Pure Augmented Data on Object Detection — Predicting Majong Tiles in Real Games with YOLOv7

Chun Hua, Lin
cl4335@columbia.edu

1. Introduction

Object detection is a computer vision task wildly used nowadays whose goal is to detect the object of certain predefined classes in images or videos — well-researched applications including face recognition, and license plate recognition. This project focuses on object detection tasks applied to Mahjong. Mahjong is a tile-based game played by four players. The game is played with a set of 144 tiles based on Chinese characters and symbols, with a total of 42 different tiles, although many regional variations may omit some tiles or add unique ones. Keeping track of the tiles played allows players to obtain information to decide their next play.

The problem of small object detection in computer vision involves accurately identifying small objects in a video feed or image; where small objects are not as accurately detected as larger objects. It is important to note that the objects themselves do not necessarily need to be physically small, but rather small relative to the size of the video frame or image. For example, in aerial computer vision, small object detection is crucial because objects in the frame will appear small relative to the overall size of the image. To accurately identify these objects, a small object detection model must be able to effectively detect and recognize objects that are small in the frame. In this project, I aim to address the issue when detecting the tiles played, overlapping, and in a jumble, which sometimes results in unusable results, as well as low confidence in the predicted bounding box.

In addition, Data augmentation is a technique used to generate new images from a base dataset. This can be particularly useful for small object detection, as it can help to prevent overfitting to the training set. Some effective augmentations for small object detection include random crop, random rotation, and mosaic augmentation. These augmentations can increase the diversity of the training set, leading to better performance on the test set. By using data augmentation, it is possible to improve the accuracy of a small object detection model.

At last, it is impossible to capture every possible real-world scenario that a model may be asked to identify during inference. This is where data augmentation can be useful. In this project, I wanted to experiment with using a purely augmented dataset to train a model that could perform well in a real-world scenario. To do this, I built a tile-detecting model based on the YOLOv7 neural network, a state-of-the-art model for object detection. I then created a completely augmented dataset that contained no real-world scenario images and used it to train the model. By using a purely augmented dataset, I was able to evaluate the effectiveness of data augmentation in improving the performance of a small object detection model.

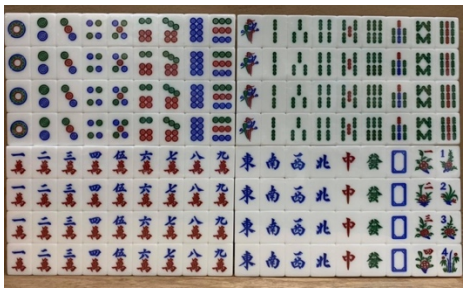

Fig. 1. Example showing typical Majong tiles, with a total of 42 classes, 144 tiles. (https://en.wikipedia.org/wiki/Mahjong_tiles#/media/File:MJTile s_Fullset. png)

Fig. 2. A real Majong game contains 4 players, where tiles are played in the middle of the tile pool. (iStock/Getty Images)

2. Related Works

2.1. Data augmentation

In a real-world scenario, our target may exist in a variety of conditions, such as different orientations, locations, scales, brightness levels, etc. A dataset of images taken under a limited set of conditions has its limitation to represent real-world cases inferencing.

The authors of YOLOv4 have made improvements to the data management and data augmentation in the training pipeline. These techniques help to increase the size and diversity of the training set, exposing the model to scenarios that it would not have seen otherwise. According to the authors, "The purpose of data augmentation is to increase the variability of the input images, so that the designed object detection model has higher robustness to the images obtained

(a) Original images (b) Augmented images (c) Mosaic augmented data

Fig. 3. (a) Images of Majong tiles with the bounding box. Note that the image only contains the tiles themselves, no background or other elements are presented in each picture. (b) The data after augmentation applied. (c) An exmaple of the Mosaic augmentation applied to the data.

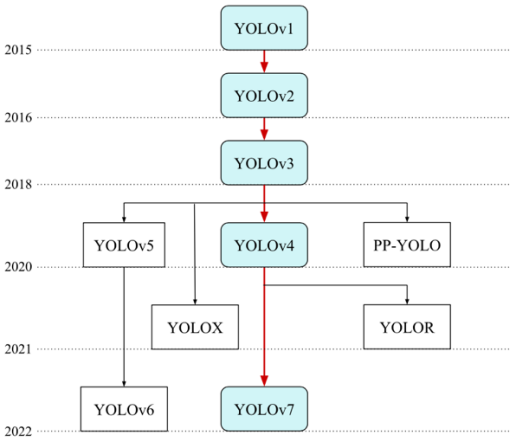

Fig. 4. The roadmap of YOLO models. The red arrows indicates the offical releases of the YOLO family.

from different environments." [1]. The YOLOv7 model builds on this approach, incorporating data augmentation into the training process. Data augmentation is an important part of computer vision, and state-of-the-art research continues to support its importance. By using data augmentation, we can improve the performance of our object detection models and make them more robust to a wide range of conditions.

