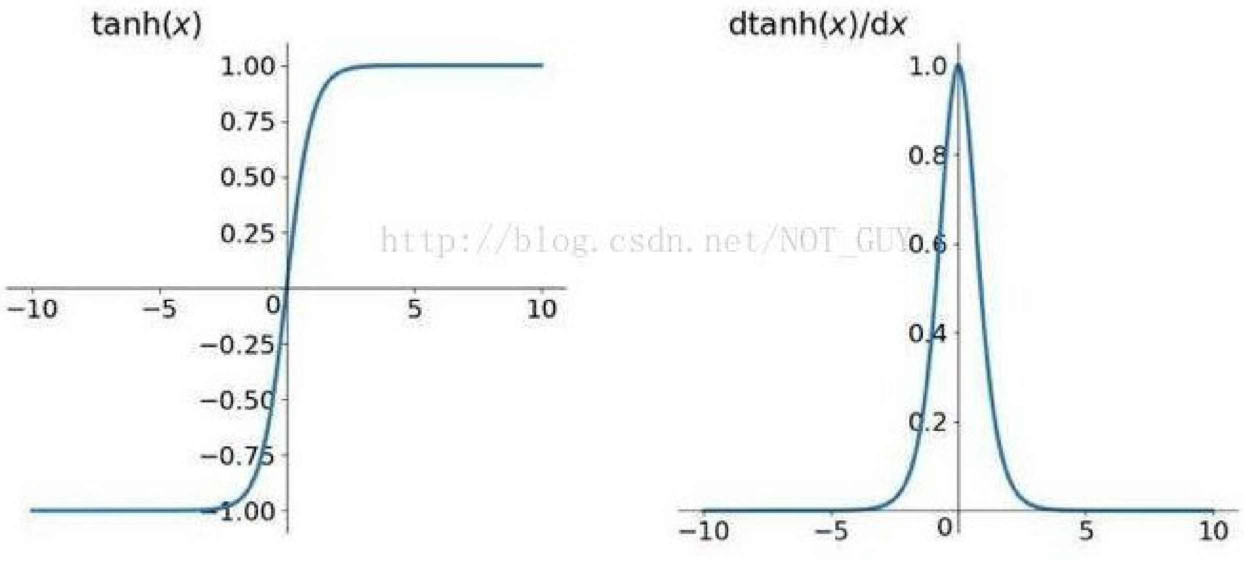


Figure 1.9

1.3.2.4 ReLU functions

ReLU stands for Rectified Linear Unit. The main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time. This means that the neurons will only be deactivated if the output of the linear transformation is less than 0.

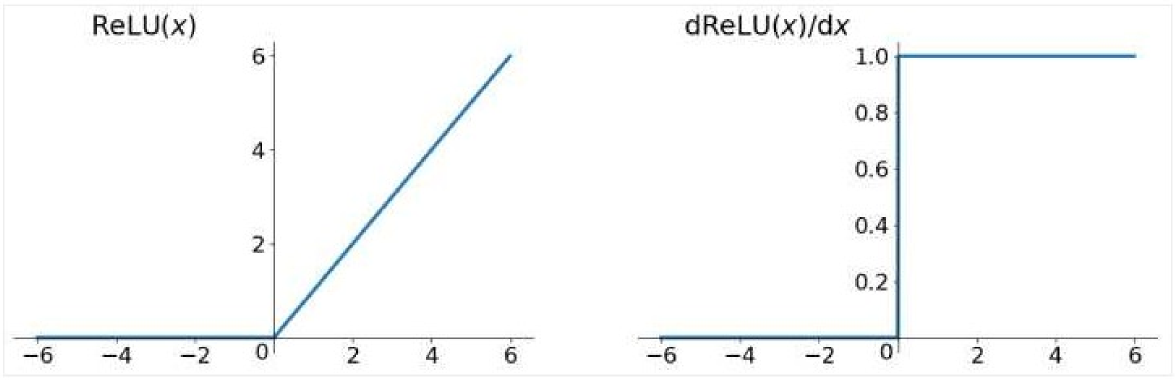


Figure 1.10

Compared to the sigmoid / tanh functions the ReLU have 2 advantages. First practically it was found that the coverage speed is much faster and the second advantage is that the calculation are easier only needs a threshold to get the activation value. x> 0, f (x) = x as consequence speed up the calculation of forward propagation. Regarding the disadvantages firs one is that the output is not zero centered and second problem is that ReLU is very fragile during training known as "Dead ReLU problem".

We can also mention functions like **Leaky ReLU function** and **ELU (Exponential Linear Units) t**hese type of functions came to solve the Dead ReLU problem based on that we can say that these tow functions are just variation of the ReLU function.

Chapter 2. Training a model

In this chapter we will describe step by stet how we train a model in order to detect face and if a person wear the mas or not, also we will mention what programming language in was used.

The main objective is to train a model using Deep Learning which as an input has an image and based on it can identify if the person is wearing the mask or not.

2.1 Python (programming language)

It is an interpreted high-level general-purpose programming language, it is used in very often in Deep Learning due to it’s simplicity and integrated modules that allow developers to concentrate on more on l training methodologies then on how to implement again some low level functions from neural networks.

Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly,procedural), object-oriented and functional programming. Python is often described as a **"batteries included"** language due to its comprehensive standard library.

Python was conceived in the late 1980s by Guido van Rossum from Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor but the first release was in 1991 as Python 0.9.0 nowadays it reach the version 3.9.6 in 2021.

There are several reason why Python in used as the main language for AI, Machine Learning and Deep Learning, firstly it is open source and has a lot of packages useful in training models for example like Keras, secondly it doesn't require a deep knowledge about the programming because it has an easy syntax, last but not least the code written in Python can run almost on any Operating system (Windows, Linux, Ios) all what we need to do is to install the corresponding packages for python.

As an development environment we used PyCharm which is an Python IDE provided by Jetbrains.

2.2 Keras module

It is an open source library that provides an interface for a deep learning models training. It was developed as part of the research effort of project Open-ended Neuro-Electronic Intelligent Robot Operating System where the primary author was François Chollet.

The main advantages of that model are:

1. Modularity. A model is understand as a sequence of fully configurable modules this leads to the possibility when neural layers, cost functions, optimizes, initialization schemes, activation functions, regularization schemes can be interpreted as individual modules and can be combined in many was in order to create new models.

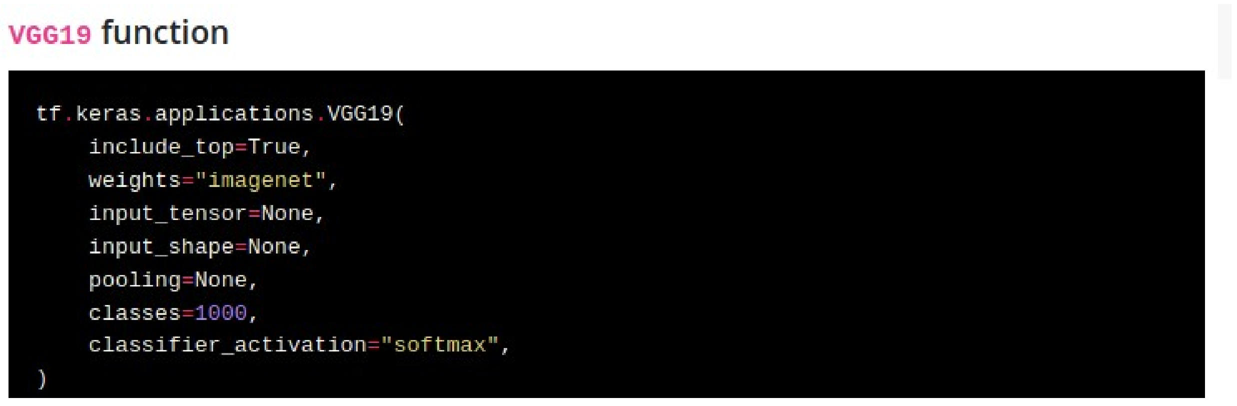
2.Minimalism. It consist in that the code is easy to use and are very intuitive even for people how don’t know programming.

3.Easy extensible. Adding new modules, classes or function are very easy and as consequence Keras is suitable for researching.

4.Can work with it in Python. As it was mentioned before Python is a very easy programming language which make also easy to work with Keras.

5. Supports arbitrary connectivity schemes (including multi-input and multi-output training).

Being a very big modules it is obvious that it supports rot of models types but in our training process we used VGG19 architecture. In the image below we can see the declaration of the VGG19 function. As an important aspect to be mention is that in order to use this type of architecture the size of the image should be 224x224.


VGG19 function

```
tf.keras.applications.VGG19(
    include_top=True,
    weights="imagenet",
    input_tensor=None,
    input_shape=None,
    pooling=None,
    classes=1000,
    classifier_activation="softmax",
)
```

Figure 2.1

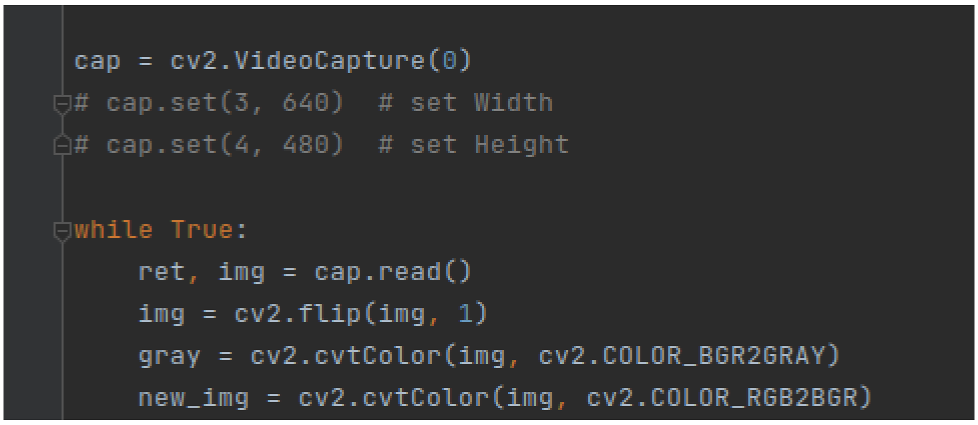
2.3 OpenCV

Figure 2.2

It is a open source library that include hundreds of computer vision algorithms, has a modular structure and have 2 main implementation OpenCV 2.x API, which is essentially a C++ API opposed to the C-based OpenCV 1.x API.

Some of mos popular modules available are:
core - a compact module defining basic data structures,
imgproc - an image processing module that includes linear and non-linear image filtering, geometrical image transformations,
video - a video analysis module that includes motion estimation,
calib3d - basic multiple-view geometry algorithms
objdetect - detection of objects and instances of the predefined classes
….

