Object Detection & Image Classification of Mask Based Dataset

Sina Ghanaat
Dept of Mechanical Engineering,
K. N. Toosi University of Technology, Tehran, Iran

***Abstract*—This is the Second of three projects for the "AIES" course instructed by Dr. Esmaeil Najafi at the Department of Mechanical Engineering of K. N. Toosi university of technology. In this project, two deep learning architectures have been implemented in order to detect three classes of people wearing mask. These three classes are wearing no mask, wearing mask and wearing mask incorrectly. Two architectures are first *YOLO version 5 small* in the object detection part, and second *inception version* 3 architecture for classification in the second part of the project.**

Keywords—Object Detection, COVID19, Yolo V5, Inception V3, Deep Learning

I. INTRODUCTION

This project will explain the comparison of functionality of object detection architecture and image classification architecture for determining three classes of masked, no masked and mask worn incorrect. *YOLO version 5 medium* architectures have been chosen as the object detection structure and *inception version 3* for image classification in the second part. The raw available dataset contains *png* images which fit into three classes. The images' annotation for object detection part is *Pascal VOC* which must be converted to *YOLO V5* format for the object detection part. The second part of the project is a classification task, therefore all the images in the raw dataset must be fit into three classes exclusively. That means that any image which contains more than two or more different classes, must be cropped in a way that resulting images fit into one class exclusively. Moreover, the images in the raw dataset have different input sizes however *inception v3* architecture accept images with constant size, therefore all images in the raw dataset must be stretched into a fixed shape size. This will satisfy the image augmentation processing as well.

II. PART I

In the first part of the project, *YOLO version 5 medium* has been chosen as the object detection architecture. There is a user friendly and fast online notebook named *Google Colab* for implementing *YOLO 5* architecture in this project.

A. Data Preprocessing

The raw dataset contains *png* 4 channels images and *Pascal VOC* annotations. Since the architecture for object detection is *YOLO version 5*, the annotation must be converted to *YOLO V5* format. For achieving this, an online developer tool called *ROBOFLOW* has been utilized. All the images and their respective annotations were uploaded and a new preprocessed and *YOLO V5* prepared dataset has been generated. No data augmentation has been exerted on the images and the train/test split for the data is 80 percent. However, in addition to test/train split ratio, 20 percent of the training dataset will be considered as validation data. After this, with support of *Google Colab* environment, the *YOLO version 5* medium have been designed and implemented.

B. YOLO V5 MEDIUM Implementation

The codes have been written in the *Google Colab* notebook *(https://colab.research.google.com/drive/1Ojlb604K22cGXKCbNwuD-S41B5QMpPN5)*. The path to implement the *YOLO version 5* was pretty much straightforward. First the *YOLO version 5* repository cloned and after installing all the requirements, the preprocessed dataset uploaded to *Google Colab* directory. Then by defining classes, training sets and test sets in *data.yaml* file and set the epoch to 64 and input size to 720 pixels, the YOLO is ready for training. There should be a consideration that no hyperparameter has been changed from their default value, therefore all the hyperparameter can be seen in the *Colab* notebook log.

C. Result

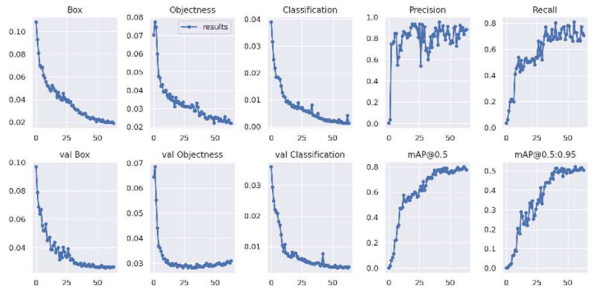

Figure 1 Result graphs for the *YOLO version 5 medium* object detection architecture (per epoch)

