Mask R-CNN
Semantic Segmentation

Team Members
KHOO WEE BENG
IAN TAN ENG KIONG

Table of Contents

EXECUTIVE SUMMARY

In this assignment, we attempted to efficiently detect objects in an image while simultaneously generating a high-quality segmentation mask for each instance. The method adopted is the Mask R-CNN. It is a deep neural network aimed to solve semantic segmentation problem in machine learning or computer vision. There are two stages of Mask RCNN. First, it generates proposals about the regions where there might be an object based on the input image. Second, it predicts the class of the object, refines the bounding box and generates a mask in pixel level of the object based on the first stage proposal.

The objects of interest in this assignment are face masks and hats. The importance of identifying these objects can be helpful in the security and safety industry. Individuals who have intention to hide their identity will probably wear hat and face mask. Hence, such suspicious individuals can be filtered out early if we have video analytics that can detect such objects. These individual can be detected early and preventive measures cab be activated. Similarly, in the safety industry, if a person does not wear a safety helmet or face mask, this can be identified easily from cameras with video analytics capabilities, and automatically alert the safety management team. However, in this assignment, we did not differentiate between a safety helmet and an ordinary hat. This differentiation can be explored in future work.

The key reference used in this assignment is from https://github.com/matterport/Mask_RCNN. This is an implementation of Mask R-CNN on Python 3, Keras, and TensorFlow. The model generates bounding boxes and segmentation masks for each instance of an object in the image. It's based on Feature Pyramid Network (FPN) and a ResNet101 backbone. However, in this assignment we adopted the ResNet-50 instead as it less resource intensive. We trained a model to detect face masks as well as hats, and then we use generated masks to keep the face masks and hats in color. while changing the rest of the image to greyscale.

Instead of training a model from scratch, we use transfer learning, where we start with a weights file that's been trained on the COCO dataset (provided in the above github repo). Although the COCO dataset does not contain hats or face masks, it contains a lot of other images (~120K), so the trained weights have already learned a lot of the features common in natural images, which really helps. Subsequently, we collected at least 200 instances of hats and face masks training data to build the model. These images need to be annotated. We use the VIA (VGG Image Annotator) because of its simplicity. It's a single HTML file that you download and open in a browser. The VIA tool saves the annotations in a JSON file, and each mask is a set of polygon points. In this assignment, we used the pre-trained COCO weights.

With the new model created, we are able to create masks for the face masks and hats. Then, we apply the color splash effect. Basically we create a grayscale version of the image, and then, in areas marked by the object mask, copy back the color pixels from original image.

The basic restnet50 model with 50 epoch steps and 10 validation steps performed rather well. It was observed that increasing epoch and validation, or with image augmentation (imgaug), did not improve the model's performance.

PROBLEM OVERVIEW

PROBLEM STATEMENT

In late 2019, a novel coronavirus emerged in China. Since then, it has rapidly spread throughout the world. This novel coronavirus is called SARS-CoV-2 and the disease that it causes is called COVID-19. During this challenging and stressful period, it is critical that patients or individuals who are ill should be wearing masks in hospitals or public areas in order to minimize the spread of the disease. Hence, video analytics that is able to detect mask on faces becomes critical in areas such as hospitals etc. Smart Monitor's facial recognition solutions should incorporate face mask detection, to enforce compliance with health regulations. Recently, Chinese intelligent interaction company Hanwang Technology, also known in English as Hanvon, says it has developed a facial recognition system that can identify people wearing masks. Please refer to this link: https://tech.newstatesman.com/security/hanwang-technology-id-mask-wearing-faces for more details. Therefore, the detection of individuals wearing mask will become a standard feature in todays or future facial recognition video analytics solution.

Each day, countless workplace injuries and illnesses are prevented through the use of personal protective equipment (PPE). Employers in the manufacturing, construction, and other sectors where hazards can threaten health and safety evaluate their operations to identify risks and develop PPE policies to minimize them. Two common PPE used in today's construction industry or factories are hard hats (safety helmets) and face masks. Ensuring PPE compliance can be assisted using computer vision solutions. Computer Vision solutions can be utilized to constantly monitor the worksite by analyzing video data from security surveillance cameras already installed on the construction site. It can alert the safety team on individuals who are not wearing their hard hats or face mask in designated areas.

