YOLO FACE MASK DETECTOR

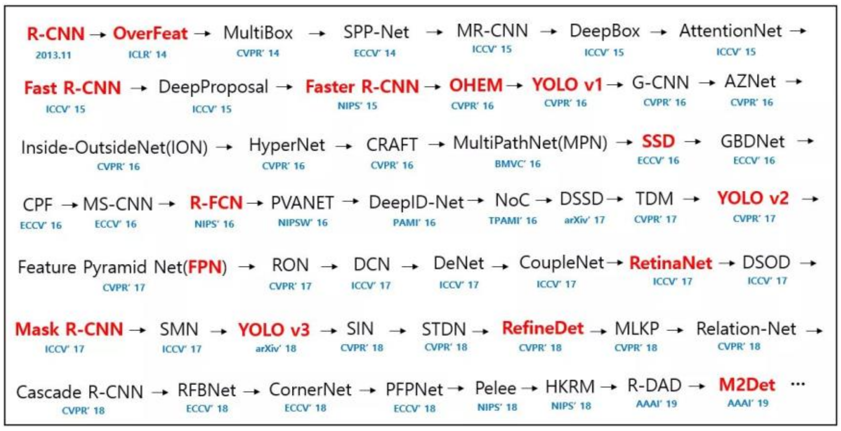
Github: https://github.com/zhukson/YOLO-Face-mask-detector

1. Introduction:

Since the outbreak of the Covid-19, the normal lives of billions of people around the world have been severely affected. large number of office workers are taking the risks of unemployment and wage reduction, companies are facing bankruptcy, students cannot return to school, and many people's lives have been ruthlessly plundered. To protect us from Covid, there are two effective and useful ways. The first is to get vaccinated, and the second is to wear a mask. Therefore, wearing masks should be every one's responsibility before entering into indoor places such as classroom. In this project, I designed YOLO real time face mask detector to remind those people who doesn't put their masks on to wear mask before they entering to the classroom which greatly ensured the safety of others. This project contains two subprojects, one is the implementation of two stage YOLO v1 model (Face YOLO V1 detector + Face mask CNN classifier) and the second is the application of YOLO V5 on face mask detection. Compared with the original YOLO V1, two stage model could gain better performance on the detection of smaller object. But YOLO V5 performs much better than Two-stage Yolo v1 because of the introduction of state-of-the art structure and techniques.

2. The model selection

Object detection, an important computer vision task, has been playing a significant role in wide range of fields from medical images detection **[1]** to vehicle detection **[2]**. The main object of object detection is to build models to provide the location and label information, in other word, the answer to "What object are there?" required by different tasks. In recent years, the rapidly development of deep learning has brought enormous of research attentions into the object detection and lead to a significant breakthrough in it. From Fig.1, we can see there are many good and well performed models been invented each year since the appearance of model R-CNN on 2013.11. However, the selection of the most suitable model for different tasks has become a tough problem. In this project, I compared two most famous models (Faster RCNN **[4]** with YOLO v1 **[3]**) to see which one is more suitable for this task.



R-CNN 2013.11 → OverFeat ICLR' 14 → MultiBox CVPR' 14 → SPP-Net ECCV' 14 → MR-CNN ICCV' 15 → DeepBox ICCV' 15 → AttentionNet ICCV' 15 →
Fast R-CNN ICCV' 15 → DeepProposal ICCV' 15 → Faster R-CNN NIPS' 15 → OHEM CVPR' 16 → YOLO v1 CVPR' 16 → G-CNN CVPR' 16 → AZNet CVPR' 16 →
Inside-OutsideNet(ION) CVPR' 16 → HyperNet CVPR' 16 → CRAFT CVPR' 16 → MultiPathNet(MPN) BMVC' 16 → SSD ECCV' 16 → GBDNet ECCV' 16 →
CPF ECCV' 16 → MS-CNN ECCV' 16 → R-FCN NIPS' 16 → PVANET NIPSW' 16 → DeepID-Net PAMI' 16 → NoC TPAMI' 16 → DSSD arXiv' 17 → TDM CVPR' 17 → YOLO v2 CVPR' 17 →
Feature Pyramid Net(FPN) CVPR' 17 → RON CVPR' 17 → DCN ICCV' 17 → DeNet ICCV' 17 → CoupleNet ICCV' 17 → RetinaNet ICCV' 17 → DSOD ICCV' 17 →
Mask R-CNN ICCV' 17 → SMN ICCV' 17 → YOLO v3 arXiv' 18 → SIN CVPR' 18 → STDN CVPR' 18 → RefineDet CVPR' 18 → MLKP CVPR' 18 → Relation-Net CVPR' 18 →
Cascade R-CNN CVPR' 18 → RFBNet ECCV' 18 → CornerNet ECCV' 18 → PFPNet ECCV' 18 → Pelee NIPS' 18 → HKRM NIPS' 18 → R-DAD AAAI' 19 → M2Det AAAI' 19 ...