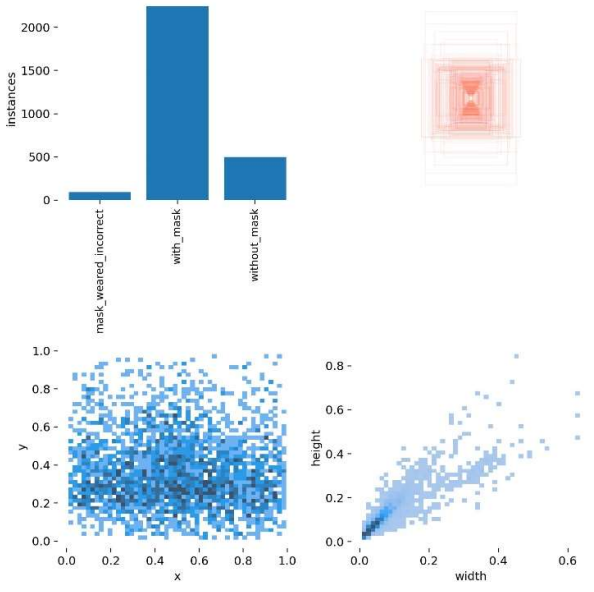

Figure 2 Class data frequency (top left); Snapshot of the labels in normalized height and width (top right); labels coordinates frequency (bottom left); labels normalized height and width graph (bottom right)

