Indonesian Journal of Artificial Intelligence and Data Mining (IJAIDM)
Vol. 4, No. 2, September 2021, pp. 97 – 107
p-ISSN: 2614-3372 | e-ISSN: 2614-6150

Real-Time Detection of Face Masked and Face Shield Using YOLO Algorithm with Pre-Trained Model and Darknet

[1]Muhamad Muhaimin, [2]Tjong Wan Sen
[2]Information Technology, Faculty of Computing, President University
Email: [1]m.muhamin@student.president.ac.id ,[2]wansen@president.ac.id

Article Info

Article history:
Received Aug 24th, 2021
Revised Sept 1st, 2021
Accepted Sept 30th, 2021

Keyword:
Average Pooling
Darknet
Deep Learning
Max Pooling
YOLO

ABSTRACT

There are new regulations requiring the use of masks or face shields to prevent the transmission of Covid-19. Using deep learning, a model can be made to detect faces that use masks and face shields by training the model using the previous pre-trained model and using a custom dataset. The purpose of this study is to create a deep learning model that can detect faces with and without masks and as well as face shields for the prevention of covid-19 transmission using You Only Look Once (YOLO) with pre-trained models and custom datasets in real-time. In this study, using pre-trained models from YOLOv3, YOLOv3-Tiny, YOLOv4, YOLOv4-Tiny, and YOLOv4-Tiny-3l with Darknet Framework and compare between average pooling and max pooling in the convolutional neural network YOLO to detect face masks and face shields as a real-time. From experiment the mAP (mean average precision) was obtained from YOLOv4 using average pooling with a value is 97.64% although the difference is not too much with YOLOv4 using max pooling with value 97.57% and the lowest was YOLOv3-Tiny using max pooling, which was 94.09%, and for the highest FPS (frame per second) was obtained by YOLOv4-Tiny with Fps values is 171 and mAP 96.75%.And for real-time detection of face masks and face shields, the best model used in testing using webcam 1080p is from YOLOv4-Tiny, because the FPS obtained is the highest of all YOLO models with a value of 171FPS and mAP is quite high with value is 96.75%.

Corresponding Author:
Muhamad Muhaimin,
Information Technology, Faculty of Computing
President University,
Jababeka Education Park, Jl. Ki Hajar Dewantara, North Cikarang, Bekasi, West Java, Indonesia.
Email: m.muhaimin@student.president.ac.id

DOI: http://dx.doi.org/10.24014/ijaidm.v2i2.13989

1. INTRODUCTION

The transmission of the covid-19 virus is too fast, that is why in some places it is mandatory to use masks and face shields as a preventive measure in reducing the transmission of this virus. However, some areas or places are too large to be supervised by one person or officer. Object Recognition is a technique that can recognize an object in an image, video or real-time using a camera that aims to follow the position of a moving object. Object tracking can be used to detect faces with masks, facechield or without both. One method in Object Recognition is to use deep learning, where there are many architectures and models [1].

Previously, there was research by Sabbir Ejaz et al [2] for Face Masked Recognition using Convolution Neural Network (CNN) and the obtained accuracy values vary with the lowest accuracy value being 63.52% and the highest reaching 98.10%, using 3 different datasets. This study concentrates on the detection of masks combined with hats, sunglasses, beards, long hair, mustaches, and medical masks, but the method in this study is not suitable for all types of masks.

Another study was conducted by Rakshitha Gopal et al [3] to detect small objects using Single Stage CNN Object Detectors and Tiny-YOLOv3, where the results from Tiny-YOLOv3 have relatively better performance with 60% better accuracy and 0.09 FPS when test object detection in the real-time. Another studies by Pranav Adarsh et al [4] for object detection using a one stage improved model and using Tiny-YOLOv3, where in this study they compared two stages of object detection, the first stage was the detector with the algorithm of R-CNN [5], Fast RCNN [6], dan Faster-RCNN [7], while other detectors use YOLOv1[8], YOLOv2 [9], YOLOv3 [10], and SSD. YOLOv3 results are faster than Faster R-CNN [7], and Tiny-YOLOv3 [4] is even faster than YOLOv3 for object detection in the real-time using camera.

YOLO is quite popular as a *state-of-the-art* for object recognition, this is proven by research conducted by Fan Wu et al [11] using YOLOv3 to detect workers who do not use helmets with CCTV and low resolution, the mean average precision (mAP) value reaches 93.5%. Mean average precision (mAP) is used to evaluate the object detection model.

Pooling Layer [12] is an important building block in CNN. Pooling layer functions to reduce input spatially which reduces the number of parameters with down-sampling operations [13]. The pooling methods commonly used are max pooling and average pooling [14], In some cases max pooling or average pooling can greatly help improve accuracy and performance, however the pooling operation has some limitations. For example, max pooling only extracts the maximum value of the region while average pooling only extracts the average value of the region [13]. In the study of Victor and Isabel [12] evaluated the performance of several pooling methods for the extraction of Drug-Dug Interaction (DDI), where the result was that max pooling got better performance with an F1 value of 64.56% while average pooling was only 58.35%. While in the research of Mao et al [15] that CNN performance with average pooling using kernel size = 5 has better performance than CNN with max pooling although the difference is not too much.