There are more modules implemented but in our project we used OpenCV for reading images from the video camera and pass them to the models trained in order to detect people faces and if they are wearing the mask or not. Below there are an image where we can see how easily is to capture image by using OpenCV.



```
cap = cv2.VideoCapture(0)
# cap.set(3, 640)  # set Width
# cap.set(4, 480)  # set Height

while True:
    ret, img = cap.read()
    img = cv2.flip(img, 1)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    new_img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
```

Figure 2.3

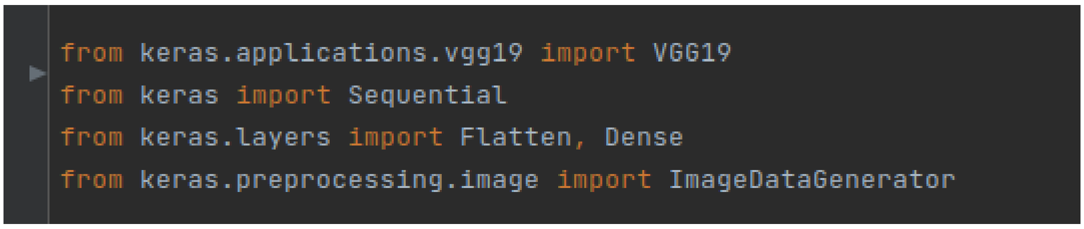
2.4 Training model Steps and resources used

One of the biggest challenge when we want to train a model is to find enough data for training. In the first half of the project we will train a model in order to recognize if persons from an image wear medical mask correctly or not.

As a train data set we used images where persons wear mask or not. Training set of data is composed from a thousands of images grouped in three categories for training , validation and testing the model. Also each category is divided in 2 subcategories one with correct images where people wear mask correct and another where people don't wear mask at all.

2.4.1 Training model

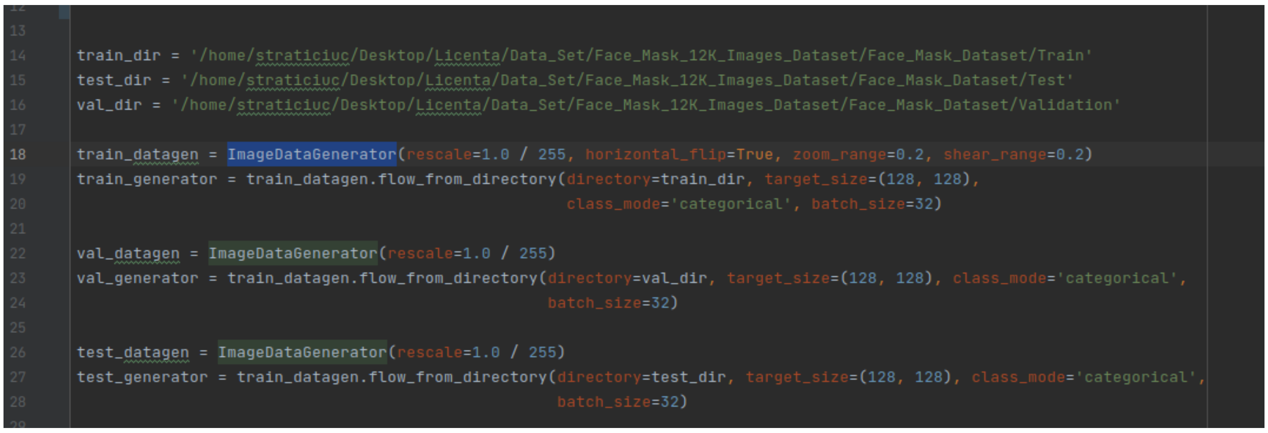
As we said previously for training model we used VGG19 architecture from Keras and as consequence firstly we import all dependence that we need



```
from keras.applications.vgg19 import VGG19
from keras import Sequential
from keras.layers import Flatten, Dense
from keras.preprocessing.image import ImageDataGenerator
```

Figure 2.4

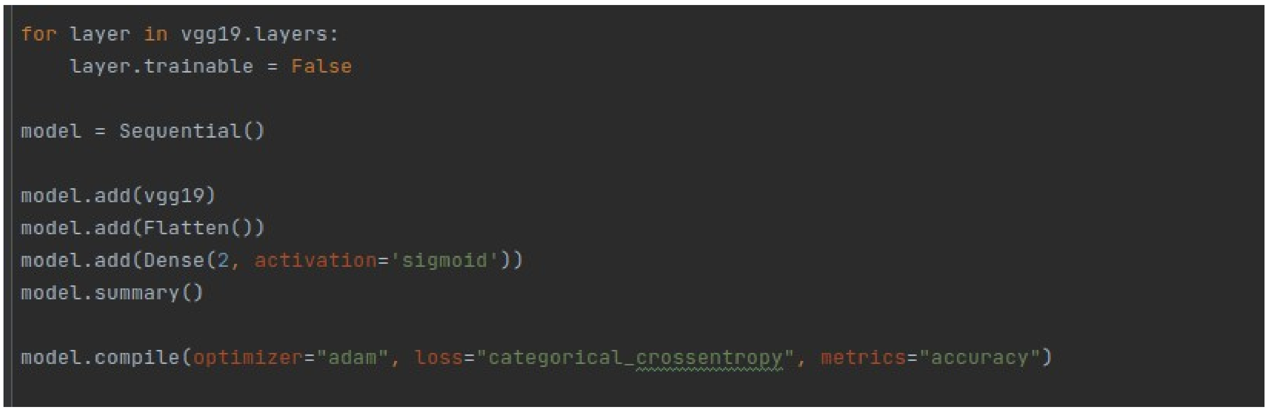
After that we need to define the paths where we can find the data-set for training , validation and testing as a a string value. Using function **ImageDataGenerator** we generate batches of tensor image data with real-time data augmentation and using **flow_from_directory** method which takes the path to a directory & generates batches of augmented data we create internally the sets of data needed for model creation and validation.



```
train_dir = '/home/straticiuc/Desktop/Licenta/Data_Set/Face_Mask_12K_Images_Dataset/Face_Mask_Dataset/Train'
test_dir = '/home/straticiuc/Desktop/Licenta/Data_Set/Face_Mask_12K_Images_Dataset/Face_Mask_Dataset/Test'
val_dir = '/home/straticiuc/Desktop/Licenta/Data_Set/Face_Mask_12K_Images_Dataset/Face_Mask_Dataset/Validation'

train_datagen = ImageDataGenerator(rescale=1.0 / 255, horizontal_flip=True, zoom_range=0.2, shear_range=0.2)
train_generator = train_datagen.flow_from_directory(directory=train_dir, target_size=(128, 128),
                                                    class_mode='categorical', batch_size=32)

val_datagen = ImageDataGenerator(rescale=1.0 / 255)
val_generator = train_datagen.flow_from_directory(directory=val_dir, target_size=(128, 128), class_mode='categorical',
                                                  batch_size=32)

test_datagen = ImageDataGenerator(rescale=1.0 / 255)
test_generator = train_datagen.flow_from_directory(directory=test_dir, target_size=(128, 128), class_mode='categorical',
                                                   batch_size=32)
```

Figure 2.5

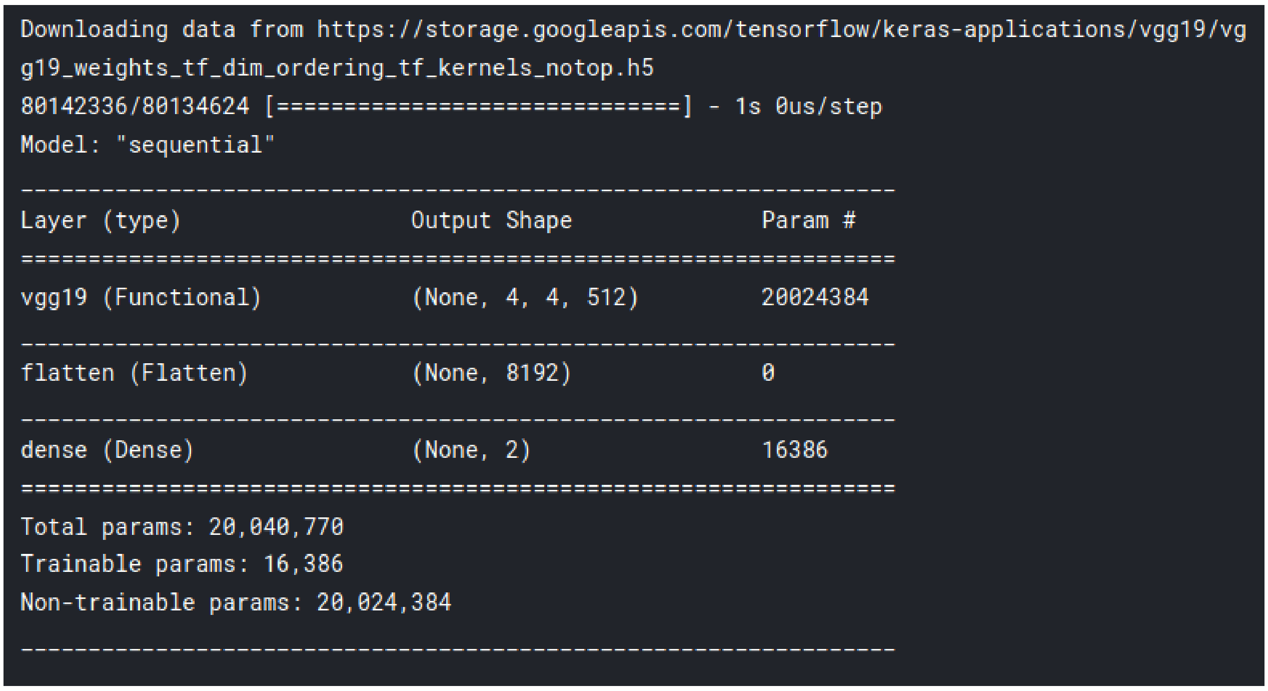
The next step is to build VGG19 transfer learning model. Below we have he sequence of code that was used in order to do that.



```
for layer in vgg19.layers:
    layer.trainable = False

model = Sequential()

model.add(vgg19)
model.add(Flatten())
model.add(Dense(2, activation='sigmoid'))
model.summary()

model.compile(optimizer="adam", loss="categorical_crossentropy", metrics="accuracy")
```

Figure 2.6

As we can see in the image for this model we have chose the sequential model instance where each layer has exactly one input tensor and one output tensor. Also we add the fatten layer in order to reshapes the tensor to have the shape that is equal to the number of elements contained in tensor. At the end we print the summary of the learning model.

