# Custom machine learning node for face mask detection on Node-RED

Final project for the course of Internet of Things

Avi Alessandro
*Department of Industrial Engineering*
*University of Trento*
Trento, Italy
alessandro.avi@studenti.unitn.it

***Abstract*—The Internet Of Things (IoT) is a technology that allows the interaction of multiple devices creating better communication and the creation of large volumes of data. The vast amount of information generated can be exploited by trained ML networks with the purpose of creating predictors or models for learning data patterns and behaviours. The usage of machine learning finds the perfect application in the world of IoT especially for the great amount of data generated in time. Here is presented the application of ML in Node-RED, a well known IoT environment commonly used for its ease of use and fast prototyping capabilities. In this application ML classification is applied on images taken from a webcam with the aim of detecting the presence of masks on the user in frame.**

***Index Terms*—datasets, tensorflow, node-red, neural networks, classification, IOT, CNN**

## I. Introduction

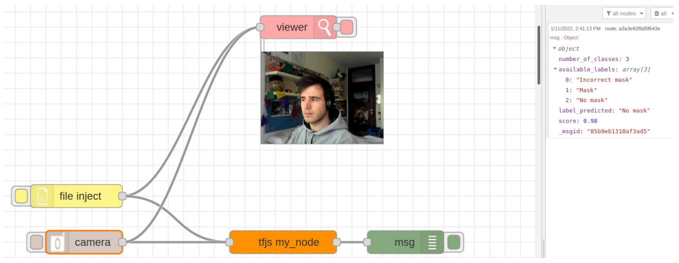
Internet of Things is an important technology that allows for the connection of devices allowing better communication and smarter management. If applied correctly, IoT can improve the daily experience of the common user in several different applications scenarios, starting from the simple domotics to industrial applications or smart cities. The ability to connect lots of different sensors, actuators and devices in the same environment allowing them to share data is a great advantage for performing data elaboration, that if paired with good management can also bring to smart applications and improve accuracy, reduce energy consumption and generally bring to an improvement of efficiency of the system. Another technology that can be added to IoT networks, allowing for enhanced data elaboration is machine learning. This tool is a great way for performing regression, which allows models to learn data patterns, or for performing predictions about the future evolution of the same data.
A well known framework used in IoT for creating chains of devices and services from ground up is Node-RED. Node-RED is a visual programming tool based on flows that allows the connection of services and data streams coming from several embedded devices. The main role of Node-RED is to behave as a central place where all the data is received and processed in a smart way and later made available for storage, applications or presentations services. Node-RED is a widely used technology that is easy to use and allows for fast prototyping of IoT chains, starting from the embedded system that generated the data to the display using services that are easy to connect.
One interesting application in the world of smart agriculture can be found in [1]. Here Node-RED is used as a data manager for the creation of an automated indoor agriculture process for lettuce cultivation. The aim is to collect data from different sensors and continuously control if the values are in the optimal zone, if not actions are taken on the cultivation with some actuators (humidifiers, light intensity, notifications to user).
Another application, this time in the field of smart cities, is explained in [2]. Here Node-RED is used for smart waste management. In this paper the application aim to create a smart IoT infrastructure consisting of smart dustbins distributed around the city. The system is able to continuously monitor the state of bins though a Node-RED dashboard which not only shows to the user the history of the waste flow in the city but also stores all the data. The information stored can then be used in an extension of the project with some algorithms for predicting the state of the waste flow in the city.
Another common application for IoT and Node-RED is domotics. In this case the paper [3] shows how this technology can be easily applied for the automation of a home, simply with the usage of a mobile device and the well known communication protocol MQTT. In this paper the goal is to create a mobile app equipped with button and switches, that uses MQTT for storing data in a database and automating several appliances in a house like a Bluetooth speaker.

The report is organized as follows; Section 2 presents a brief introduction of the Node-RED tool and the custom node created. Section 3 gives an overview on the creation and training of the ML model. Section 4 provides additional information regarding the usage and deployment of the node in the Node-RED environment, together with some comment about the results. Finally, Section 5 gives conclusion and future work.