Fig.1 the revolution of Object detection models

Yolo v1 is a one-stage model and the structure of yolo v1 is very simple and straight forward. From Fig.2, we can see the model of yolo v1 is constructed of 24 convolutional layers and 2 fully connection layers. Where convolutional layers are used to extract features, max-pooling layers is used to lower the parameters for the purpose of preventing overfitting and fully connection layers are used to make prediction. The activation function Leaky Relu is combined with each convolutional layer. Different with the Relu activation function, which is to set all negative values into zero, Leaky ReLu multiplied all negative values with a small positive constant $Leaky \in$ (0, 1]. By doing so, partial information of the negative values could be preserved. The input of YOLO model is an image with size 3(RGB 3channels) *48*48 and the output are a Tensor with size 30*7*7. Where 7*7 means the model divide the input image into 7*7=49 grid cell, each grid cell generates 2 bounding boxes. The 30 dimension is composed of 3 parts (5+5+20), where the first two 5 dimension represents the x, y, width, height, and the value of confidence (1 represents there is an object in this grid cell, 0 otherwise). The last 20 dimensions represents the classes. This is achieved by the word-embedding technique one-hot encoding. In this task, there are three classes in total (wear mask, improperly wear mask, no mask), therefore, the output should be 7*7*(5+5+3). The main benefit of YOLO V1 over other model is very straight forward: It has a fast inference speed due to the simplicity of its structure which could meet the requirement of real time detection. Also, the model of YOLO V1 is easy to implement compared with other multi-stage models. However, YOLO V1 performs bad on small objects as the size of feature maps is keep decreasing through the process of max pooling. Second, YOLO v1 performs very bad on crowed objects. The main reason for this problem is that each grid cell would only generate two bounding boxes which means YOLO v1 could only predict 7*7*2 = 98 objects in a single image, and 2 objects in one grid cell. Therefore, if an image contains objects larger than 2 in a single grid cell, YOLO V1 would fail in this task. Third, each grid cell only responsible for the prediction of a single class, which means YOLO V1 would also fail in the task when an image contains two classes of objects in a single grid cell.