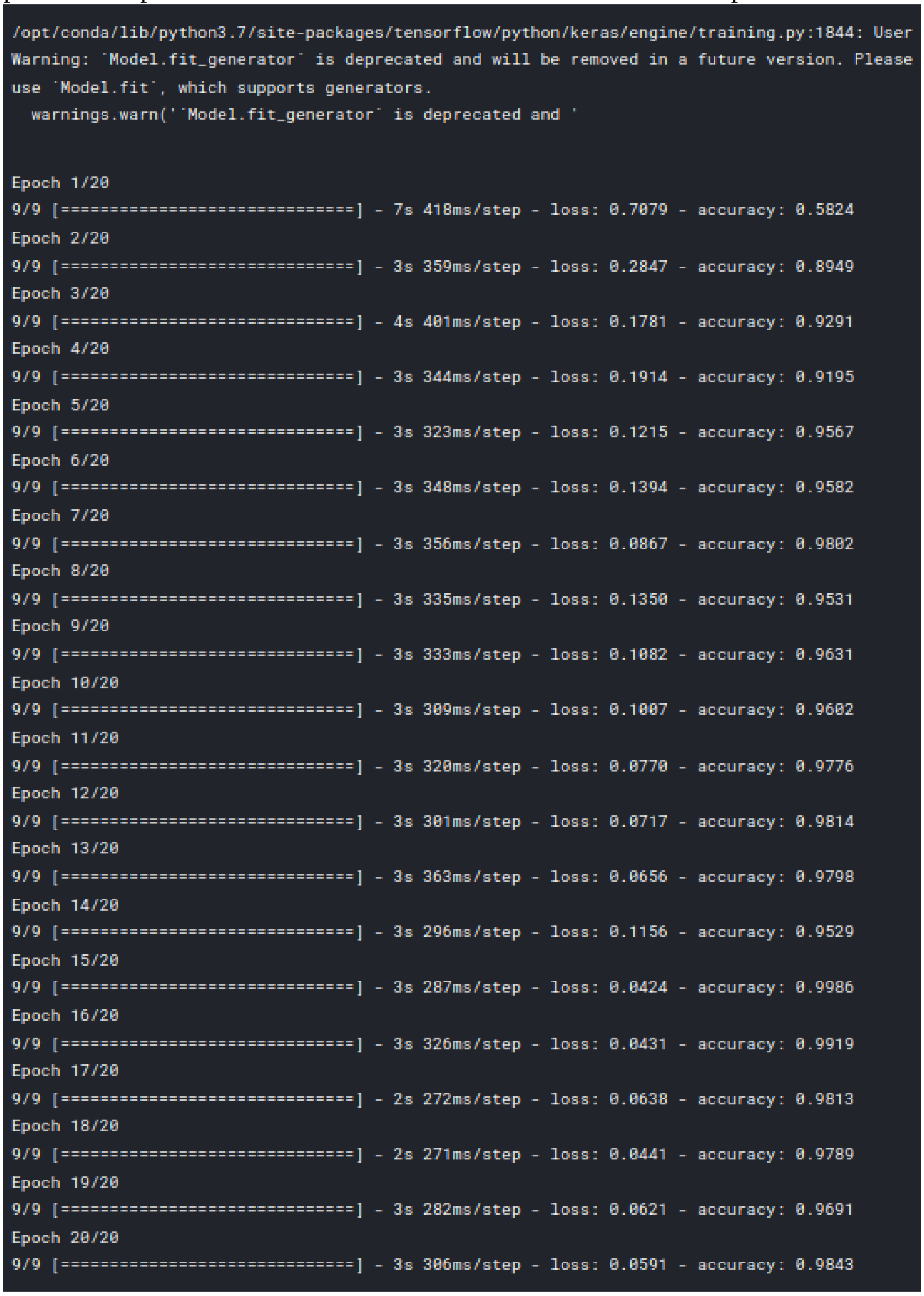


```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vg
g19_weights_tf_dim_ordering_tf_kernels_notop.h5
80142336/80134624 [==============================] - 1s 0us/step
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg19 (Functional)           (None, 4, 4, 512)         20024384
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense (Dense)                (None, 2)                 16386
=================================================================
Total params: 20,040,770
Trainable params: 16,386
Non-trainable params: 20,024,384
_________________________________________________________________
```

Figure 2.7

At the end we make a call of the function fit_generator() where we specify generators created

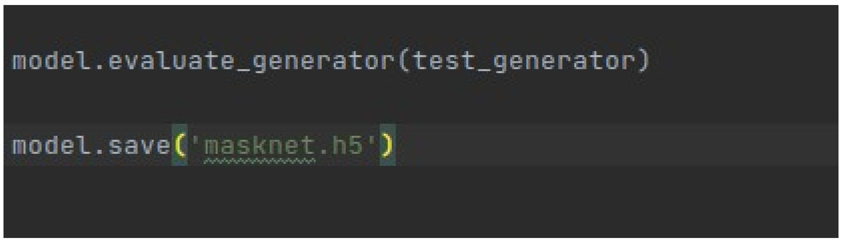
previous that will be used for training and validation of the model.
When we call the function fit_generator the following steps are performed by Keras:
1. Keras calls the generator function supplied to .fit_generator().
2. The generator function yields a batch of size BS to the .fit_generator function.
3. The .fit_generator function accepts the batch of data, performs back-propagation, and updates the weights in our model.
4.This process is repeated until we have reached the desired number of epochs.



```
/opt/conda/lib/python3.7/site-packages/tensorflow/python/keras/engine/training.py:1844: User
Warning: `Model.fit_generator` is deprecated and will be removed in a future version. Please
use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '


Epoch 1/20
9/9 [==============================] - 7s 418ms/step - loss: 0.7079 - accuracy: 0.5824
Epoch 2/20
9/9 [==============================] - 3s 359ms/step - loss: 0.2847 - accuracy: 0.8949
Epoch 3/20
9/9 [==============================] - 4s 401ms/step - loss: 0.1781 - accuracy: 0.9291
Epoch 4/20
9/9 [==============================] - 3s 344ms/step - loss: 0.1914 - accuracy: 0.9195
Epoch 5/20
9/9 [==============================] - 3s 323ms/step - loss: 0.1215 - accuracy: 0.9567
Epoch 6/20
9/9 [==============================] - 3s 348ms/step - loss: 0.1394 - accuracy: 0.9582
Epoch 7/20
9/9 [==============================] - 3s 356ms/step - loss: 0.0867 - accuracy: 0.9802
Epoch 8/20
9/9 [==============================] - 3s 335ms/step - loss: 0.1350 - accuracy: 0.9531
Epoch 9/20
9/9 [==============================] - 3s 333ms/step - loss: 0.1082 - accuracy: 0.9631
Epoch 10/20
9/9 [==============================] - 3s 309ms/step - loss: 0.1007 - accuracy: 0.9602
Epoch 11/20
9/9 [==============================] - 3s 320ms/step - loss: 0.0770 - accuracy: 0.9776
Epoch 12/20
9/9 [==============================] - 3s 301ms/step - loss: 0.0717 - accuracy: 0.9814
Epoch 13/20
9/9 [==============================] - 3s 363ms/step - loss: 0.0656 - accuracy: 0.9798
Epoch 14/20
9/9 [==============================] - 3s 296ms/step - loss: 0.1156 - accuracy: 0.9529
Epoch 15/20
9/9 [==============================] - 3s 287ms/step - loss: 0.0424 - accuracy: 0.9986
Epoch 16/20
9/9 [==============================] - 3s 326ms/step - loss: 0.0431 - accuracy: 0.9919
Epoch 17/20
9/9 [==============================] - 2s 272ms/step - loss: 0.0638 - accuracy: 0.9813
Epoch 18/20
9/9 [==============================] - 2s 271ms/step - loss: 0.0441 - accuracy: 0.9789
Epoch 19/20
9/9 [==============================] - 3s 282ms/step - loss: 0.0621 - accuracy: 0.9691
Epoch 20/20
9/9 [==============================] - 3s 306ms/step - loss: 0.0591 - accuracy: 0.9843
```

Figure 2.8

At the end by using the evaluate_generator we can test the accuracy of the model obtained and then save it. Also based on test_generator composed from test data images we arrived with a model witch accuracy is around 98%.



```
model.evaluate_generator(test_generator)

model.save('masknet.h5')
```

Figure 2.9

At the end of training process we obtain a file with the extension .h5 that can be easy loaded wherever we wand and use it by calling the method **model = load_model('masknet.h5')**.

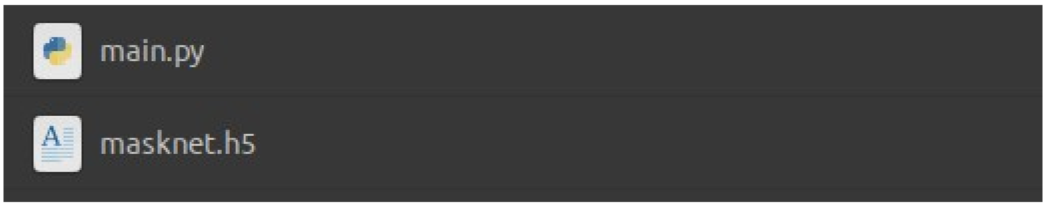
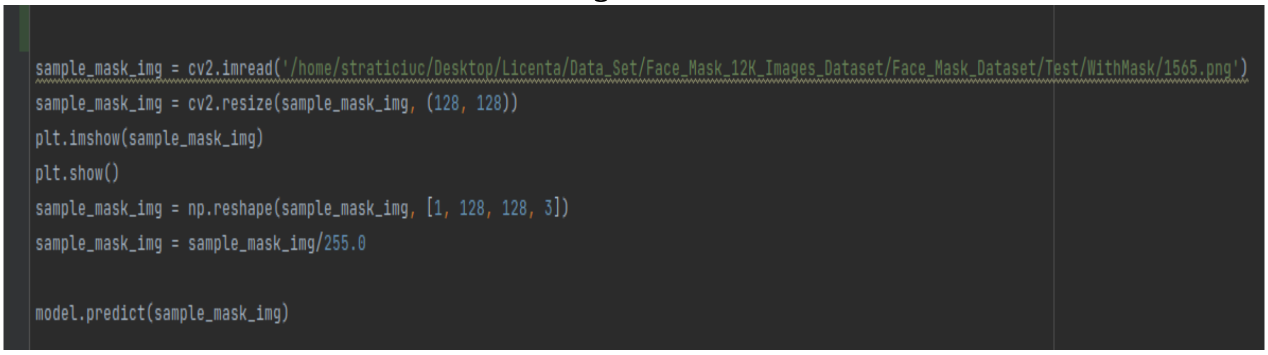


Figure 3.0

2.4.2 Testing model

For testing purpose we will use firstly an image from data set where a girl wear the mas correctly.

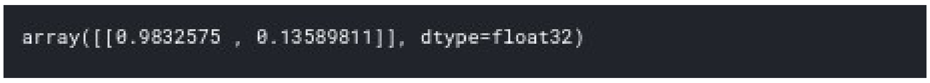
Figure 3.1



```
sample_mask_img = cv2.imread('/home/straticiuc/Desktop/Licenta/Data_Set/Face_Mask_12K_Images_Dataset/Face_Mask_Dataset/Test/WithMask/1565.png')
sample_mask_img = cv2.resize(sample_mask_img, (128, 128))
plt.imshow(sample_mask_img)
plt.show()
sample_mask_img = np.reshape(sample_mask_img, [1, 128, 128, 3])
sample_mask_img = sample_mask_img/255.0

model.predict(sample_mask_img)
```

As we can see in the Figure 3.1 we take a sample image from test folder resize it due to the fact that the model was trained to work with images with shape 128x128.

Using **predict()** method we pass to the model the sample image and evaluate if the person in the image a re wearing the mask or not. As an output of the predict method we will have an array that contain tow probability one if the persona a wearing the mask and another if not.



```
array([[0.9832575 , 0.13589811]], dtype=float32)
```

But in order to be more easy to verify we decide to print the image in order to have a clear visualization if the model identify the mask in image or not .

