Bach Khoa University

Mask recognition machine learning algorithms

Report

Alexandre Rousseau
1952001

Summary

Abstract 2
I. Introduction 2
II. Image Analysis 3
III. Artificial Neural Networks 4
a) Basic information 4
b) Neural Network Model 5
IV. Optimization of network model operation 7
V. The process of learning a neuron network 10
VI. Install the requirement 12
VII. Dataset 13
VIII. Data preprocessing 14
IX. Training 17
X. View Accuracy 21
XI. Apply the model in camera 22
XII. Result 25
XIII. Conclusion 26
References 27

Abstract

This report presents a way of using machine learning algorithms to recognize person with a mask or with no mask. To implement this task, an artificial neural network was used, which has a high adaptability and allows work with a very large set of input data. The neural network was described using a program written in Python. The basic problem faced by the designer of objects recognition is to collect a sufficient training set of images to achieve the high probability of correct recognition. The set of learning patterns in the artificial neural networks may contain from several dozen thousands to one million training samples. In this report at the beginning the neural network was pre-trained trained based on the images included in the publicly available database. It contains 3830 images assigned of 2 categories. The report contains a description the database, the requirement to the functionate of the software, the data processing, the training part and use a model in real time for the camera. The results of proposed solution are presented in the form of screenshots. And a video will be present in to the file. Attention was also paid to the impact of used database for learning the network on the speed of calculations and recognition efficiency. The proper selection of number and types of layers, number of neurons, activation function and the value of the learning factor is very important in designing the neural network in application to objects recognition contained in the images. The problems occurring in the process of learning the neural networks and suggestions for their further improvement and presented.

I. Introduction

The subject of the use of artificial intelligence, in situations where it is impossible to clearly classify data, is enjoying increasing popularity nowadays. Artificial neural networks (ANN) are used in smartphones, autonomous cars, or translational tools. Thanks to their ability to lean, based on searching for similarities between objects and their generalization, they can deal with problems where a very accurate classification is required.

The report presents an application for recognizing objects in image using machine learning algorithms, which task is to recognize objects visible in the image and assigning them the correct label. Particularly important here is the high ability to identify objects that were not previously included in the training set, which distinguished neural networks from other algorithms.

II. Image Analysis

Before it becomes possible to use the ANN as a model using machine learning algorithms, it is necessary to properly process the image, which can be divided into several stages as shown in Figure 1. Segmentation is the first activity carried out in the process of machine learning to recognize objects.

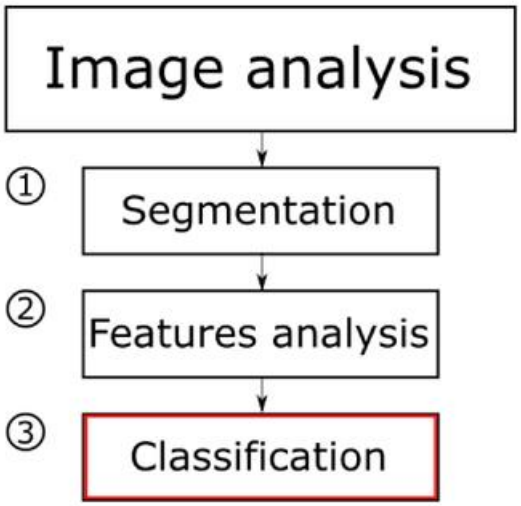

Figure 1: Image Analysis Diagram

The image is here divided into parts that are somehow related to each other. This is to pre-isolate areas that belong to a given objects, its boundaries, form, or limit sending to the next stage of unnecessary information in the computer memory.

The next stage us the analysis of image features, which makes it possible to reveal and describe the object properties, which often remain unnoticed using only eyes.

Object features can be grouped into several basic groups, e.g., geometric, nongeometric, topological. The choice of which image features are to be analyzed is an individual matter, depending on the result. For the purposes of this algorithm, however, the most important is the mathematical side of the processing the analyzed image features, resulting in the so-called signatures and skeletons, that is, one-dimensional functions representing the contour of an object. After completing these preliminary actions, there is a process of object recognition using the artificial neural networks, in which it is possible to use the deep learning methods.

III. Artificial Neural Networks

a) Basic information

ANN is the name of mathematical structures and their software or hardware implementation, consisting of individual elements called neurons capable to performing basic operations at their input.

The principle of the neuron can be represented as follows:

$$e = \sum_{i=1}^{n} xi\, wi$$

$$y = f(e)$$

Where:
xi – value of the i-th input signal, wi weight factor of the i-th signal
n – number of neurons in the input signal
e – total value of neuron stimulation
y – value on the output neuron
f – activation function

After calculating the sum e, using the weight factors wi and input signals xi, the result is multiplied by the activation function f, which should meet the following conditions:

- Continuity of change between your minimum and maximum values
- Derivative continuity (the derivative should also be not difficult to calculate)

  The ReLU activation function was used which parameters are depicted in Figure 2. This type of activation function is currently the most-used activation function for leaning neural networks that are designed to recognize objects in images. This is due to her endlessly striving response for a valid signal and zeroing the neuron value for a negative signal.

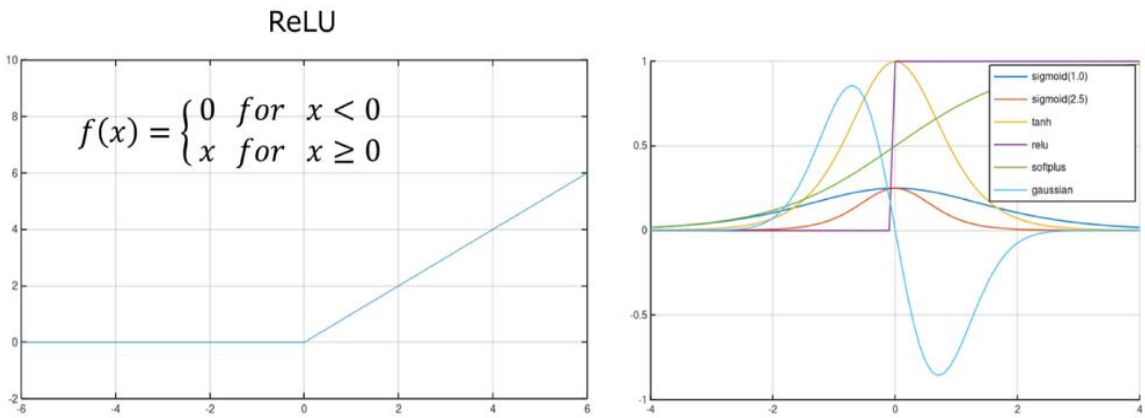

Figure 2: Presentation of the ReLu activation function and comparison of its first derivative with derivatives of other activation functions

Such approach means that not all neurons are used in the network, which protects it from overfitting and speeds up the process of network learning. In addition, ReLU calculates the derivative very simple, and its linearity enables the use of back error propagation method to male correction in weight coefficients in the network.

b) Neural Network Model

Creating a program for recognizing objects in the image requires the development of a mathematical model and a comprehensive approach due to the large amount of data necessary for processing and classification. One of the possible solutions for recognizing objects in the image, due to the effectiveness of operation, is the use of ANN, in which the correct machine learning algorithms will be implemented. The diagram od the neural network used in this program is depicted in Figure 3.