PROJECT OBJECTIVES

The primary goal of the project is to build and train a model that is capable in detecting and segmenting 2 instance classes namely face mask and hats. Training and validation datasets consisting of images of people wearing face masks or hats were collated from the web. The minimum number of instances shall be not less than 200 instance per class.

These training and validation images are required to be annotated accordingly. The VIA (VGG Image Annotator) was used to provide the annotation, and a corresponding Json files was created to store sets of polygon points which was used subsequently for the training and validation of the model.

All the corresponding codes have to be written using Python 3 and Keras. The method adopted in this assignment is Mask R-CNN.

MODEL OVERVIEW

The new model created will be created using the pre-trained COCO weights and the corresponding training images of the targeted objects namely face masks and hats. The model shall be able to detect and segment instances of the face mask and hat classes. Upon detection of these mask, a color splash will be applied on the image.

TECHNICAL DISCUSSION

INTRODUCTION

What is Semantic Segmentation?

Semantic segmentation achieves fine-grained inference by making dense predictions inferring labels for every pixel, so that each pixel is labeled with the class of its enclosing object ore region.

Mask R-CNN

Mask R-CNN is basically an extension of Faster R-CNN. Faster R-CNN is widely used for object detection tasks. For a given image, Mask R-CNN, in addition to the class label and bounding box coordinates for each object, will also return the object mask. There are two stages of Mask RCNN. First, it generates proposals about the regions where there might be an object based on the input image. Second, it predicts the class of the object, refines the bounding box and generates a mask in pixel level of the object based on the first stage proposal.

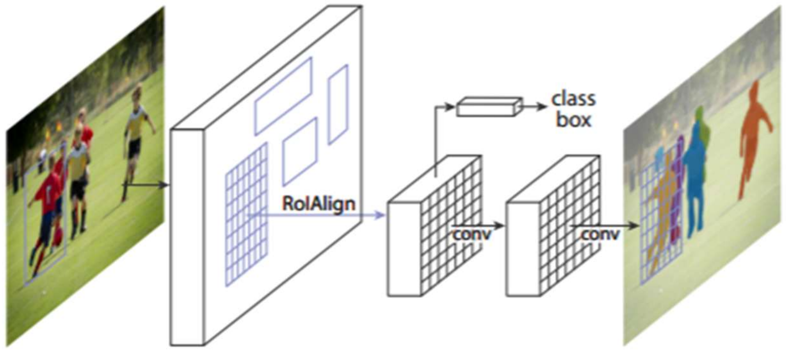

Figure 1: Mask R-CNN framework. Source: https://arxiv.org/abs/1703.06870

1) Backbone

In this assignment, we adopted the standard convolutional neural network such as ResNet50 to serve as a feature extractor for the images in Mask R-CNN. The early layers detect low level features (edges and corners), and later layers successively detect higher level features (car, person, sky).

Passing through the backbone network, the image is converted from 1024x1024px x 3 (RGB) to a feature map of shape 32x32x2048. This feature map becomes the input for the following stages.

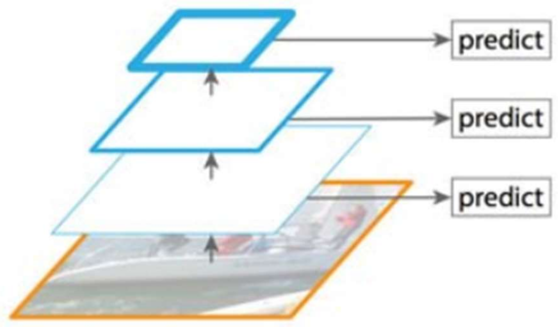

Figure 2: Pyramid of Feature Map

In this assignment, we also incorporated a pyramid of feature and use them for object detection (refer to Figure 2). This is to address the issue of detection objects in different scales in particular for small objects. Hence, Feature Pyramid Network (FPN) is a feature extractor designed for such pyramid concept with accuracy and speed in mind. It replaces the feature extractor of detectors like Faster R-CNN and generates multiple feature map layers (multi-scale feature maps) with better quality information than the regular feature pyramid for object detection.

