CLOTHING IMAGE SEGMENTATION USING TRANSFER LEARNING ON MASK RCNN

Yam Gui Peng David, Zhao Yazhi

Institute of Systems Science, National University of Singapore, Singapore 119615

ABSTRACT

In this Continuous Assessment (CA), we utilize the Mask RCNN architecture with a ResNet backbone to train and infer image segmentation masks. The initial weights were pretrained on COCO Images, and transfer learning was done onto images derived from the OpenImagesv5 dataset. The image classes are: Suit, Dress Jeans. The resulting test loss is 2.24 and the Mean Average Precision (mAP) achieved is 0.82. Lastly, 5 scoring images have been provided to showcase the model's effectiveness in inference.

***Index Terms*—** Mask RCNN, Machine Learning, Deep learning, Transfer Learning, Clothing

1. BUSINESS PROBLEM BACKGROUND

Image Segmentation creates a pixel-wise mask for each object in the image. [1] This is in contrast to Image Detection which only provides the bounding box coordinates surrounding the object and Image Classification which only provides a label. Having the object mask allows us to have greater detail and granularity when observing an image, as the segmentation mask tells us exactly which pixels are part of the object we are concerned about. In areas like autoamted vehicles [2], robotics and medicine, having the knowledge of exactly which pixels are from the object allows for finer grained actions (e.g. grasping/ shifting). Also, in the area of fashion, the masks allow for us to more easily crop out the specific object (e.g. Suit/ Dress/ Jeans) and replace the fashion model with others.

2. OBJECTIVES AND SUCCESS MEASUREMENTS

2.1. Objectives

Instance Segmentation [3] is the task of detecting and delineating each distinct object of interest appearing in an image. Prior to the age of deep learning, instance segmentation was still a new field in computer vision. For this project, our team decided to build a custom instance segmentation model to detect specific objects in images gathered from OpenImagev5.

2.2. Success Measurements

Average Precision (AP) and Mean Average Precision (mAP) are the two main metrics to evaluate a certain algorithm on instance segmentation. Average Precision (AP) is defined as the area under the precision-recall curve (PR Curve). (Where X-axis is recall, which is the percentage of correct positive predictions among all predictions made and Y-axis is precision, which is the percentage of correct positive predictions among all positive cases in reality.) mAP is the averaged value of all AP values over different classes/categories.

3. SELECTION OF ALGORITHM

All 3 possible models were initially evaluated for the CA. However, it was deemed that DeepLabv3 and YOLOACT unsuitable due to hardware/software challenges. It was found that DeepLabv3 [4] required Tensorflow version 2.x, which is currently incompatible with Windows machines when training with GPUs and we could not procure linux based machines. Also, the YOLOACT [5] implementation found was only for PyTorch and the effort required to convert it to Tensorflow was deemed as quite high. Hence we went for the Mask RCNN stable Tensorflow implementation [6].

4. CREATION OF DATASET

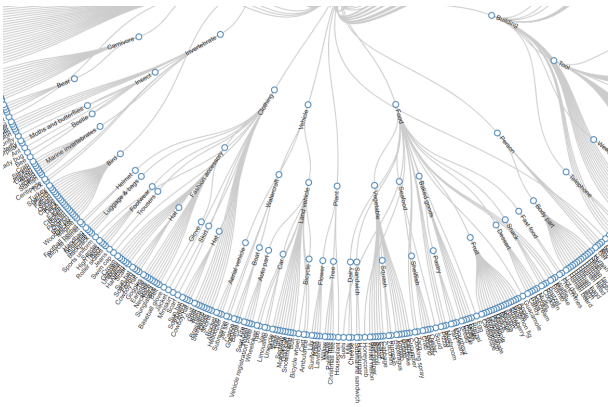

Fig. 1: Snapshot of Class Heirarchy in OpenImagesV5

While OpenImages [7] has over 7k trainable classes (see Fig.1), it has only 600 classes with BBoxes and even fewer with available segmented mask targets. OpenImages provides a JSON file within the hierarchy of the 600 trainable classes. We selected for the following superset categories: Person, Vehicles, Clothing and picked the segmentable classes to select for potential classes for the CA.