2.2. YOLOv7

The YOLO algorithm uses convolutional neural networks (CNN) to detect objects in real time. The name YOLO stands for "You Only Look Once," indicating that the algorithm only requires a single pass through the neural network to make predictions. CNN is used to predict class probabilities and bounding boxes for objects in the image. This allows YOLO to detect and locate objects simultaneously. This makes it a single stage detector.

The YOLOv7 model is the latest in the YOLO family (2022). The YOLOv7 model surpasses previous versions by inferring faster and with greater accuracy. YOLO first uses a backbone network to extract features from the input image. These features are then combined and processed in and passed to the head, where the final bounding box predictions are made.

3. Methodology

In this work, I created a completely augmented dataset that contained no real-world scenario images, then fed it into the YOLOv7 to train the model. Last, we compare the difference between different hyperparameters.

3.1. Dataset

For the dataset, I gathered 628 images of different sets of images of Majong tiles, which shows only the tile itself, one per image (see Fig. 4(a).). Intuitively, this type of dataset is better for classification tasks as it reduces possible noises of the image. Correspondingly, the real-world scenario is full of noises and variables. Such dataset would be insufficient to represent scenes the model might receive during inference. Therefore, data augmentation is applied to create additional images that simulate the various conditions and variations that the model may encounter in the real world (a total of 1628 images). For example, the augmented dataset could include images of tiles that are rotated, flipped, scaled, or taken under different lighting conditions. This would help the model to better generalize to real-world scenarios and improve its performance on the target task (see Fig. 4(b).).

The following augmentations are used to generate the data:

- Flip: Horizontal, Vertical
- Rotation: Between -45° and +45°
- Shear: ±45° Horizontal, ±44° Vertical
- Cutout: 1 box with 20% size each

While common data augmentation techniques such as flips and rotations help improve the performance of object detection models, the YOLO family employs an additional approach called Mosaic augmentation. First introduced in YOLOv4, the technique involves creating a mosaic of multiple images and using it as the input to the network during training. This can help the model to better generalize to real-world scenarios and improve its performance on the target task (See Fig. 4(c).). Other augmentations are also applied alongside the Mosaic augmentation method (see Fig. 6.). Combined we have the final image for training the model.