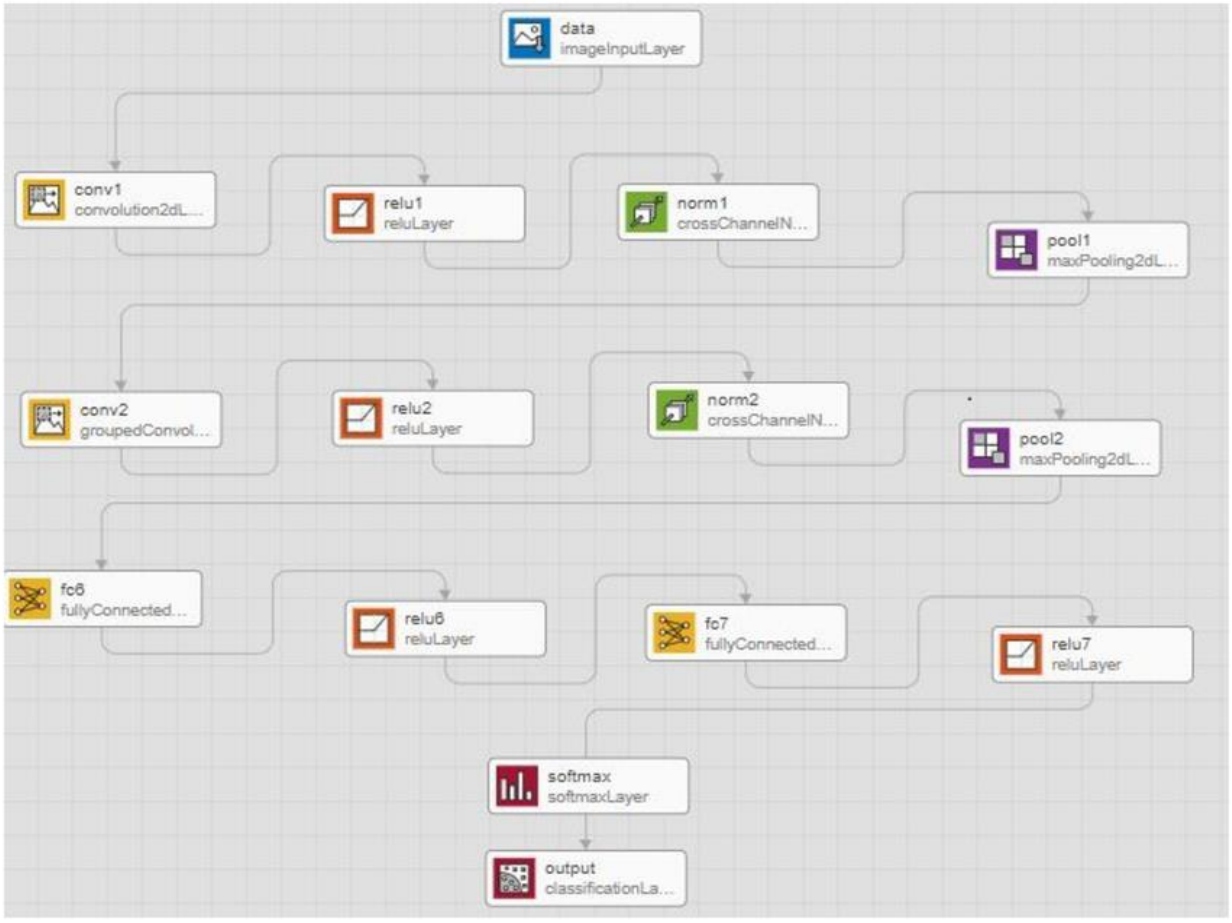

Figure 3 : Proposed model of a neural network

This is a feedforward network. Information in this type of network architecture only moves forward, Figure 4. The network consists of several layers, which is a very common solution. The result is a complex structure with many connections, prone to so-called overfitting. The network consists of an input layer, hidden layers, and an output layer, which by their nature can be further divided into two groups.

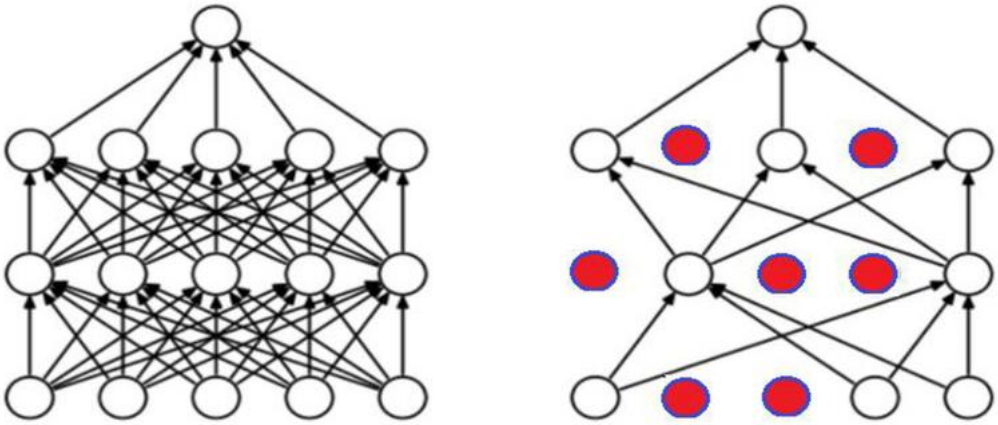

Figure 4 : A network with fully connected layers (left) and a regularized network (right)

- The first three layers of the network (including the input layer), which adapted to the expected characteristics, reduce the "dimensionality" of the image, and then using the characteristic hierarchical pattern contained in the data (several characteristics features are responsible for shape recognition not the whole image because some pixels carry more information than others), allows the use of fewer neurons and connections between them to achieve the same effect. This is particularly important in the case of multi-layer networks susceptible to data overtraining.
- The layers with full connections of neurons to each other (this network is still one-way), which are the last three layers for data processing (including the output layer, with the number of neurons equal to the number of objects possible to classify) and two layers with 4096 neurons, which are responsible for mathematical calculations. Figure 4 shows the differences between the architecture of layers.

The use of the algorithm allows to accelerate the operation of the neural network, because the programmer acts as a teacher who knows what values he expects at the output of the neural network, determining the correctness of the algorithm result. When the discrepancy is too large, he gives a suggested change in value that will allow correct recognition of the object and instructs the algorithm to make the next iteration – the network learns knowing what result it should obtain, so initially random values on individual neurons will quickly set at the level enabling assumed network operation. The idea of the back propagation method is shown in Figure 5.

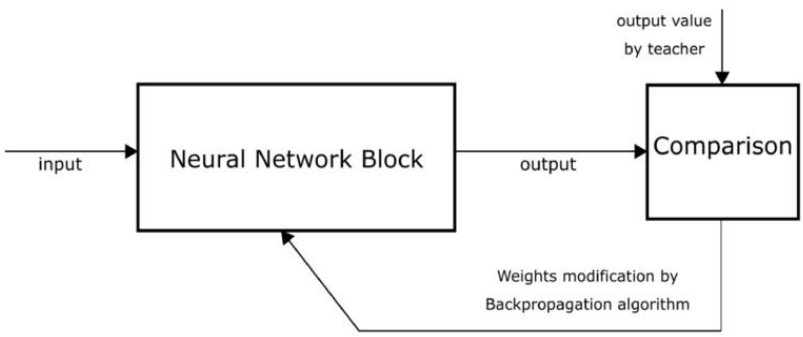

Figure 5 : Idea of learning algorithm with teacher together with weight change by back

The method is used to calculation the neuron values – it does so use mathematical formulas. The basis of its operation is the use of knowledge about the result to be obtained at the output of the network. Then the error is calculated between the suggested value and that obtained by the network and the error is corrected by changing the value on neurons. This simple task is complicated by a network consisting of several layers, which is the basic type used in more advanced artificial intelligence algorithms. In this case, the back propagation algorithm sums up the neuron errors from the hidden layer preceding the last modified layer and only then corrects their values and weights between them.