Therefore, this study will propose real-time detection of face masks and face shields using the YOLO algorithm as a model with a darknet neural network framework [16] who will be trained using google colab pro using a pre-trained model and comparing the use of average pooling with max pooling in the neural network for versions of YOLOv3, YOLOv3-Tiny, YOLOv4, YOLOv4-Tiny and YOLOv4-Tiny-3l. accuracy (mAP), F1 Score and performance (FPS) and validated and tested using the Non Maximum suppression (NMS) algorithm [17].

2. RESEARCH METHOD

This research is divided into several stages, the following is the process flow for real-time facemask and face shield detection which is described in the following figure:

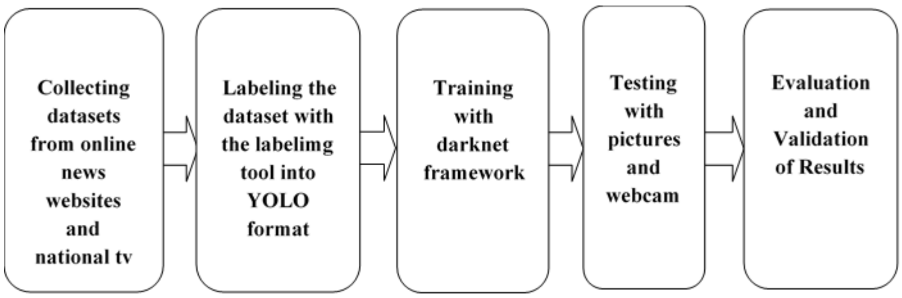


Figure 1. Flow Research Stages

2.1. Dataset Collection

The dataset collection process is divided into two, firstly, the images for the dataset are taken from online news websites, and the second is obtained from national tv broadcasts on YouTube, because the use of face shields is widely used by national tv stations. From the dataset collection process, 773 images were obtained in .jpg format and the composition of the dataset is as follows:

Table 1. Dataset Composition

No mask	Facemask	Face shield
150 Images	161 Images	462 Images

From the dataset composition table, it can be seen that there are more images for face shields than others, because face shields are transparent objects, where image angles and lighting can also be difficult to detect face shields, that's the reason why more face shield images are needed. although the number of images for the dataset is relatively small, which is less than 1000, but in one image there can be 2-10 faces or even more.

2.2 Labelling

After the dataset is collected, then the images are labeled one by one with the labeliмg tool into the YOLO format, and in this labeling process 3 classes will be used, namely "nomask", "facemask", and "faceshield".

2.3 Training Model

The model that will be built using YOLO with pre-trained model and darknet framework, and this research will propose and compare average pooling with max pooling in the neural network for versions of YOLOv3, YOLOv3-Tiny, YOLOv4, YOLOv4-Tiny and YOLOv4-Tiny-3l, to find out which performance is better, it can be seen from the result values of mAP, F1 and FPS. For the training process will use platform google colab pro with the following GPU Information:

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 465.27       Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   44C    P0    28W / 250W |      0MiB / 16280MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
```

Figure 2. Google Colab GPU

And the flow of the training process is as follows:

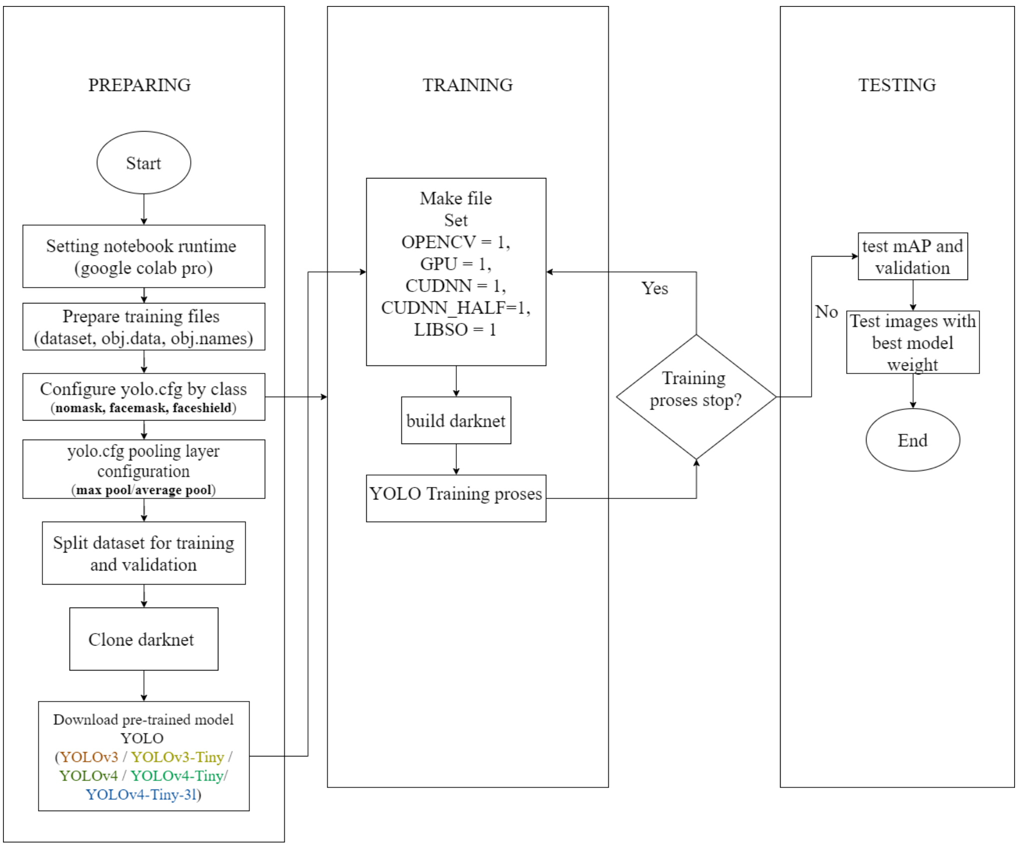


Figure 3. Flow Training Proses

The flow above is the flow of the YOLO training process with a pre-trained model that was carried out on Google Collab Pro for 1 training process. The training process with pre-trained models is carried out alternately/separately based on the YOLO model used and the pooling used, in this case max pooling and average pooling. This study will test the level of accuracy and performance of the training model for each version of YOLO by using max pooling and average pooling used in convolutional neural networks in YOLO.

2.4 Architecture Model

For the architecture of the model itself, the pre-trained model used for training on each version of the YOLO model tested is as follows:

Table 2. Pre-Trained Model YOLO

Model	YOLOv3	YOLOv3-Tiny	YOLOv4	YOLOv4-Tiny	YOLOv4-Tiny-3l
Pre-Trained weight	Darknet53.conv.74	Yolov3-tiny.conv.11	Yolov4.conv137	Yolov4-tiny.conv.29	Yolov4-tiny.conv.29
Number of Layers	106	24	161	37	44
Pooling Layer	Max Pooling	Max Pooling	Max Pooling / Average Pooling	Max Pooling / Average Pooling	Max Pooling / Average Pooling

To see the architectural differences, here are the differences in the architecture of the YOLO v4-tiny model using max pooling and average pooling on google colab pro.