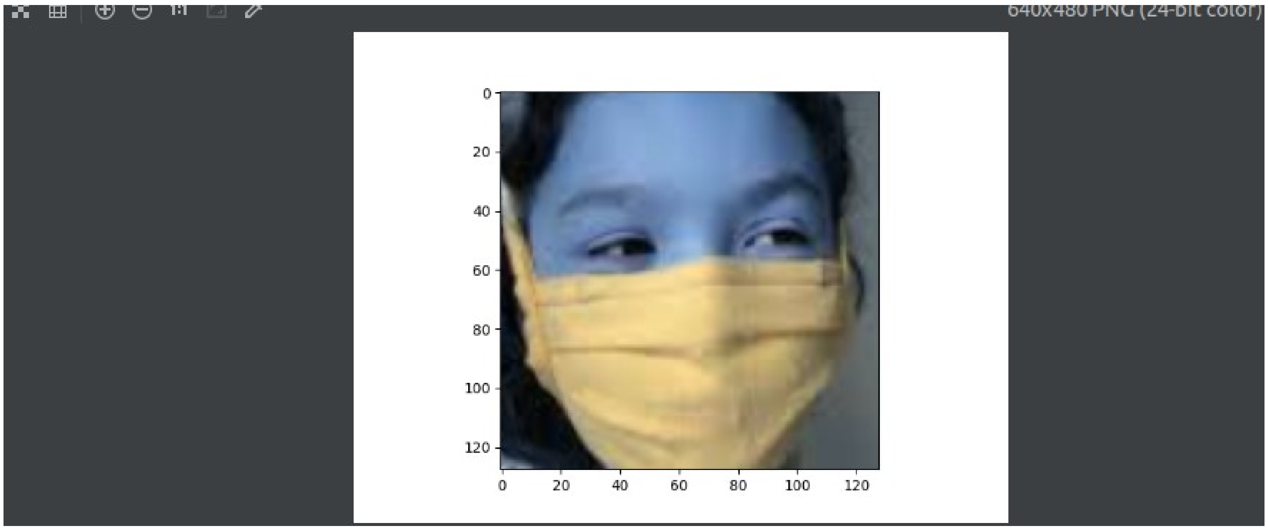


Figure 3.2

As we can see in the figure above the mask pattern was identified correctly and based on that we can arrive to the conclusion that the training process was completed with success. Know the model is ready to be used but we need to remember that in order to obtain a relevant result we always must to reshape the image to the size corresponding with tanning set images. Unfortunately when we do reshape of the image we lose the quality of that image and as we can suspect if the quality is become lower in increase the probability that the model will not be able to detect the mask correctly.

Chapter 3. Live Application Examples

In this chapter we will exemplify several application that use per-trained models in order to detect if a person wear the mask or not from an image or from an video stream.

As it was said previously the main target was to make a application that can be easily integrated in real world environment. During the development we arrived to the conclusion that its not productive to make a single application that can detect faces and if the persons wear the mask correctly on static images or live stream video simultaneously. In the end we arrived with 2 distinct application that can be used separate depending on the needs.

3.1 Test Environment Setup

As an environment setup we use the personal computer and a Raspberry Pi 4. In case of a personal computer it was used Lenovo Legion with integrated camera, processor i5 and a video card GEFORCE GTX. Even if it seems that the computer if powerful practically these resources are not enough to train a complex model, as consequence we was forced to take a pre trained model as a replacement for the model trained by us.
In case of Raspberry Pi 4 we will list a fie images below where we can see from what it consist:

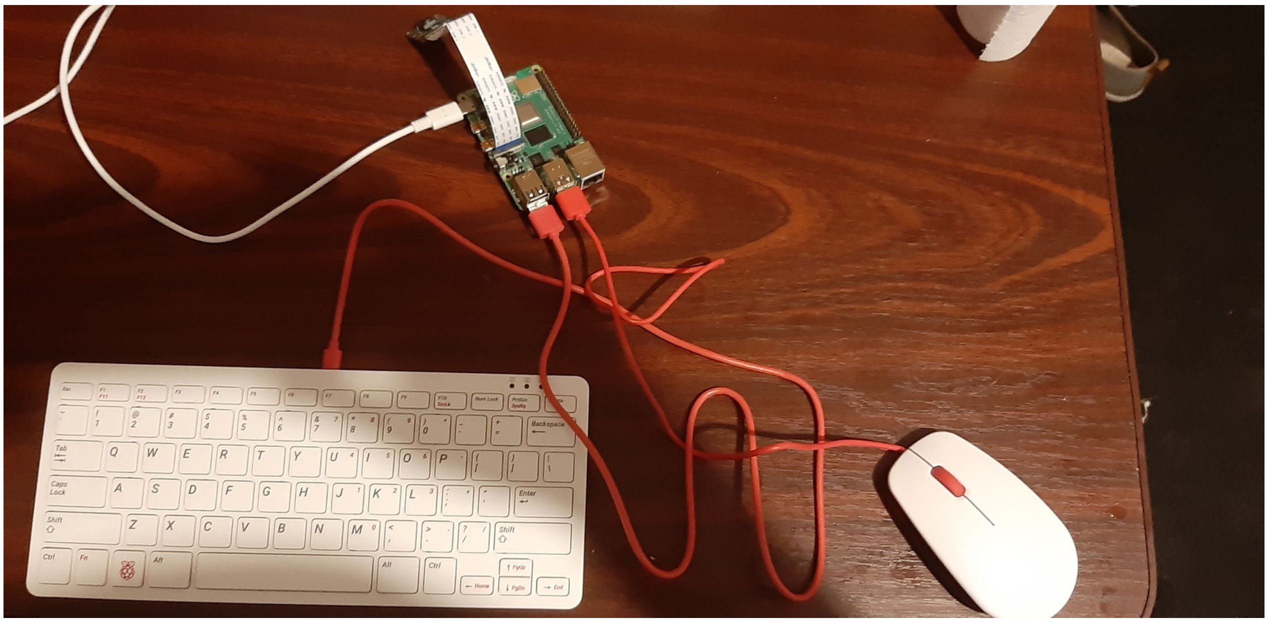


Fig 4.01 Overall Setup

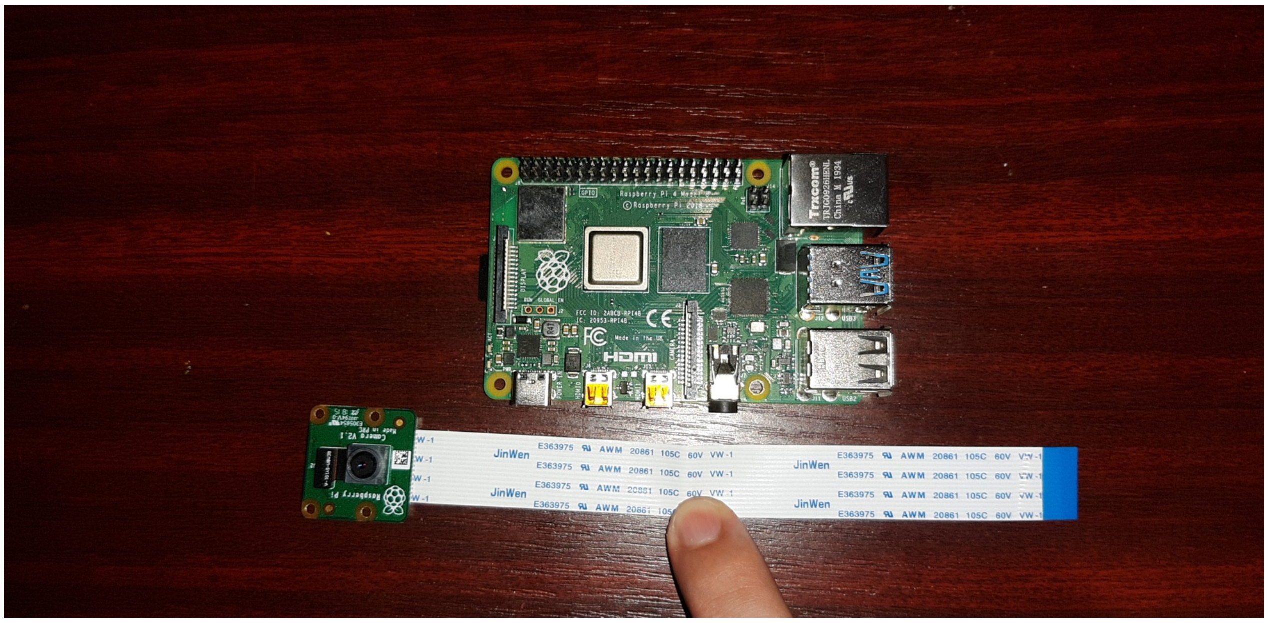


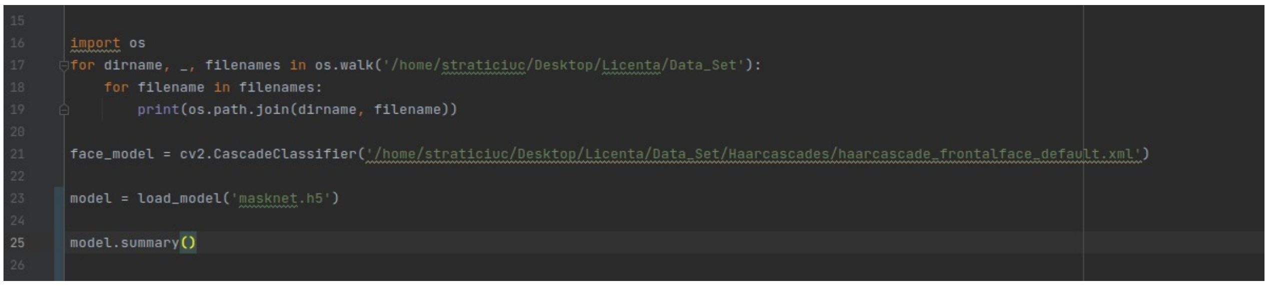
Figure 4.02

As we can see in the figure 4.01 and 4.02 the setup consist from a video camera, raspberry py development board, mouse and keyboard. The main disadvantages of that setup was: low quality camera and a lack of hardware resources, those disadvantages lead to a very bad detection even detection even running the program is not stable.

3.2 Detecting Faces and Medical Mask on static images

Let’s suppose that we are forced to analyze the video registration that was made 2 weeks ago and find there are persons that don’t wear mask. In order to do that we can discompose in frames the video recording, in fact we obtain a number of images that correspond to different time stamp in the video. Having that cluster of images we can chose one or all images and pass it to the program to be evaluated.

First of all we load model trained before, this can be easily done by using load_model method. But before loading a model we need to specify the director name where we saved the model and program will search.

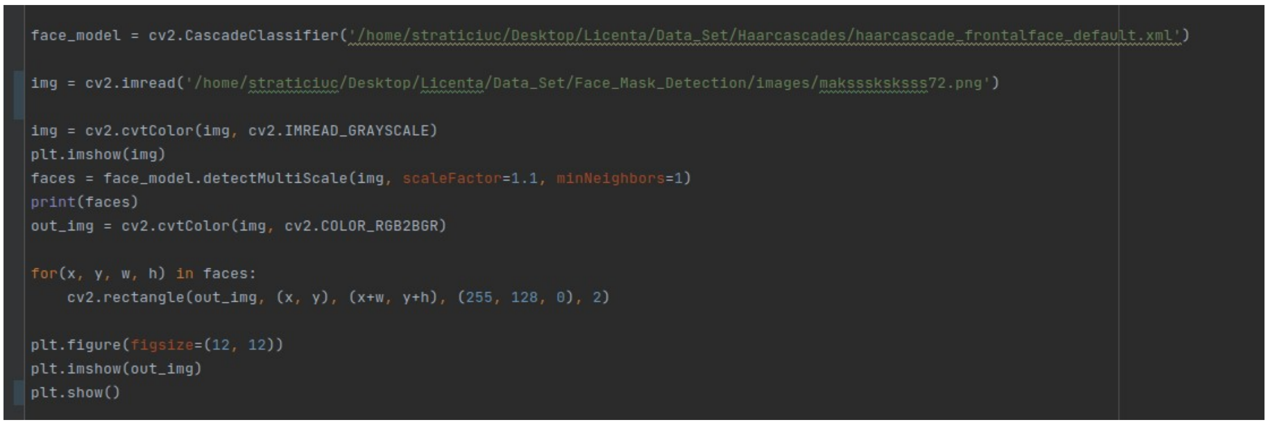


```
import os
for dirname, _, filenames in os.walk('/home/straticiuc/Desktop/Licenta/Data_Set'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

face_model = cv2.CascadeClassifier('/home/straticiuc/Desktop/Licenta/Data_Set/Haarcascades/haarcascade_frontalface_default.xml')

model = load_model('masknet.h5')

model.summary()
```

Figure 4.1

To detect faces in an image we firs of all we load the **haarcascade_frontalface_default.xml** model by using a method from CV2 package CascadeClassifier. The .xml file used it's open source that was obtained based on learning approach where was used a lot of positive and negative images.