The operations of the algorithm can be represented as follows:

- Randomizing weight values and assigning them to a neuron
- Loading the input, the data (already in mathematical form)
- Input of the suggested output value and on its basis the calculation of the values at the outputs of neurons and their comparison. Calculation of network output error
- Calculation of errors in the hidden layer behind the output layer, considering the error of the output layer and the sum of the hidden layer error (maintaining the weights between neurons)
- Repeating the procedure for the next hidden layer, using its summed errors and the output error preceding the hidden value calculated in the previous step.
- Repeating the procedure for the next hidden layer, using its total error and the output error from the preceding hidden layer, calculated in the previous step
- After calculating the errors in all hidden layers, the algorithm changes the values of neurons on the input layer in the sale way
- All weights in the network are modified
- The error decreases with each complete pass of the algorithm until it falls to the teacher's acceptable level

IV. Optimization of network model operation

The neural network model previously presented consists of several basic elements. In addition to using its typical neuros network layers consisting of neurons and activation functions, it also has elements responsible for the optimization of the network's operation, which allows its proper calculations and blocks the possibility of overtraining, despite a very large amount of data.

The model's function norm1 is responsible for the local response normalization, through the procedure so-called attenuation, whose task is to normalize the infinite activity of the neuron. Two types of normalization are possible, each of which tends to strengthen the excited neuron and suppress neighboring in the range of limited values. The method searches for the strongest neuron responses and normalizes the responses of neighboring neurons, making the selected neuron even more sensitive to object features. If all the responses of neighboring neurons are large enough, then the function will limit the values in all neurons from a given channel, because they will not accept any of them to be particularly sensitive. In addition to normalization, this function also limits the number of neurons used in the learning.

There are two types of normalization: in the same channel (a group of neighboring neurons) and between the channels, which involves considering the neighborhoods in three, but not in two dimensions.

Below is the formula describing the process of two-dimensional normalization, i.e., for neighboring groups neurons:

$$b_{x,y}^{i} = \frac{a_{x,y}^{i}}{\left(k + a\sum_{j=\max(0,i-\frac{n}{2})}^{j=\min(N-1,i+\frac{n}{2})}(a_{x,y}^{j})^{2}\right)^{3}}$$

Where:

- $b_{x,y}^{i}$: normalized output of the kernel "I" (weight summation place) at position (x, y)
- $a_{x,y}^{j}$: kernel source output attached to position (x, y)
- N: total number of neurons
- n: normalization channel size
- k: function hyperparameter

After normalization, the data is being sent to the statistical filter so-called max pooling, which extracts the maximum value from the mask and reduces the number of calculations in subsequent layers. The application of this filter to non-overlapping subregions results in passing only the largest values, which significantly reduces the amount of data necessary for further processing, without reducing the effectiveness of the algorithm. The simplicity of this solution for the 2x2 masking filter is shown in Figure 6.

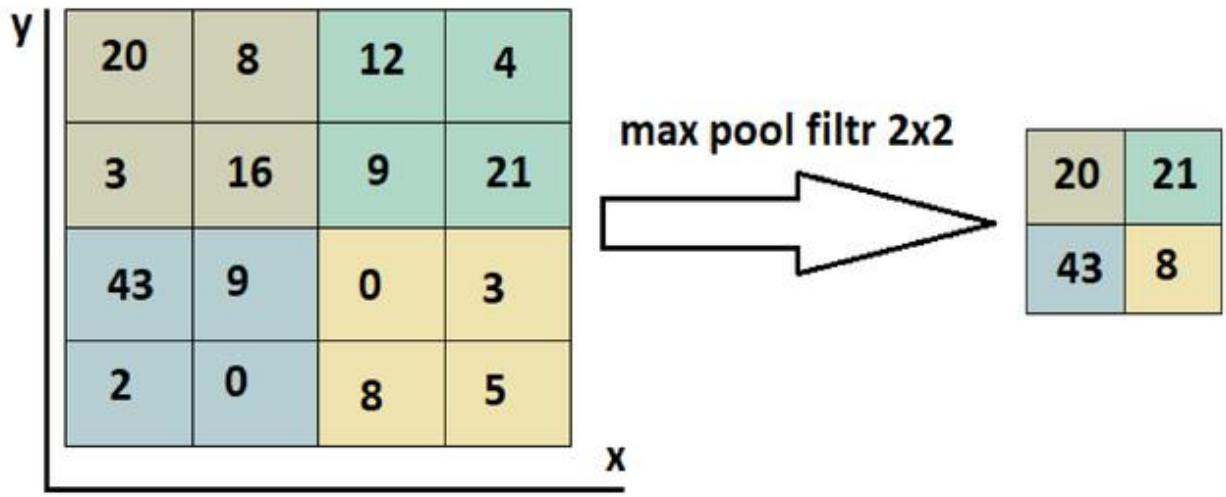

Figure 6 : Operation of max pooling filter

Before the information reaches the network output, it still must go through the SoftMax function. Its task, as a transfer function, is to change the input vector from the received value into the output information in the form of a vector with normalized values between 0 and 1, so that the output layer receives specific information that can be interpreted as a certain probability, which will help

determine the percentage of accuracy networks in case of its analysis. An example of how this function works is shown in Figure 7.

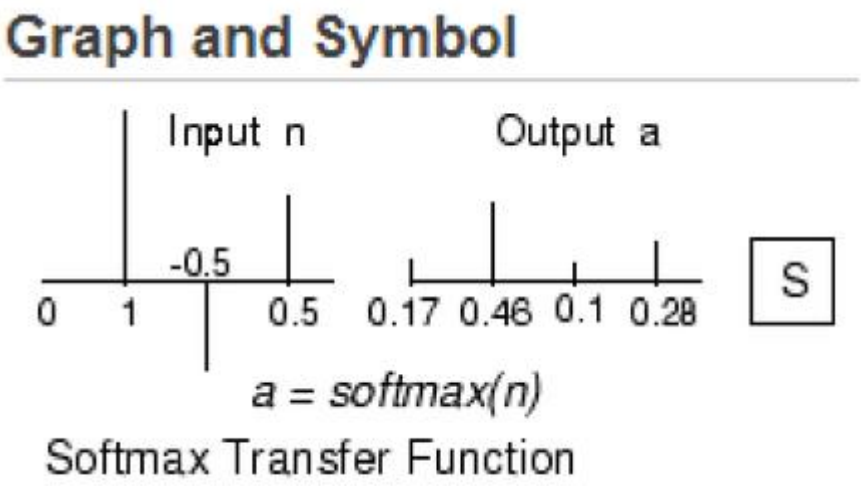

Figure 7 : Operation of SoftMax function

In adjusting the network structure help a such tools as Optimal Brain Damage and Optimal Brain Surgery, which tasks are detection and getting rid of neurons not involved in the operation. The most advanced Google algorithms for object recognition use the patented Dropout method, working similarly to the above-mentioned tools.

Cleaning the network not only ensures faster learning process, but also increases its accuracy, once again making it more resistant to overfitting. You should also remember to keep the optimal number of hidden layers, because too many of them will overwhelm the network, and too small may not be complex enough to solve the task.

V. The process of learning a neuron network

It is extremely important to set the learning rate at the appropriate level in the program. This factor defines by what maximum value can change the neuron weight. When this value is too low, the network learning process is disproportionately longer because it requires much more iterations to make the teacher-specified corrections. However, when it too high, the weight correction performed on neurons will be so large that it will prevent the correct learning process.

The weights correction with large values means that the calculated values for neurons, despite subsequent iterations, will not be able to fit into the network scheme, which will cause them to be random all the time, such as in the initial phase of the learning process. When choosing the right learning rate (Figure 8), the network learns correctly and after a few epochs already achieves the required recognition accuracy. Further learning has no greater impact on the improving recognition accuracy.