```
conv     32       3 x 3/ 2    416 x 416 x   3 ->  208 x 208 x  32 0.075 BF
   1 conv     64       3 x 3/ 2    208 x 208 x  32 ->  104 x 104 x  64 0.399 BF
   2 conv     64       3 x 3/ 1    104 x 104 x  64 ->  104 x 104 x  64 0.797 BF
   3 route  2                                  1/2 ->  104 x 104 x  32
   4 conv     32       3 x 3/ 1    104 x 104 x  32 ->  104 x 104 x  32 0.199 BF
   5 conv     32       3 x 3/ 1    104 x 104 x  32 ->  104 x 104 x  32 0.199 BF
   6 route  5 4                                    ->  104 x 104 x  64
   7 conv     64       1 x 1/ 1    104 x 104 x  64 ->  104 x 104 x  64 0.089 BF
   8 route  2 7                                    ->  104 x 104 x 128
   9 max                2x 2/ 2    104 x 104 x 128 ->   52 x  52 x 128 0.001 BF
  10 conv    128       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 128 0.797 BF
  11 route  10                                 1/2 ->   52 x  52 x  64
  12 conv     64       3 x 3/ 1     52 x  52 x  64 ->   52 x  52 x  64 0.199 BF
  13 conv     64       3 x 3/ 1     52 x  52 x  64 ->   52 x  52 x  64 0.199 BF
  14 route  13 12                                  ->   52 x  52 x 128
  15 conv    128       1 x 1/ 1     52 x  52 x 128 ->   52 x  52 x 128 0.089 BF
  16 route  10 15                                  ->   52 x  52 x 256
  17 max                2x 2/ 2     52 x  52 x 256 ->   26 x  26 x 256 0.001 BF
  18 conv    256       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 256 0.797 BF
  19 route  18                                 1/2 ->   26 x  26 x 128
  20 conv    128       3 x 3/ 1     26 x  26 x 128 ->   26 x  26 x 128 0.199 BF
  21 conv    128       3 x 3/ 1     26 x  26 x 128 ->   26 x  26 x 128 0.199 BF
  22 route  21 20                                  ->   26 x  26 x 256
  23 conv    256       1 x 1/ 1     26 x  26 x 256 ->   26 x  26 x 256 0.089 BF
  24 route  18 23                                  ->   26 x  26 x 512
  25 max                2x 2/ 2     26 x  26 x 512 ->   13 x  13 x 512 0.000 BF
  26 conv    512       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x 512 0.797 BF
  27 conv    256       1 x 1/ 1     13 x  13 x 512 ->   13 x  13 x 256 0.044 BF
  28 conv    512       3 x 3/ 1     13 x  13 x 256 ->   13 x  13 x 512 0.399 BF
  29 conv     24       1 x 1/ 1     13 x  13 x 512 ->   13 x  13 x  24 0.004 BF
  30 yolo
[yolo] params: iou loss: ciou (4), iou_norm: 0.07, obj_norm: 1.00, cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.05
nms_kind: greedynms (1), beta = 0.600000
  31 route  27                                     ->   13 x  13 x 256
  32 conv    128       1 x 1/ 1     13 x  13 x 256 ->   13 x  13 x 128 0.011 BF
  33 upsample                 2x    13 x  13 x 128 ->   26 x  26 x 128
  34 route  33 23                                  ->   26 x  26 x 384
  35 conv    256       3 x 3/ 1     26 x  26 x 384 ->   26 x  26 x 256 1.196 BF
  36 conv     24       1 x 1/ 1     26 x  26 x 256 ->   26 x  26 x  24 0.008 BF
  37 yolo
```

Figure 4. Network arsitektur YOLOv4-Tiny with max pooling