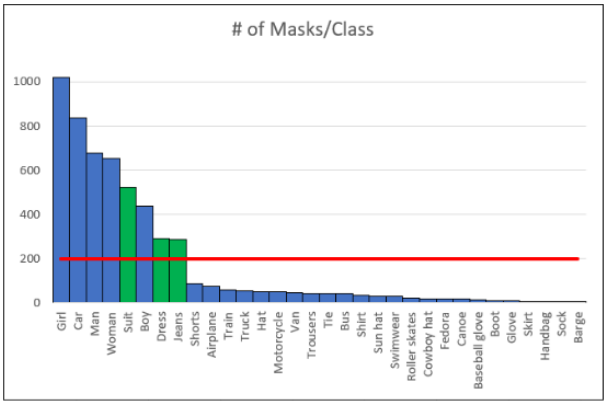

Fig. 2: Number of Mass Classes in F Partition

The dataset was taken from a single partition (F partition) of OpenImagesv5. The masks were checked if there were enough for the CA requirements and categories with less than 200 instances were discarded (e.g. Shorts, Airplane, Train, Truck, etc). Some other classes were removed for unsuitability (Girl, Man, etc), leaving us with 3 final classes (Suit, Dress, Jeans), shown in Fig.2 in Green.

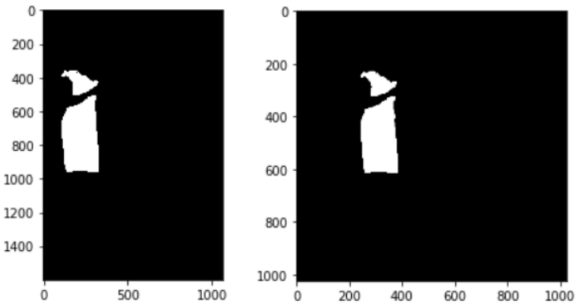

Fig. 3: Original Mask (left) and Resized Mask (right)

Whilst COCO Images stores the masks using Run Length Encoding (RLE) or (x,y)-positions from mask contours [8], OpenImagesv5 stores it as a separate mask image. Hence there was a need to convert OpenImagesv5 format to the COCO format. OpenImages also stores the masks and images in different pixel sizes (albeit the same pixel ratio), so the images/masks were converted to the same size using the function resize_image(), as show in Fig.3 and Fig.4.

The masks & bounding boxes were randomly verified to check that it fits the images as shown in Fig.5.

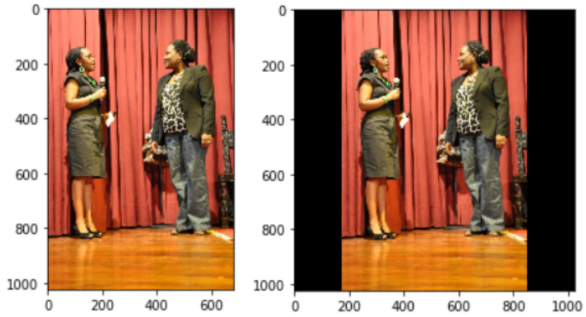

Fig. 4: Original Image (left) and Resized Image (right)

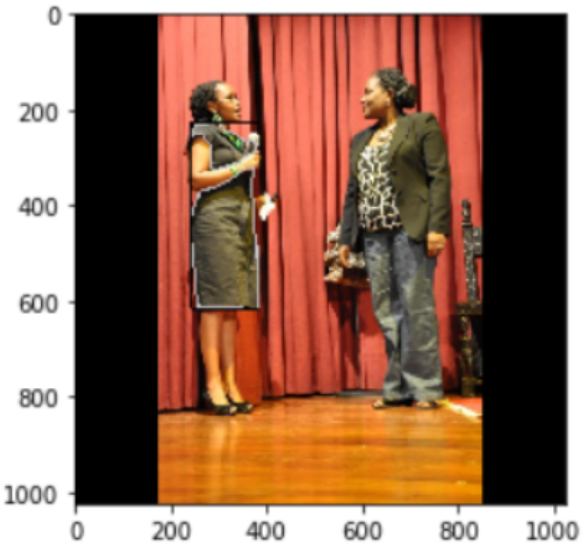

Fig. 5: Image with Bounding Box (black line) and Mask Contours (white lines)

5. WORKING DETAIL OF METHOD

Mask RCNN [9] is a deep neural network, which aims to solve instance segmentation problems for both images and videos. Mask RCNN framework is built on top of Faster RCNN. In addition to the class label and bounding box coordinates for each object, Mask RCNN can also separate different objects with masks.