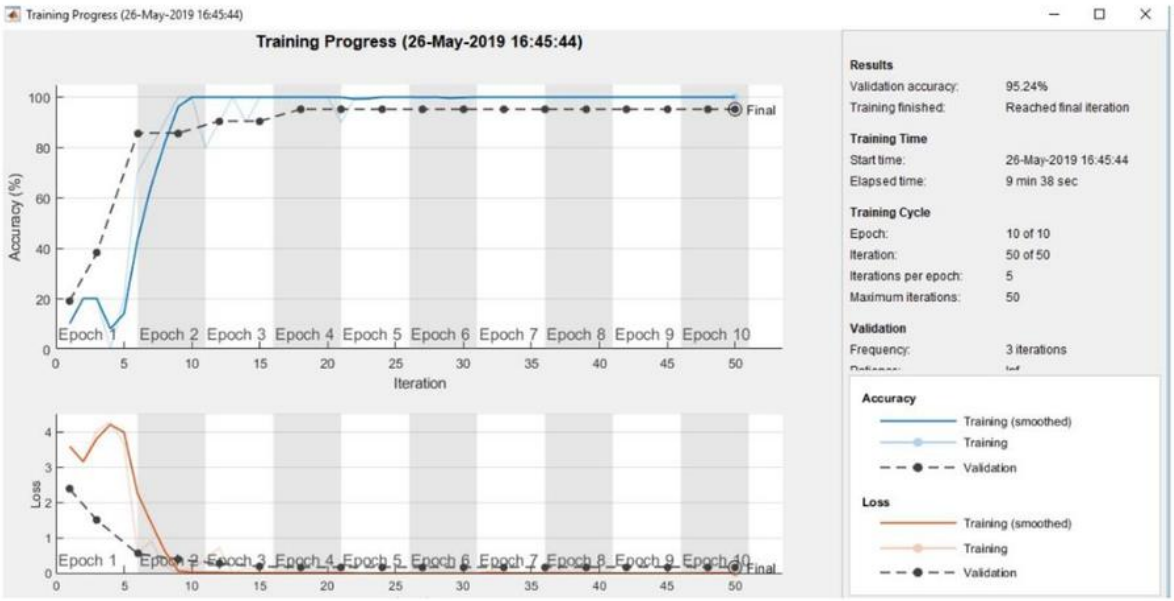

Figure 8 : Network test for learning rate = 1

If the value of this factor is set too high (Figure 9), the network learning process is very fast, which is not always a positive phenomenon, because the network can stop at the local minimum and thus will not achieve maximum recognition accuracy. Incorrect selection of weights in the process of learning a network can also extend its learning time.

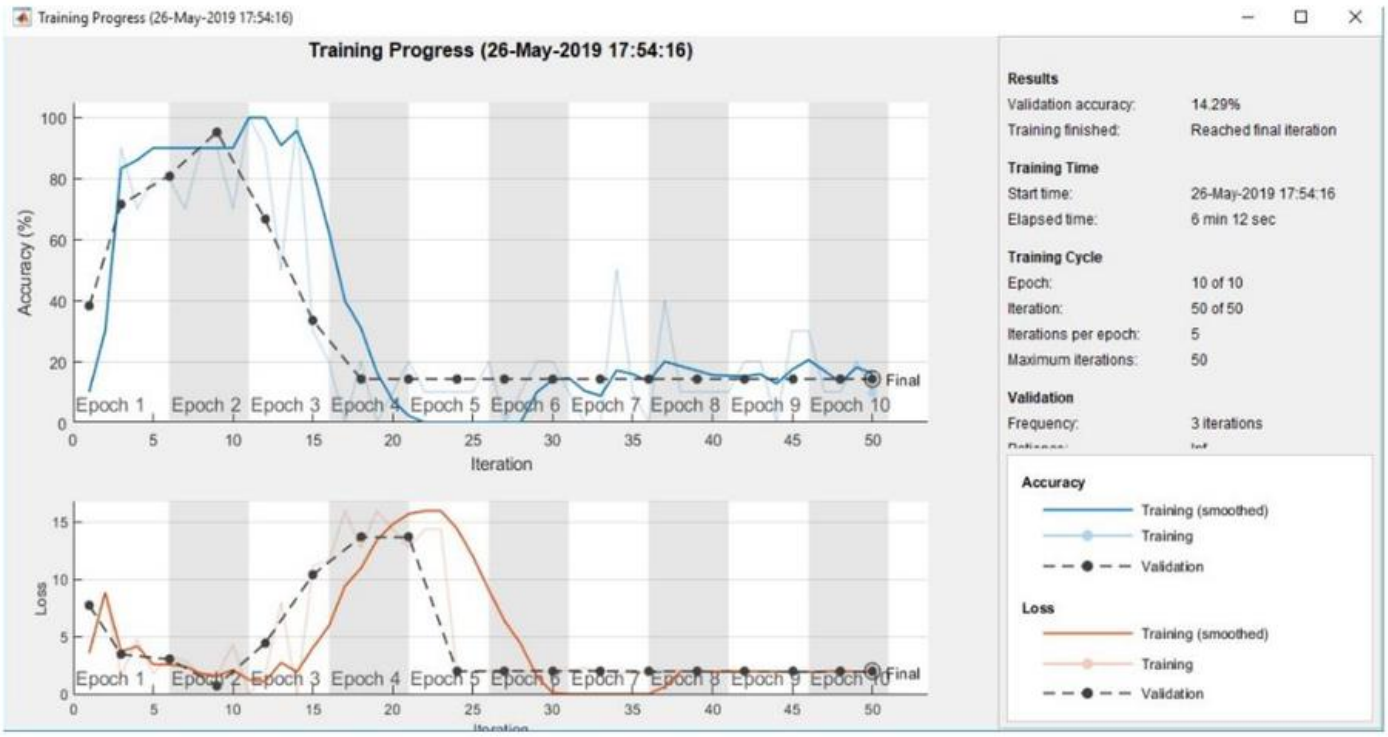

Figure 9 : Network test for learning rate = 1

In the final phase, when the ratio has decreased due to the use of the Drop Learning Factor function, the network is stuck on minimum accuracy values and will not work properly. Too little correction of weights, as shown in Figure 10, will unnecessarily extend the duration of the learning process.