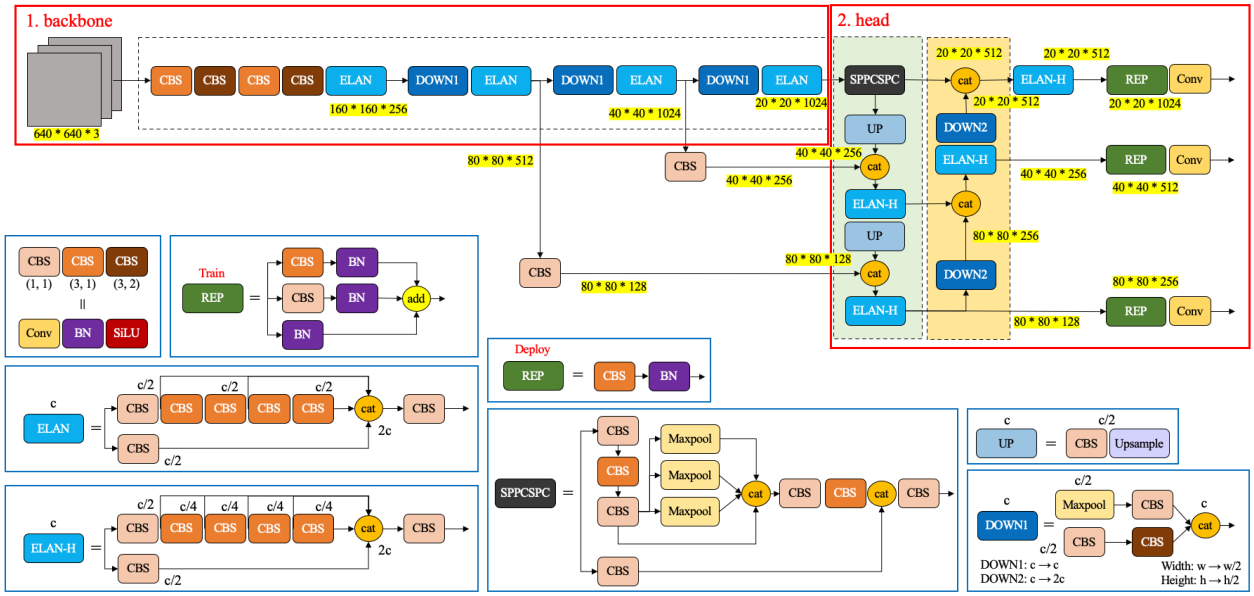

Fig. 7. The architecture of YOLOv7. The red box indicates the main structure of the model, and the blue box indicates the components included in the main model. The components are mostly built with the CBS unit. Different colors mean different (kernel size, stride). (Inspired by: https://zhuanlan.zhihu.com/p/543743278)

```
hsv_h: 0.015  # image HSV-Hue augmentation (fraction)
hsv_s: 0.7  # image HSV-Saturation augmentation (fraction)
hsv_v: 0.4  # image HSV-Value augmentation (fraction)
degrees: 0.0  # image rotation (+/- deg)
translate: 0.0  # image translation (+/- fraction)
scale: -1.0  # image scale (+/- gain)
shear: 0.0  # image shear (+/- deg)
perspective: 0.0  # image perspective (+/- fraction), range 0-0.001
flipud: 0.5  # image flip up-down (probability)
fliplr: 0.5  # image flip left-right (probability)
mosaic: 1.0  # image mosaic (probability)
mixup: 0.15  # image mixup (probability)
copy_paste: 0.0  # image copy paste (probability)
paste_in: 0.15  # image copy paste (probability), use 0 for faster training
```

Fig. 6. Hyperparameters of data augmentation methods in YOLOv7. Notice the sclae is set to -1.0 to address the small obeject problem given our dataset is fully built with images of tiles itself.

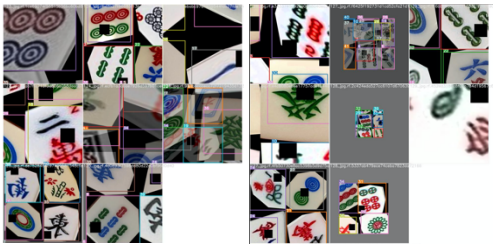

Fig. 7. The final images to train the model. This should at least mimic the complexity if real-world scenarios.

3.2. Model

As mentioned in previous sections, YOLOv7 is used as the model for this work. YOLOv7 authors made several changes to the YOLOv4 network. Different from YOLOv4, YOLOv7 consists of a backbone and a head, instead of the classic backbone, neck, and head architecture. The backbone extracts feature maps from the input image, the feature maps are then passed to the head where the feature pyramid is built and later used for detection outputs. A basic component of the YOLOv7 network includes CNN, Batch Normalization, and SiLU activation function (see Fig. 7.). We call it the **CBS** unit for simplification.

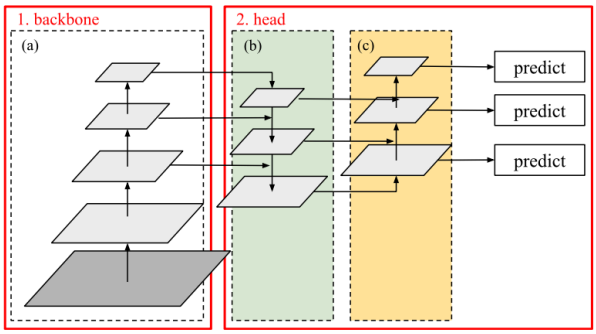

Fig. 8. A simple data flow chart of the YOLOv7. (a) Indicates the FPN backbone bottum-up pathway. (b) Indicates the FPN top-down pathway (c) Indicates the PANet bottom-up path augmentation pathway.