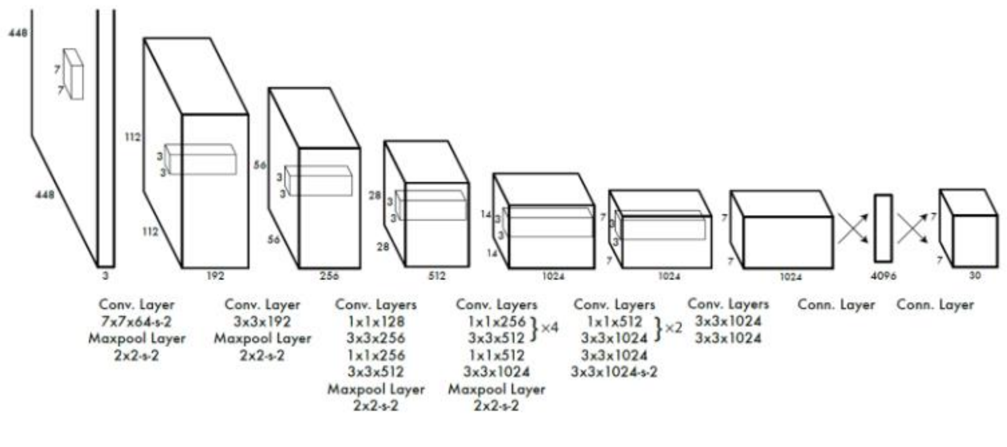


Fig.2 the construction of YOLO V1 model

After the analysis of YOLO V1 model, I then compared it with Faster RCNN. Faster RCNN is a two-stage model, the main procedure is as follows: 1. put the whole images into CNN model to get the feature maps. 2. feature maps go into rpn and generate anchors with number of (width x height x channel x k). There are two layers in rpn, one is the classification layer, and the other is regression layer. Classification layers apply softmax function to get the confidence of positive or negative of anchors. Then the anchors with higher confidence of positive would go into regression layer, and the regression layer would apply bounding regression to the positive anchors and get the proposals. 3.the Proposals and feature maps we got in step 1 would then go into the r-cnn head together. First, they would go into the roi pooling in r-cnn head which would return proposal feature maps based on Proposals and feature maps. 4.Finally, proposal feature maps go into the last two branches of r-cnn head. One is the classification branch, which return the confidence of classes based on fully connected layer + softmax. The other one is the Regression branch which apply bounding box again to get the better bounding box. Here only the bounding box with the highest confidence would be reserved. The benefits of Faster RCNN over YOLO v1 is also very straight forward: Faster RCNN performs better than YOLO V1 on detecting small objects and crowed objects. The deficiency is also obvious: the structure is too complicated compared with YOLO V1's model, leading to the slow inference speed.

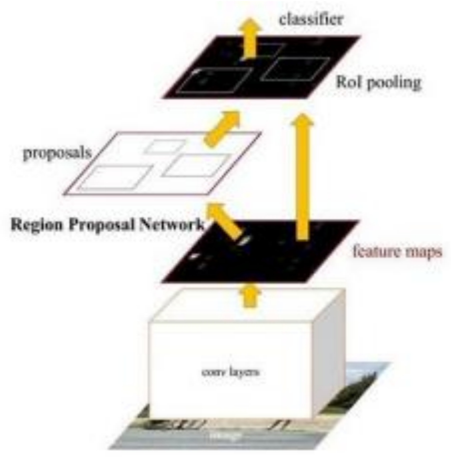


Fig3. The construction of Faster RCNN model

For this task, detecting whether a student wearing face mask or not doesn't require a very high precision compared with other tasks such as the medical detection, also the object for each image should not be very much (Usually student comes to class one by one). Therefore, the real time advantage of YOLO V1 should count more than the other two advantages of Faster RCNN. Also, it is too complicated to implement the Faster RCNN from scratch, this is another important reason why I choose YOLO V1 for this task. However, there is a fatal problem that face mask is a small object and my implemented YOLO V1 performed bad on this task. Therefore, we designed an improved YOLO V1 model for this task which is called two-stage YOLO v1. Two-stage YOLO V1 is combined with two models, YOLO V1 and CNN classifier. Since face is a much larger object compared with face mask (3 times larger), we decided to use YOLO V1 to detect the face and then train another classifier to determine whether this face contained a face mask or not. After the implementation of this model and gained a satisfied result, we then tried this task in YOLO v5, the latest, strongest, and user-friendly model in YOLO series. YOLO v5 used the Pytorch framework instead of the darknet framework used in YOLO v4 which makes it very convenient for us to train the model in our own dataset for the purpose of different tasks.

The following sections would introduce you with more details of the construction of this Two stage model and how YOLO v5 evolved from YOLO v1,

3. The Two-stage YOLO v1

3.1 The inference of yolo v1

In the previous section, we have almost done with the explanation of the YOLO v1 model. YOLO V1 is a one stage model constructed with 24 convolutional layers for feature extraction and 2 linear layers for the classification task. The output of the YOLO v1 is a matrix with the size of [num_grid x * num_grid y * num_boxes * (5+num_classes)] where "5" represents the location information: [center x, center y, width, height, confidence]. Here are some extra and necessary information I would like to mention in this process. For a single image as an input, the YOLO v1 would generate 7*7*2 = 98 bounding boxes. However, only a few of them are our target bounding boxes and most of them should be removed. Therefore, in this case we must use non-maximum suppression, so called "NMS", to get rid of most of the unrelated bounding boxes. The procedure of NMS is simple. First, we need to find the best bounding boxes for each class based on the confidence value. Second, we calculate the IOU score between other bounding boxes and the target bounding boxes, if the IOU score is higher than a specific threshold (in my project I set it to 0,7), the bounding box would be removed. Third, we need to identify another threshold which is used to filter those bounding boxes which survived after the IOU comparison and have a confidence value lower than this threshold. In other word, the third step is to remain those bounding boxes whose confidence value is higher than the threshold. Fig.4 shows the effect of NMS, we can see it worked very well.