2) Region Proposal Network (RPN)

   FPN is not an object detector by itself. It is a feature detector as explained above that works with object detectors. Feature maps obtained in the previous step and apply a region proposal network (RPN). RPN is a lightweight neural network that scans the image in a sliding-window fashion and finds areas that contain objects. The regions that the RPN scans over are called anchors which are boxes distributed over the image area. This basically predicts if an object is present in that region (or not). In this step, we get those regions or feature maps which the model predicts contain some object. The RPN generates two outputs for each anchor:

   a. Anchor Class: One of two classes: foreground or background. The FG class implies that there is likely an object in that box.

   b. Bounding Box Refinement: A foreground anchor (also called positive anchor) might not be centered perfectly over the object. So, the RPN estimates a delta (% change in x, y, width, height) to refine the anchor box to fit the object better.

Using the RPN predictions, we pick the top anchors that are likely to contain objects and refine their location and size. If several anchors overlap too much, we keep the one with the highest foreground score and discard the rest (referred to as Non-max Suppression).

3) RoI-Classifier & Bounding box Regressor

The regions obtained from the RPN might be of different shapes. Hence, apply a pooling layer and convert all the regions to the same shape. This stage runs on the regions of interest (ROIs) proposed by the RPN. And just like the RPN, it generates two outputs for each ROI:

a. **Class:** The class of the object in the ROI. Unlike the RPN, which has two classes (FG/BG), this network is deeper and has the capacity to classify regions to specific classes (person, car, chair, …etc.). It can also generate a *background* class, which causes the ROI to be discarded.

b. **Bounding Box Refinement:** Very similar to how it's done in the RPN, and its purpose is to further refine the location and size of the bounding box to encapsulate the object.

ROI pooling refers to cropping a part of a feature map and resizing it to a fixed size. It's similar in principle to cropping part of an image and then resizing it (but there are differences in implementation details).

The original paper on Mask R-CNN suggest a method they named ROIAlign, in which they sample the feature map at different points and apply a bilinear interpolation. In our implementation, we used TensorFlow's crop_and_resize function for simplicity and because it's close enough for most purposes.

Till this point, the steps are almost similar to how Faster R-CNN works. Now comes the difference between the two frameworks. In addition to this, Mask R-CNN also generates the segmentation mask.

For that, we first compute the region of interest so that the computation time can be reduced. For all the predicted regions, we compute the Intersection over Union (IoU) with the ground truth boxes. We can computer IoU like this:

4) Segmentation Mask

The mask branch is a convolutional network that takes the positive regions selected by the ROI classifier and generates masks for them. The generated masks are low resolution: 28x28 pixels. But they are *soft* masks, represented by float numbers, so they hold more details than binary masks. The small mask size helps keep the mask branch light. During training, we scale down the ground-truth masks to 28x28 to compute the loss, and during inferencing we scale up the predicted masks to the size of the ROI bounding box and that gives us the final masks, one per object.

TECHNICAL SOLUTION

STRUCTURE OF CODES

`samples/fmask/fmask.py`

This script serves as the library to be imported for training and testing the model. The main classes and methods has been described below.

`FMaskConfig` This class contains the default configuration. It is configured such that there are sufficient GPU resources to load the model. The `STEPS_PER_EPOCH` has been reduced to `50`, `VALIDATION_STEPS` reduced to `10`, and the image dimensions has been scaled down to `128`.

```
class FMaskConfig(Config):
    """Configuration for training on the face mask dataset.
    Derives from the base Config class and overrides some values.
    """
    # Give the configuration a recognizable name
    NAME = "fmask"

    # We use a GPU with 12GB memory, which can fit two images.
    # Adjust down if you use a smaller GPU.
    IMAGES_PER_GPU = 1

    # Number of classes (including background)
    NUM_CLASSES = 1 + 2  # Background + mask + hat

    # Skip detections with < 90% confidence
    DETECTION_MIN_CONFIDENCE = 0.9

    # Number of training steps per epoch
    STEPS_PER_EPOCH = 50
    # Number of validation steps to run at the end of every training epoch.
    VALIDATION_STEPS = 10
    # Backbone network architecture
    # Supported values are: resnet50, resnet101.
    BACKBONE = "resnet50"

    IMAGE_MIN_DIM = 128
    IMAGE_MAX_DIM = 128

    # Number of ROIs per image to feed to classifier/mask heads
    TRAIN_ROIS_PER_IMAGE = 32

    # Maximum number of ground truth instances to use in one image
    MAX_GT_INSTANCES = 50 #100
```

