

SHRI GURU RAM RAI UNIVERSITY,

PATEL NAGAR, DEHRADUN

{SCHOOL OF CA & IT}

MINOR PROJECT REPORT:

**FACE MASK DETECTION USING DEEP LEARNING**

{2021-2022}

SUBMITTED BY:

Anuj Panthri

BCA, Vth SEM R190529014

PROJECT GUIDE:

DR. Sanjay Sharma

Professor

Department OF CA

**CANDIDATE’S DECLARATION**

I, Anuj Panthri hereby declare that the MINOR PROJECT report entitled “Face Mask Detection Using Deep Learning” submitted to the Shri Guru Ram Rai University, Dehradun in partial fulfilment of the requirements for the award of the Degree of Bachelors of Computer Applications is a record of original Learning undergone by me under the supervision and guidance of DR. Sanjay Sharma , Professor, Faculty of School of CA & IT, Shri Guru Ram Rai University, and it has not formed the basis for the award of any Degree/Fellowship or other similar title to any candidate of any University/Institution.

Date: Signature of the Student

This is to certify that the statement made by the candidate is true to the best of my knowledge and belief.

Signature of Guide

Date: Guide Name with Designation

Countersigned Dean

**ACKNOWLEDGEMENT**

I had an opportunity to complete my MINOR PROJECT in “Face Mask Detection Using Deep Learning”. First, I wish to express my sincere gratitude to my supervisor, DR. Sanjay Sharma, for his enthusiasm, patience, insightful comments, helpful information, practical advice and unceasing ideas that have helped me tremendously at all times in my research and writing of this thesis. Without his support and guidance, this project would not have been possible. I could not have imagined having a better supervisor in my study

**CERTIFICATE**

This is to certify that Anuj Panthri, Enrollment No:R190529014, has Completed the Project report titled “Face Mask Detection Using Deep Learning” under My guidance for the partial fulfilment of the Course: MINOR PROJECT, in Semester V of the Bachelor of Computer Applications.

**Signature**

**(Faculty Guide):**

**Name**

**(Faculty Guide)**

**Introduction**

Wearing face masks is recommended as part of personal protective equipment and as a public health measure to prevent the spread of coronavirus disease 2019 (COVID-19) pandemic.

Face detection has been a fascinating problem for image processing researchers during the last decade because of many important applications such as video facerecognition at airports and security check-points, digital image archiving, etc.

In this project, we attempt to detect faces in a digital image. This project is based on face mask detection using Deep Learning .It is used to find faces in an image and tell if they are wearing mask or not .My project is capable of detecting multiple faces at once.

**Problem Definition and Algorithm**

**Task Definition**

In our problem our task is to accurately detect faces and label them as with\_mask , without\_mask , if there are a lot of people without mask we can generate a warning message ,like “please wear your mask for your and others safety” , your model should be fast and be able to detect multiple faces with accuracy.

We need a model which takes a image and can output the coordinates of the box around the faces and give a probability score for which class the faces belongs to (with\_mask & without\_mask).

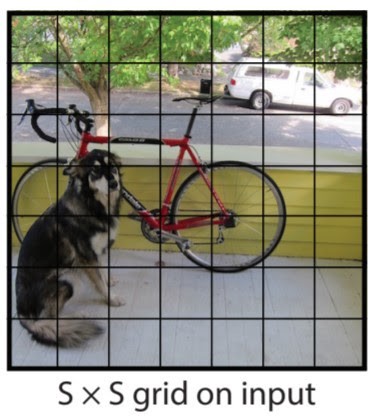
**Algorithm Definition**

**YOLO**

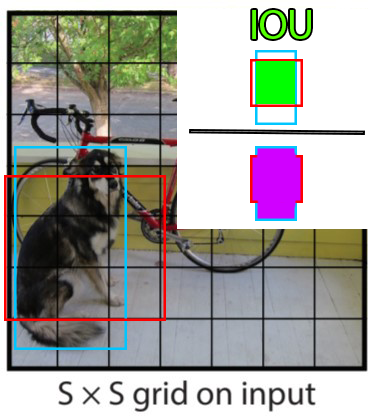
YOLO or You Only Look Once, is a popular real-time object detection algorithm. YOLO combines what was once a multi-step process, using a single neural network to perform both classification and prediction of bounding boxes for detected objects. As such, it is heavily optimized for detection performance and can run much faster than running two separate neural networks to detect and classify objects separately. It does this by repurposing traditional image classifiers to be used for the regression task of identifying bounding boxes for objects.It looks at the entire image at once, and only once — hence the name You Only Look Once — which allows it to capture the context of detected objects.

YOLO can generalize the representations of various objects, making it more applicable to a variety of new environments. Now that we have a general overview of YOLO, let’s take a look at how it really works.

YOLO is based on the idea of segmenting an image into smaller images. The image is split into a square grid of dimensions S×S, like so:



The cell in which the center of an object, for instance, the center of the dog, resides, is the cell responsible for detecting that object. Each cell will predict B bounding boxes and a confidence score for each box. The default for this architecture is for the model to predict two bounding boxes. The classification score will be from `0.0` to `1.0`, with`0.0` being the lowest confidence level and `1.0` being the highest; if no object exists in that cell, the confidence scores should be `0.0`, and if the model is completely certain of its prediction, the score should be `1.0`. These confidence levels capture the model’s certainty that there exists an object in that cell and that the bounding box is accurate. Each of these bounding boxes is made up of 5 numbers: the x position, the y position, the width, the height, and the confidence. The coordinates `(x, y)` represent the location of the center of the predicted bounding box, and the width and height are fractions relative to the entire image size. The confidence represents the IOU between the predicted bounding box and the actual bounding box, referred to as the ground truth box. The IOU stands for Intersection Over Union and is the area of the intersection of the predicted and ground truth boxes divided by the area of the union of the same predicted and ground truth boxes.

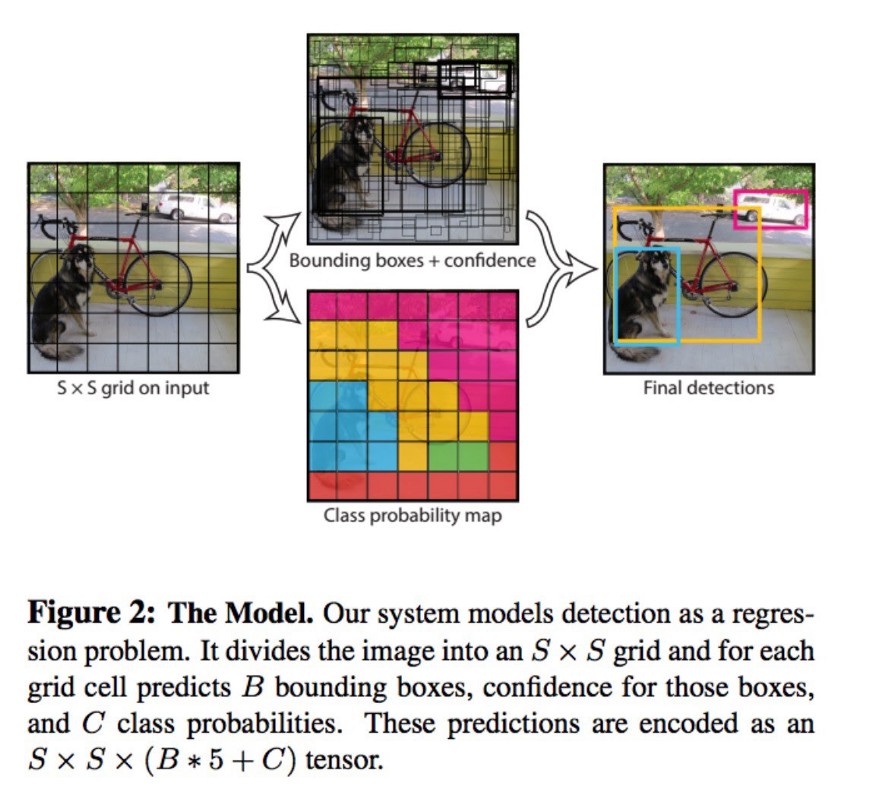


Here is an example of an IOU: the area of intersection of the ground truth and predicted box in green divided by the area of the union of the two boxes, in purple. This will be between 0 and 1, 0 if they don’t overlap at all, and 1 if they are the same box. Therefore, a higher IOU is better as it is a more accurate prediction.

In addition to outputting bounding boxes and confidence scores, each cell predicts the class of the object. This class prediction is represented by a one-hot vector length C, the number of classes in the dataset. However, it is important to note that while each cell may predict any number of bounding boxes and confidence scores for those boxes, it only predicts one class. **This is a limitation of the YOLO algorithm itself, and if there are multiple objects of different classes in one grid cell, the algorithm will fail to classify both correctly(That is why we will use yolo v2 which solves this problem using anchor boxes).** Thus, each prediction from a grid cell will be of shape C + B \* 5, where C is the number of classes and B is the number of predicted bounding boxes. B is multiplied by 5 here because it includes (x, y, w, h, confidence) for each box. Because there are S × S grid cells in each image, the overall prediction of the model is a tensor of shape S × S × (C + B ∗ 5).