The following components are then proposed and contributed to the SOTA performance of YOLOv7:

- **MPConv**: A structure that downsamples the input feature maps with 2 branches. (See Fig. 7.)
- **ELAN**: "Efficient Layer Aggregation Network". The novel structure the authors of YOLOv7 proposed. "By controlling the shortest longest gradient path, a deeper network can learn and converge effectively." [2] It allows the model to learn more diverse features for better learning. (See Fig. 7.)
- **SPPCSPC**: "Spatial Pyramid Pooling Cross-Stage-Partial Conv". The SPP is a pooling layer that removes the fixed-size constraint of the network [3]. The CSP component of the SPP-CSP separates the feature map of the base layer into two parts. One part is processed through a dense block and a transition layer, while the other part is combined with the transmitted feature map to

the next stage. This truncation of gradient flow helps to prevent an excessive amount of duplicate gradient information from being transmitted [4]. In YOLOv7, The SPPCSPC has four different scales of maximum pooling, with four different receptive fields, to distinguish between large and small targets. (See Fig. 7.)

- **REP**: "Re-Parameterized Convolution". It takes advantage of the benefits of a multi-branch model in training (high performance) and a single-path model in inference [5]. (See Fig. 7.)

Finally, the architecture of YOLOv7 is built with the components mentioned above (See Fig. 8.). To sum up the model, the detector is composed of several parts as follows:

- **Backbone**: CBS, MPConv, ELAN
- **Head**: SPPCSPC, ELAN, MP, REP

Note that the head is built based on the Path Aggregation Network (PANet) structure which is a modified version of the Feature Pyramid Network (FPN). The FPN includes the bottom-up pathway and the top-down pathway. The bottom-up pathway is done by the backbone, extracting feature maps. The top-down pathway then allows the semantic values to get passed down to higher-resolution feature maps, along with lateral connections between reconstructed layers and the corresponding feature maps helping the detector to predict the location betters [6]. It also acts as skip connection to make training easier. Finally, the Bottom-up Path Augmentation, new pathway added by PANet, shortens the path by using clean lateral connections from the lower layers to the top ones, called the "shortcut" [7].

4. Result

4.1. Setup

I implemented our system in Google Colab. Google Colab runs GPU with up to Tesla K80 with 12 GB of GDDR5 VRAM, Intel Xeon Processor with two cores @ 2.20 GHz, and 13 GB RAM. I used the official open-source WongKinYiu/yolov7 repository to implement the YOLOv7 model. I used the yolov7.yaml model setup with no modifications. The focus of this work is to exploit data augmentation methods. Therefore, the changes are only made to hyperparameters to validate the study.

4.2. Experimental result

The metric used is precision, recall, and map. Recall is a measure of the proportion of relevant items that the model was able to correctly identify. A model with a high recall can identify a large proportion of the relevant items, even if it also identifies some irrelevant items as being relevant. Precision, on the other hand, is a measure of the proportion of items that the model identified as being relevant that are relevant. A model with high precision can accurately identify only the relevant items, even if it misses some of the relevant items. AP (Average precision) is a popular metric in measuring the accuracy of object detectors.

Average Precision (AP) is a metric that measures the model's precision and recall for a given class. To calculate AP, the model's precision and recall are first calculated at various points along the range of possible detection scores (e.g. the model's confidence in its predictions). The confidence score is calculated by a numeric representation of the bounding box multiplied by the Intersection over Union (IoU):

$$IoU = \frac{Area\ of\ Intersection}{Area\ of\ Union} \quad (1)$$

$$Conf = \Pr(Class\ i|Object) * \Pr(Object) * IoU \quad (2)$$

"If no object exists in that cell, the confidence scores should be zero. Otherwise we want the confidence score to equal the intersection over union (IOU) between the predicted box and the ground truth" [8].

The area under the result precision-recall curve (AUC) represents the model's overall performance for the given class. mAP is the mean of the AP values calculated for each class. mAP is a useful metric because it provides an overall performance score for the model, taking into account its performance in all classes. A high mAP value indicates that the model is able to accurately detect a wide range of classes with a good balance of precision and recall. AP@[.5:.95] corresponds to the average AP for IoU from 0.5 to 0.95 with a step size of 0.05.

In this work, we used mosaic augmentation to simulate real-world scenarios. However, the original images often contained only one object that filled the entire image, which made the mosaic augmentation less effective. This was because the augmentation did not provide useful information for scenarios where the image was zoomed in. To improve the effectiveness of the mosaic augmentation, we modified the scale parameter to reduce the size of the augmented image. This preserved the important features of the original image and allowed us to better simulate small objects scenarios.