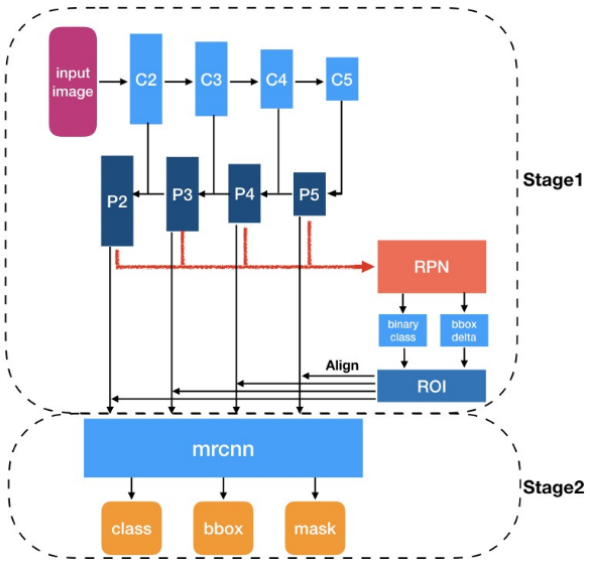

Fig. 6: Neural Network Architecture of Mask RCNN

As shown in Fig.6, there are two stages of Mask RCNN. Firstly, it generates proposals of regions where those regions might include objects based on the input images. Region Proposal Network (RPN) uses a Convolutional Neural Network to generate multiple Region of Interests (ROIs) with a lightweight binary classifier. As looking through various ROIs requires large resources and running time, Mask RCNN uses anchor boxes to improve the speed and efficiency for object detection. Anchor boxes are a set of predefined bounding boxes with a fixed height and width. Since anchor boxes have different scales and aspect ratios, RPN uses anchor boxes to detect the objects by computing Intersection over Union (IoU) values between ground truth and ROI.

Secondly, another neural network takes proposed regions generated by the first stage to predict classes of objects and generate masks in the pixel level of objects. Similar to the first stage, the second stage neural network uses a trick called ROIAlign to locate the relevant areas of the feature map, and generate masks for each object

Both stages are connected to the backbone structure (currently ResNet). With this architecture, Mask RCNN is able to detect objects with bounding box coordinates, classes labels and masks at the pixel level.

6. STRUCTURE OF CODE

Table 1: Code Section & Purpose

Section	Purpose
Imports	To depict all the required imports.
Config	Class that specifies the base configuration options, to be inherited.
Dataset	Class that specifies the base methods used to extract the data from the images/COCO formatted .json file, to be inherited.
Visualization	Functions to visualize output of the images/ masks/ metrics/ etc.
Data Formatting	Functions to parse the metadata fields.
Masks & Images	Functions dealing with resizing, expanding, moulding regarding masks/images.
Bounding Boxes	Functions dealing with extraction, computation, IoU, etc.regarding bounding boxes.
Anchors	Functions to generate anchors.
Miscellaneous (Graph/ Others)	Other miscellaneous function.
NN Architecture Layers	Functions and classes that specify the NN Architecture of the respective layers.
Loss Functions	Functions that specify the loss functions for the various layers.
Data Generator	Function to load images and ground truth masks.
MaskRCNN Class	Main class for combining all layers into a trainable/usable model.

The code is taken mainly from the matterport implementation [6]. The code is roughly broken down into the Table.1 It is designed in a modular structure with each section of code having a specific purpose.