## II. Project setup

The project presented here is the creation of a custom node to be deployed in a Node-RED environment capable of performing machine learning inference. The aim of the project is to have a node that performs image classification, specifically trained to be a surgical mask detector. The project is inspired by an already existing application done by Paul Van Eck from IBM, that also published all the workflow in a compact tutorial on their website [4]. In their application the custom node was created for performing object detection, while in this report the goal is to be able to perform just image classification in the Node-RED environment.

The node created in this project exploits the usage of the Tensorflow library. In particular the most important package on which the entire node depends is @tensorflow/tfjs−node. The usage of this tool allows to reduce the amount of code and work to do and cuts down the project in two parts. The training of a ML classification model and the creation of the inference node, which is basically composed of four functions that are able to load a model, preprocess an image, perform inference and post process the prediction. The classification model needs to be created and trained before the installation of the node and it can be easily loaded in the node simply with a link to a web page containing the model itself. For this project the entire code, the notebook for the ML trainings, the trained models, together with a brief tutorial for the repetition of the project can be found in the GitHub repository [5].



```
1/11/2022, 2:41:13 PM   node: a3a3e82f6d5f643e
msg : Object
▾ object
  number_of_classes: 3
 ▾ available_labels: array[3]
    0: "Incorrect mask"
    1: "Mask"
    2: "No mask"
  label_predicted: "No mask"
  score: 0.98
  _msgid: "85b9eb1310af3ad5"
```

Fig. 1. Flow of the inference node in the Node-RED environment

## III. Creation of the model and training

As said before one of the most important steps of the project is the creation of a ML model that is capable of performing classification of images. In this section the workflow for the training set up and the collection of the dataset is explained.

### A. Dataset preparation

Since the aim of the project is to create a model that is able to classify images of people with or without masks, a dataset containing such images is required. Originally the first iterations of training were performed with a small dataset of just 60 images taken from the webcam and myself with different masks and different face orientations. This brought to models that performed well but only on the testing images that belonged to the same dataset. To solve the problem the dataset has been replaced with the combination of two dataset available on line on Kaggle.com. The dataset used for the training contains a total of 7993 images, where 3945 are labeled as ”with mask” and the remaining 4048 are labeled as ”without mask”. The availability of such a large dataset allows for a good training not only because of the vaste amount of images but also for their variations of characteristics, like colors, backgrounds, orientation of faces, region of the body captured.

After the collection of the dataset an important step has to be performed, pre processing. In order to correctly train the model on all these different types of images it’s necessary to have consistent shapes and types of files. It has been decided to resize all the images in a smaller size of 224x224 pixels. This reduces the dimension of the image that will be elaborated and defines the size of the input data that will be fed to the model. In order to perform the resizing on such a big dataset the tools available on Roboflow.com have been used, which is a well known website used for processing datasets. It is in fact possible to merge together different types of notations in a single dataset (for example the notation of bounding boxes done with different standard notations), merge together different types of image files and most importantly it can be used for dataset augmentation (flipping of images, tilt, rotations, blur, ..). In this case the tool has been used only for resizing the entire dataset in images of size 224x224 pixels.

Note that by doing this the model will be trained only on data of shape 224x224. It’s in fact required to perform the same resizing during the inference.

### B. Training setup

The model for this application is a binary classification model, meaning that the output of the ML model is just a vector of two numbers where each represents the probability of the image to belong to one class or the other. The most suitable type of neural network for performing a classification on images is for sure the CNN, convolutional neural network. This type of network is specifically created for the elaboration of image and classification of its content. The most important part of a CNN are the convolutional layers, which are layers that perform feature extraction on the three matrices containing the colors information of the original image. The operation consists in the convolution between a small matrix called filter or kerner (usually 3x3 or 5x5) that is superimposed over a small portion of the image and on where the multiplication element by element between the two matrices is perfomed and then summed together. The result of the operation is then stored in a smaller matrix. This type of operation allows the model to perform a feature extraction over the image of interest, reducing the size of the data and extracting only the most important and relevant information. Together with the usage of convolutional layer, usually, a CNN model also exploits fully connected layers, which are the most basic structure of a neural network. These are used for performing the actual classification over the flattened and elaborated image.

