Introduction

The full Github link would be in here: https://github.com/StephenEkaputra/Mask_RCNN-TinyPascalVOC

We used the repository from: https://github.com/matterport/Mask_RCNN

Computer

Used Campus' GPU in Docker mode with Cuda Version of 10.2.

Data Preprocessing

We used data downloaded from TA's drive containing 1,349 training data and 100 testing data. Somehow, we need validation data to train the model but 'test.json' has no image id and annotation key. So, we took only 10 training data which has annotation key as validation data to train faster and set the validation step in config.py to 1. Further, we fixed the 'pascal_train.json' format into coco annotation format (http://cocodataset.org/). All data conversion processes are in JsonConversion.ipynb file (training, validation, and testing set).

Training

We used GPU campus from docker to train the model by commanding: { python coco.py train --dataset=pascal --model=imagenet } in the samples/coco folder. We trained the model until 120th epoch. At first, we trained the heads layer until 40th epoch and then we used Resnet101 backbone until 120th epoch to improve the performance.

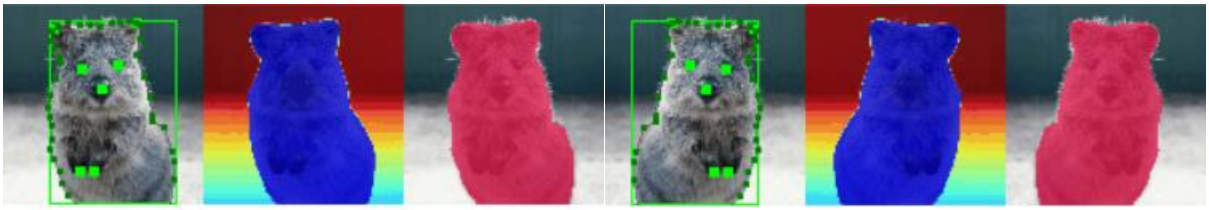
```
# Training - Stage 1
print("Training network heads")
model.train(dataset_train, dataset_val,
            learning_rate=config.LEARNING_RATE,
            epochs=40,
            layers='heads',
            augmentation=augmentation
            )

# Training - Stage 2
# Finetune layers from ResNet stage 4 and up
print("Fine tune Resnet stage 4 and up")
model.train(dataset_train, dataset_val,
            learning_rate=config.LEARNING_RATE,
            epochs=120,
            layers='4+',
            augmentation=augmentation)
```

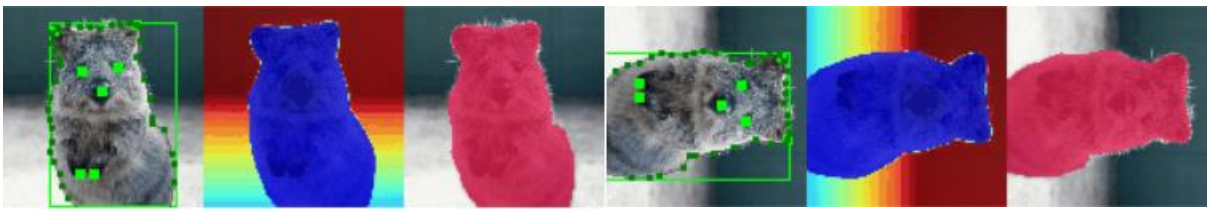
Since we don’t have enough training data, we used data augmentation to add variation of data. Imgaug library was used (https://github.com/aleju/imgaug). Using this augmentation will take the training process much longer but this library is so interesting enough to be used.

```
# Image Augmentation
augmentation = imgaug.augmenters.Sequential([
            imgaug.augmenters.Fliplr(0.5),
            imgaug.augmenters.Affine(rotate=(-45, 45), shear=(-16, 16), scale={"x": (0.8, 1.2), "y": (0.8, 1.2)}),
            imgaug.augmenters.GaussianBlur(sigma=(0, 3.0))
            ])
```

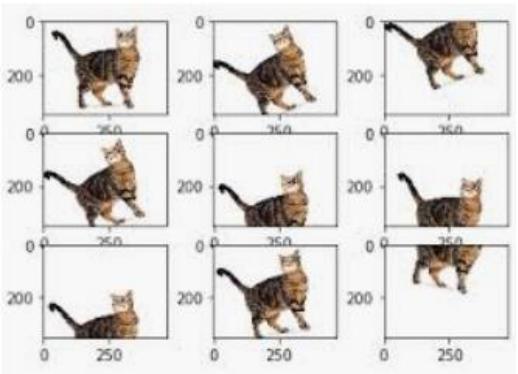
Fliplr:



Rotate:

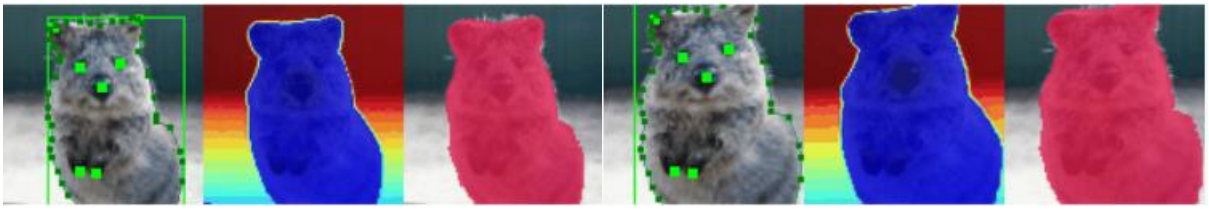


Shear:



(Reference: google)

Scale:



GaussianBlur:


We used these kinds of data augmentation because in our hypothesis these will work since we observe and compare between the train dataset and test dataset. We need to flip vertically, rotate 45°, shear, scale, and blur.

Experiments

Before we train the model, we read the original Mask-RCNN paper by Kaiming He et al. https://arxiv.org/pdf/1703.06870.pdf to know the parameter and method they used. The hyperparameter they set was similar to Faster-RCNN. They set the learning rate = 0.2, momentum = 0.9, and weight decay = 0.0001. They also used backbone ResNet101+FPN with AP = 35.4. Based on that, we trained the model with those parameters without data augmentation yet. However, the gradient exploded at early epochs so we changed the learning rate to 0.001 and it worked and learned well. But, when we test the weight, some of the result detected wrongly because overfit. Moreover, we found some people getting the same issues and here is the solution:

To prevent overfitting, you can try:

1. Getting a larger dataset (but this is probably not feasible, otherwise you would've done this already)
2. Stronger weight decay (i.e., L2 regularization)
3. Lower model capacity (e.g., ResNet-50 or even ResNet-32)
4. k-fold cross-validation

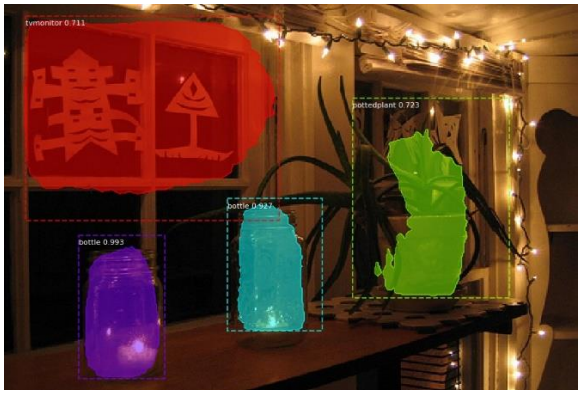
15

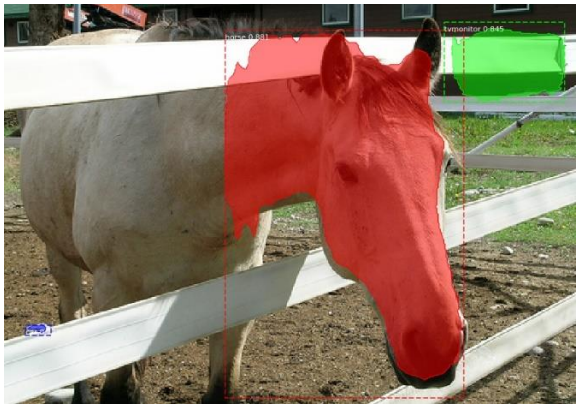
Further, we followed his opinion. We applied data augmentation to get larger dataset then we trained the model. We also had tried to use Dropout data augmentation and observed the result but it wasn't effective and lower the performance. After that, we set the weight decay stronger to 0.0003 to avoid overfit and smoothed the learning curve. We also increased the detection min confidence to 0.8 in terms of detection task. Besides that, we increased the RPN NMS threshold to 0.8 to generate more proposals. After training, we used the weight at 107th epoch which is the best from our network.

Results

Our good results:

Our bad results:





Conclusion

To improve the performance on Coco Dataset or else, we have to apply proper data augmentation by observing the testing dataset first. It is also used to generate vary dataset since our training dataset is small. We set the hyperparameter like weight decay stronger than the original Mask-RCNN to avoid overfit. For the future works, we can try to add more data augmentation which is proper with testing dataset, such as colour, crop, and perspective to improve more the performance. Finally, we can also apply the newest method from CVPR or else to boost more the performance.