After we have prepared the face model using **haarcascade_frontalface_default.xml** we need to load the image that we want to process by using the imread method from CV2 package. By processing the image using haarcascade we obtain the coordinates of all faces in the image, we do that because the model that we have trained can work with small images with faces only.



```
face_model = cv2.CascadeClassifier('/home/straticiuc/Desktop/Licenta/Data_Set/Haarcascades/haarcascade_frontalface_default.xml')

img = cv2.imread('/home/straticiuc/Desktop/Licenta/Data_Set/Face_Mask_Detection/images/maksssksksss72.png')

img = cv2.cvtColor(img, cv2.IMREAD_GRAYSCALE)
plt.imshow(img)
faces = face_model.detectMultiScale(img, scaleFactor=1.1, minNeighbors=1)
print(faces)
out_img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)

for(x, y, w, h) in faces:
    cv2.rectangle(out_img, (x, y), (x+w, y+h), (255, 128, 0), 2)

plt.figure(figsize=(12, 12))
plt.imshow(out_img)
plt.show()
```

Figure 4.2

As an output from function detectMultiScale we obtain a vector that contains coordinates of all faces found in the image. Those coordinates then can be used in order to extract the faces from the image.

Figure 4.3 Image before processing

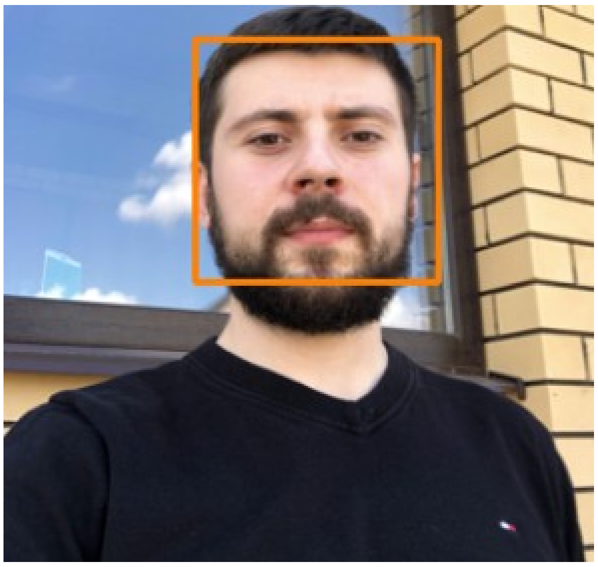


Figure 4.4 Image after passing through haarcascade.xml model

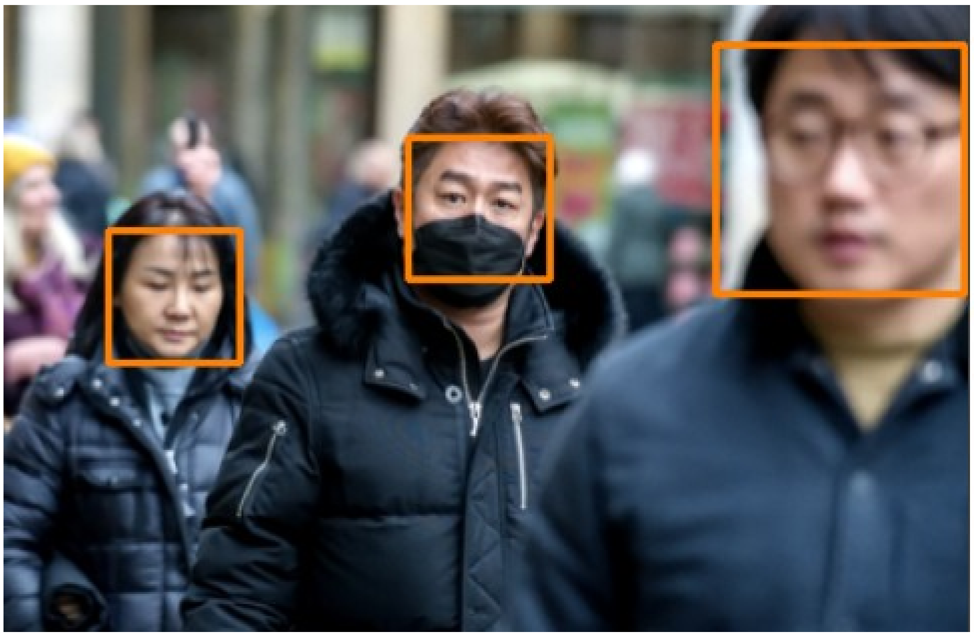
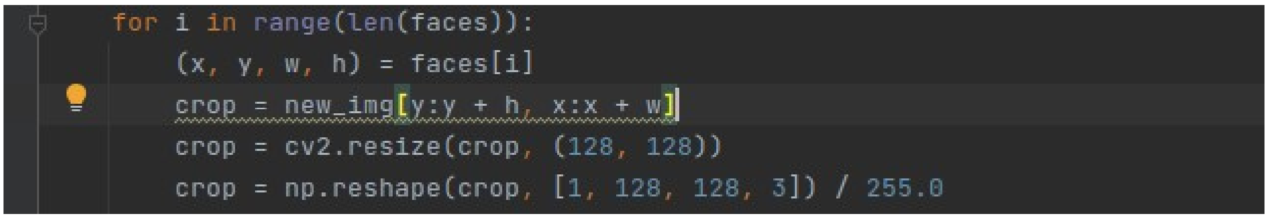


Figure 4.5 Image after processing with more the 1 person

The final step is to extract faces from the original image and create new images and those images to pass to the model and detect if a person in the image wear the mask or not. This process may look difficult bat practically having the coordinates of faces all what we need to do is to corp image, resize it in order to correspond with training images used.



```
for i in range(len(faces)):
    (x, y, w, h) = faces[i]
    crop = new_img[y:y + h, x:x + w]
    crop = cv2.resize(crop, (128, 128))
    crop = np.reshape(crop, [1, 128, 128, 3]) / 255.0
```

Figure 4.6

These corp then we pass to the model. In order to be easier to interpret the result we well put a text and draw rectangles around the faces in original images, green color of rectangle means that the person wear mask correctly and the red color means the opposite thing. In case if someone cant distinguish there clear the color we put a text “MASK” in correct case and “NO MASK” in the wrong one.

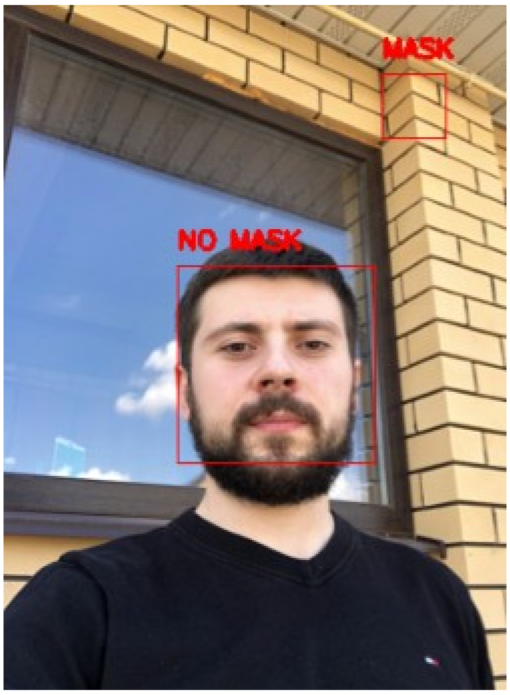


Figure 4.7

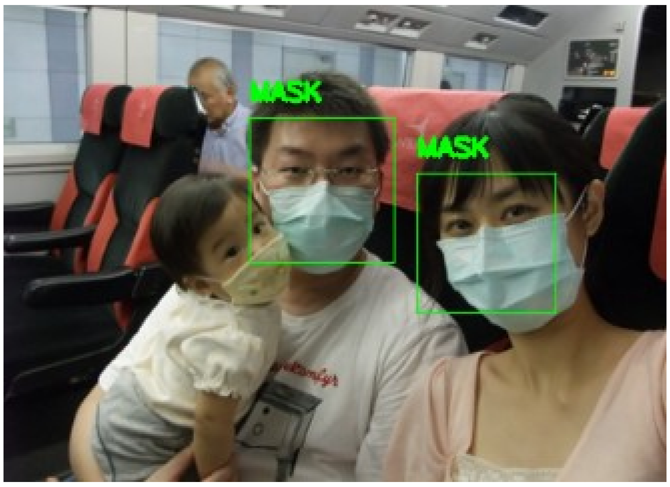


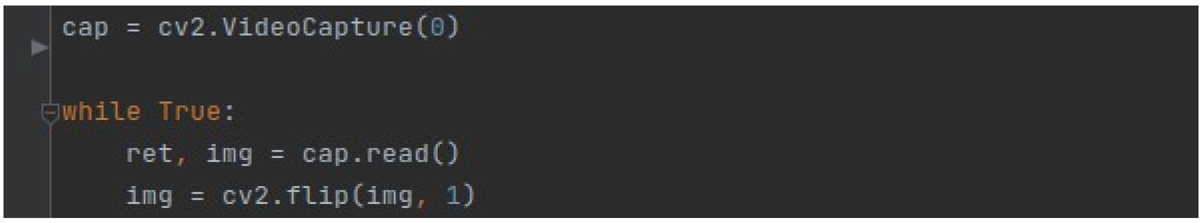
Figure 4.8

As we can observe there are errors in process of detecting faces for example in image from figure 3.6 a face was detected in thee top right corner and in the image from the figure 3.7 the face of the children it was not detected at all. Those errors is caused by the quality of the images, better quality equals better detection this aspect become more evident when we tried to use trained model to detect if a person wear a mask or not from a video stream.

3.3 Detecting Faces and Medial Mask from a video in the real time

The main purpose of the project was to make a program that can detect in real time if a person wear the mask or not. For example one usage of that program is to integrate it in a video security system of a magazine, being a public places if the personal will be able to detect who don't wear the mask this will reduce drastically the infection rate in those magazine.

As in the case of detection on static images the steps are the same with one difference in that case we don’t load image from a storage but using CV2 we capture video from a camera the we split it and the we use the image obtain for detection.