```
conv     32       3 x 3/ 2    416 x 416 x   3 ->  208 x 208 x  32 0.075 BF
   1 conv     64       3 x 3/ 2    208 x 208 x  32 ->  104 x 104 x  64 0.399 BF
   2 conv     64       3 x 3/ 1    104 x 104 x  64 ->  104 x 104 x  64 0.797 BF
   3 route  2                                 1/2 ->  104 x 104 x  32
   4 conv     32       3 x 3/ 1    104 x 104 x  32 ->  104 x 104 x  32 0.199 BF
   5 conv     32       3 x 3/ 1    104 x 104 x  32 ->  104 x 104 x  32 0.199 BF
   6 route  5 4                                   ->  104 x 104 x  64
   7 conv     64       1 x 1/ 1    104 x 104 x  64 ->  104 x 104 x  64 0.089 BF
   8 route  2 7                                   ->  104 x 104 x 128
   9 avg                2x 2/ 2    104 x 104 x 128 ->   52 x  52 x 128 0.001 BF
  10 conv    128       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 128 0.797 BF
  11 route  10                                1/2 ->   52 x  52 x  64
  12 conv     64       3 x 3/ 1     52 x  52 x  64 ->   52 x  52 x  64 0.199 BF
  13 conv     64       3 x 3/ 1     52 x  52 x  64 ->   52 x  52 x  64 0.199 BF
  14 route  13 12                                 ->   52 x  52 x 128
  15 conv    128       1 x 1/ 1     52 x  52 x 128 ->   52 x  52 x 128 0.089 BF
  16 route  10 15                                 ->   52 x  52 x 256
  17 avg                2x 2/ 2     52 x  52 x 256 ->   26 x  26 x 256 0.001 BF
  18 conv    256       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 256 0.797 BF
  19 route  18                                1/2 ->   26 x  26 x 128
  20 conv    128       3 x 3/ 1     26 x  26 x 128 ->   26 x  26 x 128 0.199 BF
  21 conv    128       3 x 3/ 1     26 x  26 x 128 ->   26 x  26 x 128 0.199 BF
  22 route  21 20                                 ->   26 x  26 x 256
  23 conv    256       1 x 1/ 1     26 x  26 x 256 ->   26 x  26 x 256 0.089 BF
  24 route  18 23                                 ->   26 x  26 x 512
  25 avg                2x 2/ 2     26 x  26 x 512 ->   13 x  13 x 512 0.000 BF
  26 conv    512       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x 512 0.797 BF
  27 conv    256       1 x 1/ 1     13 x  13 x 512 ->   13 x  13 x 256 0.044 BF