`FMaskDataset` This class is the dataset class that loads(`load_fmask`) the defined classes `mask` and `hat` based on the annotations created by VIA 2.0. The x, y coordinates of points of the polygon annotated for each instance of the object (using VIA) is extracted, and used as part of the data to be trained for object segmentation.

```
class FMaskDataset(utils.Dataset):
        """Load a subset of the FMask dataset.
        Subset to load: train or val
        """
    def load_fmask(self, dataset_dir, subset):
        :
        :
```

`fit()` This method trains the model and creates a .h5 file for subsequent object detection. Settings such number of epoch to be trained, configuration, and any image augmentation has been parameterized for ease of tuning hyperparameters.

```
def fit(fepoch,fmodel,fdataset,fconfig,save_fmodel,save_fmodel_path,faug):
    """Train the model."""
    # Training dataset.
    dataset_train = FMaskDataset()
    dataset_train.load_fmask(fdataset, "train")
    dataset_train.prepare()

    # Validation dataset
    dataset_val = FMaskDataset()
    dataset_val.load_fmask(fdataset, "val")
    dataset_val.prepare()

    print("Training network heads")
    fmodel.train(dataset_train, dataset_val, learning_rate=fconfig.LEARNING_RATE,
epochs=fepoch, layers='all', augmentation=faug)
    model_path = os.path.join(ROOT_DIR, save_fmodel)
    fmodel.keras_model.save_weights(save_fmodel_path)
```

`samples/fmask/TrainTest.ipynb`

This script is used for training and testing of the segmentation model. The main sections of the codes have been described below.

Training Part 1

The section below uses pre-trained weights from MS COCO to train the model. Image augmentation to flip the image left right, followed by CLAHE (Contrast Limited Adaptive Histogram Equalization) has been applied. Due to resource limitation, epoch was set as 30 and the.h5 file has been saved.

```
config = fmask.FMaskConfig()
config.BACKBONE = "resnet50"

config.display()
model = modellib.MaskRCNN(mode="training", config=config, model_dir=MODEL_DIR)
weights_path = COCO_WEIGHTS_PATH
model.load_weights(weights_path, by_name=True, exclude=["mrcnn_class_logits",
"mrcnn_bbox_fc","mrcnn_bbox", "mrcnn_mask"])

faug = iaa.Sequential([iaa.Fliplr(1),iaa.CLAHE()])
#faug = iaa.OneOf([iaa.ContrastNormalization((0.5, 1.5)), iaa.GaussianBlur(sigma=(0.0, 5.0))])
#faug = None
fmask.fit(30,model, train_dataset, config, "mask_rcnn_fmask.h5", FMASK_MODEL_PATH, faug)
```

Run Object Detection

The configurations have been set prior to running object detection. The image files indicated in the specified path has been retrieved to run object detection. On obtaining the results, the original image was changed to greyscale, and a colour splash has been applied on the detected objects.

```
file_names = next(os.walk(IMAGE_DIR))[2]
for file_name in file_names:
    print(file_name)
    image = skimage.io.imread(os.path.join(IMAGE_DIR, file_name))

    # Run detection
    results = model.detect([image], verbose=1)

    # Visualize results
    r = results[0]
    print(r['class_ids'])
    print(r['scores'])
    gray = skimage.color.gray2rgb(skimage.color.rgb2gray(image)) * 255
    visualize.display_instances(gray, r['rois'], r['masks'], r['class_ids'],
                                class_names, r['scores'])
```

Training Part 2 (with pre-trained weights)