In this project the training and creation of the model is performed with Tensorflow on Google Colab, which is a powerful on line tool that allows any user to remotely use available GPU for training their model or perform other type of computations. Tensorflow is a well known and very powerful framework that makes the generation of neural networks and their training fast and easy to use with very little code. In this case it has been decided to create a model based on an already existing model available on the TFhub. This is called MobileNet_v2, which is a powerful CNN architecture developed specifically for performing well on mobile devices and is explained in detail in this paper [6]. The model is a state of the art CNN model that bases its enhanced performance on linear bottlenecks between layer and short cuts between botllenecks. This model compared with its previous version requires two times less operations, has higher accuracy and requires 30% less parameters. The model shines in application regarding object detection but being it particularly efficient on mobile devices it has been decided to use its structure for this application in the IoT world.

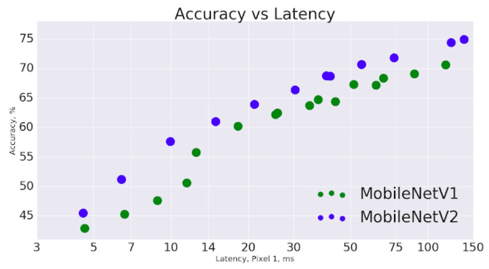


Fig. 2. MobileNetv2 perfromance compared with MobileNetv1

Another important step for the deployment of the model on Node-RED is the type of file saved. Because of the Node-RED environment and the usage of the package @tensorflow/tfjs−node it's necessary to have a model that is compatible with these frameworks. This can be done quite easily by using the model converter developed again by Tensorflow, which is installed in the Google Colab notebook. By specifying the original type of model and the desired output file, it's possible to transform the model from the usual .h5 file to a .JSON file.

## IV. Creation of the node and deployment of the model

Once the model is trained, tested and ready to use it can be deployed in the node and be used for inference. For doing so it is necessary to code the behaviour and aspect of the node and later install it in the environment. The flow relies on the usage of some external nodes (like the webcam node or the image viewer node) that need to be installed inside Node-RED. Also the code of the node is based on some Tensorflow packages. In order to have a working flow it's necessary to install the following packages:

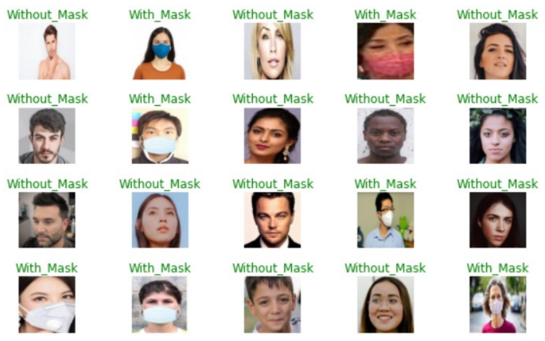
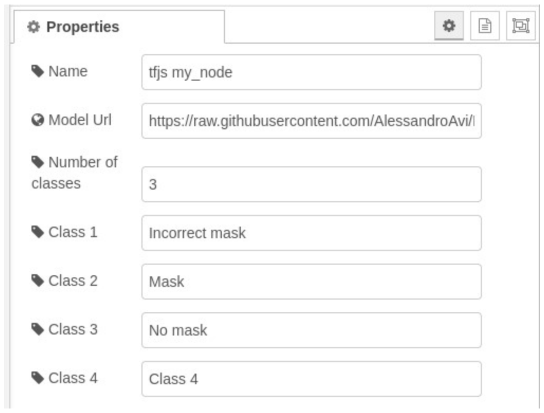


Fig. 3. Testing grid of predictions. Green text means the rpediction is correct.