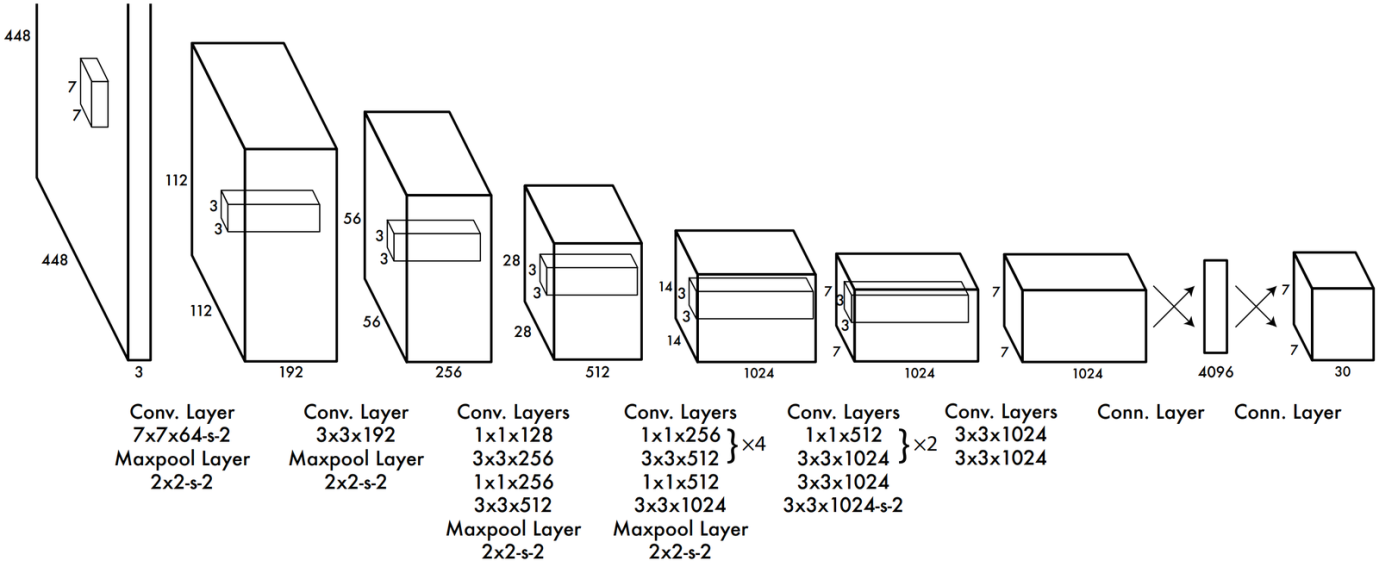


Here is an example of the output of the model when only predicting a single bounding box per cell. In this image, the dog’s true center is represented by the cyan circle labeled ‘object center’; as such, the grid cell responsible for detecting and bounding the box is the one containing the cyan dot, highlighted in dark blue. The bounding box that the cell predicts is made up of 4 elements. The red dot represents the center of the bounding box, (x, y), and the width and height are represented by the orange and yellow markers respectively. It is important to note that the model predicts the center of the bounding box with widths and heights rather than top left and bottom right corner positions. The classification is represented by a one-hot, and in this trivial example, there are 7 different classes. The 5th class is the prediction and we can see that the model is quite certain of its prediction. Keep in mind that this is merely an example to show the kind of output that is possible and so the values may not be accurate to any real values. Below is another image of all the bounding boxes and class predictions that would actually be made and their final result.



**YOLO Architecture**

The YOLO model is made up of three key components: the head, neck, and backbone. The backbone is the part of the network made up of convolutional layers to detect key features of an image and process them. The backbone is first trained on a classification dataset, such as ImageNet, and typically trained at a lower resolution than the final detection model, as detection requires finer details than classification. The neck uses the features from the convolution layers in the backbone with fully connected layers to make predictions on probabilities and bounding box coordinates. The head is the final output layer of the network which can be interchanged with other layers with the same input shape for transfer learning. As discussed earlier, the head is an S × S × (C + B ∗ 5) tensor and is 7 × 7 × 30 in the original YOLO research paper with a split size S of 7, 20 classes C, and 2 predicted bounding boxes B. These three portions of the model work together to first extract key visual features from the image then classify and bound them.



Although object detection applications have tremendously improved with latest advancements in computer vision based on deep learning, the scope of object detection is still restricted to a small set of objects, this is due to limited number of labeled datasets for detection.

The most prevailing detection dataset Pascal VOC detects 20 categories, similarly MS COCO detects 80 categories comprising thousands of to hundreds of images, while classification datasets have millions of images with hundreds of thousands of categories. This is primarily owing to the difficult task of labelling dataset for detection while user-supplied metadata such as keywords or tags associated with the image provide the job of classification effortlessly.

Furthermore, since the original YOLO model (let’s call it YOLOv1) suffers from localization errors and low recall predictions, the paper presents YOLOv2, which proposes novel and prior work-based improvements, namely SSD, to address the above constraints and further increase the speed vs accuracy trade-off.

The authors sought to make YOLO faster while performing more accurately, hence the humorous title of the paper “Better, Faster and Stronger”, thus proposing the the following improvements to achieve each of these goals :

**Adjustments for a better YOLO:**

**Batch normalization:** By adding batch normalization on all convolutional layers in YOLO, the performance improves by 2% mAP, batch norm has a regularizing effect, therefore we can remove dropout without overfitting.

**Classifying on high resolution inputs:** Yolo pretrains the classifier on ImageNet using 224x224 resolution inputs, then increases the resolution to 448x448 for detection, it requires the network time to simultaneously adjust to the new resolution input and perform the detection task. However YOLOv2 resolves this by :

Firstly train the classifier on 224x224 resolution on ImageNet.

Secondly fine tune the classifier on 448x448 resolution for 10 epochs, this make the network’s filters adjust to higher resolution inputs.

Then fine tune the resulting network on detection.

Fine tuning the classifier on higher resolution inputs increases accuracy by 4% mAP.

Convolutional layers with anchor boxes. YOLO predicts bounding box coordinates straight from fully connected layers located on top of convolutional feature extractor layers, while SSD and Faster R-CNN predict offsets to anchor boxes.

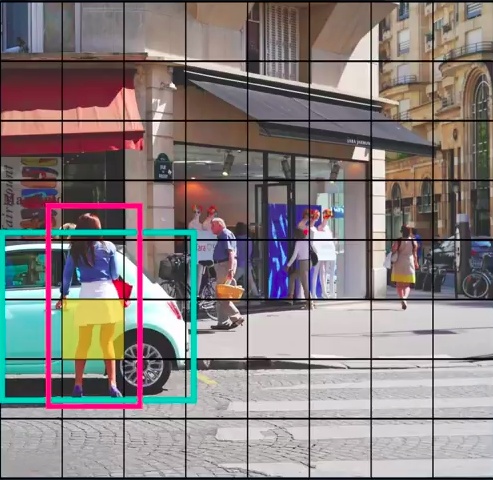
**What are anchor boxes ?**

Anchor boxes (also called default boxes) are a set of predefined box shapes selected to match ground truth bounding boxes, because most of objects in the training dataset or generally in the world (e.g. person, bicycle, etc.) have a typical height and width ratio. So when predicting bounding boxes we just adjust and refine the size of those anchor boxes to fit the objects, hence the word use of offset.

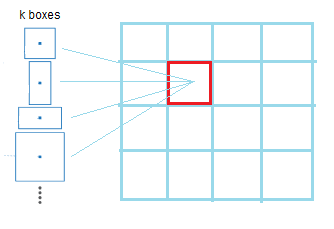
The use of anchor boxes makes the learning process tremendously easier, in addition to achieving multi-scale detection by specifying anchor boxes of varying sizes.

By using anchor boxes, YOLOv2 improved recall by 7% which means it increased the percentage of positive cases, however it decreased accuracy by a small margin.

Convolution with Anchor boxes:



YOLO v1 tries to assign object to the grid cell that contain middle of the object.In the above image,we can see that the yellow grid cell contains the middle point of both car and the girl.But since the grid cell can detect only one object the problem arise. To solve this,the authors tried to allow the grid cell detect more than one object using k bounding box(In YOLO v2).



To predict k-bounding boxes YOLO v2 uses the idea of anchor boxes.

YOLO predicts the coordinates of bounding boxes directly using fully connected layers on top of the convolutional feature extractor. In YOLO v2 all fully connected layers are removed and uses anchor boxes to predict bounding boxes.

One pooling layer is removed to increase the resolution of output

416x416 images are used to train detection network and a 13x13 feature map is obtained.ie, they are down sampled by a factor of 32.

Thus we make coordinates and confidence score(objectness prediction) prediction for each anchor boxes. Following YOLO, the objectness prediction still predicts the IOU of the ground truth and the proposed box and the class predictions predict the conditional probability of that class given that there is an object

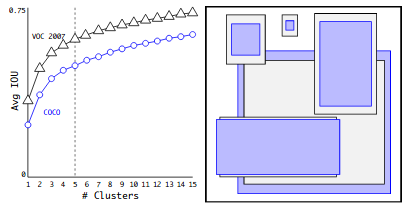
Using anchor boxes there is small decrease in accuracy. Without anchor boxes our intermediate model gets69.5mAP with a recall of 81%. With anchor boxes our model gets 69.2mAP with a recall of 88%.

**1.4. How to choose number of anchor boxes?(Dimension Clusters)**

They run k-means clustering on all bounding boxes for various values of k and plot the average IOU with the closest centroid. The important thing is that instead of using Euclidean distance they used iou between bounding box and the centroid.Using standard Euclidean distance based k-means clustering is not good enough because larger boxes generate more error than smaller boxes.

They got best results at k=5. They used the following formula to find distance between bounding box and centroid:

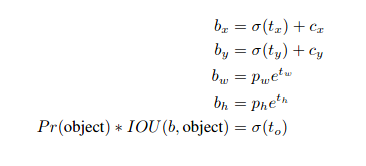
d(box,centroid) = 1−IOU(box,centroid)

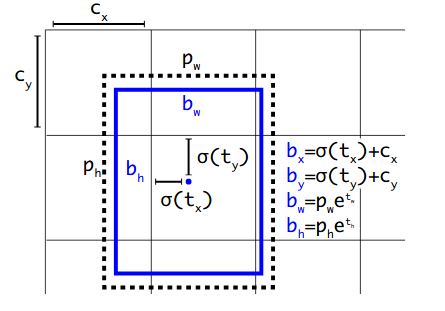


The left image shows the average iou and right image shows relative coordinates of VOC and COCO.