  28 conv    512       3 x 3/ 1     13 x  13 x 256 ->   13 x  13 x 512 0.399 BF
  29 conv     24       1 x 1/ 1     13 x  13 x 512 ->   13 x  13 x  24 0.004 BF
  30 yolo
[yolo] params: iou loss: ciou (4), iou_norm: 0.07, obj_norm: 1.00, cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.05
nms_kind: greedynms (1), beta = 0.600000
  31 route  27                                    ->   13 x  13 x 256
  32 conv    128       1 x 1/ 1     13 x  13 x 256 ->   13 x  13 x 128 0.011 BF
  33 upsample                 2x    13 x  13 x 128 ->   26 x  26 x 128
  34 route  33 23                                 ->   26 x  26 x 384
  35 conv    256       3 x 3/ 1     26 x  26 x 384 ->   26 x  26 x 256 1.196 BF
  36 conv     24       1 x 1/ 1     26 x  26 x 256 ->   26 x  26 x  24 0.008 BF
  37 yolo
```

Figure 5. Network arsitektur YOLOv4-Tiny with average pooling

3. RESULTS AND ANALYSIS

From the results of the training experiment using pre-trained models from versions of YOLOv4, YOLOv4-Tiny, YOLOv4-Tiny-3l, YOLOv3 and YOLOv3-Tiny with a darknet framework and custom datasets using both max pooling and average pooling, the difference can be seen from the mAP and loss graphs as follows:

Max Pooling Average Pooling

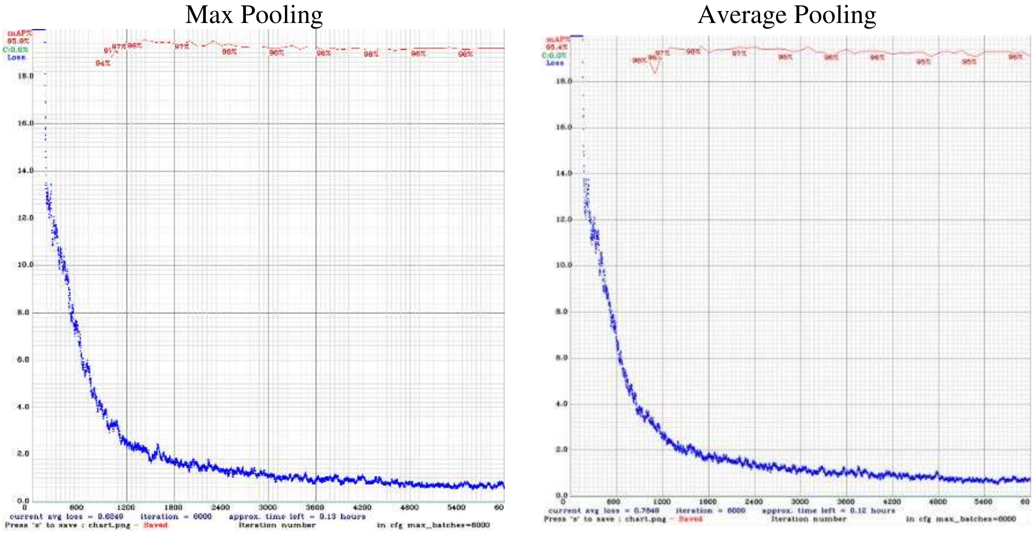


Figure 6. mAP and Loss Yolov4 Chart

Max Pooling Average Pooling

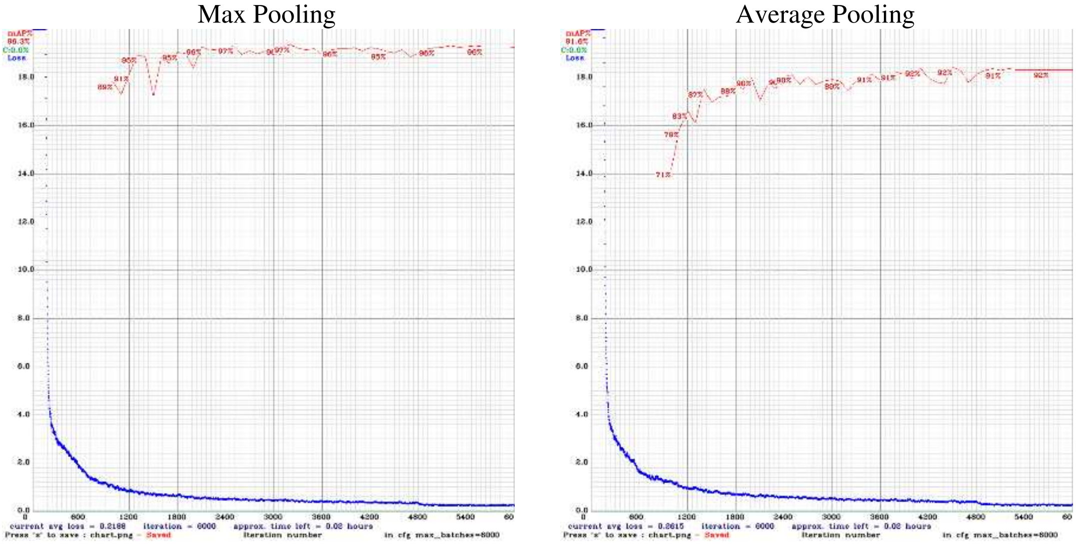


Figure 7. mAP and Loss Yolov4-Tiny Chart

Max Pooling Average Pooling

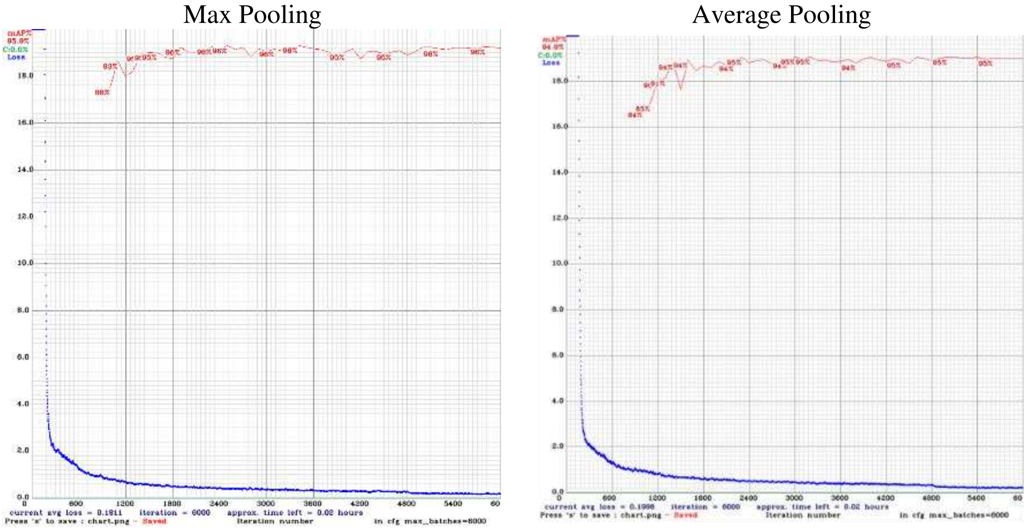


Figure 8. mAP and Loss Yolov4-Tiny-3l Chart

Max Pooling

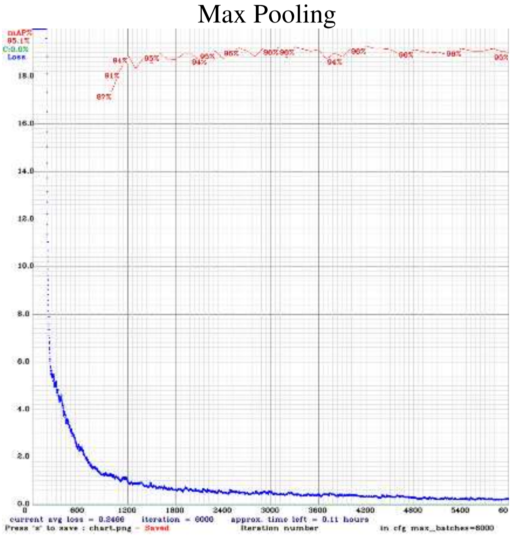


Figure 9. mAP and Loss Yolov3 Chart

Max Pooling

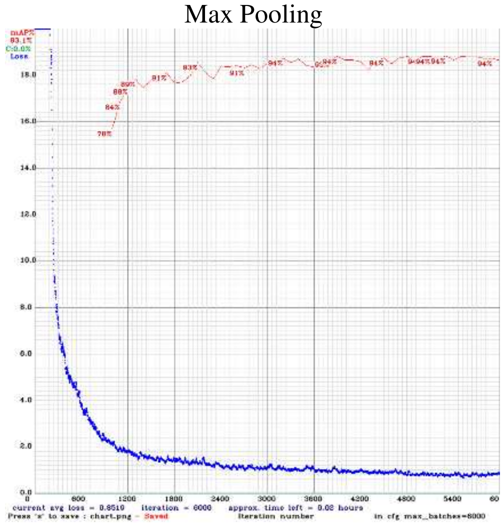


Figure 10. mAP and Loss Yolov3-Tiny Chart

From the training results, it is known that the YOLOv4 model using max pooling is better for mAP and performance loss. After training, validation and performance are carried out to determine the mAP and FPS values in each model. The following is a FPS comparison table for each model after being tested with a video file.

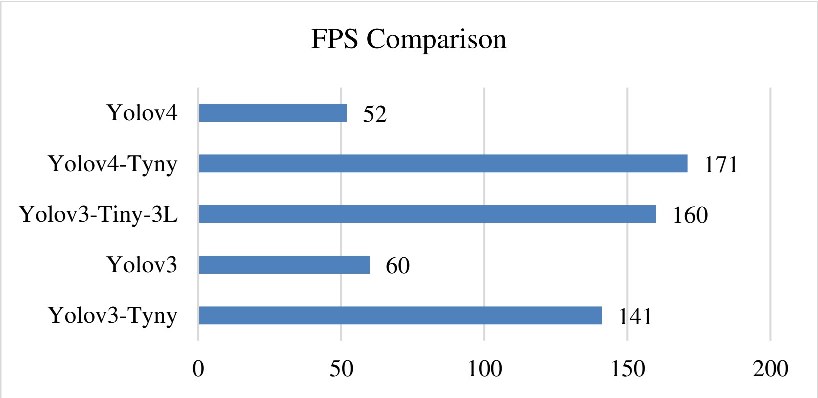
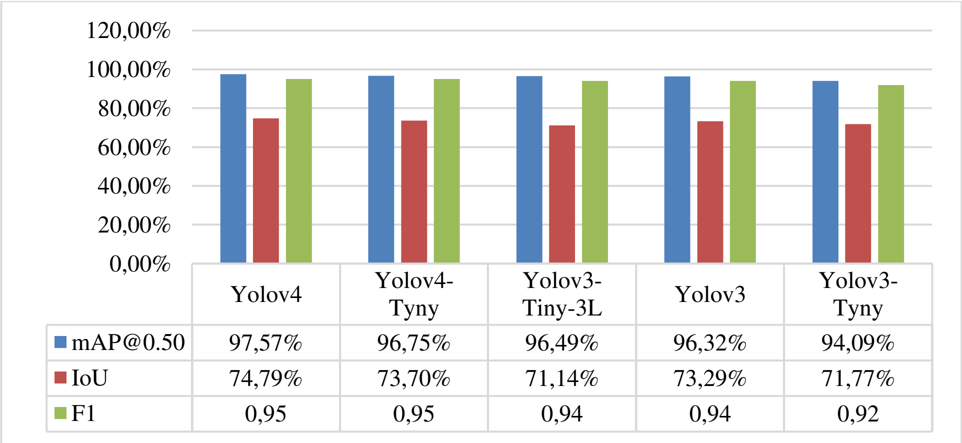