- @tensorflow/tfjs−node
- node−red−contrib−tf−module
- node−red−contrib−tf−function
- node−red−contrib−browser−utils
- node−red−contrib−image−tools

Where the first one should be installed manually in the main node red folder, while the other can be installed directly from the package manager in Node-RED.
The flow of the machine learning application is then based on three nodes: the input node, the inference node (the one that is custom created for this project) and the output node.
The input node is the one that loads the image in the flow, either by taking a picture from the webcam or by loading an image file from the local device's memory. Additionally the node creates a message with a payload and loads inside it the data about the image.
The inference node is the one that elaborates the image with machine learning techniques and generates a prediction. Its behaviour can be separated in four main functions. The first function is called as soon as the node is inserted in the environment and the flow is deployed and is the $LoadGraphModel$. This uses the Tensorflow library and loads the model from a web page previously specified in the node parameters window. The other three functions on the other hand are called only at run time, when the node receives an input message. The first function of the group is the $PreProcessing$, which takes the message payload, transforms it in a tensor, reshapes it compressing its dimension to a tensor of shape 224x224, and then ensures that the values are float. The second function is then the inference one. This will simply feed the input tensor to the model which will generate a tensor containing the prediction. The prediction tensor is then elaborated by the last function which is the $OutputProcessor$. This function extracts the numerical values from the tensor in order to understand which class got the highest prediction and inserts all the useful data inside a JSON file that is then stored inside the payload of the message.
After the inference node the message is sent to the output node, which in this case is a simple debug node. This node will display the information stored inside the JSON file in the Node-RED debug tab, making the prediction easily readable by the user.

Note that in order to make the node work correctly it's necessary to specify some information inside its properties tab. Specifically it's required to define the link from which the model is loaded (in this case it's just the raw version of the model loaded in a GitHub repository [5]), the number of prediction classes and the names of these classes.



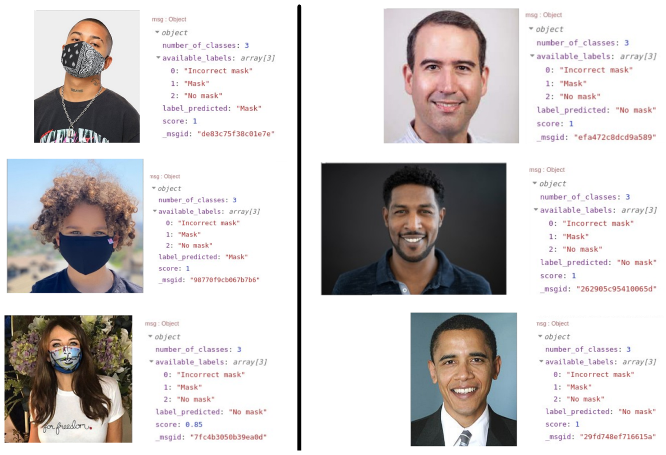
Properties

| | |
|---|---|
| Name | tfjs my_node |
| Model Url | https://raw.githubusercontent.com/AlessandroAvi/I |
| Number of classes | 3 |
| Class 1 | Incorrect mask |
| Class 2 | Mask |
| Class 3 | No mask |
| Class 4 | Class 4 |