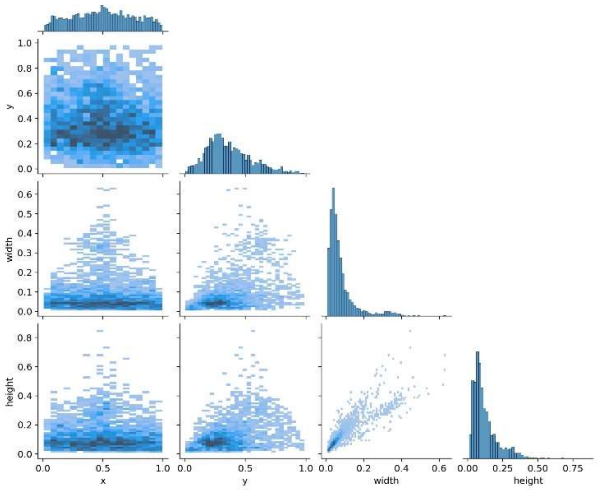

Figure 3 Frequency of dimension and coordination of label boxes

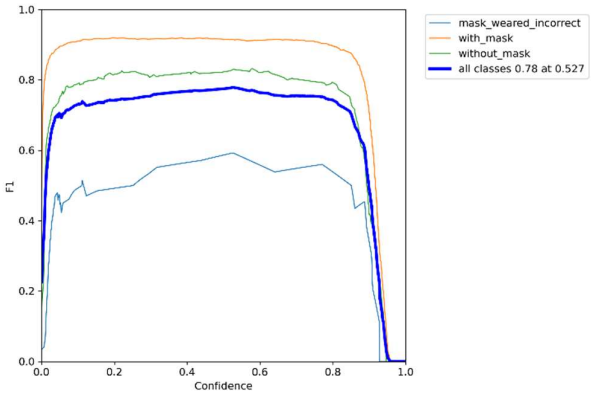

Figure 4 The f1 score for each class based on confidence score

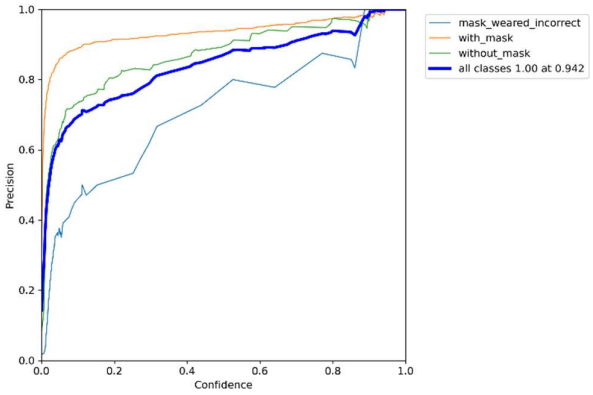

Figure 5 The precision value for each class based on confidence score

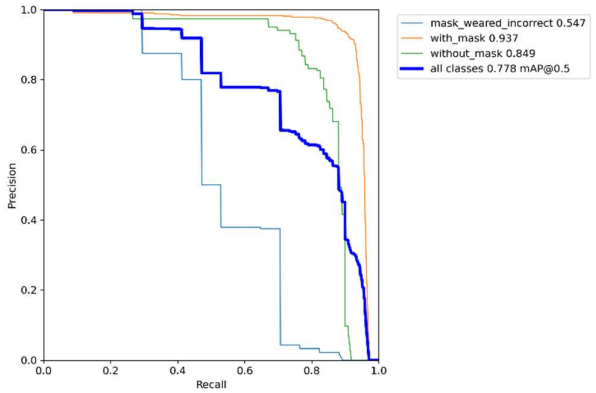

Figure 6 Precision to recall graph

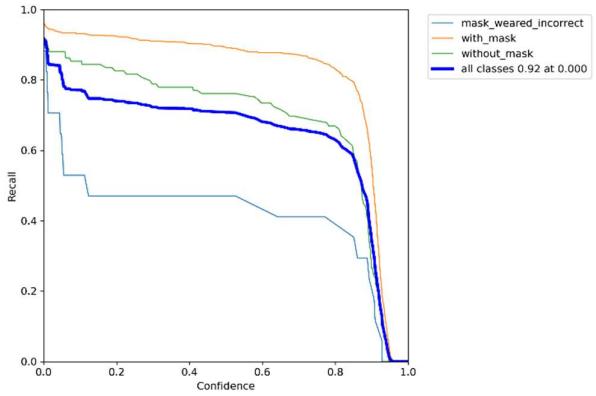

Figure 7 The recall value for each class based on confidence score

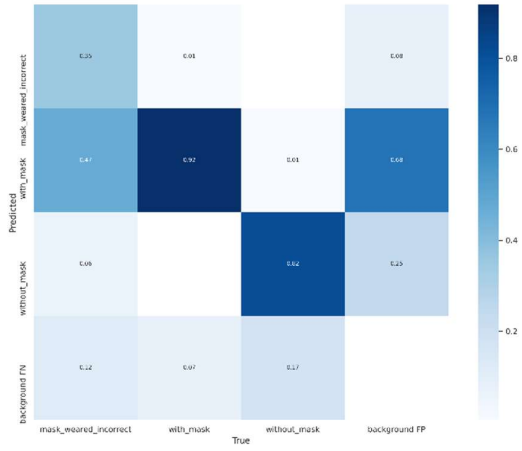

Figure 8 The confusion matrix for validation set

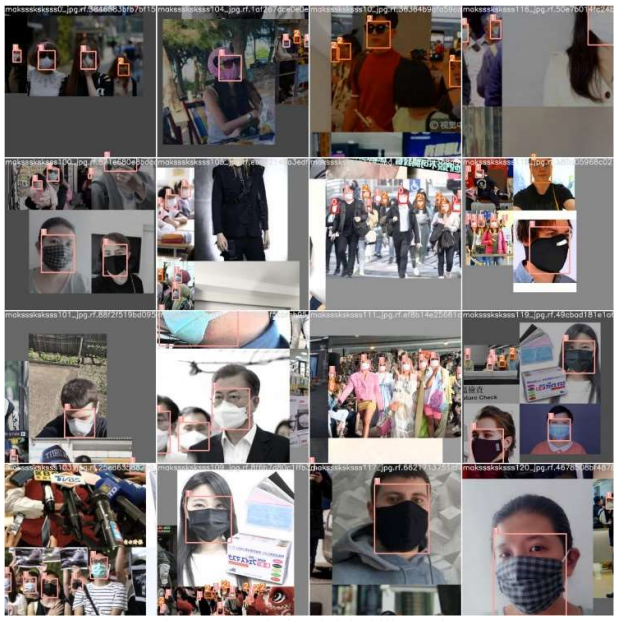

Figure 9 A train batch label illustration

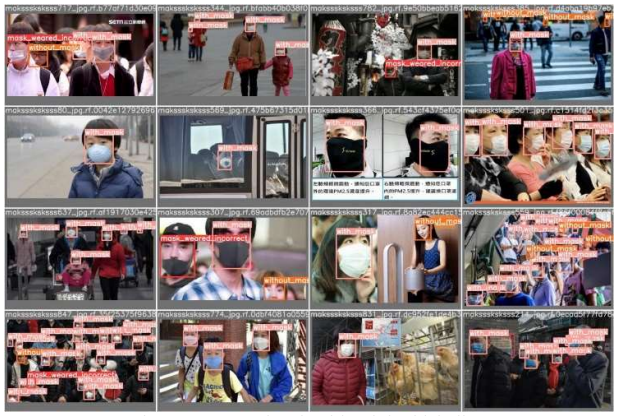

Figure 10 A test batch with original labeling

Figure 11 Same test batch after implementing detector and predicted labeling

Figure 12 A test image which object detector has not detect any class

Figure 13 A test image which object detector detect the commercial poster as a person classified with one of the three classes

Figure 14 The non-labeled image of Fig.13

As illustrated in Fig.1, the overall training of the YOLO has been a success. The accuracy graphs and their respective correlation shows no underfitting has been occurred. The overfitting for this project is not an issue since images are general in their nature due to multiple sources of origin that the come from. The positive recall and precision graphs both along with *mAP* show that the trained model is in a good shape.