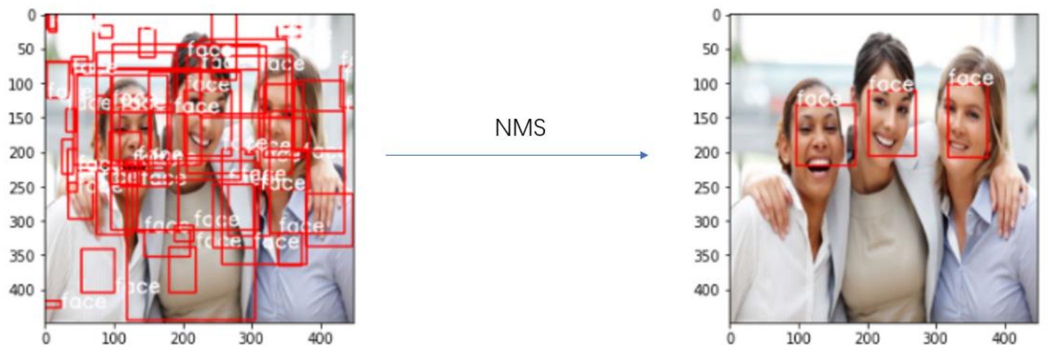


Fig4. The effect of NMS

3.2 The training of yolo v1 （data preprocessing + loss function）

To train a yolo v1 model from scratch, we need to do two things. First, we have to prepare our dataset. Second, we have defined the loss function and use gradient decent to do optimization.

3.2.1 data preprocessing

In this project, I only have limited number of data (700 images). Therefore, I labeled 1400 more images by myself using online application called Roboflow. The extraction of the location information from the xml annotation files is a very important step. First, we need to extract the location information of the upper left corner and the lower right corner. Then, we have to translate that information into the required format of YOLO V1: [center x, center y, width, height, confidence value]. Here I set the confidence value into 1 because this is the ground truth box and it do contain object.

3.2.2 loss function

$$
\begin{array}{lll}
 & \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] & \text{Center error} \\
+ & \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2 \right] & \text{width and height error} \\
+ & \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left(C_i - \hat{C}_i\right)^2 & \text{obj Condifence error} \\
+ & \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 0_{ij}^{obj} \left(C_i - \hat{C}_i\right)^2 & \text{noobj Condifence error} \\
+ & \sum_{i=0}^{S^2} 1_{i}^{obj} \sum_{ceclasses} (p_i(c) - \hat{p}_i(c))^2 & \text{classification error}
\end{array}
$$

The loss function may look a little bit confusing at the very first glance, but it is very straight.

Center loss is to calculate the center error between the box responsible for this task and the ground truth label. where λ_{coord} is the weight, S^2 stands for the number of the grid cells, B stands for the number of boxes each grid cell generates, 1_{ij}^{obj} stands for the j bounding boxes in the i grid cells containing object. Width and height loss is to calculate the width and height error between the box responsible for this task and the ground truth label. Obj confidence error is to maximize the confidence value of target bounding boxes and the noobj confidence error is used to minimize the confidence value of unrelated bounding boxes. In the original paper, this error is to compute the difference between the confidence value and its IOU score with the ground truth box. However, if we don't have a very big dataset and a long training epoch, it is very hard to ensure the confidence score of those target boxes higher than the confidence score of those unrelated boxes. Therefore, I modified this function to instead compute the difference between the confidence value of target bounding box with 100 and the difference between the confidence value of unrelated bounding box with 0 to increase the difference of confidence value between target and unrelated bounding boxes.