**Location predictions**

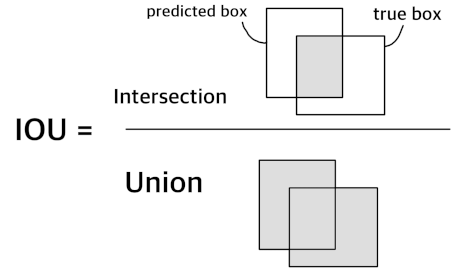
* YOLO v1 has no constrains on location predictions. This makes the model unstable at earlier iterations.
* YOLO v2 bounds the location using logistic activation σ, which makes the value fall between 0 to 1.
* The network predicts 5 bounding box for each cell. It predicts 5 coordinates for each bounding box tx,ty,tw,th and to. If the cell is offset from the top left corner of the image by (cx,cy) and the anchor box has width and height pw, ph, then the predictions correspond to:





For example if we use 2 anchor boxes on a particular grid cell, it will output two boxes(assume it as a blue one and red one). Now take case of blue box. we assign this box not only to the grid cell, but also to the anchor box(dotted box in above image) that has maximum iou with it.

**What’s IOU?**



Its full form is Intersection Over Union. This is calculated by dividing the overlapped area of a predicted box and the truth box by the whole area made by the two boxes.

This value is used as a measure of how good the prediction is.

**Loss Function composed of 3 losses:**

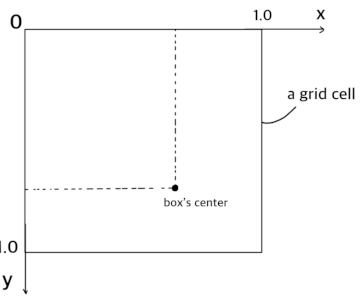
The prediction of YOLOv2 is composed of three parts:

* the four coordinates for x, y, w and h
* the probability P(obj) that an object exists in a bounding box
* the conditional probability *Cᵢ = P(the obj belongs i-th class | an obj exists in this box)*

therefore the loss is calculated for each prediction and then combined.

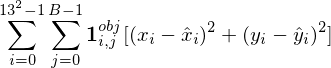
**Loss for x, y, w and h**

The coordinates of x and y lie between 0 and 1. This is because they are given by applying sigmoid on the x and y part of the output from network. This prediction is done relative to the grid cell. For example, if both x and y are 0.5, then this meas that the box’s center falls on the center of the grid cell. The reason why the center coordinates are predicted this way is just we don’t need to know the absolute coordinates if we know this grid cell position.



This image depicts one grid cell. The prediction of the center coordinates is done relative to the grid cell size. If the box’s center lies at the bottom right of the cell, then both x and y are 1.0.

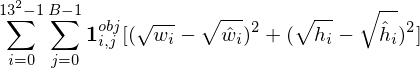
Then the loss is defined as a sum of squares of error:



The sum is over all grid cells and all bounding boxes. 1\_{i,j}^{obj} is 1.0 if the bounding box is ‘responsible’ for detecting this object, 0.0 otherwise. The term ‘responsible’ means that this bounding box has the highest IOU value among the B bounding boxes, and there is actually an object on this grid cell.

The sum is done only on the bounding boxes that are responsible for detecting objects. This is because, for bounding boxes that do not actually contain any objects, we don’t know the true center coordinates \hat{x}\_i and \hat{y}\_i. This is also the case for loss of w and h.

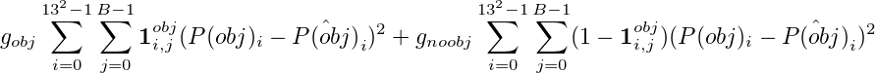
The loss function for w and h looks almost the same as the one for x and h:



Sum-squared error often becomes large for larger bounding boxes, and becomes small for smaller boxes. To address this unfairness, we use square roots of weight and height instead of using them directory.

**Loss for P(obj)**

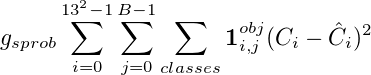
Each bounding box has the probability that the box contains an object. If the bounding box does not actually contain an object, the predicted probability should be decreased through training, and if the box actually contains an object, the predicted probability should be close to 1.0.



The first term is the loss for ‘responsible’ bounding boxes, and the second one is for the non-responsible ones. Note that g\_{obj} should be greater than g\_{noobj} according to the YOLO paper.

**Loss for Conditional Probabilities C\_i:**

We can define loss values only for responsible bounding boxes, since there are no truth labels for non-responsible bounding boxes.



g\_{sprob} is a constant.

**Background:**

**Machine Learning**

Machine learning is one of the applications of Artiﬁcial Intelligence (AI) which enables the computers to learn on their own and perform tasks without human intervention. There are numerous applications of machine learning algorithms in the ﬁeld of computer vision. With the help of machine learning, formulation of some of the most complex problems have been performed easily. Various computer programs which were previously programmed by humans, sometimes by hand, are now being programmed without any human contribution with the help of machine learning. In the recent years, due to remarkable increase in the availability of humongous sources of data and feasibility of computational resources, machine learning has become predominant with wide range of applications in our daily lives.

**Types**

* + - * Supervised Learning
      * Unsupervised Learning
      * Reinforcement Learning

#### SupervisedLearning

Supervised learning is considered to be the most elementary class of machine learning algorithms. As the name suggests, these algorithms require direct supervision. In this type of learning, the data labelled/annotated by humans is spoon-fed to the algorithm. This data contains the classes and locations of the objects of interest. Eventually, the algorithm learns from the annotated data and predicts the annotations of the new data previously not known to the algorithm, after the completion of training process . Some of the popularly utilized supervised learning algorithms are:

* Neural Networks
* Decision Trees
* Random Forest
* K-Nearest Neighbors
* Linear Regression
* Logistic Regression
* Support Vector Machines

#### Unsupervised Learning

In the unsupervised learning, the algorithm tries to learn and identify useful properties of the classes from the given annotated data, without the help or intervention of a human . Apriori algorithm, K-means clustering, etc. are some of the common unsupervised learning algorithms.

#### Reinforcement Learning

In this type of learning, the machine is allowed to train itself continually using trial and error. As a result, the machine learns from past experience and attempts to capture the best knowledge possible to predict accurately. Markov Decision Process, Q-learning, Temporal diﬀerence, etc. are some of the examples of reinforcement learning.

**Artiﬁcial Neural Networks**

Artiﬁcial neural networks are a popular type of supervised learning model.A special case of a neural network called the convolutional neural network (CNN) is the primary focus of this thesis. The name ‘Artiﬁcial Neural Networks’ was given tothis model because they were developed to imitate the neural function of the human brain.An artiﬁcial neural network consists of a set of neurons connected to each other and are grouped into layers to replicate the neural function of our brain.

Similar to the neurons in a human brain, the neurons in an artiﬁcial neural net-work function as units of calculation (see Figure 2.1).The connections between neurons are known as ‘synapses’ which are nothing but weighted values . Therefore, in a simple sense, when an input value is provided at a neuron (*x*1*, x*2,...,*xn*), it traverses the synapse, multiplying its value with the weighted value of the synapse (*w*1*, w*2, . .., *wn*) as shown in the Figure 2.2.Bias ‘*b*’ is then added to the summation of these values. This will be the output of the neuron. Since a neuron does not know its boundary, a mapping mechanism is required to map the inputs to

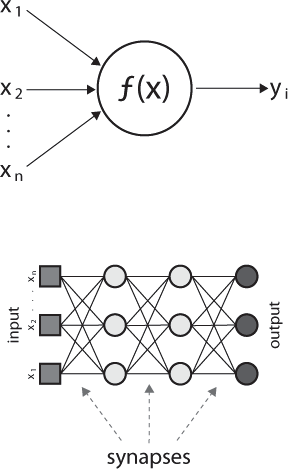


Figure2.1:A Simple Artiﬁcial Neural Network

the output , known as the ‘Activation function’. In a fully connected feed-forward multi-layer network, all the outputs of a layer of neurons is fed as an input to every neuron of the next layer. As a result, some of layers get to process the original input data, while some layers get to process the data that has been obtained from neurons from the previous layer (see Figure 2.3).Therefore, the number of weights of any neuron in the network is equal to the number of neurons in the layer previous to the layer of the neuron in question.

*n*

*y*= (*wn*∗*xn*)+*b* (2.1)

*i*=1

In the above equation, ‘*x*’ is the input value given at the neuron, ‘*w*’ is the weighted value of the synapse, ‘*n*’ is the number of neurons ,‘*b*’ is the bias and ‘*y*’ is the output of the network. Therefore, according to the equation (2.1), the value of output ‘*y*’ is equal to the summation of the product of the values of ‘*x*’ with their corresponding weights and bias ‘*b*’.

A multi-layered artiﬁcial neural network, as shown in the Figure 2.3, typically includes three types of layers: an input layer, one or more hidden layers and an output layer. The input layer usually merely passes data along without modifying it. Most of the computation happens in the hidden layers. The output layer converts the hidden layer activation to an output, such as a classiﬁcation. The outputs of each hidden layer serve as the inputs for the next hidden layer. The number of neurons in the output layer is equal to the number of classes trained for the neural network.

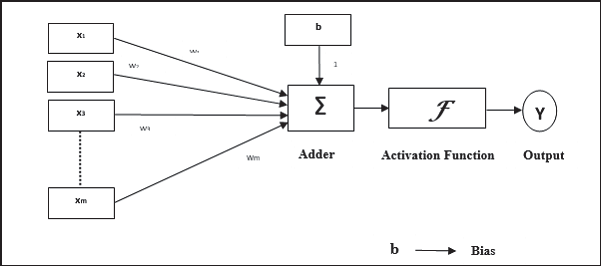


Figure2.2:Structure of a single perceptron or neuron

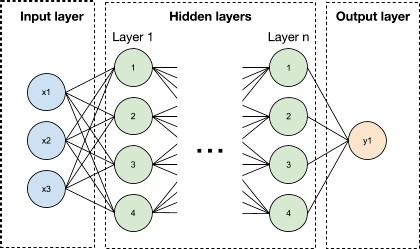


Figure2.3:Multi-layer Artiﬁcial Neural Network

**Backpropagation**

Though the artiﬁcial neural networks have shown predominant applications in various ﬁelds and aided in achieving groundbreaking innovations during recent times, the concept of neural networks is quite old. The neural networks were previously known as ‘perceptrons’ and have been in action since the 1940s. They were not popular as they are now due to the fact that they were single layered and required high computational power and data which was diﬃcult to ﬁnd during that time. They have come to limelight mainly due to the inception of a technique known as ‘Backpropagation’. The technique was ﬁrst put forth by Rumelhart et al. in the year1986. Using this technique, networks can rearrange the weights of hidden layersin case the output is diﬀerent from the expected output. The error is calculated and backpropagated to all the layers of the network to adjust the weights according to the requirement.