Function (also Method in Class)	Explanation
__init__	set values for batch size and image size, compose image meta data length
display	display configuration values
__init__	create dataset with image id and image info
add_class	check if class is already exist or append the new class
add_image	append image into dataset
prepare(clean_name)	cleans up the data and prepares it
map_source_class_id	takes a source class ID and returns the int class ID assigned to it

Fig. 7: Functions and their Explanation

Fig.7 shows more in-depth mapping of the various functions and their usage is in the appendix. The naming of the code also follows a certain structure. For example each component that creates a Neural Network (NN) Architecture graph is named (component)_graph(). (e.g. resnet_graph()

is to build the Neural Network Architecture of the backbone ResNet graph). In one sense, the code can be brown down into (component)_graph(), where (component) is resnet, rpn, fpn_classifier, rpn_bbox, etc.

Also, to aid in code readability, the python code has been made to follow PEP8 guidelines. We used the "Uncompromising Code Formatter" black (with flag -l 79) and the format checker flake8 (with flag –ignore=E203,E231,W503) to do it. (e.g. E722 do not use bare 'except', E501 line too long, W191 indentation contains tabs, etc)

7. DESIGN OF APIS

The APIs can be mainly split into the following important methods from the various classes: load_data(), prepare(), build(), load_weights(), fit() & predict(). This is designed in a modular fashion so that each action/ component can be used separately. The API has been named to be similar to the popular Python Machine Learning package sklearn that uses fit() and predict() for training and testing/validation respectively. As with the sklearn package, the model needs to be instantiated to an object (in most cases the "model" object). Then the weights are loaded into the object via the call model.load_weights().

For this MASKRCNN implementation, the model, dataset & config objects are all created as instances of the classes MASKRCNN(), CocoLikeDataset() and myMaskRCNNConfig()/ myMaskRCNNInferenceConfig() respectively. This allows for better structuring and logical organization of the objects and code.

For assessing and usage of the model, the MASKRCNN() class is the most important and contains the methods to build the NN architecture utilizing the various functions mentioned above. A brief explanation will be done for MASKRCNN.build() method: It first asserts that the mode is either "training" or "inference" Then if it is in "training" mode it gathers the inputs for the RPN class, bboxes, class IDs, bboxes and masks. - Next it adds layers for the ResNet via the function resnet_graph() - Next it adds layers for the Region Proposal Network via the function build_rpn_model() - Next it adds layers for the ProposalLayer, FPN Classifier, FPN Mask respectively Next it adds the loss functions (RPN Class, RPN Bbox, MRCNN Class, MRCNN Bbox, MRCNN Mask) - Finally it compiles the model object

The structure of the code is designed to be modular to allow for replacement/ of certain layers as necessary. Fig.8 shows a more in-depth mapping of the various main APIs and the 1st/2nd level functions that are used is attached in the appendix.

8. STEPS TO IMPROVE PERFORMANCE

The default Mask RCNN package uses the ResNet architecture with default configuration. In order to achieve lower total loss and higher mAP rate, our team has decided to improve model performance by modifying the ResNet architecture and fine tuning the configuration parameters.

Main Functions	Down 1 layer	Down 2 layers
clean_name()		
<...>.build()		
(branch "training")		
--->	keras.layers.Input()	
--->	norm_boxes_graph()	
	--->	tf.split()
	--->	tf.concat()
	--->	tf.constant()
	--->	tf.divide()
--->	resnet_graph()	
	--->	keras.layers.ZeroPadding2D()
	--->	keras.layers.Conv2D()

Fig. 8: API Function usage and their Explanation

Table 2: Model Changes Explored to Improve Performance

Item in Model Layers	Possible Value
Dropout	keras.layers.dropout(0.5)
Loss Function	L2

Our team has explored dropout and L2 loss function for modifying ResNet architecture to improve model performance as show in Table.2.

8.1. Dropout

Based on research, adding dropout layers from the back to front may help improve the model performance. Our team has tried to add multiple dropout layers starting from identity_block() function and followed by putting into cov_block() function. However, the dropout layer did not work well for our custom Mask RCNN model as the total loss does not improve much even with different dropout rates. Therefore, our team decided to keep the original ResNet architecture.

8.2. Loss Function

During the training of the model, we found out that RPN bbox loss was always the highest loss amongst the five loss categories. With further investigation into the rpn_bbox_loss_graph() function, our team realised that it uses the smooth L1 as the loss function to calculate RPN bbox loss. To prevent the model getting stuck in a local optimal instead of a global optimal, our team tried implementing the L2 regularization as loss function for RPN BBox. However, whilst testing it, the least square errors lead to higher loss instead. Therefore, we decided to keep the original smooth L1 as the loss function for RPN bbox.