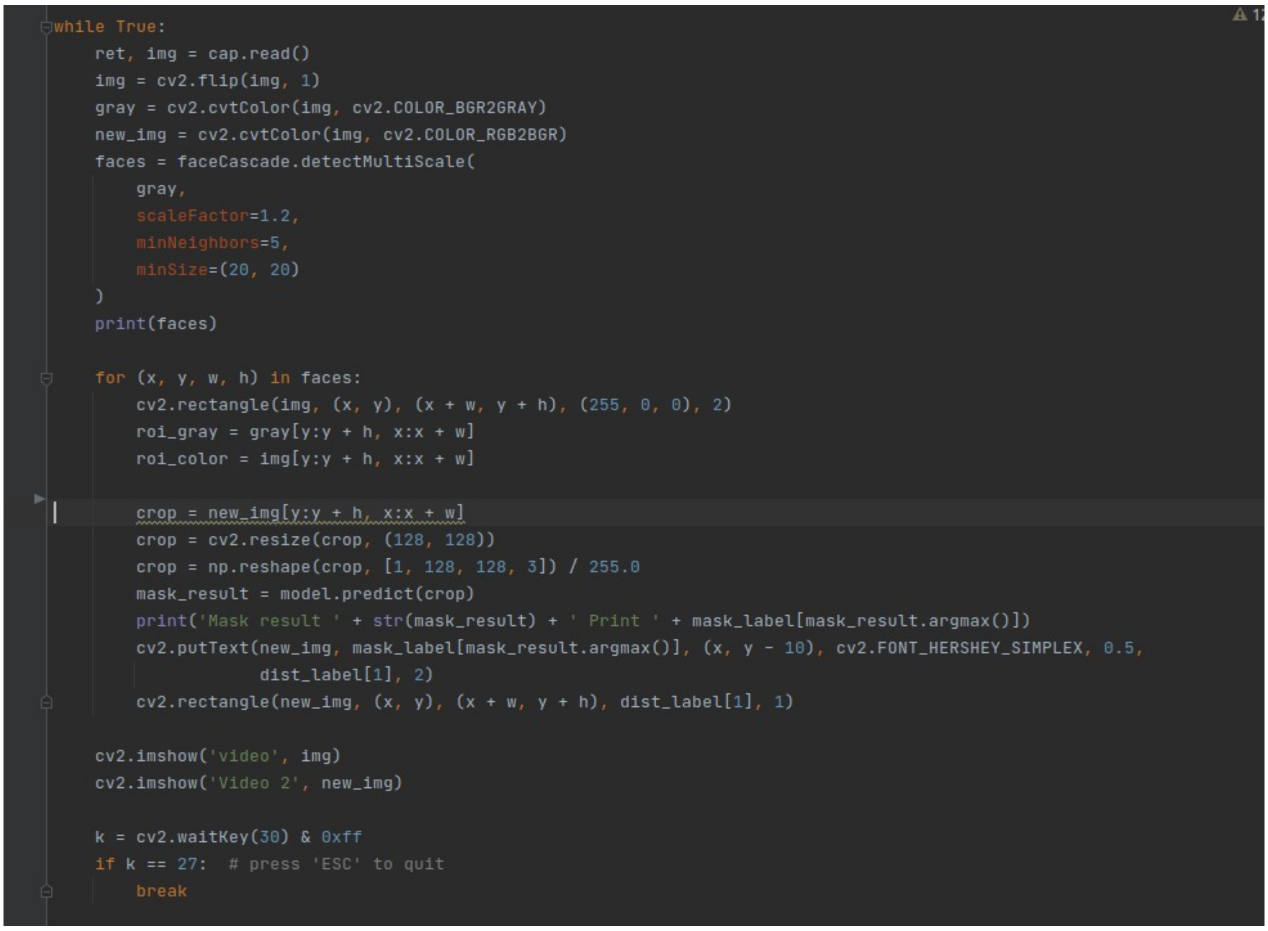


```
cap = cv2.VideoCapture(0)

while True:
    ret, img = cap.read()
    img = cv2.flip(img, 1)
```

Figure 4.9

In the Figure 3.8 we see the usage of the flip function, **cv2.cv.flip(src,flipCode[, dst]): src** is input array **dst** is the output array of the same size and type as **src flip code** is a flag to specify how to flip the array; 0 means flipping around the x-axis and positive value means flipping around y-axis and negative value means flipping around both axes.

We repeated the same steps until thee program is closed. As output using the **imshow** function we simulate the video stream by put the image because we do that very fast the user have impression that it is a video.



```
while True:
    ret, img = cap.read()
    img = cv2.flip(img, 1)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    new_img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.2,
        minNeighbors=5,
        minSize=(20, 20)
    )
    print(faces)

    for (x, y, w, h) in faces:
        cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)
        roi_gray = gray[y:y + h, x:x + w]
        roi_color = img[y:y + h, x:x + w]

        crop = new_img[y:y + h, x:x + w]
        crop = cv2.resize(crop, (128, 128))
        crop = np.reshape(crop, [1, 128, 128, 3]) / 255.0
        mask_result = model.predict(crop)
        print('Mask result ' + str(mask_result) + ' Print ' + mask_label[mask_result.argmax()])
        cv2.putText(new_img, mask_label[mask_result.argmax()], (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                    dist_label[1], 2)
        cv2.rectangle(new_img, (x, y), (x + w, y + h), dist_label[1], 1)

    cv2.imshow('video', img)
    cv2.imshow('Video 2', new_img)

    k = cv2.waitKey(30) & 0xff
    if k == 27:  # press 'ESC' to quit
        break
```

Figure 4.10

Unfortunately during the test we have found that the trained model fails to process the image and as result the detection is wrong in most of the cases, from our estimation it fails in 80% of situations.

In the image below we can see that the detection of the face is done correctly and as consequence we extract with success the face (in that case my face) and pass it to the model for evaluation.

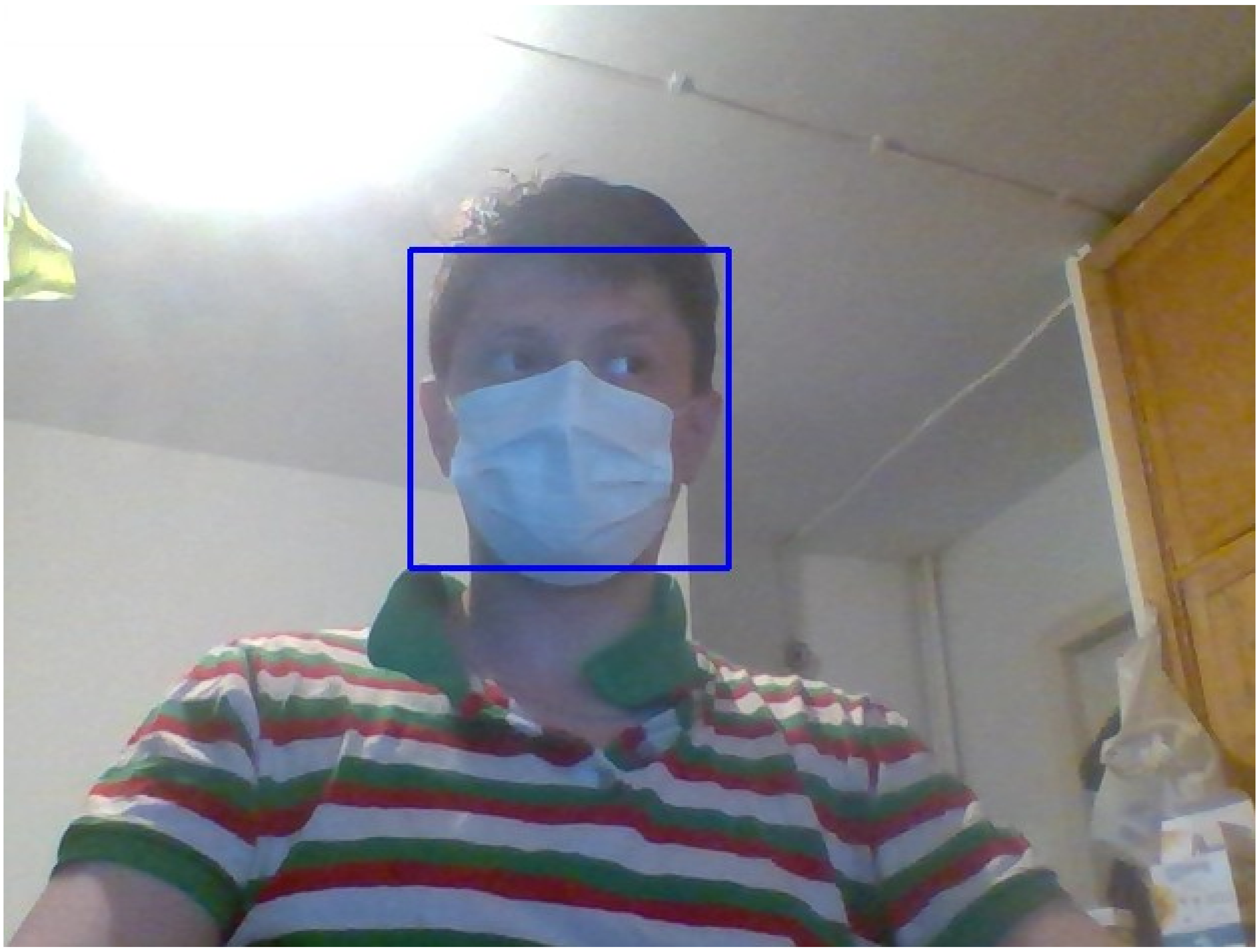
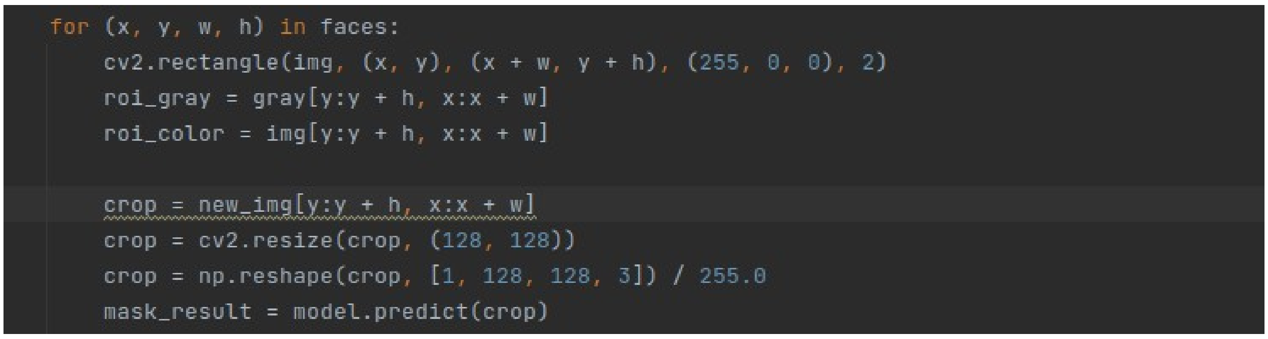


Figure 4.11



```
for (x, y, w, h) in faces:
    cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)
    roi_gray = gray[y:y + h, x:x + w]
    roi_color = img[y:y + h, x:x + w]

    crop = new_img[y:y + h, x:x + w]
    crop = cv2.resize(crop, (128, 128))
    crop = np.reshape(crop, [1, 128, 128, 3]) / 255.0
    mask_result = model.predict(crop)
```

Figure 4.12

Unfortunately the detection as I mentioned was wrong, as we can see in the figure 3.12 even if I’m wearing the mask correctly the model don't detect it.