**ComputerVision**

Computer vision is the area of study in which computers are empowered to visualize, recognize and process what they see in a similar way as that of humans. The main aim of computer vision is to generate relevant information from image and video data in order to deduce something about the world . It can be classiﬁed as a sub-ﬁeld of artiﬁcial intelligence and machine learning. This is quite different from image processing, which involves manipulating or enhancing visual information and is not concerned about the contents of the image. Applications of computer vision include image classiﬁcation, visual detection, 3D scene reconstruction from 2Dimages, image retrieval, augmented reality, machine vision and traffic automation.

Today, machine learning is a necessary component of many computer vision algorithms . These algorithms are typically a combination of image processing and machine learning techniques. The major requirement of these algorithms is to handle large amounts of image/video data and to be able to perform computation in real-time for wide range of applications. For example, real-time detection and tracking.

**Convolutional Neural Networks**

There are various types of artiﬁcial neural networks that are considered to be very important such as Radial basis function neural network, Feed-forward neural network, Convolutional neural network, Recurrent neural network, Modular neural network, etc. Among these types of networks, the convolutional neural networks (CNNs) are elective in applications such as image/video recognition, semantic parsing, natural language processing and paraphrase detection .A convolutional neural network typically comprises of three layers – Convolutional layer, Pooling layer and Fully-connected layer.

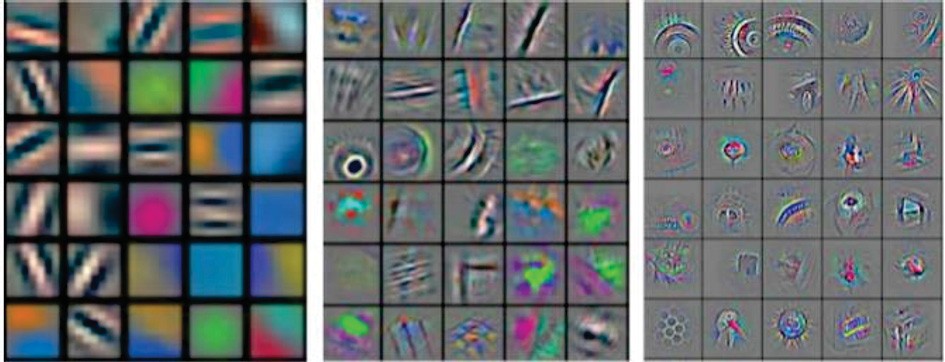


Figure2.4: Feature Filters of Front , Middle and Rear-End Layers in a CNN

**Convolutional Layer**

A convolutional neural network consists of one or more convolutional layers. These layers can either be pooled or fully connected . A convolutional layer generally executes tasks that require heavy computation. It comprises of a set of ﬁlters that have the ability to learn. Though the ﬁlters are small in size, they reach to the entire depth of the input. The dimensions of a ﬁlter are generally represented by l w d, where ‘l’ denotes the height of the length of the ﬁlter, ‘w’ denotes the width while ‘d’ denotes the depth of the feature ﬁlter which is equal to the number of color channels present.

In general, the convolution process is executed by a feature ﬁlter upon sliding on the input layer of the neural network, as a result of which a feature map is generated. The layer executing the convolution process is known as a convolutional layer. Hence, the networks that consist of convolutional layers are called as convolutional neural networks. As shown in the Figure 2.4, in the initial stages, the input layer is searched for any speciﬁc pattern by the ﬁlter. During the training of the algorithm, the ﬁlter searches for the sake of learning to recognize a pattern which eventually becomes a search to validate the existence of a speciﬁc pattern, during the testing stages. In reality, many feature ﬁlters exist, learning to recognize various patterns.

**Pooling Layer**

Pooling layers are also an important component of a convolutional neural network. The main function of a pooling layer is to decrease the number of parameters and computation present in the network by decreasing the spatial size gradually and continuously. This action is necessary to cut down the features that the ﬁlter has learnt and no longer requires the where abouts of their location. There are many beneﬁts using a pooling layer such as limiting of over-ﬁtting, which is a state that occurs when the algorithm ﬁts the data very closely by showing low bias and high variance. Though there are various types of pooling, maxpooling is one of the most popular ones in practice.

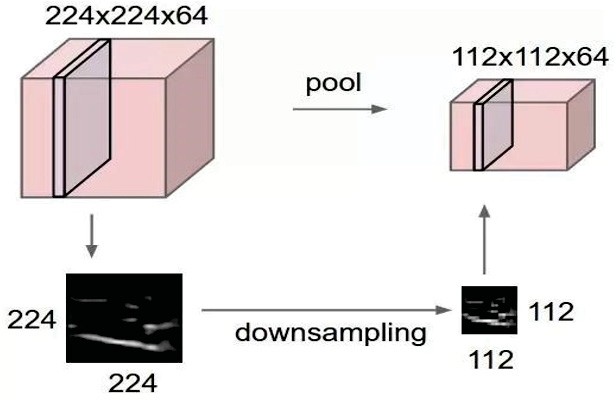


Figure2.5:PoolingLayer

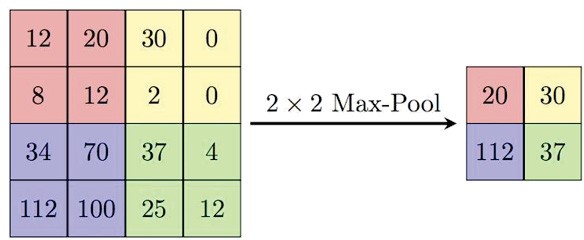


Figure2.6:Example of 2x2 Max-pooling

This type of pooling conveniently down-samples the layer while keeping the depth constant. Figure 2.5 shows the depiction of a pooling layer while Figure 2.6 provides an example of a 2x2 map pooling.

**Python :**

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Python code is concise and readable even to new developers, which is beneficial to machine and deep learning projects. Due to its simple syntax, the development of applications with Python is fast when compared to many programming languages.

**Some Libraries used :**

**Numpy:** In Python we have lists that serve the purpose of arrays, but they are slow to process.NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.

Arrays are very frequently used in data science, where speed and resources are very important.

**Matplotlib:** Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002.

One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

**Pillow:** Python Imaging Library (expansion of PIL) is the de facto image processing package for Python language. It incorporates lightweight image processing tools that aids in editing, creating and saving images. Support for Python Imaging Library got discontinued in 2011, but a project named pillow forked the original PIL project and added Python3.x support to it. Pillow was announced as a replacement for PIL for future usage. Pillow supports a large number of image file formats including BMP, PNG, JPEG, and TIFF. The library encourages adding support for newer formats in the library by creating new file decoders.

This module is not preloaded with Python.

**Tensorflow:** TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

**The Dataset:**

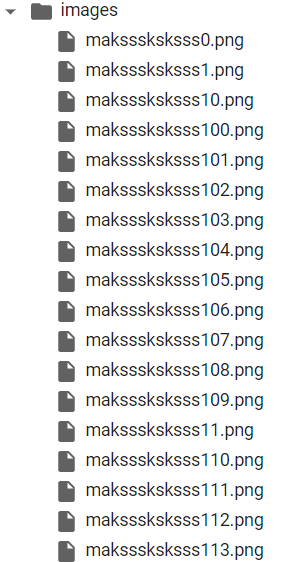
The dataset consisted of images of people wearing mask and without mask also people wearing mask incorrectly . **We had a total of 853 images with annotations for each image stored in xml files.**