Table 3: Parameters Explored to Improve Performance

Parameter in Config Setting	**Possible Value**
Image Min Dim & Image Max Dim	800 X 1024 (default) 512 X 512 256 X 256
Backbone Model	Resnet 101 (default) Resnet 50
RPN_ANCHOR _SCALES	(32, 64, 128, 256, 512) (default) (16, 32, 64, 128, 256) (8, 16, 32, 64, 128)
RPN_NMS _THRESHOLD	0.7 (default) 0.8 0.9
DETECTION_MIN _CONFIDENCE	0.7 (default) 0.75 0.8
Image Augmentation	Geometric transform: scale='x': (0.98, 1.02), 'y': (0.98, 1.02) translate_percent='x': (-0.05, 0.05), 'y': (-0.05, 0.05) rotate=(-10, 10) shear=(-5, 5) Flip image left-right: Fliplr(0.2) Brightness or Contrast: Multiply((0.8, 1.2)) ContrastNormalization((0.8, 1.2)) Blur or Sharpen: GaussianBlur(sigma=(0.0, 0.1)) Sharpen(alpha=(0.0, 0.1)) Fliplr(0.2) & GaussianBlur(sigma=(0.0, 0.1)) & Sharpen(alpha=(0.0, 0.1)) ...

Table 4: Parameters Explored to Improve Performance (cont.)

Parameter in Config Setting	**Possible Value**
Image Augmentation (cont)	... scale='x': (0.98, 1.02), 'y': (0.98, 1.02) translate_percent='x': (-0.05, 0.05), 'y': (-0.05, 0.05) rotate=(-10, 10) shear=(-5, 5) & Fliplr(0.2) & Multiply((0.8, 1.2)) ContrastNormalization((0.8, 1.2)) & GaussianBlur(sigma=(0.0, 0.1)) Sharpen(alpha=(0.0, 0.1)) scale='x': (0.98, 1.02), 'y': (0.98, 1.02) translate_percent='x': (-0.05, 0.05), 'y': (-0.05, 0.05) rotate=(-10, 10) shear=(-5, 5) & Fliplr(0.3) & Multiply((0.8, 1.2)) ContrastNormalization((0.8, 1.2)) & GaussianBlur(sigma=(0.0, 0.2)) Sharpen(alpha=(0.0, 0.2)) scale='x': (0.98, 1.02), 'y': (0.98, 1.02) translate_percent='x': (-0.05, 0.05), 'y': (-0.05, 0.05) rotate=(-10, 10) shear=(-5, 5) & Fliplr(0.35) & Multiply((0.8, 1.2)) ContrastNormalization((0.8, 1.2)) & GaussianBlur(sigma=(0.0, 0.25)) Sharpen(alpha=(0.0, 0.25))
...	...

Table 5: Parameters Explored to Improve Performance (cont2.)

Parameter in Config Setting	Possible Value
...	...
No. of epochs	60 120
Learning Rate (LR)	0.001 (default) Epoch 1-40: default LR Epoch 41-60: default LR/10 Epoch 61-80: default LR/50 Epoch 81-100: default LR/100 Epoch 101-120: default LR/500
	Epoch 1-40: default LR Epoch 41-60: default LR Epoch 61-80: default LR * 1e-1 Epoch 81-100: default LR * 1e-1 Epoch 101-120: default LR* 0.5e-1
	Epoch 1-40: default LR Epoch 41-60: default LR Epoch 61-80: default LR * 1e-1 Epoch 81-100: default LR * 1e-2 Epoch 101-120: default LR * 1e-3

Besides the model architecture, our team has also explored different values of parameters in the configuration as show in Table.3, Table.4 & Table.5.

8.3. Image (Min/Max) Dimensions

The Image Min Dim and Image Max Dim help to control the input image size. Smaller images can be helpful to reduce memory requirements and training time. Our team explored image sizes of 800*1024 (default), 512*512, and 256*256. Image Min Dim and Image Max Dim with the values of 512*512 had the lowest total loss result of 5.31.