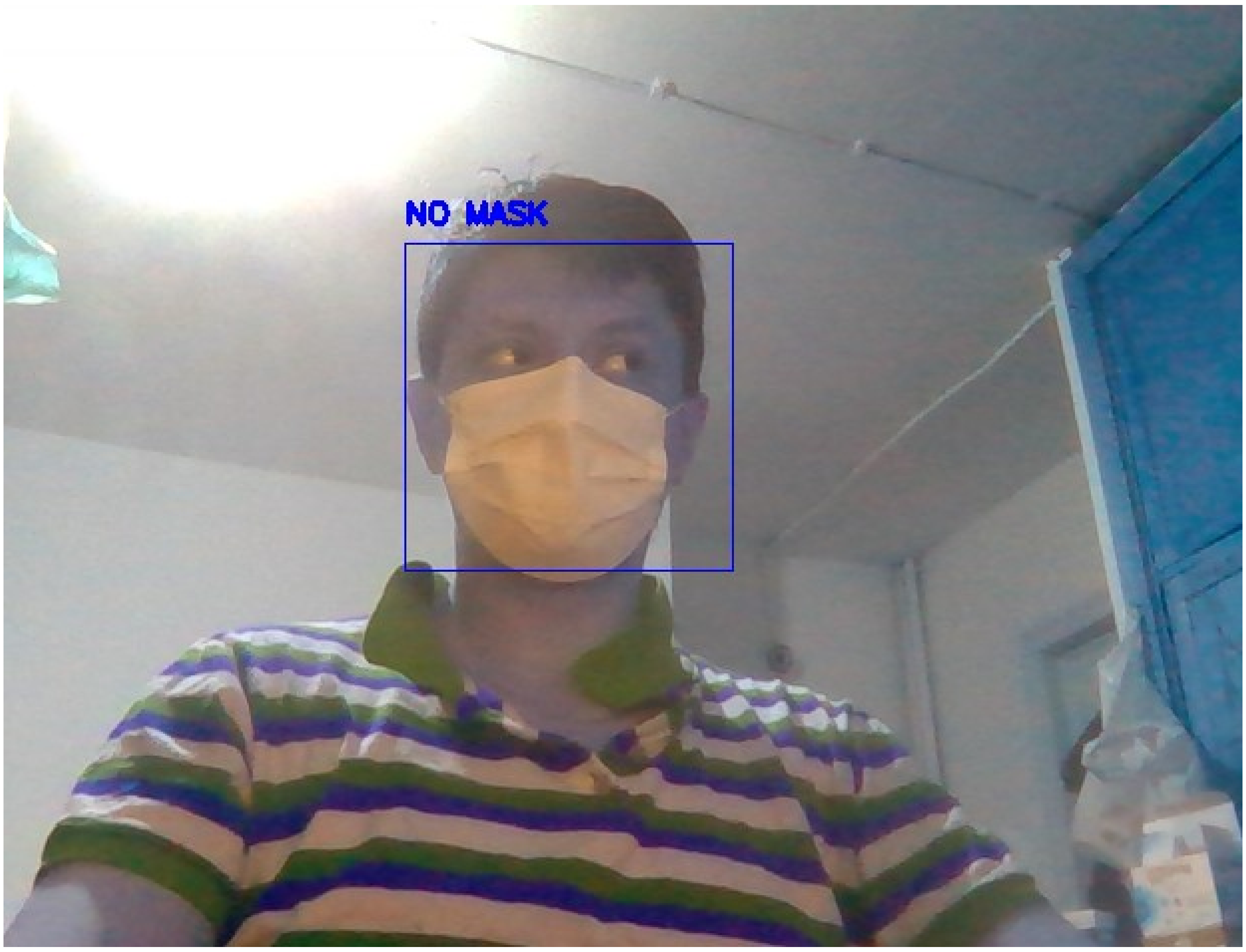
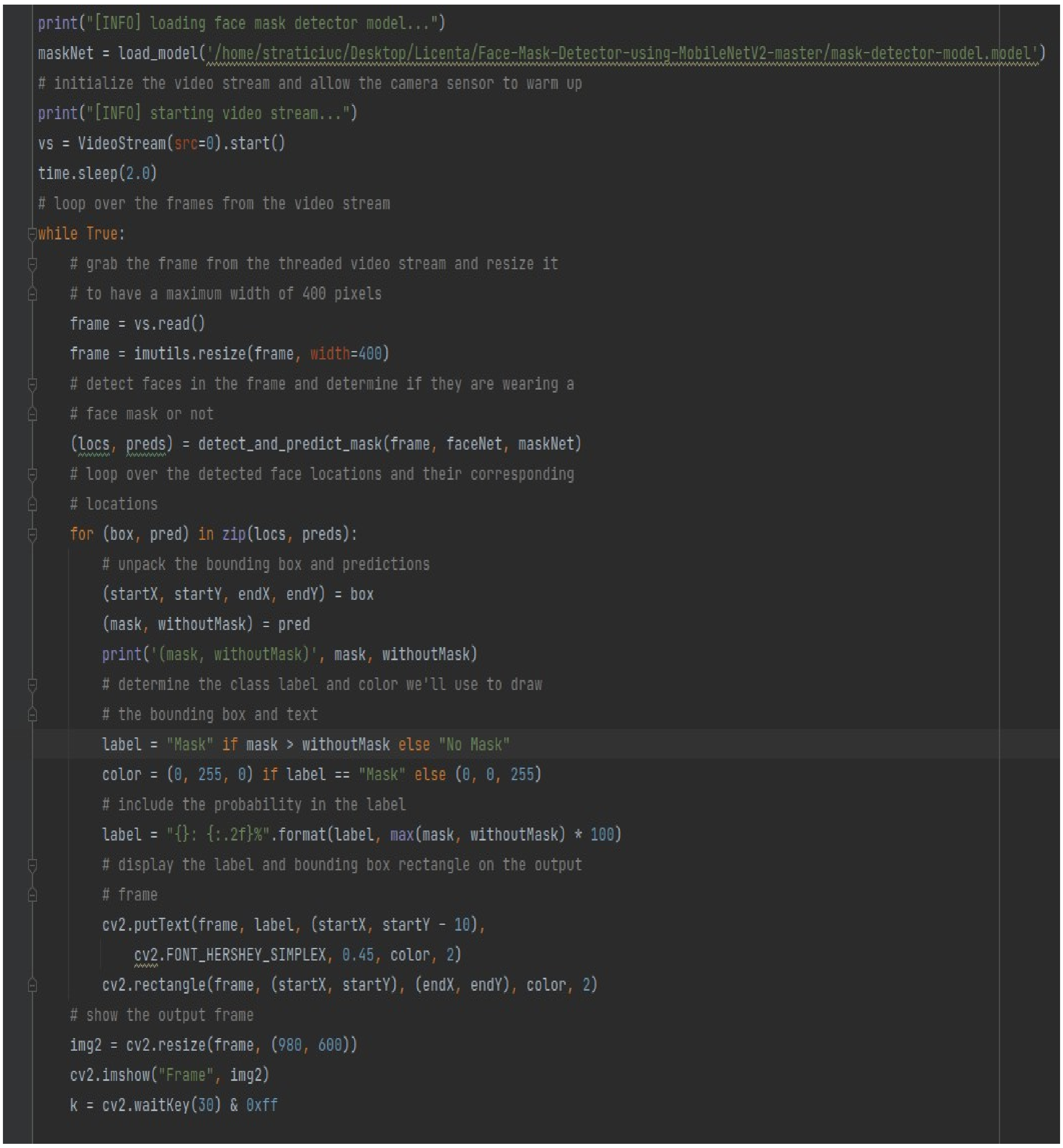


Figure 4.13

During the analysis we arrived to the conclusion that the wrong detection is the result of the low quality of the image obtained from the camera. When we resize the image in order to be processed by the model we lose even more from the quality of the image and on the resulted image mask cant be detected.

Because we cant improve the video camera we need to use another model and this one will be used only for processing the static images but due to the lack of hardware resources and a very big data set of high quality images for training for the next example we will use a per-trained model mask-detector-model.model .

In the image below we can see that the steps of processing the video is the same as in the previous cases but with one difference processing the image is passed to another function **detect_and_predict_mask(frame, faceNet, maskNet)** that function return a prediction number and coordinates of the face. Having those 2 parameters we can easily draw a rectangle around the face which will have green color in case if the person wear the mask or red otherwise.



```
print("[INFO] loading face mask detector model...")
maskNet = load_model('/home/straticiuc/Desktop/Licenta/Face-Mask-Detector-using-MobileNetV2-master/mask-detector-model.model')
# initialize the video stream and allow the camera sensor to warm up
print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()
time.sleep(2.0)
# loop over the frames from the video stream
while True:
    # grab the frame from the threaded video stream and resize it
    # to have a maximum width of 400 pixels
    frame = vs.read()
    frame = imutils.resize(frame, width=400)
    # detect faces in the frame and determine if they are wearing a
    # face mask or not
    (locs, preds) = detect_and_predict_mask(frame, faceNet, maskNet)
    # loop over the detected face locations and their corresponding
    # locations
    for (box, pred) in zip(locs, preds):
        # unpack the bounding box and predictions
        (startX, startY, endX, endY) = box
        (mask, withoutMask) = pred
        print('(mask, withoutMask)', mask, withoutMask)
        # determine the class label and color we'll use to draw
        # the bounding box and text
        label = "Mask" if mask > withoutMask else "No Mask"
        color = (0, 255, 0) if label == "Mask" else (0, 0, 255)
        # include the probability in the label
        label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)
        # display the label and bounding box rectangle on the output
        # frame
        cv2.putText(frame, label, (startX, startY - 10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
        cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)
    # show the output frame
    img2 = cv2.resize(frame, (980, 600))
    cv2.imshow("Frame", img2)
    k = cv2.waitKey(30) & 0xff
```

Figure 4.14

Below we can see the result of using the new model in the same environment.