Due to RAM and computation limitations, the batch size was reduced to 10, which is not optimal for the training. Additionally, Other studies have shown that fine-tuning the pre-trained YOLOv7 on COCO model can be done in only 55 epochs. However, when I applied this method to my own dataset, it took over 200 epochs, and even more in some cases, to converge (See Fig. 9(b).). This is likely due to several factors. First, other studies may have had access to more computation resources, which allowed them to train their models more quickly. Second, the datasets used in those studies may have been more robust than mine, because they were similar to the COCO dataset in terms of the objects being detected. In contrast, my dataset is fully augmented, which means it is less natively robust, and the task of detecting Majong tiles is not seen in the COCO dataset, making it more difficult for my model to converge.

Now focus on the results of the 250 epoch runs, we can see from Fig. 9(c) that the models with the scale parameter

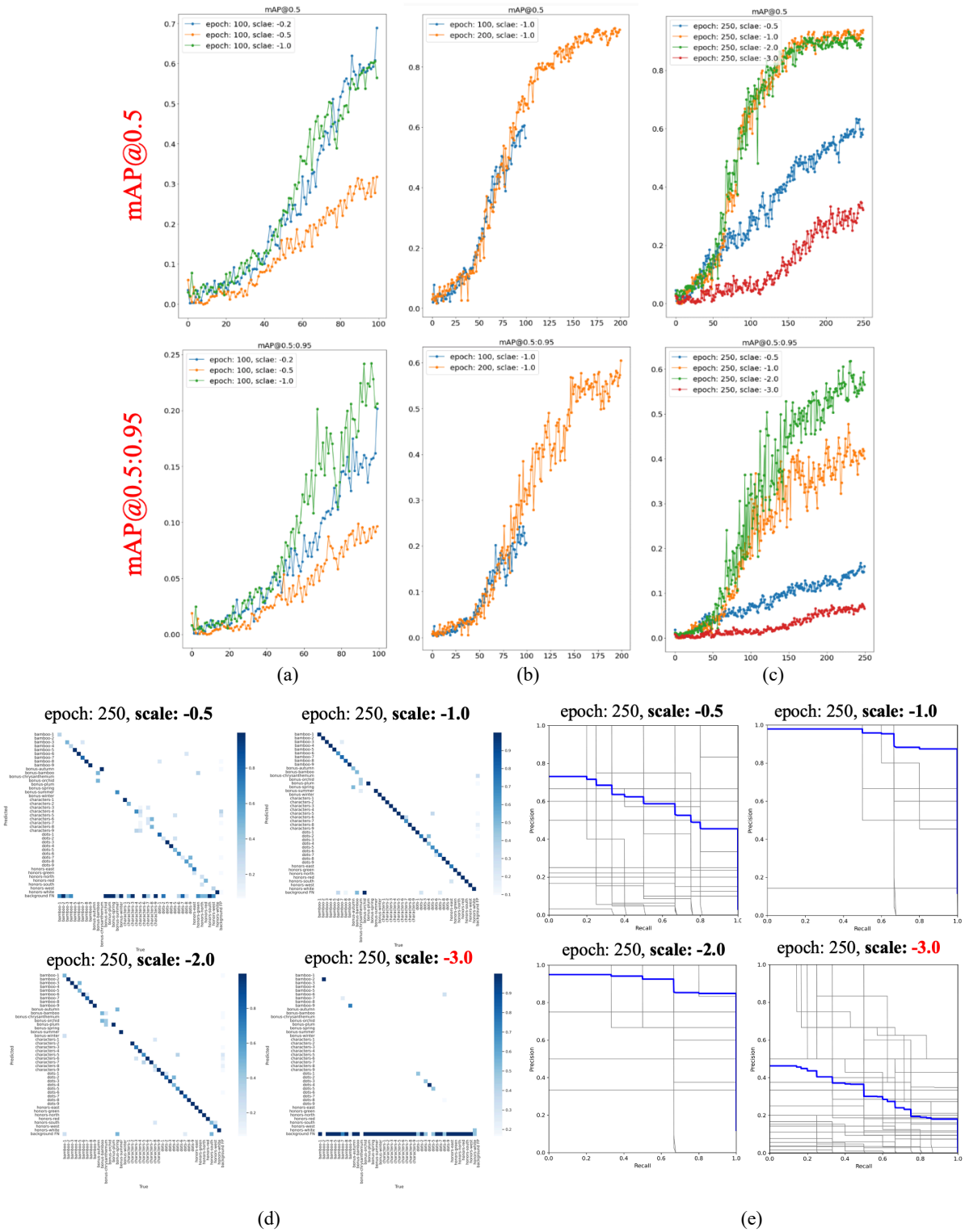

Fig. 9. The graph of different metrics and setting. (a) shows the mAP@0.5 and mAP@0.5:0.95 of different scale values when epoch is 100. (b) shows the mAP@0.5 and mAP@0.5:0.95 when the scale is set but the epoch increases. (c) shows the mAP@0.5 and mAP@0.5:0.95 when epoch is 250, differernt scale values' performance. (d) shows the confusio matrix of different scale values when epoch is 250. € shows the PR curve under the same condition as (d).