Fig.2 shows the non-homogeneous data source. As illustrated, the with mask class labels are more than 2000 which compare to without mask class with less than 500 instances and mask worn incorrect with less than a quarter of that, are a lot. So, there may be an expectation of better result on the side of the with mask class than the other two and a not good result on the side of the mask worn incorrect class. In addition, the dispersion of height and width plus label

coordinates shows that majority of the labeling boxes are relatively small and most of the positioning are in the lower portion of the images.

The f1 score, recall, precision graphs based on confidence and confusion matrix are presented in Fig.4 to Fig.8. As illustrated in figures, the behavior of the *YOLO V5* is almost same for all classes except for the differences between the metric values. Due to insufficient data for the mask worn incorrect compare to with mask class, we can see that the behavior of the *YOLO V5* for mask worn incorrect class is not as strong as with mask class. This issue is same for the without mask class but in a better shape. This problem can be vividly seen in the confusion matrix since with mask class has a good validation score while the detector confuses the without mask and mask worn incorrect class with each other. This is a result of low ppi of images and less available data om both classes.

III. Part II

In the second part of the project, Inception version 3 has been chosen as the image classification architecture. Visual studio code was the software for coding.

A. Data Preprocessing

In this part of project which is a classification task, all the images in the raw dataset must be fit into three classes exclusively. That means that any image which contains equal or more than two different classes, must be cropped in a way that resulting images fit into one class exclusively. Therefore, all images have been checked and cropped if necessary and a new dataset has been generated. Then the images have been uploaded to *Roboflow* website for more processing. Since *Inception V3* architecture accept constant shape size as its input, all images must be stretched to a constant size shape which has been chosen 256×256 pixels.