3.3 Alex net classifier

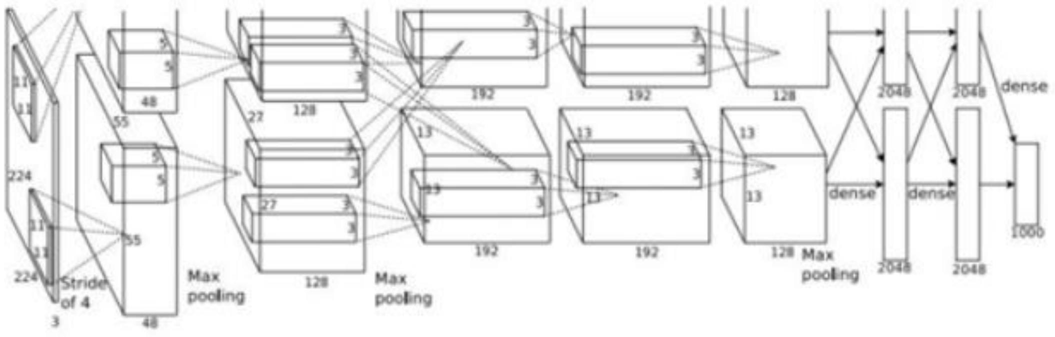


Fig5. ALEX Net classifier

Alex Net **[5]** was created by the doctoral student of Geoffrey Hinton and won the 2012 ImageNet competition. It is really a milestone in deep convolutional neural networks. Before Alex net was invented in 2012, this field had been silent for a very long time. The only usable convolutional neural network before the invention of Alex Net is LeNet-5 created in 1986. Compared with Lenet5, this model has 6 major contributions: 1. introduced the dropout method in the Linear Layer to prevent overfitting. 2. introduced ReLU as the activation function instead of the Sigmoid Function used in LeNet-5 to prevent the gradient disappear. 3. introduced the Data Augmentation method to help us get more data and thus prevent the overfitting. If our training data is very scare, we could use data augmentation to increase the size of our data thus preventing overfitting. 4.using CUDA to train the model to speed up the running time. From Fig5, we can see the author distribute the total parameters into two GTX580 GPU because the ram capacity of GTX580 is not enough. 5. Using Max pooling as the pooling layer instead of the average pooling because the blurring effect of average pooling. 6. Introduced LRN layer to enhance the generalization ability of model. In this project, I used a separate face mask classification dataset and did many data augmentation in this dataset to help this model be less sensitive to the cropped face image.

3.4 the whole procedure

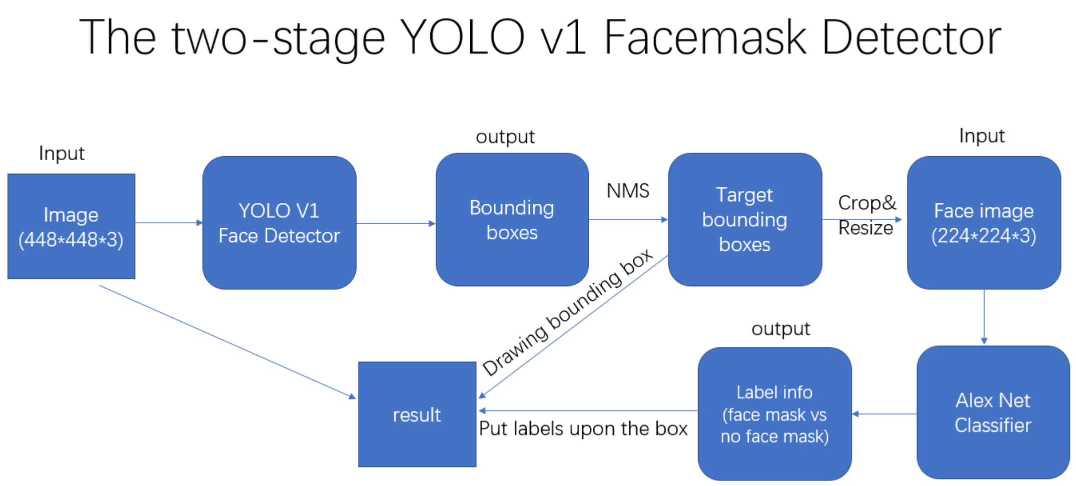


Fig6. The two-stage YOLO v1 face mask detector

Fig6. Shows the main procedure of this two-stage model. First, we put an image with the required input size into the pretrained YOLO v1 face detector, and this model would output many bounding boxes. Then we use NMS as a filter to remove those unrelated bounding boxes. After that, we cropped the face portion of the original image based on the location information provided by target bounding boxes and resize these cropped images into required input size of Alex Net. After we put this image into the pretrained Alex Net, this model would output the label information (face mask vs no face mask). Finally, we could combine the original image, the target bounding boxes and the label information to get the result.

4. YOLO V5

YOLO V5 is the latest and the strongest version in the YOLO series. To understand how YOLO v5 works, it is indispensable to know how YOLO v5 was evolved from YOLO v1.