**In a folder structure like:**





**Which are like this when expanded :**





**Distribution of our data(a single image can contain more than one face):**



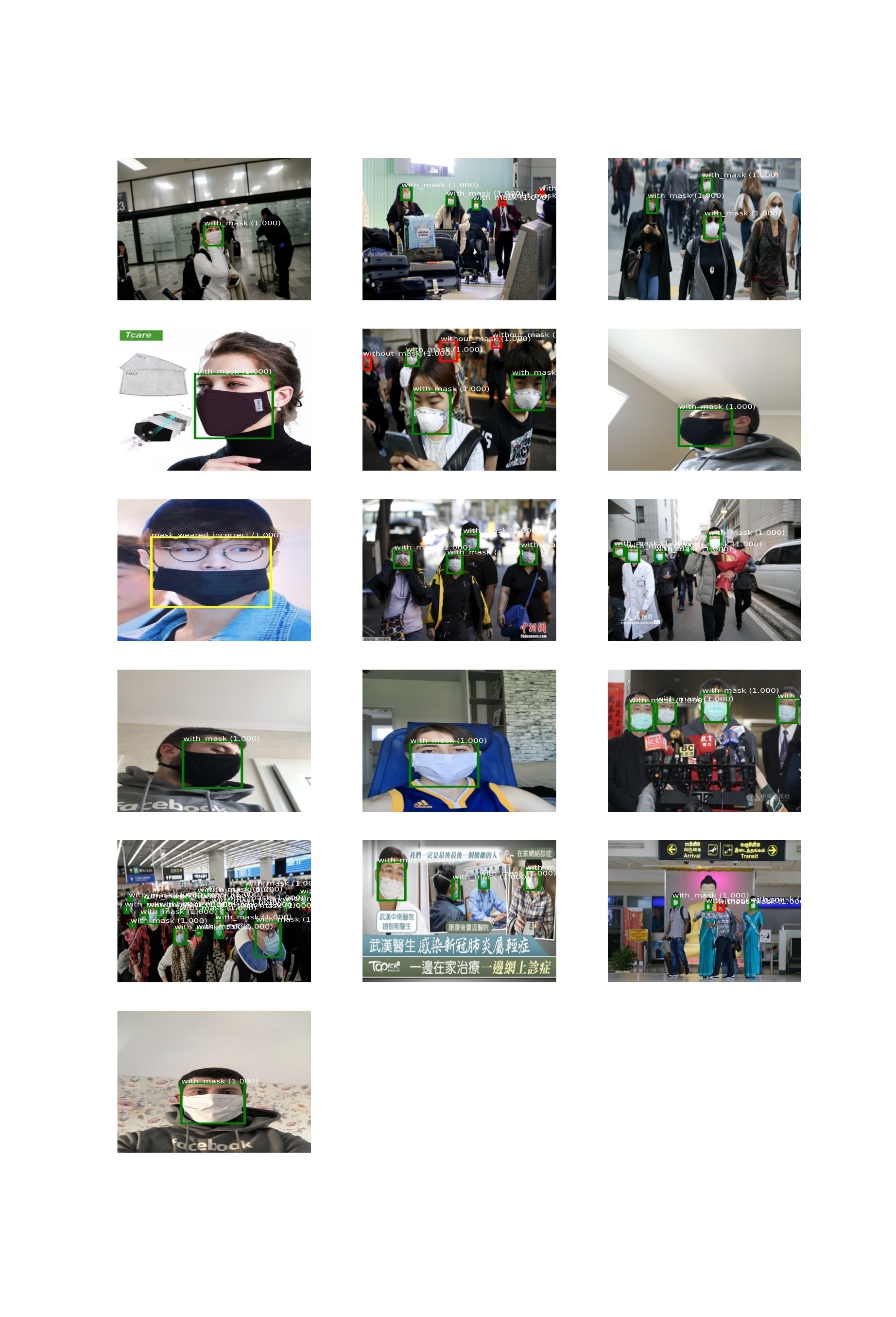
**'without\_mask':** 717 faces

**'mask\_weared\_incorrect':** 123 faces

**'with\_mask':** 3232 faces

As we can see we have a lot of faces with\_mask examples also a good amount of faces without\_mask **but we have very few examples of faces with mask\_weared\_incorrect**. This means in our dataset we have a problem of **class imbalance** with can result in our model perform badly to classify the classes having less examples (here the class “**mask\_weared\_incorrect**”).

**Lets see a few examples of our dataset(mask,without\_mask,mask\_wearing\_incorrect):**



**The annotations:**

The xml files has Class labels as well as their bounding boxes in the PASCAL VOC format.

Pascal VOC provides standardized image data sets for object detection

Pascal VOC is an XML file.

In Pascal VOC we create a file for each of the image in the dataset. In COCO we have one file each, for entire dataset for training, testing and validation.

Pascal VOC Bounding box :(xmin-top left, ymin-top left,xmax-bottom right, ymax-bottom right)

**Some of the key tags for Pascal VOC are explained below**

**Folder:**Folder that contains the images

**Filename:**Name of the physical file that exists in the folder

**Size:**Contain the size of the image in terms of width, height and depth. If the image is black and white then the depth will be 1. For color images, depth will be 3

**Object:**Contains the object details. If you have multiple annotations then the object tag with its contents is repeated. The components of the object tags are

* name
* pose
* truncated
* difficult
* bndbox

**name:**This is the name of the object that we are trying to identify

**truncated:** Indicates that the bounding box specified for the object does not correspond to the full extent of the object. For example, if an object is visible partially in the image then we set truncated to 1. If the object is fully visible then set truncated to 0

**difficult:** An object is marked as difficult when the object is considered difficult to recognize. If the object is difficult to recognize then we set difficult to 1 else set it to 0

**bounding box:**Axis-aligned rectangle specifying the extent of the object visible in the image.

For example maksssksksss0.xml(from our dataset) file looks like this :

<annotation>

<folder>images</folder>

<filename>maksssksksss0.png</filename>

<size>

<width>512</width>

<height>366</height>

<depth>3</depth>

</size>

<segmented>0</segmented>

<object>

<name>without\_mask</name>

<pose>Unspecified</pose>

<truncated>0</truncated>

<occluded>0</occluded>

<difficult>0</difficult>

<bndbox>

<xmin>79</xmin>

<ymin>105</ymin>

<xmax>109</xmax>

<ymax>142</ymax>

</bndbox>

</object>

<object>

<name>with\_mask</name>

<pose>Unspecified</pose>

<truncated>0</truncated>

<occluded>0</occluded>

<difficult>0</difficult>

<bndbox>

<xmin>185</xmin>

<ymin>100</ymin>

<xmax>226</xmax>

<ymax>144</ymax>

</bndbox>

</object>

<object>

<name>without\_mask</name>

<pose>Unspecified</pose>

<truncated>0</truncated>

<occluded>0</occluded>

<difficult>0</difficult>

<bndbox>

<xmin>325</xmin>

<ymin>90</ymin>

<xmax>360</xmax>

<ymax>141</ymax>

</bndbox>

</object>

</annotation>

**Implementation**

**Preprocessing:**

**Resizing Images and labels:**

All the images in our dataset are having different resolutions so we convert them to a fixed size which should match the input shape of our model (416,416,3).

That means we also have to scale the bounding boxes we have like:

Targetsize=416

xscale=targetsize/width

yscale=targetsize/ height

scaled\_x\_min=xscale\* (x\_min)

scaled\_y\_min=yscale\* (y\_min)

scaled\_x\_max=xscale\* (x\_max)

scaled\_y\_max=yscale\* (y\_max)

**Normalization:**

Normalization is useful in neural networks it can speed up the gradient decent converge faster.

Our images have array have values in a range from 0 to 255 which is big. So we normalize them to a range between 0 to 1.

Normalized\_image\_array=Image\_array/255

**Data Loading:**

We could have used to ways two load the data :

1. Load all the data at once to the ram and then do processing on it. This method is fast as we just have to once load the data and then we can train on it again and again.But this need a large amount of resource like large vram(video memory) or ram .Also Until the complete dataset is not loaded we can’t start training.
2. When we have limited resource we can load a batch of data and do training on it then remove it from memory and load the next batch of data and so on . Using this method we can train on any amount of data but this takes more time .

Here we used the second method as we had limited resources .

**Model:**

So I implemented Yolo v2 like deep convolutional network who’s architecture is like this:

Model: "model"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param # Connected to

==================================================================================================

input\_1 (InputLayer) [(None, 416, 416, 3 0 []

)]

conv\_1 (Conv2D) (None, 416, 416, 32 864 ['input\_1[0][0]']

)

norm\_1 (BatchNormalization) (None, 416, 416, 32 128 ['conv\_1[0][0]']

)

leaky\_re\_lu (LeakyReLU) (None, 416, 416, 32 0 ['norm\_1[0][0]']

)

max\_pooling2d (MaxPooling2D) (None, 208, 208, 32 0 ['leaky\_re\_lu[0][0]']

)

conv\_2 (Conv2D) (None, 208, 208, 64 18432 ['max\_pooling2d[0][0]']

)

norm\_2 (BatchNormalization) (None, 208, 208, 64 256 ['conv\_2[0][0]']

)

leaky\_re\_lu\_1 (LeakyReLU) (None, 208, 208, 64 0 ['norm\_2[0][0]']

)

max\_pooling2d\_1 (MaxPooling2D) (None, 104, 104, 64 0 ['leaky\_re\_lu\_1[0][0]']

)

conv\_3 (Conv2D) (None, 104, 104, 12 73728 ['max\_pooling2d\_1[0][0]']

8)

norm\_3 (BatchNormalization) (None, 104, 104, 12 512 ['conv\_3[0][0]']

8)

leaky\_re\_lu\_2 (LeakyReLU) (None, 104, 104, 12 0 ['norm\_3[0][0]']

8)

conv\_4 (Conv2D) (None, 104, 104, 64 8192 ['leaky\_re\_lu\_2[0][0]']

)

norm\_4 (BatchNormalization) (None, 104, 104, 64 256 ['conv\_4[0][0]']

)

leaky\_re\_lu\_3 (LeakyReLU) (None, 104, 104, 64 0 ['norm\_4[0][0]']

)

conv\_5 (Conv2D) (None, 104, 104, 12 73728 ['leaky\_re\_lu\_3[0][0]']

8)

norm\_5 (BatchNormalization) (None, 104, 104, 12 512 ['conv\_5[0][0]']

8)

leaky\_re\_lu\_4 (LeakyReLU) (None, 104, 104, 12 0 ['norm\_5[0][0]']

8)

max\_pooling2d\_2 (MaxPooling2D) (None, 52, 52, 128) 0 ['leaky\_re\_lu\_4[0][0]']

conv\_6 (Conv2D) (None, 52, 52, 256) 294912 ['max\_pooling2d\_2[0][0]']

norm\_6 (BatchNormalization) (None, 52, 52, 256) 1024 ['conv\_6[0][0]']

leaky\_re\_lu\_5 (LeakyReLU) (None, 52, 52, 256) 0 ['norm\_6[0][0]']

conv\_7 (Conv2D) (None, 52, 52, 128) 32768 ['leaky\_re\_lu\_5[0][0]']

norm\_7 (BatchNormalization) (None, 52, 52, 128) 512 ['conv\_7[0][0]']

leaky\_re\_lu\_6 (LeakyReLU) (None, 52, 52, 128) 0 ['norm\_7[0][0]']

conv\_8 (Conv2D) (None, 52, 52, 256) 294912 ['leaky\_re\_lu\_6[0][0]']

norm\_8 (BatchNormalization) (None, 52, 52, 256) 1024 ['conv\_8[0][0]']

leaky\_re\_lu\_7 (LeakyReLU) (None, 52, 52, 256) 0 ['norm\_8[0][0]']

max\_pooling2d\_3 (MaxPooling2D) (None, 26, 26, 256) 0 ['leaky\_re\_lu\_7[0][0]']

conv\_9 (Conv2D) (None, 26, 26, 512) 1179648 ['max\_pooling2d\_3[0][0]']

norm\_9 (BatchNormalization) (None, 26, 26, 512) 2048 ['conv\_9[0][0]']