The model has been trained again, this time, with the weights generated by the previous training (part 1). No augmentation has been set for this training. A new .h5 file has been saved (`mask_rcnn_fmask_2.h5`).

```
config = fmask.FMaskConfig()
config.BACKBONE = "resnet50"

config.display()
model = modellib.MaskRCNN(mode="training", config=config, model_dir=MODEL_DIR)
weights_path = FMASK_MODEL_PATH
model.load_weights(weights_path, by_name=True, exclude=["mrcnn_class_logits",
"mrcnn_bbox_fc","mrcnn_bbox", "mrcnn_mask"])

#faug = iaa.Sequential([iaa.Fliplr(1),iaa.CLAHE()])
#faug = iaa.OneOf([iaa.ContrastNormalization((0.5, 1.5)), iaa.GaussianBlur(sigma=(0.0, 5.0))])
faug = None
fmask.fit(30,model, train_dataset, config, "mask_rcnn_fmask_2.h5", FMASK_MODEL_PATH, faug)
```

Create Model using the newly trained weights

A new model has been created using the newly trained weights(`mask_rcnn_fmask_2.h5`) and the object detection has been performed again.

```
# newly trained weights
FMASK_MODEL_PATH_2 = os.path.join(ROOT_DIR, "mask_rcnn_fmask_2.h5")

config.GPU_COUNT = 1
config.IMAGES_PER_GPU = 1
# Create model object in inference mode.
model = modellib.MaskRCNN(mode="inference", model_dir=MODEL_DIR, config=config)

# Load weights
model.load_weights(FMASK_MODEL_PATH_2, by_name=True)
# fmask Class names
dataset = fmask.FMaskDataset()
dataset.load_fmask(FMASK_DIR, "train")
dataset.prepare()
class_names = dataset.class_names
```

FINDINGS

To train the model, 207 images with face masks and 203 images with various hats has been annotated. For the validation set, 20 images with face mask and 24 images with hat was used.

9 images were select to test the capability of the model. All 9 images have both face mask and hat object instances whereas the training and validation sets have independent object instances.

The iterations for training the model and the results are as follows:

Config backbone	: resnet50
Steps per epoch	: 50
Validation steps	: 10
<u>Results</u>	
Accuracy	: 0.6667
Precision	: 1

		Predicted	
		Positive	Negative
Actual	Positive	12	6
	Negative	0	0

Config backbone	: resnet50
Steps per epoch	: 100
Validation steps	: 10
<u>Results</u>	
Accuracy	: 0.6667
Precision	: 1

		Predicted	
		Positive	Negative
Actual	Positive	12	6
	Negative	0	0

Config backbone	: resnet50
Steps per epoch	: 50
Validation steps	: 50
<u>Results</u>	
Accuracy	: 0.6667
Precision	: 0.9231

		Predicted	
		Positive	Negative
Actual	Positive	12	5
	Negative	1	0

Config backbone	: resnet50
Steps per epoch	: 100
Validation steps	: 50
Augmentation	: Sequential
	iaa.Sharpen(alpha=(0, 1.0), lightness=(0.75, 1.5))
	iaa.AllChannelsHistogramEqualization()
<u>Results</u>	
Accuracy	: 0.5556
Precision	: 0.8333

		Predicted	
		Positive	Negative
Actual	Positive	10	6
	Negative	2	0

Config backbone	: resnet50
Steps per epoch	: 50
Validation steps	: 10
Augmentation	: Sequential
iaa.Fliplr(1)	
iaa.Affine(rotate=(-45, 45))	
<u>Results</u>	
Accuracy	: 0.7222
Precision	: 1

		Predicted	
		Positive	Negative
Actual	Positive	13	5
	Negative	0	0

Config backbone	: resnet50
Steps per epoch	: 50
Validation steps	: 10
Augmentation	: OneOf
iaa.ContrastNormalization((0.5, 1.5))	
iaa.GaussianBlur(sigma=(0.0, 5.0))	
<u>Results</u>	
Accuracy	: 0.6111
Precision	: 0.8461

		Predicted	
		Positive	Negative
Actual	Positive	11	5
	Negative	2	0

Config backbone	: resnet50
Steps per epoch	: 50
Validation steps	: 10
Augmentation	: Sequential
iaa.Fliplr(1)	
iaa.CLAHE()	
<u>Results</u>	
Accuracy	: 0.7222
Precision	: 1

		Predicted	
		Positive	Negative
Actual	Positive	13	5
	Negative	0	0

It was observed that using resnet101 as backbone faces memory issues and was unable to run. Adding image augmentation did improve the model's performance. An attempt to further improve the model's performance was made by training on the trained weights from one of the better performing set of hyperparameters.