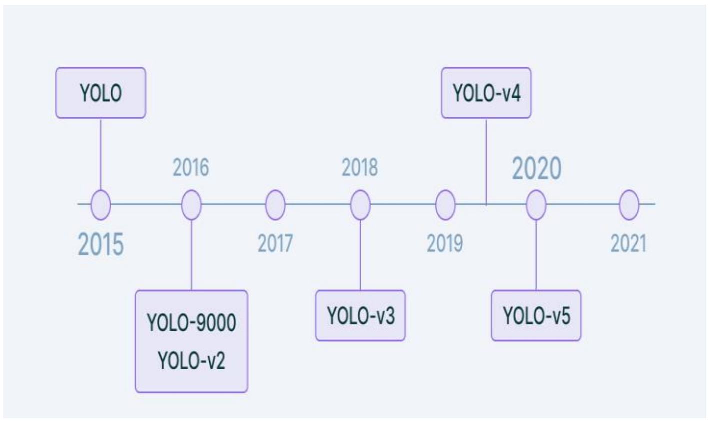


Fig.7 the evolution of YOLO V5 model

4.1 YOLO V2

YOLO V2 **[6]** introduced prior anchor boxes instead of the automatically generated bounding box to make prediction and the number and size of anchor boxes are pre-determined by using the result of k-means algorithm. The main benefit of the prior anchor boxes over the randomly generated bounding boxes is very straight forward: each anchor box has its own responsibility for predicting specific shape of object. From the left image in Fig.8, we can see the responsibility of tall anchor box is to predict tall object such as human and the responsibility of flat anchor box is to predict flat object like the car.

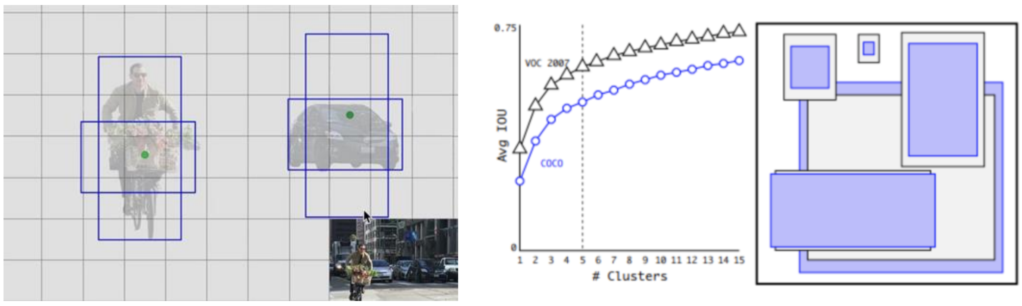


Fig.8 prior anchor boxes and the result of k-means algorithm

4.2 Yolo v3

YOLO V3 **[7]** played a very significant role in the evolution of YOLO series. The introduction of the structure of "Backbone+Neck+Head" largely improved the performance of YOLO v2 and laid the foundation for later models. The backbone is used to extract features, Neck is used to collect feature maps from different stages. In YOLO v3, the neck is constructed of Featurized image pyramid (FPN). Fig.10 shows how FPN works. First, it does down sampling to the input image, and we could get feature maps from different stages with different scales. After that, it would do up sampling to the smallest feature map at the top. It is like the inverse process of the down sampling. But the difference is, this model concatenates the feature maps from the previous stage with the feature maps from the current stage to construct a more informative feature map. The last part is Head, which is used to do dense predictions. For example, the smallest feature map would be used to predict small object, the large feature map would be used to predict large object. After that, we could combine all these anchor boxes together, do non maximum suppression to them, and get the target anchor boxes.