set to -1.0 and -2.0 performed similarly well based on the mAP@0.5 score. This outperforms the model where the scale is set to -0.5 and -3.0, indicating that the model may be scaled too small or not enough in those cases. However, when we look at the mAP@0.5:0.95 score, which is the average of 10 mAP scores with the IoU threshold set to 0.5-0.95 with a step size of 0.05, we can see that the scale -2.0 model clearly outperforms the scale -1.0 model. This suggests that as the threshold increases, the scale -1.0 model is not confident enough to make predictions.

Fig. 10. The result of real-world scenario tiles detections. When ordered the model was able to perform well and detect most of the tiles correctly with high confidence. However, when not, the model's performance gets worse as it either cannot detect the tiles, or detect the tiles wrongly, or detect the tiles correctly with low confidence.

Overall, these results indicate that the optimal scale parameter value falls around -2.0.

From Fig. 9(d), we can see that the model performs poorly on correctly detecting the tiles for the four classes: bonus-autumn, bonus bamboo, bonus-orchid, and bonus-plum. This is likely because these tiles may have different designs depending on the version or manufacturer of the Majong set. This makes these classes difficult to predict, as the data images may not be good training materials for this type of variation.

Finally, we test the model on real-world scenario images. In a real-world scenario, the Majong tiles are usually in a jumble. This scenario is simulated via data augmentation techniques (see Fig. 10.).

Overall, the model seems to be performing relatively well on ordered tiles. In these cases, it is able to correctly detect the tiles with high confidence. However, when applied to more complex, real-world scenarios, the model's performance deteriorates. It begins to make incorrect predictions, either predicting the wrong class of tile or predicting the correct class with low confidence. In some cases, the model even produces bounding boxes around non-tile objects or random areas of the image, indicating that it has learned something that is not useful for our use case. I suspect that this may be because the model has not fully converged due to limited computation resources and a small number of training epochs and batch size. It is not possible to determine whether the model could perform better without more training data and computational resources.

5. Conclusion

In this work, I demonstrated that it is possible to build a dataset from scratch using data augmentation techniques to train a model for real-world inference tasks. By carefully selecting and adjusting the hyperparameters of various data augmentation methods, it is possible to create a dataset that accurately simulates real-world scenarios. However, this study also showed that more sophisticated augmentation techniques may be necessary to fully

achieve this goal. For example, in the mosaic-augmented data, some of the images were zoomed in so much that they provided no useful information to the training batch. I believe that carefully controlling the transformations applied to non-real-world images has the potential to better represent real-world scenes, making it easier for researchers to build or augment datasets in the future.

In addition to the limitations discussed in this work, it is important to note that data augmentation is not a perfect solution for all datasets and tasks. While it can be a powerful tool for improving the performance of machine learning models, it is not always effective and may even introduce new problems. For example, if the transformations applied during data augmentation are not carefully chosen, they may introduce unrealistic variations or noise into the dataset, which can negatively affect the performance of the trained model. This problem is also reflected in the result of our tests.

References

[1] YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors .arXiv:2207.02696 [cs.CV]

[2] YOLOv4: Optimal Speed and Accuracy of Object Detection. arXiv:2004.10934 [cs.CV]

[3] Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. arXiv:1406.4729 [cs.CV]

[4] CSPNet: A New Backbone that can Enhance Learning Capability of CNN. arXiv:1911.11929 [cs.CV]

[5] Online Convolutional Re-parameterization. arXiv:2204.00826 [cs.CV]

[6] Feature Pyramid Networks for Object Detection. arXiv:1612.03144 [cs.CV]

[7] Path Aggregation Network for Instance Segmentation. arXiv:1803.01534 [cs.CV]

[8] You Only Look Once: Unified, Real-Time Object Detection. arXiv:1506.02640 [cs.CV]