8.4. Backbone

The Backbone value represents the ConvNet architecture that is used in Mask RCNN. The available options for Backbones include ResNet50, ResNet101, and ResNext101. For this project, our team has explored both ResNet50 and ResNet101. The final choice is based on the trade off between training time and accuracy. Compared to ResNet50, ResNet101 tends to be more accurate, albeit with slightly longer training time. Therefore, we decided to use ResNet101 as the backbone for the custom Mask RCNN model.

8.5. RPN Anchor Scales

The RPN Anchor scales represent the length of the square anchor sides in pixels. With the default configured RPN anchor sizes, the model was found to predict large masks very well. However, when it comes to small masks, it struggles to get the correct RPN area. Therefore, our team decided to scale down the image by applying a smaller RPN Anchor Scales. With RPN Anchor Scales value as (16, 32, 64, 126, 256), the lowest total loss result can reach 3.30.

8.6. RPN NMS Threshold

The RPN NMS Threshold is the threshold of non-max suppression, which is applied to remove boxes that have an overlap count more than the threshold. Increasing the value of RPN NMS Threshold will help keep more possible boxes during the training phase. This will help to reduce the RPN bbox loss and RPN class box. In this project, based on the tradeoff between overfitting and reducing total loss, we decided to use 0.8 as the final RPN NMS Threshold value for our custom Mask RCNN model. With this setting, the final total loss was reduced to 3.28.

8.7. Detection Min Confidence

The Detection Min Confidence is the threshold for classification of an instance. Increasing the threshold will ensure there are minimal false positives by guaranteeing the model predicts only the instances with very high confidence. In this project, our team has applied 0.7, 0.75 and 0.8 values as detection min confidence. However, as we increase the threshold, we notice a greatly decreasing value in total loss, hence we decided to use the default value 0.7 in the final model.

8.8. Image Augmentation

Table 6: Augmentation Tools

Category	Methodology
Geometric Transform	Scale, translate, rotate, shear, piecewiseAffine
Flip transform	fliplr
Brightness or Contrast	Multiply, contrast normalization
Blur or Sharpen	Gaussian blur, sharpen

Due to the limited amount of training data, the model does not return a "good" loss for RPN box and mask on the original unaugmented data. One way to improve the loss is in applying Image augmentation to generate more related training images through different methods (e.g. rotation, shearing,

flipping, sharpening, etc). Our team has grouped the augmentation tools into four categories. Please find the Table.6 as reference.

By testing a few combinations of different augmentation methods and twisting the parameter values, we fine tune the model with a lower loss value till 2.86.

8.9. No. of Epochs

On the first training of the base custom Mask RCNN model, our team used 60 epochs from both head layers (40 epochs) and all layers (20 epochs) in order to reduce the total training time. In general, increasing epochs during training will help in refining the models weights resulting in a lower loss. As expected, when we train 120 epochs instead of 60 epochs, we find that the total loss is reduced. However, we find that for mask and class label, training on more epochs occurs overfitting as well. To avoid the overfitting issue, our team has decided to apply a decreasing learning rate methodology as discussed below.

8.10. Learning Rate

As we first train the model, we apply the default learning rate in the configuration to all the epochs. However, with the weights being updated, reducing learning rate at later training epochs will help lower the loss and prevent overfitting. For this project, our team has decided to use a defined learning rate hierarchy which defines learning rate values for different ranges of epochs.

With all the fine tuning methodologies applied, the total loss is finally reduced to 2.24.

9. FINDINGS AND CONCLUSIONS

9.1. Performance

9.1.1. Fine Tune Performance Comparison

Fine tuning in deep learning models is a crucial step to improve the performance of forecasted results. There is a need to find the optimal hyper-parameters such as learning rate, dropout, image augmentation and multiple layers to lower the model loss. However, through the training process, we realized that there is no simple or easy way to set the hyper-parameters. The process of setting the hyper-parameters requires expertise and extensive trial and error. Further, not every hyper-parameter is suitable for every model, resulting in added difficulty. Upon an extensive period of trial and error to select the best hyper-parameters, our team has finally reduced the total loss of our custom Mask RCNN model to 2.24. Please find in Fig.9 and Fig.10 the progress of our model loss improvement.