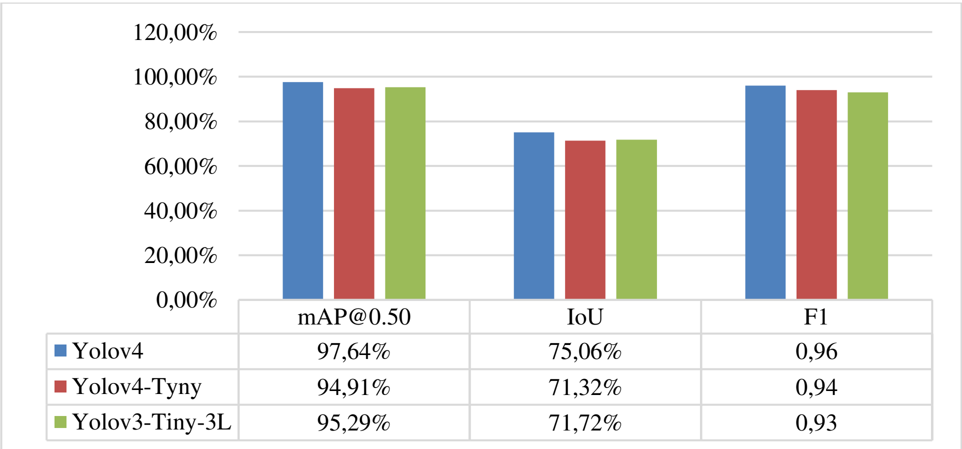
Figure 11. FPS Comparison Chart

From the validation results, that the comparison graph of mean average precision (mAP), Intersection over Union (IoU) and F1 scores for each model is as follows:



	Yolov4	Yolov4-Tyny	Yolov3-Tiny-3L	Yolov3	Yolov3-Tyny
mAP@0.50	97,57%	96,75%	96,49%	96,32%	94,09%
IoU	74,79%	73,70%	71,14%	73,29%	71,77%
F1	0,95	0,95	0,94	0,94	0,92

Figure 12. Comparison graph of validation results of each model for mAP, IoU and F1 values with max pooling



	mAP@0.50	IoU	F1
Yolov4	97,64%	75,06%	0,96
Yolov4-Tyny	94,91%	71,32%	0,94
Yolov3-Tiny-3L	95,29%	71,72%	0,93

Figure 13. Comparison graph of validation results of each model for mAP, IoU and F1 values with average pooling

From Figures 12 and 13 it is known that the highest mAP and F1 score for max pooling and average pooling is obtained by the YOLOv4 model. After validation, then the training result model is tested using images for each YOLO version model with a threshold value is 0.3, and the results are as the figure 14.

YOLOv4 (Max Pooling)

YOLOv4 (Average Pooling)

YOLOv4-Tiny (Max Pooling)

YOLOv4-Tiny (Average Pooling)

Yolov4-Tiny-3l (Max Pooling)

Yolov4-Tiny-3l (Average Pooling)

YOLOv3 (Max Pooling)

YOLOv3-Tiny (Max Pooling)

Figure 14. Test Prediction Result

After the training model results are tested with images, then testing is carried out using a webcam and real-time notifications. For this test, a simple application was developed using the python programming language with the following flow:

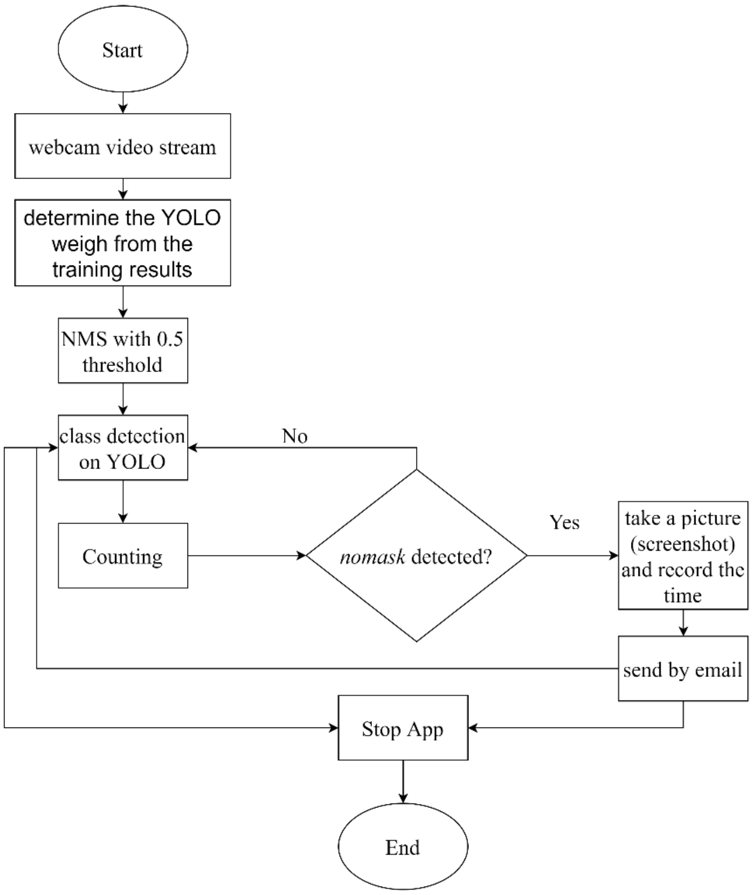


Figure 14. Flow Testing Real-time Detection

This testing process is carried out with each YOLO model from the training results of each version using the NMS Algorithm with a threshold value of 0.5. And here are the results of testing using core i7 9th gen hardware, 16GB ram with Graphic GTX 1650 and 1080p webcam.

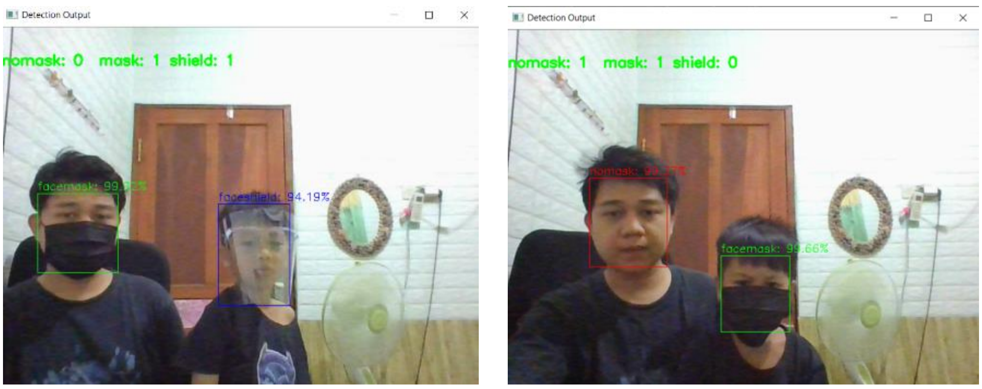


Figure 15. Real-Time Test Results

4. CONCLUSION