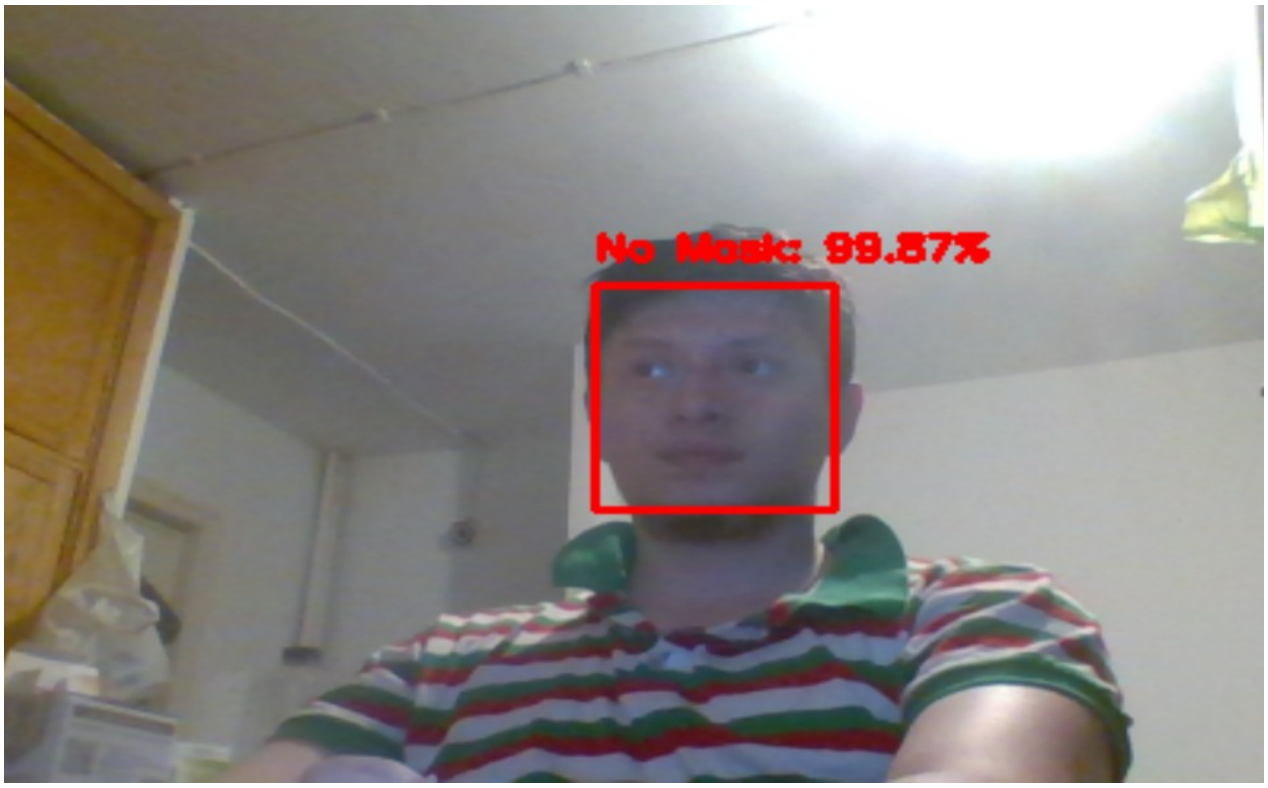


Figure 4.15

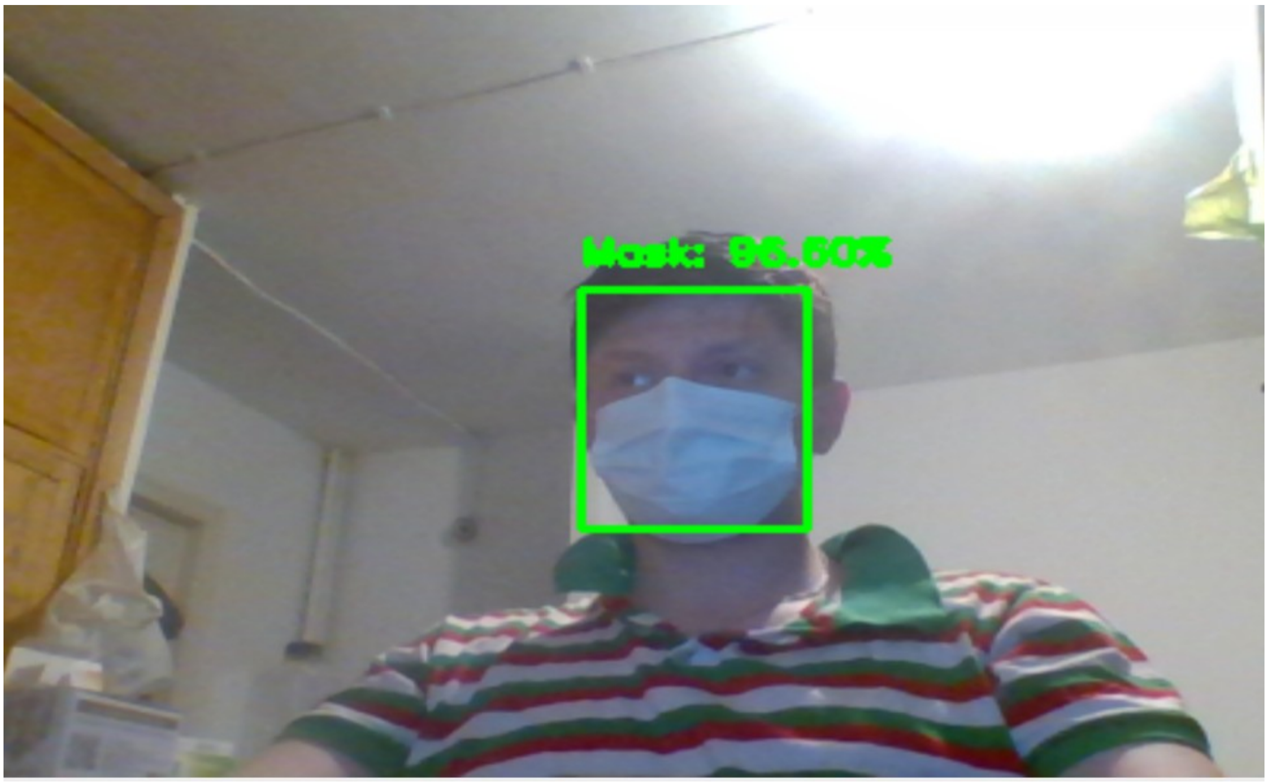
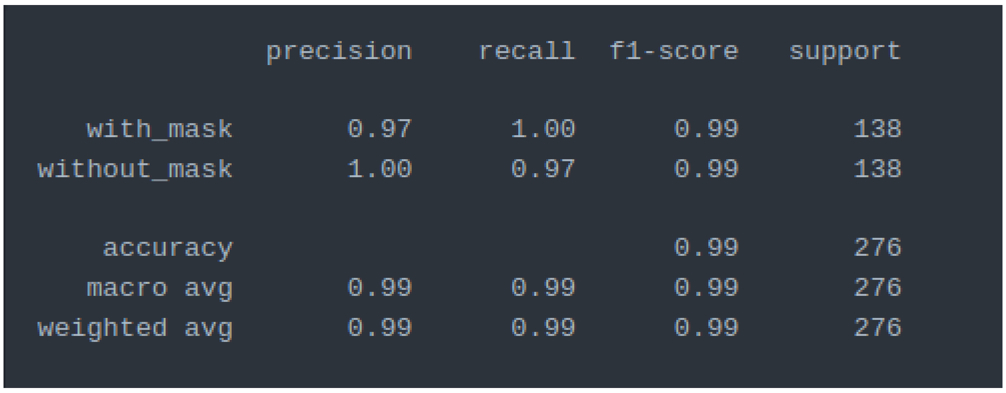


Figure 4.16

If we look at the results we can arrive to the conclusion that those model are trained better then the first one. The results are correct in 98% of the cases this is the result of better training.



```
              precision    recall  f1-score   support

   with_mask       0.97      1.00      0.99       138
without_mask       1.00      0.97      0.99       138

    accuracy                           0.99       276
   macro avg       0.99      0.99      0.99       276
weighted avg       0.99      0.99      0.99       276
```

Figure 4.17

In the figure 4.17 we can see the summary of the model after training and we can observe that the accuracy is quite high.

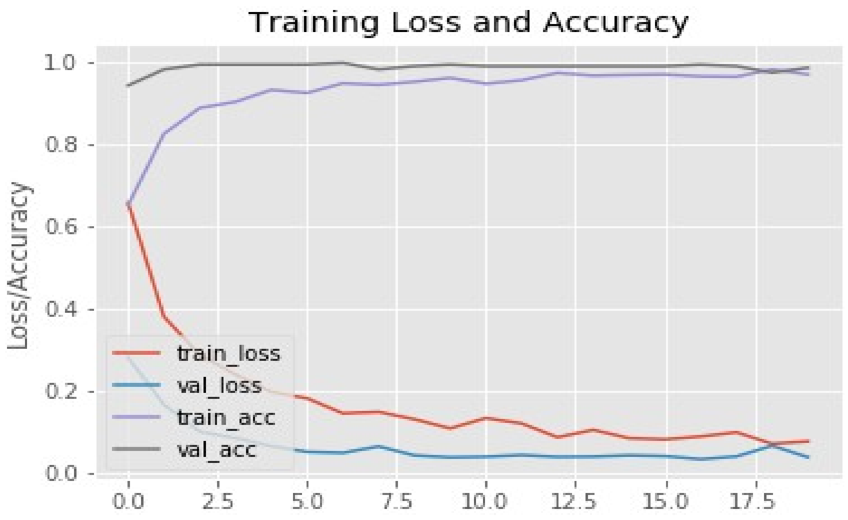


Figure 4.18

In figure 4.18 we can see a graph with training / value loss and accuracy for a easier interpretation because mainly we a interested in models with low losses and high accuracy.

Conclusions

Artificial neural networks, Deep Learning and Artificial Intelligence are the 3 fields of the computer science that in fact have the one purpose, to make machines to think as a humans. Nowadays those fields are the most active one and in a continuous growing because the amount of data increase logarithmic and the old ways of processing are no more efficient. According to the statistics we can process only 30-40% from the total amount of data that we posses. We can easily say that we are entering in the era of artificial intelligence.

Image recognition domain register one of the biggest grow in the last years from the domains where are used machine learning and deep learning. This effect was caused by COVID-19 where the detection if the rules are respected or not is crucial in order to maintain situation under control and also by the increasing domain of automatically driving. Practically it becomes impossible to implement algorithms of processing images in the classical way due to a high calculations process that should be done with matrices and that is why machine learning is used. A big advantage is that we can train a thousands of models or agents at the same time and at the final chose the best one in a reasonable time, on the opposite side if we will try to implement thousands of variations of the same algorithm we will never end. Even thought the machine learning cant perform miracles, we need to be very carefully with input data set and architecture in order to train correctly. If we will be careful and train correctly models not only image processing domain will become better but also another scientific and commercial applications.

In these thesis it was presented tow applications   those propose is to help us to solve a contemporaneous problem known as COVID-19 using deep learning progress in processing images. At the current development stage the first application is good at detecting persons on static images and if they wear the mask or not when the second application is good at detecting if a person wear the mas or no from a live stream video and show the results in the real time.

Like in in cases of another application there are limitations such: we need a powerful environment in order to run those programs and the quality of the image should be as high as possible. Even if the results are not perfect overall after the training we obtained a model that can be used on static images with reasonable results and the second application also proved that it is able to process even a low quality video and give a good result.

Bibliography

1. https://www.who.int/emergencies/diseases/novel-coronavirus-2019/question-and-answers-hub/q-a-detail/coronavirus-disease-covid-19-masks

2.https://www.datacamp.com/community/tutorials/machine-deep-learning

3. https://towardsdatascience.com/what-is-deep-learning-and-how-does-it-work-f7d02aa9d477

4.https://www.ibm.com/cloud/learn/supervised-learning

5.https://www.ibm.com/cloud/learn/unsupervised-learning

6.https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/

7.https://www.mygreatlearning.com/blog/hierarchical-clustering/

8.https://www.ibm.com/cloud/learn/neural-networks

9.https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/

10.https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f

11.https://www.programmersought.com/article/85153864604/11.

12.https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/

13.https://faroit.com/keras-docs/1.2.0/

14.https://keras.io/api/applications/vgg/#vgg19-function

15.https://docs.opencv.org/master/d1/dfb/intro.html

16.https://keras.io/api/preprocessing/image/

17.https://keras.io/api/layers/reshaping_layers/flatten/

18.https://www.pyimagesearch.com/2018/12/24/how-to-use-keras-fit-and-fit_generator-a-hands-on-tutorial/

19.https://github.com/ikigai-aa/Face-Mask-Detector-using-MobileNetV2