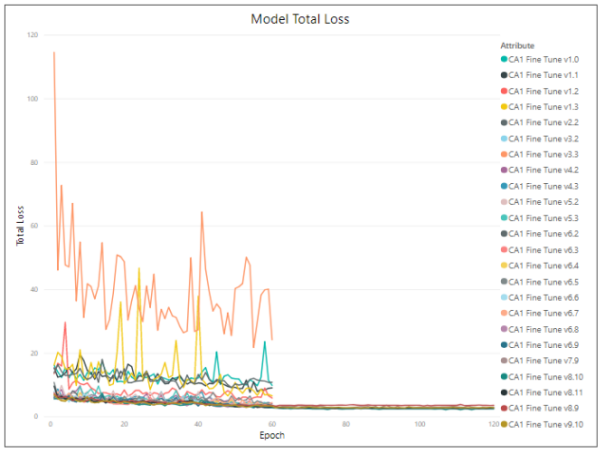

Fig. 9: Total Loss vs Epoch (All Models)

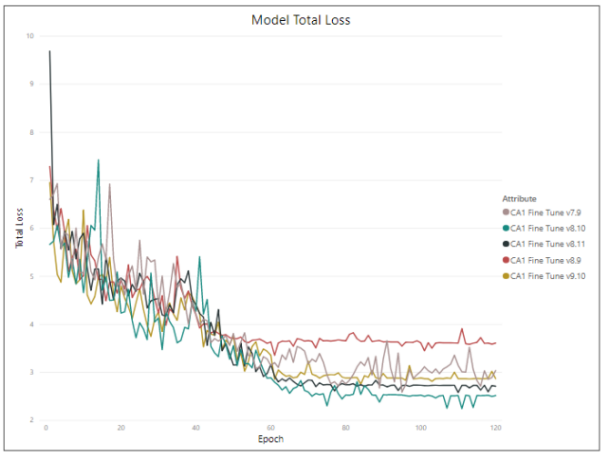

Fig. 10: Total Loss vs Epoch (Models with 120 epochs)

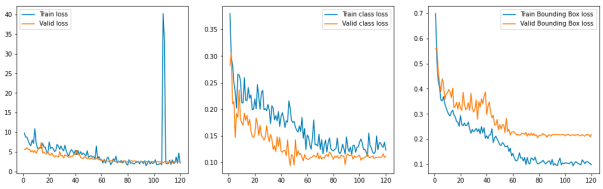

Fig. 11: Overall, Classification & Bounding Box (Train/Test) Loss vs Epochs

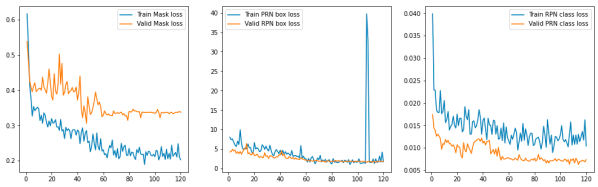

Fig. 12: Mask, RPN Bounding Box & RPN Classification (Train/Test) Loss vs Epochs

9.1.2. Loss

In training the custom Mask RCNN model, 6 categories are measured, total loss, class loss, bounding box loss, mask loss, RPN class loss and RPN bounding box loss. After fine tuning of the model hyper-parameters, our best model achieves a total loss of 2.24 over 120 epochs. Also, for each loss category, the loss value gradually drops as the model learns. Please find in Fig.11 and Fig.12 the loss values for the 6 categories.

9.1.3. mAP

Mean Average Precision (mAP) measures the overall average areas under the Precision-Recall Curve. Our custom Mask RCNN model can achieve a value of 0.82 for mAP when using all the image instances in the validation dataset.

9.2. Validation (Scoring Dataset)

Fig. 13: Scoring Image 1 (Suit)

Fig. 14: Scoring Image 2 (Suit)

Our team has prepared 5 scoring images covering all 3 categories (Suit:2, Dress:2, Jeans:1). Images are taken from different angles, lighting and background environments to test that the custom Mask RCNN model can handle various scenarios. Overall, the Model performs well on the scoring data. Please find the result of the individual images in Fig.13, Fig.14, Fig.15, Fig.16 and Fig.17.