Based on the results of the study, it can be concluded that each model is quite good at detecting face mask and face shield objects, and the highest mAP value is obtained by the YOLOv4 model using average pooling with a value is 97.64% although the difference is not too much with YOLOv4 using max pooling with value 97.57%, and the lowest mAP value by the YOLOv3-Tiny model using max pooling with a value is 94.09%, the mAP value obtained from each YOLO model in training is also quite good because it is above 90%. And the highest FPS is obtained by the YOLOv4-Tiny model, which is 171 FPS with 96.75% mAP. Its because YOLOv4-Tiny model is smaller version from YOLOv4 and this means that Tiny-YOLOv4 is even less accurate from YOLOv4 because YOLOv4 get more mAP than Tiny-YOLOv4. When testing with real-time camera with webcam 1080p and using hardware core i7 9th, 16GB RAM with Graphic GTX1650, Tiny-YOLOv4 is accurate for face mask and face shield detection and quite fast but more accurate with YOLOv4 but the fps is lower than Tiny-YOLOv4.

REFERENCES

[1] L. Arnold, S. Rebecchi, S. Chevallier, and H. Paugam-Moisy, "An introduction to deep learning," *ESANN 2011 - 19th Eur. Symp. Artif. Neural Networks*, pp. 477–488, 2011, doi: 10.1201/9780429096280-14.

[2] M. S. Ejaz and M. R. Islam, "Masked face recognition using convolutional neural network," *2019 Int. Conf. Sustain. Technol. Ind. 4.0, STI 2019*, vol. 0, pp. 1–6, 2019, doi: 10.1109/STI47673.2019.9068044.

[3] R. Gopal, S. Kuinthodu, M. Balamurugan, and M. Atique, "Tiny object detection: Comparative study using single stage CNN object detectors," *2019 5th IEEE Int. WIE Conf. Electr. Comput. Eng. WIECON-ECE 2019 - Proc.*, pp. 2019–2021, 2019, doi: 10.1109/WIECON-ECE48653.2019.9019951.

[4] P. Adarsh, P. Rathi, and M. Kumar, "YOLO v3-Tiny: Object Detection and Recognition using one stage improved model," *2020 6th Int. Conf. Adv. Comput. Commun. Syst. ICACCS 2020*, pp. 687–694, 2020, doi: 10.1109/ICACCS48705.2020.9074315.

[5] R. Girshick, J. Donahue, T. Darrell, J. Malik, and U. C. Berkeley, "R-Cnn-Cvpr," 2012.

[6] R. Girshick, "Fast R-CNN," *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2015 Inter, pp. 1440–1448, 2015, doi: 10.1109/ICCV.2015.169.