Combination using trained weights: Attempt 1 (`TrainTest.ipynb)`

Training Part 1

Config backbone	: resnet50
Steps per epoch	: 50
Validation steps	: 10
Augmentation	: Sequential
iaa.Fliplr(1)	
iaa.CLAHE()	
Results	
Accuracy	: 0.8333
Precision	: 1

		Predicted	
		Positive	Negative
Actual	Positive	15	3
	Negative	0	0

Training Part 2 (using pre-trained weights from previous training)

Config backbone	: resnet50
Steps per epoch	: 50
Validation steps	: 10
Augmentation	: None
Results	
Accuracy	: 0.7778
Precision	: 0.9333

		Predicted	
		Positive	Negative
Actual	Positive	14	3
	Negative	1	0

Combination using trained weights: Attempt 2 (`TrainTest_2.ipynb)`

Training Part 1

Config backbone	: resnet50
Steps per epoch	: 50
Validation steps	: 10
Augmentation	: None
Results	
Accuracy	: 0.6667
Precision	: 1

		Predicted	
		Positive	Negative
Actual	Positive	12	6
	Negative	0	0

Training Part 2 (using pre-trained weights from previous training)

Config backbone	: resnet50
Steps per epoch	: 50
Validation steps	: 10
Augmentation	: Sequential
iaa.Fliplr(1)	
iaa.CLAHE()	
Results	
Accuracy	: 0.7778
Precision	: 0.9333

		Predicted	
		Positive	Negative
Actual	Positive	14	3
	Negative	1	0

CONCLUSION

The final model achieved has reasonably performance. Using Mask R-CNN we can perform both:

1) Object detection, giving us the (x, y)-bounding box coordinates of for each object in an image.
2) Instance segmentation, enabling us to obtain a pixel-wise mask for each individual object in an image.

Hence, the output of the model is the bounding box of the targeted objects in the image, but also pixel-wise masks for each object as well, enabling us to segment each the objects, something that object detection solution alone does not provide.

The best accuracy and precision scores obtained was 0.833 and 1 respectively as shown in the results in training part 1. This was based on the approach of using pre-trained weights from MS COCO to train the model. Image augmentation to flip the image left right, followed by CLAHE (Contrast Limited Adaptive Histogram Equalization) has been applied. This is expected as the features from the MS COCO has rich generic features.

The codes and report for this project can be found at https://github.com/dion797/VSE_CA1.

LIMITATIONS

The model created in this assignment is to detect and segment 2 instance classes namely face mask and hats. Given that hat is a very broad category, it can encompass many types of hats namely hard hats, caps, helmets, fashion hats etc. In order to build a robust model, we would need to provide an exhaustive type of hat training data. This may not be pragmatic for this assignment. It will be more appropriate to create an ontology of hats, based on multiple classes of hats such as hard hats, caps, helmets, fashion hats etc. Hence, rather than limiting only to these 2 broad class, create targeted sub classes instead.

REFERENCES

1) Renu Khandelwal. “Computer Vision: Instance Segmentation with Mask R-CNN”, towards data science, Jul 31, 2019, .https://towardsdatascience.com/computer-vision-instance-segmentation-with-mask-r-cnn-7983502fcad1

2) “What is the difference between semantic segmentation, object detection and instance segmentation?” , https://datascience.stackexchange.com/questions/52015/what-is-the-difference-between-semantic-segmentation-object-detection-and-insta

3) Xiang Zhan, “Simple Understanding of Mask RCNN”, Medium.com, Apr 23, 2018, https://medium.com/@alittlepain833/simple-understanding-of-mask-rcnn-134b5b330e95

4) Waleed Abdulla ,“Splash of Color: Instance Segmentation with Mask R-CNN and TensorFlow”, Medium.com, Mar 20, 018, https://engineering.matterport.com/splash-of-color-instance-segmentation-with-mask-r-cnn-and-tensorflow-7c761e238b46

5) Waleed Abdulla, “Mask R-CNN for Object Detection and Segmentation”, Github, Mar 10, 2019 https://github.com/matterport/Mask_RCNN