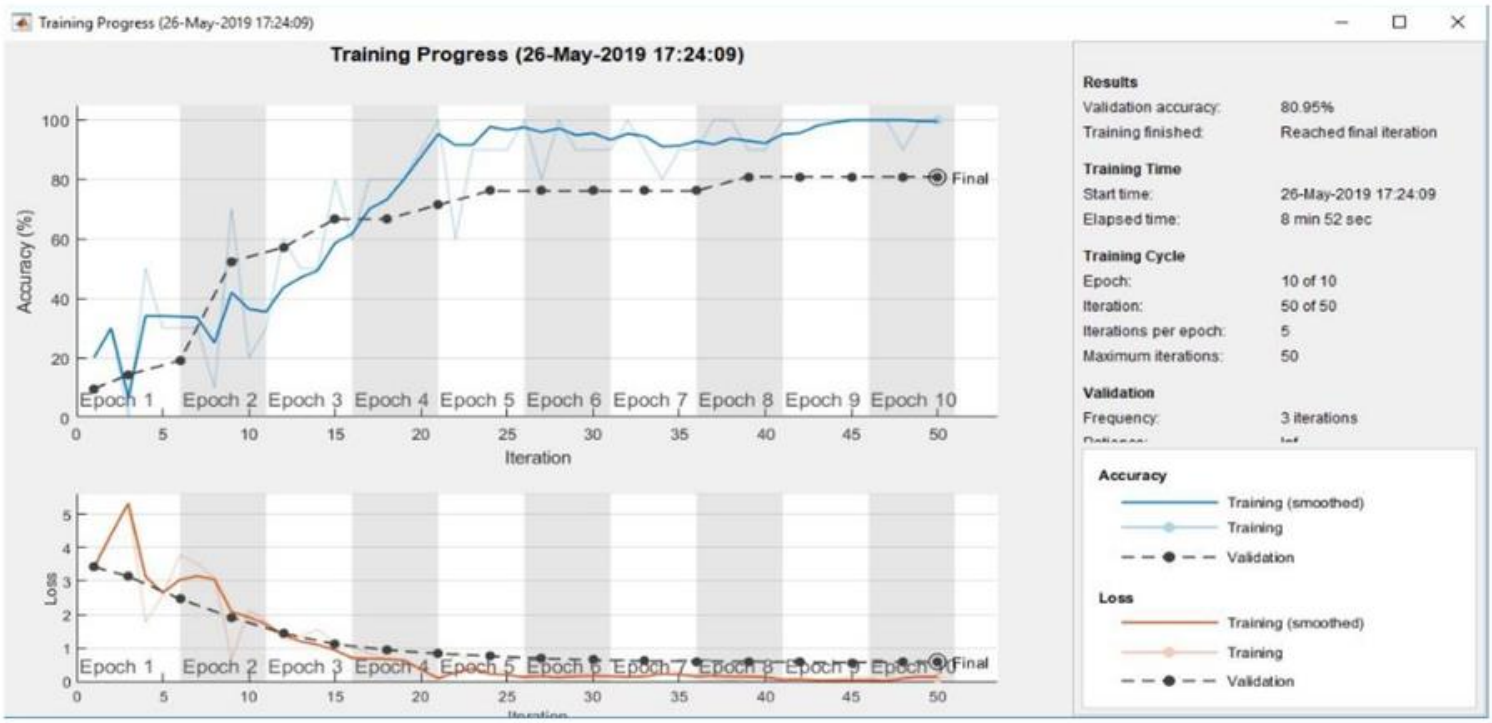

Figure 10 : Network test for learning rate = 1

As you can see from the above calculation result, the worst the network is setting too high weight correction, which can cause the mean square error to accidentally set bellow the value, even though the weight distribution on the neurons will not yet guarantee the correct operation of the network. To prevent such a situation, the momentum rule was used in the developed software application. It works by coupling together the correction values of weights from the previous and current iteration, through the factor α. Such use the momentum causes that the weights correction (learning rate) starts from a certain high level, and then decreases with each iteration as it approaches the value specified by the teacher at the output. This approach extends the learning process and refines it and at the same time protects against accidental ending of the program due to a lot number of weights correction.

The method of using the parameter α is represented by the formula, which describes the change in weights used in the gradient method:

$$\Delta w_i^t = -\eta \frac{\partial Q^t}{\partial w_i} + \alpha \Delta w_i^{t-1}$$

Where:

- Δw_i^t : modification of weight i after iteration t
- η : coefficient of change in the value of neuron weights (learning rate)
- $\frac{\partial Q^t}{\partial w_i}$: gradient fall
- α: weight correction factor from previous and current iteration

The process of network learning depends on the correct use of all the above algorithms and their methods. For the networks to work properly, it needs to provide a lot of training data, while using as few neurons as possible and connections between them to avoid overtraining.

VI. Install the requirement

In the project, we must install the requirements into our Windows Prompt. In this case, I used Anaconda Prompt. Anaconda is a Python distribution for scientific computing (data science, machine learning application, large scale data processing, predictive analytics), that aims to simplify package management and deployment.

So, the requirements are:

- TensorFlow
- Keras
- Imutils
- Numpy
- OpenCV-Python
- Matplotlib
- Scipy

TensorFlow is an open-source framework for machine learning on decentralized data.

Keras is an API in Python for deep learning. It follows best practices for reducing cognitive load, it offers consisting and simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable error messages.

Imutils is a series of convenience functions to make basic image processing functions such as translation, rotation, resizing, skeletonization, and displaying Matplotlib images easier with OpenCV and Python.

Numpy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived object (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation.

OpenCV-Python is library of Python bindings designed to solve computer vision problem.

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

Scipy provides algorithms for optimization integration, interpolation, eigenvalue problems, algebraic equations, differential equations, statistics, and many other classes of problems.

VII. Dataset

In the dataset, there are two different categories, the first one is “mask” category and the second one is “without mask” category.

The mask category contains all the images of people with mask wearing in their faces. So, there are around 1900 images. I took these images from Kaggle, and few open sources image libraries and google images. And there is also another folder called without mask inside which there are cases of people without mask. this is our data set. And only with this data set, we are going to train our model.

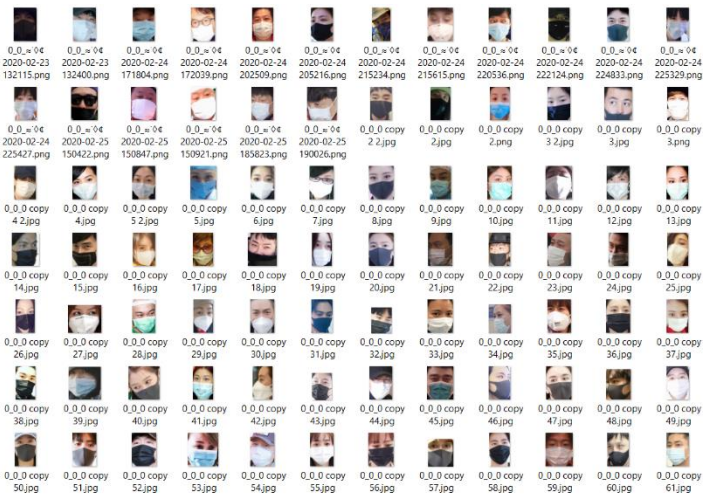

Figure 11: Dataset with Mask

Figure 12: Dataset without Mask

VIII. Data preprocessing

In this part, what we are going to do is we are going to convert all our images from the folders that we have the width mask and without mask into arrays. So that with those arrays will create a deep learning model. Create a file called train_mask_detector.py. And these are my inputs, I'll tell what these imports are as we go through the code.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import os
```

Figure 13: Imports

Going down there is my directory. Inside my directory, I mentioned where my data set folder is present. And there are these categories. Inside categories, I'm having two values called with mask and without mask, which are the folders that this person inside this directory.

```
DIRECTORY = r"C:\Users\alexp\Desktop\projet recognization mask\Face-Mask-Detection-master\dataset"
CATEGORIES = ["with_mask", "without_mask"]
```

Figure 14: Directory

Print loading images so that we have context on what's happening. And there are two lists, I have created two lists called data and labels. Both are empty list. And inside this data, I'm going to append all my image arrays inside this data list. And inside this labels list, I'm going to append all those corresponding image which are with masker without much so basically, they contain the label of those images indicating whether those images are with mask images, or without mask images.

```
34  print("[INFO] loading images...")
35
36  data = []
37  labels = []
```

Figure 15: Data / Labels

Through the categories inside Categories list. There are two values with mask and without mask. We can see the directory here is the data set folder inside which there is two values called with mask and without mask. So, what we do is we first looping through the with Mask and I'm going to loop through the without mask. And once that is done once I get the path of this particular with mask or without mask. what list there does this It kind of lists down all the images inside that directory. With the mask inside with us there are 1900 images, which will be listed by this. We are going to join the path of this particular with mask to the corresponding image. We are going to call a function called load image. So, this load image is coming from the keras processing preprocessing load_image. So inside keras that preprocessing image there is a function called load_image. This load image loads our image path. And we give the target size to be 224 and 224. What this target size is nothing, but it is the height and width of the image. We are converting all the size of the image uniformly to 224 and 224. So that our model will be perfect in that shape. And then once we load the image, just save it to a variable called image. We are going to convert that image to an array by using the image to array function. This image to array function is coming from the keras preprocessing image module. Once we convert the image to array, and then we are going to use the preprocess input. What preprocessing input is nothing, but we are going to use mobile nets for this model. We'll dive deep into mobile nets later. Once we convert our image to array, and we have preprocessed the input successfully, then what we need to do is we need to append our image which is an array to the data list. And once that is done, we need to open their corresponding labels in inside this label. Category is nothing but as you know, with “with mask” or “without mask”, so that now we have all the image array inside this data list. And all those are label array inside these labels.