leaky\_re\_lu\_8 (LeakyReLU) (None, 26, 26, 512) 0 ['norm\_9[0][0]']

conv\_10 (Conv2D) (None, 26, 26, 256) 131072 ['leaky\_re\_lu\_8[0][0]']

norm\_10 (BatchNormalization) (None, 26, 26, 256) 1024 ['conv\_10[0][0]']

leaky\_re\_lu\_9 (LeakyReLU) (None, 26, 26, 256) 0 ['norm\_10[0][0]']

conv\_11 (Conv2D) (None, 26, 26, 512) 1179648 ['leaky\_re\_lu\_9[0][0]']

norm\_11 (BatchNormalization) (None, 26, 26, 512) 2048 ['conv\_11[0][0]']

leaky\_re\_lu\_10 (LeakyReLU) (None, 26, 26, 512) 0 ['norm\_11[0][0]']

conv\_12 (Conv2D) (None, 26, 26, 256) 131072 ['leaky\_re\_lu\_10[0][0]']

norm\_12 (BatchNormalization) (None, 26, 26, 256) 1024 ['conv\_12[0][0]']

leaky\_re\_lu\_11 (LeakyReLU) (None, 26, 26, 256) 0 ['norm\_12[0][0]']

conv\_13 (Conv2D) (None, 26, 26, 512) 1179648 ['leaky\_re\_lu\_11[0][0]']

norm\_13 (BatchNormalization) (None, 26, 26, 512) 2048 ['conv\_13[0][0]']

leaky\_re\_lu\_12 (LeakyReLU) (None, 26, 26, 512) 0 ['norm\_13[0][0]']

max\_pooling2d\_4 (MaxPooling2D) (None, 13, 13, 512) 0 ['leaky\_re\_lu\_12[0][0]']

conv\_14 (Conv2D) (None, 13, 13, 1024 4718592 ['max\_pooling2d\_4[0][0]']

)

norm\_14 (BatchNormalization) (None, 13, 13, 1024 4096 ['conv\_14[0][0]']

)

leaky\_re\_lu\_13 (LeakyReLU) (None, 13, 13, 1024 0 ['norm\_14[0][0]']

)

conv\_15 (Conv2D) (None, 13, 13, 512) 524288 ['leaky\_re\_lu\_13[0][0]']

norm\_15 (BatchNormalization) (None, 13, 13, 512) 2048 ['conv\_15[0][0]']

leaky\_re\_lu\_14 (LeakyReLU) (None, 13, 13, 512) 0 ['norm\_15[0][0]']

conv\_16 (Conv2D) (None, 13, 13, 1024 4718592 ['leaky\_re\_lu\_14[0][0]']

)

norm\_16 (BatchNormalization) (None, 13, 13, 1024 4096 ['conv\_16[0][0]']

)

leaky\_re\_lu\_15 (LeakyReLU) (None, 13, 13, 1024 0 ['norm\_16[0][0]']

)

conv\_17 (Conv2D) (None, 13, 13, 512) 524288 ['leaky\_re\_lu\_15[0][0]']

norm\_17 (BatchNormalization) (None, 13, 13, 512) 2048 ['conv\_17[0][0]']

leaky\_re\_lu\_16 (LeakyReLU) (None, 13, 13, 512) 0 ['norm\_17[0][0]']

conv\_18 (Conv2D) (None, 13, 13, 1024 4718592 ['leaky\_re\_lu\_16[0][0]']

)

norm\_18 (BatchNormalization) (None, 13, 13, 1024 4096 ['conv\_18[0][0]']

)

leaky\_re\_lu\_17 (LeakyReLU) (None, 13, 13, 1024 0 ['norm\_18[0][0]']

)

conv\_19 (Conv2D) (None, 13, 13, 1024 9437184 ['leaky\_re\_lu\_17[0][0]']

)

norm\_19 (BatchNormalization) (None, 13, 13, 1024 4096 ['conv\_19[0][0]']

)

conv\_21 (Conv2D) (None, 26, 26, 64) 32768 ['leaky\_re\_lu\_12[0][0]']

leaky\_re\_lu\_18 (LeakyReLU) (None, 13, 13, 1024 0 ['norm\_19[0][0]']

)

norm\_21 (BatchNormalization) (None, 26, 26, 64) 256 ['conv\_21[0][0]']

conv\_20 (Conv2D) (None, 13, 13, 1024 9437184 ['leaky\_re\_lu\_18[0][0]']

)

leaky\_re\_lu\_20 (LeakyReLU) (None, 26, 26, 64) 0 ['norm\_21[0][0]']

norm\_20 (BatchNormalization) (None, 13, 13, 1024 4096 ['conv\_20[0][0]']

)

lambda (Lambda) (None, 13, 13, 256) 0 ['leaky\_re\_lu\_20[0][0]']

leaky\_re\_lu\_19 (LeakyReLU) (None, 13, 13, 1024 0 ['norm\_20[0][0]']

)

concatenate (Concatenate) (None, 13, 13, 1280 0 ['lambda[0][0]',

) 'leaky\_re\_lu\_19[0][0]']

conv\_22 (Conv2D) (None, 13, 13, 1024 11796480 ['concatenate[0][0]']

)

norm\_22 (BatchNormalization) (None, 13, 13, 1024 4096 ['conv\_22[0][0]']

)

leaky\_re\_lu\_21 (LeakyReLU) (None, 13, 13, 1024 0 ['norm\_22[0][0]']

)

conv\_23 (Conv2D) (None, 13, 13, 32) 32800 ['leaky\_re\_lu\_21[0][0]']

reshape (Reshape) (None, 13, 13, 4, 8 0 ['conv\_23[0][0]']

)

==================================================================================================

Total params: 50,580,736

Trainable params: 50,560,064

Non-trainable params: 20,672

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

This is a 23 layers Deep Convolutional Neural Network.It takes a input tensor of shape [n,416,416,3] and outputs a tensor(matrix) of shape [n,13,13,4,8] where n is the number of images to be processed at a time .

In the Output : [n,13,13,4,8]

Corresponds to : [b\_size,height,width,anchor boxes,(xywh+pc+classes)]

**B\_size:** it is the batch size .

**Height:** the height of the grid

**Width:** the width of the grid

**Anchor boxes:** we have used 4 anchor boxes

**xywh:** center point’s x and y coordinates(relative to the grid cell it corresponds to) of the face detected(between 0 and 1) and width and height of the face (relative to the grid could be more than 1).

**pc:**probability score ranging from 0 to 1 corresponding to if a object of present in a grid cell.

**Classes:** 3 classes mask , without mask , mask worn incorrectly.

After creating the model we loaded our model with **yolo v2 pre-trained weights** to do **transfer learning**.

**Pre-trained Model:** A pre-trained model is a saved network that was previously trained on a large dataset, typically on a large-scale image-classification task. You either use the pre-trained model as is or use transfer learning to customize this model to a given task.

**Transfer Learning:** The intuition behind transfer learning for image classification is that if a model is trained on a large and general enough dataset, this model will effectively serve as a generic model of the visual world. You can then take advantage of these learned feature maps without having to start from scratch by training a large model on a large dataset.

**Training:**

We first split our dataset in two sets. We had a total of 853 images we split them as :

Training set: 707 Images

Validation set: 146 Images

That is a 83% , 17% train , val split.

**HyperParameter:** HyperParameter are some parameters which affects the performance of our model we tweak them so we get the best accuracy.

**Some HyperParameter :**

* We used a **batch size of 16** .
* A learning rate of 0.001 .
* Decay of 1e -20
* We used **Adam** as our optimizer.
* 416x416 as the input image size

**A epoch** means we have once passed through the whole dataset.

We train our model for 20 epochs .

**Train Logs:**

Epoch 1/20

44/44 [==============================] - 144s 3s/step - loss: 8.6312 - Category\_acc: 78.6476 - val\_loss: 27580.2930 - val\_Category\_acc: 78.7532

Epoch 2/20

44/44 [==============================] - 135s 3s/step - loss: 4.6952 - Category\_acc: 84.7987 - val\_loss: 119.5311 - val\_Category\_acc: 83.3243

Epoch 3/20

44/44 [==============================] - 134s 3s/step - loss: 3.4679 - Category\_acc: 92.3135 - val\_loss: 21.4074 - val\_Category\_acc: 85.4701

Epoch 4/20

44/44 [==============================] - 133s 3s/step - loss: 2.8081 - Category\_acc: 94.5822 - val\_loss: 4.2748 - val\_Category\_acc: 86.0555

Epoch 5/20

44/44 [==============================] - 134s 3s/step - loss: 2.3745 - Category\_acc: 96.0089 - val\_loss: 3.6333 - val\_Category\_acc: 89.5511

Epoch 6/20

44/44 [==============================] - 135s 3s/step - loss: 2.2484 - Category\_acc: 97.6640 - val\_loss: 3.6914 - val\_Category\_acc: 89.9082

Epoch 7/20

44/44 [==============================] - 135s 3s/step - loss: 2.0944 - Category\_acc: 98.4698 - val\_loss: 3.4166 - val\_Category\_acc: 90.5126

Epoch 8/20

44/44 [==============================] - 134s 3s/step - loss: 1.9228 - Category\_acc: 99.0206 - val\_loss: 3.9315 - val\_Category\_acc: 90.0042

Epoch 9/20

44/44 [==============================] - 133s 3s/step - loss: 1.9054 - Category\_acc: 99.3677 - val\_loss: 3.3267 - val\_Category\_acc: 90.6230

Epoch 10/20

44/44 [==============================] - 133s 3s/step - loss: 1.5319 - Category\_acc: 99.8564 - val\_loss: 3.3183 - val\_Category\_acc: 90.9218

Epoch 11/20

44/44 [==============================] - 134s 3s/step - loss: 1.4947 - Category\_acc: 99.9567 - val\_loss: 3.2640 - val\_Category\_acc: 90.8106

Epoch 12/20

44/44 [==============================] - 133s 3s/step - loss: 1.3789 - Category\_acc: 99.9790 - val\_loss: 3.1627 - val\_Category\_acc: 90.6616