[7] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, 2017, doi: 10.1109/TPAMI.2016.2577031.

[8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 779–788, 2016, doi: 10.1109/CVPR.2016.91.

[9] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 6517–6525, 2017, doi: 10.1109/CVPR.2017.690.

[10] J. Redmon and A. Farhadi, "YOLO v.3," *Tech Rep.*, pp. 1–6, 2018, [Online]. Available: https://pjreddie.com/media/files/papers/YOLOv3.pdf.

[11] F. Wu, G. Jin, M. Gao, Z. He, and Y. Yang, "Helmet detection based on improved YOLO V3 deep model," *Proc. 2019 IEEE 16th Int. Conf. Networking, Sens. Control. ICNSC 2019*, pp. 363–368, 2019, doi: 10.1109/ICNSC.2019.8743246.

[12] V. Suárez-Paniagua and I. Segura-Bedmar, "Evaluation of pooling operations in convolutional architectures for drug-drug interaction extraction," *BMC Bioinformatics*, vol. 19, pp. 92–101, 2018, doi: 10.1186/s12859-018-2195-1.

[13] M. Sun, Z. Song, X. Jiang, J. Pan, and Y. Pang, “Learning Pooling for Convolutional Neural Network,” *Neurocomputing*, vol. 224, pp. 96–104, Feb. 2017, doi: 10.1016/j.neucom.2016.10.049.

[14] H. Gholamalinezhad and H. Khosravi, “Pooling Methods in Deep Neural Networks, a Review,” 2020, [Online]. Available: http://arxiv.org/abs/2009.07485.

[15] Y. Mao and S. Lee, “Deep Convolutional Neural Network for Air Quality Prediction,” *J. Phys. Conf. Ser.*, vol. 1302, no. 3, 2019, doi: 10.1088/1742-6596/1302/3/032046.

[16] J. Redmon, “Darknet: Open source neural networks in c.” 2014, [Online]. Available: http://pjreddie.com/darknet.

[17] Y. Wang, Derui and Li, Chaoran and Wen, Sheng and Han, Qing-Long and Nepal, Surya and Zhang, Xiangyu and Xiang, “Daedalus: Breaking Nonmaximum Suppression in Object Detection via Adversarial Examples,” *IEEE Trans. Cybern.*, pp. 1–14, 2021, doi: 10.1109/TCYB.2020.3041481.

BIBLIOGRAPHY OF AUTHORS

Muhamad Muhaimin is a master's degree student in computer science at the President University, and also a senior programmer and analyst (AR, VR, MR and Interactive application). full resume can be seen on page https://www.muhamadmuhaimin.my.id/.

Mr. Tjong Wan Sen is a master's degree computer science lecturer at the President University.