```
for category in CATEGORIES:
    path = os.path.join(DIRECTORY, category)
    for img in os.listdir(path):
        img_path = os.path.join(path, img)
        image = load_img(img_path, target_size=(224, 224))
        image = img_to_array(image)
        image = preprocess_input(image)

        data.append(image)
        labels.append(category)
```

Figure 16: Category

We are going to convert that into one of our arrays. To do that, we are calling the label binarized method. Which is coming from the SK learn preprocessing module. With that, what we are going to do is I'm going to convert the “with mask” or “without mask” into categorical variables. We successfully convert that into numerical values, like zeros and ones, what we need to do is we need to convert them into NumPy arrays, basically, now it is a list both data and our labels are just a list as we mentioned here, but we kind of need to convert this data and labels into NumPy arrays because mainly with arrays or deep learning models will work. We are giving 20 percentage of the images to the testing set the 80 percentage of the image will be for the training. This random state is nothing but if we use 42.

```
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)

data = np.array(data, dtype="float32")
labels = np.array(labels)

(trainX, testX, trainY, testY) = train_test_split(data, labels,
    test_size=0.20, stratify=labels, random_state=42)
```

Figure 17: Label Binarizer

IX. Training

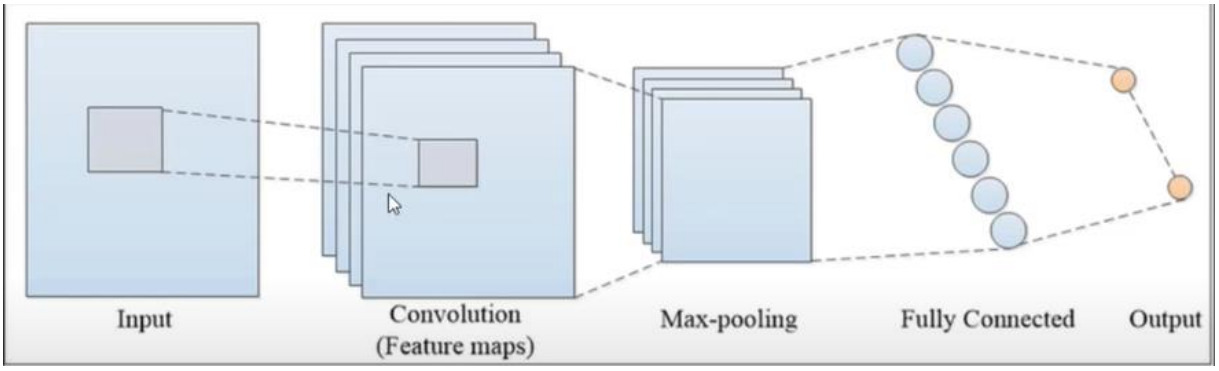

Figure 18: Training processing

After the input image is processed as an array, we will send that into the mobile net. And then we'll do the max pooling, and then we'll flatten it and then create a fully connected layer and get the output. What is the use of mobile net is that mobile net is very faster in process when compared to the convolution neural networks? And mobile nets also use less lesser parameters. But even when mobile nets have a lot of advantages, they do have disadvantage mobile ads tend to be less accurate when compared to its competitors.

I'm giving the initial learning rate to be one e to the power of minus four. So always make sure your learning rate is less. When your learning rate is less, your loss will be calculated properly, which means you will get the better accuracy very soon. In this case, my learning rate to be 0.0001. And we give 20 bucks, and my batch size is 32.

```
INIT_LR = 1e-4
EPOCHS = 20
BS = 32
```

Figure 19: Learning Rate

What image data generator? Does this image data generator kind of creates a documentation? It creates many images with a single image by adding various properties like flipping, rotating the image, shifting the image, and many other properties. So, we can create more data set with this. We give rotation angle is 20, Zoom ranges, 0.15.

```
aug = ImageDataGenerator(
    rotation_range=20,
    zoom_range=0.15,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.15,
    horizontal_flip=True,
    fill_mode="nearest")
```

Figure 20: Image Data Generator

With the use of mobilenetv2, we are creating a base model here. There is a parameter called weights is equal to image net. So, ImageNet is nothing but there are some pre trained model specifically for images. When we use this ImageNet, those weights will be initialized for us, and which will give us better results. But this weight is defaulted to none. But we are not going to use none because ImageNet is what we prefer here. And I mentioned include top is equal to false, where include top is a Boolean value, which is to say whether include a fully connected layer at the top of our network. So, we'll connect the fully connected layer later by ourselves. Input tensor is nothing but the shape of the image that is going through. As we mentioned, in the data preprocessing data, we have set the image size to 224, and 224. So that is what we give here. This three is nothing but the three channels in that image. We are inputting the colored images inside this. So colored images of three channels with this, which is RGB (red, blue, and green). Essentially, shapers, 224 by 234, which is the height and width, and three channels that is red, green, and blue.

```
baseModel = MobileNetV2(weights="imagenet", include_top=False,
    input_tensor=Input(shape=(224, 224, 3)))
```

Figure 21: MobileNetV2

Once our base model is done, the next is to construct our fully connected layer by using the pooling and stubs. What we are doing if we are first creating a head model object in which we are parsing the base model output as the first parameter. And then we create the pooling and pool sizes seven by seven. And then we go to flatten these layers, and then we add a dense layer with 128 neurons. And my activation layer here is “relu”. So, “relu” is basically the go to activation function for for nonlinear use cases. And then we use dropout just to avoid overfitting of our models. And then we go to our final output model.

Our output has two layers just because one is for with mask and the other is for without mask and I'm giving the activation function a SoftMax generally in the output layer, so, you can always go for a SoftMax activation function or a sigmoid activation function because they are probability based as zeros or one’s values.

```
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(128, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation="softmax")(headModel)
```

Figure 22: Head Models

And once that is done, we need to call the model function this model function accepts two parameters one is inputs and the other is output. Our inputs will be the base model dot input and the outputs will be the head model.

```
model = Model(inputs=baseModel.input, outputs=headModel)
```

Figure 23: Model (input and output)

And initially what we need to do is we need to freeze the layers in the base model so that they won't be updated during the first training process, because they are just a replacement for convolutional neural networks. So, we are just freezing them for training.

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

Figure 24: Freeze Layer

And once that is done, we are giving an initial learning rate for this, which is one to the power of four that we mentioned at the top of our code. And then I'm giving decay. And for the loss function in the compiler, I'm giving binary cross entropy. And the optimizer that we are going to use is the Adam optimizer. So, Adam optimizer is also like relu, which is always the go to optimizer for any image prediction methods. And we are going to track the accuracy metrics here. And that is the only metric that we are going to track for.

```
print("[INFO] compiling model...")
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
    metrics=["accuracy"])
```

Figure 25: Model Compile

Now I'm fitting my model and the image data generator that we used right in the beginning. So, I'm also flowing that data here so that we get more training data to train our images. Since we use only 1900 images, I'm going for the image data generator. It is always better to use the image data generator when you have a lesser data set. And for the validation data, we are using the testing data set test X and test Y. And we give the number of epochs that we mentioned before like 20 epochs.

```
print("[INFO] training head...")
H = model.fit(
    aug.flow(trainX, trainY, batch_size=BS),
    steps_per_epoch=len(trainX) // BS,
    validation_data=(testX, testY),
    validation_steps=len(testX) // BS,
    epochs=EPOCHS)
```

Figure 26: Training head

We are going to going to evaluate our network by using the model dot predict method.

```
print("[INFO] evaluating network...")
predIdxs = model.predict(testX, batch_size=BS)
```

Figure 27: Predict Method

Similarly for each image in the testing set, we need to find the index of the label corresponding to the largest predicted probability.

```
predIdxs = np.argmax(predIdxs, axis=1)
```

Figure 28: Predicted probability

We are getting the classification report with a very good formatting.