Epoch 13/20

44/44 [==============================] - 134s 3s/step - loss: 1.3980 - Category\_acc: 99.9773 - val\_loss: 3.3149 - val\_Category\_acc: 91.0661

Epoch 14/20

44/44 [==============================] - 136s 3s/step - loss: 1.3243 - Category\_acc: 99.9151 - val\_loss: 3.4074 - val\_Category\_acc: 90.3605

Epoch 15/20

44/44 [==============================] - 134s 3s/step - loss: 1.2993 - Category\_acc: 99.9026 - val\_loss: 4.3849 - val\_Category\_acc: 87.9824

Epoch 16/20

44/44 [==============================] - 134s 3s/step - loss: 1.3504 - Category\_acc: 99.9296 - val\_loss: 3.3734 - val\_Category\_acc: 89.3321

Epoch 17/20

44/44 [==============================] - 134s 3s/step - loss: 1.3367 - Category\_acc: 99.9615 -val\_loss: 4.1475 - val\_Category\_acc: 86.7306

Epoch 18/20

44/44 [==============================] - 136s 3s/step - loss: 1.3170 - Category\_acc: 99.9666 - val\_loss: 3.8315 - val\_Category\_acc: 87.1416

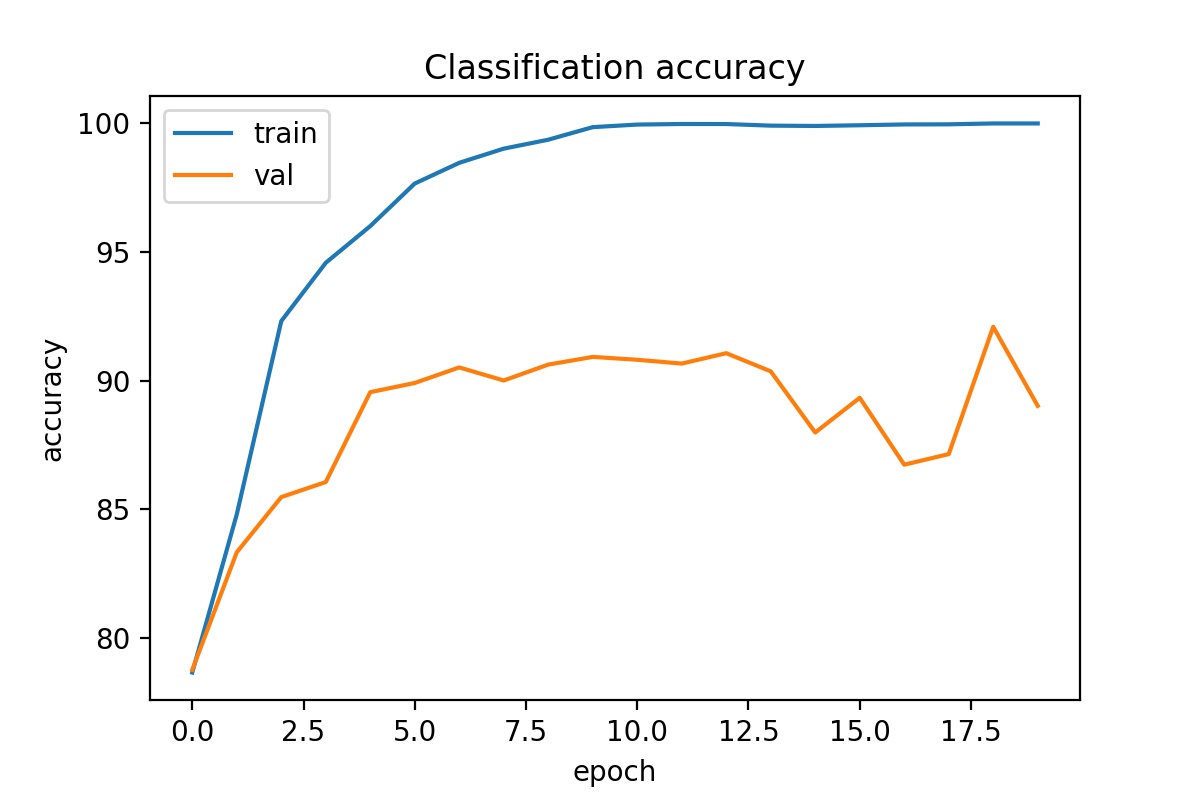
Epoch 19/20

44/44 [==============================] - 135s 3s/step - loss: 1.2818 - Category\_acc: 100.0000 - val\_loss: 3.2166 - val\_Category\_acc: 92.0925

Epoch 20/20

44/44 [==============================] - 135s 3s/step - loss: 1.1973 - Category\_acc: 100.0000 - val\_loss: 3.3556 - val\_Category\_acc: 89.0169

**Categorical Accuracy :**



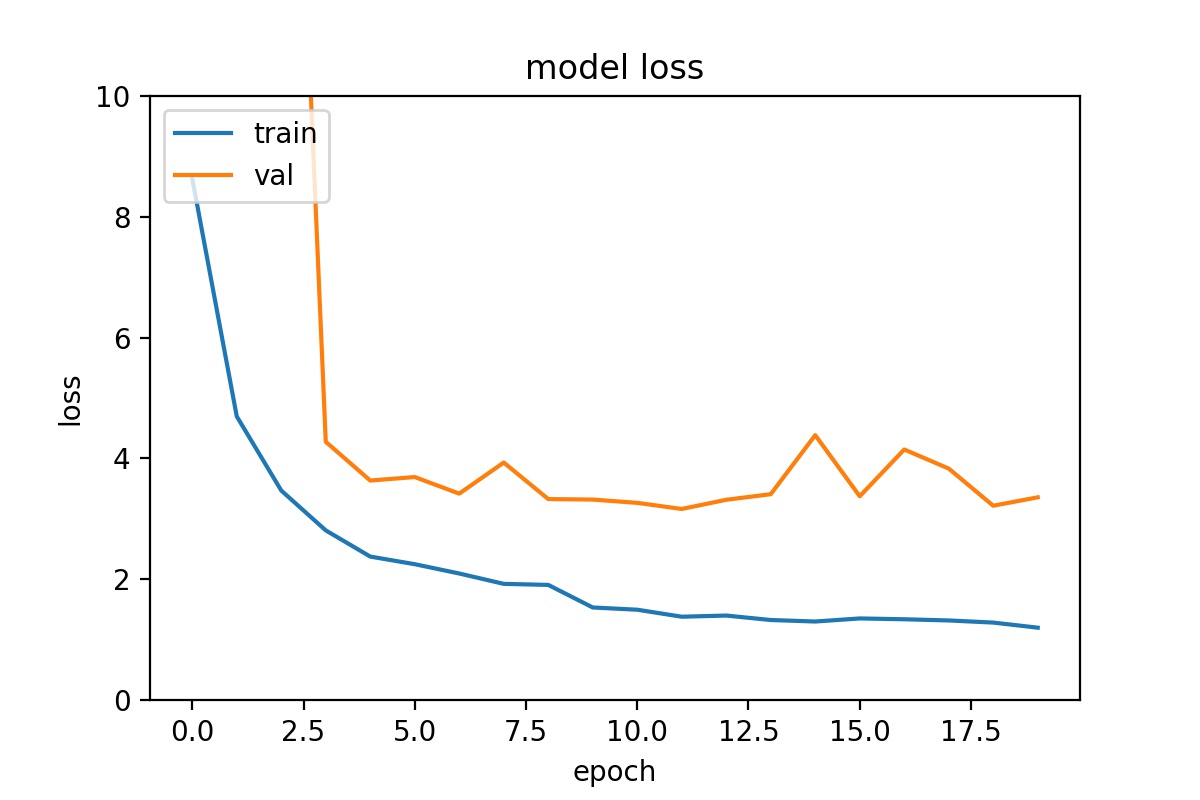
this is the plot of classification accuracy vs epochs

**Overfitting :**When a model fits training data (the samples it seen) too well but does worse on unseen examples .

**Underfitting:**When a model doesn’t do well on either the unseen data nor on the training data.

As we can see the accuracy on training data got a lot better than on validation data in our case.This condition is called **overfitting**.