Fig. 4. Properties tab of the inference node

## V. Experimental Results

Once everything is installed and all the external nodes are deployed inside the flow it's possible to test the project. For doing so it's enough to load an image from the local device memory or take a picture with the webcam. Once the input is generated node elaborates the prediction in just a couple of seconds and displays the results in the debug tab.
It can be said that the node is quite reliable and often the prediction has an accuracy that is above $90\%$. By experimenting with some random images downloaded from the web it's clear how the prediction of the model is robust and is able to predict correctly and reliably the classes.
Since the custom node requires just a link to a JSON model loaded on a web page it's quite easy to exploit the node's flexibility and perform classifications with other models. Ir is n fact required just to train another classification model with different characteristics (different model structure, different dataset, different classes, ...), load it on the web and provide to the node the correct link. To demonstrate this, other two models have been trained. One for the classifications of dogs and cats, the other for the classification of once again images with masks. In the latter case the classes are ”with mask”, ”without mask” and ”incorrect mask”. The results of the pet classification mode show once again great great accuracy and reliability, while the same cannot be said for the version 2 of the mask detector. In this last case in fact it often happens that the model wrongly associates the images of ”incorrect mask” to the other two labels. This most probably is due to a bad dataset creation that required the augmentation of just 500 images for that the ”incorrectly worn” class. In any case the application of this model with a classification of three different labels is a proof of flexibility and simplicity of use for this Node-RED machine learning node.



```
msg : Object
▾ object
  number_of_classes: 3
  ▾ available_labels: array[3]
    0: "Incorrect mask"
    1: "Mask"
    2: "No mask"
  label_predicted: "Mask"
  score: 1
  _msgid: "de83c75f38c01e7e"
```

```
msg : Object
▾ object
  number_of_classes: 3
  ▾ available_labels: array[3]
    0: "Incorrect mask"
    1: "Mask"
    2: "No mask"
  label_predicted: "Mask"
  score: 1
  _msgid: "98770f9cb067b7b6"
```

```
msg : Object
▾ object
  number_of_classes: 3
  ▾ available_labels: array[3]
    0: "Incorrect mask"
    1: "Mask"
    2: "No mask"
  label_predicted: "No mask"
  score: 0.85
  _msgid: "7fc4b3050b39ea0d"
```

```
msg : Object
▾ object
  number_of_classes: 3
  ▾ available_labels: array[3]
    0: "Incorrect mask"
    1: "Mask"
    2: "No mask"
  label_predicted: "No mask"
  score: 1
  _msgid: "efa472c8dcd9a589"
```

```
msg : Object
▾ object
  number_of_classes: 3
  ▾ available_labels: array[3]
    0: "Incorrect mask"
    1: "Mask"
    2: "No mask"
  label_predicted: "No mask"
  score: 1
  _msgid: "262905c95410065d"
```

```
msg : Object
▾ object
  number_of_classes: 3
  ▾ available_labels: array[3]
    0: "Incorrect mask"
    1: "Mask"
    2: "No mask"
  label_predicted: "No mask"
  score: 1
  _msgid: "29fd748ef716615a"
```

Fig. 5. Example of predictions obtained by the node

## VI. Conclusions

The project goal was to create a custom mode in the Node-RED environment for running image classification with machine learning techniques. The project can be considered successful since the final scope has been reached. Additionally it can be said that since the node is based on models loaded on the web it is very flexible and is possible to expand its usage with minimal effort.
A couple of future work could be the usage of similar nodes but applied on different machine learning models, like object detection, speech recognition or else. Another interesting application, since the project proved to be possible and easy to reproduce, could be the deployment of the entire flow on a powerful embedded device such as Raspberry Pi.
This type of technology could be implemented easily in an IoT chain for enhancing the elaboration of data. One example could be an application similar to what seen in [2], where ML techniques could be used for predicting the flow of waste in bins distributed in a city.

## References

[1] V. David, H. Ragu, R. K. Duraiswamy, and P. Sasikumar, “Iot based automated indoor agriculture system using node-red and ibm bluemix,” in *2021 6th International Conference on Inventive Computation Technologies (ICICT)*. IEEE, 2021, pp. 157–162.

[2] D. Bramhankar, “Implementation of iot based smart waste management system in node red.”

[3] S. Bharath, “Iot based smart switch with bluetooth speaker using mqtt protocol: Node-red framework,” in *2021 2nd International Conference on Advances in Computing, Communication, Embedded and Secure Systems (ACCESS)*. IEEE, 2021, pp. 61–64.

[4] Tutorial developed by ibm. [Online]. Available: https://developer.ibm.com/tutorials/building-a-machine-learning-node-for-node-red-using-tensorflowjs/

[5] Github repository for the entire project. [Online]. Available: https://github.com/AlessandroAvi/Node$_{r}ed_{t}fjs_{c}lassification$

[6] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.