```
print(classification_report(testY.argmax(axis=1), predIdxs,
    target_names=lb.classes_))
```

Figure 29: Classification Report

And finally, we are saving the model that we generated now, and we are saving it in the h5 format.

```
print("[INFO] saving mask detector model...")
model.save("mask_detector.model", save_format="h5")
```

Figure 30: Saving The Model

We are going to plot our accuracy and metrics by using matplotlib.

```
N = EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
```

Figure 31: Matplotlib

X. View Accuracy

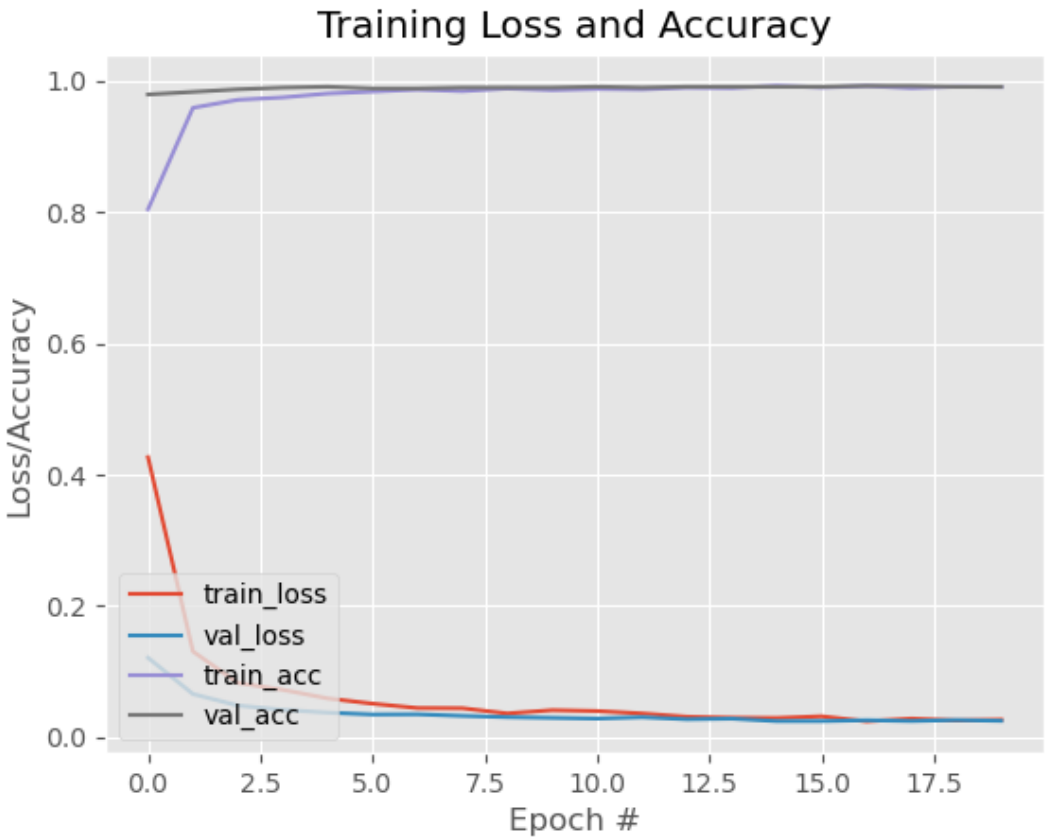

Figure 32: Training Loss & Accuracy

We can see here the training loss and accuracy has been here. The accuracy has been good, and the loss has also been reduced. We got a good model the box down here is up to 20 Epochs that we gave.

XI. Apply the model in camera

With the help of the face detector files will detect the face and with the help of the deep learning model mask detector model, we are going to detect the mask. For this camera operation, we will be using Open CV.

We load the FirstNet person it is nothing but the couple of files that we had for the face detection. They are under the face detector folder. And they're their file names. We just give the path to those files and just saving it in a variable. And then to use them. We use a method called ReadNet, which is under a CV2 and the module called dnn. So, dnn stands for deep neural network. So, CV2 recently developed this dnn with many methods, many useful methods in them So if we give this couple of parts there, so we can use this face to detect the face.

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

Figure 33: ReadNet

Similarly, we are loading our model using load model, the model that we use for mass production. This is the model that we developed. We now have the models for face detection, and the model for the mask detection as well.

```
maskNet = load_model("mask_detector.model")
```

Figure 34: Load Model

Next thing that we need to do is we need to load our camera. To do that, we are using the video stream. Inside this video stream function, there is something called SRC source is equal to zero. So, source is nothing but the camera that you use. If you have three or four cameras, then you can give the index here, so I'm using my primary camera, so you're given zero. if you have two cameras in and you want to use the second camera that you have, you can use one instead of zero here, it kind of works like that. And the start method loads the camera.

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

Figure 35: Starting Video Stream

Then we'll go into the while loop. while true, we are reading the frame, so every frame is nothing but an image. Every image flowing through sequence surely. And with those frames per second. After we read the frame, we just open a frame. And we name it as a frame, and I give the width to be as 400.

```
while True:
    # grab the frame from the threaded video stream
    # to have a maximum width of 400 pixels
    frame = vs.read()
    frame = imutils.resize(frame, width=400)
```

Figure 36: While True Frame

We call that method, and we get the location and print predictions as a tuple.

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

Figure 37: Tuple

Now, we just do a tuple unpacking and get the X and Y coordinates. With these points will draw the rectangle and masking without mask prediction. This is nothing but the percentage of prediction here. The first prediction will correspond to the mask and the second prediction will correspond to the without mask.