B. Inception V3 Implementation

The codes have been written in the visual studio code notebook. We used *Tensorflow*, *Keras* modules for implementing pretrained *Inception V3*. We consider the minimum consecutive epochs for training as 12 and maximum of 64. *Inception V3* in this project has all layers feezed except for the last 75 layers of it. Then the top classifier is a three layer fully connected network with 1028-512 – 256 to three outputs. We used the SoftMax activation function at the output layer. Out optimizer is Adam with learning rate of 0.0001. we determine the performance of the network based on the validation accuracy. No data augmentation has been implemented although the effort to make all images' size same caused a semi data augmentation in the meantime. The network has been trained by CPU.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inception_v3 (Functional)    (None, 2048)              21802784
_________________________________________________________________
dropout_1 (Dropout)          (None, 2048)              0
_________________________________________________________________
dense_3 (Dense)              (None, 1028)              2106372
_________________________________________________________________
dense_4 (Dense)              (None, 512)               526848
_________________________________________________________________
dense_5 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_6 (Dense)              (None, 3)                 771
=================================================================
Total params: 24,568,103
Trainable params: 14,765,831
Non-trainable params: 9,802,272
```

Figure 15 Overview of the network architecture

C. Result

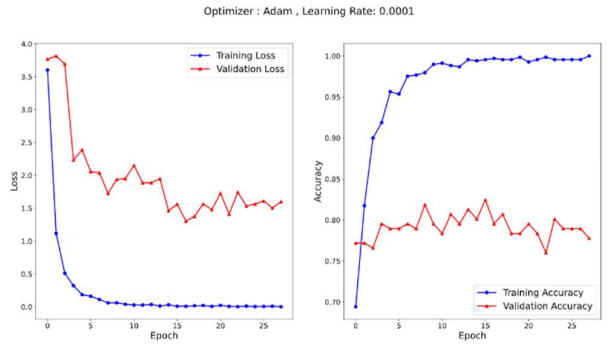

Figure 16 Loss and accuracy chart for train set and validation

	Train
loss	0.025085
accuracy	0.991304

Figure 17 Best weights loss and accuracy for train data

	Validation
loss	1.936321
accuracy	0.795322

Figure 18 Best weights loss and accuracy for validation data

Validation Classification Report

	precision	recall	f1-score	support
Mask	0.85	0.90	0.87	131
MaskIn	0.00	0.00	0.00	11
NoMask	0.52	0.55	0.53	29
accuracy			0.78	171
macro avg	0.46	0.48	0.47	171
weighted avg	0.74	0.78	0.76	171

Figure 19 Validation classification report for best weight selection

	Predicted Mask	Predicted MaskIn	Predicted NoMask
True Mask	118	1	12
True MaskIn	8	0	3
True NoMask	13	0	16

Figure 20 Validation matrix

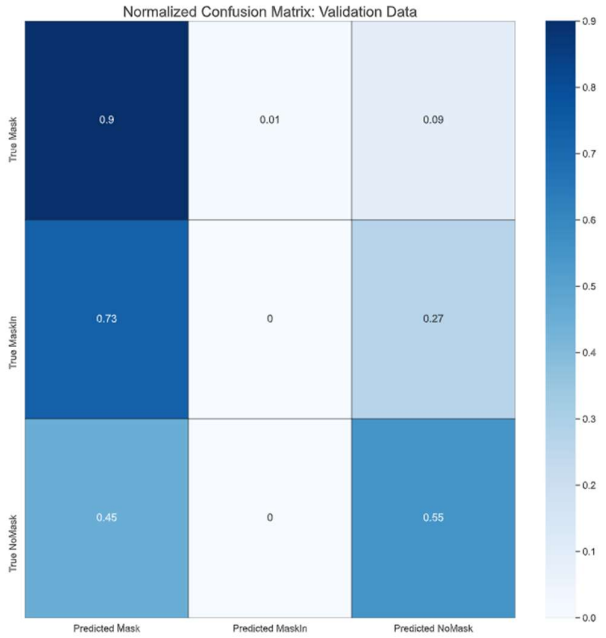

Figure 21 Confusion matrix for validation set

	Test
loss	0.978966
accuracy	0.863158

Figure 22 Loss and accuracy for test set

Test Classification Report

	precision	recall	f1-score	support
Mask	0.93	0.97	0.95	73
MaskIn	0.00	0.00	0.00	6
NoMask	0.72	0.81	0.76	16
accuracy			0.88	95
macro avg	0.55	0.60	0.57	95
weighted avg	0.84	0.88	0.86	95

Figure 23 Classification report for test set

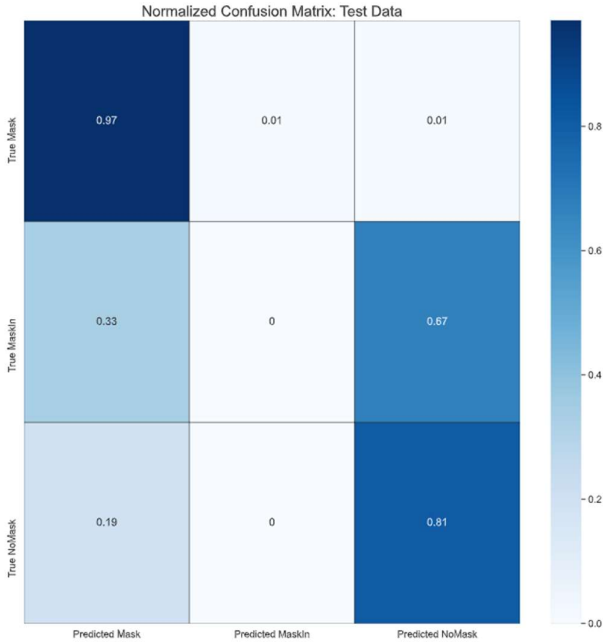

Figure 24 Confusion matrix for test set

As we can see in Fig.16, the model is overfit which is something we would expect and it won't cause problem. Based on the validation accuracy, the best weights for the trained network have been chosen.

As shown in Fig.19 the with mask class is properly identified by the network but the other two classes were poorly trained which the mask worn incorrect class has no valid data which can be seen in the Fig. 21 by the confusion matrix.

This same for the test data which shown in Fig.23 and Fig.24.

All in all, this project showed that the object detection architecture is far more accurate and efficient in classifying than a deep CNN architecture. In the meantime, the *YOLO V5* can detect the classes within the same image but the image classification architecture must be feed with exclusive class images.