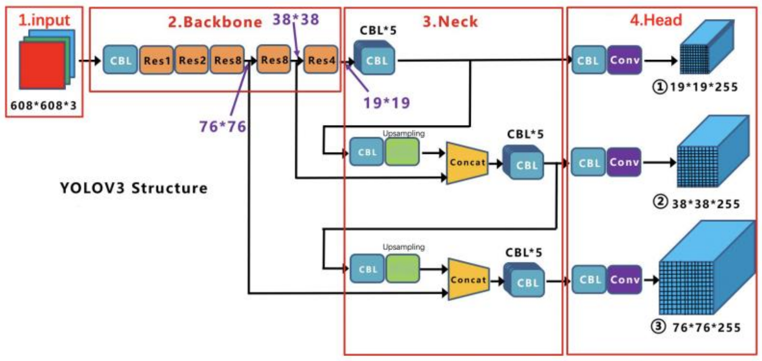


Fig.9 the YOLO v3 structure

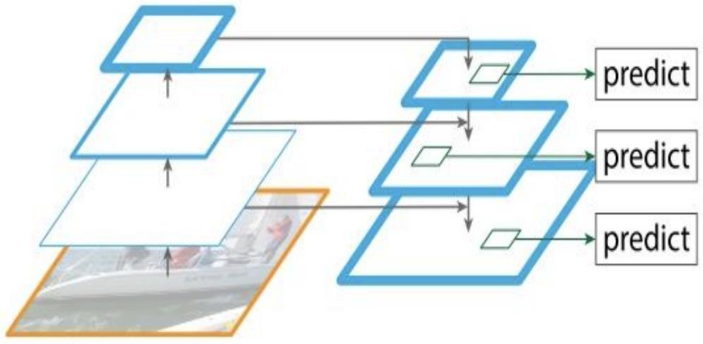


Fig.10 The featurized image pyramid (FPN)

4.3 Yolo v4

There are three types of innovation in scientific research: the first is complete innovation, just like God created the world, this one is the most difficult. For example, yolo v1 is this kind of innovation. The second is to make very significant change on the existing model, such as YOLO V3 which replaced the entire model structure of YOLO v2, resulting in a dramatic improvement on the detection of small and crowed objects. The third innovation is to optimize all aspects of an existing model after tens of thousands of experiments without making significant changes (such as the hyperparameter tuning of random forest), leading to a great leap-forward improvement of the model. YOLO V4 **[8]** is this kind of model. The structure of YOLO v4 is very similar to the structure of YOLO V3, but it improves each of the three parts of this structure, lead to the significantly improvement of performance. Here I would like to list one of the most famous improvements of YOLO V4: the introduction of mosaic data augmentation in the input. This is the first time that YOLO series introduce data augmentation into the model. From the Fig.11, we can see Mosaic data augmentation is to concatenate four images into one image. There are two main benefits. First, this could Make the model learn to recognize objects in a smaller range which further enhanced the model's performance on detecting small object. Second, it significantly reduces the demand for batch-size thus decrease the demand for GPU resources.

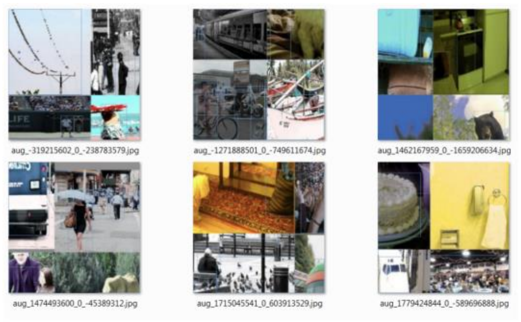


Fig.11 mosaic data augmentation

4.4 YOLO v5

YOLO v5 is released only one months after the releasing of YOLO v4. Therefore, the YOLO v5 is very similar to YOLO v4. However, there are several main differences. First, YOLO V5 Used the Pytorch framework instead of the Darknet which is very user-friendly and can easily be trained on our own data sets. Second, YOLO V5 Requires significant less time for training than YOLO v4. Third, the inference speed of YOLO V5 model is incredibly fast: it can process 140 images in one second. (140 FPS)

5. Demo

The predictions results of two-stage YOLO v1 and YOLO v5 are showed alternatively in Fig.12 and Fig.13. we can see YOLO v5 performed very well on detecting small and crowed object.

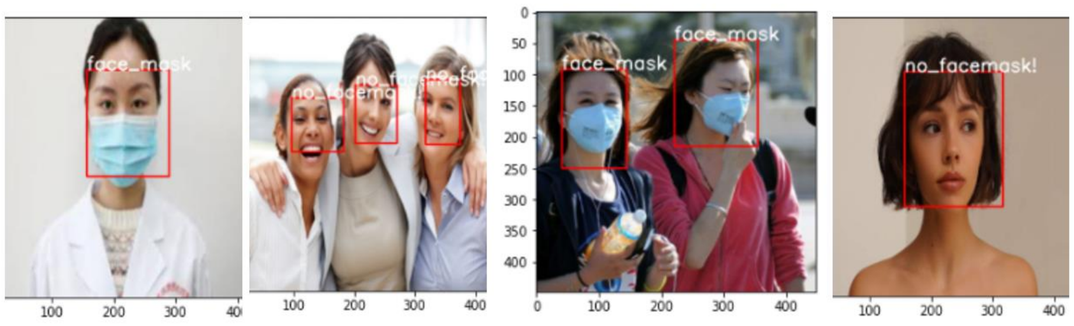


Fig.12 The prediction result of the two-stage YOLO v1.