```
for (box, pred) in zip(locs, preds):
    # unpack the bounding box and predictions
    (startX, startY, endX, endY) = box
    (mask, withoutMask) = pred
```

Figure 38: Tuple Unpacking

Now that we got mask and without mask, we will just add labels to that. Label will be masked if there is a mask under without mask, so we'll mention no mask. We are now ready to apply this upon the rectangle that we have. For color, we choose RGB. So RGB is the color-coding standard that we have. Green color if there is a mask, and if there is no mask, then we'll put red color. That is what this color coding here is and then we'll display the label.

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

Figure 39: Label Color

We are using format strings here for the label what we are doing is we just display the percentage of the prediction that we have. So, we are just displaying the maximum of mask and without mask, as I said, if there is a mask, this mask will be like 90 percentage and if there is no mask, so this will be 10 percentage. So, the maximum probability here would be nice. We are giving the maximum of those two values here.

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

Figure 40: Label Percentage

The x values are directly there and the y values are practically 10 pixels so that it kind of appears a bit above the box and it does not overlap. We are just drawing the rectangle is drawn on the frame. Every single frame we are drawing the rectangle and we are giving the x and y coordinates for the rectangle and the color coding.

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

Figure 41: Rectangle Frame

We are showing the frame and this frame is nothing but the sequence of pictures that is going to flow through.

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

Figure 42: Frame

We are breaking the while loop here. We are breaking the while loop after we hit the cue button. If key is equal to equal to ordinal of “q”, then we break the loop.

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

Figure 43: Break The Loop

We finally destroy all the windows and stop the video streaming, which is a bit of cleanup here. So, guys, now that we have completed our coding.

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

Figure 44: Stop All Windows

XII. Result

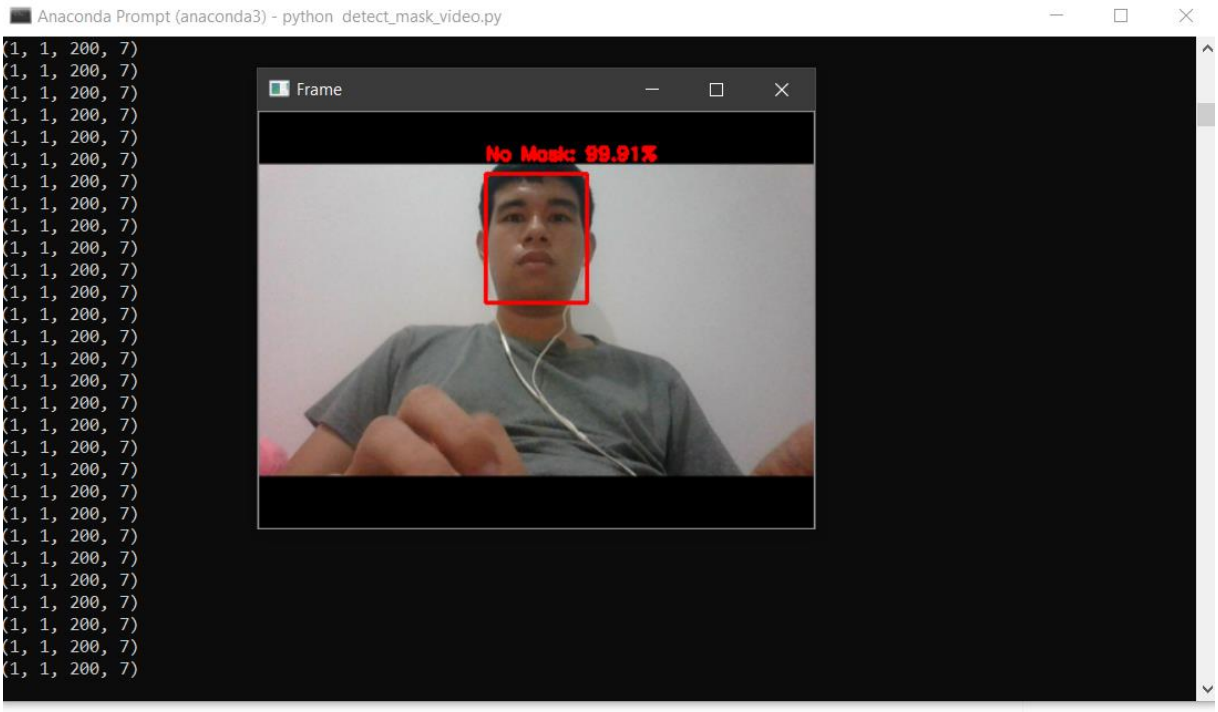

Figure 46: Result Without Mask

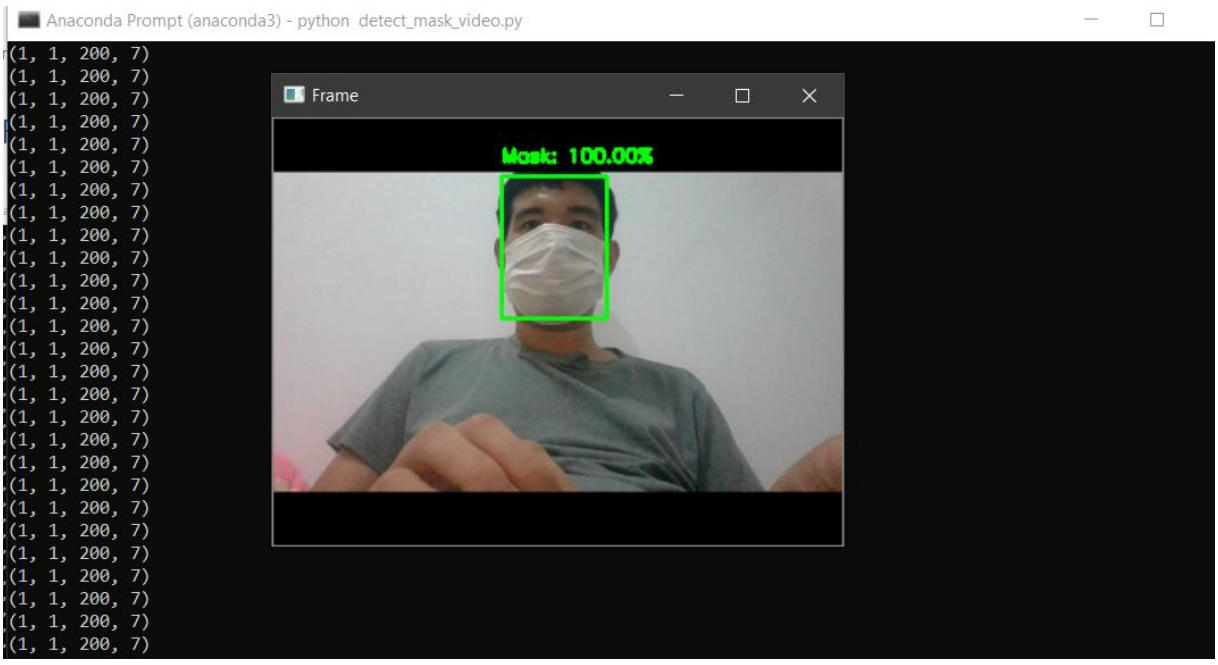

Figure 45: Result With Mask

I made a video for demo. Click here.

Here is my project Folder.

XIII. Conclusion

To sum up, the presented above application enables the different objects recognition in images, applying the machine learning algorithms for classification with using the artificial neural networks. The neural network is an excellent tool for recognizing objects in images, but it should remember about the appropriate selection of its model. The proper selection of number and types of layers, number of neurons, activation functions and the value of the learning factor is also extremely important. The interface of the computer application is based on the use of the Deep Learning Toolbox tool, enabling easy uploading to the previously designed neural networks, databases, or selecting the percentage of data to be used in the process of network learning, object recognition and validation.

The presented experiment results show the advantages of the software used in ANN processing. Knowledge of the neural network architecture allows to decrease the learning time and recognize objects in images in the final system in real time. To be able to use the ANN for recognition in real systems, it is necessary to have many known reference objects, which can be used in the images learning process. The recognition model presented here can be adapted to the needs of any object's recognition.

References

- Chen, S.;Wang, H.; Xu, F. and Jin, Y.Q., "Target Classification Using the Deep Convolutional Networks for SAR Images," IEEE Transactions on Geoscience and Remote Sensing, 54, 4806–4817 (2016).
- Ciresan D. C., Meier U., and Schmidhuber J., "Multi-column Deep Neural Networks for Image Classification," IEEE Conf. on Computer Vision and Pattern Recognition CVPR (2012).
- Hryvachevskyi, A., Prudyus, I., Lazko, L. and Fabirovskyy, S., "Improvement of segmentation quality of multispectral images by increasing resolution," 2nd International Conference on Information and Telecommunication Technologies and Radio Electronics, UkrMiCo 2017 - Proceedings 8095371, DOI: 10.1109/UkrMiCo.2017.8095371 (2017).
- Parrallel Neural Network Training with OpenCL: https://bib.irb.hr/datoteka/584308.MIPRO_2011_Nenad.pdf (27 November 2018).
- Rogers, S.K., Colombi, J.M., Martin, C.E. and Gainey, J.C., "Neural networks for automatic target recognition," Neural Networks, 8, 1153–1184, (1995)
- http://docplayer.pl/13483611-Sieci-neuronowe-wprowadzenie-agnieszka-nowak-brzezinska.html (05.05.2019)
- https://www.cs.toronto.edu/~kriz/cifar.html
- Prudyus, I. and Hryvachevskyi, A., "Image segmentation based on cluster analysis of multispectral monitoring data," Modern Problems of Radio Engineering, Telecommunications and Computer Science, Proc. of the 13th International Conference on TCSET 2016, 7452020, 226-229, DOI: 10.1109/TCSET.2016.7452020 (2016)
- https://www.tensorflow.org/
- https://keras.io/
- https://github.com/PyImageSearch/imutils
- https://numpy.org/
- https://docs.opencv.org/4.x/da/df6/tutorial_py_table_of_contents_setup.html
- https://matplotlib.org/stable/index.html
- https://scipy.org/