Fig. 15: Scoring Image 3 (Dress)

Fig. 16: Scoring Image 4 (Dress)

Fig. 17: Scoring Image 5 (Jeans)

9.3. Challenges

9.3.1. Data Quantity

It would have been time consuming and tedious to prepare 200 instances for each category if we were to draw the annotation by using label tools. Meanwhile, there are insufficient annotated datasets for us to directly use. However, crawling of images on the internet, filtering out inappropriate images, and preparing COCO Format annotations still requires a big effort. Furthermore there is no single agreed upon data format but multiple forms for the annotation of masks (e.g. separate binary mask for OpenImagesv5, RLE & (x-y) polygon for COCO).

9.3.2. Data Quality

As the OpenImage gets its data from various sources, the images obtained with the OpenImagesv5 dataset had many different resolutions when we fitted them into the model. When using the original images and binary masks from OpenImage, our team was required to match the binary mask(s) onto the original image to obtain a COCO style annotation. In doing so there might be pixel-level changes that could cause annotation errors.

9.3.3. Computation Resource

Image processing and deep learning networks are computationally intensive activities. Image processing such as converting the images to desired format, is still manageable, as it only mainly involves the CPU. However, when training a deep learning model to get a mask at the pixel level, using the CPU is far slower than using the GPU. Hence, we attempted to use the tensorflow GPU backend to train the model. However, on our local GPU(s), the full running script returned in an "Out of Memory" error. The only platform that we can properly train the custom Mask RCNN model is Google Colab. By connecting and running with Colab online GPU services, the model could be run and completed within 3 hour

9.4. Conclusion

For this project, our team has built a custom Mask RCNN model by creating a training and validation dataset, building COCO style annotation, implementing transfer learning on Mask RCNN and fine tuning hyper parameters and evaluating the model with various metrics. The final custom Mast RCNN model has an acceptable mAP value of 0.82 for all the validation images. We have also as well as successfully and correctly managed to provide class labels, bounding boxes and masks for five scoring images.

10. APPENDIX

For further information about the Functions and APIs please refer to:
Functions,APIs (usage & description).xlsx

Training Config:
NAME = 'MaskRCNN_config'
GPU_COUNT = 1
IMAGES_PER_GPU = 1
NUM_CLASSES = 3 + 1
STEPS_PER_EPOCH = 131
IMAGE_MIN_DIM = 512
IMAGE_MAX_DIM = 512
BACKBONE = 'resnet101'
RPN_ANCHOR_SCALES = (16, 32, 64, 128, 256)
RPN_NMS_THRESHOLD = 0.8
DETECTION_MIN_CONFIDENCE = 0.7

Inference Config:
NAME = 'MaskRCNN_inference_config'
GPU_COUNT = 1
IMAGES_PER_GPU = 1d

11. REFERENCES

[1] Analytics Vidhya, "Computer Vision Tutorial: Implementing Mask R-CNN for Image Segmentation (with Python Code)," https://www.analyticsvidhya.com/blog/2019/07/computer-vision-implementing-mask-r-cnn-image-segmentation/?utm_source=blog&utm_medium=introduction-image-segmentation-techniques-python, 2020, [Online; accessed 18-April-2020].

[2] Niels Ole Salscheider, "Simultaneous object detection and semantic segmentation," 2019.

[3] Aydin Ayanzadeh, "A study review: Semantic segmentation with deep neural networks," 03 2019.

[4] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *ECCV*, 2018.

[5] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee, "Yolact: Real-time instance segmentation," in *ICCV*, 2019.

[6] Waleed Abdulla, "Mask r-cnn for object detection and instance segmentation on keras and tensorflow," https://github.com/matterport/Mask_RCNN, 2017.

[7] Google, "Overview of Open Images V5," https://storage.googleapis.com/openimages/web/factsfigures_v5.html, 2020, [Online; accessed 18-April-2020].

[8] COCO, "COCO Dataset Format," http://cocodataset.org/#format-data, 2020, [Online; accessed 18-April-2020].

[9] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick, "Mask r-cnn," 2017.