**Loss :**



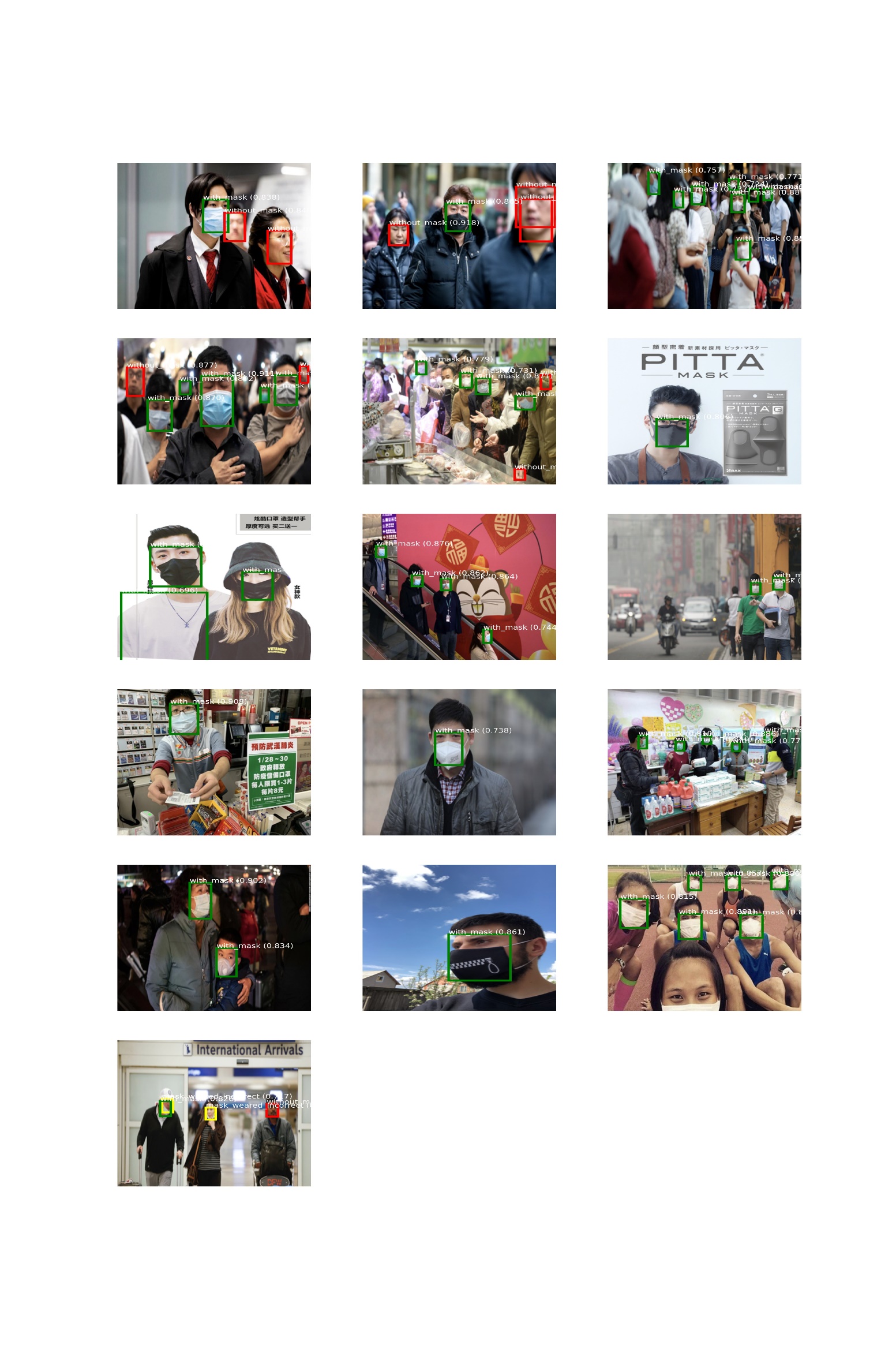
this is the plot of classification accuracy vs epochs

Loss is the amount of error a model makes while predicting .

Typically, with neural networks, we seek to minimize the error. As such, the objective function is often referred to as a cost function or a loss function and the value calculated by the loss function is referred to as simply “loss.”

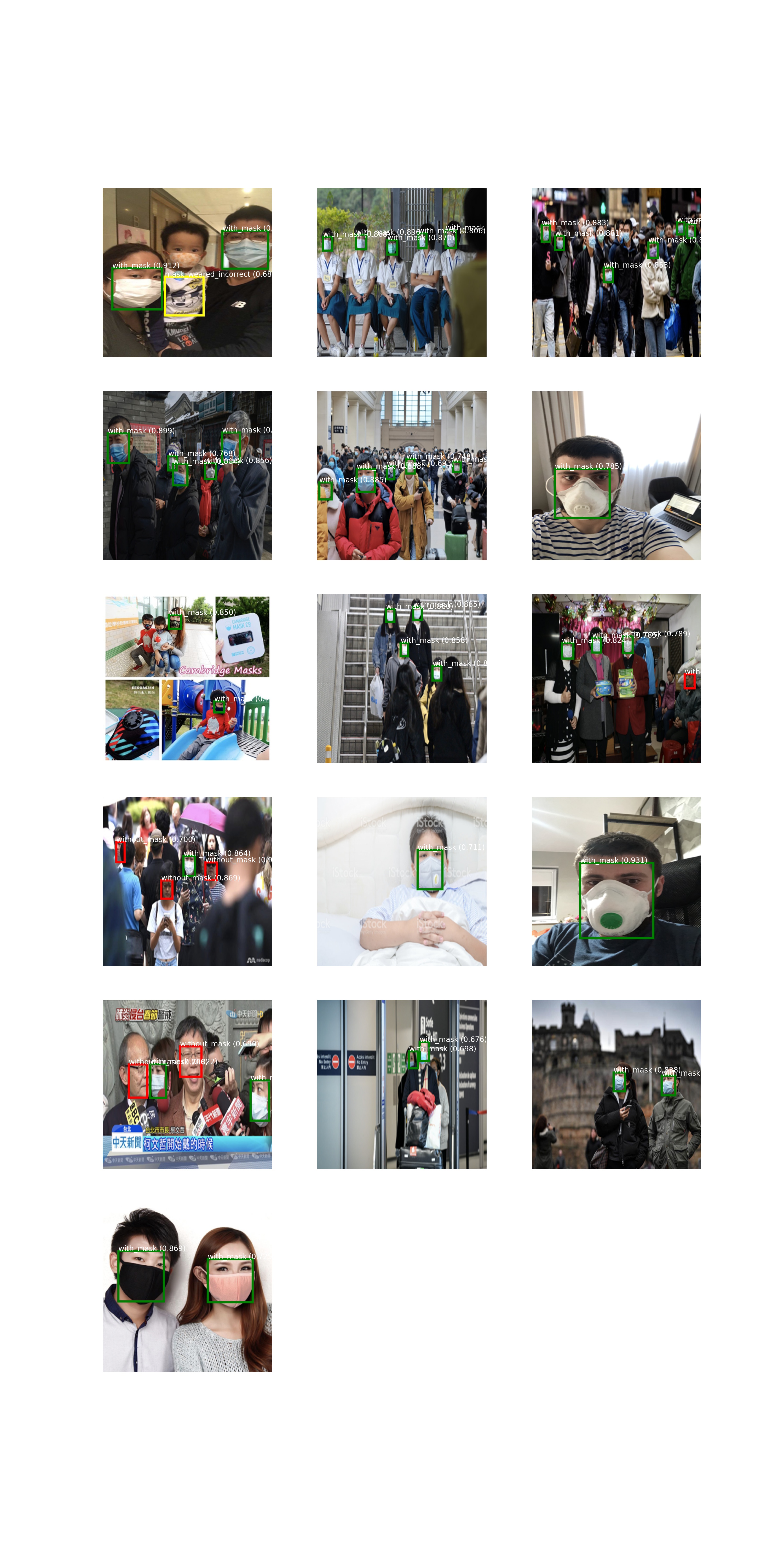
**Lower the loss the better it is.**

**Results:**

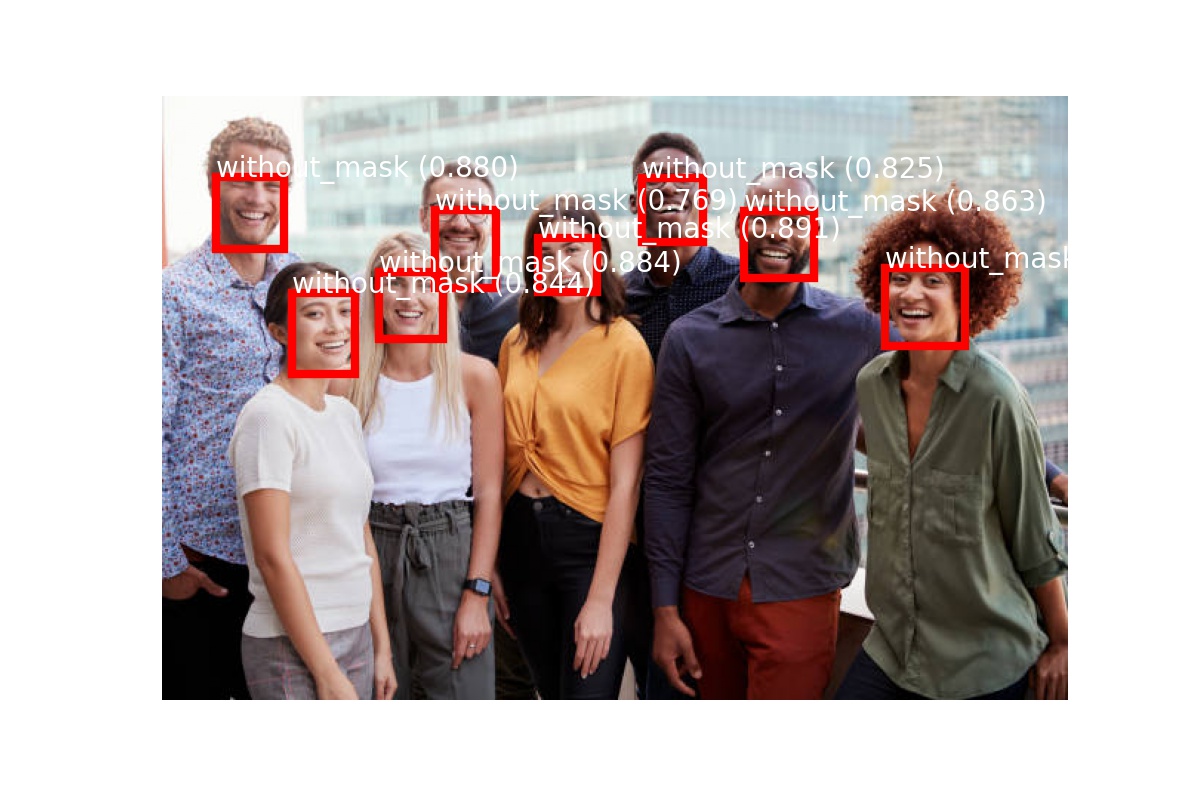
****

Predictions on training data

Predictions on Validation data:



**Predictions on data which doesn’t belongs to the dataset:**



**Future Work :**

1. As we are suffering from overfitting we can do things to avoid it and get better perform like getting more training data so our model generalize better,we could use regularization techniques like using dropouts, l1 or l2 regularization.
2. We saw that dataset had a class imbalance problem , which caused the model to get confused with mask\_wearing\_incorrectly class, we could to something to solve this problem like , getting more examples of mask\_wearing\_incorrectly class.
3. We noticed our model is unable to find blurry and small faces , we could use data Augmentation to make our model see more of such examples, Data Augmentation also helps with the overfitting problem.
4. Our model current takes images of a fixed size(416,416,3) we can change this to take input of any size which could enable us to do multiscale learning.And While processing a bigger image we can even detect the small faces as we will have more data than a low resolution image.