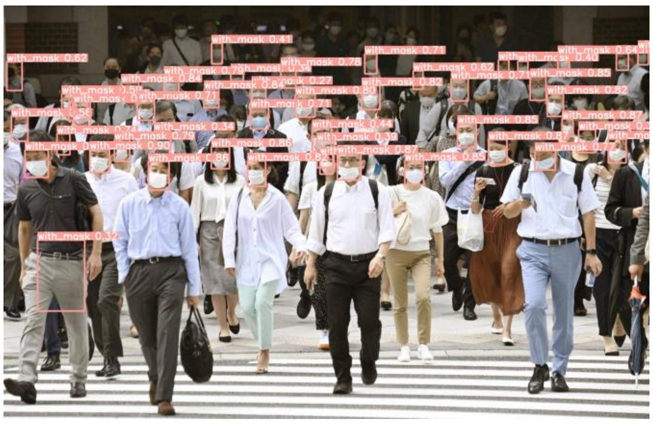


Fig.13 The inference result of YOLO v5

6. The Comparison between two models

6.1 the training time:

I trained Two-stage YOLO v1 and YOLO v5 on the same device based on the same amount of data and time. From Fig.14, we can see YOLO v5 only takes half of the time to finish training compared with Two-stage YOLO v1.

	Two Stage YOLO V1(600 images 60 epochs)	Yolo v5 (600 images 60 epochs)
Time	1 hour	30 minutes
Device	GPU	GPU

Fig.14 The training time used by Two-stage YOLO v1 and YOLO v5 based on the same condition

6.2 the input image requirement:

From Fig.14, we can see YOLO v5 required significantly smaller number of data to finish the training task.

	Two Stage YOLO V1	Yolo v5
Input data	2000 for face detection and 4000 for CNN classification.	600 images
Device	GPU	GPU

Fig.14 The training data used by Two-stage YOLO v1 and YOLO v5 based on the same condition

7. Conclusion

YOLO V5 performs much better than two-stage YOLO V1 in almost every aspect including the training time, training image required, inference time and ability to handle small objects.

This is reasonable because YOLO v5 has endured so many improvements and revolutions Since YOLO v1, and the two-stage YOLO V1 is only an enhanced version of YOLO V1. There are still many tasks we can try to conquer to improve the performance on this two-stage YOLO V1 model. First, we can implement some temporal smoothing to this model to decreasing the flickering effect. This could be done by putting the weight of the new detection in relation to the shift of the bounding box compared to the previous one. Second, we can add another class "incorrectly wearing mask" while doing the classification task since incorrectly wearing mask is also very harmful for our health.

8. Acknowledgement

Thanks to Dr. Liang for giving me valuable instructions and providing me with informative ideas on designing two stage models.

References:

[1] Li, Zhuoling, et al. "CLU-CNNs: Object detection for medical images." *Neurocomputing* 350 (2019): 53-59.

[2] Mao, Qi-Chao, et al. "Finding every car: a traffic surveillance multi-scale vehicle object detection method." *Applied Intelligence* 50.10 (2020): 3125-3136.

[3] Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

[4] Ren, Shaoqing, et al. "Faster R-CNN: Towards real-time object detection with region proposal networks." *Advances in neural information processing systems* 201 (2015).

[5] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks." *Communications of the ACM* 60.6 (2017): 84-90.

[6] Redmon, Joseph, and Ali Farhadi. "YOLO9000: better, faster, stronger." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.

[7] Redmon, Joseph, and Ali Farhadi. "Yolov3: An incremental improvement." *arXiv preprint arXiv:1804.02767* (2018).

[8] Bochkovskiy, Alexey, Chien-Yao Wang, and Hong-Yuan Mark Liao. "Yolov4: Optimal speed and accuracy of object detection." *arXiv preprint arXiv:2004.10934* (2020).